# Progetto Finale di Reti Logiche

a.a 2020/2021

Samuele Messineo
codice persona: 10616398
email: samuele.messineo@mail.polimi.it

# Sommario

1 Introduzione ..... 3
2 Descrizione algoritmo ..... 3
2.1 Esempio ..... 4
3 Architettura ..... 5
3.1 Macchina a stati finiti FSM ..... 5
3.1.1 Lettura e calcolo parametri ..... 6
3.1.2 Calcolo pixel equalizzati e scrittura ..... 6
4 Descrizione segnali interni ..... 7
5 Risultati sperimentali ..... 8
6 Simulazioni ..... 9
6.1 Immagine 2x2 ..... 9
6.2 Reset asincrono ..... 9
6.3 Immagine vuota ..... 9
7 Conclusione ..... 10

# 1 Introduzione

Obbiettivo della Prova Finale di Reti Logiche è quello di realizzare, in linguaggio VHDL, un componente hardware, per l'equalizzazione dell'istogramma di una immagine.
Questo metodo è usato per ricalibrare i livelli di contrasto di un'immagine in input, specialmente quando questa ha una gamma ristretta di valori di intensità.


***Figura 1**: Confronto prima e dopo l'equalizzazione*

Il componente sviluppato si interfaccia con una memoria RAM, con indirizzi di 16 bit e celle da 8 bit, contenente nei primi due indirizzi le dimensioni dell'immagine da equalizzare, e poi sequenzialmente i valori di tutti i suoi pixel. Al termine dell'elaborazione i valori dei nuovi pixel equalizzati saranno scritti sulla RAM.

# 2 Descrizione algoritmo

Nella componente sviluppata, è stata implementata una versione semplificata dell'algoritmo standard, applicato solo su immagini in scala grigi a 256 livelli.

Data un'immagine in input, è richiesta una prima fase di lettura e calcolo dei parametri secondo cui avverrà l'equalizzazione:

DELTA_VALUE = MAX_PIXEL_VALUE – MIN_PIXEL_VALUE

SHIFT_LEVEL = (8 – FLOOR (LOG2(DELTA_VALUE +1)))

E (Sia CURRENT_PIXEL_VALUE il valore di un generico pixel dell'immagine in input)
una seconda fase per il calcolo dei nuovi valori equalizzati dei pixel:

TEMP_PIXEL = (CURRENT_PIXEL_VALUE - MIN_PIXEL_VALUE) << SHIFT_LEVEL

NEW_PIXEL_VALUE = MIN (255, TEMP_PIXEL)

## 2.1 Esempio

L'esempio presentato mostra il contenuto della memoria al termine di un'elaborazione di un'immagine 3x3.

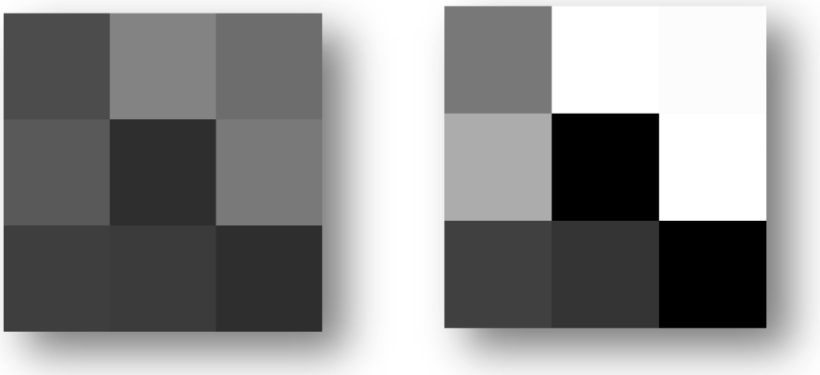


***Figura 2**: Immagine originale e l'immagine equalizzata*

Contenuto RAM:

