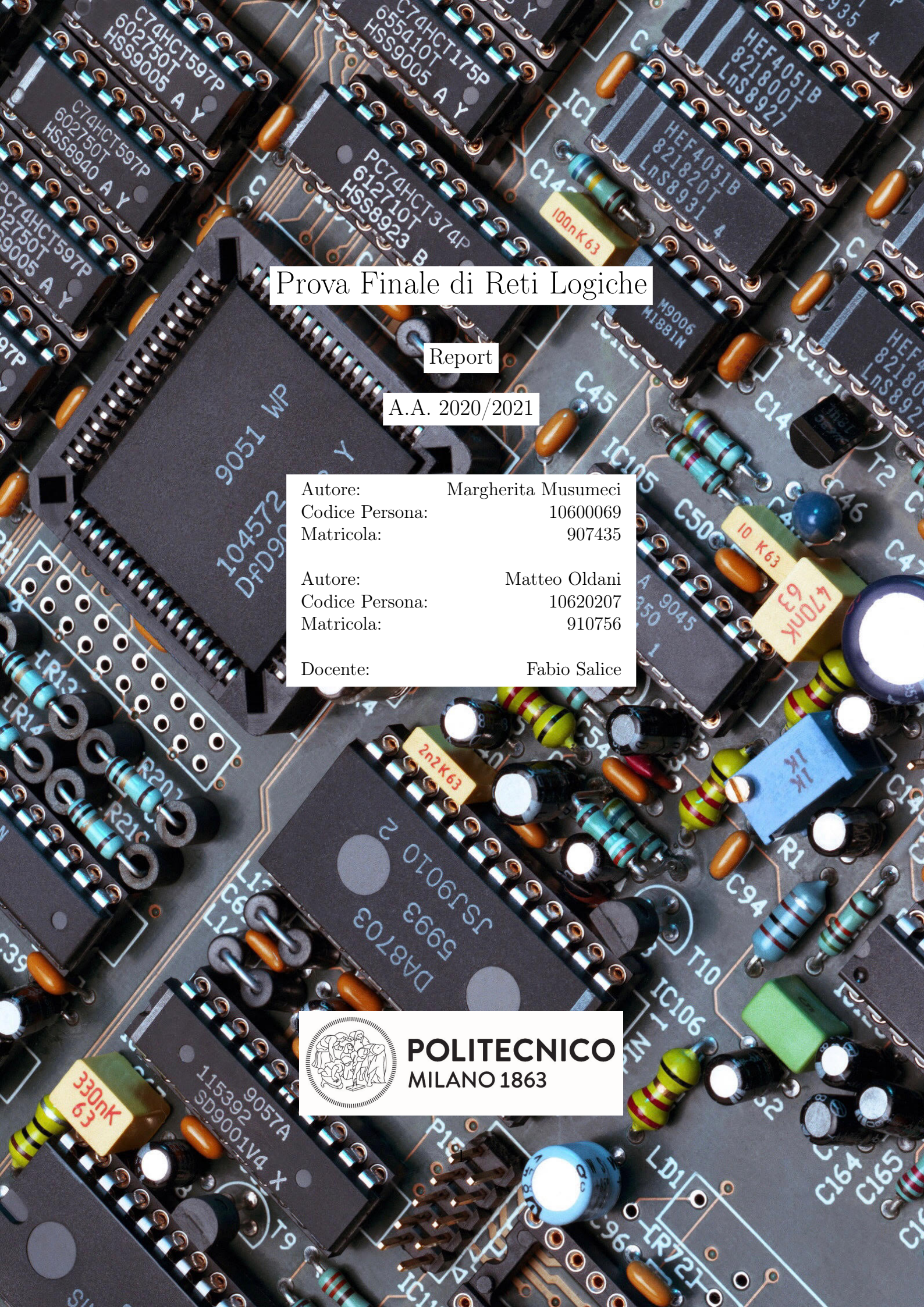


Prova Finale di Reti Logiche

Report

A.A. 2020/2021

Autore:	Margherita Musumeci
Codice Persona:	10600069
Matricola:	907435
Autore:	Matteo Oldani
Codice Persona:	10620207
Matricola:	910756
Docente:	Fabio Salice

POLITECNICO
MILANO 1863

Indice

1 Introduzione

1.1 Scopo del progetto

Il progetto richiesto consiste nell'implementazione in VHDL del metodo di equalizzazione dell'istogramma di una immagine. L'istogramma di un'immagine è una funzione che per ogni livello di grigio, riporta il numero di pixel aventi quel valore. L'equalizzazione è una tecnica che mira a modificare la forma dell'istogramma ridistribuendo i valori dei livelli di grigio in modo che l'istogramma sia quanto più uniforme possibile. L'obiettivo è quello di migliorare il contrasto dell'immagine.

