Reti Logiche
Progetto di Prova Finale

Gruppo composto da:
Valeria Maria Fortina -10537962
Alessio Galluccio - 10568346

Indice

1. Introduzione al problema

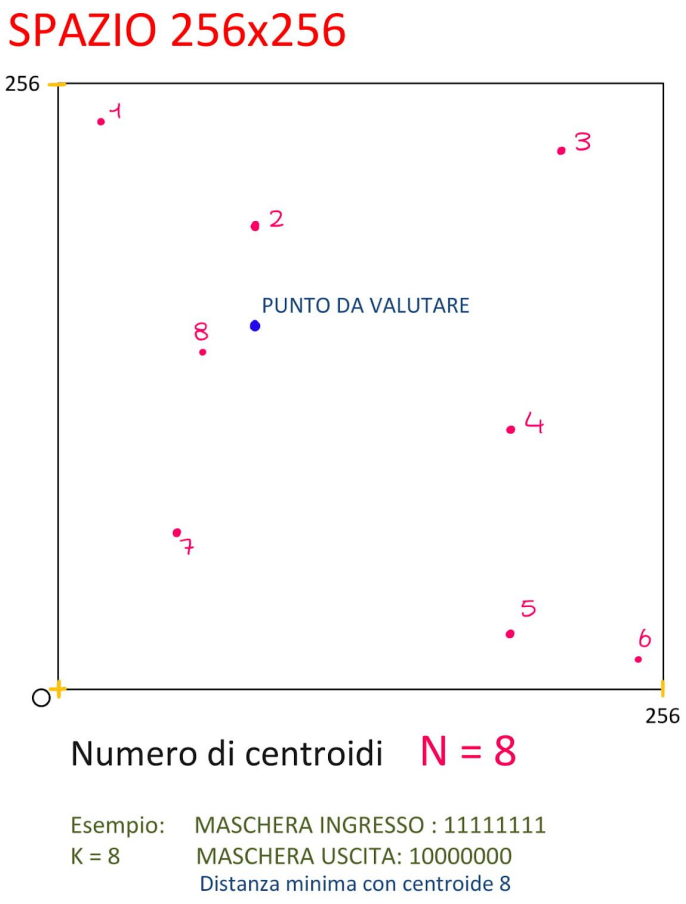
Dato uno spazio bidimensionale e date le posizioni di N punti, detti “centroidi” e di un ulteriore punto da valutare, l’obiettivo del progetto è l’implementazione di un componente HW descritto in VHDL che sia in grado di trovare quali centroidi siano più vicini al punto da valutare in termini di distanza di Manhattan.

Per il progetto, si è considerato che il numero di centroidi fosse pari a N=8. Degli N centroidi, K<=N sono quelli su cui calcolare la distanza dal punto dato. I K centroidi sono indicati da una maschera di ingresso a N bit. Ogni centroide corrisponde a un preciso bit secondo un preciso ordine, il centroide 1 al bit meno significativo e il centroide 8 al più significativo. Se il bit è a 1, significa che il centroide corrispondente è valido (punto dal quale calcolare la distanza) mentre il bit a 0 indica che il centroide non deve essere esaminato.
La maschera di uscita è sempre composta da N bit, dove il bit a 1 indica che il centroide corrispondente è considerato tra quelli a distanza minima.
Le maschere e le coordinate dei punti sono memorizzati in una memoria con indirizzamento al Byte partendo dalla posizione 0 e sono di lunghezza 8 bit. La maschera di uscita deve essere scritta nella posizione corrispondente della RAM.

INDIRIZZO DI MEMORIA	COMMENTO
0	Maschera di ingresso
1	X centroide 1
2	Y centroide 1
3	X centroide 2
4	Y centroide 2
5	X centroide 3
6	Y centroide 3
7	X centroide 4
8	Y centroide 4
9	X centroide 5
10	Y centroide 5
11	X centroide 6
12	Y centroide 6

13	X centroide 7
14	Y centroide 7
15	X centroide 8
16	Y centroide 8
17	X del punto da valutare
18	Y del punto da valutare
19	Maschera di uscita

L'elaborazione inizia quando un segnale START è portato a 1. Esso rimarrà alto fino a quando il segnale DONE non sarà portato alto. Al termine della computazione, il modulo da progettare deve alzare il segnale DONE, che notifica la fine dell'elaborazione. Esso rimane alto fino a quando il segnale START non è riportato a 0. Un nuovo segnale START non può essere dato fino a quando DONE non è stato riportato a 0.



Per completare si riportano qui di seguito gli input e output primari del componente:

```
entity 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 project_reti_logiche;
```

• 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.

Per ulteriori dettagli si rimanda alla specifica ufficiale.

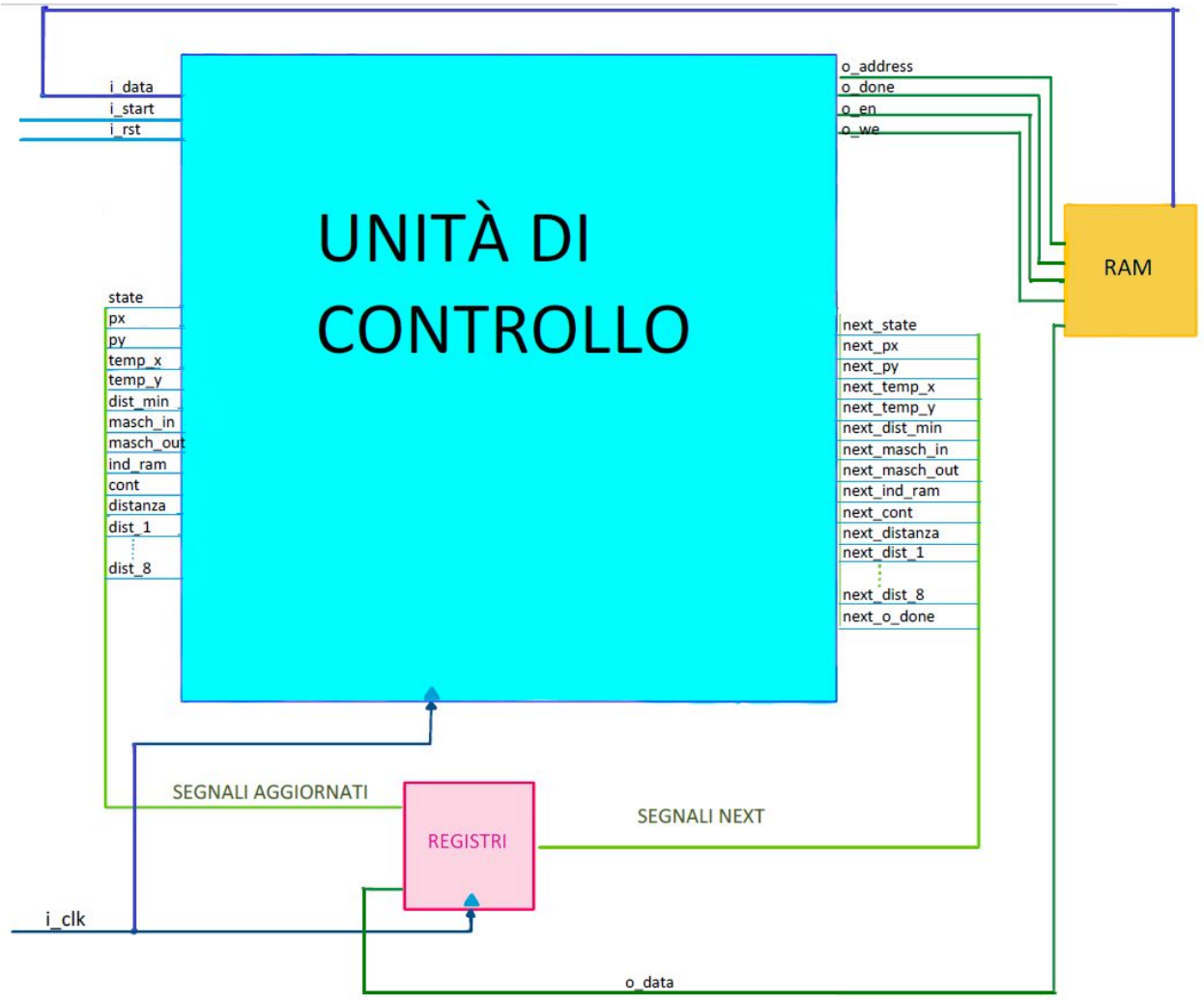
Ipotesi aggiuntive

Per la realizzazione del componente si sono fatte delle ipotesi aggiuntive alle specifiche:

