Prova Finale

Progetto Reti Logiche AA 2020/2021:

Progettazione componente VHDL per equalizzazione istogramma di un'immagine a scala di grigi a 256 livelli.

POLITECNICO
MILANO 1863

Relatore: Giulio Saulle

Matricola: 890805

Docente: Gianluca Palermo

INDICE

Introduzione

1.1 Obbiettivi e Algoritmo

Il progetto di seguito presentato si pone l'obbiettivo di modificare un'immagine data per poterne ottenere una identica, in cui i contrasti risultino calibrati secondo specifica: in questo caso si prendono in considerazione immagini di dimensioni massime 128x128 pixel, con scala di grigi a 256 livelli.

Tramite una versione semplificata dell'algoritmo possiamo automatizzare il processo costruendo un componente apposito in VHDL:

```
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)
```

I valori di *MAX_PIXEL_VALUE* e *MIN_PIXEL_VALUE* sono rispettivamente il massimo e il minimo valore dei pixel dell'immagine, quindi possono assumere al massimo valori da 0 a 255.

SHIFT_LEVEL è uno scostamento a livello di bit del valore corrente calcolato sulla base del *DELTA_VALUE*. Il valore *CURRENT_PIXEL_VALUE* è il valore del pixel corrente da modificare, passandolo a *NEW_PIXEL_VALUE*, che lo andrà a sovrascrivere in memoria.

Entrambi sono salvati come sequenza di 8 bit in un blocco di memoria RAM, quindi indirizzata a byte, inizialmente contenente i seguenti dati:

INDIRIZZO MEMORIA	VALORE
0	Byte più significativo NUMERO COLONNE
1	Byte più significativo NUMERO RIGHE
2	Primo pixel
3	Secondo pixel
…	. . .
N + 1	Ultimo pixel immagine

Conseguentemente all'esecuzione del processo mi aspetto una memoria strutturata come segue:

INDIRIZZO MEMORIA	VALORE
0	Byte più significativo NUMERO COLONNE
1	Byte più significativo NUMERO RIGHE
2	Primo pixel
3	Secondo pixel
…	. . .
N +1	Ultimo pixel immagine
N + 2	Primo pixel equalizzato
N + 3	Secondo pixel equalizzato
…	. . .
N + (*N* + 1)	Ultimo pixel equalizzato

Dove N è il numero di pixel dell'immagine (che viene maggiorato di uno data la presenza delle prime due righe occupate). Le celle 0 e 1 saranno sempre occupate dalle dimensioni di colonne e righe.

1.2 Note

Alcune considerazioni che si possono fare sulla specifica della richiesta sono:

- *FLOOR(LOG2(DELTA_VALUE +1))* è un numero intero con valori compresi tra 0 e 7, ricavabile tramite controlli di soglia.
- Il modulo è progettato per codificare più immagini, ma l'immagine da codificare non verrà mai cambiata all'interno della stessa esecuzione.
- Il modulo sarà sempre inizializzato correttamente a livello di segnali, mentre per le successive elaborazioni il setup è gestito dal componente stesso.
- La memoria è stata popolata da vari ed eventuali TestBench che determinano le casistiche a cui il componente deve far fronte, non tutti i test hanno come risultato un'immagine poiché vengono utilizzati per studiare gli *Edge Cases*.

Componenti e Architettura

2.1 Interfaccia componente project_reti_logiche

```
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;
```

I segnali presenti nel seguente modulo sono:

