POLITECNICO
MILANO 1863

Prova Finale:
Progetto di Reti Logiche

Lorenzo Guerrieri

Codice persona – 10571386 Matricola - 886670

Questa documentazione ha lo scopo presentare e spiegare le scelte effettuate durante la realizzazione del Progetto di Reti Logiche, una delle Prove Finali del corso di Laurea triennale in Ingegneria Informatica presso il Politecnico di Milano.

Sommario

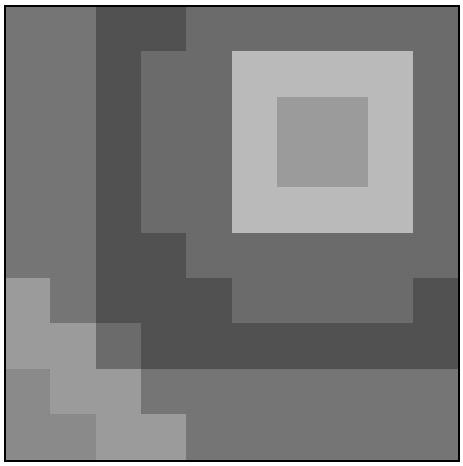
1. Introduzione 2
1.1. Equalizzazione dell'istogramma 2
1.2. Specifiche della consegna 2
1.3. Interfaccia del componente 3
1.4. Protocollo di comunicazione 4
1.5. Descrizione della Memoria 4
2. Architettura 5
2.1. Macchina a stati finiti 5
2.2. Acquisizione dei dati 6
2.3. Elaborazione 6
2.4. Reset e Idle state 7
2.5. Scelte progettuali 7
3. Risultati 8
3.1. Osservazioni su tempo di esecuzione e memoria 8
3.2. Report di sintesi 8
4. Test Bench 9
4.1. Test semplici 9
4.2. Test specifici 10
5. Conclusione 11

1. Introduzione

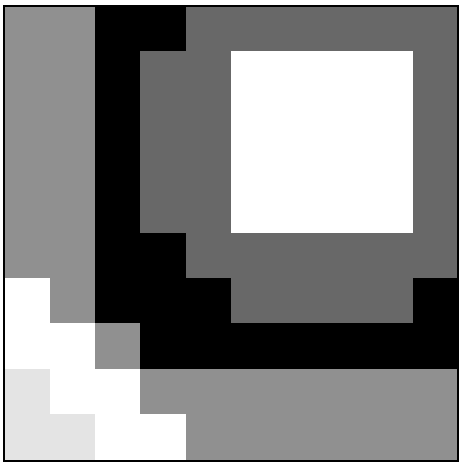
1.1. Equalizzazione dell'istogramma

Lo scopo del progetto è quello di elaborare un modulo hardware, descritto in VHDL, che implementi una versione semplificata dell'algoritmo di equalizzazione dell'istogramma.

Tale algoritmo è utilizzato per aumentare il livello di contrasto tra pixel di una stessa immagine in modo proporzionale, spalmando i valori dei pixel sull'intera scala a disposizione.



Originale



Equalizzata

117	117	81	81	107	107	107	107	107	107
117	117	81	107	107	186	186	186	186	107
117	117	81	107	107	186	155	155	186	107
117	117	81	107	107	186	155	155	186	107
117	117	81	107	107	186	186	186	186	107
117	117	81	81	107	107	107	107	107	107
155	117	81	81	81	107	107	107	107	81
155	155	117	81	81	81	81	81	81	81
138	155	155	117	117	117	117	117	117	117
138	138	155	155	117	117	117	117	117	117

144	144	0	0	104	104	104	104	104	104
144	144	0	104	104	255	255	255	255	104
144	144	0	104	104	255	255	255	255	104
144	144	0	104	104	255	255	255	255	104
144	144	0	104	104	255	255	255	255	104
144	144	0	0	104	104	104	104	104	104
255	144	0	0	0	104	104	104	104	0
255	255	144	0	0	0	0	0	0	0
228	255	255	144	144	144	144	144	144	144
228	228	255	255	144	144	144	144	144	144

1.2. Specifiche della consegna

Si richiede che il modulo realizzato sia in grado di operare con pixel in scala di grigi ciascuno rappresentato da singolo valore contenuto tra 0 e 255.

