POLITECNICO
MILANO 1863

Prova finale (Progetto di Reti Logiche) 2018/2019

Autore:

Daniele Comi 10528029 844102

Professore e Tutor:

William Fornaciari

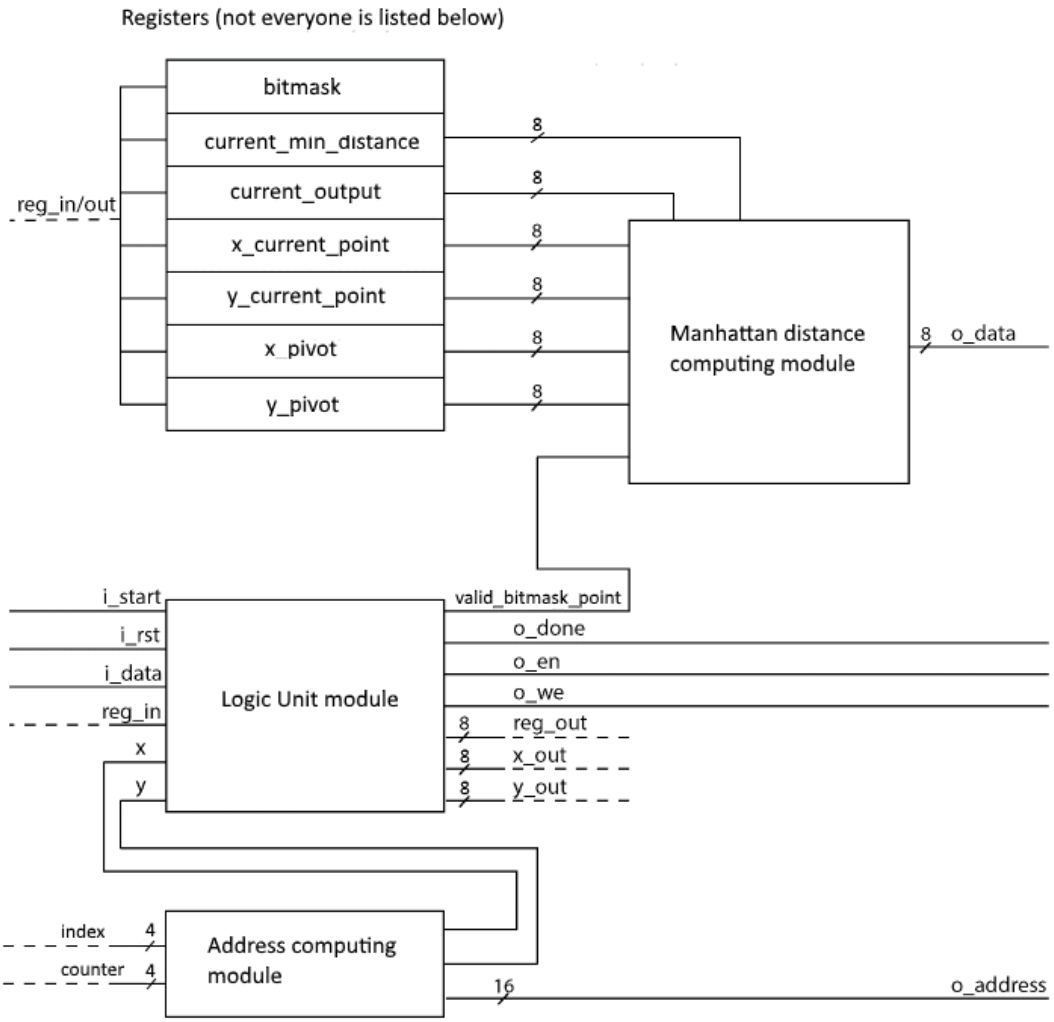
Davide Zoni

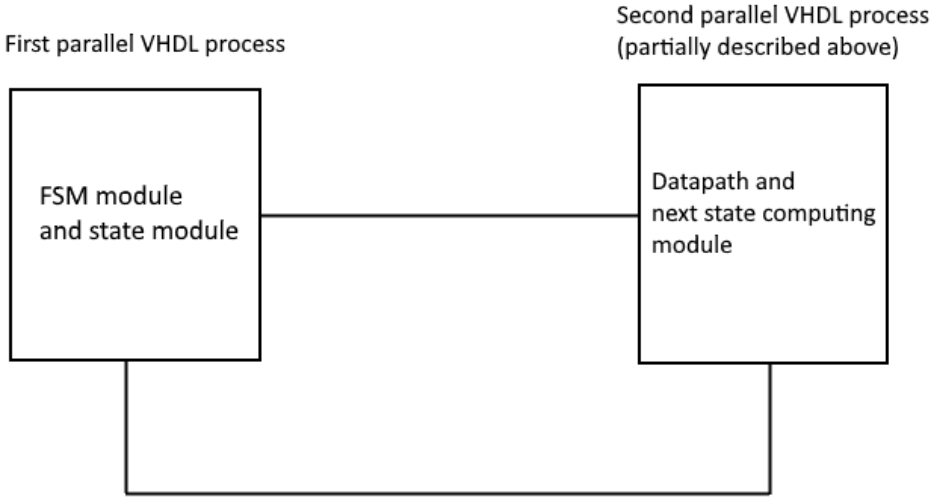
INDICE

INTRODUZIONE

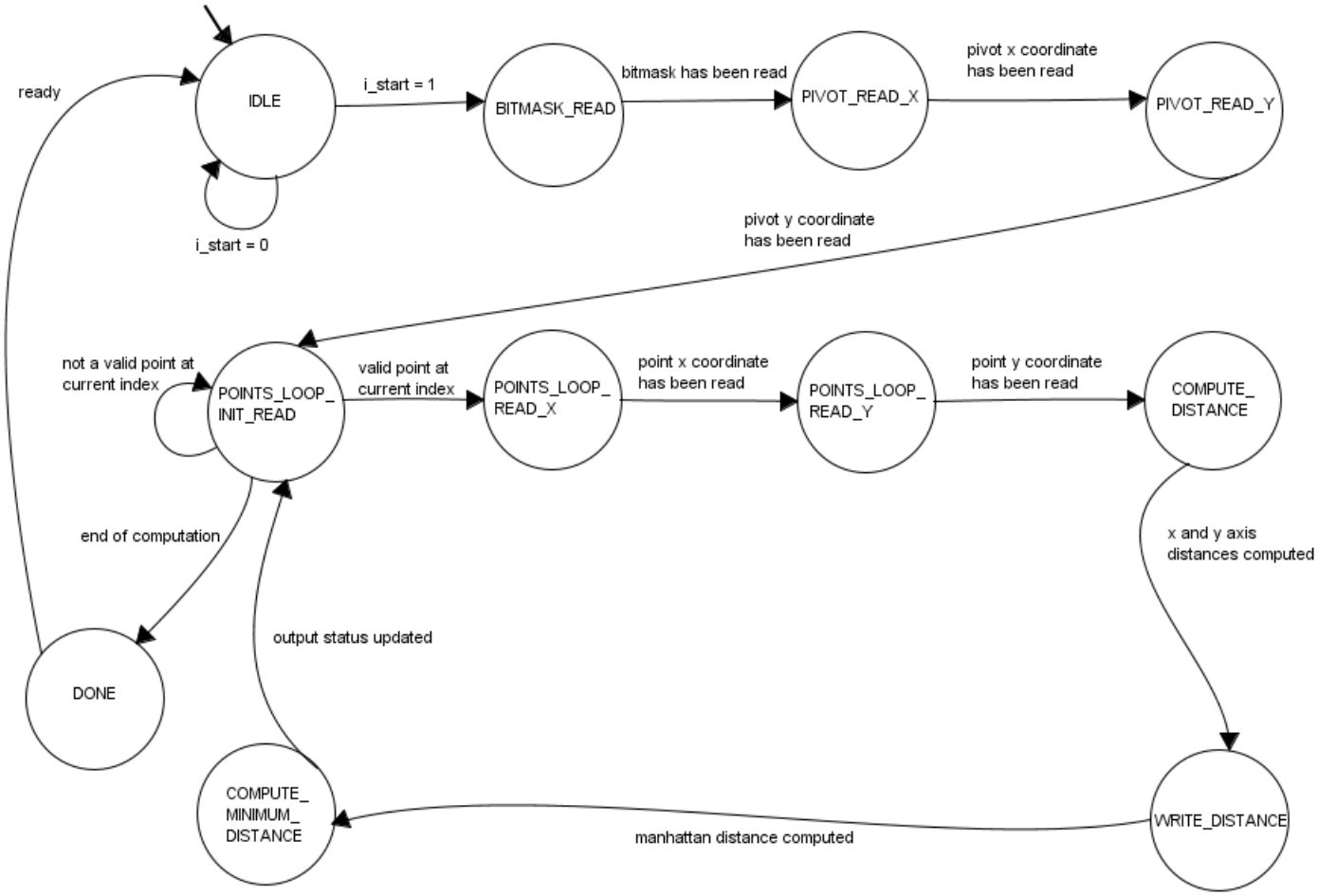
L'obiettivo di questo report è quello di discutere le scelte progettuali intraprese per creare un componente che, dato in ingresso una serie di coordinate di punti ed un punto pivot, riesca a calcolare la minima distanza di Manhattan tra i vari punti e il pivot in ingresso.

