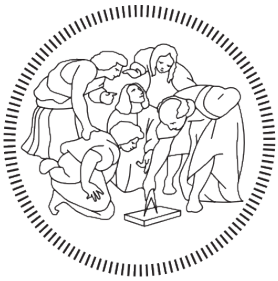

POLITECNICO
MILANO 1863

Prova Finale Reti Logiche

Giuseppe Calcagno
Matricola 910210 - Codice Persona 10659505

Anno Accademico 2020/2021

Indice

1 Introduzione

1.1 Scopo del Progetto

Lo scopo del progetto è, data un'immagine in scala di grigi a 256 livelli, realizzare un modulo hardware in grado di elaborare l'immagine eseguendo l'equalizzazione dell'istrogramma che consiste nell'aumento del contrasto tramite il ripristino dei punti più chiari e più scuri, nonché la successiva distribuzione uniforme della gamma cromatica sui pixel dell'immagine.

52	55	61	59	79	61	76	61
62	59	55	104	94	85	59	71
63	65	66	113	144	104	63	72
64	70	70	126	154	109	71	69
67	73	68	106	122	88	68	68
68	79	60	70	77	66	58	75
69	85	64	58	55	61	65	83
70	87	69	68	65	73	78	90

(a) Immagine Non Equalizzata

0	12	53	32	190	53	174	53
57	32	12	227	219	202	32	154
65	85	93	239	251	227	65	158
73	146	146	247	255	235	154	130
97	166	117	231	243	210	117	117
117	190	36	146	178	93	20	170
130	202	73	20	12	53	85	194
146	206	130	117	85	166	182	215

(b) Immagine Equalizzata

1.2 Specifiche Generali

Il modulo hardware dovrà leggere l'immagine da una memoria e dovrà supportare immagini di dimensioni massime pari a 128x128 pixel con un periodo di clock non superiore ai 100ns. Per l'elaborazione dell'immagine verrà usata una versione semplificata dell'algoritmo di equalizzazione:

$$\Delta Value = MaxPixelValue - minPixelValue$$

$$ShiftLevel = 8 - \lfloor \log_2(\Delta Value + 1) \rfloor$$

$$TempPixel = (currentPixelValue - minPixelValue) << ShiftLevel$$

$$NewPixelValue = min(255, TempPixel)$$

Dove *MaxPixelValue* e *minPixelValue* sono il massimo e il minimo valore dei pixel dell'immagine, *currentPixelValue* è il valore del pixel da trasformare e *NewPixelValue* è il valore del nuovo pixel.

1.3 Specifica Memoria

I dati dell'immagine, ciascuno di dimensione 8 bit, sono salvati in una memoria ad indirizzamento al byte. La dimensione dell'immagine è definita sui primi 2 byte della memoria, mentre i suoi pixel sono salvati a partire dall'indirizzo 2 in byte contigui. L'immagine equalizzata deve essere scritta in memoria immediatamente dopo l'immagine originale.

Memoria RAM:

[0] N° Colonne C
[1] N° Righe R
[2] Inizio Immagine
...
[C*R+1] fine immagine
[C*R+2] inizio risultato
...
[2(C*R)+2] fine risultato

1.4 Interfaccia del Componente

L'interfaccia del componente è la seguente:

```
entity project_reti_logiche is
port (
        i_clk :         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;
```

In particolare:

- i_clk è il segnale di *CLOCK* in ingresso generato dal TestBench
- i_rst è il segnale di *RESET* in ingresso generato dal TestBench
- i_start è il segnale di *START* in ingresso generato dal TestBench
- 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
- o_en è il segnale di *ENABLE* da dover mandare alla memoria per comunicare
- o_we è il segnale di *WRITEENABLE* da dover mandare alla memoria per poter scrivere
- o_data è il segnale (vettore) di uscita dal componente verso la memoria

