Politecnico di Milano

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE
Corso di Laurea in Ingegneria Informatica

Prova Finale di Reti Logiche

Candidati:
Gabriele Daglio
Matricola 866337, Codice Persona 10537168

Luca Cattaneo
Matricola 865870, Codice Persona 10521219

Professore:
William Fornaciari

Tutor:
Davide Zoni

Anno Accademico 2018–2019

Indice

Capitolo 1

Documentazione

Dalla specifica si vuole implementare un componente HW descritto in VHDL che, una volta fornite le coordinate di un punto appartenente a uno spazio di 256 x 256, sia in grado di valutare a quale/i dei centroidi risulti più vicino (Manhattan distance).

1.1 Diagramma degli stati

Abbiamo scelto di implementare una FSM, di cui in figura 1.1 si ha il diagramma degli stati.

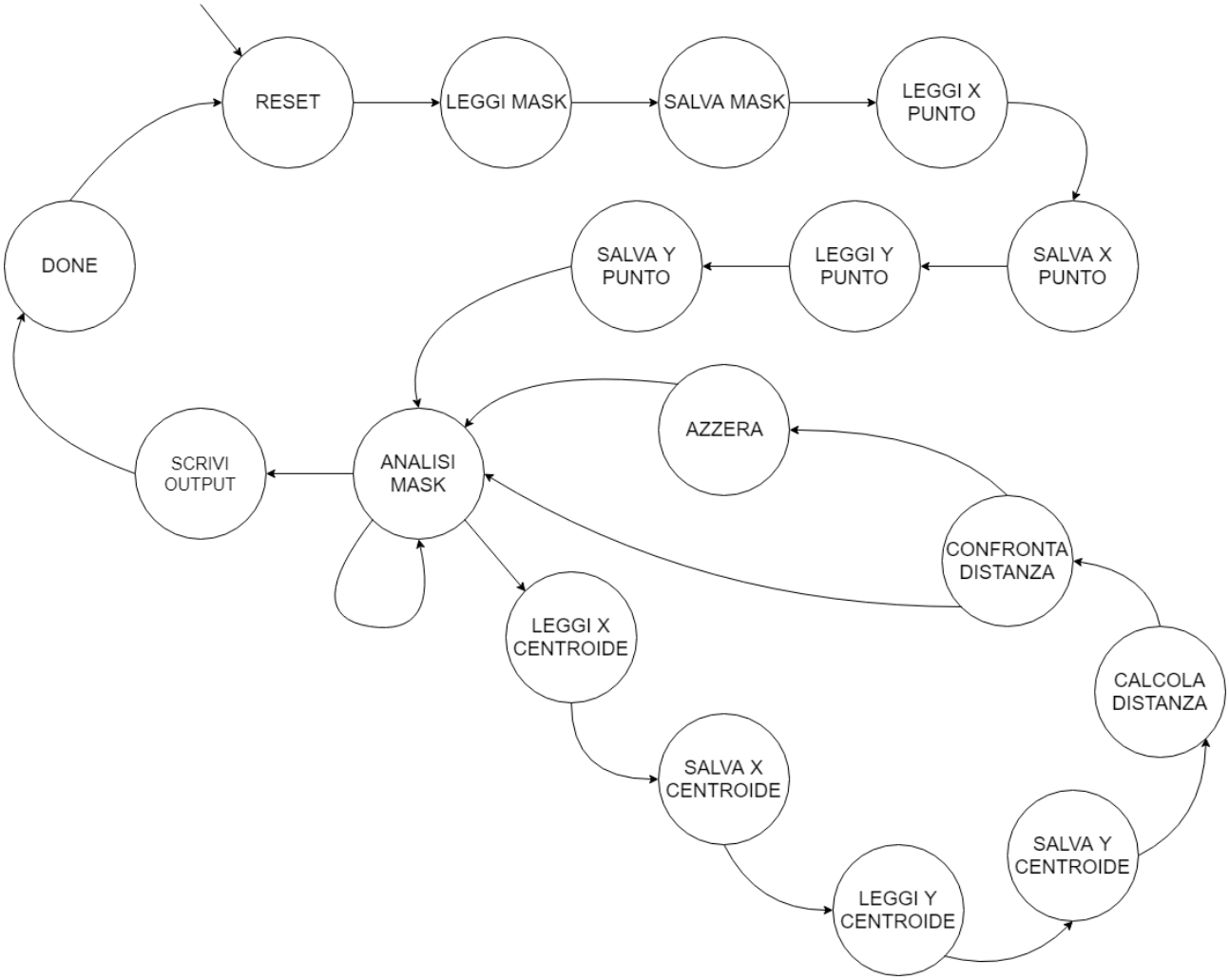

Figura 1.1: Diagramma degli stati della FSM implementata

RESET Stato iniziale e finale della macchina. Quando il segnale `i_start` è pari a **1** la macchina passa allo stato *LEGGI MASK*.

