PROGETTO FINALE DI RETI LOGICHE

Prof. Salice Fabio – Anno 2020/2021

Fossati Stefano (Codice Persona 10569836 – Matricola 910769)

Guggiari Sofia (Codice Persona 10661333 – Matricola 910391)

INDICE

1. INTRODUZIONE

Lo scopo del progetto è la realizzazione di un componente hardware in VHDL. Esso riceve in ingresso un'immagine in scala di grigi su 256 livelli e, dopo aver applicato l'algoritmo di equalizzazione dell'istogramma dell'immagine su ogni pixel, scrive in output l'immagine equalizzata. Il metodo di equalizzazione dell'istogramma incrementa il contrasto di un'immagine, riequilibrando le intensità in modo uniforme.

L'algoritmo di equalizzazione deve trasformare ogni pixel nel modo 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.

Esempio di immagine 9x4 pixel prima e dopo l'equalizzazione:

0	55	2	96	119	54	237	17	41
83	84	95	158	90	41	49	0	152
33	60	131	68	229	178	181	155	251
57	90	147	29	90	203	31	186	66

0	110	4	192	238	108	255	34	82
166	168	190	255	180	82	98	0	255
66	120	255	136	255	255	255	255	255
114	180	255	58	180	255	62	255	132

I dati, ciascuno di 8 bit, sono memorizzati in una memoria RAM con indirizzamento al byte secondo la seguente struttura:

- L'indirizzo 0 contiene il numero di colonne `N_COL`
- L'indirizzo 1 contiene il numero di righe `N_ROW`
- Dall'indirizzo 2 all'indirizzo `1+(N_COL*N_ROW)` sono memorizzati i pixel dell'immagine
- Dall'indirizzo `2+(N_COL*N_ROW)` sono memorizzati i pixel dell'immagine equalizzata

Ogni immagine può essere al massimo grande 128 x 128 pixel.

2. ARCHITETTURA

L'interfaccia del componente da realizzare è stata presentata nelle specifiche nel seguente modo:

```
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;
```

Dove:

- `i_clk` è il segnale di clock in ingresso,
- `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 vettore che arriva dalla memoria in seguito ad una richiesta di lettura,
- `o_address` è il 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 poter comunicare,
- `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 vettore di uscita dal componente verso la memoria.

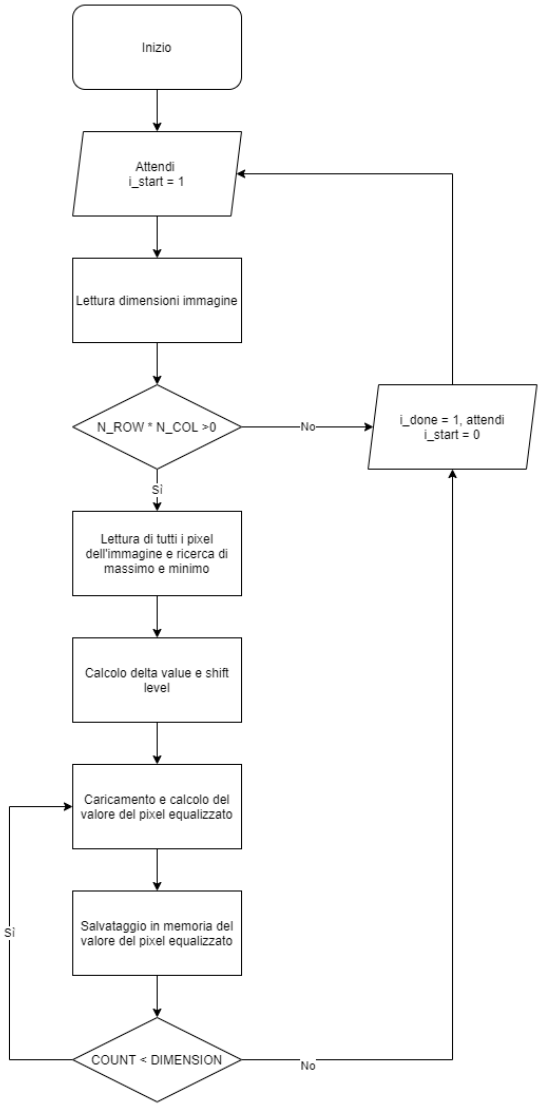
Questo componente si interfaccia con una memoria RAM su cui sono salvate le immagini che deve elaborare e su cui deve scrivere l'immagine alla fine del processo di equalizzazione.

