# Prova Finale – Progetto di Reti Logiche

## Politecnico di Milano

A.A. 2020/2021

Prof. Fabio Salice

Luca Gabaglio (Codice persona 10578930 – Matricola 889641)

# Indice

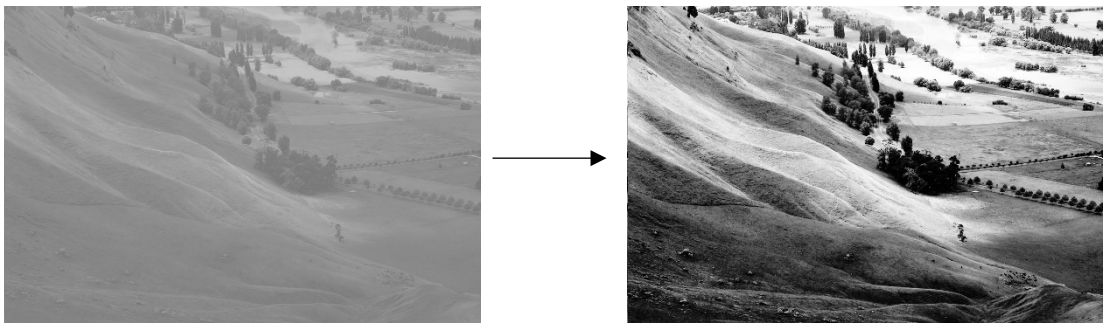
# 1. Specifiche di progetto

## 1.1. Descrizione generale

La specifica richiede l'implementazione di un componente hardware che realizzi una versione semplificata dell'algoritmo di equalizzazione dell'istogramma di una immagine.
L'algoritmo verrà applicato solamente a immagini in scala di grigio a 256 livelli (ogni pixel sarà dunque codificabile con 8 bit), e consiste nel trasformare ogni pixel secondo le seguenti operazioni, al fine di aumentare il contrasto:

*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)*

Verrà dunque prodotto per ogni immagine elaborata un risultato simile a quello dell'esempio seguente, rendendola molto più chiara.



Un esempio pratico molto semplice consiste nell'equalizzazione di un'immagine di 4 pixel, 2x2, con valori iniziali di

1. 46 (00101110)
2. 131 (10000011)
3. 62 (00111110)
4. 89 (01011001)

I pixel con valore massimo e minimo sono il primo ed il secondo, con valori rispettivamente di 46 e 131. Procediamo dunque a calcolare DELTA_VALUE come 131 – 46 = 85 (01010101).
Proseguiamo nell'algoritmo calcolando SHIFT_LEVEL = 8 – FLOOR ( LOG2 ( 85 +1 ) = 2.
Calcoliamo ora TEMP_PIXEL relativo ad ogni bit dell'immagine iniziale, sottraendo il valore del pixel minimo ad ognuno e shiftando a sinistra di due posizioni il risultato in codifica binaria:

1. 46 – 46 = 0 (00000000) => 0000000000 (0)
2. 131 – 46 = 85 (01010101) => 0101010100 (340)
3. 62 – 46 = 16 (00010000) => 0001000000 (64)
4. 89 – 46 = 43 (00101011) => 0010101100 (172)

Poniamo ora il valore di ogni pixel dell'immagine equalizzata uguale al valore minore tra il rispettivo TEMP_PIXEL ora calcolato e 255 e troviamo così il risultato finale:

1. 0 < 255 => 0
2. 340 > 255 => 255
3. 64 < 255 => 64
4. 172 < 255 => 172

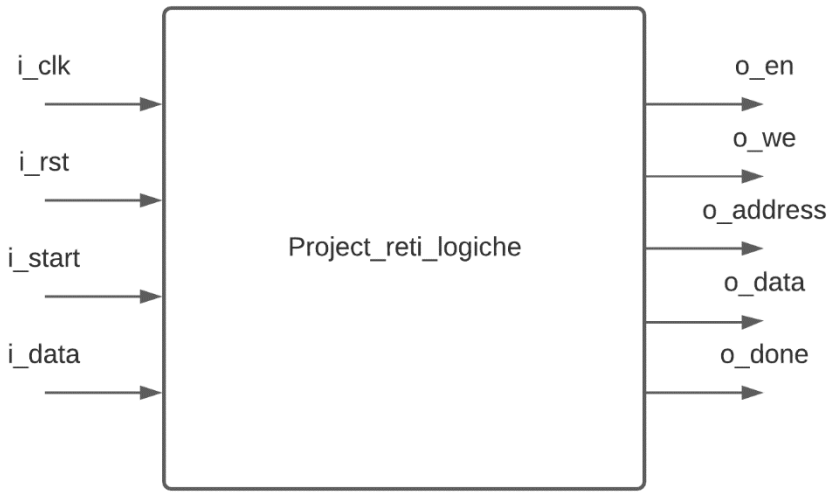
## 1.2. Componente realizzato

L'interfaccia del componente è definita nel modo seguente:

```
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;
```

dove:

- i_clk è il segnale di CLOCK in ingresso.
- 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 in ingresso.
- 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 mandare alla memoria per poter comunicare con essa.
- o_we è il segnale di WRITE ENABLE da mandare alla memoria per scriverci. Deve essere a '0' per leggere da essa.
- o_data è il segnale (vettore) di uscita dal componente verso la memoria.



## 1.3. Utilizzo della memoria

Viene utilizzata una memoria con indirizzamento al byte.
All'indirizzo *0* è memorizzato il numero di colonne dell'immagine da elaborare, mentre all'indirizzo *1* è memorizzato il numero di righe.
Il numero di bit totali che compongono l'immagine sarà quindi *n = numero di righe * numero di colonne*.
I bit da elaborare sono memorizzati a partire dall'indirizzo *2* fino all'indirizzo *n+1*.
Il risultato dell'elaborazione dei bit deve essere salvato in memoria a partire dall'indirizzo *n+2* fino all'indirizzo *2n+1*.

| | |
|---|---|
| Indirizzo 0 | N° di colonne dell'immagine |
| Indirizzo 1 | N° di righe dell'immagine |
| Indirizzo 2 | Primo pixel da elaborare |
| ..... | ..... |
| Indirizzo n + 1 | Ultimo pixel da elaborare |
| Indirizzo n + 2 | Risultato elaborazione primo pixel |
| ..... | ..... |
| Indirizzo 2n + 1 | Risultato elaborazione ultimo pixel |

# 2. Architettura

Per la soluzione del problema è stato utilizzato un singolo process, che implementa una macchina a stati finiti composta da 13 stati.

