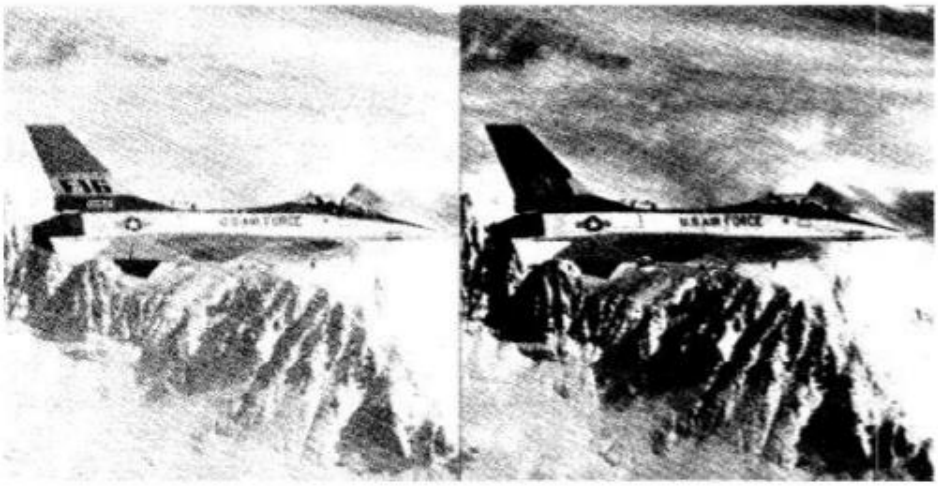
# Progetto Reti Logiche 2020/2021

Prof. Fabio Salice

Roberto Molgora (Matricola: 908160 – Codice Persona 10628824)

## 1. Introduzione

Lo scopo di questo progetto è realizzare in VHDL una versione semplificata dell'algoritmo utilizzato per realizzare il metodo di equalizzazione dell'istogramma di un'immagine. Questo metodo è pensato per incrementare il contrasto di un'immagine quando l'intervallo delle intensità dei pixel è molto piccolo, effettuandone una distribuzione su tutto l'intervallo di intensità.



### 1.1 L'algoritmo

La versione da sviluppare tratterà immagini di una dimensione massima di 128x128 pixel su 8 bit, ovvero le immagini saranno su una scala di grigi a 256 livelli. L'algoritmo utilizzato per modificare ogni singolo pixel è il seguente:

```
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 , 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 nuovo pixel.

Per esempio, supponendo di avere un'immagine di 4x3 pixel (qui rappresentata da una matrice in cui in ogni casella viene riportato il valore del pixel corrispondente), l'immagine equalizzata sarebbe la seguente:

$$\begin{bmatrix} 76 & 131 & 109 \\ 89 & 46 & 121 \\ 62 & 59 & 46 \\ 77 & 68 & 94 \end{bmatrix} \longrightarrow \begin{bmatrix} 120 & 255 & 252 \\ 172 & 0 & 255 \\ 64 & 52 & 0 \\ 124 & 88 & 192 \end{bmatrix}$$

Come si può notare l'algoritmo sopra descritto ha modificato il valore dei pixel "spalmandoli" sull'intero range a disposizione (in rosso il MAX_PIXEL_VALUE e in azzurro il MIN_PIXEL_VALUE prima e dopo l'equalizzazione).