1. Nel caso in cui la maschera di ingresso sia tutta a zero, la maschera di uscita sarà tutta a zero.
2. Dopo che il segnale di start è stato portato a 0, esso può essere riportato a 1 senza che il componente sia prima resettato.

2. Architettura

L’architettura realizzata si basa su un unico modulo che realizza una macchina a stati, escludendo il modulo della RAM fornito dal test bench. Il modulo utilizza al suo interno due processi sincronizzati.
Il processo ”state_reg_update” è sincronizzato al fronte di salita del clock. Esso si occupa di aggiornare i segnali che rappresentano i registri della macchina. Si è scelto di sincronizzare questo processo al fronte di salita, perché anche il modulo della RAM aggiorna i propri segnali alla salita del clock. In questo modo, la macchina aggiorna coerentemente tutti i propri segnali.
Il processo “main”, invece, implementa la macchina a stati vera e propria. Esso è sincronizzato al fronte di discesa del clock. Poiché il clock è relativamente ampio, si ipotizza che il tempo di transizione del valore dei segnali sia molto minore della metà del tempo di clock. Di conseguenza, al fronte di discesa del clock, il processo “main” può leggere i valori corretti dei segnali e decidere come essi verranno aggiornati al fronte di salita del clock successivo. Le simulazioni hanno confermato questa ipotesi come corretta. Si è quindi deciso di realizzare l’architettura in questo modo.
Il processo “main”, inoltre, si occupa di gestire il segnale di input reset. Quando questo segnale è portato a 1, la macchina torna allo stato di RESET al ciclo di clock successivo, qualunque sia lo stato corrente della macchina.
Di seguito, la macchina a stati è analizzata in dettaglio e si mostra uno schema funzionale del modulo.



Per semplicità, si è rappresentata una sola linea entrante ed uscente dai registri, quando invece rappresenta l'intero insieme dei segnali.

Modello di FSM e Algoritmo

La macchina a stati realizzata utilizza al suo interno 14 stati, che possono essere suddivisi nelle seguenti fasi:

```
type state_type is (

--stato di avviamento
RESET,

--stati di copiatura maschera di ingresso
READ_MASCH,COPY_MASCH,

--stato di modifica indirizzo memoria
ADD_IND,

--stati di lettura coordinate
READ_X,COPY_X, READ_Y,COPY_Y,

--stati calcolo distanze
CALC_DIST, INSERISCI,

--stati di ricerca distanza minima
TROVA_MIN, CONFRONTA,

--stati scrittura risultato e conclusione
WRITE_MASCH, FINE);
```

RESET

Lo stato in cui la macchina inizia la sua computazione è RESET. Esso si occupa di inizializzare a 0 i valori dei segnali di lettura e scrittura dalla RAM, in modo che non avvengano comportamenti anomali, e di aspettare la salita del segnale di start. Quando il segnale di start è a 1, la macchina inizializza il valore di next_ind_ram al primo indirizzo che dovrà essere letto (indirizzo zero) e passa allo stato READ_MASCH. Se il segnale di start è a 0, invece, la macchina rimane nello stato RESET ciclando su se stesso.
Si può ritornare in questo stato in due modi dopo l'inizio della computazione: alla fine della computazione, quando il segnale o_done è riportato a 0, o quando il segnale di reset è portato a 1.

READ_MASCH e COPY_MASCH

In questi due stati la macchina si occupa di leggere la maschera di ingresso dalla RAM e di salvarne il valore in un segnale apposito. Utilizziamo due stati per avere la garanzia di leggere e scrivere i valori corretti. Nello stato READ_MASCH, la macchina si occupa di

impostare in modo corretto i segnali o_address, o_en e o_we, in modo da poter leggere dal corretto indirizzo di memoria. La macchina passa al clock successivo allo stato COPY_MASCH, nel quale viene copiato il segnale di ingresso i_data proveniente dalla RAM, e contenente il valore della maschera, nel segnale apposito. La macchina passa poi allo stato ADD_IND.

ADD_IND

Lo stato ADD_IND si occupa di modificare il segnale interno ind_ram, impostando il segnale next_ind_ram. Questo stato assume diversi comportamenti a seconda del valore che il segnale ind_ram assume quando la macchia arriva in questo stato.

1) Nel caso in cui il suo valore sia pari all'indirizzo della macchina di ingresso (0), significa che la macchina ha appena letto la maschera di ingresso. E' necessario che la prossima azione della macchina sia leggere e salvare il valore del punto da studiare. Il segnale next_ind_ram viene quindi aggiornato con il valore dell'indirizzo della coordinata x del punto da studiare (17).
2) Nel caso in cui il suo valore sia pari all'indirizzo del punto da studiare, significa che la macchina dovrà iniziare a leggere le coordinate dei vari centroidi. next_ind_ram viene aggiornato con il valore dell'indirizzo della coordinata x del primo centroide (1).
3) Se non fa parte dei primi due casi, significa che si è nel mezzo della lettura delle coordinate dei centroidi. Il next_ind_ram è quindi aggiornato con il valore di ind_ram aumentato di 2, in modo che indichi la cella in cui è salvata la coordinata x del prossimo centroide. Non salviamo mai in questo registro gli indirizzi delle coordinate y, perché basta aggiungere 1 all'indirizzo della coordinata x per raggiungerli, come viene fatto nello stato READ_Y. Non si pone il controllo di aver fatto o meno la lettura dell'ultimo centroide, perché ciò viene gestito nello stato CALC_DIST.

READ_X,COPY_X, READ_Y e COPY_Y

Il principio di funzionamento di questi stati è analogo a quello di READ_MASCH e COPY_MASCH. Lo stato READ imposta i segnali o_address, o_en e o_we e lo stato COPY copia il segnale di ingresso i_data proveniente dalla RAM nel segnale apposito: next_px e next_py per il punto da studiare o next_temp_x e next_temp_y per i centroidi. In particolare, lo stato READ_X utilizza il valore del segnale interno ind_ram per leggere nella RAM, mentre lo stato READ_Y utilizza il valore di ind_ram aumentato di 1, perché la coordinata y di un punto è immediatamente successiva a quella x nella memoria. Dopo lo stato COPY_Y, la macchina può andare in due stati diversi:

1) CALC_DIST se le coordinate sono di un centroide.
2) ADD_IND se le coordinate sono del punto da studiare.

CALC_DIST e INSERISCI

Lo stato CALC_DIST si occupa di calcolare le distanze di Manhattan dei centroidi dal punto da studiare. INSERISCI si occupa di salvare tale valore nel segnale apposito (next_dist_i, con i da 1 a 8). Lo stato CALC_DIST, in particolare, prima di svolgere il calcolo della distanza, controlla attraverso il valore della maschera di ingresso se il punto appartenga effettivamente al problema. Se il suo bit corrispondente è a 0, salva nel segnale della

distanza corrispondente il valore “1111111111”. Esso rappresenta in questo circuito la *distanza infinita*. Ciò servirà per i successivi stati.

