PROVA FINALE (PROGETTO DI RETI LOGICHE)

A.A. 2018/19

Sez. Professore William Fornaciari

Tutor Davide Zoni

Studenti: Giuseppe Asaro (10523267, 879796), Salvatore Cassata (10560790, 869034)

15/05/19

INDICE

Introduzione

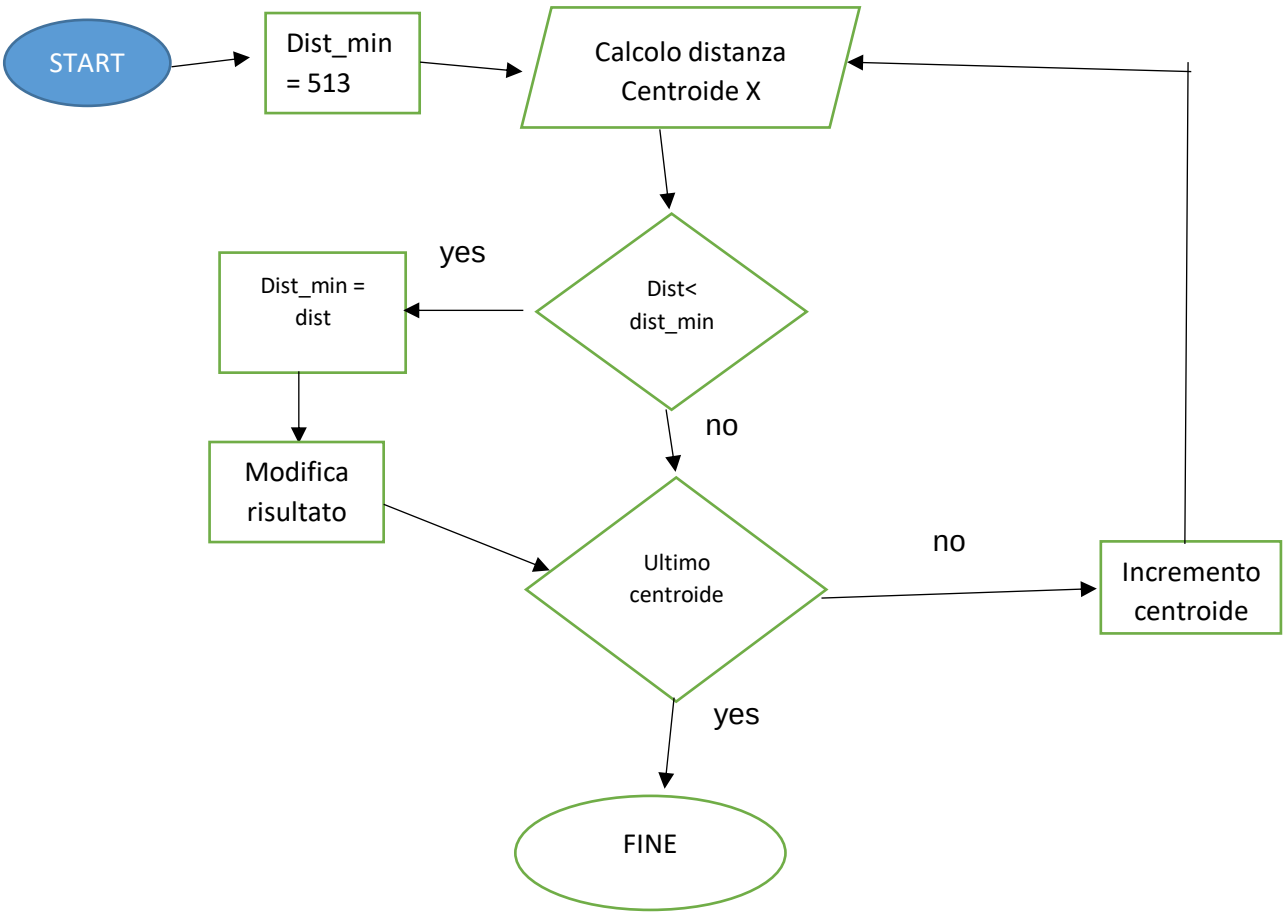
L'obbiettivo per la Prova Finale per l'A.A. 2018/19 consisteva nell'implementazione di un componente HW descritto in VHDL che, una volta fornite le coordinate di un punto appartenente ad uno spazio bidimensionale e quelle di N punti detti CENTROIDI, appartenenti a tale spazio, fosse in grado di valutare a quale/i dei centroidi il punto risultasse più vicino (Manhattan distance).