2.1 ALGORITMO

Prima di cominciare a scrivere il codice VHDL si è pensato all'algoritmo da implementare. Questo si comporta nel seguente modo:

- Quando il segnale di start è a 1, leggo il numero di colonne e di righe che costituiscono l'immagine, se `N_COL * N_ROW = 0`, l'immagine ha dimensione nulla, porto a uno il segnale di done
- Cerco i valori massimo e minimo dei pixel dell'immagine analizzandoli tutti
- Calcolo delta value e shift level
- Carico il valore del pixel da equalizzare e lo salvo in memoria una volta equalizzato al giusto indirizzo, ripeto l'operazione per tutti i pixel dell'immagine
- Porto a 1 il segnale di done e mi metto in attesa che il segnale di start si annulli e torni a 1 per iniziare ad equalizzare una nuova immagine

Per implementare l'algoritmo abbiamo deciso di utilizzare una macchina a stati finiti.



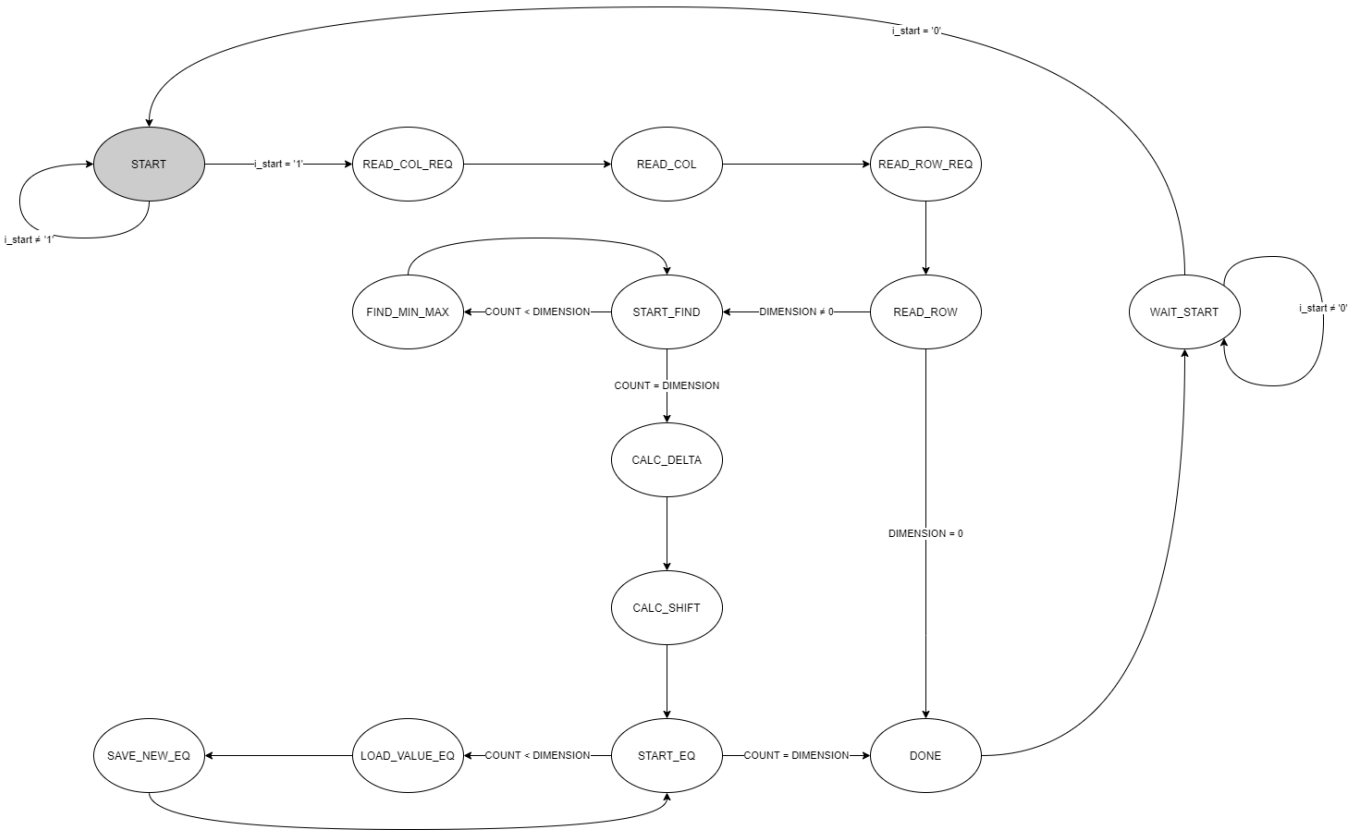
2.2 MACCHINA A STATI

Per rappresentare gli stati della macchina a stati finiti si utilizzano due signal di tipo enumerativo state_type: `current_state` e `next_state`. Il primo rappresenta lo stato attuale in cui si trova la macchina, mentre il secondo rappresenta lo stato al successivo fronte di salita del clock

La macchina a stati che abbiamo creato ha 14 stati:

0	START	MAX = 0 MIN = 255 COUNT = 0 o_done = 0 o_en = 0 o_we = 0
Stato iniziale, viene raggiunto ogni volta che il segnale di reset (`i_reset`) viene posto a uno. In questo stato si attende che il segnale di start (`i_start`) venga posto a uno		
1	READ_COL_REQ	o_en = 1 o_we = 0 o_ address = 0000000000000000
Stato in cui si carica l'indirizzo di memoria delle colonne in uscita		

2	READ_COL	N_COL = TO_INTEGER(unsigned(i_data))
Stato in cui viene letto e salvato nella variabile N_COL il valore che indica il numero di colonne dell'immagine		
3	READ_ROW_REQ	o_en = 1 o_we = 0 o_address = 0000000000000001
Stato in cui si carica l'indirizzo di memoria delle righe in uscita		
4	READ_ROW	N_ROW = TO_INTEGER(unsigned(i_data)) DIMENSION = N_COL * N_ROW Se DIMENSION > 0 -> o_en = 1 e o_we =0