TROVA_MIN e CONFRONTA

I due stati TROVA_MIN e CONFRONTA si occupano di ricercare la distanza minima.
TROVA_MIN, grazie all’uso di un contatore, compie una serie di cicli su se stesso, confrontando ogni volta il segnale della distanza minima dist_min con quello della distanza punto-centroide dist_i (con i da 1 a 8).
Ad ogni ciclo la macchina copia il valore della distanza dist_i se questa risulta minore di dist_min nel segnale next_dist_min, che aggiornerà dist_min alla salita del clock. In questo modo, dist_min risulterà aggiornato a ogni ciclo compiuto su TROVA_MIN.
Quando avviene il passaggio allo stato CONFRONTA, il segnale dist_min contiene la distanza minima.
Lo stato CONFRONTA compie un controllo iniziale: se dist_min contiene il valore “1111111111”, ovvero il valore della distanza infinita, allora compie un salto allo stato WRITE_MASCH, poiché significa che la maschera di ingresso è composta da soli zeri.
Altrimenti, grazie sempre ad un contatore, compie una serie di cicli su se stesso, confrontando ogni volta il segnale della distanza minima con quello della distanza punto-centroide e, nel caso di uguaglianza tra i valori dei segnali, somma il segnale masch_out e una potenza di due adeguata all centroide considerato, in modo da avere un 1 nella posizione corrispondente al centroide nella maschera di uscita.
Per esempio, se il centroide considerato è il quarto, il codice sarà:

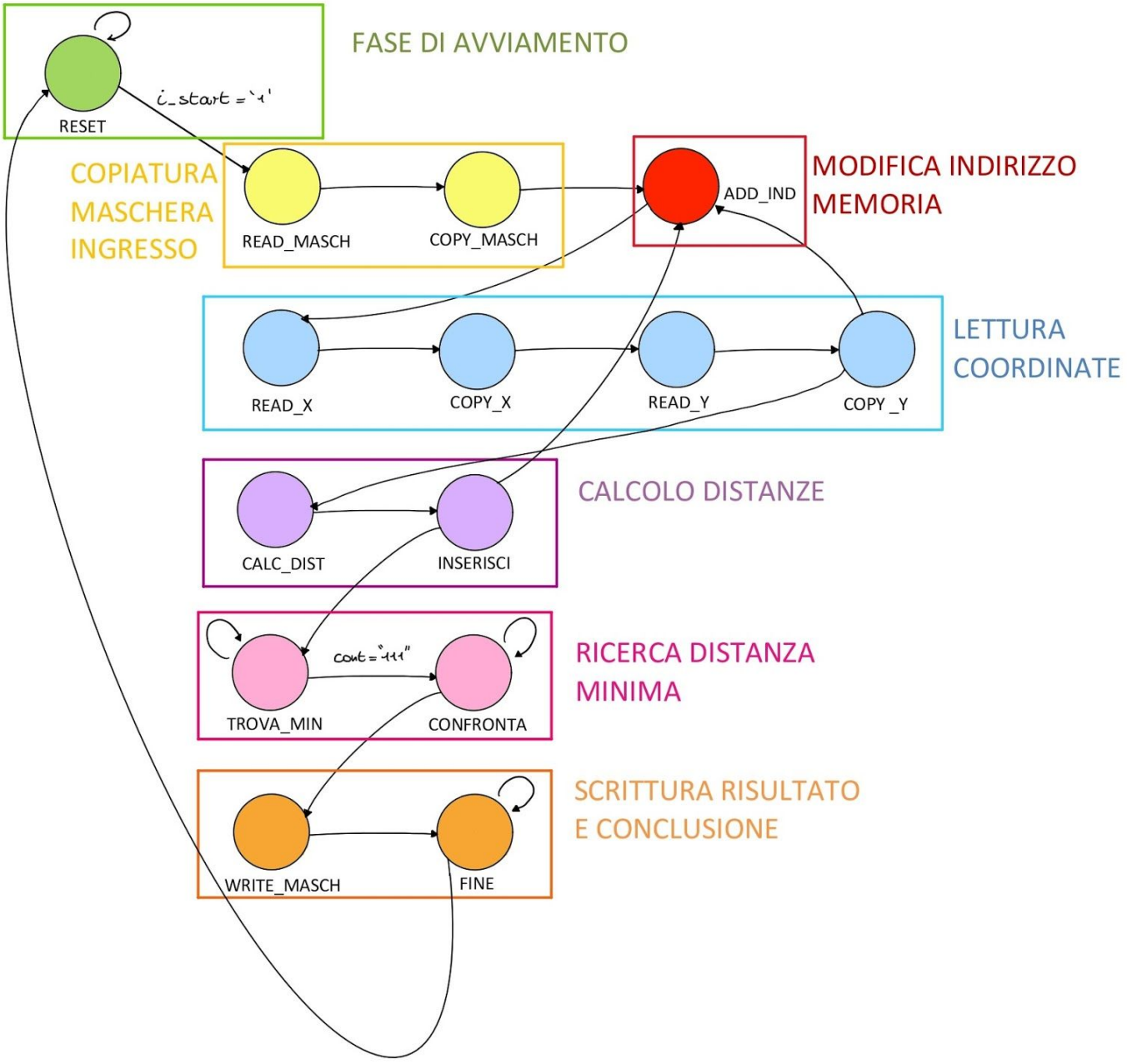
```
if (dist_min = dist_4) then
     next_masch_out <= std_logic_vector( unsigned (masch_out) + 8);
```

dove in questo caso abbiamo la somma tra masch_out e 2^3.

La somma viene salvata nel segnale next_masch_out che aggiornerà masch_out alla salita del clock. In questo modo al ciclo successivo di CONFRONTA masch_out risulta aggiornato.

WRITE_MASCH, FINE

Lo stato WRITE_MASCH si occupa di impostare i segnali o_address, o_en e o_we per copiare il valore della maschera d’uscita nell’indirizzo apposito della RAM (19). Lo stato FINE pone il segnale o_done a 1 e si pone in attesa ciclando su se stesso. Quando il segnale di input i_start è portato a 0, pone il segnale o_done a 0 e ritorna allo stato RESET, pronto per una prossima computazione.


FASE DI AVVIAMENTO
RESET
i_start = '1'
COPIATURA MASCHERA INGRESSO
READ_MASCH
COPY_MASCH
ADD_IND
MODIFICA INDIRIZZO MEMORIA
READ_X
COPY_X
READ_Y
COPY _Y
LETTURA COORDINATE
CALC_DIST
INSERISCI
CALCOLO DISTANZE
cont = "111"
TROVA_MIN
CONFRONTA
RICERCA DISTANZA MINIMA
WRITE_MASCH
FINE
SCRITTURA RISULTATO E CONCLUSIONE