Le immagini avranno dimensioni uguali o inferiori ai 128 pixel sia in altezza che in larghezza.

La versione ridotta dell'algoritmo di cui è richiesta l'implementazione è definita da quattro operazioni:

```
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. L'operatore "<<" corrisponde allo shift logico verso sinistra.

1.3. Interfaccia del componente

Il componente implementa la seguente interfaccia:

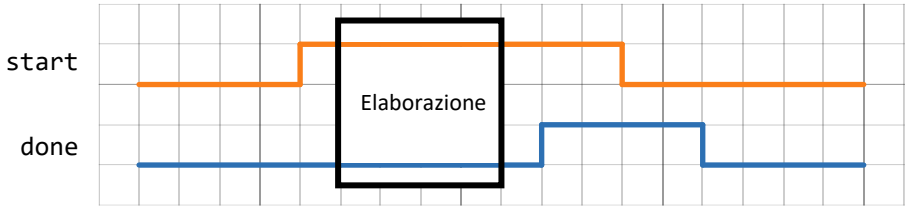
```
entity project_reti_logiche is
    port (
        i_clk     : in  std_logic;
        i_rst     : in  std_logic;
        i_start   : in  std_logic;
        i_data    : in  std_logic_vector(7 downto 0);
        o_address : out std_logic_vector(15 downto 0);
        o_done    : out std_logic;
        o_en      : out std_logic;
        o_we      : out std_logic;
        o_data    : out std_logic_vector (7 downto 0);
    );
end project_reti_logiche;
```

In particolare:

- 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 che indica al modulo di iniziare l'elaborazione;
- 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 indica alla memoria l'indirizzo con cui si vuole interagire (read o write);
- 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 poterci scrivere. Per leggere da memoria esso deve essere 0;
- o_data è il segnale (vettore) di uscita dal componente verso la memoria.

1.4. Protocollo di comunicazione

Si richiede che il modulo rispetti il seguente protocollo di comunicazione, dove `start` è un segnale fornito dall'utilizzatore esterno al modulo e done è un segnale fornito dal modulo verso l'esterno.



- L'elaborazione inizia quando il modulo legge il segnale `start` a `'1'`;
- Quando l'elaborazione termina, il modulo pone a `'1'` il segnale done;
- In risposta, `start` viene posto a `'0'`;
- Quando il modulo legge `start = '0'` viene abbassato anche done.

1.5. Descrizione della Memoria

Per ricavare i dati necessari all'elaborazione il modulo interagisce con una memoria con indirizzamento al byte indicizzata a partire da 0.

I dati al suo interno devo essere inseriti come segue:

- Posizione 0: byte contenente il numero di colonne dell'immagine da elaborare;
- Posizione 1: byte contenente il numero di righe dell'immagine da elaborare;
- Posizione 2: byte contenente il primo pixel dell'immagine;
- Posizione `(col*rig)+1`: byte contenente l'ultimo pixel dell'immagine;

Ogni pixel che compone il risultato dell'elaborazione viene salvato nella stessa memoria a partire dalla posizione `(col*rig)+2` con lo stesso ordine con cui sono salvati i pixel dell'immagine originale.

Esempio: Immagine 2x2.

20	170
225	0