Stato in cui viene letto e salvato nella variabile N_ROW il valore che indica il numero di righe dell'immagine. Se la dimensione dell'immagine è pari a 0, la macchina viene portata allo stato di DONE		
5	START_FIND	Se COUNT < DIMENSION -> TMP_COUNT = COUNT + 1 e o_address = std_logic_vector(TO_UNSIGNED(2 + COUNT, 16))
Da questo stato inizia la ricerca del valore minimo e massimo dell'immagine, una volta che vengono letti tanti pixel quanti ne compongono l'immagine, la macchina viene portata allo stato di CALC_DELTA		
6	FIND_MIN_MAX	VALUE = TO_INTEGER(unsigned(i_data)) COUNT = TMP_COUNT
Stato di confronto, dove vengono salvati i valori massimo e minimo dei pixel che compongono l'immagine tramite confronto		
7	CALC_DELTA	DELTA = MAX – MIN COUNT = 0
Stato in cui viene calcolato il delta value		
8	CALC_SHIFT	SHIFT_LEVEL = nuovo valore
Stato in cui viene calcolato lo shift level tramite dei controlli a soglia		
9	START_EQ	Se COUNT = DIMENSION -> o_we = 0 e o_en = 0 Altrimenti -> TMP_COUNT = COUNT + 2, o_we = 0 e o_address = std_logic_vector(TO_UNSIGNED(2 + COUNT, 16))
Stato della macchina che inizializza l'equalizzazione dell'immagine, una volta che tutti i pixel sono stati equalizzati, la macchina viene portata allo stato di DONE		
10	LOAD_VALUE_EQ	VALUE = TO_INTEGER(unsigned(i_data)) TMP_VALUE = std_logic_vector(shif_level(unsigned(TO_UNSIGNED((VALUE – MIN), 16)), SHIFT_VALUE)) o_we = 1 o_address = std_logic_vector(TO_UNSIGNED(2 + COUNT + DIMENSION, 16))
Stato in cui viene caricato il valore del pixel da equalizzare		
11	SAVE_NEW_VALUE	Se TMP_VALUE >= 255 -> o_data = 11111111 Altrimenti -> o_data = TMP_VALUE(7 downto 0)
Stato in cui salvo il valore del pixel equalizzato in memoria		
12	DONE	o_done = 1
Stato della macchina che porta a uno il segnale di done (o_done)		
13	WAIT_START	
Stato in cui si attende che il segnale di start (i_start) diventi zero per poter tornare allo stato di START e iniziare a leggere una nuova immagine		