2 Architettura

2.1 Scelte Progettuali

Ho deciso di dividere il componente in 2 parti: un **data path** composto da diversi registri (successivamente descritti), 2 processi in grado di calcolare il valore corretto dello *ShiftLevel* ed eseguire il calcolo del *TempPixel* ed infine alcuni selettori e comparatori grazie ai quali viene deciso il corretto valore dei registri. Il tutto è governato da una **Macchina a Stati Finiti** che, grazie a dei segnali di controllo, gestisce il passaggio da uno stato al successivo e quindi il funzionamento del data path. Ad inizio computazione la macchina è in uno stato di *Idle* e viene azionata quando il segnale di ingresso *i_start* viene posto ad "1". A questo punto il componente leggerà i primi 2 byte della memoria e calcolerà la dimensione dell'immagine e, se valida, inizierà la scansione di tutti i pixel dell'immagine per trovare i valori di *minPixelValue* e *MaxPixelValue*. Successivamente il registro degli indirizzi verrà riportato all'inizio dell'immagine e contemporaneamente verrà calcolato il valore di*ShiftValue*. Inizierà in questo stato il processo di lettura, calcolo del nuovo valore e scrittura in memoria di ogni pixel dell'immagine che terminerà portando a "1" il segnale di *o_done*. Quando il segnale di *i_start* sarà di nuovo posto ad un livello basso il componente tornerà nello stato di *Idle*.

2.2 Funzionamento nel dettaglio

2.2.1 Data Path

In questo paragrafo vengono descritti tutti i registri utilizzati e i segnali più importanti:

regCol	Contiene il numero di colonne dell'immagine
regRow	Contiene il numero di righe dell'immagine
regAddress	Utilizzato per salvare l'indirizzo di memoria da mandare in output
regDim	Contiene la dimensione dell'immagine
regMin	Contiene il valore di *minPixelValue*
regMax	Contiene il valore di *MaxPixelValue*
regPixel	Contiene il valore di un pixel dell'immagine decrementato di *minPixelValue*
regOdata	Contiene il valore da mandare in output alla memoria

checkIn	É alto quando il valore su *i_data* è pari a 0. Viene utilizzato per verificare che le dimensioni siano maggiori di 0
endSum	É alto quando *regRow*=0. Viene utilizzato per terminare il calcolo della dimensione dell'immagine
endjpg	É alto quando *regAddress* raggiunge l'ultimo Pixel dell'immagine. Viene utilizzato per terminare lo scorrimento dei pixel.
muxMinMax_load	É alto quando inizia la ricerca dei valori estremi. Viene utilizzato per non inserire i primi 2 byte nella ricerca di min e Max
o_address	É l'indirizzo in uscita dal data Path per impostare *o_address*
o_data	Sono i Dati da mandare in output su *o_data*

Tutti gli altri segnali vengono utilizzati per impostare il caricamento dei registri.

2.2.2 Macchina a Stati

In questo paragrafo vengono descritti tutti gli stati dell'automa:

SI	Stato di Idle: in questo stato vi è un segnale *datapath_rst* porto ad "1" e collegato al reset del data path che pone in datapath nel suo stato iniziale
S0	Stato dove viene richiesto alla memoria il primo byte in memoria e viene incrementato il registro regAddress.
S1	Stato dove si verifica che il primo byte non sia "0", se lo è la computazione termina e *o_done* viene portato ad "1" altrimenti viene impostato il caricamento del dato nel registro *regCol* e viene richiesto il secondo byte in memoria.
S2	Stato dove viene eseguito il medesimo controllo di S1 sul secondo byte. Se ha riscontro negativo si imposta il caricamento nel registro *regRow* e *regAddress* viene incrementato
S3	Stato dove viene calcolata la dimensione dell'immagine tramite somme successive su *regCol* e ad ogni operazione viene decrementato il valore di *regRow*. Quando *regRow*=0 il segnale *endsum* viene posto ad 1 dal datapath e lo stato cambia.
S4	Stato dove viene richiesto il dato dalla memoria e viene impostato l'incremento di *regAddress*.
S5	Stato dove viene analizzato il dato in Input per trovare *minPixelValue* e *MaxPixelValue*, allo stesso tempo viene incrementato *regAddress* e richiesto il nuovo dato alla memoria. Quando vengono letti tutti i Pixel *endjpg*=1 e lo stato cambia.
S6	Stato dove viene analizzato l'ultimo Pixel e viene riportato *regAddress* all'inizio dell'immagine
S7	Stato dove viene richiesto il dato alla memoria
S8	Stato dove viene calcolata da differenza tra il valore del pixel attuale e *minPixelValue*e impostato il caricamento sul registro *regPixel*
S8bis	Stato dove viene calcolato e impostato il caricato nel registro *regOdata* il corretto valore di *NewPixelValue*
S9	Stato dove viene impostato l'indirizzo di Output e richiesto il caricamento sulla memoria del dato presente in *regOdata*. Viene inoltre incrementato il valore di *regAddress*. L'ASF si riporta allo stato S7 finché il segnale *endjpg=0*, altrimenti passa a S10
S10	Stato dove si attende che i dati vengano scritti sulla memoria
S11	Stato che porta *o_done* ad un livello alto e si attende che *i_start* venga riportato a "0" per ritornare allo stato SI