SEGNALI

Per memorizzare i valori ottenuti nelle varie fasi della computazione sono utilizzati dei segnali interni. Ogni segnale è affiancato da un segnale next. La macchina a stati, come descritto dettagliatamente in precedenza, modifica i segnali next, i quali a loro volta aggiornano i segnali al fronta di salita del clock durante il process `state_reg_update` riportato di seguito.

```
begin

-- processo di update dei segnali al fronte di salita del clock
state_reg_update: process(i_clk)
begin
    if rising_edge(i_clk) then
          state <= next_state;
          dist_1 <= next_dist_1;
          dist_2 <= next_dist_2;
          dist_3 <= next_dist_3;
          dist_4 <= next_dist_4;
          dist_5 <= next_dist_5;
          dist_6 <= next_dist_6;
          dist_7 <= next_dist_7;
          dist_8 <= next_dist_8;
          px <= next_px;
          py <= next_py;
          temp_x <= next_temp_x;
          temp_y <= next_temp_y;
          dist_min <= next_dist_min;
          masch_in <= next_masch_in;
          masch_out <= next_masch_out;
          ind_ram <= next_ind_ram;
          cont <= next_cont;
          o_done <= next_o_done;
          distanza <= next_distanza;
    end if;
end process;
```

Segnali per la memorizzazione delle distanze

I valori delle distanze centroide-punto da studiare sono copiati all'interno di segnali dist_i dove i rappresenta un numero da 1 a 8, ovvero il corrispondente centroide. Tali segnali sono a 10 bit in modo da memorizzare fino alla *distanza infinita* "1111111111".
Nel codice tali segnali sono tutti inizializzati a tale distanza infinita.
Inoltre sono presenti due ulteriori segnali `distanza` e `next_distanza` utilizzati nello stato CALC_DIST per salvare il valore della distanza calcolata.

```
signal dist_1, next_dist_1 : std_logic_vector(9 downto 0) := "1111111111";
signal dist_2, next_dist_2 : std_logic_vector(9 downto 0) := "1111111111";
signal dist_3, next_dist_3 : std_logic_vector(9 downto 0) := "1111111111";
signal dist_4, next_dist_4 : std_logic_vector(9 downto 0) := "1111111111";
signal dist_5, next_dist_5 : std_logic_vector(9 downto 0) := "1111111111";
signal dist_6, next_dist_6 : std_logic_vector(9 downto 0) := "1111111111";
signal dist_7, next_dist_7 : std_logic_vector(9 downto 0) := "1111111111";
signal dist_8, next_dist_8 : std_logic_vector(9 downto 0) := "1111111111";

signal distanza, next_distanza :std_logic_vector(9 downto 0) :=
"1111111111";
```

Segnali per coordinate punto x e y

Segnali in cui vengono copiati valori delle coordinate x e y del punto di riferimento, composti da 8 bit tutti inizializzati a zero.

```
signal px, next_px : std_logic_vector(7 downto 0) := "00000000";
signal py, next_py : std_logic_vector(7 downto 0) := "00000000";
```

Segnali per valori temporali di x e y

Segnali in cui vengono copiati valori delle coordinate x e y del centroide preso in considerazione, composti da 8 bit tutti inizializzati a zero.

```
signal temp_x, next_temp_x : std_logic_vector(7 downto 0) := "00000000";
```

```
signal temp_y, next_temp_y : std_logic_vector(7 downto 0):= "00000000";
```

Segnale per la distanza minima

Segnali che contengono il valore della distanza minima, composti da 10 bit, inizializzati tutti a zero. Utilizzati durante la computazione per confrontare quali distanze sono uguali alla minima.

```
signal dist_min, next_dist_min :std_logic_vector(9 downto 0) :=
"0000000000";
```

Segnale maschera input

Segnali che memorizzano il contenuto della maschera di input, formati da 8 bit, tutti inizializzati a zero.

```
signal masch_in, next_masch_in :std_logic_vector(7 downto 0):= "00000000";
```

Segnale maschera output

Segnali che memorizzano il contenuto della maschera di output, formati da 8 bit, tutti inizializzati a zero. Alla fine della computazione il valore di `masch_out` viene copiato nel segnale di uscita.

```
signal masch_out, next_masch_out :std_logic_vector(7 downto 0):=
"00000000";
```

Segnale indirizzo RAM

Segnali che indicano il prossimo indirizzo di memoria da leggere, se non è una coordinata di un punto, o l'indirizzo della coordinata x del prossimo punto da leggere. Sono inizializzati a zero.

```
signal ind_ram, next_ind_ram :std_logic_vector(15 downto 0) :=
"0000000000000000";
```

Contatore

Segnali usati come contatore, utilizzati per compiere dei cicli su alcuni stati della macchina, come TROVA_MIN e CONFRONTA. Composti da 3 bit inizializzati a zero.

```
signal cont, next_cont : std_logic_vector(2 downto 0) := "000";
```

Stati

Segnali utilizzati per passare da uno stato all'altro della macchina, sono inizializzati al primo stato RESET.

```
signal state : state_type := RESET;
signal next_state : state_type := RESET;
```

Segnale next_o_done

Segnale utilizzato per aggiornare il valore di o_done al fronte di salita del clock. Inizialmente è posto a zero.

```
signal next_o_done: std_logic := '0';
```

3. Risultati Sperimentali

1. Sintesi

Si riporta in seguito il report di sintesi:

```
#-----------------------------------------------------------
# Vivado v2018.3 (64-bit)
# SW Build 2405991 on Thu Dec  6 23:38:27 MST 2018
# IP Build 2404404 on Fri Dec  7 01:43:56 MST 2018
# Start of session at: Sat Mar 30 12:06:14 2019
# Process ID: 6764
# Current directory: C:/Users/Valeria/project_reti_logiche/project_reti_logiche.runs/synth_1
# Command line: vivado.exe -log project_reti_logiche.vds -product Vivado -mode batch -messageDb vivado.pb -notrace -source project_reti_logiche.tcl
# Log file: C:/Users/Valeria/project_reti_logiche/project_reti_logiche.runs/synth_1/project_reti_logiche.vds
# Journal file: C:/Users/Valeria/project_reti_logiche/project_reti_logiche.runs/synth_1\vivado.jou
#-----------------------------------------------------------
source project_reti_logiche.tcl -notrace
Command: synth_design -top project_reti_logiche -part xc7a200tfbg484-1
Starting synth_design
Attempting to get a license for feature 'Synthesis' and/or device 'xc7a200t'
INFO: [Common 17-349] Got license for feature 'Synthesis' and/or device 'xc7a200t'
INFO: Launching helper process for spawning children vivado processes
INFO: Helper process launched with PID 10192
---------------------------------------------------------------------------------
Starting Synthesize : Time (s): cpu = 00:00:01 ; elapsed = 00:00:02 . Memory (MB): peak = 363.418 ; gain = 101.191
---------------------------------------------------------------------------------
INFO: [Synth 8-638] synthesizing module 'project_reti_logiche' [C:/Users/Valeria/Desktop/10537962_10568346.vhd:46]
INFO: [Synth 8-256] done synthesizing module 'project_reti_logiche' (1#1) [C:/Users/Valeria/Desktop/10537962_10568346.vhd:46]
---------------------------------------------------------------------------------
Finished Synthesize : Time (s): cpu = 00:00:02 ; elapsed = 00:00:03 . Memory (MB): peak = 419.195 ; gain = 156.969
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
Finished Constraint Validation : Time (s): cpu = 00:00:02 ; elapsed = 00:00:03 . Memory (MB): peak = 419.195 ; gain = 156.969
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
Start Loading Part and Timing Information
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
Loading part: xc7a200tfbg484-1
```

```
---------------------------------------------------------------------------
Finished Loading Part and Timing Information : Time (s): cpu = 00:00:02 ; elapsed = 00:00:03 . Memory (MB): peak = 419.195 ; gain = 156.969
INFO: [Device 21-403] Loading part xc7a200tfbg484-1
---------------------------------------------------------------------------
INFO: [Synth 8-5818] HDL ADVISOR - The operator resource <adder> is shared. To prevent sharing consider applying a KEEP on the output of the operator [C:/Users/Valeria/Desktop/10537962_10568346.vhd:234]
INFO: [Synth 8-5818] HDL ADVISOR - The operator resource <adder> is shared. To prevent sharing consider applying a KEEP on the output of the operator [C:/Users/Valeria/Desktop/10537962_10568346.vhd:129]
INFO: [Synth 8-5818] HDL ADVISOR - The operator resource <adder> is shared. To prevent sharing consider applying a KEEP on the output of the operator [C:/Users/Valeria/Desktop/10537962_10568346.vhd:129]
INFO: [Synth 8-5818] HDL ADVISOR - The operator resource <adder> is shared. To prevent sharing consider applying a KEEP on the output of the operator [C:/Users/Valeria/Desktop/10537962_10568346.vhd:129]
INFO: [Synth 8-802] inferred FSM for state register 'state_reg' in module 'project_reti_logiche'
INFO: [Synth 8-5546] ROM "next_px" won't be mapped to RAM because it is too sparse
INFO: [Synth 8-5546] ROM "next_temp_x" won't be mapped to RAM because it is too sparse
INFO: [Synth 8-5546] ROM "next_cont" won't be mapped to RAM because it is too sparse
INFO: [Synth 8-5546] ROM "next_cont" won't be mapped to RAM because it is too sparse
INFO: [Synth 8-5546] ROM "next_state" won't be mapped to RAM because it is too sparse
INFO: [Synth 8-5546] ROM "next_state" won't be mapped to RAM because it is too sparse
INFO: [Synth 8-5544] ROM "next_state" won't be mapped to Block RAM because address size (1) smaller than threshold (5)
INFO: [Synth 8-5546] ROM "next_state" won't be mapped to RAM because it is too sparse
---------------------------------------------------------------------------
State | New Encoding | Previous Encoding
---------------------------------------------------------------------------
reset | 0000 | 0000
read_masch | 0001 |0001
copy_masch |0010 |  0010
add_ind |  0011 |   0011
read_x | 0100 |  0100
copy_x | 0101 | 0101
read_y | 0110 | 0110
copy_y |0111 | 0111
calc_dist |  1000 |  1000
inserisci |1001 | 1001
trova_min | 1010 |  1010
confronta | 1011 | 1011
write_masch |1100 | 1100
fine | 1101 |  1101
---------------------------------------------------------------------------
INFO: [Synth 8-3354] encoded FSM with state register 'state_reg' using encoding 'sequential' in module 'project_reti_logiche'
---------------------------------------------------------------------------
Finished RTL Optimization Phase 2 : Time (s): cpu = 00:00:03 ; elapsed = 00:00:03 . Memory (MB): peak = 445.508 ; gain = 183.281
```

```
---------------------------------------------------------
-----------------------

Report RTL Partitions:
+-+--------------+------------+----------+
| |RTL Partition |Replication |Instances |
+-+--------------+------------+----------+
+-+--------------+------------+----------+
No constraint files found.
---------------------------------------------------------
-----------------------
Start RTL Component Statistics
---------------------------------------------------------
-----------------------
Detailed RTL Component Info :
+---Adders :
            2 Input     16 Bit       Adders := 2
            5 Input     10 Bit       Adders := 1
            2 Input      8 Bit       Adders := 2
            2 Input      3 Bit       Adders := 1
+---Registers :
                  16 Bit    Registers := 3
                  10 Bit    Registers := 20
                   8 Bit    Registers := 13
                   3 Bit    Registers := 2
                   1 Bit    Registers := 4
+---Muxes :
            3 Input     16 Bit       Muxes := 1
           14 Input     16 Bit       Muxes := 2
            2 Input     10 Bit       Muxes := 11
            2 Input      8 Bit       Muxes := 4
            7 Input      8 Bit       Muxes := 1
           14 Input      8 Bit       Muxes := 2
           14 Input      4 Bit       Muxes := 2
            2 Input      4 Bit       Muxes := 5
            8 Input      4 Bit       Muxes := 2
            8 Input      3 Bit       Muxes := 1
           14 Input      3 Bit       Muxes := 1
            2 Input      1 Bit       Muxes := 24
            9 Input      1 Bit       Muxes := 10
           14 Input      1 Bit       Muxes := 27
---------------------------------------------------------
-----------------------
Finished RTL Component Statistics
---------------------------------------------------------
-----------------------
---------------------------------------------------------
-----------------------
Start RTL Hierarchical Component
Statistics
---------------------------------------------------------
-----------------------
Hierarchical RTL Component report
Module project_reti_logiche
Detailed RTL Component Info :
+---Adders :
            2 Input     16 Bit       Adders := 2
            5 Input     10 Bit       Adders := 1
            2 Input      8 Bit       Adders := 2
            2 Input      3 Bit       Adders := 1
+---Registers :
                  16 Bit    Registers := 3
                  10 Bit    Registers := 20
                   8 Bit    Registers := 13
                   3 Bit    Registers := 2
                   1 Bit    Registers := 4
+---Muxes :
            3 Input     16 Bit       Muxes := 1
           14 Input     16 Bit       Muxes := 2
            2 Input     10 Bit       Muxes := 11
            2 Input      8 Bit       Muxes := 4
            7 Input      8 Bit       Muxes := 1
           14 Input      8 Bit       Muxes := 2
           14 Input      4 Bit       Muxes := 2
            2 Input      4 Bit       Muxes := 5
            8 Input      4 Bit       Muxes := 2
            8 Input      3 Bit       Muxes := 1
           14 Input      3 Bit       Muxes := 1
            2 Input      1 Bit       Muxes := 24
            9 Input      1 Bit       Muxes := 10
           14 Input      1 Bit       Muxes := 27
---------------------------------------------------------
-----------------------
Finished RTL Hierarchical Component
Statistics
---------------------------------------------------------
-----------------------
---------------------------------------------------------
-----------------------
Start Part Resource Summary
---------------------------------------------------------
-----------------------
```

Part Resources:
DSPs: 740 (col length:100)
BRAMs: 730 (col length: RAMB18 100 RAMB36 50)

Finished Part Resource Summary

No constraint files found.

Start Cross Boundary and Area Optimization

Warning: Parallel synthesis criteria is not met
INFO: [Synth 8-5546] ROM "next_px" won't be mapped to RAM because it is too sparse
INFO: [Synth 8-5546] ROM "next_temp_x" won't be mapped to RAM because it is too sparse
INFO: [Synth 8-5544] ROM "p_0_out" won't be mapped to Block RAM because address size (3) smaller than threshold (5)
INFO: [Synth 8-5546] ROM "next_state" won't be mapped to RAM because it is too sparse
INFO: [Synth 8-5546] ROM "next_state" won't be mapped to RAM because it is too sparse
INFO: [Synth 8-5546] ROM "next_state" won't be mapped to RAM because it is too sparse
INFO: [Synth 8-5546] ROM "next_cont" won't be mapped to RAM because it is too sparse
---
Finished Cross Boundary and Area Optimization : Time (s): cpu = 00:00:07 ; elapsed = 00:00:14 . Memory (MB): peak = 621.102 ; gain = 358.875

Report RTL Partitions:
+-+-------------+-----------+----------+
| |RTL Partition |Replication |Instances |
+-+-------------+-----------+----------+
+-+-------------+-----------+----------+
No constraint files found.