LEGGI MASK Manda richiesta di lettura, della maschera di input, alla RAM. Si sposta nello stato *SALVA MASK*.

SALVA MASK Una volta ottenuto[1] il valore della maschera di input, lo salva in un registro. Entra nello stato *LEGGI X PUNTO*.

LEGGI X PUNTO Manda richiesta di lettura, della coordinata X del punto, alla RAM. Si sposta nello stato *SALVA X PUNTO*.

SALVA X PUNTO Una volta ottenuta risposta dalla RAM, la salva in un registro. Passa allo stato *LEGGI Y PUNTO*.

LEGGI Y PUNTO Manda richiesta di lettura, della coordinata Y del punto, alla RAM. Si sposta nello stato *SALVA Y PUNTO*.

SALVA Y PUNTO Una volta ottenuta risposta dalla RAM, la salva in un registro. Passa ad *ANALISI MASK*.

ANALISI MASK Analizza la maschera utilizzando un segnale interno i, inizializzato a 0.

- Se tale segnale i è minore di 8, continua valutando il bit in posizione corrispondente. Se tale bit è:
  - pari a 0: pone a 0 il bit corrispondente nella maschera di output (che è salvata in un registro), incrementa il segnale i di 1 e il segnale *indirizzo* (utilizzato per leggere gli indirizzi dei centroidi, inizializzato a 1) di 2. Resta in *ANALISI MASK*.
  - pari a 1: si sposta in *LEGGI X CENTROIDE*.
- Se non è minore di 8 si sposta in *SCRIVI OUTPUT*.

LEGGI X CENTROIDE Manda richiesta di lettura, della coordinata X del centroide, alla RAM. Incrementa *indirizzo* di 1 e passa a *SALVA X CENTROIDE*.

SALVA X CENTROIDE Una volta ottenuta risposta dalla RAM, la salva in un registro. Passa allo stato *LEGGI Y CENTROIDE*.

LEGGI Y CENTROIDE Manda richiesta di lettura, della coordinata Y del CENTROIDE, alla RAM. Incrementa *indirizzo* di 1 e si sposta nello stato *SALVA Y CENTROIDE*.

SALVA Y CENTROIDE Una volta ottenuta risposta dalla RAM, la salva in un registro. Passa a *CALCOLA DISTANZA*.

CALCOLA DISTANZA Calcola la Distanza di Manhattan, la salva in un registro (*distanza*) si sposta in *CONFRONTA DISTANZA*.

CONFRONTA DISTANZA Confronta *distanza* con quella minima[2] (*dist_min*, salvata in un registro).

- Se *distanza* $<$ *dist_min* allora aggiorna il valore di *dist_min* ponendolo pari a *distanza* e si sposta in *AZZERA*.
- Se *distanza* $=$ *dist_min* pone a 1 il bit corrispondente a i nella maschera di output e incrementa i di 1 . Si sposta in *ANALISI MASK*.
- Se *distanza* $>$ *dist_min* pone a 0 il bit corrispondente a i nella maschera di output e incrementa i di 1. Si sposta in *ANALISI MASK*.

[1] Si noti che in tutti gli stati in cui arriva un input dalla RAM è stato inserito il comando `after 5 ns` per sopperire al ritardo della RAM.

[2] All'inizio è posta pari a 511.

AZZERA Azzera la maschera di output per poi porre a 1 il bit corrispondente a i nella maschera, incrementa i di 1 e si sposta in *ANALISI MASK*.

SCRIVI OUTPUT Scrive nella RAM il valore della maschera di output. Si sposta in *DONE*.

DONE Setta `o_done` a 1. In seguito a ciò si sposta in *RESET*.

1.2 Testing

Test effettuati in Pre-sintesi e in Post-sintesi functional:

-Testbench di esempio

-Maschera di input pari a 000000000
Obiettivo: verificare il funzionamento dello stato *ANALISI MASK* in presenza di soli zeri in input.

-Centroidi coincidenti in (0,0), punto da valutare in (255, 255) (e viceversa) e maschera di input a 11111111
Obiettivo: verificare il funzionamento di *CALCOLA DISTANZA* e di *CONFRONTA DISTANZA* in presenza di distanza Manhattan massima (510).