Il segnale di reset è asincrono e potrebbe essere portato ad uno in un qualunque istante della computazione, ogni stato dovrebbe avere una transizione che porta nello stato di `START`; in questo grafico non vengono mostrate tutte le transizioni che portano in `START` per migliorare la leggibilità della rappresentazione.

2.3 SCELTE IMPLEMENTATIVE

La prima operazione descritta dall'algoritmo viene svolta dalla macchina negli stati `START`, `READ_COL_REQ`, `READ_COL`, `READ_ROW_REQ` e `READ_ROW`. In questa fase vengono letti dalla memoria i primi due indirizzi, che fanno riferimento rispettivamente al numero di colonne e di righe dell'immagine, e vengono salvati in due variabili di tipo integer (`N_COL` e `N_ROW`). Dopo aver fatto ciò, nello stato `READ_ROW` viene anche controllata l'effettiva grandezza dell'immagine (`DIMENSION = N_COL * N_ROW`), in modo da verificare che non si stia lavorando con un'immagine di dimensioni nulle. Se così fosse, la macchina a stati passa direttamente al penultimo stato (`DONE`), altrimenti viene portata allo stato di `START_FIND`.

La seconda operazione consiste nella ricerca del valore massimo e minimo (salvati in due variabili di tipo intero `MIN` e `MAX`) dei pixel dell'immagine; questa viene svolta negli stati `START_FIND` e `FIND_MIN_MAX`. In particolare, nello stato `FIND_MIN_MAX` si confronta ogni pixel con il massimo e il minimo finora trovati nel seguente modo:

```
if(MIN > VALUE) then
     MIN := VALUE;
end if;
if(MAX < VALUE) then
      MAX := VALUE;
end if;
```

Dove `VALUE` è il valore letto dalla memoria.

Dopo aver trovato i valori MAX e MIN dell'immagine, occorre calcolare il delta value e lo shift level, definiti dalle specifiche come:

```
DELTA_VALUE = MAX_PIXEL_VALUE – MIN_PIXEL_VALUE
SHIFT_LEVEL = (8 – FLOOR(LOG2(DELTA_VALUE + 1)))
```

Questa operazione viene svolta negli stati CALC_DELTA e CALC_SHIFT, e data la forma dello shift level, il suo valore può essere facilmente trovato con un controllo a soglie fatto come segue:

```
if(DELTA = 0) then
      SHIFT_LEVEL := 8;
elsif(DELTA >= 1 AND DELTA < 3) then
      SHIFT_LEVEL := 7;
elsif(DELTA >= 3 AND DELTA < 7) then
      SHIFT_LEVEL := 6;
elsif(DELTA >= 7 AND DELTA < 15) then
      SHIFT_LEVEL := 5;
elsif(DELTA >= 15 AND DELTA < 31) then
      SHIFT_LEVEL := 4;
elsif(DELTA >= 31 AND DELTA < 63) then
      SHIFT_LEVEL := 3;
elsif(DELTA >= 63 AND DELTA < 127) then
      SHIFT_LEVEL := 2;
elsif(DELTA >= 127 AND DELTA < 255) then
      SHIFT_LEVEL := 1;
elsif(DELTA = 255) then
      SHIFT_LEVEL := 0;
end if;
```