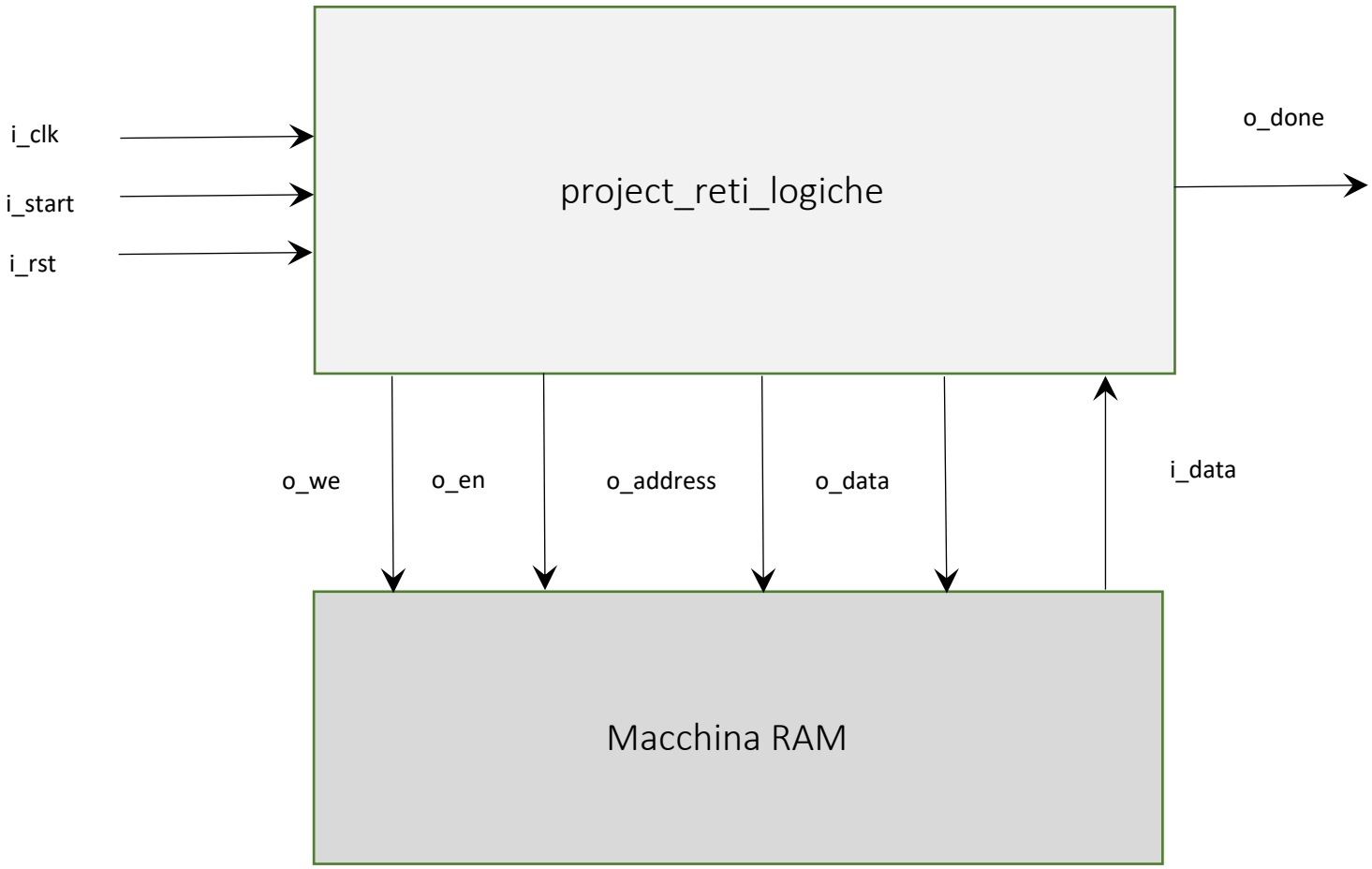
- **i_clk** è il segnale di CLOCK in ingresso (periodo di clock almeno di 100 *ns*).
- **i_rst** è il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START.
- **i_start** è il segnale di START.
- **i_data** è il segnale (vettore) che arriva dalla memoria in seguito 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 dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura).
- **o_we** è il segnale di WRITE ENABLE da dover mandare alla memoria (=1) per poter scriverci. Per leggere da memoria esso deve essere 0.
- **o_data** è il segnale (vettore) di uscita dal componente verso la memoria.

Il modulo partirà nella elaborazione quando un segnale *i_start* in ingresso verrà portato a 1, dopodiché il segnale di START rimarrà alto fino a che il segnale di *o_done* non verrà portato alto. Al termine della computazione (e una volta scritto l'ultimo pixel in memoria), il modulo alza il segnale *o_done* che notifica la fine dell'elaborazione. Il segnale *o_done* rimane alto fino a che il segnale di *i_start* non è riportato a 0!

Sapendo che "un nuovo segnale start non può essere dato fin tanto che *o_done* non è stato riportato a zero" il componente è stato progettato seguendo questa specifica. Se a questo punto viene rialzato il segnale di *i_start*, il modulo ripartirà con la fase di codifica (nuovo setup).

2.2 Architettura

Il componente è implementato a modulo singolo, gestendo gli stati sequenzialmente per poter implementare la macchina a stati in maniera corretta. Tale scelta progettuale è dovuta alla dimestichezza con cui modificare il codice in seguito e la volontà di rendere tale modulo adibito a una singola routine, potendolo interfacciare esternamente a componenti in cascata se richiesto.
Essa viene avviata dal segnale *i_start* e conclusa con *o_done.*
Di seguito la macchina sequenziale:

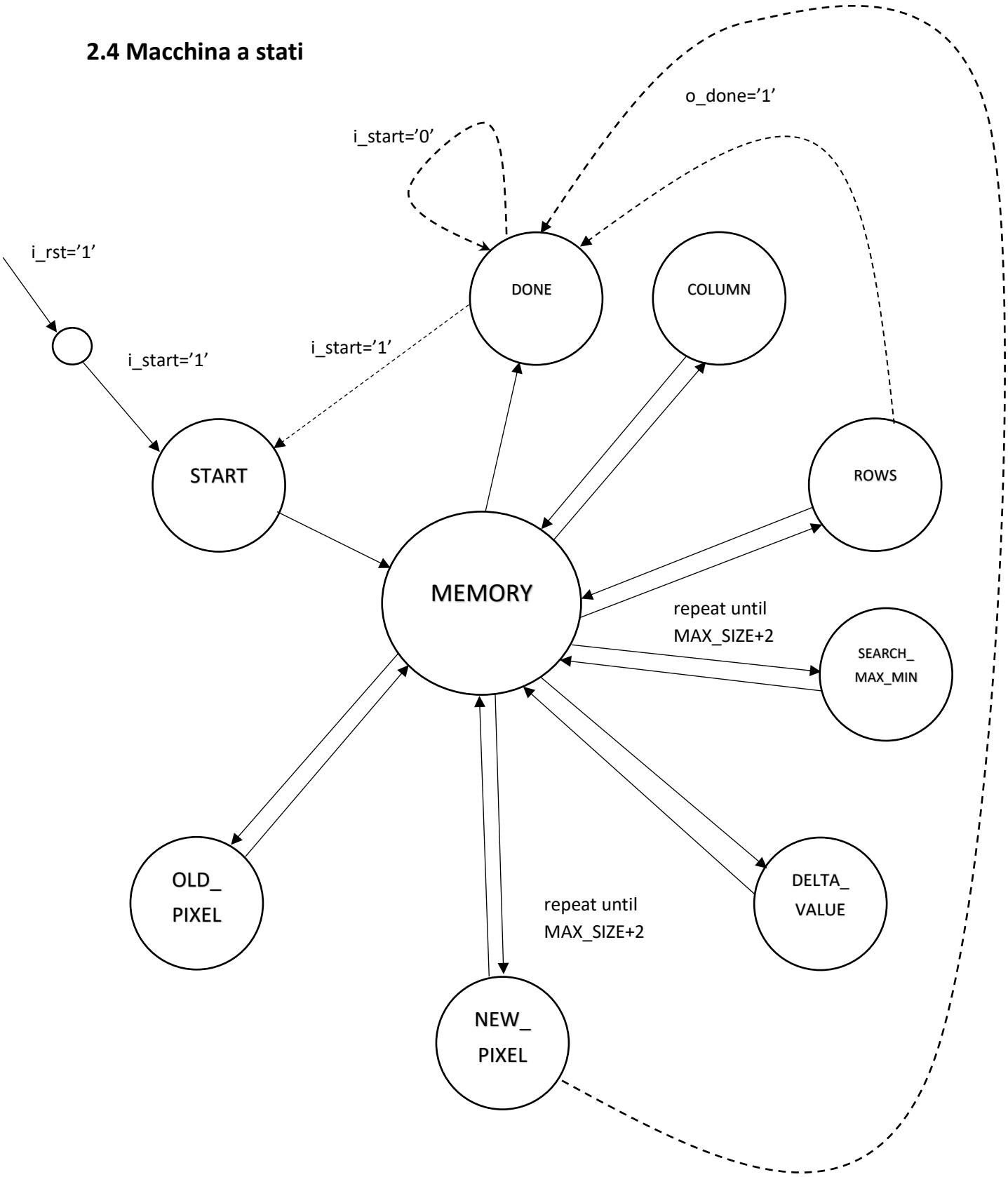


2.3 Segnali interni

```
signal STATE, P_STATE: state_type;
signal MAX: std_logic_vector (7 downto 0);
signal MIN: std_logic_vector (7 downto 0);
signal DELTA: std_logic_vector (7 downto 0);
signal shift_level: integer range 0 to 8;
```

- **STATE:** indica lo stato attuale del componente, permettendo alla macchina a stati di essere implementata.
- **P_STATE:** indica lo stato precedente al corrente e nel caso si inizi la computazione esso è posto a START. L'importanza di questo segnale deriva dal fatto che per scegliere il prossimo stato viene comparato con i possibili stati del componente, permettendo di fare delle scelte a seconda delle variabili (cosa molto importante nella gestione degli Edge Cases).
- **MAX:** contiene il valore del pixel più scuro.
- **MIN:** contiene il valore del pixel più chiaro.
- **DELTA:** contiene il valore MAX – MIN + 1.
- **shift_level:** contiene il valore che determina di quanti bit il pixel deve scorrere a sinistra. Tale valore è calcolato tramite controlli di soglia, comparando i bit di DELTA in modo decrescente, dato che il valore FLOOR(LOG2(DELTA_VALUE +1)) è un intero tra 0 e 7 (+1), e implicitamente sottratto a 8 per ottenere il corretto valore di shift (i valori assegnati sono già quelli finali).

2.4 Macchina a stati



La macchina a stati è composta da 9 stati, di cui uno principale usato per caricare le variabili da e verso memoria e check intra-stato per reindirizzare lo stato corrente. Lo stato subito dopo il comando i_*rst*='1' viene attraversato solo in fase di inizializzazione, dove avviene la fase di setup.

Gli stati e ciò che svolgono sono indicati di seguito:

- **START:** Stato in cui avviene la fase di setup del dispositivo, eseguita ogni volta che viene resettato (NON con segnali di reset) per convertire la successiva immagine. Manda STATE=MEMORY e P_STATE=START appena prima di terminare. Tutte le variabili e segnali sono posti a 0, a parte MIN che viene posto a 255. Fino a che *i_start*='1' il processo risulterà bloccato.

- **MEMORY:** Il più importante tra gli stati, poiché indica alla FSM quale sarà il prossimo stato corrente, confrontando il valore di P_STATE. Ad esempio, quando P_STATE = ROWS c'è un ulteriore check per controllare se l'immagine ha dimensione di righe maggiore di zero. Solo in questo caso torna a DONE, poiché negli altri casi il rientro è affidato ad altri stati. Inoltre, vengono salvati e calcolati i valori di *o_address* da aggiornare ad ogni scrittura in memoria, viene inizializzato il *count* che permette di ricominciare a leggere dal primo pixel dopo aver calcolato MAX_SIZE. Si crea un ciclo tramite l'utilizzo degli stati OLD_PIXEL e NEW_PIXEL con MEMORY come intermezzo per poter leggere e scrivere rispettivamente i valori nuovi e vecchi dei pixel all'interno della memoria.

