Savoia Diego (866981 - 10535515)
Stradi Francesco Emanuele (865754 - 10538326)

PROVA FINALE DI RETI LOGICHE

Anno Accademico 2018/2019

Professor. Gianluca Palermo

INDICE

SPECIFICHE DEL PROGETTO

Si desidera implementare un componente HW descritto in VHDL che, leggendo da memoria le coordinate di 8 centroidi (disposti in uno spazio quadrangolare di dimensioni 256x256) e di un punto specifico, scrive in memoria quale/i centroidi risultino più vicini al suddetto punto (in base alla distanza di Manhattan).
Nello specifico, viene fornita al componente una maschera di 8 bit, che va ad identificare quali centroidi vanno considerati nel calcolo (1 → SI, 0 → NO).

Bit più significativo → centroide n°1
……
bit meno significativo → centroide n°8

Le coordinate di tutti i punti sono a loro volta rappresentate da stringhe di 8 bit (una rappresentante l'ascissa, l'altra rappresentante l'ordinata).
Infine, la maschera di uscita ha una rappresentazione del tutto analoga a quella precedentemente descritta.
Da considerare inoltre i segnali di input:
- **Start** per l'inizio dell'esecuzione
- **Reset** per l'inizializzazione della macchina
- **Clock** per il funzionamento della memoria

Ed i segnali di output:
- **Done** che indica la fine dell'elaborazione
- **En** e **We** per il funzionamento della memoria

Di seguito viene riportata l'interfaccia del componente

```
entity project_reti_logiche is
      port (
            i_clk
            i_start
            i_rst
            i_data
            o_address
            o_done
            o_en
o_we o_data

      );
end project_reti_logiche;
: in  std_logic;
: in  std_logic;
: in  std_logic;
: in  std_logic_vector(7 downto 0);
: out std_logic_vector(15 downto 0);
: out std_logic;
: out std_logic;
: out std_logic;
: out std_logic_vector (7 downto 0)
```

In particolare:

- i_clk è il segnale di CLOCK in ingresso generato dal TestBench;
- i_start è il segnale di START generato dal Test Bench;
- i_rst è il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START;
- 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.

SCELTE PROGETTUALI

- La entity veniva fornita dalla consegna.
- Ci siamo avvalsi di tre librerie:
  - IEEE.STD_LOGIC_1164
  - IEEE.NUMERIC_STD (per le operazioni aritmetiche, in particola la funzione abs)
  - IEEE.STD_LOGIC_UNSIGNED (per la funzione conv_integer)

  Tutte librerie standard.
- Abbiamo optato per un approccio di tipo behavioral, più vicino ad un procedimento algoritmico.

Analizziamo ora l'esecuzione passo-passo del componente:
Abbiamo istanziato un segnale *mask* per la maschera di ingresso, 16 segnali per le coordinate x ed y degli 8 centroidi (*temp1* → *temp16*), 2 segnali per le coordinate del punto specifico (*px* e *py*), ed un segnale *first* utilizzato come contatore.
È stato sufficiente utilizzare un solo processo, nella cui sensitivity list sono presenti i segnali *i_clk* e *i_rst* → CLOCK è legato all'utilizzo della memoria, la quale lavora solo in presenza di un fronte di salita, mentre RESET serve per inizializzare la macchina.
Ci siamo serviti di tre variabili:

- *dist* (INTEGER) che salva la distanza di Manhattan rispetto ad ogni centroide
- *distMin* (INTEGER), inizializzata a 512 (cioè la distanza massima), che memorizza la distanza minima trovata fino ad un determinato momento
- *usc* (VETTORE), inizializzata a 0, la maschera di uscita fino ad un determinato momento.

All'inizio del begin del processo abbiamo inserito tre if:

- un primo controllo per verificare che il reset non sia attivo
- un secondo controllo per verificare che start sia attivo
- un terzo ed ultimo controllo per verificare di essere sul fronte di salita, in modo che la memoria funzioni correttamente.