Start Timing Optimization

Finished Timing Optimization : Time (s): cpu = 00:00:07 ; elapsed = 00:00:14 . Memory (MB): peak = 621.102 ; gain = 358.875

Report RTL Partitions:
+-+-------------+-----------+----------+
| |RTL Partition |Replication |Instances |
+-+-------------+-----------+----------+
+-+-------------+-----------+----------+

Start Technology Mapping

Finished Technology Mapping : Time (s): cpu = 00:00:07 ; elapsed = 00:00:14 . Memory (MB): peak = 621.109 ; gain = 358.883

Report RTL Partitions:
+-+-------------+-----------+----------+
| |RTL Partition |Replication |Instances |
+-+-------------+-----------+----------+

+-+-------------+-----------+----------+

Start IO Insertion

Start Flattening Before IO Insertion

Finished Flattening Before IO Insertion

Start Final Netlist Cleanup

Finished Final Netlist Cleanup

Finished IO Insertion : Time (s): cpu = 00:00:08 ; elapsed = 00:00:15 . Memory (MB): peak = 621.109 ; gain = 358.883

Report Check Netlist:
+------+-----------------+-------+---------+-------+-----------------+
| |Item |Errors |Warnings |Status |Description |
+------+-----------------+-------+---------+-------+-----------------+
|1 |multi_driven_nets | 0| 0|Passed |Multi driven nets |
+------+-----------------+-------+---------+-------+-----------------+

Start Renaming Generated Instances

Finished Renaming Generated Instances : Time (s): cpu = 00:00:08 ; elapsed = 00:00:15 . Memory (MB): peak = 621.109 ; gain = 358.883

Report RTL Partitions:
+-+-------------+-----------+----------+
| |RTL Partition |Replication |Instances |
+-+-------------+-----------+----------+
+-+-------------+-----------+----------+

Start Rebuilding User Hierarchy

Finished Rebuilding User Hierarchy : Time (s): cpu = 00:00:08 ; elapsed = 00:00:15 . Memory (MB): peak = 621.109 ; gain = 358.883

Start Renaming Generated Ports

Finished Renaming Generated Ports : Time (s): cpu = 00:00:08 ; elapsed = 00:00:15 . Memory (MB): peak = 621.109 ; gain = 358.883


```
---------------------------------------------------------
---------------------
Start Handling Custom Attributes
---------------------------------------------------------
---------------------
---------------------------------------------------------
---------------------
Finished Handling Custom Attributes :
Time (s): cpu = 00:00:08 ; elapsed =
00:00:15 . Memory (MB): peak = 621.109 ;
gain = 358.883
---------------------------------------------------------
---------------------
---------------------------------------------------------
---------------------
Start Renaming Generated Nets
---------------------------------------------------------
---------------------
---------------------------------------------------------
---------------------
Finished Renaming Generated Nets :
Time (s): cpu = 00:00:08 ; elapsed =
00:00:15 . Memory (MB): peak = 621.109 ;
gain = 358.883
---------------------------------------------------------
---------------------
---------------------------------------------------------
---------------------
Start Writing Synthesis Report
---------------------------------------------------------
---------------------

Report BlackBoxes:
+-+--------------+----------+
| |BlackBox name |Instances |
+-+--------------+----------+
+-+--------------+----------+

Report Cell Usage:
+------+-------+------+
|      |Cell   |Count |
+------+-------+------+
|1     |BUFG   |     1|
|2     |CARRY4 |    37|
|3     |LUT1   |     1|
|4     |LUT2   |    11|
|5     |LUT3   |    22|
|6     |LUT4   |   105|
|7     |LUT5   |    32|
|8     |LUT6   |   116|
|9     |MUXF7  |    10|
|10    |FDCE   |     4|
|11    |FDRE   |   286|
|12    |FDSE   |    80|
|13    |IBUF   |    11|
|14    |OBUF   |    27|
+------+-------+------+

Report Instance Areas:
+------+---------+-------+------+
|      |Instance |Module |Cells |
+------+---------+-------+------+
|1     |top      |       |   743|
+------+---------+-------+------+
---------------------------------------------------------
---------------------
Finished Writing Synthesis Report : Time
(s): cpu = 00:00:08 ; elapsed = 00:00:15 .
Memory (MB): peak = 621.109 ; gain =
358.883
---------------------------------------------------------
---------------------
Synthesis finished with 0 errors, 0 critical
warnings and 0 warnings.
Synthesis Optimization Runtime : Time (s):
cpu = 00:00:08 ; elapsed = 00:00:15 .
Memory (MB): peak = 621.109 ; gain =
358.883
Synthesis Optimization Complete : Time
(s): cpu = 00:00:08 ; elapsed = 00:00:15 .
Memory (MB): peak = 621.109 ; gain =
358.883
INFO: [Project 1-571] Translating
synthesized netlist
INFO: [Netlist 29-17] Analyzing 47 Unisim
elements for replacement
INFO: [Netlist 29-28] Unisim
Transformation completed in 0 CPU
seconds
INFO: [Project 1-570] Preparing netlist for
logic optimization
```

INFO: [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).
Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.001 . Memory (MB): peak = 717.086 ; gain = 0.000
INFO: [Project 1-111] Unisim Transformation Summary:
No Unisim elements were transformed.

INFO: [Common 17-83] Releasing license: Synthesis
32 Infos, 0 Warnings, 0 Critical Warnings and 0 Errors encountered.
synth_design completed successfully
synth_design: Time (s): cpu = 00:00:13 ; elapsed = 00:00:32 . Memory (MB): peak = 717.086 ; gain = 465.945
Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.001 . Memory (MB): peak = 717.086 ; gain = 0.000
WARNING: [Constraints 18-5210] No constraints selected for write.
Resolution: This message can indicate that there are no constraints for the design, or it can indicate that the used_in flags are set such that the constraints are ignored. This later case is used when running synth_design to not write synthesis constraints to the resulting checkpoint. Instead, project constraints are read when the synthesized design is opened.
INFO: [Common 17-1381] The checkpoint 'C:/Users/Valeria/project_reti_logiche/project_reti_logiche.runs/synth_1/project_reti_logiche.dcp' has been generated.
INFO: [runtcl-4] Executing : report_utilization -file project_reti_logiche_utilization_synth.rpt -pb project_reti_logiche_utilization_synth.pb
INFO: [Common 17-206] Exiting Vivado at Sat Mar 30 12:06:51 2019...

2. Simulazioni

Tutte i test riportati in seguito sono stati svolti e superati in Behavioural Simulation, Post-Synthesis Functional Simulation e Post-Synthesis Timing Simulation.

Prima fase di Testing

Nella prima fase di testing si è voluto verificare il corretto funzionamento del componente focalizzandosi sulle possibili variazioni della maschera di ingresso e sulle posizioni dei punti nello spazio.

Primo Test bench -- Casuale senza sovrapposizioni

Il primo test bench testa il componente in uno scenario casuale in cui si mantiene la distinzione tra i punti.
Si vuole porre attenzione sulla correttezza del componente data una maschera di ingresso qualsiasi senza valutare per ora eventuali sovrapposizioni dei centroidi.
La maschera di ingresso, quindi, è generata in modo casuale: i centroidi sono posti casualmente in posizioni distinte dello spazio e il punto da valutare è a sua volta distinto da essi.