- **COLUMN:** Viene estratto il numero di colonne dell'immagine. Viene posto *o_en*='1' e mantenuto, l'*o_address* viene modificato per la prossima lettura. P_STATE è posto allo stato attuale.

- **ROWS:** Viene estratto il numero di righe dell'immagine. l'*o_address* viene modificato per la prossima lettura. P_STATE è posto allo stato attuale.

- **SEARCH_MAX_MIN:** Cicla fino al raggiungimento di MAX_SIZE+2 (il 2 serve a considerare anche i valori di memoria 0 e 1 che sono le colonne e le righe, che vanno considerate nel totale degli indirizzi) confrontando i valori di MIN e MAX con il pixel attuale, cambiando le variabili ogni volta che il valore è minore di MIN e maggiore di MAX. Vengono utilizzati *count* e *prev_count* per poter evitare dei latch che potrebbero portare a problemi legati al timing durante il processo. *count* incrementa il valore dell'*o_address* da leggere ad ogni ciclo. Ad ogni step del ciclo viene coinvolto lo stato MEMORY per caricare i valori. Una volta ultimato, si esegue else in cui viene calcolato direttamente il DELTA, passando a MEMORY con P_STATE=DELTA_VALUE.

- **DELTA_VALUE:** Una volta ottenuto il DELTA viene fatto un controllo nel quale si richiedono i valori di MAX e MIN: se questi non sono stati modificati (è impossibile che MIN rimanga maggiore di MAX, nel caso ci fosse anche solo un valore sarebbero entrambi uguali e il DELTA risulterebbe 1) vuol dire che in memoria non è presente alcun pixel oltre le dimensioni dell'immagine, portando a uno shift nullo. Shift viene assegnato seguendo la formula *8 - FLOOR(LOG2(DELTA_VALUE +1))* comparando i bit di DELTA in ordine decrescente per simulare una lookup table per confrontare il contrasto tra i grigi e i valori più e meno scuro. Al termine *count* viene riportato a 2, per permettere di leggere l'immagine dal primo pixel. Successivamente P_STATE=OLD_PIXEL facendo iniziare il ciclo per lo shift e la scrittura dei nuovi pixel.

- **OLD_PIXEL:** Stato necessario per poter caricare l'*o_address* corretto e aggiornare efficacemente i valori scritti in memoria. *o_en='1', o_we='0'* garantisce di iniziare NEW_PIXEL in fase di lettura. Successivamente si passa a MEMORY.

- **NEW_PIXEL:** Cicla fino a MAX_SIZE+2 e pone *o_en='1', o_we='1'* per inizializzare la fase di scrittura del valore in memoria, tale valore è calcolato tramite uno *shift_left* e un ampliamento dei valori massimi assumibili dalle variabili a causa della natura dello scostamento, che provocherebbe errori di calcolo se mantenuto uguale. Nel caso in cui il valore calcolato sia maggiore di 255 però, interviene un if posto di seguito che rende massimo il valore attuale, in accordo con la specifica presa in considerazione. Altrimenti viene aggiornato *o_data* in modo da poter scrivere il pixel in posizione corrente di *o_address* nello stato successivo. Se il contatore ha concluso il ciclo, passa allo stato DONE.

- **DONE:** In questo stato vengono resettati *o_en='0'* e *o_we='0'* e posto *o_done='1'* e aspetta *i_start='1'.* Quando questo avviene, viene portato a 0 *o_done* e lo stato è riportato a START.

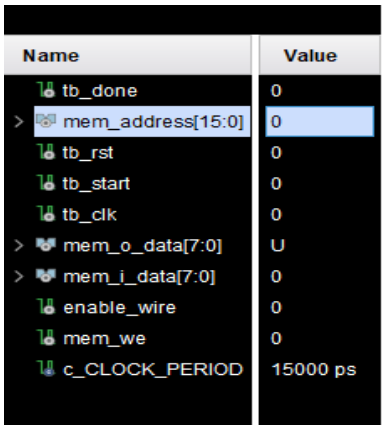
Simulazione e sintesi

I test indicati di seguito sono eseguiti per dimostrare il funzionamento del componente sia in ***Behavioral*** che in ***Post Syntesis***. Tali test rappresentano casi generali e Edge Cases a cui il componente deve far fronte. I test sono stati eseguiti con un clock pari a 15 *ns.*