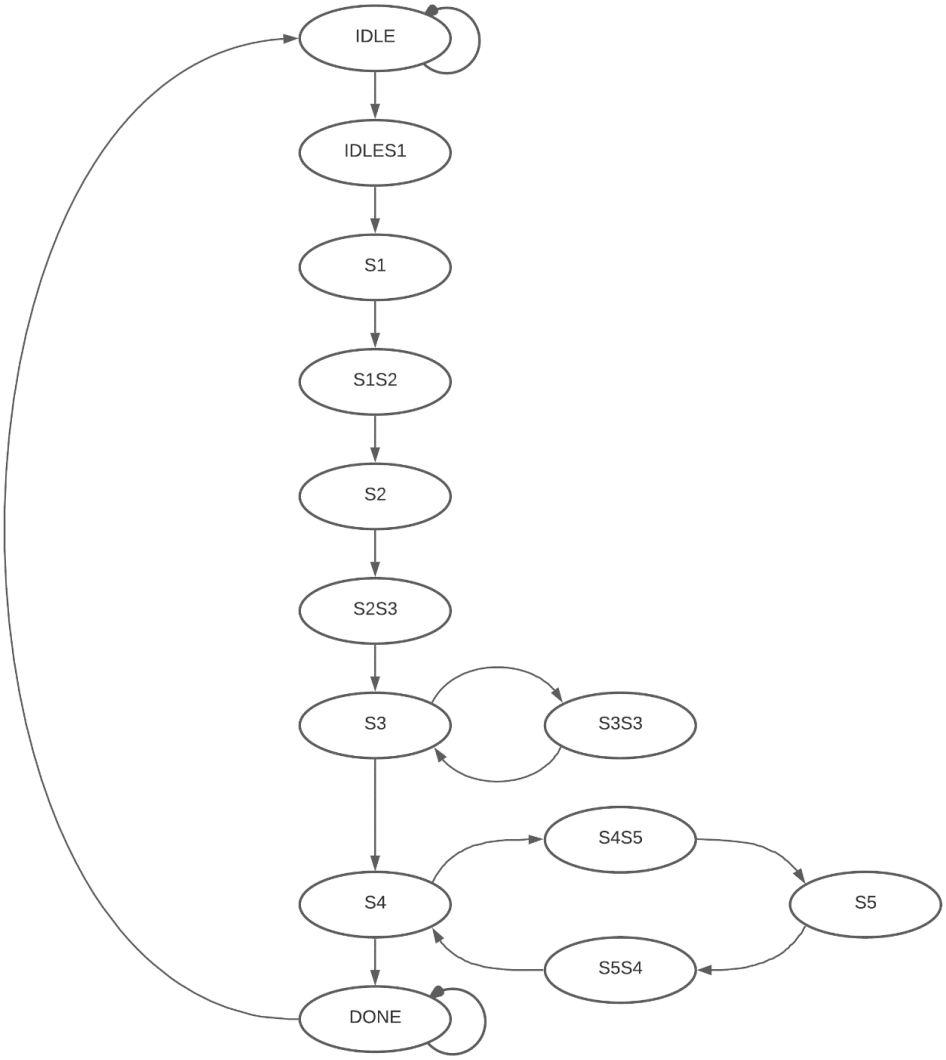
## 2.1. Variabili e segnali

Nell'esecuzione dell'algoritmo vengono usati i seguenti segnali e variabili:

- *state* - tipo_stato
  Segnale utilizzato per tenere traccia dello stato.
- *max, min* - std_logic_vector(7 downto 0)
  Segnali utilizzati per memorizzare il valore massimo tra i pixel dell'immagine non equalizzata e quello minimo.
- *counter* - std_logic_vector(15 downto 0)
  Segnale utilizzato per memorizzare il numero totale di pixel che compongono l'immagine. Usato per tenere traccia del punto di elaborazione in cui si è arrivati nei diversi cicli.
- *deltaValue* - std_logic_vector(7 downto 0)
  Variabile utilizzata per calcolare il risultato dell'operazione:
  *DELTA_VALUE=MAX_PIXEL_VALUE - MIN_PIXEL_VALUE.*
- *shiftLevel* - std_logic_vector(3 downto 0)
  Segnale utilizzato per calcolare il risultato dell'operazione:
  *SHIFT_LEVEL = (8 – FLOOR(LOG2(DELTA_VALUE + 1))).*
- *newPixelValue* - std_logic_vector(7 downto 0)
  Variabile utilizzata per calcolare il risultato dell'operazione:
  *NEW_PIXEL_VALUE = MIN (255, TEMP_PIXEL)*
- *tempPixel* - std_logic_vector(15 downto 0)
  Segnale utilizzato per calcolare il risultato dell'operazione
  *TEMP_PIXEL = (CURRENT_PIXEL_VALUE – MIN_PIXEL_VALUE) << SHIFT_LEVEL*
  che può produrre nel caso peggiore un vettore da 16 bit.
- *nRighe, nColonne* – integer
  Variabili utilizzate per memorizzare il numero di righe ed il numero di colonne dell'immagine.
- *counterSave* – integer
  Variabile utilizzata per memorizzare il valore intero del numero di pixel totali. Usata per ripristinare il segnale counter al valore corretto al termine dei diversi cicli.
- *log2* – integer
  Variabile utilizzata per calcolare il risultato dell'operazione:
  *FLOOR(LOG2(DELTA_VALUE + 1)).*