A tal scopo si è scelto di realizzare una macchina a stati finiti, eseguendo poi un'opportuna batteria di test per valutare la robustezza dell'algoritmo.

Nei paragrafi successivi verranno esposti, e commentate, le scelte procedurali, alcuni dei test eseguiti e delle possibili ottimizzazioni alla macchina.

Descrizione algoritmo

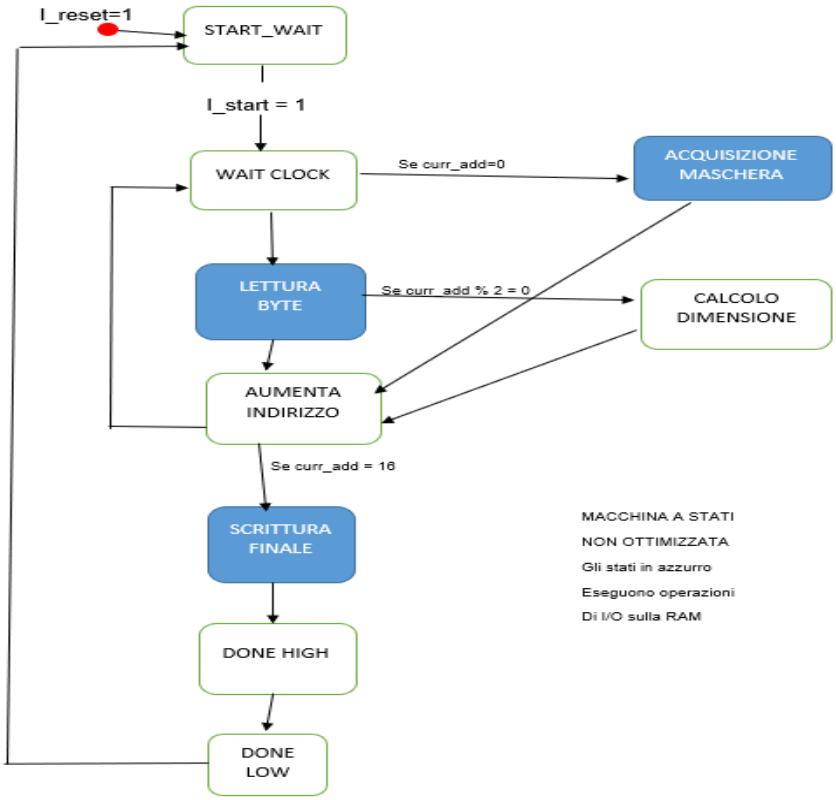
Di seguito è mostrato il flow chart dell'algoritmo utilizzato: dopo l'acquisizione da memoria delle coordinate dei punti, qui omessa, viene calcolata la distanza dei centroidi segnalati "attivi" rispetto al punto sentinella; di volta in volta viene poi modificato un risultato temporaneo, che al termine dell'elaborazione verrà trasmesso come output.