La quarta operazione è quella di equalizzazione dell'immagine, e viene svolta negli stati START_EQ, LOAD_VALUE_EQ e SAVE_NEW_EQ. In questa fase si leggono dalla memoria i valori originali di ogni pixel dell'immagine, e grazie allo shift level calcolato nella fase precedente, si elaborano i valori dei pixel equalizzati, che vengono poi salvati in memoria.
Visto che lo shift level è uno shift logico di n posizioni codificato in binario senza segno, ci è sembrato opportuno salvare il valore del pixel originale in una variabile di tipo vettore logico, in modo da poter effettuare l'operazione di shift logico attraverso l'apposita funzione fornita da Vivado (shift_left). In questo modo, inoltre, il valore che si ottiene è un vettore logico, ed è quindi già nella forma che ci serve per poterlo salvare in memoria. Ma, visto che alcuni valori temporanei (TEMP_PIXEL = (CURRENT_PIXEL_VALUE - MIN_PIXEL_VALUE) << SHIFT_LEVEL) potrebbero superare il valore massimo salvabile in un vettore logico da 8 bit, ossia 255, si è deciso di salvare i valori originali dei pixel in un vettore logico di dimensione 16 bit, in modo da coprire il massimo valore che potrebbe assumere un pixel equalizzato. Per questo, nello stato SAVE_NEW_VALUE, la prima operazione svolta è il confronto tra il valore temporaneo calcolato convertito in intero, e 255, se il valore calcolato supera 255, allora in memoria viene salvato il vettore "11111111", altrimenti, il valore temporaneo viene troncato agli ultimi 8 bit (da destra), nel seguente modo:

```
if(TO_INTEGER(unsigned(TMP_VALUE)) >= 255) then
    o_data <= "11111111";
else
    o_data <= TMP_VALUE(7 downto 0);
end if;
```

Dopo aver concluso l'equalizzazione, la macchina viene portata nello stato di DONE, in cui il segnale di done (o_done) viene posto a uno, e viene poi portata nello stato WAIT_START, dove attende che il segnale di start (i_start) venga messo a zero per poter incominciare una nuova equalizzazione.

La nostra soluzione è stata sviluppata con un singolo process che contiene i segnali `i_clock` e `i_reset` nella sensitivity list. Per questo motivo la macchina modifica il suo stato quando cambia il segnale di clock (lungo il suo fronte di salita), o quando viene posto a uno il segnale di reset. Grazie a questo accorgimento, la macchina è in grado di modificare correttamente il suo stato ogni qual volta che il segnale di reset viene azionato, in qualunque stato si trovi la macchina.

2.4 OTTIMIZZAZIONI

Il progetto iniziale era costituito da 17 stati. Erano presenti due stati aggiuntivi tra `START_FIND` e `FIND_MIN_MAX`, che servivano per caricare l'indirizzo di memoria da leggere e per caricare il valore contenuto nell'indirizzo di memoria. Questi sono stati eliminati utilizzando la variabile `COUNT`, che consente di calcolare l'indirizzo di memoria già nello stato `START_FIND`, e la variabile `VALUE`, che consente di trovare il valore contenuto dell'indirizzo di memoria nello stato `FIND_MIN_MAX`. Un altro stato che abbiamo eliminato si trovava tra `START_EQ` e `LOAD_VALUE_EQ` e serviva a calcolare l'indirizzo di memoria in cui caricare l'immagine equalizzata, ma grazie alla costante `COUNT` è stato possibile spostare questo calcolo direttamente nello stato `START_EQ`.