| INDIRIZZO MEMORIA | VALORE | COMMENTO |
|---|---|---|
| 0 | 3 | \\ numero colonne |
| 1 | 3 | \\ numero righe |
| 2 | 76 | \\ primo pixel immagine |
| 3 | 131 | |
| 4 | 109 | |
| 5 | 89 | |
| 6 | 46 | |
| 7 | 121 | |
| 8 | 62 | |
| 9 | 59 | |
| 10 | 46 | \\ ultimo pixel immagine |
| 11 | 120 | \\ primo pixel immagine equalizzata |
| 12 | 255 | \\(risultato) |
| 13 | 252 | |
| 14 | 172 | |
| 15 | 0 | |
| 16 | 255 | |
| 17 | 64 | |
| 18 | 52 | |
| 19 | 0 | |

# 3 Architettura

L'algoritmo richiede due fasi distinte:

-Lettura e calcolo parametri conversione: ricerca del valore massimo e minimo dei pixel che compongono l'immagine memorizzati in memoria e il calcolo dello shift level.

-Calcolo pixel equalizzati e scrittura: per ogni pixel letto in memoria avviene il calcolo del rispettivo pixel equalizzato e poi viene scritto in memoria.

Il comportamento dell'algoritmo può essere modellato dalla seguente FSM

## 3.1 Macchina a stati finiti FSM

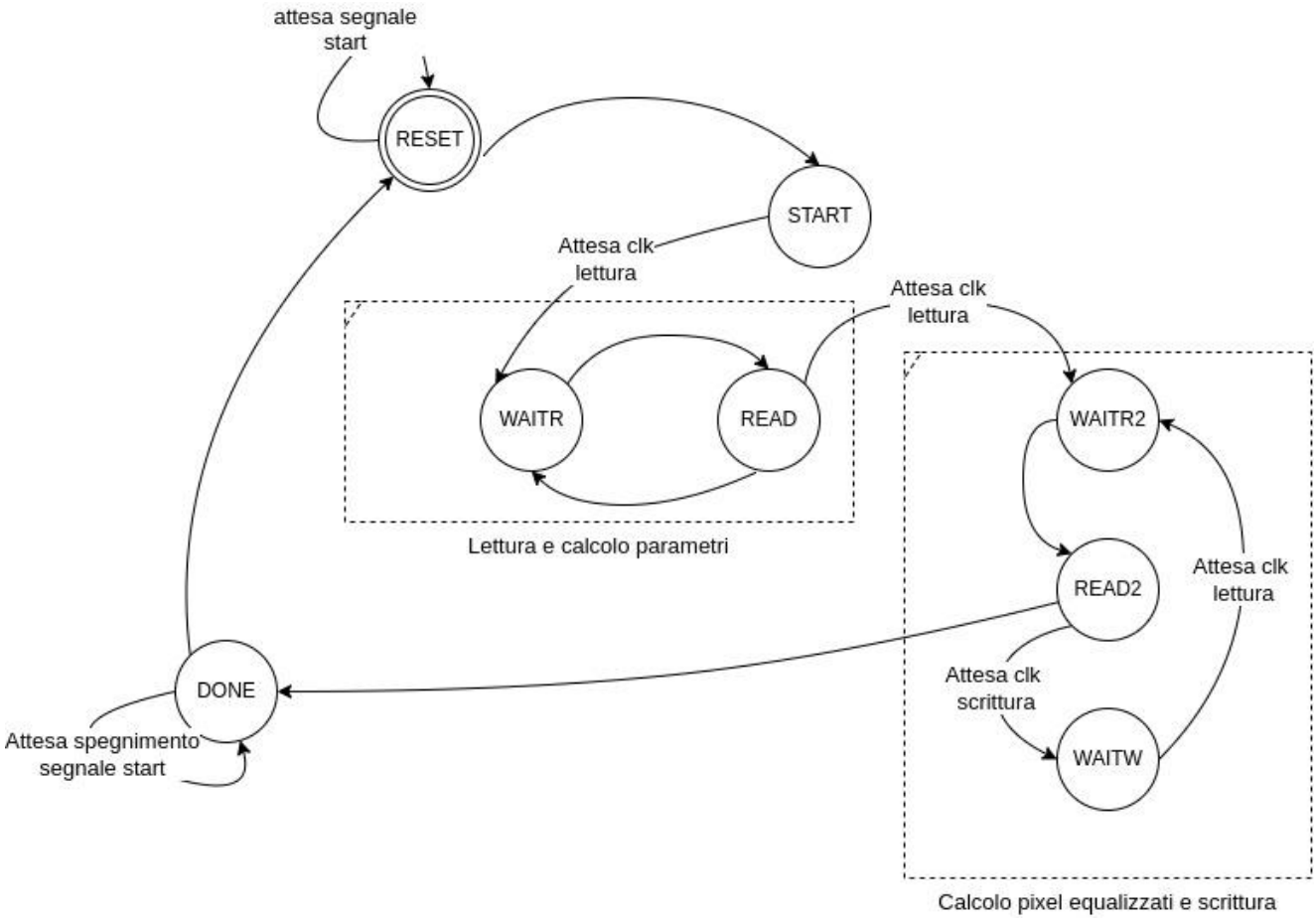



***Figura 3**: Rappresentazione della macchina a stati finiti*

Dallo stato di RESET, tutti i segnali vengono reimpostati al loro valore di default, in attesa che il segnale di start sia portato a 1. A questo punto si passa allo stato START, qui settiamo la configurazione di lettura, ponendo il segnale di *enable* (o_en) ad 1, mentre quello di *write* (o_we) rimane a 0 dallo stato di RESET. Ci prepariamo a leggere il primo indirizzo di memoria e passiamo allo stato di WAITR.

### 3.1.1 Lettura e calcolo parametri

Avviene alternando il passaggio tra due stati WAITR e READ: il primo permette di saltare un ciclo di clock attendendo che la RAM prepari il valore richiesto in output. Il Secondo legge in input dalla RAM.
Nelle prime due iterazioni READ andrà a memorizzare le dimensioni delle immagini, nelle successive avviene la ricerca dei valori di massimo e minimo.
