

# Prova Finale Progetto di Reti Logiche 2020/2021

Professore Fabio Salice

Alessandro Mazza

Cod. Persona 10670112 Matricola 908651

Davide Marinotto

Cod. Persona 10675966 Matricola 909291

## Indice

|  |          |
|--|----------|
| <b>1. Introduzione .....</b>                                 | <b>1</b> |
| <i>1.1. Scopo .....</i>                                      | <i>1</i> |
| <i>1.2. Specifica del progetto .....</i>                     | <i>1</i> |
| <i>1.3. Interfaccia del Componente .....</i>                 | <i>2</i> |
| <b>2. Architettura.....</b>                                  | <b>3</b> |
| <i>2.1. Rappresentazione grafica del funzionamento .....</i> | <i>3</i> |
| <i>2.2. Stati interni .....</i>                              | <i>4</i> |
| <i>2.3. Segnali Interni.....</i>                             | <i>5</i> |
| <i>2.4. Scelte Progettuali .....</i>                         | <i>6</i> |
| <b>3. Risultati sperimentali .....</b>                       | <b>7</b> |
| <i>3.1. Uso di risorse .....</i>                             | <i>7</i> |
| <b>4. Simulazioni .....</b>                                  | <b>8</b> |
| <b>5. Conclusioni .....</b>                                  | <b>9</b> |

# 1. Introduzione

## 1.1. Scopo

Lo scopo del progetto è la realizzazione di un componente hardware il cui funzionamento è ispirato al metodo di equalizzazione dell'istogramma di un'immagine, pensato per ricalibrare il contrasto dell'immagine data in input distribuendo i valori di intensità dei pixel su un intervallo più ampio.

Il componente implementa dunque una versione semplificata di questo metodo, dove le immagini in input sono rappresentate come una sequenza di valori interi che rappresentano l'intensità su una scala di grigi dei pixel, seguendo questo algoritmo:

