Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria

Progetto di Reti Logiche
2020-2021

Riccardo Pomarico
10661306

Studente: Riccardo Pomarico
Matricola: 910309
Codice Persona: 10661306
Email: riccardo.pomarico@mail.polimi.it
Professore: Gianluca Palermo
Anno Accademico: 2020/2021

Indice

1. Introduzione 4
1.1 Descrizione ad alto livello dell'implementazione

2. Struttura del codice 6
2.1 Processo state_reg_process
2.2 Processo lambda_process
2.3 Processo delta_process

3. FSM scelto 9

4. Schematic 11

5. Risultato di simulation 12

6. Test benches 13
6.1 Immagine 128x128
6.2 Immagine da 1 pixel
6.3 Immagine da 0 pixel
6.4 Equalizzazione di 2 immagini successive
6.5 Equalizzazione di 3 immagini successive
6.6 Altri test assegnando valori casuali

1. Introduzione

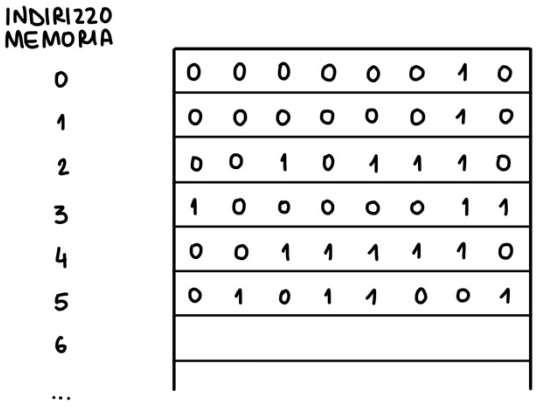
Il progetto richiesto consiste nell'implementazione in VHDL del metodo di codifica basato sul metodo di equalizzazione dell'istogramma di una immagine. L'obiettivo è quello di ricalibrare il contrasto quando i valori di intensità dell'immagine risultano troppo ravvicinati.

Nella versione sviluppata è richiesta l'implementazione dell'algoritmo solo per immagini in scala di grigi a 256 livelli.
Come da specifica funzionale del progetto, ad ogni indirizzo corrisponde un pixel dell'immagine. La dimensione della stessa è definita da 2 byte, ognuno di 8 bit, memorizzati rispettivamente all'indirizzo 0, per quanto riguarda la dimensione di colonna, e all'indirizzo 1, per quanto riguarda la dimensione di riga.

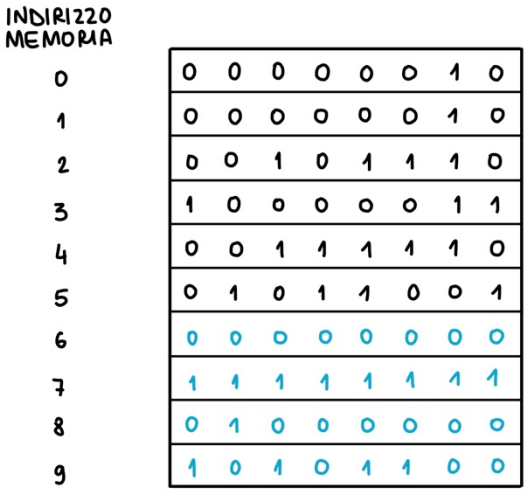
Esempio:

Immagine 2x2 con i seguenti valori: 46, 131, 62 e 89.

I pixel sono salvati negli indirizzi come in figura.



I nuovi pixel dell'immagine equalizzata (che saranno rispettivamente 0, 255, 64 e 172) verranno salvati a partire dall'indirizzo 6.



1.1 Descrizione ad alto livello dell'implementazione

Ad una prima lettura dei pixel dell'immagine da analizzare i valori vengono salvati in un array, e vengono individuati e poi assegnati il max_pixel_value e il min_pixel_value.
Si procede, quindi, con il calcolo del delta_value, dato dalla differenza tra il max_pixel_value e il min_pixel_value. Grazie al delta è possibile ricavare il valore dello shift, il quale è un numero intero con valore compreso tra 0 e 8, facilmente ricavabile da controlli a soglia, come da tabella sotto riportata.
Segue la conversione di ogni singolo pixel: si calcola il valore del nuovo pixel sottraendo al current_pixel_value quello del min_pixel_value, ed eseguendo lo shift relativo su una variabile da 16 bit, in tal modo, un eventuale valore maggiore o uguale a 256 potrà essere correttamente segnalato ed approssimato a 255. Infine si salva il valore trovato nel new_pixel in 8 bit, e si procede finchè non vengono convertiti tutti i pixel dell'immagine.