### 1.2 La memoria

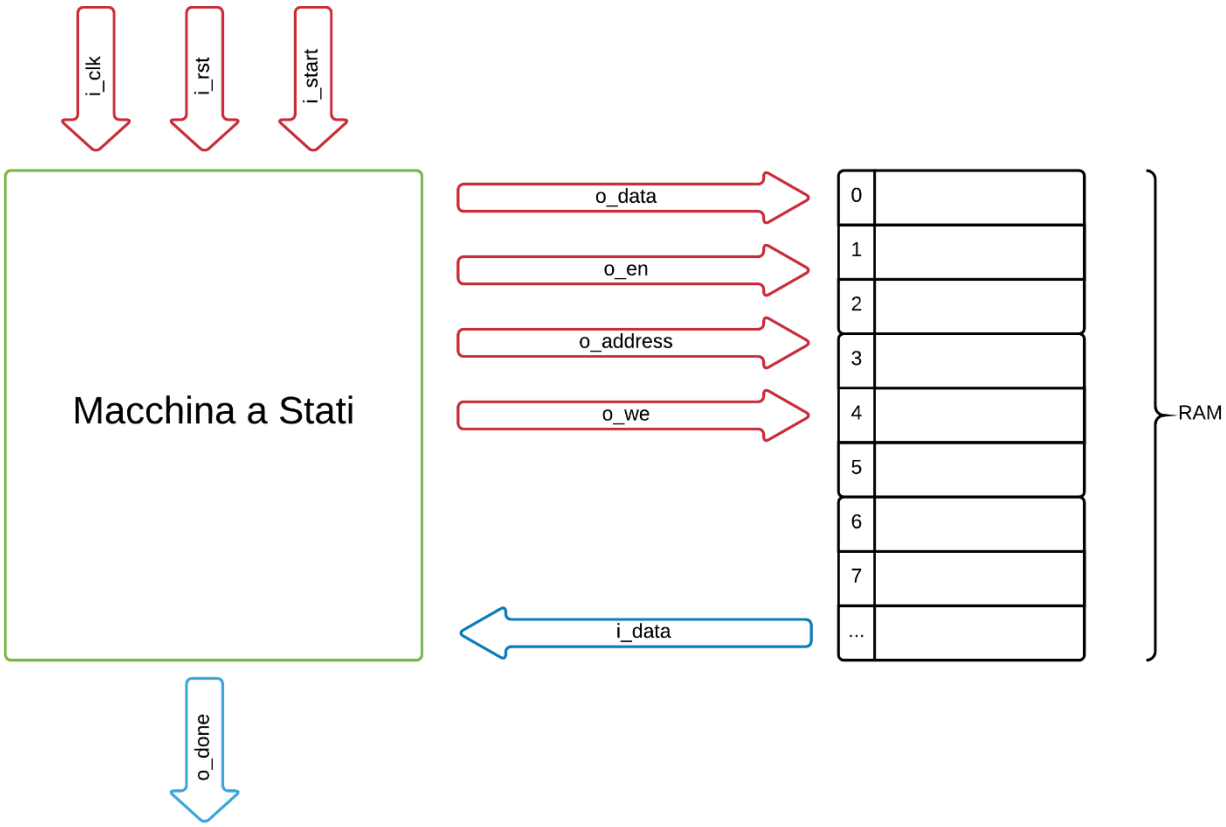
I dati in ingresso vengono forniti su una memoria indirizzata la byte nella quale si troverà all'indirizzo 0 il numero di colonne, all'indirizzo 1 il numero di righe e nei successivi in sequenza i valori dei pixel di ingresso. Se ad esempio si dovesse avere all'indirizzo 0 il valore 4 e all'indirizzo 1 il valore 2 si avrebbero le celle da 0 a 9 occupate (da 2 a 9 ci sarebbero i valori degli 8 pixel). L'immagine equalizzata viene scritta nei byte immediatamente successivi all'ultimo byte della prima immagine.

### 1.3 Sincronia

La specifica impone di prestare particolarmente attenzione alla sincronizzazione dei segnali, fornisce infatti dei passi predefiniti da compiere una volta finita la lettura di un'immagine per poterne leggere un'altra.

La sincronizzazione riguarda in particolare i segnali i_start e o_done le quali combinazioni possono indicare in quale stato della computazione si trova la macchina.

