Laurea Triennale in Ingegneria Informatica

Progetto Finale di Reti Logiche

Implementazione componente Hardware in VHDL

[Fahed Ben Tej fahed.ben@mail.polimi.it]

Codice persona: 10549663 Matricola: 871416

Codice Identificativo Insegnamento: 051228
Prova Finale - Progetto di Reti Logiche
Anno Accademico 2018/19 Scaglione: Prof. William Fornaciari Tutor: Davide Zoni

Contents

1

Introduzione

1.1 Breve descrizione del problema

Sia dato uno spazio bidimenzionale 256x256 e N punti di cui K validi. Dato un punto generico P, si vogliono trovare i punti ad esso più vicini.
I dati sono forniti da una memoria esterna indirizzata a byte. Nell'indirrizo 0 vi è una maschera di N bit che definisci i punti validi tramite una codifica one-hot. Negli indirizzi k e $k+1$ sono presenti le coordinate X e Y del k-esimo centroide, dove $k \in \{1, 2, .., 15\}$. L'indirizzo 17 e 18 contengono le coordinate di P.
La macchina deve scrivere nell'indirizzo 19 la soluzione.
L'obbiettivo di questo progetto è quello di descrivere un componente hardware tramite linguaggio VHDL che possa risolvere il dato problema.
Per maggiori dettagli si è invitati a consultare il documento di specifica.

1.2 Soluzione ad alto livello

La Manhattan distance tra due punti $p1(x_1; y_1)$ e $p2(x_2; y_2)$ viene definita come:

$$dist(p1, p2) = \mid x_1 - x_2 \mid + \mid y_1 - y_2 \mid$$

Si osservi che due o più punti possono avere la stessa distanza dal punto P. La soluzione consiste quindi nell'insieme :

$$S = \{p_i : \nexists k \, dist(P, p_k) < dist(P, p_i)\}$$

L'idea è quella di iterare su tutti i punti dati e tenere conto della minima distanza fino a quel momento. Sia d_{min} tale distanza e venga inizializzata a M sufficientemente grande.
Per ogni punto p_k si possono verificare tre casi:
- $dist(P, p_k) > d_{min}$ allora sicuramente $p_k \notin S$
- $dist(P, p_k) = d_{min}$ allora aggiungo p_k ad S
- $dist(P, p_k) < d_{min}$ allora $S = \{p_k\}$

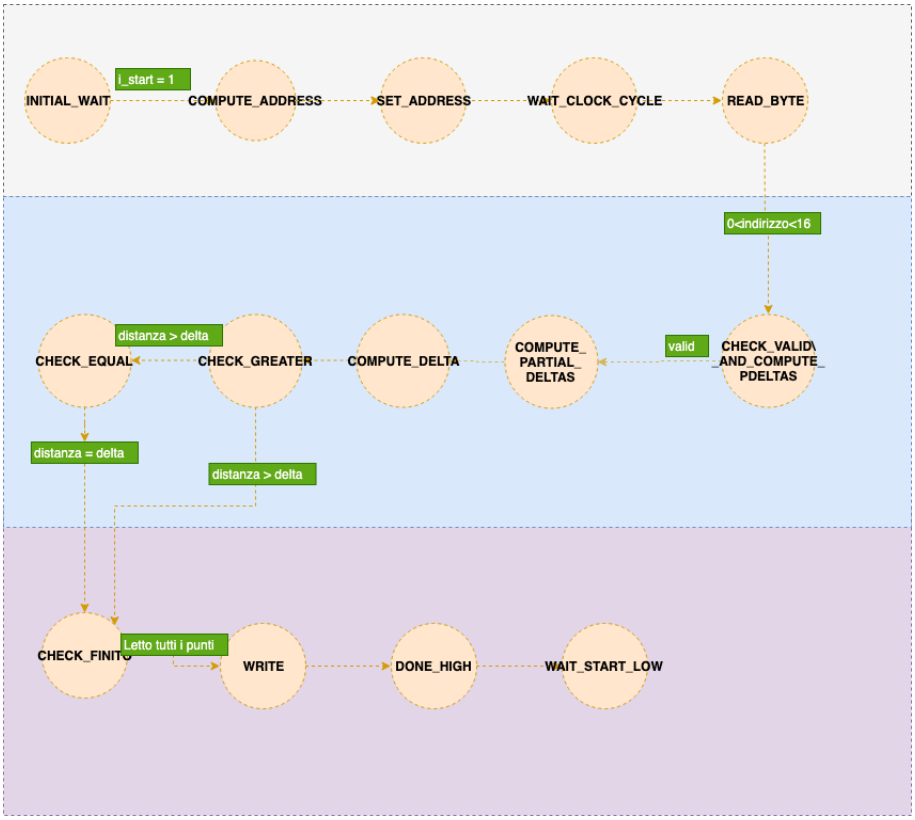
Nel nostro caso S ha una dimensione massima di N, quindi è rappresentabile come una maschera di n bit in cui il k-esimo bit è a 1 se $p_k \in S$. Le operazioni di aggiunta ed eliminazione di un elemento si riducono quindi ad una somma e differenza .