DELTA_VALUE	FLOOR(x)	SHIFT_LEVEL
0	0	8
1	1	7
2	1	7
3	2	6
⋮	⋮	⋮
6	2	6
7	3	5
8	3	5
⋮	⋮	⋮
14	3	5
15	4	4
16	4	4
⋮	⋮	⋮

DELTA_VALUE	FLOOR(x)	SHIFT_LEVEL
30	4	4
31	5	3
32	5	3
⋮	⋮	⋮
62	5	3
63	6	2
64	6	2
⋮	⋮	⋮
126	6	2
127	7	1
128	7	1
⋮	⋮	⋮
254	7	1
255	8	0

2. Struttura del codice

Nel codice sono stati definiti 9 signals e uno state_type per realizzare il FSM.

```
type state_type is (IDLE, RST, S0, S1, S2, S3, S4, S5, S6, S7);
signal current_state : state_type := IDLE;
signal next_state : state_type := IDLE;
signal check : std_logic := '0';
signal shift_level : integer range 0 to 8;
signal n_col : std_logic_vector(7 downto 0) := (others => '0');
signal n_rig : std_logic_vector(7 downto 0) := (others => '0');
signal delta_value : std_logic_vector(7 downto 0) := (others => '0');
signal max_pixel_value : std_logic_vector(7 downto 0) := (others => '0');
signal min_pixel_value : std_logic_vector(7 downto 0) := (others => '0');
```

Nel codice sono presenti 3 processi: *state_reg_process*, *lambda process* e *delta process*.

2.1 Processo state_reg process

Realizza il cambiamento di stato di FSM. Quando input signal i_rst viene portato a 1, FSM entra nello stato RST, quando si verifica falling_edge avviene un cambio di stato del FSM.

```
state_reg: process(i_clk, i_rst)
    begin
        if i_rst = '1' then
            current_state <= RST;
        elsif falling_edge(i_clk) then
            current_state <= next_state;
        end if;
    end process state_reg;
```

2.2 Processo lambda process

Definisce il cambiamento di stato del FSM. Come lo state_reg process, utilizza il falling edge per la sincronizzazione. La definizione della specifica della FSM si trova nelle sezioni successive.

```
lambda: process(current_state, i_rst, i_start, i_clk, check)
    begin
        if i_rst = '1' then
            next_state <= RST;
        elsif falling_edge(i_clk) then
            case current_state is
                when IDLE =>

                when RST =>
                    if i_start = '1' then
                        next_state <= S0;
                    end if;

                when S0 =>
                    next_state <= S1;

                when S1 =>
                    next_state <= S2;

                when S2 =>
                    next_state <= S3;

                when S3 =>
                    if check = '1' then
                        next_state <= S4;
                    else
                        next_state <= S3;
                    end if;

                when S4 =>
                    next_state <= S5;

                when S5 =>
                    if check = '0' then
                        next_state <= S6;
                    else
                        next_state <= S5;
                    end if;

                when S6 =>
                    next_state <= S7;

                when S7 =>
                    if i_start = '0' then
                        next_state <= RST;
                    else
                        next_state <= S7;
                    end if;
            end case;
        end if;
    end process lambda;
```

2.3 Processo delta process

Definisce l'esecuzione dei vari stati. Come nei processi precedenti, utilizza il falling edge per la sincronizzazione. Il processo contiene 8 variabili:

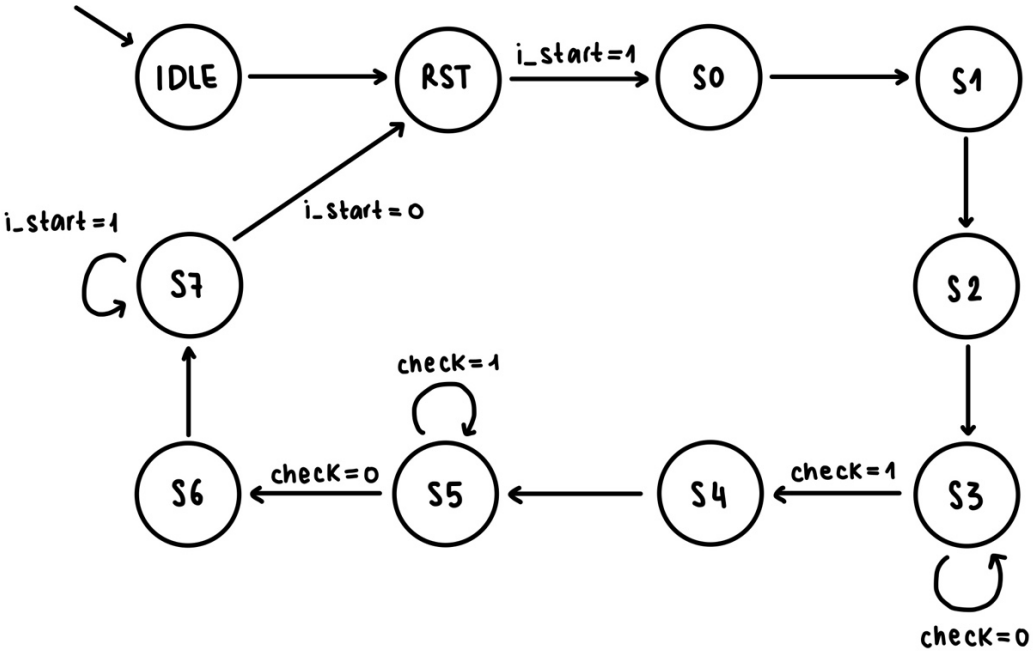
- ***controllo*** indica quel valore che incrementa di una unità ad ogni lettura di un pixel; quando il suo valore coincide con quello di `n_rig*n_col - 1`, il valore del signal `check` viene cambiato per consentire alla macchina di procedere con lo stato successivo;
- ***temp_new*** indica il valore del pixel dopo lo shift e viene rappresentato su 16 bit così da consentire, nel caso in cui il valore del pixel risulti maggiore o uguale di 256, l'approssimazione a 255;
- ***current_pixel_value*** indica il valore del pixel letto;
- ***new_pixel*** indica il valore del nuovo pixel equalizzato;
- ***temp_pixel*** indica il valore dato dalla differenza tra `current_pixel_value` e `min_pixel_value`;
- ***i*** indica la posizione dell'array in cui avviene sia l'operazione di scrittura, come accade negli stati S2 e S3, sia l'operazione di lettura, come accade nello stato S5;
- ***totalcount*** indica quel valore che incrementa di una unità ogni volta che la macchina passa all'indirizzo successivo; in questo modo, durante l'operazione finale di scrittura dei pixel equalizzati, si usa questa variabile come parametro per l'indirizzo dove effettuare l'output;
- ***v*** indica l'array usato per memorizzare i pixel dell'immagine da equalizzare dopo la lettura.

```
delta: process(current_state, i_clk, i_start, i_data, check,
shift_level, n_col, n_rig, delta_value, max_pixel_value,
min_pixel_value)

    variable controllo : std_logic_vector(15 downto 0:=(others => '0');
    variable temp_new : std_logic_vector(15 downto 0):=(others => '0');
    variable current_pixel_value : std_logic_vector(7 downto 0):=(others=>'0');
    variable new_pixel : std_logic_vector(7 downto 0):=(others => '0');
    variable temp_pixel : std_logic_vector(7 downto 0):=(others => '0');
    variable i : integer range 0 to 16384;
    variable totalcount : integer;
    TYPE array1 is ARRAY (0 to 16384) of std_logic_vector(7 downto 0);
    variable v : array1;

    begin
        if falling_edge(i_clk) then
            case current_state is
```

3. FSM scelto



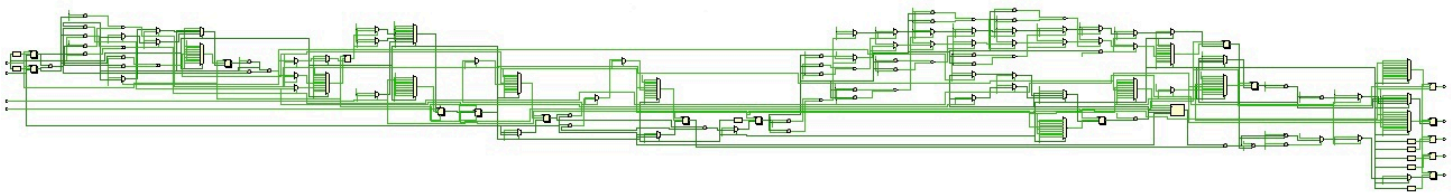
Signal check: indica se è possibile proseguire con lo stato successivo.