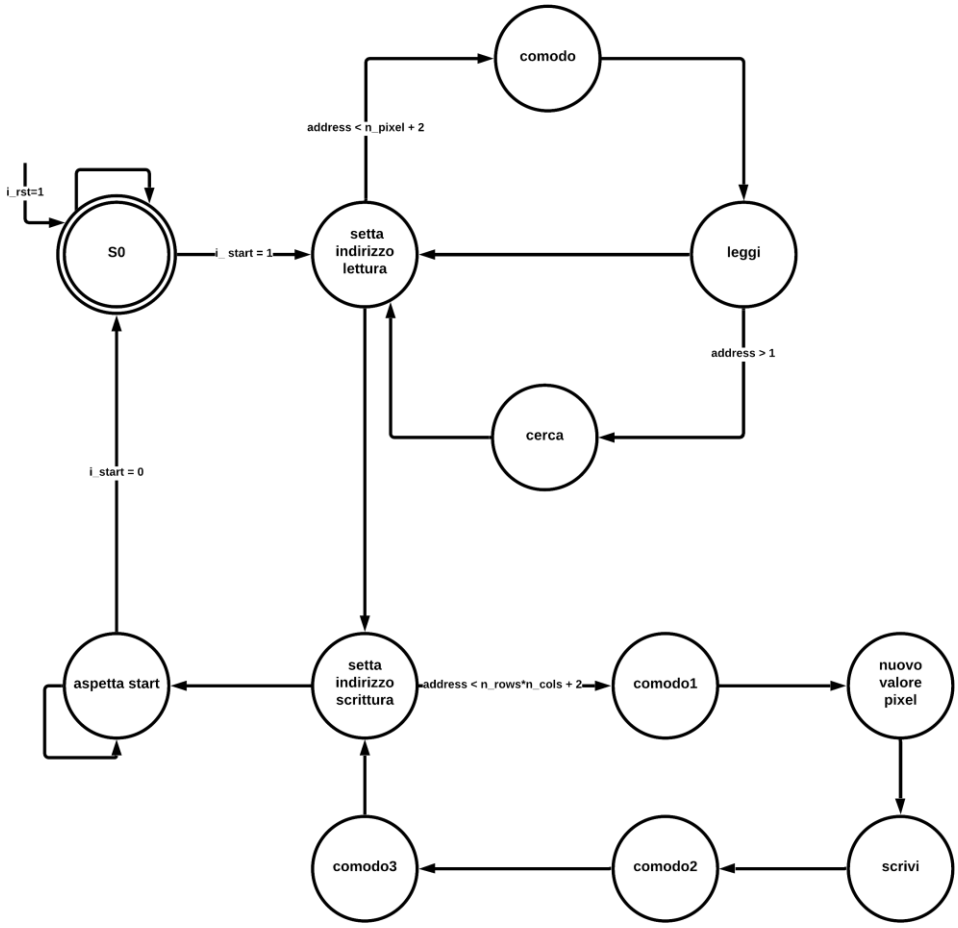
## 2. Architettura



## 2.1 La Macchina

Il modulo della macchina a stati è composto da un process che implementa una macchina a stati composta da 12 stati:

- S0: attivato quando il segnale di reset viene portato ad 1, il programma ci resta fino a quando i_start non viene portato a 1 e quindi può essere processata un'immagine.
- setta_indirizzo_lettura: indica a o_address l'indirizzo dal quale leggere il prossimo dato. In caso l'immagine sia finita, viene calcolato lo shift_level.
- comodo: come indica il nome è uno stato "di comodo" che serve solo a far trascorrere un ciclo di clock; necessario in quanto i segnali non vengono aggiornati immediatamente ma vengono aggiornati dopo un ciclo di clock, in questo caso necessito di due cicli di clock in quanto uno serve per aggiornare o_address e uno per aggiornare i_data.
- leggi: legge il dato che trova in i_data e incrementa l'indirizzo di lettura.
- cerca: trova il MAX_PIXEL_VALUE e il MIN_PIXEL_VALUE.
- setta_indirizzo_scrittura: setta l'indirizzo da cui leggere i pixel che verranno modificati.
- comodo1: stesso ruolo di comodo.
- nuovo valore pixel: legge il valore del pixel attuale e lo modifica implementando l'algoritmo fornito dalla specifica e comunica a o_address l'indirizzo in cui dovrà scrivere questo dato.
- scrivi: scrive il nuovo valore del pixel nella cella corrispondente.
- comodo2: stesso ruolo di comodo.
- comodo3: imposta o_we a 0 in modo da poter leggere.
- aspetta_start: è una specie di secondo processo IDLE, ovvero attende finché i_start non viene portato a 0, quindi riporta o_done a 0 e riporta ad S0.



## 3. Risultati sperimentali

Il progetto descritto utilizza 91 FF, 320 LUTs, 0 Latch. è stato usato l'FPGA xc7a200tfbg484-1, il quale presenta una quantità di FF più che sufficiente a soddisfare le richieste hardware (molto contenute) del progetto.