3.1 Test fornito

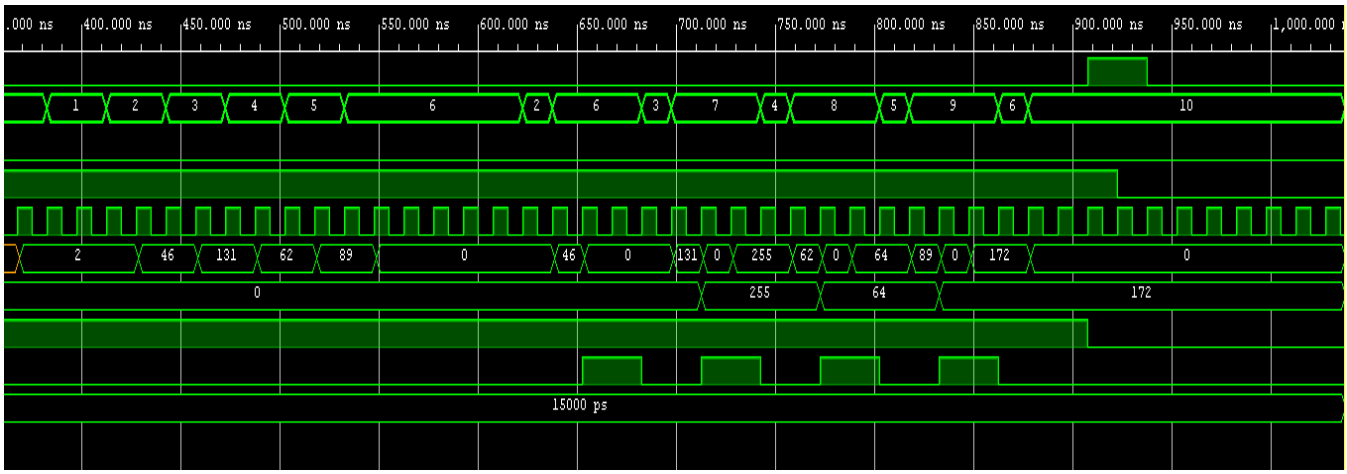
Elaborazione di un'immagine 2x2.

INDIRIZZO MEMORIA	VALORE
0	2
1	2
2	46
3	131
4	62
5	89
6	0
7	255
8	64
9	172



Di lato la legenda dei segnali rappresentati: tali segnali sono da interpretare come binari, diversamente da quanto rappresentato, poiché in memoria sono sequenze di bit.

E i segnali d'onda:

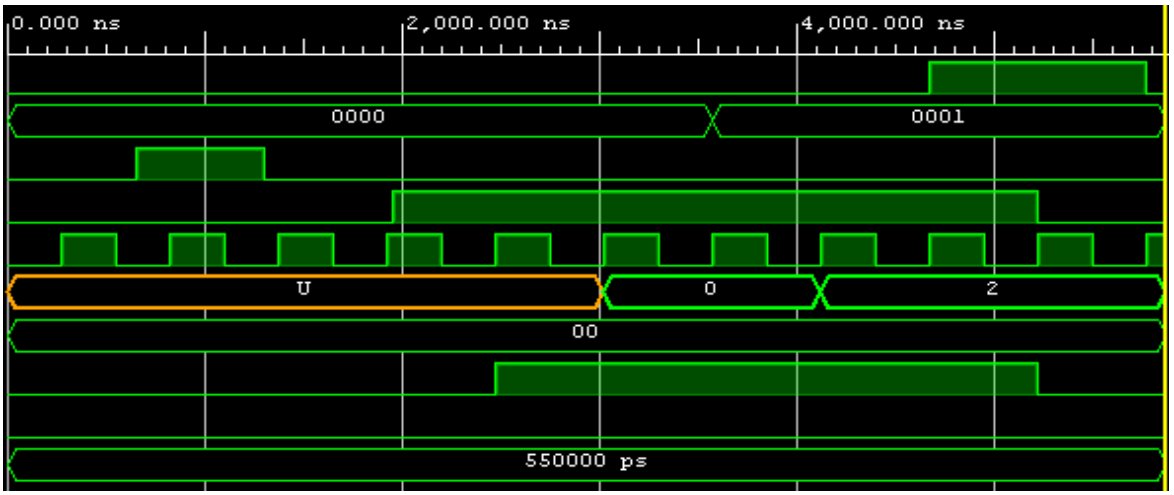


Dato che nel test si ha uno start solo dopo 350 *ns*.

3.2 Test dimensione nulla

Elaborazione di un'immagine 0 x 0, 0 x N o M x 0 dove N, M sono interi positivi minori di 256.