2

Schema degli stati e Test

2.1 Schema degli stati

Al fine di implementare la soluzione, si é deciso di ricorrere alla costruzione di una semplice FSM (Finite State Machine). Ogni stato esegue delle operazioni elementari su delle celle di memoria e decide lo stato seguente. Nella figura sottostante vi é diagramma degli stati che descrive tale FSM.



2.2 Discussione

La computazione nella FSM é divisibile in quattro fasi:
- Attesa iniziale e Lettura
- Calcolo distanza
- Check finali e scrittura soluzione

La FSM, dopo aver ricevuto il segnale di start alto, entra nella fase di lettura. In tale fase, vengono letti la maschera di validità e il punto P in esame. Viene inizializzata una variabile $d_{min} = \infty$
In seguito, viene letta la coordinata X del punto successivo.
Viene calcolato il delta parziale dell'ascissa tramite la formula:

$$\Delta_x = \mid P_x - X \mid$$

Dopo aver letto la coordinata Y, viene calcolato Δ_y usando la stessa formula.
Una volta calcolato $\Delta = \Delta_x + \Delta_y$, lo si confronto con d_{min}.
Si possono verificare tre casi:
- $\Delta > d_{min}$ allora il punto esaminato non appartiene alla soluzione. Si ritorna allo stato READ_BYTE per leggere la coordinata del punto successivo.
- $\Delta = d_{min}$ allora aggiungo il punto alla maschera della soluzione insieme agli atri punti.
- $\Delta < d_{min}$ allora la maschera della soluzione contiene solo questo punto.
Una volta letti tutti e 16 punti, si scrive la soluzione definitiva e si porta il bit DONE a 1 per poi riportarlo a 0 nel ciclo di clock successivo.

2.3 Test

Per testare questo componente, si é sottoposto al componente sia casi 'ordinari' che casi 'limite'. Un esempio di input 'ordinario' é stato fornito nelle specifiche.
I casi limite studiati sono i seguenti:
- Caso in cui tra i centroidi forniti ci sia il punto P stesso, ovvero la distanza d_{min} sia nulla. Tale caso va a sollecitare la fase del calcolo della distanza. In particolare, verifica la correttezza degli stati $COMPUTE\_PARTIAL\_DELTA$, $COMPUTE\_DELTA$ e $CHECK\_GRATER$.
- Caso in cui tutti i centroidi forniti sono coincidenti, quindi la maschera di uscita è 1111 1111. Tale test va a sollecitare lo stato $CHECK\_EQUAL$. Infatti tale check ha sempre esito positivo.
- Nessun centroide fornito è valido. In tale caso la soluzione è 0000 0000. Tale test va a studiare la robustezza del componente. Infatti il comportamento che deve tenere il componente non é definito esplicitamente nella specifica. Lo stato in esame é $CHECK\_VALID$.

Inoltre, si é studiato il comportamento del componente quando il testbench fornisce un nuovo segnale di start, ovvero il funzionamento 'a regime'.

3

Ottimizzazione

3.1 Ottimizzazione

Come visto nella sezione precedente, esistono dei particolari input che rendono il problema banale e quindi é possibile fornire una soluzione immediatamente. Ad ogni modo, si è deciso di non aggiungere tali controlli sugli input poichè si presuppone che tali input siano 'rari' e che quindi i costi sono maggiori dei benefici. In particolare con l'aggiunta di check sugli input si aumeneterebbe la complessità di alcuni stati. Ciò si tradurebbe in un maggiore consumo di spazio.
Dopo aver aver osservato il diagramma degli stati, si è giunto alla conclusione che alcuni stati si possono aggregare al fine di ridurne il numero.// In particolare si è deciso di unire gli stati $CHECK\_VALID$ e $COMPUTE\_PARTIAL\_DELTA$. Questa operazione è giustificata dal fatto che i due stati operano su strutte dati differenti, quindi non si generano problemi di interferenza tra le operazione di lettura e scrittura. Si crea così lo stato $CHECKVALID\_AND\_COMPUTE\_PDELTAS$.
In generale, durante il progetto si è cercato di seguire il principio del riutizzo degli stati.
Per esempio, inizialmente si erano previsti uno stato per la lettura delle coordinate X e uno stato per la lettura delle cordinate Y. Si é optato per rendere la lettura una operazione parametrica rispetto a dei flag. In questo modo si é riuscito a diminuire il numero di stati, ma si é incrementato la complessità degli stati $COMPUTE\_ADDRESS$ e $READ\_BYTE$. L'inserimento di diverse formule if ha prodotto un discreto aumento nel numero di componenti logici utilizzati. Si è giunti così alla seguente FSM: