Gangemi Giovanni Codice: 10539352 Matricola: 843719

Luciano Franchin Codice: Matricola:

Progetto Reti Logiche

Descrizione:

L'algoritmo che risolve il problema è stato realizzato utilizzando una macchina a stati sensibile ad un segnale di clock i_clk. Oltre a questo segnale esiste ne esiste anche uno di reset che riporta tutti i segnali utilizzati allo stato iniziale per poter iniziare un nuovo calcolo. Gli spostamenti da uno stato all'altro avvengono tramite il segnale my_address che viene prontamente incrementato in ogni stato per poter accedere allo stato successivo. Dopo aver trovato i dati necessari per il calcolo dell'area si procede alla sua memorizzazione nella memoria. Una volta finita la sua esecuzione, l'algoritmo si mette in attesa di un nuovo segnale per ripetere la procedura del calcolo.

Fasi dell'algoritmo:

Inizialmente lo stato si trova a "0000", questo stato passa subito al successivo cioè "0001" che passa poi allo stato "0010" dove inizierà la vera lettura da RAM. Questi primi due stati servono solo per spostarsi nella posizione contenente il numero di colonne che verrà memorizzato nel segnale col, visto che le prime due posizioni vanno lasciate libere per il posizionamento del risultato . Negli altri due stati leggiamo il numero di righe, quello delle colonne e la soglia e le memorizziamo rispettivamente nei segnali rig, col e soglia. In questo punto avviene il primo controllo per verificare se la figura abbia righe o colonne nulle. Caso per il quale l'algoritmo salta gli stati fino a giungere subito allo stato "0110" dove avviene il calcolo del risultato e la sua successiva memorizzazione.

In seguito, negli stati successivi leggiamo i valori dei pixel sigolarmente e, tramite l'uso di contatori, controlliamo se siamo arrivata nell'ultima casella dell'immagine. Terminata l'analisi ci spostiamo nello stato "1010" dove vengono calcolati il lato A e il lato B della figura. Passando allo stato "0110" avviene il calcolo dell'area che sarà poi salvata in memoria. Negli stati "0111" e "1000" avviene la scrittura del risultato sulla memoria e il passaggio allo stato di reset (avviene nello stato "1111") dove la macchina viene configurata per essere usata un'altra volta.

Funzionalità di ogni stato:

RST: Tutti i segnali utilizzati nella macchina vengono riportati al loro valore iniziale;

S0: Spostamento nella memoria;

S1: Spostamento nella memoria nella posizione delle colonne;

S2: Memorizzazione del numero delle colonne nel segnale col;

S3: Memorizzazione del numero delle righe nel segnale rig;

S4: Memorizzazione del numero della soglia nel segnale soglia e controllo della dimensione dell'immagine (eventuale salto in S6);

S5: Esecuzione dell'algoritmo in modo iterativo facendo i controlli di riga e colonna per Poter uscire dal ciclo e passare in S6;

S6: Calcolo del latoA e del latoB della figura;

S7: Calcolo del risultato;

S8: Scrittura del risultato nella posizione 1;

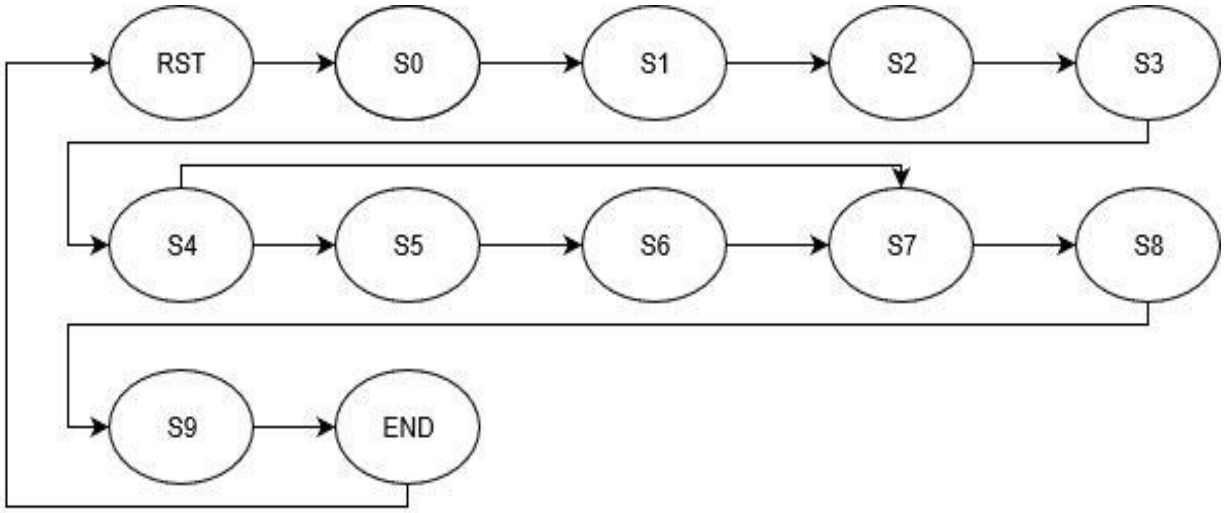
S9: Scrittura del risultato in posizione 0;

END: Ritorno allo stato di reset.

TestBench effettuati:

Sono stati effettuati tre gruppi di testbench. Questi tre vanno a calcolare una serie di 100 aree ciascuno calcolate in modo casuale. Abbiamo usato inoltre un testbench contenente 30 aree in serie con casi particolari (alcuni dei quali verranno aggiunti nella parte finale) , tra i quali un'immagine con righe o colonne nulle. Nella parte finale ho messo alcuni esempi di testBench utilizzati per la prova. Dal punto di vista della post sintesi il programma esegue il calcolo in 5.2 ns.