Schema degli stati

Ricevuto il segnale di *reset* il componente entra nel primo stato, **START_WAIT**, dove resta fino al segnale di *start*. Lo stato successivo, **WAIT_CLOCK**, ha il compito di far passare un ciclo di clock per acquisire i dati dalla memoria e, secondo l'indirizzo corrente, esso rimanda o all'*acquisizione della maschera* o alla *lettura del byte*. Quindi nel caso in cui il *curr_add* sia uguale a 0 allora la WAIT_CLOCK avrà come stato successivo **ACQUISIZIONE_MASCHERA**, dove in un array verranno settati a "1" gli indici dei centroidi attivi (ovverosia rispetto ai quali eseguire poi l'effettivo corpo dell'algoritmo), se invece il *curr_add* è diverso da 0 allora lo stato successivo alla WAIT_CLOCK sarà **LETTURA_BYTE**, che ha il compito di acquisire la coordinata del punto o quella del centroide sentinella a seconda dell'indirizzo, inoltre se abbiamo acquisito la coordinata "y" di un centroide si passa allo stato **CALCOLA_DISTANZA** che *calcola la distanza* del centroide dal punto; nel caso tale distanza sia minore della corrente distanza minima l'indice del centroide viene considerato "1" nel risultato finale temporaneo. A questo punto sia ACQUISIZIONE_MASCHERA, sia CALCOLA_DISTANZA, sia LETTURA_BYTE hanno come stato successivo **AUMENTA_INDIRIZZO** che procede col *selezionare l'indirizzo della coordinata successiva da prelevare*. Nel caso in cui siano stati analizzati tutti i centroidi, si passa allo stato **SCRITTUR_FINALE** che *effettua la scrittura in memoria del risultato*. Infine, gli stati **DONE_HIGH** e **DONE_LOW** alzano il segnale *o_done* per un ciclo di clock, dopodiché il componente ritorna in attesa del segnale di start nello stato START_WAIT.

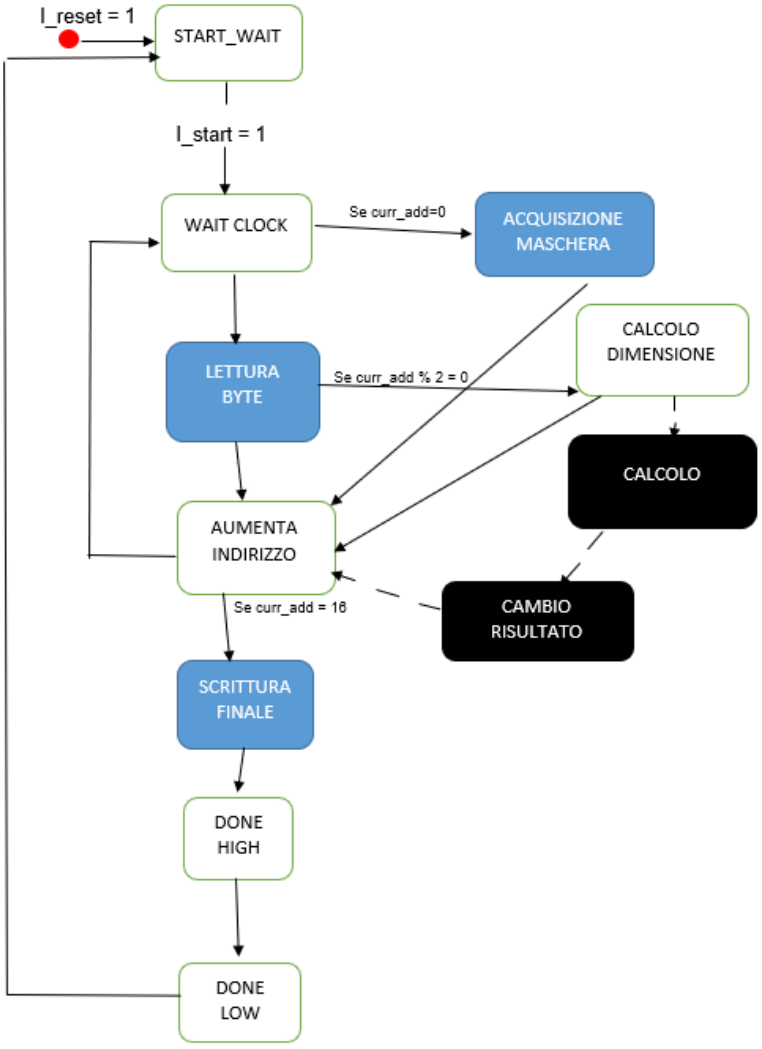
La macchina così come descritta è stata implementata utilizzando come architecture del componente un processo che ha come sensitivity list il clock generato dal test bench e il segnale di i_reset per poter ottenere reset asincroni. Gli stati sono implementati come case di un costrutto switch, una variabile provvede a tenere memoria dello stato corrente. Poiché il processo è avviato ad ogni commutazione dello stato del clock l'istruzione *rising_edge(i_clk)* provvede alla sincronizzazione sul fronte di salita.