SCHEMA DEL COMPONENTE





SCHEMA DELLA MACCHINA A STATI



La macchina a stati implementata è una FSMD a due processi, il suo funzionamento comincia con l'attesa che in ingresso venga ricevuto il comando di avvio attraverso l'innalzamento ad 1 del bit d'ingresso i_start. Se questo non avviene si continua a rimanere nello stato IDLE, così chiamato per indicare, facendo riferimento allo scheduling di una coda di processi, il non far nulla in attesa di un dato evento. Inoltre esso corrisponde allo stato in cui si trova nel caso di reset, ovvero quando i_rst ha valore 1.

Una volta arrivati nello stato BITMASK_READ si procede alla lettura della BITMASK all'indirizzo 0 della memoria RAM. Successivamente si passa alla lettura delle coordinate del punto pivot, agli indirizzi 17 e 18 della RAM, negli stati PIVOT_READ_X e PIVOT_READ_Y.

Nello stato POINTS_LOOP_INIT_READ comincia la lettura del primo degli 8 punti presenti in RAM, sottoforma di coordinate x e y da 8 bit, dall'indirizzo 1 al 16 attraverso i successivi due stati POINTS_LOOP_READ_X e POINTS_LOOP_READ_Y.

Nello stato di COMPUTE_DISTANCE viene calcolata la distanza sui rispettivi assi tra un punto degli 8 presenti e il pivot, successivamente nello stato WRITE_DISTANCE viene calcolata la distanza di Manhattan.

Nello stato COMPUTE_MINIMUM_DISTANCE si passerà al controllare se la distanza precedentemente calcolata è minore o uguale del minor valore di distanza di Manhattan precedentemente calcolata. Nel caso fosse minore solo l'attuale punto degli 8 sarà considerato a minima distanza mentre nel caso sia uguale esso verrà considerato minore assieme agli altri punti di egual distanza di Manhattan dal pivot.

Dopo di che si ritornerà allo stato POINTS_LOOP_INIT_READ controllando se ci sono altri punti di cui dover effettuare una computazione rieseguendo i precedenti passi, oppure se i punti da considerare sono terminati si arriverà allo stato di DONE dove verrà scritto nella RAM all'indirizzo 19 il risultato finale sottoforma di 8 bit contente '1' nella posizione corrispondente del punto, o più, alla minore distanza di Manhattan.

TESTING

La fase di testing è stata effettuata in maniera esaustiva seguendo queste due principali fasi:

- Testing mirato a una o più parti del modulo
- Testing utilizzando varie istanze casuali di casi con una buona distribuzione dei valori

Nella prima fase sono stati effettuati test che andavano a sollecitare le parti più particolari del modulo quali per esempio la somma per il calcolo della distanza di Manhattan nel caso in cui ci sia un overflow fino ad arrivare ad un risultato non più da 8 bit ma bensì da 9 bit nel caso la distanza fosse maggiore uguale a 255. Questa parte è presente nello stato WRITE_DISTANCE che si occuperà del salvataggio in un registro della distanza.

Altri casi particolari presenti sono stati la bitmask letta con tutti gli 8 bit a 0 : in questo caso per come il modulo è stato pensato non ci si dovrebbe mai spostare dallo stato di POINTS_LOOP_INIT_READ perché ognuno degli 8 punti presenti non sono considerati validi per via della bitmask completamente posta a 0.

Ulteriori casi interessanti su cui il testing è stato necessario è il caso in cui la RAM dia in input tutti gli 8 punti alla medesima distanza dal pivot, per fare questo è stato deciso di inglobare questo caso di test nel caso di test dove tutta la RAM avesse ogni singolo bit presente in ogni singola sua parola da 8 bit posto a '1'. In questo modo oltre a testare il caso in cui ogni punto sia alla stessa distanza dal pivot venivano testati contemporaneamente i casi in cui il pivot e uno o più punti avessero le medesime coordinate e il caso in cui, avendo la RAM posta completamente ad '1' in ogni singolo bit, si avesse la complessità temporale peggiore in termini di cicli di clock.

Un ulteriore caso di test simile al precedente è stato considerare che ogni punto avesse stessa distanza dal pivot ed in particolare che questa distanza generasse un overflow andando ad essere espressa a 9 bit. Questo è ottenibile ponendo la bitmask completamente a 1 e le coordinate del pivot agli indirizzi 17 e 18 tutti posti a valore a 0 e le coordinate di tutti gli altri 8 punti tutti posti a valore 1 tra gli indirizzi 1 e 16, o viceversa.

Nella seconda fase del testing si ha avuto un approccio non più mirato ai casi sensibili ma bensì al provare il numero maggiore di casi di test possibili con una buona distribuzione dei valori in RAM in maniera casuale.