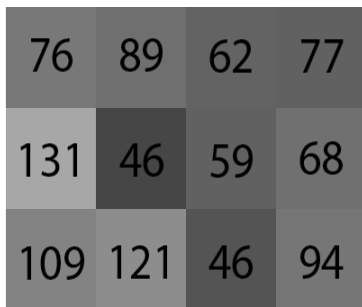
```
DELTA_VALUE = MAX_PIXEL_VALUE - MIN_PIXEL_VALUE
```

```
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.



|     |     |    |    |
|-----|-----|----|----|
| 76  | 89  | 62 | 77 |
| 131 | 46  | 59 | 68 |
| 109 | 121 | 46 | 94 |

Fig 1 Immagine da equalizzare



|     |     |    |     |
|-----|-----|----|-----|
| 120 | 172 | 64 | 124 |
| 255 | 0   | 52 | 88  |
| 252 | 255 | 0  | 192 |

Fig 2 Immagine equalizzata

## 1.2. Specifica del progetto

Il componente implementato legge l'immagine di input dalla memoria, memorizzata sequenzialmente e riga per riga, inoltre permette di codificare più immagini in maniera sequenziale, passando all'elaborazione di una nuova immagine solo dopo che l'esecuzione dell'immagine precedente è terminata. L'immagine in bianco e nero è formata da pixel codificati ciascuno con 8 bit in modo da rappresentare 256 possibili tonalità di grigio. Nell'indirizzo 0 e 1 della memoria vi è rappresentato rispettivamente il numero di colonne e di righe dell'immagine codificate anch'esse con 8 bit ma da specifica dovranno avere un valore decimale compreso tra 0 e 128. Dall'indirizzo 2 iniziano i pixel dell'immagine da equalizzare. La nuova immagine, ovvero il gruppo dei nuovi valori dei pixel calcolati dall'algoritmo di equalizzazione, verrà quindi scritta partendo dal primo indirizzo di memoria libero dopo l'immagine originale.

|                       |   |   |
|-----------------------|---|---|
| Numero di Colonne (n) | } | addr: 00000000                                    |
| Numero di Righe (m)   |   |   |
| Pixel da elaborare    | } | addr: 00000010<br>to<br>addr: ( m x n ) + 1       |
|                       |   |   |
| Pixel elaborati       | } | addr: ( m x n ) + 2<br>to<br>addr: 2( m x n ) + 1 |
|                       |   |   |

Fig 3 Struttura della Memoria RAM

### 1.3. Interfaccia del Componente

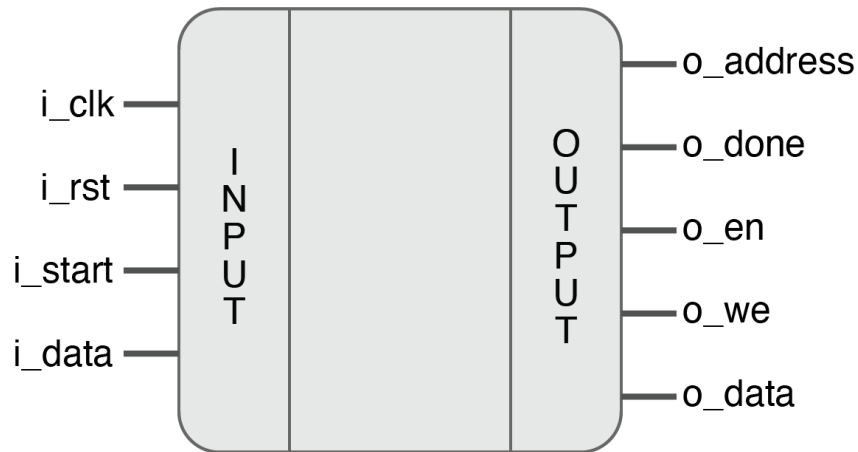
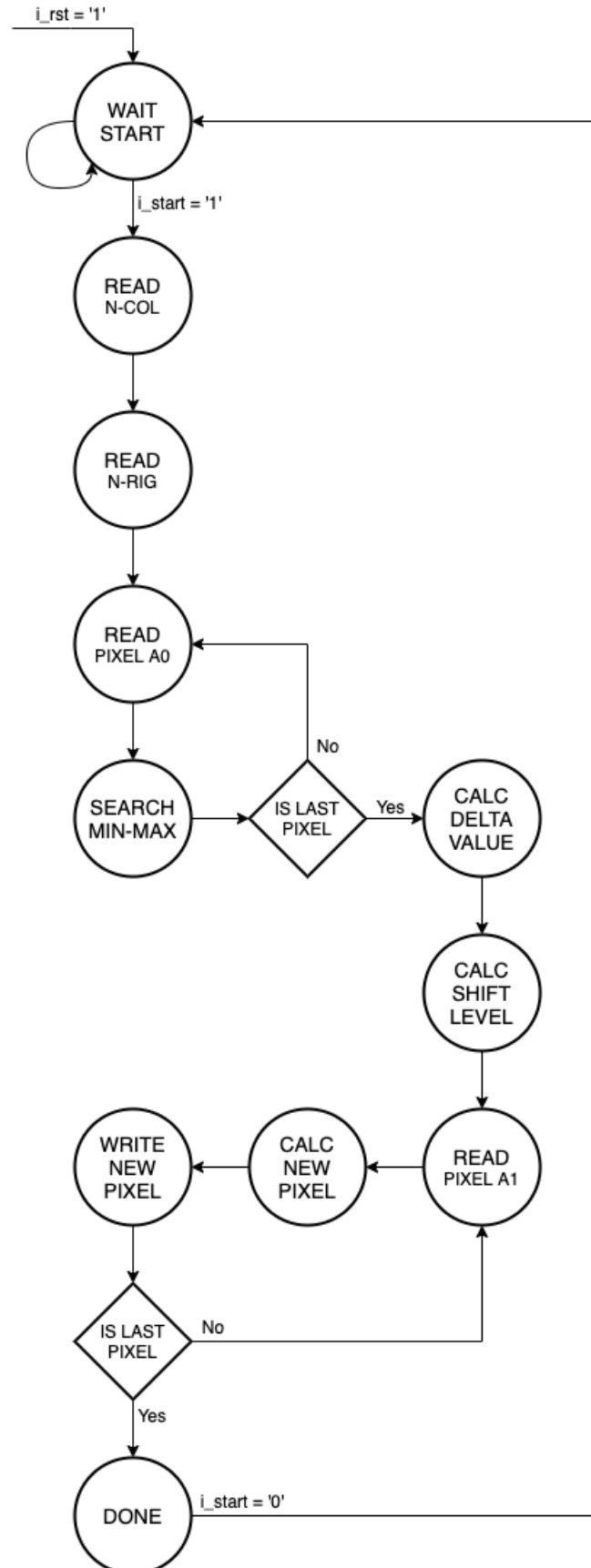


Fig 4 Rappresentazione grafica dell'interfaccia del componente

| Etichetta | Tipo di segnale                       | Descrizione   |
|-----------|---------------------------------------|---|
| i_clk     | in std_logic                          | Segnale di CLOCK in ingresso  |
| i_rst     | in std_logic                          | Segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START      |
| i_start   | in std_logic                          | Segnale di START che avvia il processo  |