A questo punto ci sono una serie di if in serie scanditi dal contatore; il processo, accedendo ad uno solo di questi if, esegue il corpo di quello specifico if, prima di terminare quella singola esecuzione del processo. Nelle esecuzioni successive il processo non entrerà più in if a cui ha acceduto precedentemente, a meno che non venga inizializzato nuovamente il reset.
N.B. Questo approccio è stato reso necessario dal fatto che i segnali di un processo vengono aggiornati solo alla fine dell'esecuzione di quest'ultimo.
Nel primo if si inizializzano *o_en* a 1, per utilizzare la memoria, e *o_we* a 0 per leggere dalla suddetta. Si inserisce il primo indirizzo in cui si dovrà leggere e ovviamente (varrà per tutti gli if) viene incrementato il contatore.
Successivamente si legge prima la maschera di ingresso, poi le coordinate dei centroidi e quelle del punto specifico, salvandole nei rispettivi segnali, ogni volta inserendo l'indirizzo successivo rispetto al valore di *o_address* precedente.
Quando first arriva al valore *'10101'* viene calcolata la distanza di Manhattan da ciascun centroide da considerare (bit=1). Nello specifico, vi è un if per ogni bit della maschera d'ingresso: se questo è 1 la sua distanza viene paragonata alla più piccola trovata fino a quel momento; qualora sia minore viene aggiornata la maschera di uscita mettendo ad 1 il bit corrispondente ed i restanti a 0, poiché quel centroide sarà il più vicino al punto tra quelli analizzati; qualora siano uguali, i restanti bit verrano mantenuti come in precedenza, poiché ci sono più centroidi alla stessa distanza.
Infine viene scritta in memoria la maschera d'uscita e con l'if seguente *o_done* viene posto ad 1.
I due if finali servono per inizializzare il componente → start=0 implica che *o_done* possa tornare a 0, mentre reset=1 implica che la macchina venga resettata, ovvero che il contatore venga azzerato, come anche i segnali di memoria *o_en* e *o_we*.

CASI DI TEST (TEST BENCH)

Oltre al test bench di esempio, che rappresenta un caso comune, abbiamo implementato 3 test bench rappresentanti casi limite:

1) Maschera d'ingresso a tutti 0 → maschera d'uscita a tutti 0

```
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 := 100 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);

-- come da esempio su specifica
signal RAM: ram_type := (0 => std_logic_vector(to_unsigned( 0 , 8)),
                         1 => std_logic_vector(to_unsigned( 75 , 8)),
                         2 => std_logic_vector(to_unsigned( 32 , 8)),
                         3 => std_logic_vector(to_unsigned( 111 , 8)),
                         4 => std_logic_vector(to_unsigned( 213 , 8)),
                         5 => std_logic_vector(to_unsigned( 79 , 8)),
                         6 => std_logic_vector(to_unsigned( 33 , 8)),
                         7 => std_logic_vector(to_unsigned( 1 , 8)),
                         8 => std_logic_vector(to_unsigned( 33 , 8)),
                         9 => std_logic_vector(to_unsigned( 80 , 8)),
                         10 => std_logic_vector(to_unsigned( 35 , 8)),
                         11 => std_logic_vector(to_unsigned( 12 , 8)),
                         12 => std_logic_vector(to_unsigned( 254 , 8)),
                         13 => std_logic_vector(to_unsigned( 215 , 8)),
                         14 => std_logic_vector(to_unsigned( 78 , 8)),
                         15 => std_logic_vector(to_unsigned( 211 , 8)),
                         16 => std_logic_vector(to_unsigned( 121 , 8)),
                         17 => std_logic_vector(to_unsigned( 78 , 8)),
                         18 => std_logic_vector(to_unsigned( 33 , 8)),
                                                 others => (others =>'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);
    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 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 2 ns;
         else
            mem_o_data <= RAM(conv_integer(mem_address)) after 2 ns;
```

```
          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;
   wait until tb_done  =  '1';
   wait for c_CLOCK_PERIOD;
   tb_start <= '0';
   wait until tb_done = '0';

   -- Maschera di output = 00000000
   assert RAM(19) = std_logic_vector(to_unsigned( 0 , 8)) report "TEST FALLITO" severity failure;

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

end projecttb;
```

