Progetto finale di Reti Logiche

Prof. Fornaciari, Prof. Palermo e Prof. Salice

Versione (9 dicembre -> 28 dicembre)

MODIFICA 1:

In tutti gli esempi era riportato nel commento sia per il byte 0 che per il byte 1 che si trattava del numero di colonne. I commenti sono stati corretti segnalando che il byte 1 è relativo al numero di righe.

MODIFICA 2:

La definizione di SHIFT_LEVEL aveva una parentesi chiusa di troppo (erano 4 ora sono 3).

MODIFICA 3:

E' stato sistemato un problema sulle "ulteriori Note": L'elenco partiva da 3 invece che da 1.

Versione (30 novembre -> 9 dicembre)

MODIFICA 1:

Da:

[...] I **pixel della immagine equalizzata**, ciascuno di un 8 bit, sono memorizzati in memoria con indirizzamento al Byte partendo dalla posizione `2+(N-COL*N-RIG)+1`.

A:

[...] I **pixel della immagine equalizzata**, ciascuno di un 8 bit, sono memorizzati in memoria con indirizzamento al Byte partendo dalla posizione `2+(N-COL*N-RIG)`.

MODIFICA 2:

Da:

[...] Note ulteriori sulla specifica

- Si noti che nel modulo da implementare, FLOOR(LOG2(X)) è un numero intero con valori tra 0 e 7 facilmente ricavabile da controlli a soglia.

A:

[...] Note ulteriori sulla specifica

- Si noti che nel modulo da implementare, FLOOR(LOG2(DELTA_VALUE +1)) è un numero intero con valori tra 0 e 8 facilmente ricavabile da controlli a soglia.

Progetto finale di Reti Logiche

Prof. Fornaciari, Prof. Palermo e Prof. Salice

(AGGIORNATO AL 9 Dicembre 2020)

Descrizione generale

La specifica della Prova finale (Progetto di Reti Logiche) 2020 è ispirata al metodo di equalizzazione dell'istogramma di una immagine[1].

Il metodo di equalizzazione dell'istogramma di una immagine è un metodo pensato per ricalibrare il contrasto di una immagine quando l'intervallo dei valori di intensità sono molto vicini effettuandone una distribuzione su tutto l'intervallo di intensità, al fine di incrementare il contrasto.

fig.1 - Esempi di immagine pre e post equalizzazione (sorgente Wikipedia)

Nella versione da sviluppare non è richiesta l'implementazione dell'algoritmo standard ma di una sua versione semplificata. L'algoritmo di equalizzazione sarà applicato solo ad immagini in scala di grigi a 256 livelli e deve trasformare ogni suo 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.

Il modulo da implementare dovrà leggere l'immagine da una memoria in cui è memorizzata, sequenzialmente e riga per riga, l'immagine da elaborare. Ogni byte corrisponde ad un pixel dell'immagine.

La dimensione della immagine è definita da 2 byte, memorizzati a partire dall'indirizzo 0. Il byte all'indirizzo 0 si riferisce alla dimensione di colonna; il byte nell'indirizzo 1 si riferisce alla dimensione di riga. La dimensione massima dell'immagine è 128x128 pixel.

L'immagine è memorizzata a partire dall'indirizzo 2 e in byte contigui. Quindi il byte all'indirizzo 2 è il primo pixel della prima riga dell'immagine.

[1] https://it.wikipedia.org/wiki/Equalizzazione_dell%27istogramma

L’immagine equalizzata deve essere scritta in memoria immediatamente dopo l’immagine originale

Dati

Le **dimensioni dell’immagine**, ciascuna di dimensione di 8 bit, sono memorizzati in una memoria con indirizzamento al Byte partendo dalla posizione 0: il byte in posizione 0 si riferisce al numero di colonne (`N-COL`), il byte in posizione 1 si riferisce al numero di righe (`N-RIG`).

I **pixel dell’immagine**, ciascuno di un 8 bit, sono memorizzati in memoria con indirizzamento al Byte partendo dalla posizione 2.

I **pixel della immagine equalizzata**, ciascuno di un 8 bit, sono memorizzati in memoria con indirizzamento al Byte partendo dalla posizione `2+(N-COL*N-RIG)`.

Note ulteriori sulla specifica

1. Si noti che nel modulo da implementare, FLOOR(LOG2(DELTA_VALUE +1)) è un numero intero con valori tra 0 e 8 facilmente ricavabile da controlli a soglia.
2. Si faccia attenzione al numero di bit necessari in ogni passaggio.
3. Il modulo deve essere progettato per poter codificare più immagini, ma l’immagine da codificare non verrà mai cambiata all’interno della stessa esecuzione, ossia prima che il modulo abbia segnalato il completamento tramite il segnale DONE. Si veda il prossimo punto per il protocollo di re-start.
4. Il modulo partirà nella elaborazione quando un segnale START in ingresso verrà portato a 1. Il segnale di START rimarrà alto fino a che il segnale di DONE non verrà portato alto; Al termine della computazione (e una volta scritto il risultato in memoria), il modulo da progettare deve alzare (portare a 1) il segnale DONE che notifica la fine dell’elaborazione. Il segnale DONE deve rimanere alto fino a che il segnale di START non è riportato a 0. Un nuovo segnale start non può essere dato fin tanto che DONE non è stato riportato a zero. Se a questo punto viene rialzato il segnale di START, il modulo dovrà ripartire con la fase di codifica.
5. Il modulo deve essere progettato considerando che prima della prima codifica verrà sempre dato il reset al modulo. Invece, come descritto nel protocollo precedente, una seconda elaborazione non dovrà attendere il reset del modulo.