| i_data    | in std_logic_vector<br>(7 downto 0)   | Segnale che arriva dalla memoria in seguito ad una richiesta di lettura                         |
| o_address | out std_logic_vector<br>(15 downto 0) | Segnale di uscita che manda l'indirizzo alla memoria  |
| o_done    | out std_logic                         | Segnale di uscita che comunica la fine dell'elaborazione e il dato di uscita scritto in memoria |
| o_en      | out std_logic                         | Segnale di ENABLE.<br>Settare a 1 per comunicare con la memoria sia in lettura che in scrittura |
| o_we      | out std_logic                         | Segnale di WRITE ENABLE.<br>Da settare a 1 per scrivere in memoria, invece a 0 per leggere      |
| o_data    | out std_logic_vector<br>(7 downto 0)  | Segnale di uscita dal componente verso la memoria   |

## 2. Architettura

### 2.1. Rappresentazione grafica del funzionamento



## 2.2. Stati interni

### WAIT START

Stato iniziale, può essere chiamato anche dal segnale `i_rst` in modo asincrono. Quando riceve il segnale `i_start` prosegue con lo stato `READ N-COL`.

### READ N-COL

Stato nel quale viene letto il numero di colonne dell'immagine e salvato nella variabile `ncol`. Proseguire con lo stato `READ N-RIG`.

### READ N-RIG

Stato nel quale viene letto il numero di righe dell'immagine salvato nella variabile `nrig` e calcolato il numero di pixel totali per poi salvare anche quest'ultimo nella variabile `npixel`. Proseguire con lo stato `READ PIXEL`.

### READ PIXEL A0

Stato nel quale viene letto il singolo pixel dell'immagine all'indirizzo memorizzato nella variabile `addrRead` e temporaneamente salvato nella variabile `pixel`, proseguire con lo stato `READ PIXEL`.

### SEARCH MIN-MAX

Stato nel quale si esegue un algoritmo di ricerca del minimo e del massimo confrontando i valori precedenti salvati rispettivamente sulle variabili `minpixel` e `maxpixel`. Se il pixel confrontato è l'ultimo dell'immagine da equalizzare allora prosegue con `CALC DELTA VALUE` altrimenti ritorna a `READ PIXEL` incrementando la variabile `addrRead`.

### CALC DELTA VALUE

Stato che calcola il valore di `DELTA_VALUE` come descritto dall'algoritmo di equalizzazione, ovvero  $DELTA\_VALUE = MAX\_PIXEL\_VALUE - MIN\_PIXEL\_VALUE$ . Salva il risultato nella variabile `deltavalue` e prosegue con lo stato `CALC SHIFT LEVEL`.

### CALC SHIFT LEVEL

Stato che calcola il valore di `SHIFT_LEVEL` solo tramite il valore della variabile `deltavalue`. Salva il risultato nella variabile `shiftval` e prosegue con lo stato `CALC NEW PIXEL`.