1.2 Specifiche generali

L'istogramma di un'immagine a L valori di intensità è la funzione discreta:

$$h(r_k) = n_k, \qquad r_k \in [0, ..., L-1] \tag{1}$$

dove n_k è il numero di pixel aventi intensità r_k. È utile, però, considerare l'istogramma normalizzato rispetto al numero totale di pixel dell'immagine:

$$p(r_k) = \frac{n_k}{R * C} \tag{2}$$

dove R e C rappresentano rispettivamente il numero di righe e di colonne dell'immagine. L'equalizzazione si ottiene trasformando i pixel di intensità r_k in s_k come segue:

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p(r_j) = \frac{(L-1)}{R * C}\sum_{j=0}^{k} n_j \tag{3}$$

A noi non era richiesto di implementare l'algoritmo standard ma una sua versione semplificata. In particolare:

- Verranno sottoposte a equalizzazione solamente immagini in scala di grigi a 256 livelli;
- L'immagine non può superare la dimensione massima di 128 x 128 pixel;
- La funzione di trasformazione è

$$s_k = min[255, (curr_{pixel} - min_{pixel}) * 2^{8-\lfloor \log_2 (max_{pixel} - min_{pixel} + 1) \rfloor}] \tag{4}$$

Inoltre, l'implementazione deve essere in grado di gestire sia un segnale di Reset, che l'elaborazione sequenziale di più immagini.

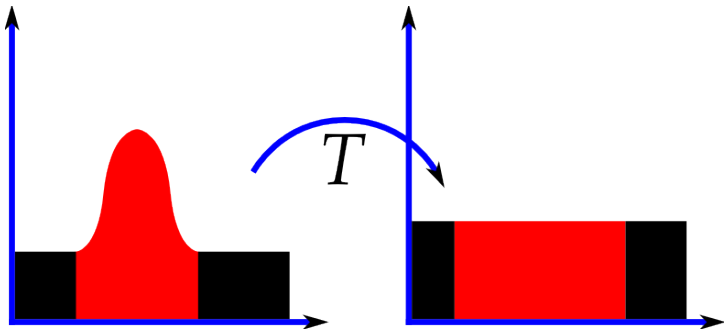


Figura 1: Equalizzazione dell'istogramma

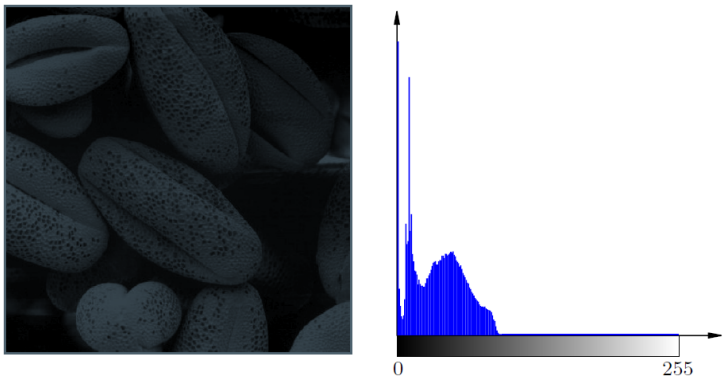


Figura 2: Esempio di immagine non equalizzata. Le componenti dell'istogramma sono localizzate in una banda piuttosto stretta.

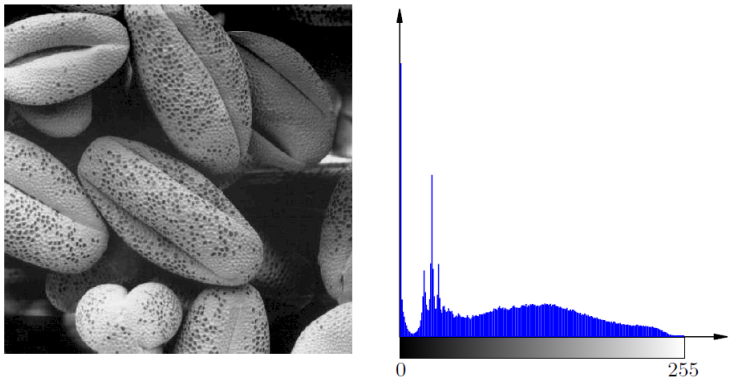


Figura 3: Esempio di immagine equalizzata. Le componenti dell'istogramma sono distribuite su tutto l'intervallo rappresentabile, la distribuzione è quasi uniforme con pochi picchi.

1.3 Descrizione della memoria

Il modulo comunica con la memoria RAM avente un indice di indirizzamento al byte.

- All'indirizzo 0 è salvato il numero di colonne (C) dell'immagine;
- All'indirizzo 1 è salvato un valore che si riferisce al numero di righe (R) dell'immagine;
- I pixel dell'immagine originale sono salvati sequenzialmente partendo dall'indirizzo 2;
- I pixel dell'immagine equalizzata saranno memorizzati a partire dall'indirizzo 2+(C*R).

1.4 Interfaccia del componente

```
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_clock	Segnale di CLOCK dato in ingresso dal Test Bench
i_rst	Segnale di RESET che riporta la macchina nello stato iniziale e la prepara ad una nuova esecuzione
i_start	Segnale di START generato dal Test Bench per iniziare l'esecuzione
i_data	Vettore di segnali in cui viene caricato il contenuto della memoria RAM a seguito di una richiesta di lettura
o_address	Segnale di uscita che indica l'indirizzo della memoria su cui lavorare

o_done	Segnale di uscita che comunica la fine dell'elaborazione
o_en	Segnale di ENABLE che abilita l'accesso in memoria
o_we	Segnale di ENABLE che abilita la scrittura in memoria
o_data	Vettore di segnali di output del componente che contiene il dato da scrivere in memoria

1.5 Metodo di sintetizzazione

Strumento di sintesi utilizzato: XILINX VIVADO WEBPACK
FPGA target: FPGA xc7a200tfbg484-1.

1.6 Esempio

RAM all' inizio della computazione:

Indirizzo	Contenuto
0	2
1	4
2	76
3	131
4	109
5	89
6	46
7	121
8	62
9	59
10	-
11	-
12	-
13	-
14	-
15	-
16	-
17	-

RAM al termine della computazione:

Indirizzo	Contenuto
0	2
1	4
2	76
3	131
4	109
5	89
6	46
7	121
8	62
9	59
10	120
11	255
12	252
13	172
14	0
15	255
16	64
17	67

Esempio di applicazione dell'algoritmo al primo pixel:

$$s_k = min[255, (curr_{pixel} - min_{pixel}) * 2^{8-\lfloor \log_2 (max_{pixel} - min_{pixel} + 1) \rfloor}] \quad (5)$$

Dove:

min_{pixel}= 46

max_{pixel}= 131

$\lfloor log_2(max_{pixel} - min_{pixel} + 1) \rfloor = \lfloor 6, 426 \rfloor -> 6$

$2^{8-\lfloor log_2(max_{pixel} - min_{pixel} + 1) \rfloor} = 4$

$\text{pixel}_1 = min[255, (76 - 46) * 4] = min[255, 120] = 120$

2 Architettura

Per poter risolvere il problema presentato si è deciso di progettare una macchina a stati finiti che rendesse possibile la risoluzione della problematica presentata nella specifica in VHDL.

2.1 Funzionamento alto livello

La macchina dovrà svolgere i seguenti passi:

a. Accedere alla memoria RAM;

b. Valutare l'istogramma;

c. Calcolare la funzione di equalizzazione;

d. Eseguire la trasformazione

e. Scrivere il risultato in memoria.

f. Terminare l'esecuzione o ripetere dal passo 'd'.

In particolare, la valutazione dell'istogramma consiste nel determinare il range di valori che assume l'immagine originale (individuare min e max).

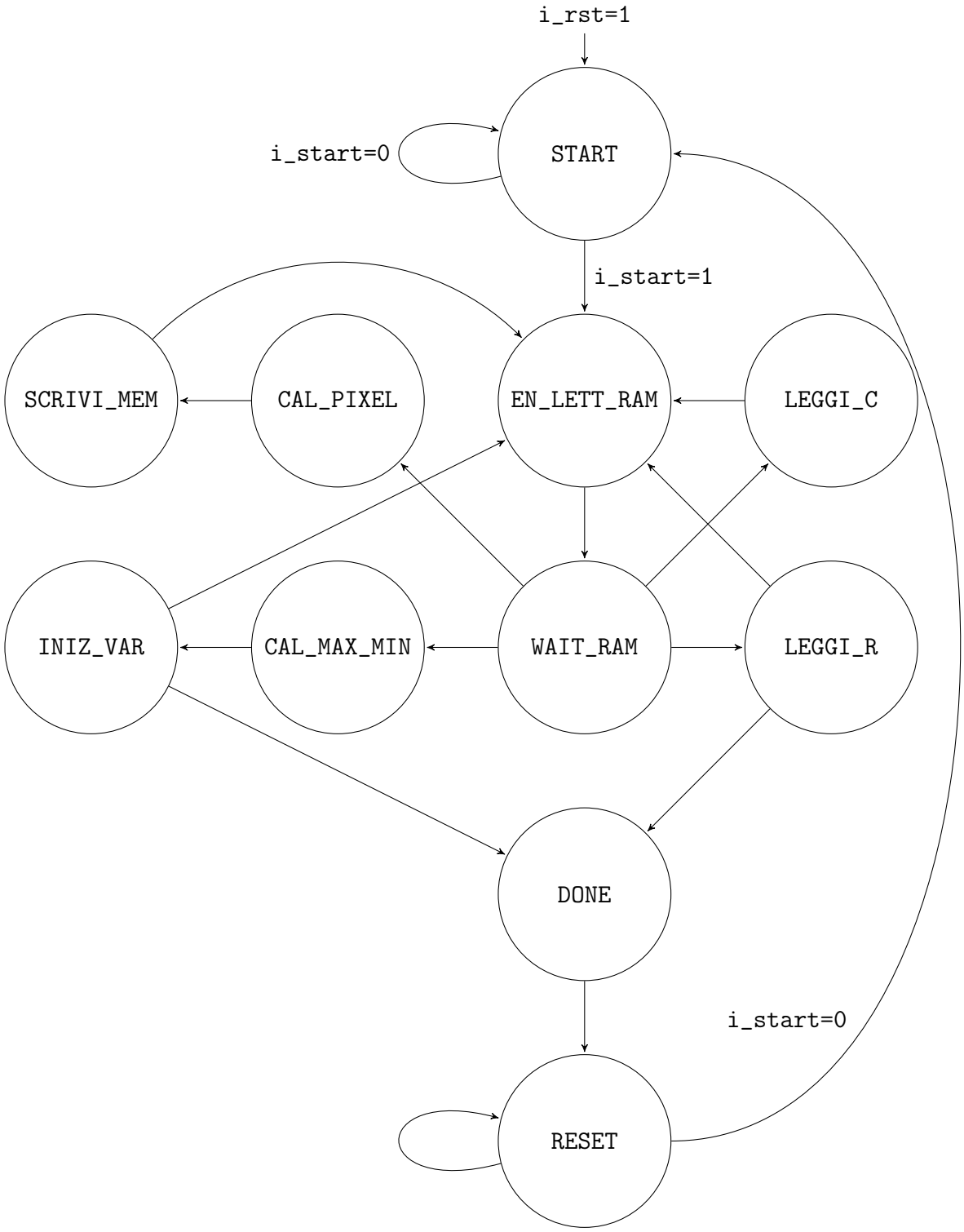
Il calcolo della funzione di equalizzazione è un'azione che a sua volta si suddivide in

- Calcolo del delta -> $max_{pixel} - min_{pixel}$
- Calcolo del logaritmo -> $\log_2(delta + 1)$
- Calcolo dello shift -> 8 -$\lfloor log_2(delta + 1) \rfloor$

Mentre la trasformazione dell'immagine avviene pixel per pixel. Esso viene letto, trasformato e riscritto nell'apposita zona della memoria.

2.2 Diagramma degli stati della Macchina a Stati Finiti

È dato per scontato e quindi non riportato nello schema, che a fronte di un segnale di Reset, in qualunque stato si trovi la macchina, torna a START.


i_rst=1
START
i_start=0
i_start=1
SCRIVI_MEM
CAL_PIXEL
EN_LETT_RAM
LEGGI_C
INIZ_VAR
CAL_MAX_MIN
WAIT_RAM
LEGGI_R
DONE
i_start=0
RESET

2.3 Descrizione degli stati

START	Stato iniziale in cui la macchina rimane finché il segnale di START non viene posto a 1. In quel caso inizializza i segnali e le variabili interne
EN_LETT_RAM	Stato in cui vengono impostati i segnali o_en e o_address
WAIT_RAM	Stato in cui attendo che la RAM possa caricare il valore richiesto. I cicli di clock di attesa sono sfruttati per incrementare l'indirizzo di lettura
LEGGI_C	Stato in cui salvo il dato relativo al numero di colonne
LEGGI_R	Stato in cui salvo il dato relativo al numero di righe e calcolo la dimensione effettiva dell'immagine
CAL_MAX_MIN	Stato in cui verifico se il valore letto in RAM è un caso di massimo o di minimo confrontandolo con i pixel letti fino a quel momento
INIZ_VAR	Stato in cui viene calcolata la funzione di equalizzazione
CAL_PIXEL	Stato in cui viene applicata la funzione di equalizzazione al pixel letto dell'immagine originale e vengono impostati i segnali che permettono la scrittura in memoria
SCRIVI_MEM	Stato in cui attendo l'avvenuta scrittura in memoria. I cicli di clock di attesa sono sfruttati per incrementare l'indirizzo di scrittura
DONE	Stato in cui imposto il segnale o_done alto per indicare la terminazione della computazione in corso dell'immagine
RESET	Stato in cui rimane la macchina finché il segnale i_start non viene posto a 0, in tal caso viene posto anche o_done a 0 e la macchina è pronta alla computazione di una nuova immagine

2.4 Segnali interni e registri utilizzati

Per descrivere la macchina in VHDL si è deciso di specificare un tipo di dato (type_state) che contenesse una lista di tutti gli stati della macchina. Questo tipo di dato è stato poi usato per gestire le transazioni di stato (next_state).

address_read	Segnale che indica l'indirizzo contenente il pixel da leggere

address_write	Segnale che indica l'indirizzo dove andrà scritto il pixel equalizzato
delta	Variabile in cui viene salvato il risultato della differenza $(max_{pixel} - min_{pixel})$
shift	Variabile in cui viene salvato di quanto devo shiftare il valore originale
temp	Variabile in cui viene salvato il risultato del pixel equalizzato
max	Variabile che conterrà il livello massimo raggiunto dall'immagine
min	Variabile che conterrà il livello minimo raggiunto dall'immagine
colonne	Variabile che conterrà il numero di colonne dell'immagine
righe	Variabile che conterrà il numero di righe dell'immagine
log	Variabile in cui viene salvato il risultato del logaritmo
esp	Variabile ausiliaria per il calcolo del logaritmo
scelta_lettura	Variabile booleana che permette di discriminare tra la prima lettura di analisi dell'immagine e la seconda in cui opera l'equalizzazione
count	Variabile che conterrà la dimensione totale dell'immagine

2.5 Funzionamento nel dettaglio

La macchina parte dallo stato di START dal quale esce solo a seguito dell'arrivo del segnale i_start. Il primo compito eseguito dalla macchina è il salvataggio delle dimensioni dell'immagine (colonne e righe) tramite gli stati LEGGI_COLONNE e LEGGI_RIGHE.

Vengono, quindi, letti i pixel dell'immagine originale sequenzialmente al fine di individuare i livelli minimo e massimo assunti tramite lo stato CALCOLA_MIN_MAX. Individuati i valori, si procede calcolando la funzione di equalizzazione all'interno dello stato INIZIALIZZA_VARIABILI. Per il calcolo del logaritmo si è scelto di utilizzare l'algoritmo seguente:

a. inizializzazione della variabile esp=1;

b. inizio del ciclo;

c. confronto dell' esp con l'argomento del logaritmo da calcolare;

d. se l'argomento del logaritmo è maggiore di esp allora il risultato è il numero di cicli compiuti decrementato di uno;

e. se l'argomento è minore viene raddoppiato esp e rieseguito il ciclo (torno al punto b).

Infine, la macchina rilegge l'immagine originale pixel per pixel. Ad ogni lettura calcola il valore equalizzato e lo scrive nell'apposito spazio di memoria tramite gli stati CALCOLA_NUOVO_PIXEL e SCRIVI_IN_MEMORIA.

La macchina segnala la terminazione della computazione entrando nello stato DONE e impostando il segnale o_done alto. Si metterà poi nello stato di RESET nel quale rimarrà finché il segnale i_start non viene messo a 0. Successivamente la macchina sarà pronta per una nuova computazione.

Per la gestione della lettura la macchina fa uso dei due stati: ENABLE_LETTURA_RAM e WAIT_RAM. I due stati sono comuni per tutte le letture del programma. Lo stato WAIT_RAM è in grado di gestire l'evoluzione della macchina (scelta dello stato prossimo) tramite segnali di controllo.

2.6 Scelte progettuali

Per l'implementazione si è scelto di descrivere un componente in logica sequenziale. Si è scelto di utilizzare un singolo processo per migliorare la comprensibilità dell'implementazione a scapito di una massimizzazione dell'ottimizzazione non richiesta dalla complessità del problema.

Il componente è descritto da un solo processo che viene attivato dalla variazione del clock o da una variazione del segnale di reset. Il controllo del segnale di reset risulta asincrono in quanto la sua variazione viene valutata indipendentemente dalla variazione del segnale di clock. L'evoluzione della macchina, invece, è sincronizzata con il fronte di salita del clock.

Gli stati LEGGI_COLONNE e LEGGI_RIGHE potevano essere accorpati introducendo un segnale booleano che indicava l'operazione da compiere. Si è scelto di non intraprendere quella strada a vantaggio di una maggior chiarezza dell'implementazione.

La scelta di utilizzare l'operatore '*' all'interno del progetto è stata giustificata dal fatto che VIVADO è in grado di ottimizzare la moltiplicazione di due variabili da 8 bit in fase di sisntesi. Infatti, dopo aver reimplementato l'algoritmo senza l'uso dell'operatore '*', non abbiamo riscontrato differenze significative nei risultati dei test eseguiti sulle due implementazioni sia in termini di tempo che di risorse utilizzate.

Una possibile alternativa più ottimale sarebbe parallelizzare l'elaborazione del pixel i-esimo con la scrittura in memoria del pixel precedentemente equalizzato.

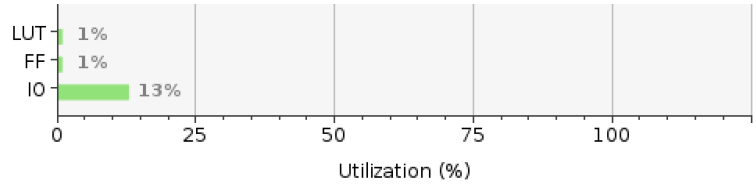
3 Sintesi

3.1 Report Utilization

Analizzando i report forniti da Vivado possiamo valutare il numero di componenti utilizzati. Dal "Report Utilization" leggiamo che il progetto utilizza 367 LUT e 129 FF, una percentuale irrisoria della disponibilità della FPGA.

Summary

Resource	Utilization	Available	Utilization %
LUT	373	134600	0.28
FF	129	269200	0.05
IO	38	285	13.33



3.2 Timing Report

Analizzando il "Timing Report" si può vedere quanto del tempo di clock sia effettivamente utile per svolgere il lavoro. Si è ottenuto con periodo di clock del test bench di 100.000ns un Worst Negative Slack pari a 91.005ns.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	91.005 ns	Worst Hold Slack (WHS):	0.139 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:	284	Total Number of Endpoints:	284	Total Number of Endpoints:	130

All user specified timing constraints are met.

4 Simulazioni

Il componente, dopo aver superato i test generici, è stato sottoposto a test benches che ne verificavano il funzionamento in situazioni limite. Di seguito una breve descrizione di essi:

4.1 Test Bench con reset asincrono

Caso di test che valuta il comportamento del componente a seguito della ricezione del segnale di reset durante la computazione dell'immagine. La scelta di sottoporlo a tale test è dettata della scelta implementativa. Infatti si è scelto di introdurre il segnale di reset asincrono rispetto al segnale di clock.

Effettivamente non viene compromesso il funzionamento del componente, ma si porta la macchina dello stato START.

4.2 Test Benches con dimensioni delle immagini al limite del dominio

Casi di test che valutano il comportamento del componente quando deve elaborare immagini con dimensioni alle estremità del range consentito. In particolare, le dimensioni analizzate sono (CxR):

- 0x0
- 0xR
- Cx0
- 128x128

Con tali test verifichiamo come il componente gestisce immagini degeneri e immagini di grandi dimensioni.

L'algoritmo è stato ideato, fin da subito, per terminare solo dopo aver processato

$$count = CxR \tag{6}$$

pixel. Pertanto, se il componente si accorge di dover processare un'immagine degenere termina la computazione senza applicare l'algoritmo di equalizzazione.

4.3 Test Benches con livelli dei pixel al limite del dominio

Con questi test benches viene verificato il modo in cui il componente calcola l'algoritmo di equalizzazione se vengono sottoposte immagini con livelli di intensità estremi. Ossia:

- $min_{pixel} = 0$; $max_{pixel} = 255$
- $min_{pixel} = 0$; $max_{pixel} = casuale$
- $min_{pixel} = casuale$; $max_{pixel} = 255$

L'obiettivo di tali prove è testare se ai limiti del dominio non ci siano problemi di sottostima della dimensione dei segnali utilizzati.

4.4 Test Benches con pixel dello stesso valore

Tale insieme di casistiche completa il gruppo di test precedente. In particolare si testa la bontà dell'implementazione a fronte di un'immagine monocromatica. Includiamo all'interno di tali casi l'immagine composta da un solo pixel.

- $min_{pixel} = 0$; $max_{pixel} = 0$
- $min_{pixel} = 255$; $max_{pixel} = 255$
- $min_{pixel} = max_{pixel} = casuale$
- C=1 x R=1

Come atteso, l'immagine equalizzata risulta essere ancora monocromatica ma tutti i pixel assumono livello 0.

4.5 Test Benches con sequenze numerose di immagini

Infine, si è deciso di "stressare" il componente con una sequenza numerosa di immagini da elaborare. Sia immagini standard che immagini "eccezionali". In tal modo viene valutato il comportamento dell'entity quando deve codificare più immagini sequenzialmente.

5 Conclusioni

Per raggiungere la soluzione siamo partiti dalla costruzione teorica della macchina a stati. Questo lavoro, unito alla scelta di sviluppare un solo processo, ci ha permesso di descrivere il componente in VHDL in modo semplice e ordinato. Dopo averlo sottoposto a vari test e aver corretto eventuali imprecisioni, siamo riusciti ad ottenere un componente funzionante in post-sintesi.

Si ritiene che la struttura ideata rispetti le specifiche assegnate ed abbia tempi di funzionamento adeguati alla complessità del problema.

Il vincolo relativo al clock è stato rispettato. In aggiunta, effettuando vari test abbiamo concluso che la soglia minima del segnale di clock applicabile al design creato è 9.000ns a fronte dei 100.000ns richiesti.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.005 ns	Worst Hold Slack (WHS):	0.139 ns	Worst Pulse Width Slack (WPWS):	4.000 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:	284	Total Number of Endpoints:	284	Total Number of Endpoints:	130

All user specified timing constraints are met.

L'architettura è facilmente scalabile: nel caso si voglia aumentare il numero di livelli di grigio utilizzati o la dimensione dell'immagine è sufficiente aumentare la dimensione dei segnali.