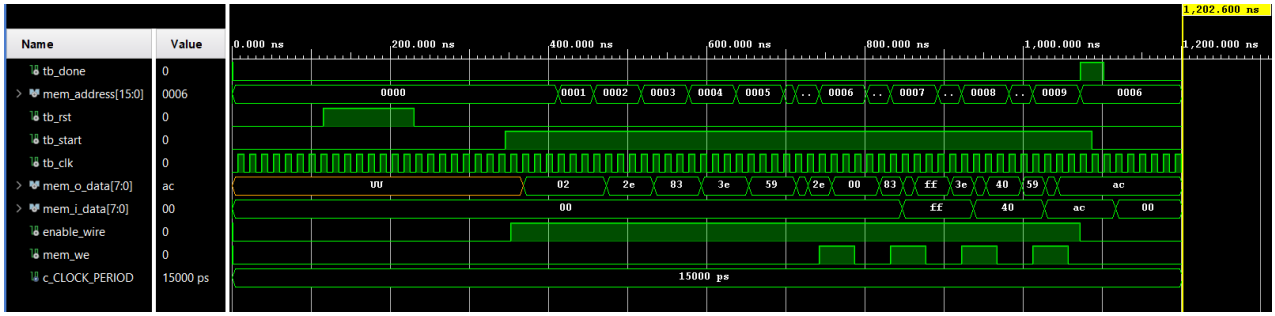
## 4. Simulazioni

Il progetto è stato testato con diversi test bench, TB_2x2, ResetAsincrono, ZeroPixel, DimensioneMassima e Test3Immagini. Come si potrà notare il progetto viene testato con un clock di 15 ns, molto al di sotto del 100 ns massimi richiesti da specifica.

Sono stati selezionati questi casi di test in modo da ricoprire i principali casi limite della specifica.

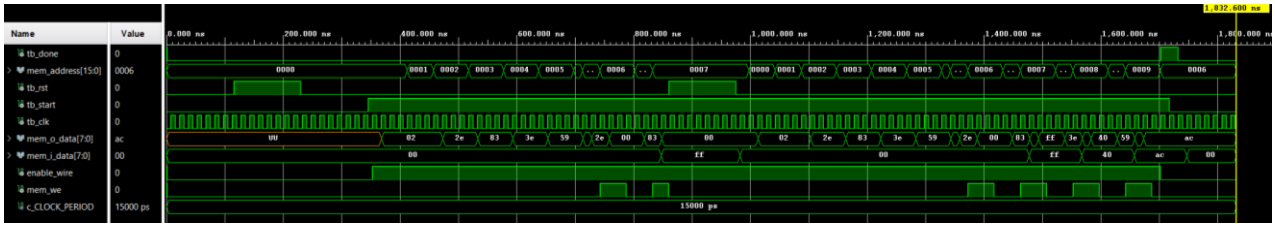
### 4.1 TB_2x2

È il test bench fornito, testa una semplice immagine quadrata di 4 pixel, è un test basilare.



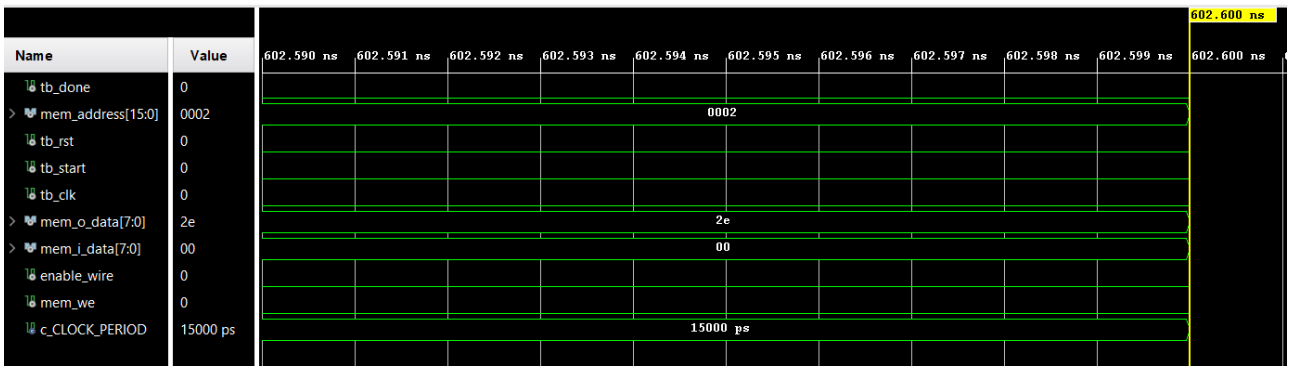

### 4.2 ResetAsincrono

Fornisce un reset durante la computazione che resetta lo stato della macchina e fa ripartire la lettura.