## 2.2. Macchina a stati finiti

La macchina a stati finiti realizzata ha il seguente schema:

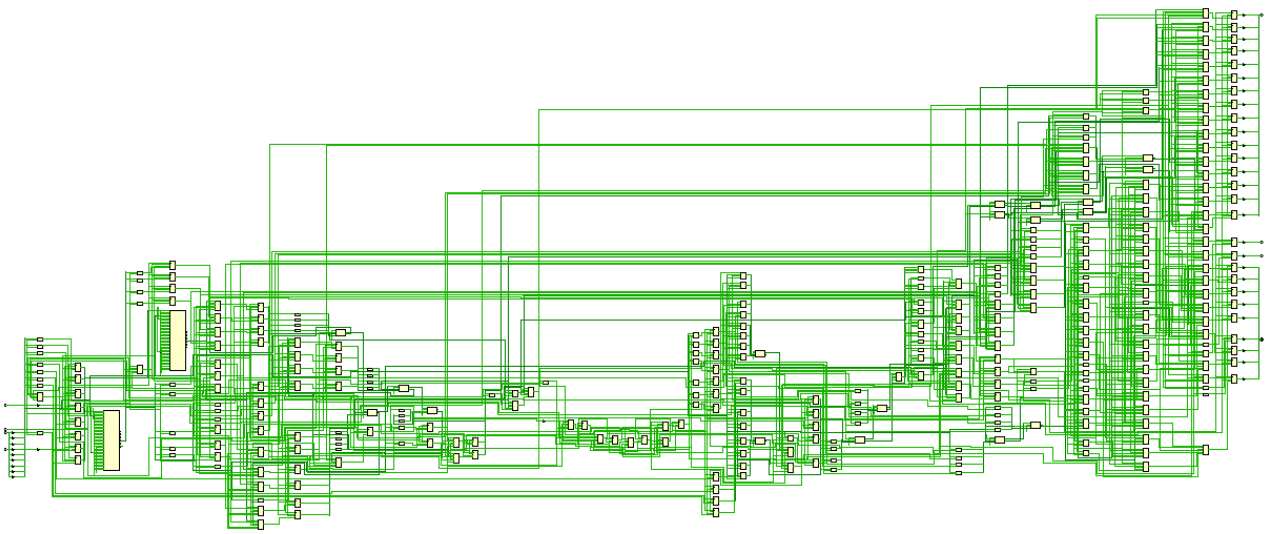



Segue una descrizione della funzione dei singoli stati, e con essa del funzionamento generale della macchina:

- **IDLE**
  Stato in cui la macchina attende che il segnale di start in ingresso venga portato a '1' per iniziare l'elaborazione dei dati.
  Se tale segnale vale '1' viene portato o_en a '1' e viene richiesta la lettura del valore all'indirizzo 0 della memoria.
- **IDLES1**
  Stato di transizione da IDLE a S1 in cui si attende la lettura del numero di colonne dalla memoria.
- **S1**
  Stato in cui viene letto il numero di colonne. Viene richiesta la lettura del valore all'indirizzo 1 della memoria.
- **S1S2**
  Stato di transizione da S1 a 21 in cui si attende la lettura del numero di righe dalla memoria.