### READ PIXEL A1

Stato nel quale viene letto il singolo pixel dell'immagine all'indirizzo memorizzato nella variabile `addrRead` e temporaneamente salvato nella variabile `pixel`. Proseguire con lo stato `CALC NEW PIXEL`.

### CALC NEW PIXEL

Stato che calcola il valore del nuovo pixel leggendo l'originale all'indirizzo salvato in `addrRead`. Il calcolo che esegue è esattamente quello descritto dall'algoritmo di equalizzazione, ovvero:  
 $TEMP\_PIXEL = (CURRENT\_PIXEL\_VALUE - MIN\_PIXEL\_VALUE) \ll SHIFT\_LEVEL$   
 $NEW\_PIXEL\_VALUE = MIN(255, TEMP\_PIXEL)$   
Salva il valore nella variabile `pixel`.

### WRITE NEW PIXEL

Stato che scrive il valore della variabile `pixel` in memoria all'indirizzo `addrWrite`. Se il pixel è l'ultimo dell'immagine da equalizzare allora prosegue con `DONE` altrimenti ritorna a `READ PIXEL A1` incrementando la variabile `addrRead`.

### DONE

Stato che imposta il segnale `o_done` a '1' e procede con lo stato `WAIT START`

## 2.3. Segnali Interni

### **state\_curr**

Segnale di tipo `state`. Indica lo stato corrente in cui si trova la macchina.

### **state\_next**

Segnale di tipo `state`. Indica lo stato prossimo della macchina.

### **ncol**

Segnale di tipo `unsigned` codificato su 8 bit (7 downto 0).

Viene utilizzato per salvare il numero di colonne dell'immagine.

### **nrig**

Segnale di tipo `unsigned` codificato su 8 bit (7 downto 0).

Viene utilizzato per salvare il numero di righe dell'immagine.

### **npixel**

Segnale di tipo `unsigned` codificato su 16 bit (15 downto 0).

Viene utilizzato per salvare il numero di pixel dell'immagine.

### **pixel**

Segnale di tipo `unsigned` codificato su 8 bit (7 downto 0).

Viene utilizzato in tre diversi momenti dal componente.

Il primo utilizzo è per salvare il valore di un pixel dell'immagine alla volta per poi cercare il massimo e il minimo. Il secondo utilizzo è la seconda scansione dell'immagine dove salva sempre qui i valori letti dei pixel e il terzo utilizzo è salvare il risultato dell'algoritmo e quindi il nuovo pixel in modo poi da scriverlo sulla memoria.

### **minpixel**

Segnale di tipo `unsigned` codificato su 8 bit (7 downto 0).

Viene utilizzato per salvare il valore del pixel più basso dell'immagine originale.

### **maxpixel**

Segnale di tipo `unsigned` codificato su 8 bit (7 downto 0).

Viene utilizzato per salvare il valore del pixel più alto dell'immagine originale.

### **addrRead**

Segnale di tipo `unsigned` codificato su 16 bit (15 downto 0).

Viene utilizzato per scorrere tutti i pixel dell'immagine originale un indirizzo alla volta

### **addrWrite**

Segnale di tipo `unsigned` codificato su 16 bit (15 downto 0).

Viene utilizzato per scorrere tutti i pixel dell'immagine equalizzata un indirizzo alla volta.

### **deltavalue**

Segnale di tipo `integer` con valori compresi nell'intervallo chiuso tra 0 e 255.

Viene utilizzato per salvare il valore `deltavalue` una volta calcolato.

### **shiftval**

Segnale di tipo `integer` con valori compresi nell'intervallo chiuso tra 0 e 8.

Viene utilizzato per salvare il valore `shiftval` una volta calcolato.

### **tmpshiftval**

Segnale di tipo `integer` con valori compresi nell'intervallo chiuso tra 0 e 8.

## 2.4. Scelte Progettuali

Nello sviluppo del progetto abbiamo fatto tre scelte progettuali che riteniamo importanti da citare. La prima riguarda la lettura e scrittura nella memoria RAM: queste fasi si dividono in tre stati consecutivi che fanno rispettivamente la richiesta del dato a un indirizzo, aspettano la memoria e poi o leggono dall'indirizzo o scrivono all'indirizzo.

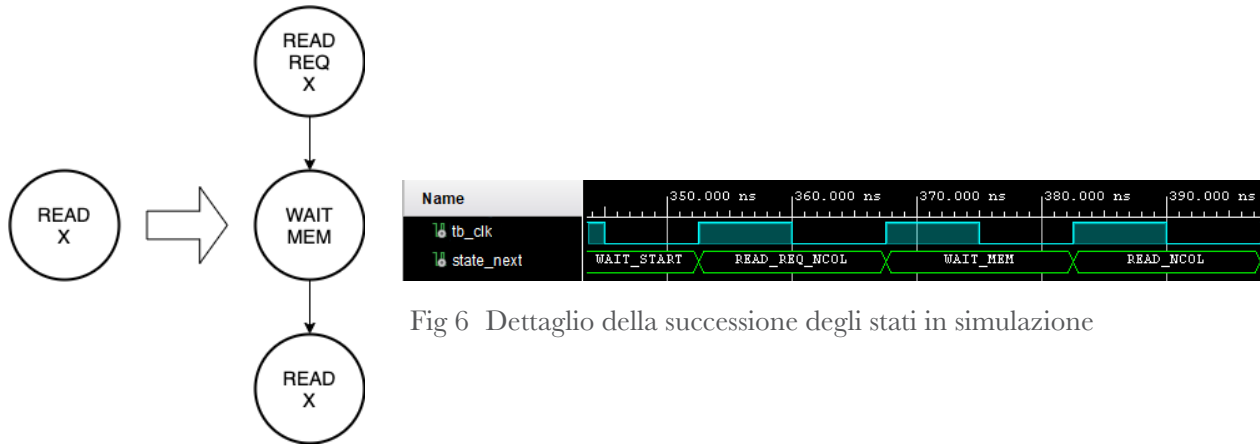


Fig 6 Dettaglio della successione degli stati in simulazione

Fig 5 Implementazione reale di una READ

La seconda scelta importante che abbiamo fatto riguarda un'implementazione ottimizzata del calcolo del valore *Shift Level*. Questa implementazione non esegue propriamente il calcolo richiesto ma implementa una semplice tabella di possibilità, i cui valori si basano sui risultati del calcolo nell'intervallo di valori richiesto, in tal modo con il solo valore di *Delta Value* il componente può dedurre in modo diretto quale sarà lo *Shift Level*.

| DELTA VALUE | SHIFT LEVEL |
|-------------|-------------|
| 0           | 8           |
| [1,2]       | 7           |
| [3,6]       | 6           |
| [7,14]      | 5           |
| [15,30]     | 4           |
| [31,62]     | 3           |
| [63,126]    | 2           |
| [127,254]   | 1           |
| 255         | 0           |

Fig 7 Tabella dei risultati di SHIFT LEVEL in funzione di DELTA VALUE

La terza ed ultima scelta riguarda una semplice ottimizzazione del particolare caso in cui si voglia provare a elaborare un'immagine vuota, ovvero con zero pixel. Leggendo il valore 0 come numero di colonne il componente non esegue le successive letture ma passa direttamente allo stato *DONE*, completando direttamente l'esecuzione dell'immagine corrente e iniziando eventualmente l'esecuzione del processo sull'immagine successiva.