2.3 Dettagli Implementativi

Per il calcolo dello *ShiftLevel* ho utilizzato 1 processo con controllo a soglia sulla differenza tra *regMax* e *RegMin*, ho deciso di non salvare il valore in un registro in quanto questo non varia dopo aver impostato i 2 registri. Per il calcolo di *TempPixel* ho utilizzato invece un processo con controllo a soglia su *ShiftLevel* che traslava il valore di *regPixel* del valore corretto. Non ho utilizzato nessun registro per dividere i_data dal calcolo degli estremi e dal calcolo di regPixel in quanto il periodo di clock minimo (100ns) è più che sufficiente per il calcolo di questi valori. Infine ho deciso di utilizzare lo stato S10 per attendere che tutti i valori sulla memoria si assestino.

Di seguito è riportata una rappresentazione grafica della macchina a stati e del data path:

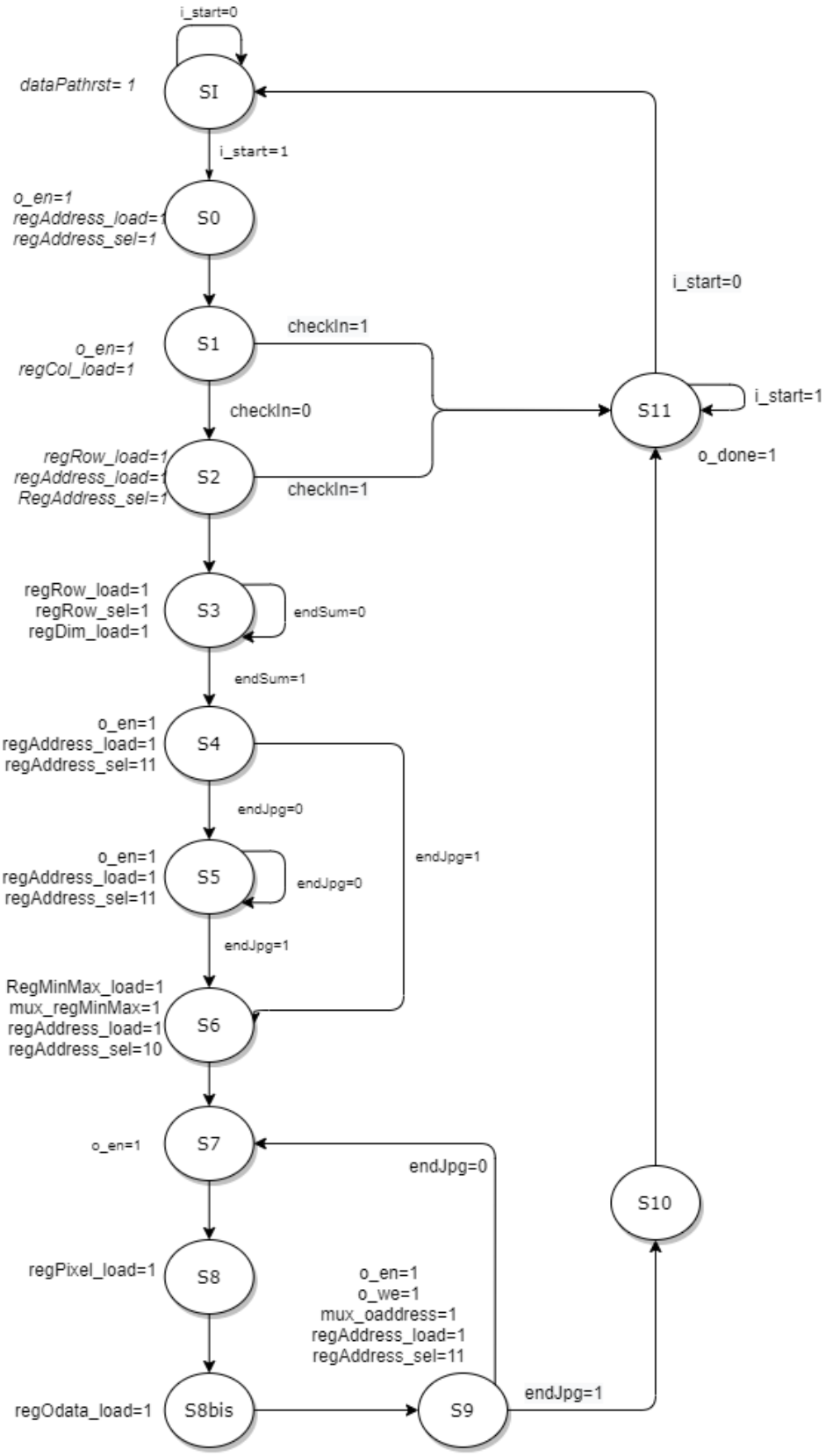

Figure 2: ASF del componente

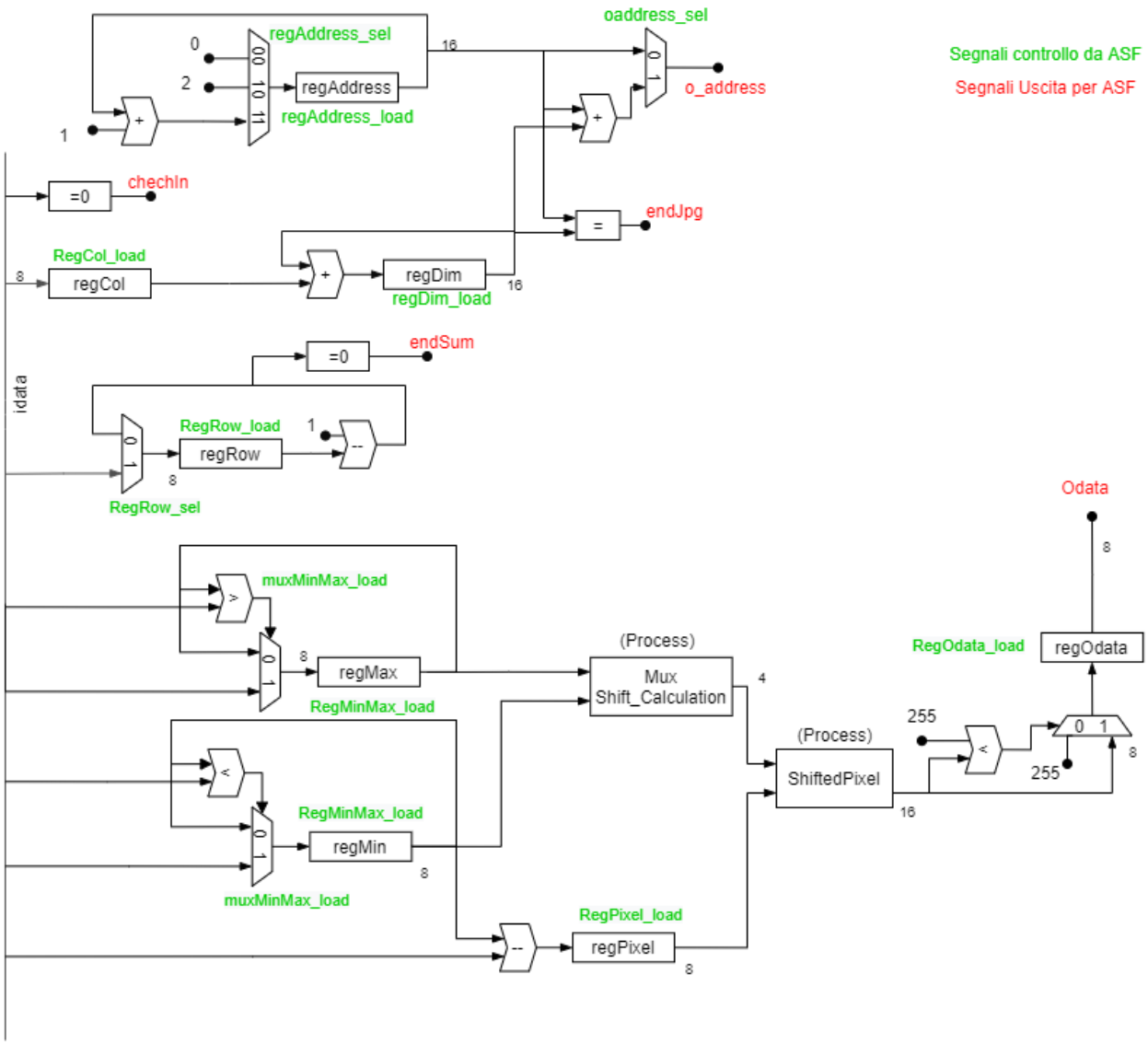

Figure 3: Data Path del Componente

Sono stati omessi i segnali di Clock e Reset dal data path per una questione di visibilità

3 Risultati Sperimentali

Dopo aver descritto il componente ho utilizzato *Vivado* per sintetizzarlo e analizzarlo.

3.1 Report Utilization

