Prova Finale (Progetto di Reti Logiche) - Anno Accademico 2018-2019

Nome : frANCESCO

COGNOME: CECCHETTI

MATRICOLA: 000000

codice persona: 1111111

A.A 2018-2019

Sommario

[Titolo 1 1](#_Toc7703307)

[Titolo 2 1](#_Toc7703308)

**Struttura Generale**

Il progetto è stato sviluppato attraverso l’implementazione in VHDL di una *FSM* (Finite State Machine), più precisamente di una macchina di Mealy, in quanto la funzione del segnale di output lavora anche conl’input del componente.

Nell’implementazione in VHDL, è stata effettuata la scelta di basare l’automa a stati finiti su 3 processi, scelta che trova la sua logica in un’ottica basata sulla estrema specializzazione di questi ultimi:

* Il processo *curr\_state\_update* si occupa della parte sequenziale del componente, sensibile al clock, gestisce la transizione dello stato corrente propagando lo stato successivo nello stato corrente, oppure imposta lo stato corrente su uno stato di reset nel caso quest’ultimo venga imposto dalla RAM.
* Il processo *next\_state\_update* fa parte del blocco combinatorio e si occupa di propagare lo stato successivo in base allo stato corrente.
* Il processo *ouput\_update* consiste sostanzialmente nella funzione di output ed è dove la logica è effettivamente implementata attraverso il costrutto Case di VHDL.

Tutti i cambi di stato avvengono sul fronte di salita del clock(*i\_clk)* e la macchina può essere riportata nello stato di reset da qualunque stato alzando il segnale di reset(*i\_rst*).

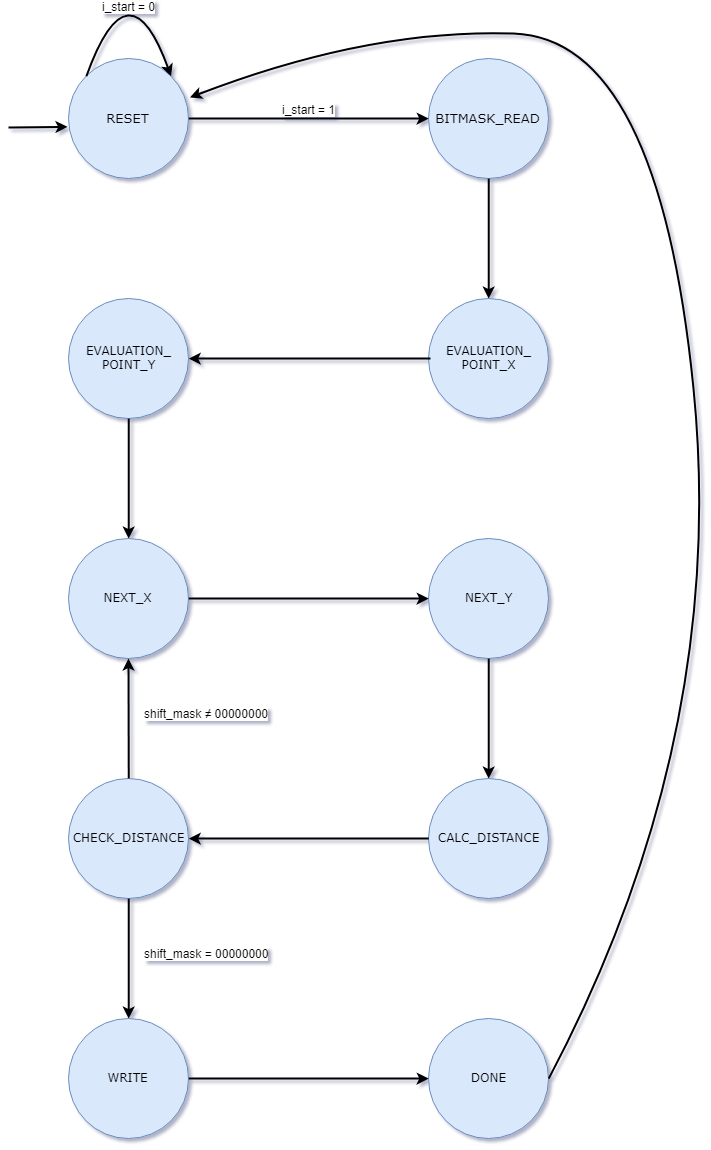
**Algoritmo**

L’algoritmo con cui si è deciso di procedere è molto semplice e consta dei seguenti steps concettuali:

1. Al segnale di start vengono letti e immagazzinati i segnali relativi alla maschera d’ingresso e alle coordinate del punto da valutare.
2. Vengono lette e salvate le coordinate X e Y del centroide da analizzare.
3. Viene calcolata la distanza di Manhattan tra il centroide appena letto e il punto da valutare
4. Se la distanza appena calcolata è minore della distanza minima fino ad ora trovata, e il centroide in analisi è valido in base alla maschera d’ingresso, allora il centroide sarà il più vicino (o uno dei più vicini) al punto da valutare. In tutti i casi viene aggiornata conseguentemente la maschera d’uscita.
5. Se quello appena letto era l’ultimo centroide, procedo a scrivere in memoria RAM la maschera d’uscita costruita fino a questo punto, altrimenti si ritorna al punto 2.

Vediamo nel paragrafo successivo la macchina a stati impiegata per risolvere questo algoritmo.

**Diagramma macchina a stati**

****

**Stati**

* RESET: rappresenta lo stato di Reset/Start, in base al fatto che il segnale *i\_start* sia rispettivamente 0 o 1. In caso di *i\_start* = 1, viene richiesta la lettura dalla RAM all’indirizzo della maschera d’ingresso e vengono salvati i segnali, inizializzati a valori di reset, all’interno di *master\_preserv\_reg*.
* BITMASK\_READ: stato in cui viene letta, e memorizzata nel registro *bitmask\_reg*, la maschera d’ingresso.
* EVALUATION\_POINT\_X: stato in cui viene letta, e memorizzata nel registro *ev\_point\_x\_reg*, la coordinata X del punto da valutare.
* EVALUATION\_POINT\_Y: stato in cui viene letta, e memorizzata nel registro *ev\_point\_y\_reg*, la coordinata Y del punto da valutare.
* NEXT\_X: lettura, e memorizzazione nel registro *curr\_x\_reg*, della coordinata X del nuovo centroide.
* NEXT\_Y: lettura, e memorizzazione nel registro *curr\_y\_reg*, della coordinata Y del nuovo centroide.
* CALC\_DISTANCE: stato necessario per calcolare la distanza di Manhattan tra il punto da valutare e il centroide nei due stati precedenti. Viene, inoltre, eseguito uno shift a sinistra, di 1 bit, del segnale *shift\_mask*, fondamentale per la valutazione della distanza.
* CHECK\_DISTANCE: stato che si occupa di controllare la validità del centroide rispetto alla maschera d’ingresso, e la sua distanza dal punto da valutare. Se il centroide è un punto valido, il segnale di output viene aggiornato in base alla sua distanza rispetto a quella minima registrata fino al centroide in analisi. Basandosi sul segnale *shift\_mas*k, si valuterà se i centroidi da analizzare sono finiti, e quindi procedere alla scrittura dell’output in memoria, oppure tornare a leggere la coordinata successiva nello stato NEXT\_X.
* WRITE: stato in cui viene richiesto l’accesso in scrittura alla RAM (*o\_en* = 1, *o\_we* = 1) e che scriverà quindi l’output, eventualmente calcolato nello stato precedente, all’indirizzo di output.
* DONE: stato che conclude l’esecuzione della macchina, segnalando che la macchina ha terminato alzando il segnale *o\_done*. Si procederà dunque a tornare nello stato di reset della macchina.

**Note su segnali:**

* *shift\_mask* è un segnale di un byte inizializzato a “00000001” e che subisce uno shift a sinistra di un bit per ogni centroide calcolato e valutato. Il suo scopo è quello di avere un flag per capire rapidamente che è stato letto l’ultimo centroide da valutare, che coincide con la condizione *shift\_mask* = ”00000000”, ossia quando il bit ‘1’ è stato shiftato di 8 posizioni, corrispondenti agli 8 centroidi analizzati.
* *new\_address* è un segnale pilotato dal contatore counter, che fornisce l’indirizzo di lettura corretto per la maschera d’ingresso, le coordinate del punto da valutare e di tutti i centroidi.
* *reg\_params\_e* e *save\_signals* sono segnali di enable per i registri; quando alzati, attivano il salvataggio di nuovi valori. *reg\_params\_e* utilizza una codifica One Hot per abilitare il registro corretto.