Determinati massimi e minimi, passiamo alla parte finale di questa fase, il calcolo dello *shift level* tramite una serie di controlli soglia.

### 3.1.2 Calcolo pixel equalizzati e scrittura

Ritornando all'indirizzo di memoria contenente il primo pixel dell'immagine per poter ripetere una lettura di tutti i pixel, come nella fase precedente e per ognuno di questi creiamo il rispettivo pixel equalizzato, secondo l'algoritmo e specifichiamo l'indirizzo in cui sarà scritto in output:

indirizzo_corrente+(num_righe*num_colonne)

Anche qui avremo bisogno di attendere un ciclo di clock perché la RAM scriva in memoria, ci poniamo in attesa in WAITR per poi passare al prossimo pixel tornando in WAITR.

Equalizzati tutti i pixel non ci resta che passare dallo stato di READ2 a quello finale di DONE portando il segnale di *o_done* a 1. In DONE abbassiamo il segnale di *i_start* e ci poniamo in RESET, qui terminiamo riportando tutti i segnali al loro valore di default e ci poniamo in attesa di una nuova immagine (i-start=1).

# 4 Descrizione segnali interni

| Segnale/Variabile | Dettagli | Descrizione |
|---|---|---|
| ncolonne | Vettore 8 bit<br>std_logic_vector(7 downto 0); | Numero colonne dell'immagine |
| nrighe | Vettore 8 bit<br>std_logic_vector(7 downto 0); | Numero righe dell'immagine |
| max_pixel | Vettore 8 bit<br>std_logic_vector(7 downto 0); | Valore più alto raggiunto dai pixel nell'immagine |
| min_pixel | Vettore 8 bit<br>std_logic_vector(7 downto 0); | Valore più basso raggiunto dall'immagine |
| current_state | State_type | Stato in cui si trova la FSM |
| temp_address | Vettore 16 bit<br>std_logic_vector(15 downto 0); | Indirizzo cella lettura |
| shift_level | Intero da 0 a 8<br>integer range 0 to 8; | Numero di shift a sinistra da eseguire per ottenere i nuovi pixel |
| temp_pixel | Vettore 8 bit<br>std_logic_vector(7 downto 0); | Variabile per passaggi intermedi e che conterrà i valori dei pixel equalizzati |