Analizzato il *Synthesis Report* trovo che il numero di FF utilizzati corrisponde alla somma dei FF utilizzati per i registri con i FF utilizzati per sintetizzare gli stati con codifica one-hot. Come previsto non sono stati inseriti latch inferiti e la percentuale di utilizzo della FPGA risulta irrisoria. Ciò conferma la scelta di non ritenere importante la riduzione dell'area occupata in fase di progettazione.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	141	0	41000	0.34
LUT as Logic	141	0	41000	0.34
LUT as Memory	0	0	13400	0.00
Slice Registers	94	0	82000	0.11
Register as Flip Flop	94	0	82000	0.11
Register as Latch	0	0	82000	0.00
F7 Muxes	0	0	20500	0.00
F8 Muxes	0	0	10250	0.00

Figure 4: Report Utilization Vivado

3.2 Timing Report

Analizzando invece il *Timing Report* posso vedere quanto del periodo di clock è effettivamente utile per svolgere il lavoro. Dalla voce *Data Path Delay* noto che il segnale più lento impiega per commutare 3,681ns quindi posso portare la frequenza di utilizzo del mio componente fino a 271Mhz, abbondantemente sopra la frequenza minima di 10Mhz. Questo conferma la scelta progettuale di non dividere il percorso critico di selezione di *regMin* e *regMax* ed inoltre il calcolo di *regPixel* dall'ingresso con un registro. Viene inoltre confermata la scelta di non dividere in più cicli di clock il calcolo di *oData*.

```
Timing Report

Slack (MET) :             95.897ns  (required time - arrival time)
  Source:                 DataPath/regDim_reg[4]/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@5.000ns period=100.000ns}
  Destination:            FSM_onehot_cur_state_reg[0]/CE
                            (rising edge-triggered cell FDPE clocked by clock  {rise@0.000ns fall@5.000ns period=100.000ns}
  Path Group:             clock
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            100.000ns  (clock rise@100.000ns - clock rise@0.000ns)
  Data Path Delay:        3.681ns  (logic 1.626ns (44.173%)  route 2.055ns (55.827%))
  Logic Levels:           8  (CARRY4=5 LUT6=3)
  Clock Path Skew:        -0.145ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    1.860ns = ( 101.860 - 100.000 )
    Source Clock Delay      (SCD):    2.117ns
    Clock Pessimism Removal (CPR):    0.112ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
```

Figure 5: Report Timing Vivado

3.3 Testing

Per testare la correttezza del mio componente ho utilizzato dei Test Bench randomici di dimensioni e valore variabile su più esecuzioni consecutive. Inoltre ho individuato dei casi limite per cui ho scritto dei test appositi:

- **Immagine Vuota**: Con questo test ho voluto verificare il controllo sulle dimensioni dell'immagine con il corrispettivo cambio del segnale di *o_done*