Immagine originale

8	255
255	0

Immagine equalizzata

Pos	Byte
0	2
1	2
2	20
3	170
4	225
5	0
6	8
7	255
8	255
9	0

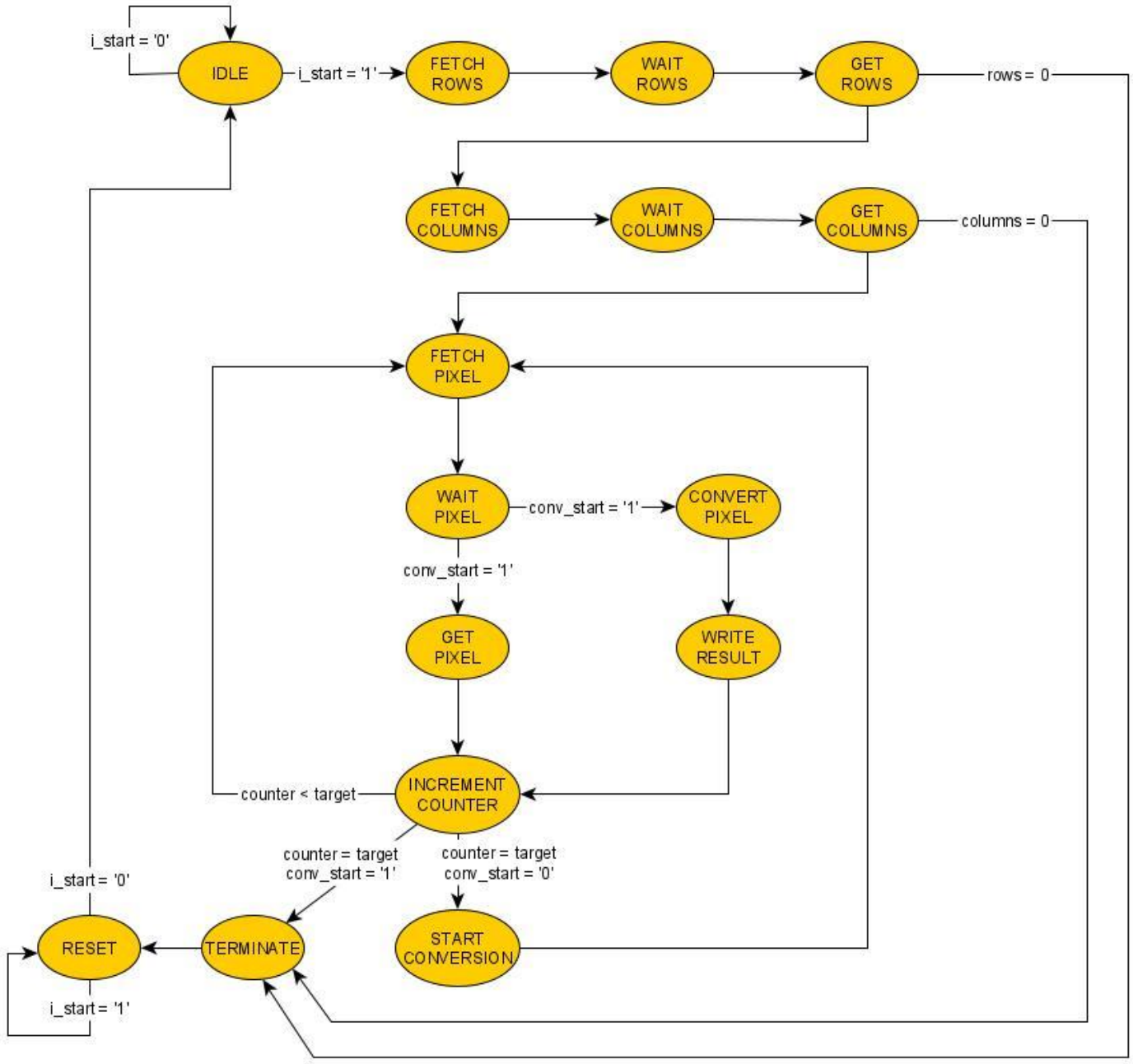
Dati presenti in memoria necessari per l'elaborazione

Dati inseriti dal modulo rappresentanti il risultato dell'elaborazione

2. Architettura

2.1. Macchina a stati finiti

Il modulo implementa l'algoritmo mediante una macchina a stati finiti. Lo schema completo è riportato di seguito, nei paragrafi seguenti si presentano le tre sezioni principali di cui si compone la macchina.



2.2. Acquisizione dei dati

La prima fase della macchina a stati finiti è quella in cui il componente legge i dati necessari all'elaborazione. Ciò significa che vengono letti il numero di righe e di colonne e di seguito tutti i pixel che compongono l'immagine originale. In questa prima fase non vengono effettuate operazioni sui pixel, ma vengono salvati il massimo e il minimo valore tra tutti i pixel.

Gli stati FETCH_X, WAIT_X E GET_X svolgono la stessa operazione di base su diversi oggetti X:

- FETCH_X: richiede alla memoria il valore di X. Nel caso di ROWS e COLUMNS l'indirizzo da cui prendere il dato è fissato (rispettivamente 0 e 1), mentre nel caso di PIXEL l'indirizzo viene calcolato in questo stato.
- WAIT_X: viene fatto passare il tempo necessario perché la memoria renda disponibile il dato richiesto nello stato precedente.
- GET_X: legge il dato fornito dalla memoria. Nel caso di PIXEL esegue il confronto tra il valore del pixel e il valore massimo e minimo letti fino a quel punto, eventualmente aggiornandoli. Nel caso di ROWS e COLUMNS si valuta se il dato appena letto sia zero. In tal caso non è necessaria alcuna conversione (un'immagine con una dimensione a zero non esiste), quindi si passa allo stato TERMINATE.

Dopo aver acquisito il valore di un pixel, lo stato INCREMENT_COUNTER confronta il valore del contatore che tiene traccia di quanti dati sono stati letti con il valore obiettivo (determinato dal valore di righe e colonne). Se l'obiettivo non è stato raggiunto il contatore viene incrementato e si prosegue con la lettura di un altro pixel (passaggio allo stato GET_PIXEL). Se invece sono stati letti tutti i pixel, si prosegue con lo stato START_CONVERSION.

Nota: I percorsi che portano da INCREMENT_COUNTER a TERMINATE e da WAIT_PIXEL a CONVERT_PIXEL vengono attivati quando la macchina passa per lo stato START_CONVERSION.

2.3. Elaborazione

La fase di elaborazione dei dati inizia con lo stato START_CONVERSION. In questo stato il contatore che tiene traccia dei pixel letti viene resettato e viene calcolato il valore SHIFT_LEVEL (si veda il paragrafo 1.2). In questo stato viene posto a 1 il segnale CONVERSION_STARTED che comunica agli altri stati di operare la conversione sui prossimi pixel letti.