Si riporta il codice completo

```
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;                                   0');
signal   tb_done        : std_logic;
signal   mem_address        : std_logic_vector (15 downto 0) := (others =>
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( 86 , 8)),
                         1 => std_logic_vector(to_unsigned( 56 , 8)),
                         2 => std_logic_vector(to_unsigned( 12 , 8)),
                         3 => std_logic_vector(to_unsigned( 7 , 8)),
                         4 => std_logic_vector(to_unsigned( 145 , 8)),
                         5 => std_logic_vector(to_unsigned( 3 , 8)),
                         6 => std_logic_vector(to_unsigned( 147 , 8)),
                         7 => std_logic_vector(to_unsigned( 10 , 8)),
                         8 => std_logic_vector(to_unsigned( 23 , 8)),
                         9 => std_logic_vector(to_unsigned( 13 , 8)),
                         10 => std_logic_vector(to_unsigned( 35 , 8)),
                         11 => std_logic_vector(to_unsigned( 120 , 8)),
                         12 => std_logic_vector(to_unsigned( 54 , 8)),
                         13 => std_logic_vector(to_unsigned( 21 , 8)),
                         14 => std_logic_vector(to_unsigned( 73 , 8)),
                         15 => std_logic_vector(to_unsigned( 61 , 8)),
                         16 => std_logic_vector(to_unsigned( 102 , 8)),
                         17 => std_logic_vector(to_unsigned( 5 , 8)),
                         18 => std_logic_vector(to_unsigned( 146 , 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 = 00000110
    assert RAM(19) = std_logic_vector(to_unsigned( 6 , 8)) report "TEST FALLITO" severity failure;

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

end projecttb;
```

Il primo test bench, prende come esempio specifico la maschera di ingresso “01010110” come si può leggere dal codice, mentre i centroidi 2 e 3 hanno la stessa distanza di Manhattan dal punto da valutare. Dunque, essendo entrambi i bit corrispondenti ai centroidi posti ad uno nella maschere in ingresso, la maschera di uscita sarà 00000110.

Sostituendo la maschera di esempio con un’altra maschera generata casualmente e sostituendo le coordinate dei punti sempre casualmente mantenendo la distinzione tra essi, è possibile fare numerosi esempi dello stesso tipo, in ogni caso il risultato finale risulta corretto.

Secondo Test bench -- Maschera di ingresso “00000000”-- Caso limite

Il secondo test bench testa la capacità del componente di leggere la maschera di ingresso composta da soli zeri e di conseguenza di non valutare nessuno dei centroidi.
Anche in questo caso non siamo ancora interessati a valutare il comportamento del componente quando i centroidi possono essere sovrapposti quindi essi sono posti casualmente in posizioni distinte dello spazio, e il punto da valutare è a sua volta distinto da essi.

Come risultato la maschera di uscita è composta a sua volta da 8 bit a zero, poiché nessun centroide viene preso in considerazione e l’algoritmo pone tutte le distanze uguali alla *“distanza infinita”* 1111111111.

Viene riportato solo la parte significativa del codice che si distingue dal precedente:

```
signal RAM: ram_type := (0 => std_logic_vector(to_unsigned( 0 , 8)),
                         1 => std_logic_vector(to_unsigned( 56 , 8)),
                         2 => std_logic_vector(to_unsigned( 12 , 8)),
                         3 => std_logic_vector(to_unsigned( 7 , 8)),
                         4 => std_logic_vector(to_unsigned( 145 , 8)),
                         5 => std_logic_vector(to_unsigned( 3 , 8)),
                         6 => std_logic_vector(to_unsigned( 147 , 8)),
                         7 => std_logic_vector(to_unsigned( 10 , 8)),
                         8 => std_logic_vector(to_unsigned( 23 , 8)),
                         9 => std_logic_vector(to_unsigned( 13 , 8)),
                         10 => std_logic_vector(to_unsigned( 35 , 8)),
                         11 => std_logic_vector(to_unsigned( 120 , 8)),
                         12 => std_logic_vector(to_unsigned( 54 , 8)),
                         13 => std_logic_vector(to_unsigned( 21 , 8)),
                         14 => std_logic_vector(to_unsigned( 73 , 8)),
                         15 => std_logic_vector(to_unsigned( 61 , 8)),
                         16 => std_logic_vector(to_unsigned( 102 , 8)),
                         17 => std_logic_vector(to_unsigned( 5 , 8)),
                         18 => std_logic_vector(to_unsigned( 146 , 8)),
                  others => (others =>'0'));
```

…… codice identico al test precedente …..

```
-- 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;
```

Terzo Test bench -- Sovrapposizioni tra i centroidi -- Caso limite

Il terzo test bench testa il componente nel il caso in cui i centroidi si trovano tutti nella medesima posizione nello spazio, per valutare la capacità del componente di salvare tutte le distanze con lo stesso valore e mantenere la distinzione tra i centroidi posti alle stesse coordinate.

La maschera di ingresso è composta esclusivamente da bit posti a uno ‘11111111” , per valutare il caso di massimi centroidi nella stessa posizione.
Il punto da valutare è distinto da essi.

La maschera di uscita è composta dunque a sua volta da 8 bit tutti a uno, poiché tutti i centroidi vengono considerati nella maschera di ingresso e si trovano tutti alla medesima distanza dal punto.

```
signal RAM: ram_type := (0 => std_logic_vector(to_unsigned( 255 , 8)),
                         1 => std_logic_vector(to_unsigned( 56 , 8)),
                         2 => std_logic_vector(to_unsigned( 12 , 8)),
                         3 => std_logic_vector(to_unsigned( 56 , 8)),
                         4 => std_logic_vector(to_unsigned( 12 , 8)),
                         5 => std_logic_vector(to_unsigned( 56 , 8)),
                         6 => std_logic_vector(to_unsigned( 12, 8)),
                         7 => std_logic_vector(to_unsigned( 56 , 8)),
                         8 => std_logic_vector(to_unsigned( 12 , 8)),
                         9 => std_logic_vector(to_unsigned( 56 , 8)),
                         10 => std_logic_vector(to_unsigned( 12 , 8)),
                         11 => std_logic_vector(to_unsigned( 56 , 8)),
                         12 => std_logic_vector(to_unsigned( 12 , 8)),
                         13 => std_logic_vector(to_unsigned( 56 , 8)),
                         14 => std_logic_vector(to_unsigned( 12 , 8)),
                         15 => std_logic_vector(to_unsigned( 56 , 8)),
                         16 => std_logic_vector(to_unsigned( 12 , 8)),
                         17 => std_logic_vector(to_unsigned( 5 , 8)),
                         18 => std_logic_vector(to_unsigned( 146 , 8)),
                others => (others =>'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;
```

Quarto Test bench -- Sovrapposizione centroidi e punto da valutare -- Caso limite

Il quarto test bench testa il componente nel caso in cui sia i centroidi, sia il punto da valutare si trovano tutti nella medesima posizione nello spazio, in questo modo si vuole testare la capacità del componente di riconoscere la distanza minima (pari a zero) e uguale per tutti, e mantenere la distinzione tra i centroidi e il punto da valutare.

La maschera di ingresso è composta esclusivamente da bit posti a uno ‘11111111” , per valutare il caso di massimi centroidi nella stessa posizione.

La maschera di uscita è composta dunque a sua volta da 8 bit tutti a uno, poichè tutti i centroidi vengono considerati nella maschera di ingresso e si trovano tutti a distanza nulla con il punto da valutare.

```
signal RAM: ram_type := (0 => std_logic_vector(to_unsigned( 255 , 8)),
                         1 => std_logic_vector(to_unsigned( 56 , 8)),
                         2 => std_logic_vector(to_unsigned( 12 , 8)),
                         3 => std_logic_vector(to_unsigned( 56 , 8)),
                         4 => std_logic_vector(to_unsigned( 12 , 8)),
                         5 => std_logic_vector(to_unsigned( 56 , 8)),
                         6 => std_logic_vector(to_unsigned( 12, 8)),
                         7 => std_logic_vector(to_unsigned( 56 , 8)),
                         8 => std_logic_vector(to_unsigned( 12 , 8)),
                         9 => std_logic_vector(to_unsigned( 56 , 8)),
                         10 => std_logic_vector(to_unsigned( 12 , 8)),
                         11 => std_logic_vector(to_unsigned( 56 , 8)),
                         12 => std_logic_vector(to_unsigned( 12 , 8)),
                         13 => std_logic_vector(to_unsigned( 56 , 8)),
                         14 => std_logic_vector(to_unsigned( 12 , 8)),
                         15 => std_logic_vector(to_unsigned( 56 , 8)),
                         16 => std_logic_vector(to_unsigned( 12 , 8)),
                         17 => std_logic_vector(to_unsigned( 56 , 8)),
                         18 => std_logic_vector(to_unsigned( 12 , 8)),
                others => (others =>'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;
```

Quinto Test bench -- Massima distanza minima -- Caso limite

Il quinto test bench valuta il comportamento del componente quando la distanza tra il punto da valutare e un centroide è massima. In questo modo, testiamo la capacità di gestire la massima distanza minima possibile e di trovare eventuali limiti del nostro componente.