Interfaccia del Componente

Il componente da descrivere deve avere 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:

- il nome del modulo deve essere project_reti_logiche
- i_clk è il segnale di CLOCK in ingresso generato dal TestBench;
- 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 generato dal Test Bench;
- 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.

ESEMPIO:
La seguente sequenza di numeri mostra un esempio del contenuto della memoria al termine di una elaborazione. I valori che qui sono rappresentati in decimale, sono memorizzati in memoria con l'equivalente codifica binaria su 8 bit senza segno.

Esempio: (immagine 4 x 3 : *indirizzo* - *valore*) RANDOM

INDIRIZZO MEMORIA	VALORE	COMMENTO
0	4	\\ Byte più significativo numero colonne
1	3	\\ Byte meno significativo numero righe
2	76	\\ primo Byte immagine
3	131	
4	109	
5	89	
6	46	
7	121	
8	62	
9	59	
10	46	
11	77	
12	68	
13	94	\\ ultimo Byte immagine
14	120	\\ primo Byte immagine equalizzata (risultato)
15	255	
16	252	
17	172	
18	0	
19	255	
20	64	
21	52	
22	0	
23	124	
24	88	
25	192	

Esempio 2 - tra 0 e 120 (incremento di 10)

INDIRIZZO MEMORIA	VALORE	COMMENTO
0	4	\\ Byte più significativo numero colonne
1	3	\\ Byte meno significativo numero righe
2	0	\\ primo Byte immagine
3	10	
4	20	
5	30	
6	40	
7	50	
8	60	
9	70	

10	80	
11	90	
12	100	
13	120	\\ ultimo Byte immagine
14	0	\\ primo Byte immagine equalizzata (risultato)
15	40	
16	80	
17	120	
18	160	
19	200	
20	240	
21	255	
22	255	
23	255	
24	255	
25	255	

Esempio 3 tra 122 e 133 centrato in 128

INDIRIZZO MEMORIA	VALORE	COMMENTO
0	4	\\ Byte più significativo numero colonne
1	3	\\ Byte meno significativo numero righe
2	122	\\ primo Byte immagine
3	123	
4	124	
5	125	
6	126	
7	127	
8	128	
9	129	
10	130	
11	131	
12	132	
13	133	\\ ultimo Byte immagine
14	0	\\ primo Byte immagine equalizzata (risultato)
15	32	
16	64	
17	96	
18	128	
19	160	
20	192	
21	224	
22	255	
23	255	
24	255	
25	255	

Esempio 4 - tra valori 0 128 e 255

INDIRIZZO MEMORIA	VALORE	COMMENTO
0	4	\\ Byte più significativo numero colonne
1	3	\\ Byte meno significativo numero righe
2	0	\\ primo Byte immagine
3	0	
4	0	
5	0	
6	128	
7	128	
8	128	
9	128	
10	255	
11	255	
12	255	
13	255	\\ ultimo Byte immagine
14	0	\\ primo Byte immagine equalizzata (risultato)
15	0	
16	0	
17	0	
18	128	
19	128	
20	128	
21	128	
22	255	
23	255	
24	255	
25	255	

Esempio5 - da 0 a 255 in intervalli regolari

INDIRIZZO MEMORIA	VALORE	COMMENTO
0	4	\\ Byte più significativo numero colonne
1	3	\\ Byte meno significativo numero righe
2	0	\\ primo Byte immagine
3	23	
4	46	
5	69	
6	92	
7	115	
8	139	
9	162	
10	185	
11	208	
12	231	
13	255	\\ ultimo Byte immagine
14	0	\\ primo Byte immagine equalizzata (risultato)

15	23
16	46
17	68
18	92
19	115
20	139
21	162
22	185
23	208
24	231
25	255

APPENDICE: Descrizione Memoria
NOTA: **La memoria è già istanziata all'interno del Test Bench e non va sintetizzata**

La memoria e il suo protocollo può essere estratto dalla seguente descrizione VHDL che fa parte del test bench e che è derivata dalla User guide di VIVADO disponibile al seguente link:
https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_3/ug901-vivado-synthesis.pdf

```
-- Single-Port Block RAM Write-First Mode (recommended template)
--
-- File: rams_02.vhd
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity rams_sp_wf is
port(
  clk  : in  std_logic;
  we   : in  std_logic;
  en   : in  std_logic;
  addr : in  std_logic_vector(15 downto 0);
  di   : in  std_logic_vector(7 downto 0);
  do   : out std_logic_vector(7 downto 0)
);
end rams_sp_wf;

architecture syn of rams_sp_wf is
type ram_type is array (65535 downto 0) of std_logic_vector(7 downto 0);
signal RAM : ram_type;
begin
  process(clk)
    begin
    if clk'event and clk = '1' then
      if en = '1' then
        if we = '1' then
          RAM(conv_integer(addr)) <= di;
          do                      <= di after 2 ns;
        else
          do <= RAM(conv_integer(addr)) after 2 ns;
        end if;
      end if;
    end if;
  end process;
end syn;
```