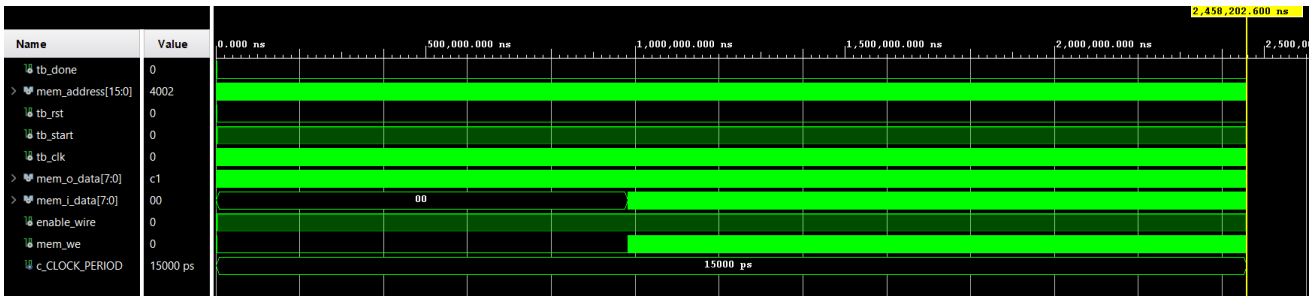
### 4.3 ZeroPixel

Fornisce "un'immagine" di 0x2 pixel, ovvero di 0 pixel.



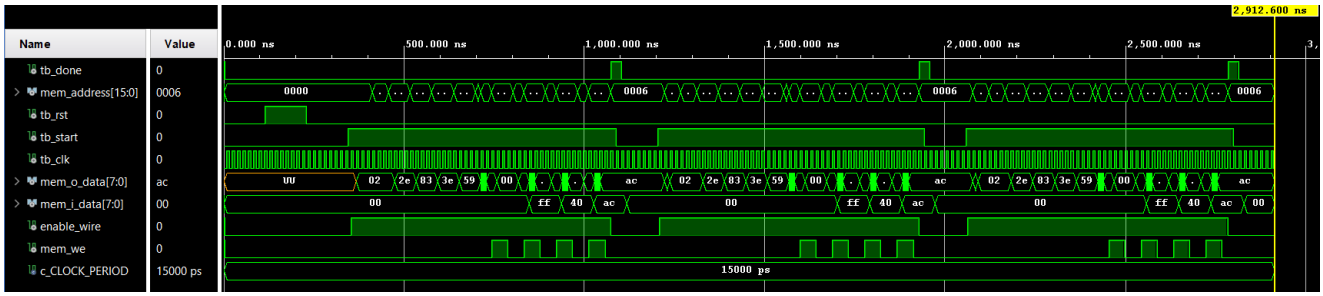
### 4.4 DimensioneMassima

Fornisce un'immagine di dimensione massima secondo la specifica, ovvero di 128x128 pixel (per completezza ne vengono riportati i segnali, tuttavia sono troppi per poterli distinguere chiaramente)




### 4.5 Test3Immagini

Fornisce 3 immagini consecutive, da specifica è possibile ricevere consecutivamente svariate immagini.




## 5. Conclusioni

Il componente implementato funziona sia in Beavioral Simulation che in Post Synthesis Functional, fino ad un periodo di clock di 13 ns.

Sono state fatte delle scelte progettuali, in particolare si può notare nel codice la presenza di alcune moltiplicazioni (operazione *) che normalmente dovrebbero essere evitate in quanto comportano un forte dispendio di risorse del componente; tuttavia vari tentativi di perfezionamento hanno portato al mantenimento di queste operazioni che sono risultate meno dispendiose (in termini di hardware) rispetto ad uno stato sostitutivo.