Stato:

IDLE	Attende l'i_rst signal senza effettuare nessun lavoro.
RST	Inizializza tutti i signal e le variabili; quando l'i_start signal viene portato a 1, alza l'o_en signal a 1.
S0	Memorizza il valore della dimensione di colonna.
S1	Memorizza il valore della dimensione di riga.
S2	Legge il primo pixel dell'immagine e lo memorizza in un array dedicato. Fissa come valore massimo e come valore minimo dell'immagine il primo pixel letto.
S3	Ad ogni ciclo legge un nuovo pixel dell'immagine e lo memorizza in un array dedicato. Controlla se il valore letto è più grande del max_pixel_value, o più piccolo del min_pixel_value, aggiornando in caso il segnale.

S4	Memorizza il valore del delta_value e il relativo shift_level.
S5	Per ogni pixel memorizzato nell'array compie lo shift e lo assegna come new_pixel.
S6	Una volta terminate le operazioni con tutti i pixel, pone l'o_done signal a 1.
S7	Pone l'o_done signal a 0.

4. Schematic



5. Project Summary

Overview | Dashboard

Settings Edit

Project name:	riccardopomarico
Project location:	C:/Users/Ospite/Desktop/riccardopomarico
Product family:	Artix-7
Project part:	xc7a200tiffv1156-1L
Top module name:	project_reti_logiche
Target language:	VHDL
Simulator language:	VHDL

Synthesis

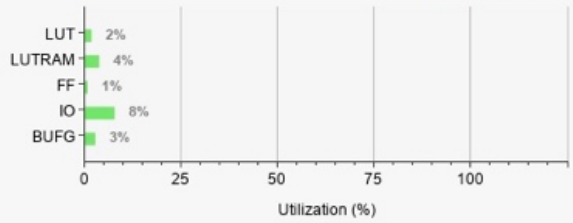
Status:	✓ Complete
Messages:	! 1 warning
Part:	xc7a200tiffv1156-1L
Strategy:	Vivado Synthesis Defaults
Report Strategy:	Vivado Synthesis Default Reports
Incremental synthesis:	None

Implementation **Summary** | Route Status

Status:	✓ Complete
Messages:	! 1 warning
Part:	xc7a200tiffv1156-1L
Strategy:	Vivado Implementation Defaults
Report Strategy:	Vivado Implementation Default Reports
Incremental implementation:	None