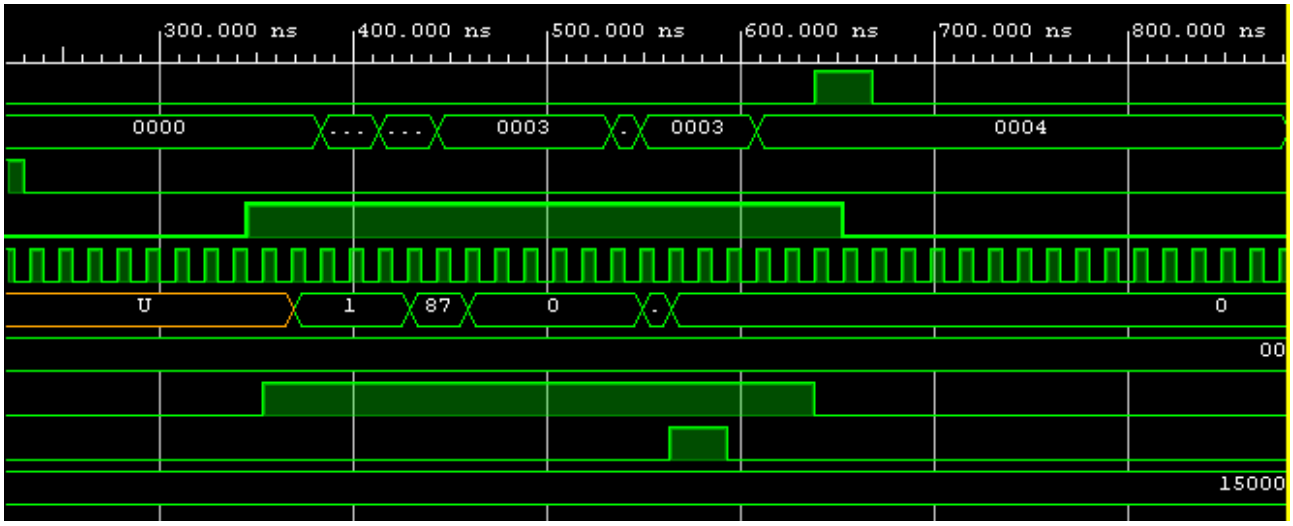
INDIRIZZO MEMORIA	VALORE
0	0
1	2
2	42



3.3 Test singolo pixel

Valuta un'immagine di un singolo pixel e viene trascritto 0 in memoria poiché *shift_level* (già sottratto a 8) è 7.

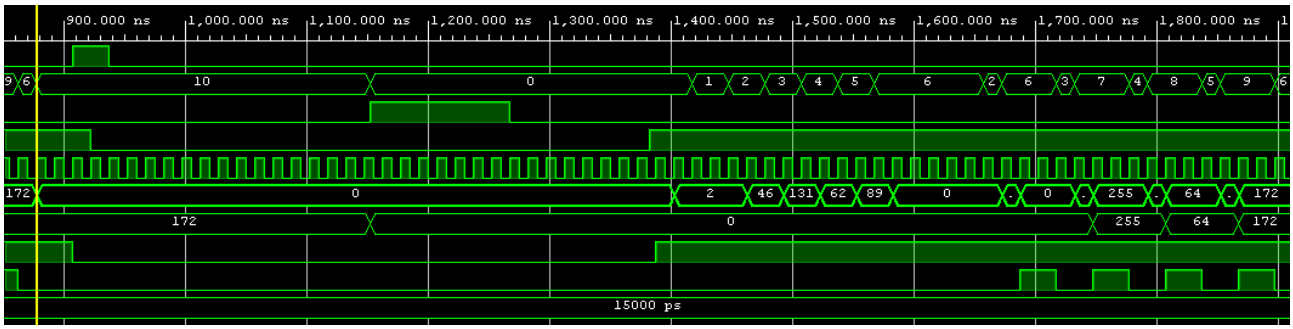
INDIRIZZO MEMORIA	VALORE
0	1
1	1
2	87
3	0



3.4 Test equalizzazioni consecutive

Elaborazione di un'immagine 2x2 ripetuta nella stessa simulazione, testando il reset e il setup corretto dei segnali, come si può notare dall'immagine dei segnali d'onda sottostante 172 è l'*o_address* iniziale e finale, dato che si hanno due esecuzioni e il segnale di reset prima della seconda esecuzione.

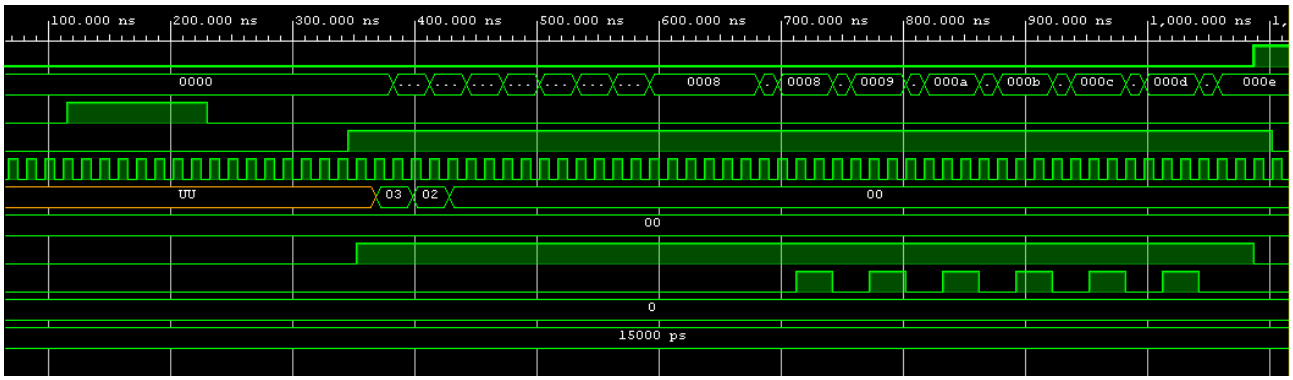
INDIRIZZO MEMORIA	VALORE
0	2
1	2
2	46
3	131
4	62
5	89
6	0
7	255
8	64
9	172



3.5 Test tutti 0

Valuta un'immagine di soli zeri, trascrive solo zeri in memoria poiché *shift_level* (già sottratto a 8) è 7.