Gli stati FETCH_PIXEL e WAIT_PIXEL svolgono a stessa funzione descritta nel paragrafo precedente, ma questa volta si prosegue con CONVERT_PIXEL. In questo stato si esegue il calcolo del valore TEMP_PIXEL (si veda il paragrafo 1.2).

Lo stato WRITE_RESULT decide quale valore rappresenta la conversione del pixel e lo prepara per essere scritto in memoria. Si noti che se fosse necessario eseguire altre operazioni di lettura o scrittura subito dopo questo stato, sarebbe necessaria la presenza di uno stato di attesa per consentire alla memoria di terminare la scrittura del dato. Poiché nel caso specifico si procede con lo stato INCREMENT_COUNTER che non interagisce con la memoria, si può essere sicuri che alla prossima richiesta la memoria avrà già terminato l'operazione di scrittura.

Nello stato INCREMENT_COUNTER vengono svolte le stesse operazioni descritte al paragrafo precedente. A questo punto, però, è attivo il cammino che porta allo stato TERMINATE. Questa transizione si verifica quando il contatore segnala che sono stati letti (e quindi trasformati) tutti i pixel.

2.4. Reset e Idle state

L'ultima sezione della macchina a stati finiti comincia con lo stato TERMINATE. Come previsto dal protocollo di comunicazione (si veda il paragrafo 1.2), in questo stato viene posto a '1' il segnale in uscita o_done.

Nello stato di RESET si attende che il segnale i_start venga posto (esternamente) a '0', dopodiché vengono riportati ai valori di default tutti i segnali (per esempio il contatore dei pixel e riportato a 0). Si noti che questo stato non ha a che fare con il reset della macchina che viene eseguito quando si riceve l'apposito segnale. Il reset causato esternamente è gestito in modo indipendente dalla macchina a stati finiti (si veda il prossimo paragrafo).

Dopo il reset (che sia causato dalla fine dell'elaborazione o dalla ricezione del segnale) la macchina a stati finiti finisce nello stato di IDLE. In questo stato non viene eseguito alcun tipo di operazione. Il modulo è pronto per una nuova elaborazione e attende che il segnale I_START sia posto a '1' per iniziare la conversione.

2.5. Scelte progettuali

Tra le scelte progettuali effettuate ci si vuole soffermare su due: la gestione del segnale di reset esterno e l'implementazione del contatore dei pixel.

Come già accennato, il segnale di reset viene gestito in modo indipendente dalla macchina a stati finiti. Ciò viene realizzato in VHDL come segue:

```
process (i_clk, i_rst)
begin
    if(i_rst = '1') then
        --Reset output signals
        --Reset internal signals

    elsif (rising_edge(i_clk)) then
        --Finite State Machine
    end if;
end process;
```

Il controllo dello stato di i_rst prima di considerare la macchina a stati permette gestire il segnale di reset a prescindere dallo stato in cui il modulo si trova nel momento in cui viene ricevuto. Inoltre, avere il process sensibile sia al clock che al segnale di reset permette di gestire reset asincroni.

Per quanto riguarda il contatore del numero di pixel letti (e poi equalizzati) si è deciso di usare due diversi contatori per righe e colonne. Questo permette di avere maggiore flessibilità dal punto di vista del codice VHDL in caso di future modifiche alla specifica. Inoltre, si è optato per rappresentare i due contatori con soli 7 bit nonostante 128, il numero massimo per righe e colonne, venga rappresentato con 8 bit. Questo viene realizzato contando il valore 0 come primo numero contato dai contatori e fermando il conteggio un numero prima del numero di righe e colonne. In questo modo si risparmiano bit e quindi potenza dispersa durante la conversione del valore di un bit. In minima parte si guadagna anche dal punto di vista delle risorse messe a disposizione dalla FPGA su cui verrà implementato il modulo, sebbene questo guadagno sia marginale.

3. Risultati

3.1. Osservazioni su tempo di esecuzione e memoria

L'implementazione dell'algoritmo è stata realizzata in modo da avere tempo di esecuzione costante a parità di dimensioni delle immagini. Sono state effettuate diverse simulazioni funzionali in post-sintesi per convalidare il risultato.

Si può verificare che i tempi di esecuzione dipendono esclusivamente dalle dimensioni (righe e colonne) dell'immagine, quindi dal numero di pixel complessivi. Di conseguenza un'immagine 1x16 e una 4x4 saranno elaborate nello stesso tempo. Né la disposizione, né i valori dei pixel influenzano il tempo della simulazione.

I tempi della simulazione post sintesi aumentano linearmente con l'aumento del numero di pixel, come ci si aspetta considerando l'implementazione dell'algoritmo (due cicli sui pixel e altre operazioni costanti).

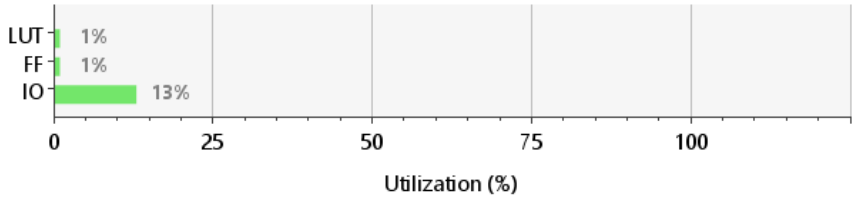
I pixel vanno sempre letti due volte, una per trovare massimo e minimo e una per equalizzare l'immagine. Si è deciso di leggere entrambe le volte i dati direttamente dalla memoria, pagando il prezzo dei tempi delle comunicazioni tra modulo e RAM, per evitare di salvare tutti i pixel, cosa che avrebbe reso necessario avere a disposizione, a prescindere dall'immagine in input, lo spazio per salvare 16kB di dati (dato che aumenterebbe quadraticamente con futuri aumenti del valore massimo di righe e colonne).

3.2. Report di sintesi