2) Tutti i punti coincidenti tra loro (con maschera d'ingresso a tutti 1) → maschera d'uscita a tutti 1

```
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 := 100 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);

-- come da esempio su specifica
signal RAM: ram_type := (0 => std_logic_vector(to_unsigned( 255 , 8)),
                         1 => std_logic_vector(to_unsigned( 78 , 8)),
                         2 => std_logic_vector(to_unsigned( 33 , 8)),
                         3 => std_logic_vector(to_unsigned( 78 , 8)),
                         4 => std_logic_vector(to_unsigned( 33 , 8)),
                         5 => std_logic_vector(to_unsigned( 78 , 8)),
                         6 => std_logic_vector(to_unsigned( 33 , 8)),
                         7 => std_logic_vector(to_unsigned( 78 , 8)),
                         8 => std_logic_vector(to_unsigned( 33 , 8)),
                         9 => std_logic_vector(to_unsigned( 78 , 8)),
                         10 => std_logic_vector(to_unsigned( 33 , 8)),
                         11 => std_logic_vector(to_unsigned( 78 , 8)),
                         12 => std_logic_vector(to_unsigned( 33 , 8)),
                         13 => std_logic_vector(to_unsigned( 78 , 8)),
                         14 => std_logic_vector(to_unsigned( 33 , 8)),
                         15 => std_logic_vector(to_unsigned( 78 , 8)),
                         16 => std_logic_vector(to_unsigned( 33 , 8)),
                         17 => std_logic_vector(to_unsigned( 78 , 8)),
                         18 => std_logic_vector(to_unsigned( 33 , 8)),
                                                 others => (others =>'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);
     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 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 2 ns;
         else
            mem_o_data <= RAM(conv_integer(mem_address)) after 2 ns;
         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;
   wait until tb_done  =  '1';
   wait for c_CLOCK_PERIOD;
   tb_start <= '0';
   wait until tb_done = '0';

   -- Maschera di output = 11111111
   assert RAM(19) = std_logic_vector(to_unsigned( 255 , 8)) report "TEST FALLITO" severity failure;

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

end projecttb;
```