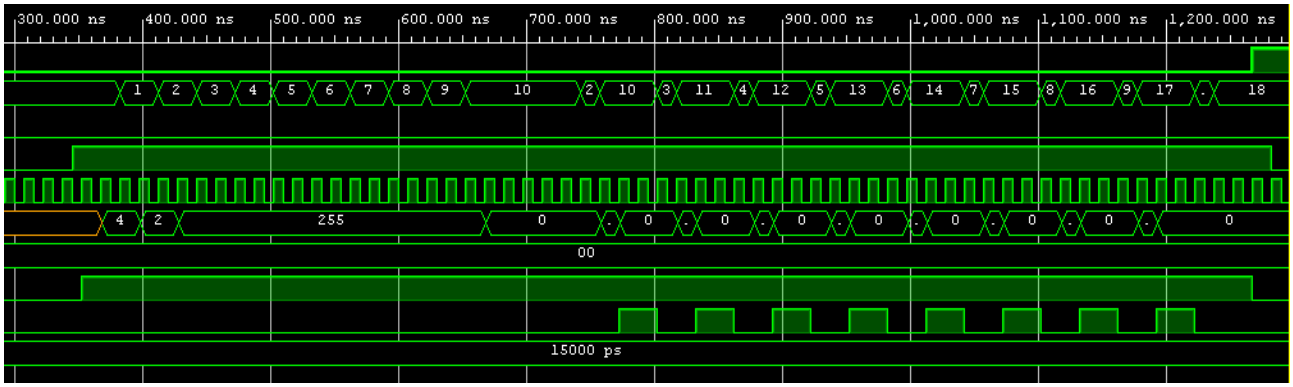
INDIRIZZO MEMORIA	VALORE
0	Righe (3)
1	Colonne (2)
2	0
…	0
N + 1	0
N + 2	0
…	0
N + N + 1	0



3.6 Test tutti 255

Valuta un'immagine di soli numeri 255, trascrive solo zeri in memoria poiché *shift_level* (già sottratto a 8) è 7. Nel caso in cui ci sia solo un valore per ogni pixel, il delta risulta 1; quindi, il numero scritto in memoria sarà sempre 0.

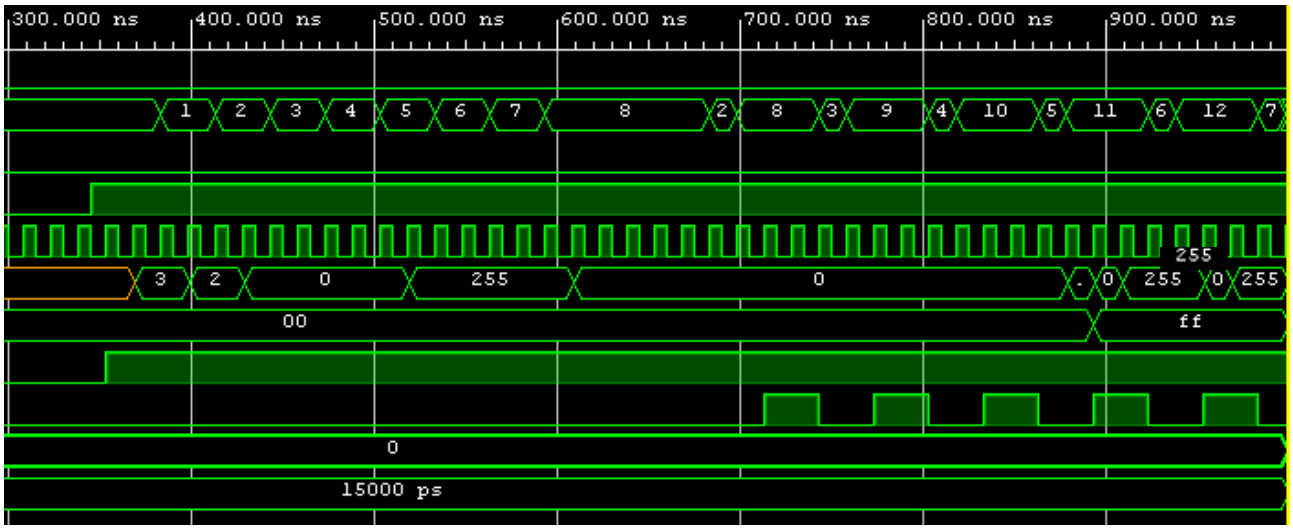
INDIRIZZO MEMORIA	VALORE
0	Righe (4)
1	Colonne (2)
2	255
…	255
N + 1	255
N + 2	0
…	0
N + N + 1	0



3.7 Test full range

Ho due valori specifici, quali 0 e 255, in memoria ho i valori inalterati a causa dello *shift_level=0*.