Altri segnali non citati hanno il mero scopo di calcolo e/o rendere possibile il funzionamento della macchina.

**Componenti**

L’unità di base (*project\_reti\_logiche*)è, come presumibile dalla descrizione fatta nei paragrafi precedenti, supportata da altre 3 entità, ovvero 2 registri e 1 contatore:

* *reg\_prev* si occupa sostanzialmente di propagare il valore di segnali di lunghezza variabile tra 8 e 9 bit, utili durante l’esecuzione della FSM, in modo da evitare possibili inferring latch e può essere resettato per impostare i segnali salvati a valori base.
* *reg* è un semplice registro che si occupa di salvare dei segnali di lunghezza fissa di un byte, letti dalla RAM, di cui la FSM ha bisogno pressoché sempre durante la sua esecuzione, come il byte della maschera d’ingresso, o le coordinate del punto da valutare e del punto correntemente in analisi.
* *counter* è un contatore che incrementa di 1 il valore dell’indirizzo della RAM da leggere, fino alla completa lettura dei segnali in RAM necessari.

Tutti queste entità sono controllabili da dei segnali di enable gestiti nella FSM.

**Casi di test**

Sono stati manualmente prodotti dei test per verificare il corretto funzionamento della macchina nei casi limite.

NB:

* \*random\* vuol dire che il valore, o valori, è stato impostato con un valore casuale e ha un significato solamente esplicativo in questo report.
* \*same\_random\* vuol dire che i valori sono casuali, ma uguali tra di loro.

signal RAM: ramtype := (others => (others => ’0’));

assert RAM(19) = “00000000” report "TEST FALLITO" severity failure;

signal RAM: ramtype := (0 => "00000000", others => (others => \*random\*));

assert RAM(19) = “00000000” report "TEST FALLITO" severity failure;

signal RAM: ramtype := (0 => "11111111", others => (others => ’0’));

assert RAM(19) = “11111111” report "TEST FALLITO" severity failure;

signal RAM: ramtype := (0 => "11111111", others => (others => \*same\_random\*));

assert RAM(19) = “11111111” report "TEST FALLITO" severity failure;

*a* = \*random\*

signal RAM: ramtype := (others => (others => *a*));

assert RAM(19) = a report "TEST FALLITO" severity failure;

Un ulteriore test effettuato, di cui si riportano qui sotto i dettagli solo per renderlo riproducibile, è stato realizzato per sollecitare la parte di calcolo e controllo delle distanze dei vari centroidi, in quanto le quest’ultime risultano molto simili.

signal RAM: ram\_type := (0 => std\_logic\_vector(to\_unsigned( 255 , 8)), 1 => std\_logic\_vector(to\_unsigned( 223 , 8)), 2 => std\_logic\_vector(to\_unsigned( 93 , 8)), 3 => std\_logic\_vector(to\_unsigned( 223 , 8)), 4 => std\_logic\_vector(to\_unsigned( 93 , 8)), => std\_logic\_vector(to\_unsigned( 223 , 8)), 7 => std\_logic\_vector(to\_unsigned( 93 , 8)), 8 => std\_logic\_vector(to\_unsigned( 223 , 8)), 9 => std\_logic\_vector(to\_unsigned( 223 , 8)), 10 => std\_logic\_vector(to\_unsigned( 93 , 8)), 11 => std\_logic\_vector(to\_unsigned( 223 , 8)), 12 => std\_logic\_vector(to\_unsigned( 93 , 8)), 13 => std\_logic\_vector(to\_unsigned( 223 , 8)), 14 => std\_logic\_vector(to\_unsigned( 93 , 8)), 15 => std\_logic\_vector(to\_unsigned( 80 , 8)), 16 => std\_logic\_vector(to\_unsigned( 245 , 8)), 17 => std\_logic\_vector(to\_unsigned( 0 , 8)), 18 => std\_logic\_vector(to\_unsigned( 0 , 8)), others => (others =>'0'));

assert RAM(19) = std\_logic\_vector(to\_unsigned( 127 , 8)) report "TEST FALLITO" severity failure;