# 5 Risultati sperimentali

Dall'utilization report

| Site Type | Used |
|---|---|
| Slice LUTs* | 207 |
| LUT as Logic | 207 |
| LUT as Memory | 0 |
| Slice Registers | 82 |
| Register as Flip Flop | 82 |
| Register as Latch | 0 |
| F7 Muxes | 0 |
| F8 Muxes | 0 |

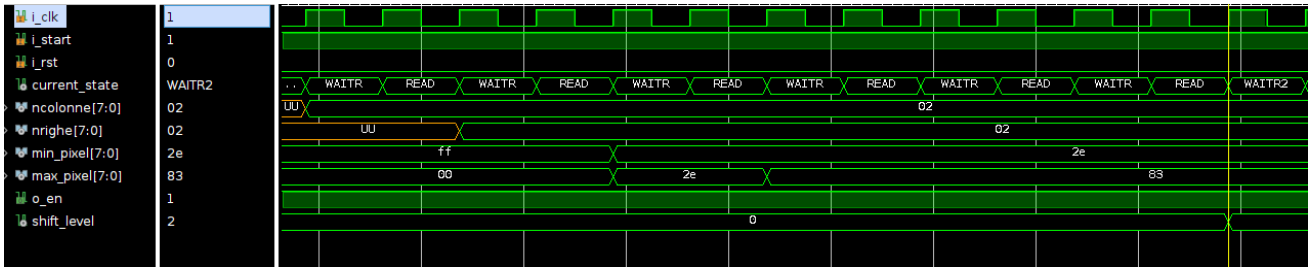
Dalla synsetis report, la costituzione del componente realizzato:

| Component | Inputs number | Output size | Count |
|---|---|---|---|
| Adders | 3 Input | 17 Bit | Adress := 1 |
| | 3 Input | 16 Bit | Adress := 1 |
| | 3 Input | 16 Bit | Adress := 1 |
| | 3 Input | 16 Bit | Adress := 1 |
| Registers | | 16 Bit | Registers := 2 |
| | | 8 Bit | Registers := 7 |
| | | 4 Bit | Registers := 1 |
| | | 3 Bit | Registers := 1 |
| | | 1 Bit | Registers := 3 |
| Muxes | 2 Input | 16 Bit | Muxes := 3 |
| | 8 Input | 16 Bit | Muxes := 2 |
| | 8 Input | 8 Bit | Muxes := 3 |
| | 2 Input | 8 Bit | Muxes := 1 |
| | 8 Input | 4 Bit | Muxes := 1 |
| | 8 Input | 3 Bit | Muxes := 1 |
| | 5 Input | 3 Bit | Muxes := 1 |
| | 2 Input | 3 Bit | Muxes := 1 |
| | 8 Input | 1 Bit | Muxes := 14 |
| | 2 Input | 1 Bit | Muxes := 2 |