Utilization Post-Synthesis | **Post-Implementation**

Graph | Table



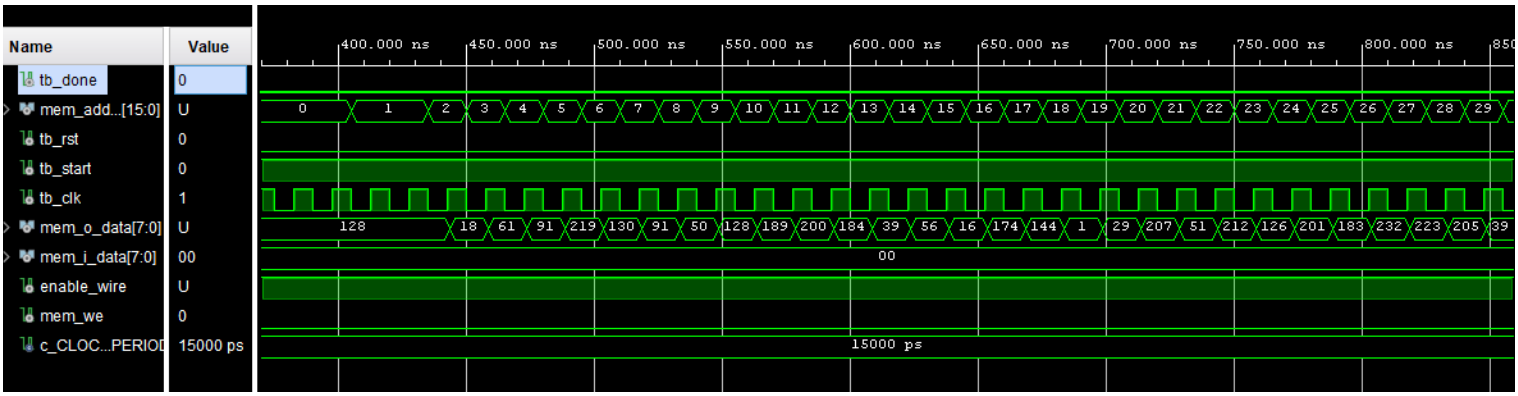
Power **Summary** | On-Chip

Total On-Chip Power:	**0.113 W**
Junction Temperature:	**25,2 °C**
Thermal Margin:	74,8 °C (51,2 W)
Effective ϑJA:	1,5 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

6. Test personali

6.1 Immagini 128x128



Ho effettuato alcuni test con immagini da 255 pixel. Lo scopo del test bench è la verifica del corretto funzionamento del progetto per l'intero spazio di indirizzamento consentito.

6.2 Immagine da 1 pixel

Il test bench verifica che un'immagine costituita da 1 pixel venga equalizzata correttamente.

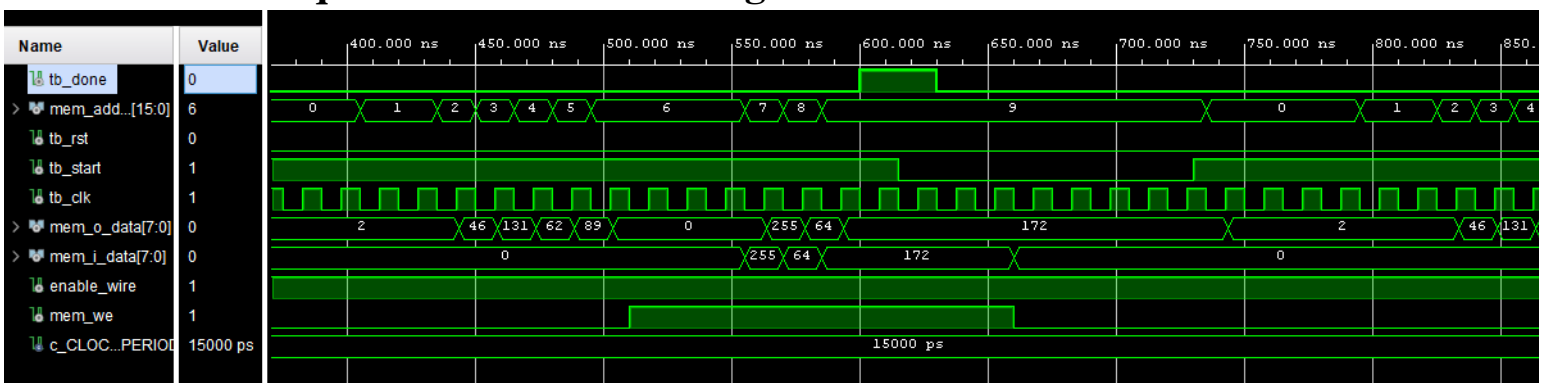
6.3 Immagine da 0 pixel

Il test bench verifica che un'immagine costuita da 0 pixel venga equalizzata correttamente.

6.4 Equalizzazione di 2 immagini successive

Il test bench verifica che l'equalizzazione di 2 immagini successive avvenga correttamente, controllando, quindi, che vengano gestiti correttamente i valori di START e di DONE e che la seconda elaborazione avvenga senza attendere il reset.

6.5 Equalizzazione di 3 immagini successive



Il test bench verifica che l'equalizzazione di 3 immagini successive avvenga correttamente, controllando, quindi, che vengano gestiti correttamente i valori di START e di DONE e che la seconda e terza elaborazione avvengano senza attendere il reset.

6.6 Altri test assegnando valori casuali

Il test bench verifica che il circuito codifichi correttamente i pixel presenti in immagini da equalizzare di dimensioni diverse, dando in risposta un output corretto.