Si riporta di seguito l'utilizzo delle diverse risorse (LUT, FF e IO) estratto dalla sintesi del modulo tramite Vivado. Non essendo richieste ai fini del progetto particolari ottimizzazioni, non è stato ricercato un perfezionamento del consumo di tali risorse, quanto un primo bilanciamento tra utilizzo di memoria e tempo utilizzati per l'equalizzazione di ogni immagine (come presentato nel paragrafo precedente).

Nota: le risorse disponibili e le percentuali di utilizzo fanno riferimento alla FPGA xc7a200tfbg484-1.

Resource	Utilization	Available	Utilization %
LUT	283	134600	0.21
FF	98	269200	0.04
IO	38	285	13.33



4. Test Bench

4.1. Test semplici

Il modulo è stato testato tramite il test fornito dal professore e numerosi altri test generati in modo casuale per dimensioni e valori dei pixel. Tutti i test sono stati passati con successo anche quando eseguiti senza reset esterno del modulo. A titolo esemplificativo si analizza qui l'esecuzione del test bench fornito, in cui l'immagine in input e quella equalizzata sono le seguenti.

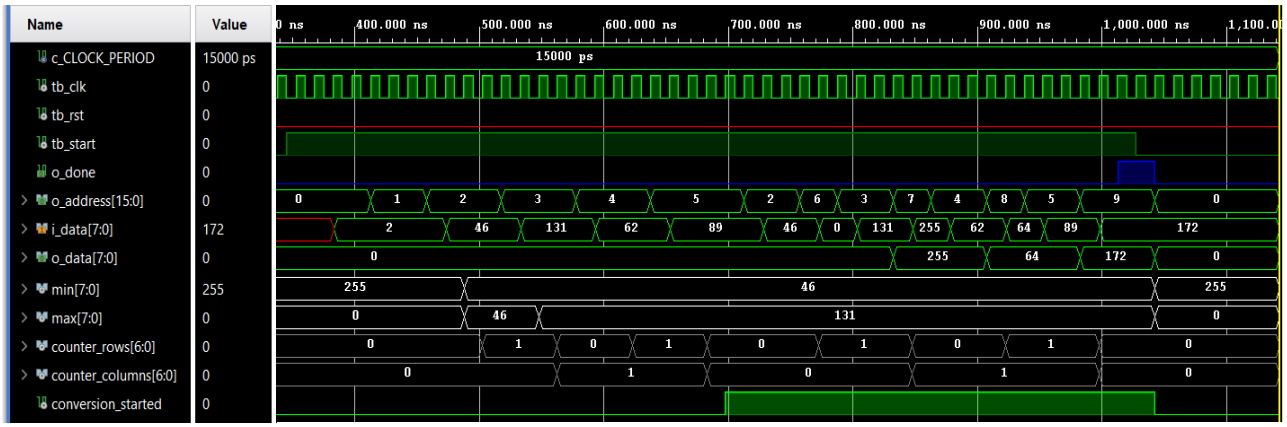
Immagine originale

46	131
62	89

Immagine equalizzata

0	255
64	172

Si riporta di seguito la forma d'onda generata dalla simulazione funzionale post-sintesi del componente (si è omessa la parte inziale con la ricezione del segnale di reset).



Come previsto, vengono letti i valori nelle posizioni `0` e `1`, corrispondenti a righe e colonne dell'immagine. Durante queste letture `i_data` resta a 2, essendo il numero di righe uguale al numero di colonne. Dopodiché si procede con la prima lettura dei pixel (valori in memoria da 2 a 5). `i_data` mostra come in ingresso vengano presentati tutti i valori dei pixel. Dopo ogni lettura vengono aggiornati `min` e `max`. In particolare, `min` viene settato a 46 quando viene letto il primo pixel e poi resta invariato, mentre `max` viene settato a inizialmente a 46 e diventa `131` una volta letto il secondo pixel, per poi rimanere invariato.

Dopo la prima lettura dei pixel si alternano la lettura di un pixel e la scrittura in memoria del pixel equalizzato (`o_address` mostra l'alternarsi degli indirizzi con cui interagire). Si può notare come le operazioni eseguite abbiano durata diversa osservando il tempo per cui `o_address` resta ad un certo valore.

Una volta scritto in memoria l'ultimo pixel equalizzato viene alzato `o_done` e si attende che `i_start` (nell'immagine `tb_start`) venga posto a `0` per riabbassarlo. A quel punto vengono resettati tutti i segnali e il modulo è pronto per una nuova elaborazione.

4.2. Test specifici

Oltre ai test casuali sono stati studiati dei casi di test per verificare il comportamento del modulo sotto condizioni particolari. I test specifici effettuati sono stati:

- **Immagine nulla**: immagini che avessero i valori di righe e colonne a zero (contemporaneamente e non). Da immagini di questo tipo non ci si aspetta nessun risultato, anche l'output sarà un'immagine nulla. La corretta valutazione di questo caso è garantita, come già detto, dagli stati `GET_ROWS` e `GET_COLUMNS` della macchina a stati (paragrafo 2.2).