# 6 Simulazioni

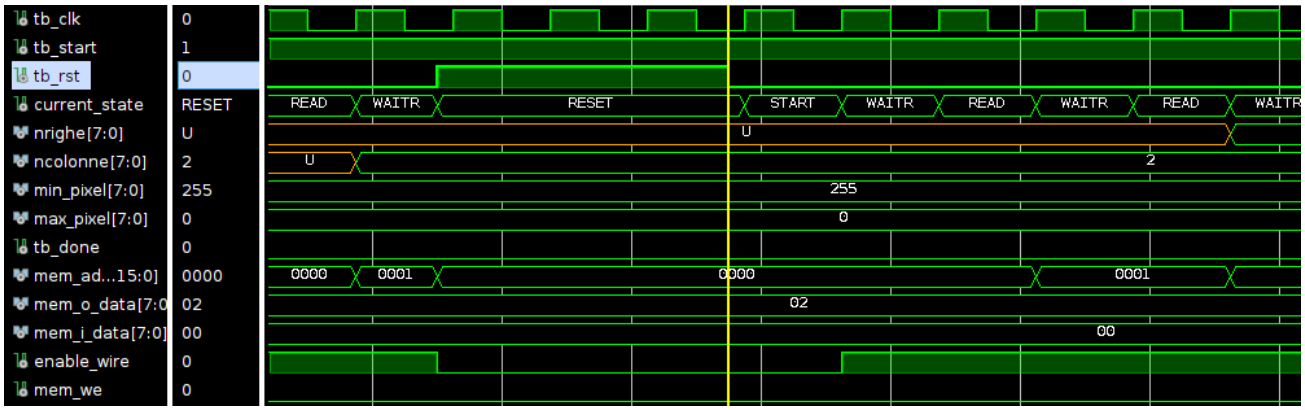
A dimostrazione del funzionamento del componente sviluppato sono stati presentati tre esempi: uno standard e due casi limite.

## 6.1 Immagine 2x2



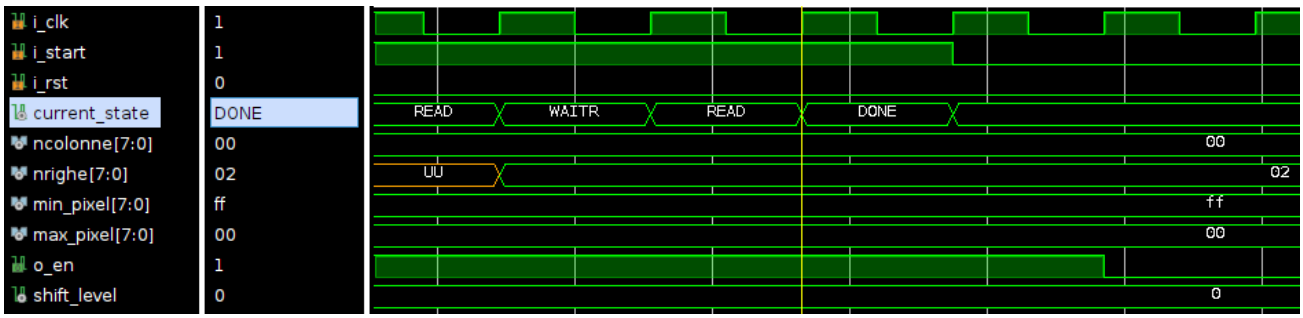

## 6.2 Reset asincrono

Nel caso in cui il segnale di *i_rst* venga alzato ad 1 in modo asincrono al segnale di clock, indipendente dallo stato in cui si trova la FSM, questa torna allo stato di RESET.




## 6.3 Immagine vuota

Nel caso in cui venga inserita in input un'immagine vuota, dopo che nella prima fase è stato verificato che una delle sue due dimensioni è pari a zero, si passa dallo stato di READ a quello di DONE settando il segnale o_done ad 1.



# 7 Conclusione

Viviamo in un mondo dove i metodi di elaborazione digitale delle immagini hanno assunto un ruolo sempre più importante.
Per questo, poter comprendere meglio e farsi un'idea della logica applicativa alla base di uno di questi metodi è stato affascinante, ma arrivare a pensare e progettare una componente in modo personale che ne implementasse direttamente uno, seppure in maniera semplificata, lo è stato ancora di più.
Ho avuto la possibilità di realizzare un progetto e poterne simulare l'utilizzo, vedendo così i risultati del mio lavoro e studio; per questo motivo, questa è stata un'occasione per mettere alla prova e integrare le conoscenze teoriche apprese durante il corso di reti logiche.