A.A 2018-2019

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

NOME: FRANCESCO
COGNOME: CECCHETTI
MATRICOLA: 84****
CODICE PERSONA: 10******

PROFESSORE: WILLIAM FORNACIARI
TUTOR: DAVIDE ZONI

Indice

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 con l'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*).

L'implementazione supera sia la simulazione comportamentale di pre-sintesi, sia la simulazione funzionale di post-sintesi.

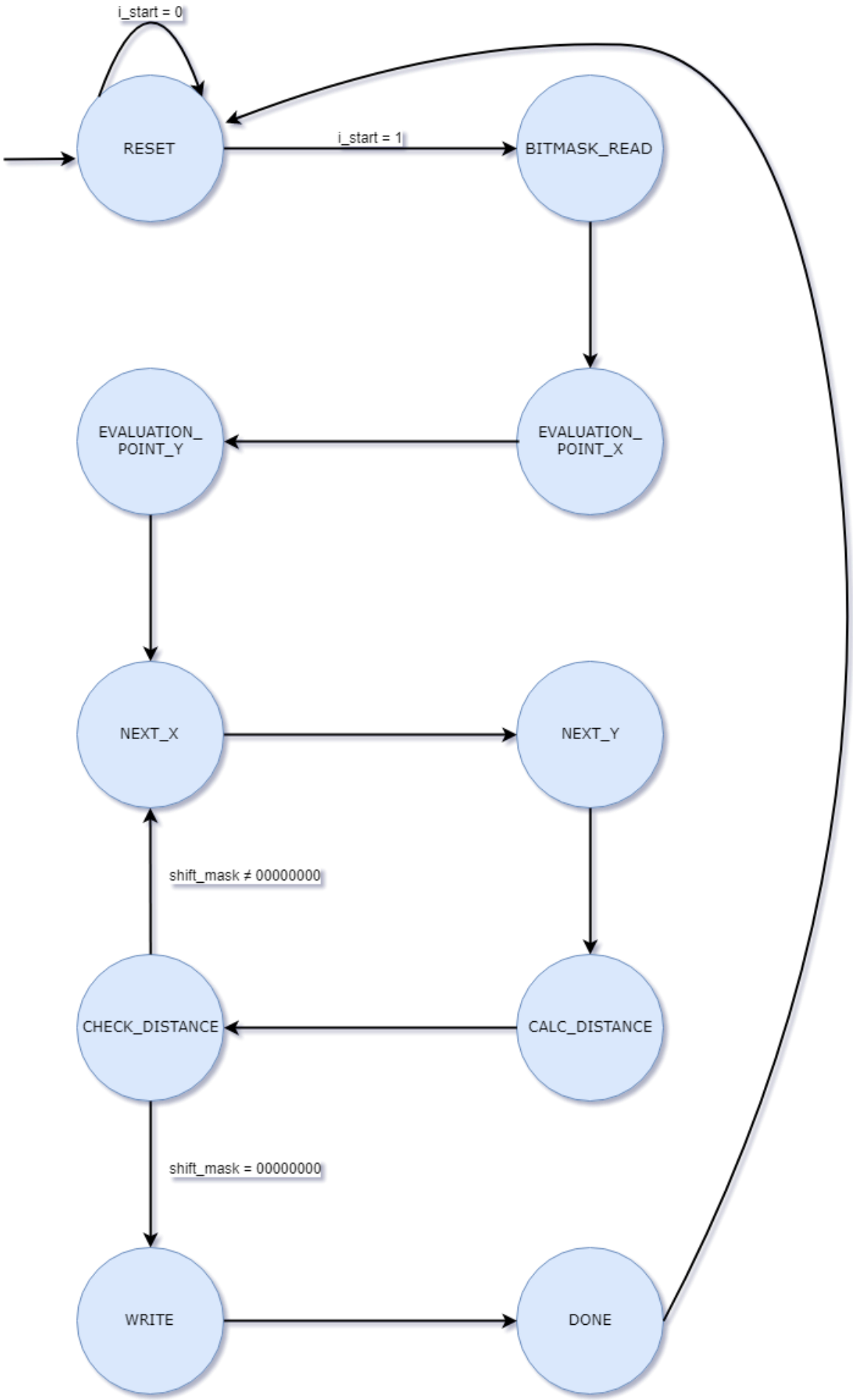
Algoritmo

L'algoritmo astratto 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 ed eventualmente la distanza minima trovata.
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


i_start = 0
RESET
i_start = 1
BITMASK_READ
EVALUATION_
POINT_Y
EVALUATION_
POINT_X
NEXT_X
NEXT_Y
shift_mask ≠ 00000000
CHECK_DISTANCE
CALC_DISTANCE
shift_mask = 00000000
WRITE
DONE

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 di lavoro, inizializzati ai 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 decidere che transizione eseguire al termine dell'esecuzione dello stato CHECK_DISTANCE.
- 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 e di distanza minima (*min_distance*) vengono aggiornati 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 prodotti dei test randomici per testare un ampio spettro di casi e controllare il comportamento dell'unità sul grande numero. Sono stati, inoltre, prodotti manualmente altri test, di maggior interesse, per verificare il corretto funzionamento della macchina nei casi limite.

Per ogni test è sempre interessante valutare i valori ai loro estremi di lavoro, "00000000" e "11111111" per le coordinate dei punti.

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.

Test studiati:

- **Casi limite "bit_mask = 0":**

  La maschera d'ingresso messa a 0 pone un caso limite in quanto nessun centroide deve essere considerato. Viene dunque portata al limite quella parte del design che deve valutare le condizioni di analisi di ogni centroide.

  1. *signal RAM: ram_type := (0 => "00000000", others => (others => '0'));*

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

  2. *signal RAM: ram_type := (0 => "00000000", others => (others => *random*));*

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

  3. *signal RAM: ram_type := (0 => "00000000", others => (others => * same_random*));*

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

- **Casi limite "bit_mask = 1" e coordinate dei punti tutte uguali:**

La maschera rende valutabile ogni punto, l'intento di questa casistica è sollecitare quella parte di design che deve valutare le distanze; la coincidenza di tutti punti rende il test particolarmente interessante per verificare che la macchina si comporti come da specifica.

1. *signal RAM: ram_type := (0 => "11111111", others => (others => *same_random*));*

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

2. *signal RAM: ram_type := (0 => "11111111", others => (others =>'0'));*

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

3. *signal RAM: ram_type := (0 => "11111111", others => (others => '1'));*

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

- **Caso limite "tutto uguale":**

Maschera d'ingresso, punto da valutare e ogni centroide tutti con la stessa coordinata X coincidente con la rispettiva Y. Anche l'output dovrà coincidere sempre con questo valore; L'obbiettivo è testare sia la parte di valutazione di analisi del punto, sia di calcolo della distanza, con un test molto particolare. I casi limite "tutto 0" e "tutto 1" sono già stati testati nelle casistiche precedenti.

1. *a <= *random*; -- semplificazione di esposizione del report*

   signal RAM: ramtype := (0 => a, 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 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)) ,5 => std_logic_vector(to_unsigned(223 , 8)), 6 => std_logic_vector(to_unsigned(93 , 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;