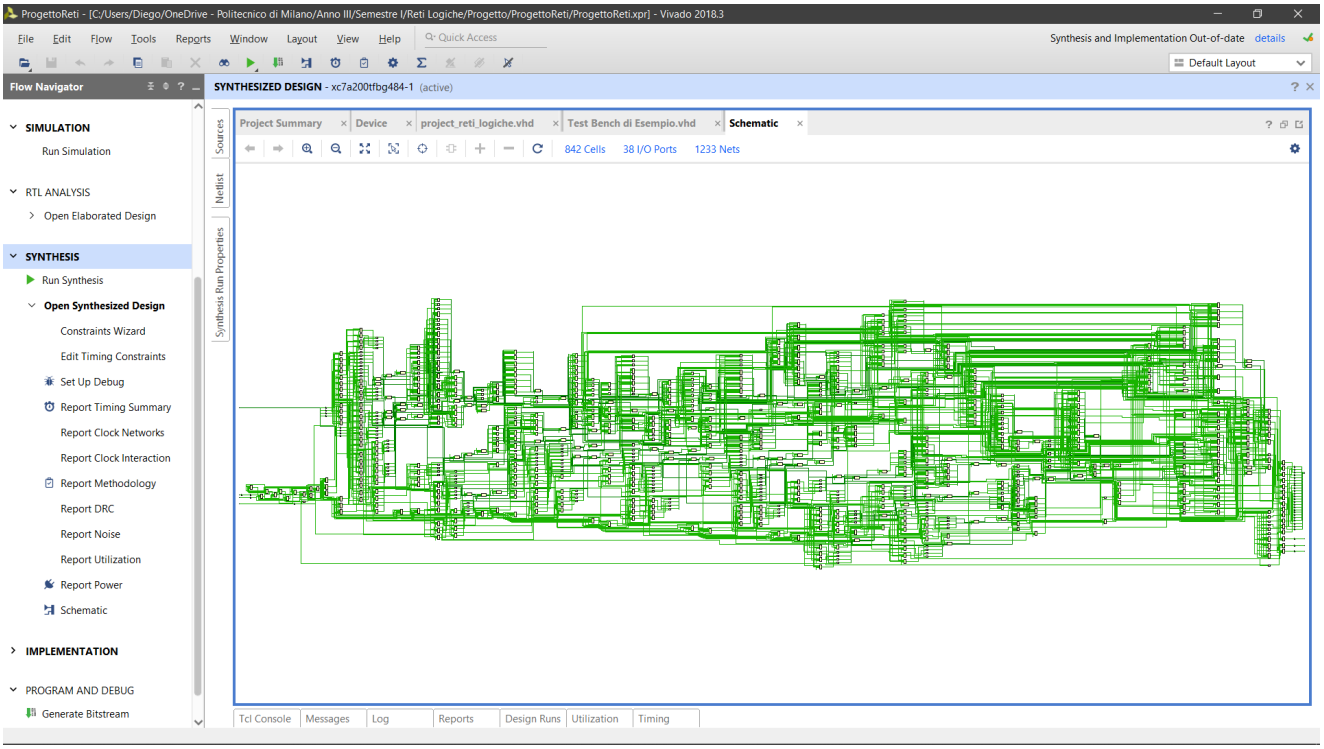
3) Distanza maggiore rappresentabile = 512 → Maschera d'uscita 00000001 a fronte di una equivalente maschera d'ingresso

```
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 := 100 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);

-- come da esempio su specifica
signal RAM: ram_type := (0 => std_logic_vector(to_unsigned( 1 , 8)),
                         1 => std_logic_vector(to_unsigned( 255 , 8)),
                         2 => std_logic_vector(to_unsigned( 255 , 8)),
                         3 => std_logic_vector(to_unsigned( 111 , 8)),
                         4 => std_logic_vector(to_unsigned( 213 , 8)),
                         5 => std_logic_vector(to_unsigned( 79 , 8)),
                         6 => std_logic_vector(to_unsigned( 33 , 8)),
                         7 => std_logic_vector(to_unsigned( 1 , 8)),
                         8 => std_logic_vector(to_unsigned( 33 , 8)),
                         9 => std_logic_vector(to_unsigned( 80 , 8)),
                         10 => std_logic_vector(to_unsigned( 35 , 8)),
                         11 => std_logic_vector(to_unsigned( 12 , 8)),
                         12 => std_logic_vector(to_unsigned( 254 , 8)),
                         13 => std_logic_vector(to_unsigned( 215 , 8)),
                         14 => std_logic_vector(to_unsigned( 78 , 8)),
                         15 => std_logic_vector(to_unsigned( 211 , 8)),
                         16 => std_logic_vector(to_unsigned( 121 , 8)),
                         17 => std_logic_vector(to_unsigned( 0 , 8)),
                         18 => std_logic_vector(to_unsigned( 0 , 8)),
                                                 others => (others =>'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);
      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 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 2 ns;
            else
                mem_o_data <= RAM(conv_integer(mem_address)) after 2 ns;
            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;
    wait until tb_done  =  '1';
    wait for c_CLOCK_PERIOD;
    tb_start <= '0';
    wait until tb_done = '0';

    -- Maschera di output = 00000001
    assert RAM(19) = std_logic_vector(to_unsigned( 1 , 8)) report "TEST FALLITO" severity failure;

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

end projecttb;
```

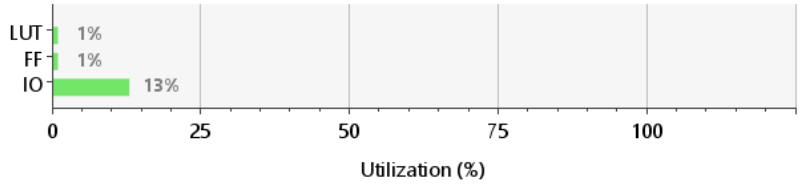
RISULTATI DELLA SINTESI

842 Cells **38 I/O Ports** **1233 Nets**



Summary

Resource	Utilization	Available	Utilization %
LUT	485	134600	0.36
FF	181	269200	0.07
IO	38	285	13.33



Post-sintesi functional simulation → **TEST PASSATO medesimo comportamento (output)**

Posto-sintesi timing simulation → **TEST PASSATO medesimo comportamento (output)**