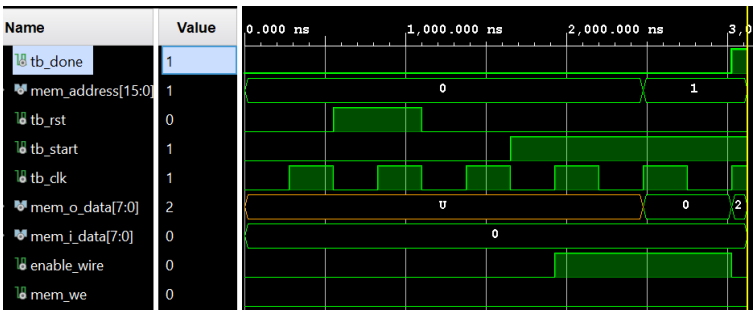

Figure 6: Simulazione Post-Synthesis Immagine di Dimensione 0

- **Tutti i Pixel a 255**: Con questo test ho voluto verificare il comportamento per i valori estremi di pixel (Sono stati eseguiti test anche per tutti i pixel pari a 0 ed ad un valore intermedio, non riportati per evitare ridondanza). Si nota dall'immagine che tutti i pixel vengono portati a 0 come da algoritmo.

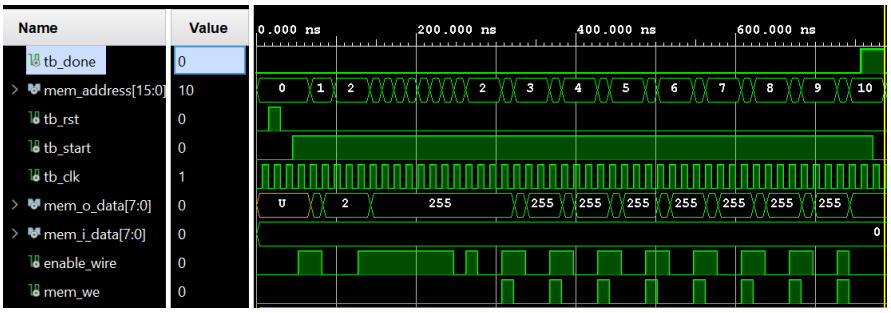

Figure 7: Simulazione Post-Synthesis Immagine Con tutti i pixel a 255

- **Reset Asincrono**: Con questo test ho voluto verificare il corretto comportamento del componente nel caso di Reset Asincrono. Nel grafico troviamo i segnali di un test bench dove viene iniziata l'elaborazione della prima immagine che viene fermata da un Reset asincrono. Successivamente viene iniziata una nuova elaborazione portata a termine con successo.

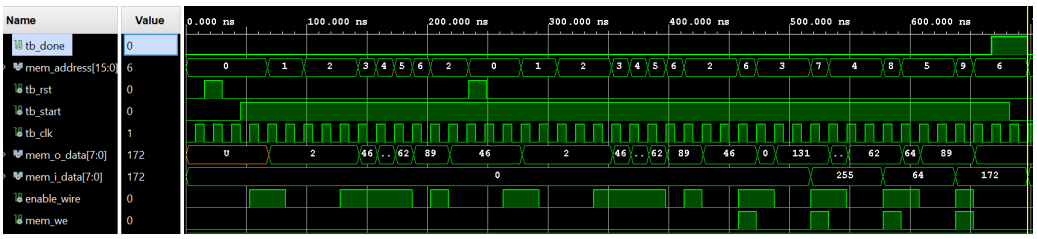

Figure 8: Simulazione Post-Synthesis con Reset Asincrono

Inoltre sono stati testati i casi con dimensione massima a anche con dimensione pari a 1. Tutti i test sono Stati superati sia in *Behavioral* che in *Post-Synthesis* simulation

4 Conclusioni

Grazie a questo progetto ho toccato con mano la differenza tra programmazione software e simulazione hardware, con tutti i suoi pro e i suoi contro. Ho imparato a valutare le caratteristiche del componente descritto e capire le conseguenze che le scelte di progettazione hanno sulle prestazioni, così da prendere la decisione migliore in base alle specifiche richieste.