Per fare ciò è stato necessario scrivere un tool per generare un numero N a scelta di test bench sottoforma di file chiamati *tb_FSM_<i-esimo>.vhd,* ogni file si distingueva per come la RAM fosse inizializzata. E' stato scelto di utilizzare Java per la scrittura di questo tool ottimizzato attraverso il parallelismo perché la creazione di centinaia di migliaia o milioni di casi di test e la loro successiva scrittura in memoria su disco come file di testo è per un calcolatore moderno un calcolo che può richiede diversi minuti di tempo.

Successivamente è stato deciso l'utilizzo di una libreria chiamata VUnit perché ha i principi dell'unit testing ma sviluppata per VHDL, così da poter automatizzare la simulazione ed il testing del modulo. Utilizzando VUnit attraverso uno script in Python è stato possibile coprire più di duecentomila casi di test. Tutti passati con esito positivo, lo scopo era andare a stimolare il modulo in modalità sempre differenti.

Anche in questo caso la simulazione che comportava la trasduzione di VHDL in C++ e successiva compilazione via GCC è piuttosto oneroso quindi è stato necessario parallelizzare anche lo script Python. Purtroppo Vivado di Xilinx non è supportato da VUnit, così è stato deciso di utilizzare GHDL : simulatore freeware ed open source il cui utilizzo risultava compatibile con la libreria di unit testing. Per una migliore gestione di questo enorme quantitativo di casi di test è stato in più optato in alcuni casi l'utilizzo di Sigasi Studio XPRT di cui la compagnia Sigasi, produttore di questo software, è stata così gentile da offrirmi la licenza in via completamente gratuita del loro prodotto di punta a pagamento per questo progetto.

Tutti i casi di test sono stati effettuati sia in pre che in post sintesi per accertarsi del funzionamento del componente.

OTTIMIZZAZIONI

Per poter abbassare il costo computazionale e la complessità temporale del modulo sono state fatte alcune ottimizzazioni.

La prima è stata ovviamente il non leggere tutti i punti ma solo quelli che avessero il corrispettivo bit della bitmask posto a 1 così da non perdere tempo e cicli di clock nella lettura dei punti da non considerare.

La seconda ottimizzazione è stata il non leggere tutti i punti in un solo blocco e poi effettuare il calcolo della distanza di Manhattan di ciascuno di essi ma bensì è stato optata la lettura di una coppia di coordinate, solo se il corrispettivo punto avesse 1 nella sua posizione nella bitmask come spiegato nella prima ottimizzazione, e successivamente il calcolo della distanza di Manhattan dal pivot viene fatto immediatamente. Se la distanza risultasse minore della precedente considerata minore, la parola in output viene resettata con tutti i bit a 0 tranne il bit del corrispettivo punto che viene messo ad 1 perché risulti l'attuale a minore distanza. Se la distanza risultasse uguale alla corrente distanza minore, nella parola di output viene posto il corrispettivo bit del corrente punto ad 1 così che l'altro o gli altri precedenti punti rimanessero al contempo i minori.
Se invece la distanza risultasse maggiore, non viene effettuata alcuna scrittura sulla parola di output.

Questa scelta è stata preferita rispetto ad un calcolo parallelo delle 8, o meno, distanze di Manhattan dal pivot e calcolando l'output come OR degli 8, o meno, output temporanei perché andavano ad incidere sull'utilizzo della memoria richiedendo un totale di 14 byte in più rispetto a quelli già utilizzati. Tra l'altro questa scelta avrebbe portato ad una soluzione più complessa e più costosa temporalmente per via dell'aggiornamento concorrente della distanza minore globale tra gli 8 punti diversi calcolati parallelamente.

Una terza e più banale ottimizzazione è il salvataggio delle coordinate del pivot durante l'intera computazione della maschera di output per non sprecare cicli di lettura: 8 bit per la coordinata x e 8 bit per la y. Stessa ottimizzazione è stata fatta localmente al ciclo di lettura e alla locale computazione della distanza di un punto dal pivot attraverso un registro da 16 bit che contiene le coordinate del corrente punto. Queste due ottimizzazioni permettono di sprecare 2 cicli di CLOCK per ogni lettura in meno per singolo stato, così da evitare la continua lettura dalla RAM delle coordinate dei punti che andrebbero a sprecare cicli di clock.

Come complessità temporale nel caso peggiore è stato calcolato che la FSMD ci metterà nel caso peggiore in totale 54 cicli di clock ovvero, considerando che il clock in questo progetto va ad una frequenza di 10 MHz, quindi con un periodo di 100 ns, il modulo ci metterà 5.4 µs a terminare la computazione nel caso peggiore.