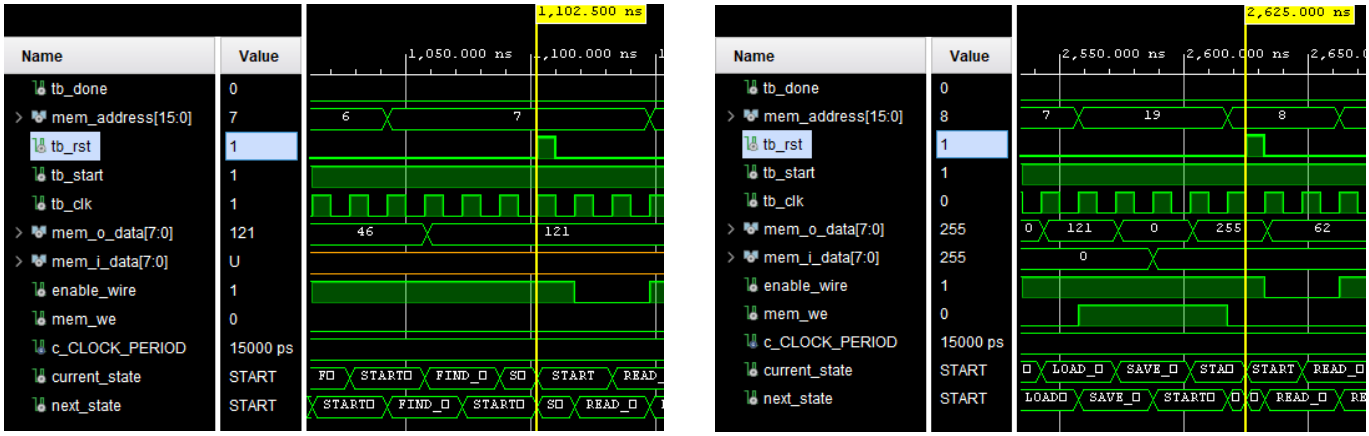
La FSM realizzata non rappresenta la soluzione a numero di stati minimo in quanto è possibile ridurre ulteriormente il diagramma per ottenere una soluzione più compatta (ad esempio il calcolo del `DELTA_VALUE` e dello `SHIFT_LEVEL` potevano essere messi in un unico stato). Ma abbiamo ritenuto che fosse la scelta migliore in termini di chiarezza del codice, in quanto facilmente modificabile per eventuali ottimizzazioni future.

3. SIMULAZIONI

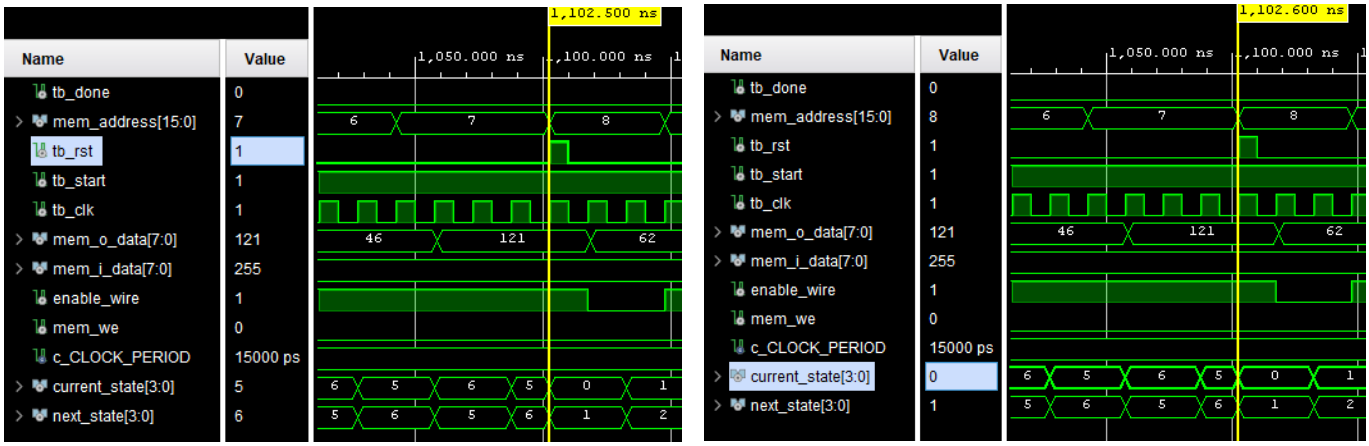
Sono stati effettuati molti test, qui vengono riportati i casi limite testati:

Immagine 128x128	Caso limite di immagine con dimensione massima
Immagine 1x1	Caso limite di immagine con dimensione minima
Immagine 0xn	Caso limite di immagine con dimensione nulla, effettuato per controllare che la macchina vada correttamente dallo stato di `READ_ROW` a quello di `DONE`, evitando di elaborare l'immagine
Immagine con tutti i pixel a 255	Test effettuato per verificare il corretto funzionamento del calcolo del shift level, dato che con questa casistica `MAX` = `MIN` = 255 quindi il delta value risulta pari a zero, e lo shift level pari a 8, quindi l'immagine risultante ha tutti i pixel a 0
Immagine con tutti i pixel a 1	Anche questo test, come il precedente, è stato effettuato per verificare il corretto funzionamento del calcolo dello shift level, e anche in questo caso l'immagine risultante ha tutti i pixel a 0
Reset sincroni e asincroni	Per verificare il corretto funzionamento della macchina a stati a fronte del segnale di reset, sono stati effettuare numerosi test con reset sincroni e asincroni, in particolare sono stati fatti diversi test con reset casuali, anche più volte sulla stessa immagine. Oltre a questi è stato verificato che il componente rilevasse reset di durata molto breve (clk/4).
Immagini consecutive	Il componente è stato stressato grazie a test autoprodotti e creati mediante il generatore per verificare il suo corretto funzionamento quando riceve in ingresso immagini consecutive

Sono stati effettuati diversi test, con periodi di clock differenti, in particolare da 1 ns, 15 ns e 100ns, tutti hanno terminato la simulazione con esito positivo, dimostrando il corretto funzionamento del componente progettato.



Simulazione di tipo behavioural con reset sincrono (a sinistra) e con reset asincrono (a destra)



Simulazione di tipo post-sintesi con reset sincrono, si vuole evidenziare il tempo di aggiornamento di `current_state`, ossia 0.1 ns.

Dall'analisi dell'andamento dei grafici dei test è possibile notare che, sia nella simulazione behavioural, sia in quella post sintesi, ogni stato della macchina dura due cicli di clock. In particolare, nella simulazione behavioural il segnale `current_state` viene aggiornato un ciclo di clock dopo rispetto al cambiamento dello stato `next_state`, mentre nella simulazione di tipo post-sintesi i segnali `current_state` e `next_state` vengono aggiornati nello stesso istante.

4. SINTESI

Tramite “Report di Timing” si può analizzare la velocità della computazione del componente rispetto ad un constrain di clock fornito (impostato a 100 ns, come da specifiche). Sono stati trovati i seguenti valori:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	90,000 ns	Worst Hold Slack (WHS):	0,155 ns	Worst Pulse Width Slack (WPWS):	49,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	387	Total Number of Endpoints:	387	Total Number of Endpoints:	176

All user specified timing constraints are met.

Di conseguenza, il periodo minimo di clock con cui può funzionare il componente è di 100ns – 90 ns = 10 ns, che corrisponde a una frequenza di 1/10 ns = 100MHz.

Utilizzando invece il tool “Report utilization” è stato possibile analizzare l’implementazione fisica sulla FGPA del componente hardware progettato, sono stati ottenuti i seguenti risultati: 167 Look Up Table (LUT), corrispondenti allo 0.12% dell’area totale e 175 Flip-Flop (FF), corrispondenti allo 0.07% dell’area totale.

5. CONCLUSIONE

Il componente è stato realizzato con Vivado (versione 2020.2), lavorando sulla FPGA Artix-7 xc7a200tfbg484-1 (come consigliato da specifiche), ha dimostrato di superare correttamente, in tutti i casi di test, la simulazione behavioural e post-syhtesis, in accordo con quanto richiesto dalle specifiche, e funziona in pre e post-sintesi senza la presenza di warning o latch.

Si ritiene quindi di aver sviluppato un componente hardware in grado di risolvere in modo adeguato il problema.