Per fare ciò, sette centroidi non sono considerati (nella maschera di ingresso hanno il bit a 0), mentre uno di essi è posizionato all'estremità dello spazio, con coordinate x = 255, y = 255 e il punto da considerare è posto a coordinate x =0 e y = 0.

```
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( 7 , 8)),
                         4 => std_logic_vector(to_unsigned( 145 , 8)),
                         5 => std_logic_vector(to_unsigned( 3 , 8)),
                         6 => std_logic_vector(to_unsigned( 147 , 8)),
                         7 => std_logic_vector(to_unsigned( 10 , 8)),
                         8 => std_logic_vector(to_unsigned( 23 , 8)),
                         9 => std_logic_vector(to_unsigned( 13 , 8)),
                         10 => std_logic_vector(to_unsigned( 35 , 8)),
                         11 => std_logic_vector(to_unsigned( 120 , 8)),
                         12 => std_logic_vector(to_unsigned( 54 , 8)),
                         13 => std_logic_vector(to_unsigned( 21 , 8)),
                         14 => std_logic_vector(to_unsigned( 73 , 8)),
                         15 => std_logic_vector(to_unsigned( 1 , 8)),
                         16 => std_logic_vector(to_unsigned( 5 , 8)),
                         17 => std_logic_vector(to_unsigned( 0 , 8)),
                         18 => std_logic_vector(to_unsigned( 0 , 8)),
                others => (others =>'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;
```

Il componente reagisce in modo corretto, producendo una maschera di uscita pari a "00000001". Ponendo quindi una massima distanza minima non viene compromesso il risultato.

Seconda fase di Testing: più computazioni successive

In questa fase si testano i comportamenti del componente in caso di reset, richieste di computazioni successive e modifiche del contenuto della RAM dopo la prima computazione.

Sesto Test bench -- Due computazioni identiche successive

Questo test serve per controllare che il componente riesca a gestire due richieste di computazione successive senza che la macchina sia resettata prima della seconda computazione. Viene infatti semplicemente riposto a 1 il segnale di start. Questo test conferma, inoltre, che il circuito non modifica in modo anomalo i valori della maschera di ingresso e delle coordinate dei punti nella RAM.

```
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';

    --seconda computazione

    wait for 100 ns;
    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 = 00000110
    assert RAM(19) = std_logic_vector(to_unsigned( 6 , 8)) report "TEST FALLITO" severity failure;

    assert false report "Simulation Ended!, TEST PASSATO" severity failure;
```

Settimo Test bench -- Vari reset

In questo test, il componente viene sottoposto a più reset durante la sua computazione. Il primo reset è posto all'incirca durante la salita del clock, un momento critico per il componente, perché avviene l'aggiornamento dei segnali, in particolare quello di stato della macchina. Il secondo reset è invece casuale, posto ragionevolmente nel mezzo della computazione. Il componente supera il test.

```
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;

    --reset a meta' clock
    wait for 150 ns;
    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';

    --reset casuale
    wait for 1737 ns;
    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 = 00000110
    assert RAM(19) = std_logic_vector(to_unsigned( 6 , 8)) report "TEST FALLITO" severity failure;

    assert false report "Simulation Ended!, TEST PASSATO" severity failure;
```

Ottavo Test bench -- RAM cambia valori dopo la prima computazione

Un test simile a quello delle due computazioni successive identiche, ma prima della seconda computazione la RAM viene modificata, usufruendo di un segnale aggiunto a proposito:
`signal new_comp : std_logic := '0';`
Il processo MEM del test bench viene modificato in tal modo:

```
MEM : process(tb_clk, new_comp)
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;
    if new_comp = '1' then
        RAM(0)  <= "10000011";
        RAM(1)  <= "00000001";
        RAM(2)  <= "00000001";
        RAM(3)  <= "00000001";
        RAM(4)  <= "00000001";
        RAM(5)  <= "00000000";
        RAM(6)  <= "00000000";
        RAM(7)  <= "00000000";
        RAM(8)  <= "00000000";
        RAM(9)  <= "00000000";
        RAM(10)  <= "00000000";
        RAM(11)  <= "00000000";
        RAM(12)  <= "00000000";
        RAM(13)  <= "00000000";
        RAM(14)  <= "00000000";
        RAM(15)  <= "00000010";
        RAM(16)  <= "00000010";
        RAM(17)  <= "00000000";
        RAM(18)  <= "00000000";

        --maschera di output riportata a 0
        RAM(19)  <= "00000000";
    end if;
end process;
```

Il process test risulta così:

```
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';

    --RAM viene modificata
    wait for 100 ns;
    new_comp <= '1';
    wait for 50 ns;
    new_comp <= '0';

    --nuova computazione
    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 = 00000011
    assert RAM(19) = std_logic_vector(to_unsigned( 3 , 8)) report "TEST FALLITO" severity failure;

    assert false report "Simulation Ended!, TEST PASSATO" severity failure;
```

Il componente supera il test, confermando che non solo esso non modifichi la RAM in modo anomalo, ma che effettivamente compia una seconda computazione. Basandosi solo sul test delle computazioni identiche, infatti, potrebbe sorgere il dubbio che il circuito nella seconda computazione si limiti a lasciare invariata la maschera di uscita scritta nella RAM nella computazione precedente.

Nono Test bench -- Reset, modifica della RAM e nuova computazione

Seguendo il filo logico del test precedente, si vuole confermare che il reset del componente avvenga realmente, e che quindi una nuova computazione avvenga. Il codice del test bench è identico a quello del test successivo, salvo questi cambiamenti nel process test. Il circuito supera anche questo test.

```
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';

    --reset
    wait for 1267 ns;
    tb_rst <= '1';
    wait for c_CLOCK_PERIOD;
    tb_rst <= '0';

    --RAM viene modificata
    wait for 100 ns;
    new_comp <= '1';
    wait for 50 ns;
    new_comp <= '0';

    --nuova computazione
    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 = 00000011
    assert RAM(19) = std_logic_vector(to_unsigned( 3 , 8)) report "TEST FALLITO" severity failure;

    assert false report "Simulation Ended!, TEST PASSATO" severity failure;
```

4.Conclusioni

Nella realizzazione del modulo si sono fatte delle scelte di progettazione che si sono rivelate vincenti.
L'algoritmo creato ci ha permesso di risparmiare componenti per la realizzazione della memoria, ad esempio non salvando il valore delle coordinate dei centroidi, perché considerati dei dati superflui, e salvando solo la distanza punto-centroide.
Inoltre, l'utilizzo della convenzione *distanza infinita* ha permesso di realizzare un algoritmo semplice e completo, che ha permesso di valutare efficacemente quali centroidi considerare in ogni computazione.

Il componente, come richiesto dalla consegna, è quindi correttamente simulabile e sintetizzabile.