- **Immagine minima**: immagini formate da un solo pixel. Ci si aspetta che queste immagini vengano trasformate in singoli pixel con valore `0`. Per garantire la corretta valutazione dei valori massimo e minimo dei pixel in questo tipo di immagini è necessario inizializzare i segnali che salvano queste informazioni rispettivamente a `0` e `255`. Inoltre, questo test è particolarmente importante in seguito alla scelta di utilizzare un bit in meno per i contatori di righe e colonne (paragrafo 2.5) in quanto garantisce che la gestione del primo step di tali contatori sia corretta.
- **Immagine massima**: immagini 128x128. Questo test permette di essere sicuri che il numero di bit dei segnali che trattano le dimensioni dell'immagine dei bit sia stato correttamente dimensionato. In particolare, nel progetto si è deciso di utilizzare solo 7 bit per i contatori di righe e colonne, trattando lo zero come primo valore contato: questo tipo di test permette di garantire che tale scelta sia effetivamente valida.
- **Tutti i delta**: immagini 2x2 che presentassero tutte le combinazioni di valori massimo-minimo possibili. Osservando la struttura dell'algoritmo (paragrafo 1.2) diventa subito evidente come questo permetta di coprire tutti i possibili cammini dell'algoritmo. Infatti, questo dipende solo dalla coppia minimo-massimo dei valori dei pixel e dal valore del pixel corrente. Come detto, questi test provano tutte le combinazioni max-min, di conseguenza anche tutte le combinazioni di pixel corrente e minimo.
- **Tutti i pixel uguali**: Immagini di diverse dimensioni che presentano lo stesso valore per tutti i pixel; questo test può essere considerato un'estensione del test "immagine minima" e ci si aspetta come risultato immagini con tutti i pixel a `0`.
- **Reset durante l'elaborazione**: un test in cui viene alzato esternamente il segnale di reset durante l'elaborazione. Questo assicura che il modulo possa essere resettato in qualsiasi momento senza causare alcun malfunzionamento. Il metodo con cui questo caso viene gestito è stato presentato nel paragrafo 2.5.
- **Reset asincrono**: un test in cui viene inviato il segnale esterno di reset non allineato rispetto al fronte di salita del clock. Anche questo gestione è stata descritta nel paragrafo 2.5.

5. Conclusione

Il modulo esegue correttamente la funzione richiesta. Come detto non si sono ricercate particolari ottimizzazioni che tuttavia sono sicuramente possibili.

L'architettura scelta è facilmente estendibile modificando pochi parametri. A tal fine, un'ottimizzazione dal punto di vista del codice VHDL sarebbe la generalizzazione di alcune righe che al momento sono fortemente legate ai limiti dichiarati nella consegna.

Per concludere, come già detto il modulo risulta sintetizzabile e, inoltre, è implementabile su una FPGA xc7a200tfbg484-1.