### 3. Risultati sperimentali

Una volta assicurato il corretto esito delle simulazioni in fase Behavioural siamo passati alla Sintesi sfruttando i tool forniti da XILINX VIVADO, usando la FPGA target xc7a200tsbg484-1 da cui abbiamo ottenuto un design con i seguenti valori.

#### 3.1. Uso di risorse

| Site Type             | Usate | Disponibili | Utilizzo % |
|-----------------------|-------|-------------|------------|
| Slice LUTs *          | 202   | 134600      | 0.15       |
| LUTs as Logic         | 202   | 134600      | 0.15       |
| LUTs as Memory        | 0     | 46200       | 0.00       |
| Slice Registers       | 131   | 269200      | 0.05       |
| Register as Flip Flop | 131   | 269200      | 0.05       |
| Register as Latch     | 0     | 269200      | 0.00       |
| IO                    | 38    | 285         | 13.33      |
| BUFG                  | 1     | 32          | 3.13       |

\* Il numero di Look Up Tables è solitamente inferiore una volta completata l'implementazione fisica con le relative ottimizzazioni, questi dati invece si basano sulla sola sintesi e quindi potrebbero non essere completamente realistici.

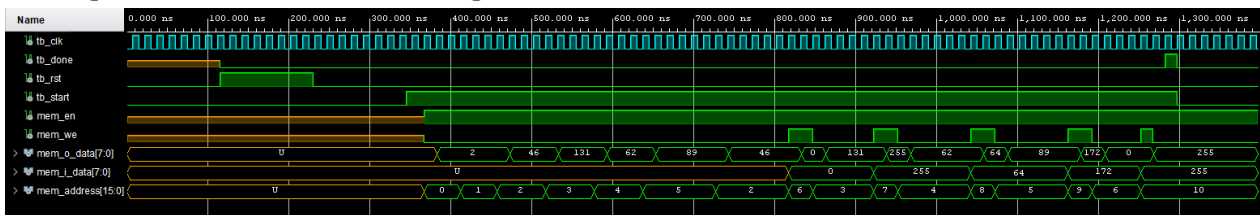


## 4. Simulazioni

Il componente compie simulazioni corrette sia in fase Behavioural che in Post-Sintesi, e ne abbiamo verificato il funzionamento sia in casi generali sia nei casi limite riportati in seguito, in modo da essere sicuri che il componente funzionasse correttamente anche nei casi più particolari della computazione. Per mantenere la leggibilità delle simulazioni in questa relazione, solo alcuni test presentano la relativa forma d'onda. Allo stesso modo, non sono riportati i tempi di esecuzione dei test in quanto dipendenti dalla macchina su cui si effettua la simulazione, e quindi non sono considerabili dati assoluti o completamente affidabili.

### Test base

Verifica il corretto funzionamento in un caso generico usando il test bench fornito, ovvero un'unica immagine 2x2 che non fa uso del segnale di reset.

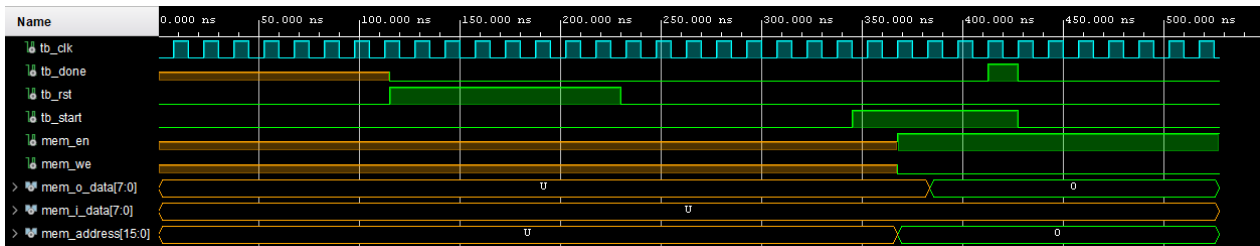


### Test di dimensione massima

Verifica il corretto funzionamento nel caso dal tempo di esecuzione più grande, ovvero la rielaborazione di un'immagine di dimensione 128x128, come richiesto dalla specifica.

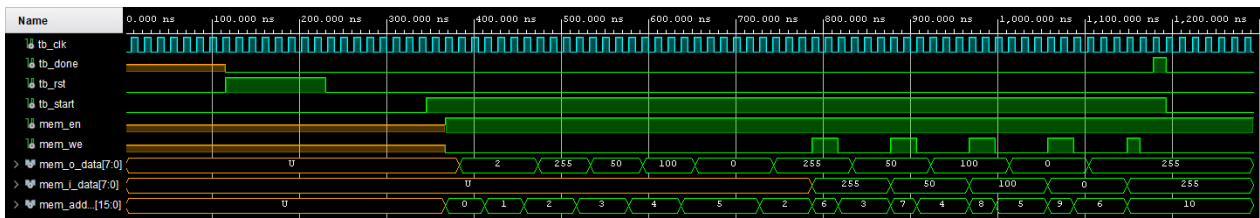