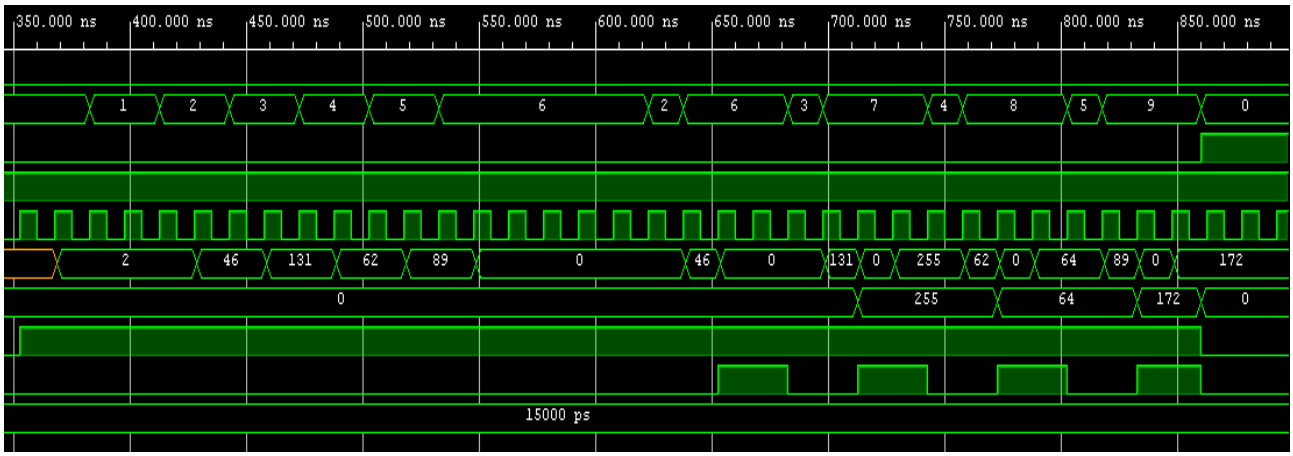
INDIRIZZO MEMORIA	VALORE
0	Righe (3)
1	Colonne (2)
2	0
…	…
N +1	255
N + 2	0
…	…
N + N + 1	255



3.8 Test reset asincrono

Vi è un segnale di reset durante l'elaborazione, ma non intacca l'elaborazione corrente a causa del segnale di *o_done* che non viene alterato.

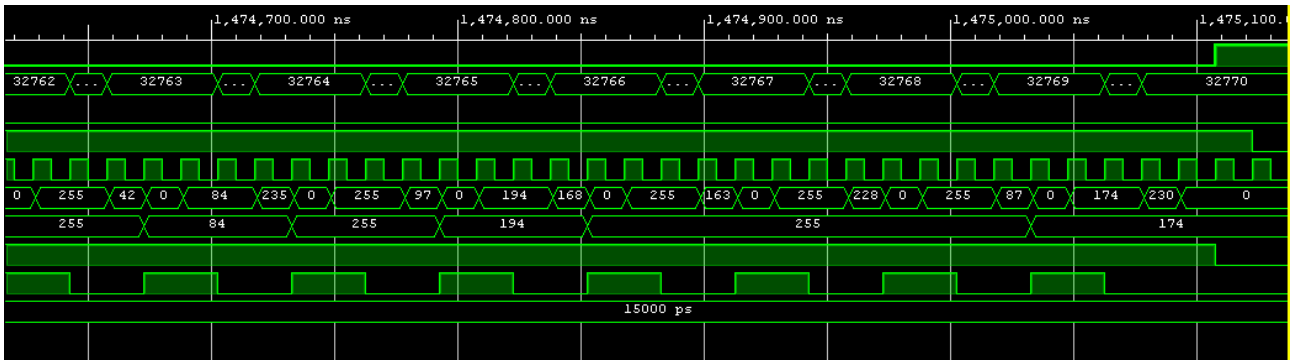
INDIRIZZO MEMORIA	VALORE
0	2
1	2
2	46
3	131
4	62
5	89
6	0
7	255
8	64
9	172



3.8 Test 128x128

Test sul massimo numero di pixel dell'immagine ammissibile.

INDIRIZZO MEMORIA	VALORE
0	2
1	2
2	..
…	…
16385	89
16386	…
…	…
32770	174



3.9 Sintesi

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF
synth_1	constrs_1	synth_design Complete!								214	108
impl_1	constrs_1	route_design Complete!	NA	NA	NA	NA	NA	6.218	0	214	108

Utilizzo del componente di 214 LUT 108 FF in totale, contando sul fatto che l'area disponibile è maggiore rispetto a questo quantitativo, che comunque permette una computazione rapida dell'immagine. Di seguito l'utilizzo in dettaglio:

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	214	0	134600	0.16
LUT as Logic	214	0	134600	0.16
LUT as Memory	0	0	46200	0.00
Slice Registers	108	0	269200	0.04
Register as Flip Flop	108	0	269200	0.04
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

Conclusione

4.1 Conclusione

Il componente sintetizzato è funzionante secondo la specifica richiesta, la decisione di utilizzare un singolo modulo è legata alla possibilità di gestire gli stati e i segnali internamente con maggiore chiarezza e visione d'insieme, minimizzando la possibilità di comportamenti anomali dei segnali (il componente così risulta fortemente sequenziale, posta la condizione che le immagini da codificare abbiano dimensioni relativamente piccole). Lo stato MEMORY è risultato fondamentale per leggere e scrivere in memoria, dando la possibilità a tutti i segnali di potersi allineare correttamente ai calcoli che si considerano corretti, successivamente testati a dovere tramite appositi TestBench.

L'utilizzo di VHDL si è rivelato meticolosamente utile per questo genere di task: permettere di realizzare un singolo componente interfacciato con dei "semplici" segnali lo rende uno strumento adeguato in tutte le mansioni per cui sarebbe "uno spreco" l'utilizzo di calcolatori più complessi, ponendolo come valida soluzione in una gran moltitudine di casi specifici.