- **S2**
  Stato in cui viene letto il numero di righe, calcolato il valore di counter e salvato anche in counterSave. Viene richiesta la lettura del valore all'indirizzo counter+1 della memoria.
- **S2S3**
  Stato di transizione da S2 a S3 in cui si attende la lettura del valore dell'ultimo pixel dalla memoria.
- **S3**
  Stato in cui si leggono tutti i valori dei pixel da elaborare al fine di trovare quello con il valore massimo e quello con il valore minimo. Questa elaborazione viene fatta leggendo il bit all'indirizzo counter+1, comparandolo ai precedenti valori di max e min e decrementando counter ad ogni iterazione.
  Al termine della lettura, prima di passare allo stato S4 vengono inoltre calcolati deltaValue e shiftLevel e viene ripristinato il valore di counter al numero di pixel.
- **S3S3**
  Stato di transizione, usato per attendere la lettura del pixel seguente durante lo scorrimento della lista operato in S3.
- **S4**
  Stato in cui viene richiesto alla memoria il valore del prossimo pixel da elaborare, ovvero quello all'indirizzo counter+1.
  Se sono stati elaborati tutti i pixel, ovvero se counter vale 0, vengono azzerate le variabili, portato o_done a '1', o_en a '0' e si passa allo stato DONE.
- **S4S5**
  Stato di transizione da S4 a S5 in cui si attende la lettura del valore del prossimo pixel da elaborare dalla memoria.
- **S5**
  Stato in cui viene posto o_we a '1', o_address al corretto indirizzo di destinazione per il pixel elaborato e computato il valore nell'immagine equalizzata del pixel letto. Viene inoltre decrementato counter di 1.
- **S5S4**
  Stato di transizione da S5 a S4 in cui viene posto o_we a '0'.
- **DONE**
  Stato finale in cui la macchina rimane finché non viene portato a '0' il segnale di start, nel caso in cui viene portato o_done a '0' e si torna allo stato IDLE.

## 2.3. Schematic



# 3. Report di sintesi

## 3.1. Area occupata

Come ricavabile dal report di utilizzo, il componente sintetizzato occupa lo 0.14% di LUT e lo 0.03% di FLIP FLOP, ovvero una piccola parte di quelli disponibili sulla FPGA.

# 4. Testing

## 4.1. Test generico

Il primo test condotto è stato svolto su un'immagine composta da 5880 pixel, generati casualmente, per verificare il corretto funzionamento del componente con un generico input, di dimensioni abbastanza importanti. Il componente ha superato il test salvando in RAM i valori corretti sia in Behavioral simulation che Post Synthesis Functional simulation.

## 4.2. Test con valori di ingresso nulli

Il secondo test condotto è stato svolto inserendo 0 come numero di righe e di colonne, per verificare che il componente non esegua effettivamente calcoli e non scriva nulla in RAM.
Il componente ha superato il test, non svolgendo nessuna operazione e passando subito attraverso tutti gli stati fino allo stato di DONE.

## 4.3. Test con reset asincrono

Il terzo test condotto è stato svolto per verificare il corretto funzionamento del componente in caso riceva un segnale di reset mentre sta elaborando un'immagine e debba quindi ripartire con l'esecuzione.
Il componente ha superato il test resettando correttamente i valori delle variabili e dei segnali e calcolando risultati corretti sia in Behavioral simulation che Post Synthesis Functional simulation.

## 4.4. Test con doppio start e cambio valori in ingresso

Il quarto test condotto è stato svolto per verificare il corretto funzionamento del componente nel caso in cui debba elaborare una nuova immagine dopo aver concluso l'elaborazione della prima, senza ricevere segnale di reset tra le due esecuzioni.
Il componente ha superato il test salvando in RAM i valori corretti sia in Behavioral simulation che Post Synthesis Functional simulation.

# 5. Conclusioni

Sono state applicate alcune ottimizzazioni atte a ridurre il numero degli stati rispetto al design concepito inizialmente, come ad esempio la rimozione di stati il cui unico scopo era richiedere la lettura di dati dalla memoria, incorporando la loro funzione negli stati che li precedevano.
Il componente realizzato supera correttamente tutti i test congegnati, sia in Behavioral simulation che in Post Synthesis.