-Centroidi coincidenti col punto da valutare (maschera input 11101100)

```
signal RAM: ram_type := (0 => std_logic_vector(to_unsigned( 236 , 8)),
                         1 => std_logic_vector(to_unsigned( 100 , 8)),
                         2 => std_logic_vector(to_unsigned( 78 , 8)),
                         3 => std_logic_vector(to_unsigned( 100 , 8)),
                         4 => std_logic_vector(to_unsigned( 78 , 8)),
                         5 => std_logic_vector(to_unsigned( 100 , 8)),
                         6 => std_logic_vector(to_unsigned( 78 , 8)),
                         7 => std_logic_vector(to_unsigned( 100 , 8)),
                         8 => std_logic_vector(to_unsigned( 78 , 8)),
                         9 => std_logic_vector(to_unsigned( 100 , 8)),
                         10 => std_logic_vector(to_unsigned( 78 , 8)),
                         11 => std_logic_vector(to_unsigned( 100 , 8)),
                         12 => std_logic_vector(to_unsigned( 78 , 8)),
                         13 => std_logic_vector(to_unsigned( 100 , 8)),
                         14 => std_logic_vector(to_unsigned( 78 , 8)),
                         15 => std_logic_vector(to_unsigned( 100 , 8)),
                         16 => std_logic_vector(to_unsigned( 78 , 8)),
                         17 => std_logic_vector(to_unsigned( 100 , 8)),
                         18 => std_logic_vector(to_unsigned( 78 , 8)),
                others => (others =>'0'));
```

Obiettivo: verificare il funzionamento di *ANALISI MASK*, in presenza di maschera generica, e di *CALCOLA DISTANZA*, con distanza nulla tra il punto e i centroidi.

-Centroidi equidistanti dal punto da valutare (maschera input 11111111)

```
signal RAM: ram_type := (0 => std_logic_vector(to_unsigned( 255 , 8)),
                         1 => std_logic_vector(to_unsigned( 122 , 8)),
                         2 => std_logic_vector(to_unsigned( 122 , 8)),
                         3 => std_logic_vector(to_unsigned( 132 , 8)),
                         4 => std_logic_vector(to_unsigned( 132 , 8)),
                         5 => std_logic_vector(to_unsigned( 137 , 8)),
                         6 => std_logic_vector(to_unsigned( 127 , 8)),
                         7 => std_logic_vector(to_unsigned( 117 , 8)),
                         8 => std_logic_vector(to_unsigned( 127 , 8)),
                         9 => std_logic_vector(to_unsigned( 121 , 8)),
                         10 => std_logic_vector(to_unsigned( 123 , 8)),
                         11 => std_logic_vector(to_unsigned( 133 , 8)),
                         12 => std_logic_vector(to_unsigned( 131 , 8)),
                         13 => std_logic_vector(to_unsigned( 127 , 8)),
                         14 => std_logic_vector(to_unsigned( 137 , 8)),
                         15 => std_logic_vector(to_unsigned( 135 , 8)),
                         16 => std_logic_vector(to_unsigned( 129 , 8)),
                         17 => std_logic_vector(to_unsigned( 127 , 8)),
                         18 => std_logic_vector(to_unsigned( 127 , 8)),
                others => (others =>'0'));
```

Obiettivo: verificare il funzionamento di *CALCOLA DISTANZA* in presenza di centroidi equidistanti dal punto.

-Test generico (maschera input 11001000)

```
signal RAM: ram_type := (0 => std_logic_vector(to_unsigned( 218 , 8)),
                         1 => std_logic_vector(to_unsigned( 200 , 8)),
                         2 => std_logic_vector(to_unsigned( 110 , 8)),
                         3 => std_logic_vector(to_unsigned( 190 , 8)),
                         4 => std_logic_vector(to_unsigned( 100 , 8)),
                         5 => std_logic_vector(to_unsigned( 240 , 8)),
                         6 => std_logic_vector(to_unsigned( 20 , 8)),
                         7 => std_logic_vector(to_unsigned( 150 , 8)),
                         8 => std_logic_vector(to_unsigned( 80 , 8)),
                         9 => std_logic_vector(to_unsigned( 70 , 8)),
                         10 => std_logic_vector(to_unsigned( 140 , 8)),
                         11 => std_logic_vector(to_unsigned( 140 , 8)),
                         12 => std_logic_vector(to_unsigned( 90 , 8)),
                         13 => std_logic_vector(to_unsigned( 170 , 8)),
                         14 => std_logic_vector(to_unsigned( 60 , 8)),
                         15 => std_logic_vector(to_unsigned( 100 , 8)),
                         16 => std_logic_vector(to_unsigned( 90 , 8)),
                         17 => std_logic_vector(to_unsigned( 120 , 8)),
                         18 => std_logic_vector(to_unsigned( 50 , 8)),
                others => (others =>'0'));
```

Obiettivo: verificare il funzionamento di *AZZERA*.

1.3 Ottimizzazioni

Non si ravvisano eventuali ottimizzazioni significative.