POSSIBILI OTTIMIZZAZIONI

Tra le possibili ottimizzazioni è sicuramente possibile innanzitutto ridurre il numero di accessi in memoria: nell'algoritmo da noi implementato, dopo aver acquisito la maschera di attivazione, vengono lette dalla memoria le coordinate di tutti gli otto centroidi, tuttavia una vasta ed interessante casistica presenta delle maschere in cui diversi centroidi vengono messi a 0, essendo dunque inattivi non sarebbe necessario andar ad accedere alla memoria per leggere le loro coordinate; a tal fine sarebbe sufficiente inserire un controllo (un semplice "if") sull'indice della maschera relativo al centroide di interesse e nel caso in cui questo sia 0 andare direttamente allo stato successivo, saltando LETTURA_BYTE.

Nel caso in cui si dovesse far fronte ad un segnale di clock più stringente sarebbe interessante provar a rendere più rapido lo stato CALCOLA_DISTANZA, dove oltre a venir calcolata la distanza di un centroide rispetto al punto sentinella, viene modificato il risultato temporaneo con una serie di "if" a cascata. Per provare a migliorare leggermente la situazione si è pensato di dividere lo stato CALCOLA_DISTANZA in due sottostati, ovvero uno in cui venga effettivamente calcolata la distanza, per poi andare in *wait* e solo al ciclo di clock successivo modificare il risultato temporaneo.



TESTING

Una volta testato il componente con il test bench fornito ed aver constatato il corretto funzionamento in *Behavioral* e *Post-Synthesis Functional.* Sono stati cercati alcuni casi di test che stressassero di più il componente tra i quali:

- Maschera generica e centroidi generici;
- Centroidi con coordinate tutte coincidenti tra loro => maschera output=maschera input;
- Tutti i centroidi equidistanti dal punto sentinella => maschera output=maschera input;
- Centroidi con coordinate generiche e maschera con tutti zero => maschera output=tutti zero;
- Centroidi con coordinate generiche e maschera con tutti uno => maschera output=uno nei centroidi effettivamente più vicini alla sentinella, zero altrimenti;
- Tutti i centroidi coincidono tra loro e con il punto (distanza=0 per ogni centroide)=> maschera output=maschera input;
- Maschera con unico uno in corrispondenza del centroide più distante dal punto sentinella => maschera output=maschera input;
- Maschera con tutti uno e i tre centroidi più vicini posti ad ugual distanza dal punto sentinella => maschera output= 3 “uno” in corrispondenza dei tre centroidi a uguale distanza;
- Maschera con tutti uno, centroidi disposti a due a due sui quattro angoli e punto sentinella al centro => maschera output= tutti uno;
- Modifica del clock period;
- Inserimento segnali di reset.

Con i test effettuati si è cercato di coprire più casi limite possibili in modo da poter testare a fondo la struttura logica dell’algoritmo; si è inoltre tentato di abbassare il clock period per osservare come un clock eccessivamente basso non permetta alla macchina di portare a termine la parte di programma relativa agli stati più onerosi.