### Test di dimensione minima

In modo analogo al test precedente, verifica il funzionamento nel caso di un'immagine di dimensione 0x0 che non contiene alcun pixel, seguendo la nostra scelta di design di concludere direttamente l'esecuzione quando legge il numero di colonne uguale a 0.



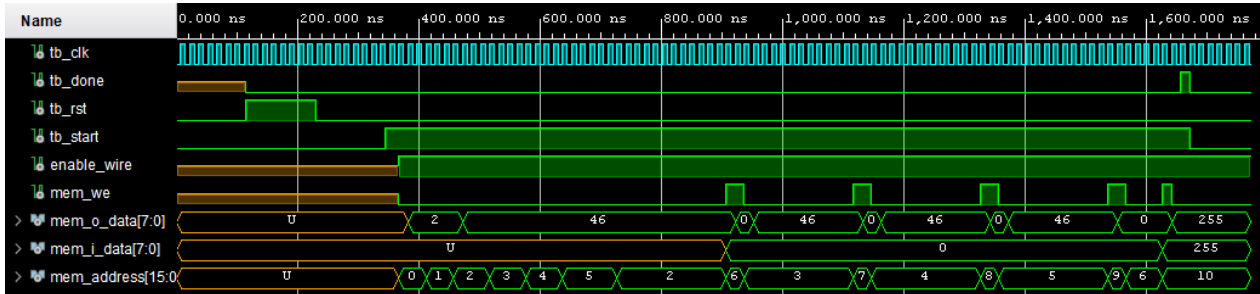
### Test con immagine dagli ampi valori

Verifica il caso particolare in cui i pixel dell'immagine già di partenza coprono l'intero intervallo [0, 255] con i valori minimo (pixel di valore 0) e massimo (pixel di valore 255). Abbiamo deciso di verificare questo stato particolare in quanto è l'unico in cui il segnale shiftval è uguale a 0, ovvero immagine di input e output risultano uguali.



### ***Test con immagine monocromatica***

Verifica il caso particolare in cui tutti i pixel hanno lo stesso valore sulla scala di grigi. In tal caso, l'immagine di output sarà composta di tutti zeri. Questo caso particolare include anche il caso di un'immagine 1x1, che essendo composta da un solo pixel sarà necessariamente un'immagine monocromatica, e abbiamo deciso di verificarlo in quanto è l'unico caso in cui il segnale `shiftval` è uguale a 8.



### ***Test con reset asincrono***

Verifica il corretto funzionamento del reset asincrono, ovvero che il componente ricominci l'elaborazione dell'immagine corrente quando il segnale `i_rst` si alza a 1 prima di aver completato l'elaborazione stessa, come richiesto da specifica.

### ***Test con immagini multiple***

Verifica la corretta elaborazione di più immagini presenti in memoria, passando all'immagine successiva solo una volta completata l'elaborazione di un'immagine, come richiesto da specifica.

## **5. Conclusioni**

Il componente implementato funziona correttamente come richiesto da specifica e l'implementazione di scelte di design come il calcolo del segnale `shiftvalue` e la gestione dell'immagine vuota lo rendono veloce ed efficiente, tanto da riuscire a lavorare correttamente con tempi di clock molto bassi (i vari test proposti usano un clock di 15 ns, ma il componente ha dimostrato di riuscire a lavorare con clock anche inferiori).

Questa efficienza di calcolo è però ottenuta a discapito della adattabilità del componente: lavorare su una scala di grigi con un diverso intervallo di valori richiederebbe di modificare di conseguenza i possibili valori di `shiftvalue` ottenibili, allo stesso modo espandere il componente per la modifica di immagini a colori richiederebbe un diverso calcolo e quindi diventerebbe necessario implementare una diversa tabella di valori.