Disegno della macchina a stati:



Esempio di testBench effettuato:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity project_tb is
end project_tb;


architecture projecttb of project_tb is
constant c_CLOCK_PERIOD           : time := 15 ns;
signal   tb_done       : std_logic;
signal   mem_address         : std_logic_vector (15 downto 0) := (others
=> '0');
signal   tb_rst            : std_logic := '0';
signal   tb_start      : std_logic := '0';
signal   tb_clk            : std_logic := '0';
signal   mem_o_data,mem_i_data          : std_logic_vector (7 downto 0);
signal   enable_wire         : std_logic;
signal   mem_we        : std_logic;

type ram_type is array (65535 downto 0) of std_logic_vector(7 downto 0);
signal RAM: ram_type := (2 => "00011000", 3 =>"00000111",4 => "00010101",
30 => "00000011", 31 => "00000011", 32 => "00000011", 33 => "00000011",
36 => "00000111", 37 => "00000111", 38 => "00000111", 39 => "00000111",
42 => "00001011", 43 => "00001011", 44 => "00001011", 45 => "00001011",
48 => "00001111", 54 => "00000011", 60 => "00000111", 66 => "00011111",
72 => "00011001", 78 => "00000011", 79 => "00000011", 80 => "00000011",
84 => "00000111", 85 => "00000111", 86 => "00000111", 90 => "00011111",
91 => "00011111", 92 => "00011111", 96 => "00011001", 102 => "00000011",
108 => "00000111", 114 => "00011111", 120 => "00011001", 126 =>
"00000011", 132 => "00000111", 133 => "00000111", 134 => "00000111", 135
=> "00000111", 138 => "00001011", 139 => "00001011", 140 => "00001011",
141 => "00001011", 144 => "00001111", 145 => "00001111", 146 =>
"00001111", 147 => "00001111", others => (others =>'0'));
signal count : integer := 0;

component project_reti_logiche is
    port (
            i_clk         : in  std_logic;
            i_start       : in  std_logic;
            i_rst         : in  std_logic;
            i_data       : in  std_logic_vector(7 downto 0); --1 byte
            o_address     : out std_logic_vector(15 downto 0); --16 bit
addr: max size is 255*255 + 3 more for max x and y and thresh.
            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 component project_reti_logiche;


begin
```

```
      UUT: project_reti_logiche
      port map (
              i_clk          => tb_clk,
           i_start       => tb_start,
           i_rst        => tb_rst,
           i_data       => mem_o_data,
           o_address    => mem_address,
           o_done       => tb_done,
           o_en    => enable_wire,
              o_we      => mem_we,
           o_data    => mem_i_data
);

p_CLK_GEN : process is
  begin
    wait for c_CLOCK_PERIOD/2;
    tb_clk <= not tb_clk;
  end process p_CLK_GEN;


MEM : process(tb_clk)
   begin
    if tb_clk'event and tb_clk = '1' then
    if enable_wire = '1' then
      if mem_we = '1' then
         RAM(conv_integer(mem_address))              <= mem_i_data;
         mem_o_data                        <= mem_i_data after 1 ns;
      else
        if tb_rst = '1' then
              if count = 0 then
              --test # 0
                count <= 1;
              elsif count = 1 then
              --test # 1
                RAM <= (2 => "00011000", 3 =>"00000111",4 => "00000000",
30 => "00000011", 31 => "00000011", 32 => "00000011", 33 => "00000011",
36 => "00000111", 37 => "00000111", 38 => "00000111", 39 => "00000111",
42 => "00001011", 43 => "00001011", 44 => "00001011", 45 => "00001011",
48 => "00001111", 54 => "00000011", 60 => "00000111", 66 => "00011111",
72 => "00011001", 78 => "00000011", 79 => "00000011", 80 => "00000011",
84 => "00000111", 85 => "00000111", 86 => "00000111", 90 => "00011111",
91 => "00011111", 92 => "00011111", 96 => "00011001", 102 => "00000011",
108 => "00000111", 114 => "00011111", 120 => "00011001", 126 =>
"00000011", 132 => "00000111", 133 => "00000111", 134 => "00000111", 135
=> "00000111", 138 => "00001011", 139 => "00001011", 140 => "00001011",
141 => "00001011", 144 => "00001111", 145 => "00001111", 146 =>
"00001111", 147 => "00001111", others => (others =>'0'));
                count <= 2;

          end if;
          else
             mem_o_data <= RAM(conv_integer(mem_address)) after 1 ns;
          end if;
      end if;
```

```
     end if;
    end if;
   end process;


test : process is
begin
        wait for 100 ns;
        wait for c_CLOCK_PERIOD;
        tb_rst <= '1';
        wait for c_CLOCK_PERIOD;
        tb_rst <= '0';
        wait for c_CLOCK_PERIOD;
        tb_start <= '1';
        wait for c_CLOCK_PERIOD;
        tb_start <= '0';
        wait until tb_done = '1';
        wait until tb_done = '0';
        wait until rising_edge(tb_clk);

        assert RAM(1) = "00000000" report "FAIL high bits" severity
failure;
        assert RAM(0) = "00010101" report "FAIL low bits" severity
failure;

assert false report "Simulation Ended!, test passed" severity failure;
end process test;

end projecttb;
```

Metto anche parti di codice che testano casi particolari come una immagine completamente composta da 0 o una immagine priva di dimensioni. La parte di codice che metto è la sola che cambia rispetto all'esempio completo

```
RAM <= (others => (others =>'0'));
```

```
RAM <= (others => (others =>'1'))
```