Anno Accademico 2018/2019

Prova Finale (Progetto di Reti Logiche)

Prof William Fornaciari

Andrea Calici

Codice Persona: 10490117

Numero Matricola: 843651

**Indice**

**INTRODUZIONE**....................................3

**DESCRIZIONE DEI SEGNALI**.........................3

**DESCRIZIONE DEGLI STATI DELLA MACCHINA A STATI FINITI**..........................................4

**CASI DI TEST**....................................5

**SCELTE ARCHITETTURALI**...........................6

**INTRODUZIONE**

Dato uno spazio bidimensionale quadrato (256x256), obiettivo del progetto è quello di realizzare un componente che, assegnate le coordinate di un punto dello spazio, calcola la distanza minima tra lo stesso e 8 centroidi, ovvero punti appartenenti a tale spazio. In output dovrà dare un vettore di 8 bit, i cui bit sono tutti a ‘0’, eccetto che quelli relativi ai centroidi con distanza minima del punto da valutare, settati a ‘1’.

Il componente che può soddisfare la specifica è una macchina a stati finiti che definisce come prima cosa all’interno dell’*architecture*, il tipo ***state*** definito dall’utente:

20. type state is (RESET, START, READ\_MASK, READ\_X, READ\_Y, EVALUATE\_MASK, CALC\_DISTANCE, EVALUATE\_DISTANCE, READ\_CENTROID\_X, READ\_CENTROID\_Y, WRITE\_MEM);

che svolge la funzione di creare uno stato per ogni possibile stato della macchina. Successivamente vengono definiti 11 segnali, approfonditi in seguito. La maggior parte di essi sono *std\_logic\_vector*, ovvero vettori di bit da 8 o 9 bit. Per alcuni di essi invece è stato scelto il tipo *bit* perché più adatto agli **if-else**, in quanto in grado di assumere solamente i valori ‘0’ e ‘1’, andando così a definire esattamente 2 casi.

Successivamente definisce un process sensibile al segnale ***i\_clock***. Infatti ad ogni cambiamento di ***i\_clock*** il processo è attivato, altrimenti rimane inattivo:

40. process(i\_clk)

Il processo quindi si apre con un **if** che permette l’esecuzione solo nel caso in cui ci sia un fronte di discesa del clock (analogo a if(i\_clk’event and i\_clk = ’0’ ) then):

42. if falling\_edge(i\_clk) then

Successivamente si incontra un **if-else** che gestisce i 3 casi principali:

* ***i\_rst*** *= ‘1’*: tutti i segnali vengono azzerati e la macchina è pronta per ricevere il primo segnale di start, passando allo stato ***START***;
* ***i\_start*** *= ‘1’ and CURRENT\_STATE = START*: viene settato il primo indirizzo di memoria, ovvero quello relativo alla maschera di ingresso;
* casi rimanenti: se la macchina entra nell’**else** è perché la computazione può cominciare. Tramite uno statement **case** vengono analizzati tutti i possibili stati della macchina, e in base allo stato in cui la stessa si trova vengono eseguite determinate operazioni.

**DESCRIZIONE DEI SEGNALI**

Oltre ai 9 segnali definiti dalla specifica e appartenenti alla entity, l’architecture definisce altri 16 segnali necessari all’elaborazione:

* *current\_state*: di tipo *state*, definisce lo stato corrente in cui si trova la macchina;
* *mask*: contiene la maschera di ingresso da cui comincia l’elaborazione;
* *x*: contiene la prima coordinata del punto da valutare, relativa all’indirizzo 17 di memoria;
* *y*: contiene la seconda coordinata del punto da valutare, relativa all’indirizzo 18 di memoria;
* *centroid\_x*: contiene la prima coordinata del centroide attualmente analizzato;
* *centroid\_y*: contiene la seconda coordinata del centroide attualmente analizzato;
* *temp\_x*: contiene il valore assoluto della differenza tra la x del punto da valutare e la x del centroide analizzato;
* *temp\_y*: contiene il valore assoluto della differenza tra la y del punto da valutare e la y del centroide analizzato;
* *distance*: salva il valore della distanza di Manhattan minima calcolato fino a quell’istante di esecuzione;
* *new\_distance*: per ogni centroide valutato, contiene il valore della distanza di Manhattan tra lo stesso e il punto da valutare (contenuto agli indirizzi 17 e 18 di memoria);
* *address*: salva l’indirizzo di memoria da cui leggere nel successivo ciclo di clock. Evolve nello stesso modo di ***o\_address***, ma è necessario nei casi in cui debbano essere svolte operazioni su di esso, dato che ***o\_address***, essendo un segnale di uscita, non può stare “a destra” di un assegnamento;
* *centroid\_number*: tiene conto dell’attuale numero di centroide analizzato. Da specifica saranno definiti 8 centroide, di conseguenza ***centroid\_number*** avrà un valore compreso tra 0 e 7 che verrà incrementato ogni volta che verrà analizzato un nuovo centroide. Quando il suo valore diventa maggiore di 7, allora l’esecuzione della macchina deve terminare con la scrittura in memoria;
* *distance\_boolean*: è un segnale di tipo bit, utilizzato all’interno dello stato ***EVALUATE\_DISTANCE***, necessario all’aggiornamento dei segnali ***new\_distance*** e ***distance***;
* *new\_distance\_boolean*: è un segnale di tipo bit, utilizzato all’interno dello stato ***EVALUATE\_DISTANCE***, settato a ‘1’ nel caso in cui venga trovata una nuova distanza minore o uguale all’attuale distanza di Manhattan salvata in ***distance***, per garantire l’aggiornamento di ***o\_data***;
* *o\_data\_boolean*: è un segnale di tipo bit, utilizzato all’interno dello stato ***EVALUATE\_DISTANCE***, necessario all’aggiornamento del segnale ***o\_data***;
* *centroid\_boolean*: è un segnale di tipo bit, utilizzato all’interno dello stato ***EVALUATE\_DISTANCE***, necessario all’aggiornamento del segnale ***centroid\_number***.

**DESCRIZIONE DEGLI STATI DELLA MACCHINA A STATI FINITI**

Il componente hardware che implementa la specifica richiesta è una macchina a stati finiti che si compone di 11 stati totali:

* RESET: è lo stato iniziale in cui si trova la macchina;
* START: è lo stato di partenza della macchina, da cui può cominciare l’esecuzione. Appena il segnale ***i\_start*** viene settato a ‘1’, viene assegnato il primo indirizzo di memoria che conterrà la maschera di ingresso che andrà a definire quale centroide analizzare.
* READ\_MASK: salva il valore della maschera e setta l’indirizzo successivo, relativo alla prima coordinata del punto da valutare;
* READ\_X: salva il valore della prima coordinata del punto da valutare e setta l’indirizzo successivo, relativo alla seconda coordinata dello stesso;
* READ\_Y: salva il valore della seconda coordinata del punto da valutare e setta l’indirizzo successivo, relativo alla prima coordinata del primo centroide;
* EVALUATE\_MASK: contiene due strutture **if-else** di controllo. Il primo **if** valuta il numero di centroidi analizzati. Infatti dopo aver valutato tutti e 8 i centroidi l’esecuzione dovrà terminare passando direttamente allo stato ***WRITE\_MEM*** che scriverà in memoria. Il secondo **if** valuta se è necessario analizzare il centroide corrente in base alla maschera d’ingresso. In caso positivo passa allo stato ***READ\_CENTROID\_X***, altrimenti passa al centroide successivo incrementando di 2 l’indirizzo corrente, e aumenta di 1 il segnale ***centroid\_number*** che tiene conto del numero di centroide valutato;
* READ\_CENTROID\_X: salva il valore della prima coordinata del centroide corrente e setta l’indirizzo successivo, relativo alla seconda coordinata dello stesso;
* READ\_CENTROID\_Y: salva il valore della seconda coordinata del centroide corrente e setta l’indirizzo successivo, relativo alla prima coordinata del prossimo centroide o del punto da valutare, nel caso in cui siano stati già analizzati tutti i centroidi;
* CALC\_DISTANCE: attraverso due **if-else** salva nei due segnali ***temp\_x*** e ***temp\_y*** il valore assoluto della differenza tra le x e le y del punto da valutare e del centroide analizzato in quel momento;
* EVALUATE\_DISTANCE: attraverso più strutture di controllo **if-else** esegue tutte le possibili ramificazioni dell’esecuzione:

1. Viene calcolato il valore della distanza tra il punto da valutare e il centroide analizzato e salvata nel segnale ***new\_distance***;
2. Se la nuova distanza è minore di quella attualmente minima, oppure è la prima distanza ad essere calcolata viene assegnata al segnale ***distance***, e viene “resettato” il valore di ***o\_data*** che conterrà la maschera finale;
3. Viene settato a ‘1’ il bit di ***o\_data*** relativo al centroide con distanza minore dal punto da valutare.
4. Viene incrementato il segnale ***centroid\_number*** che tiene conto del numero del centroide valuato.

* WRITE\_MEM: setta l’indirizzo di memoria ***o\_address*** a 19, dove poi verrà scritto ***o\_data***. Successivamente alza ad ‘1’ i segnali ***o\_en***, ***o\_we*** e ***o\_done*** che, rispettivamente, abilitano la comunicazione con la memoria, la scrittura in memoria e la fine dell’elaborazione.

**CASI DI TEST**

Per testare l’effettivo funzionamento del componente sono stati svolti diversi test, in modo da sollecitarne i casi limite. Tutti i test sono stati realizzati partendo dal file “esempio testbench.vhd” fornito dai docenti sulla piattaforma Beep, modificandone alcune parti.

* *test\_bench\_2*: nelle righe 86 e 88 viene aumentato il ritardo da 2 ns a 5 ns per testare il funzionamento del componente anche in caso di ritardo

86. mem\_o\_data <= mem\_i\_data after 5 ns;

88. mem\_o\_data <= RAM(conv\_integer(mem\_address)) after 5 ns;

* *test\_bench\_3*: la maschera di ingresso viene modificata da 185 (10111001) a 189 (10111101) per garantire la valutazione del terzo centroide, prima ignorata. In questo modo la distanza di Manhattan minima sarà pari a 1 e mi aspetto che la maschera di uscita sarà uguale a 4 (00000100)

24. signal RAM: ram\_type := (0 => std\_logic\_vector(to\_unsigned( 189 , 8)),

112. assert RAM(19) = std\_logic\_vector(to\_unsigned( 4 , 8)) report "TEST FALLITO" severity failure;

* *test\_bench\_4*: questo test, dopo aver ricevuto il segnale ***i\_start*** a ‘1’, parte con l’esecuzione e dopo 55 cicli di clock, resetta il segnale facendo ricominciare l’esecuzione dallo stato iniziale. L’importanza di questo test consiste nel valutare la robustezza del componente in situazioni in cui, nonostante la richiesta della specifica sia già stata soddisfatta, ma non venga settato a ‘1’ il segnale ***o\_done***, riesca a reagire ad un reset ripartendo con l’esecuzione

104. wait for c\_CLOCK\_PERIOD \* 55;

106. tb\_rst <= '1';

107. wait for c\_CLOCK\_PERIOD;

108. tb\_rst <= '0';

109. wait for c\_CLOCK\_PERIOD;

110. tb\_start <= '1';

111. wait for c\_CLOCK\_PERIOD;

* *test\_bench\_5*: vengono modificati la maschera d’ingresso ponendola uguale a 128 (10000000) in modo che venga valutato solamente l’ultimo centroide, le coordinate del punto da valutare, ovvero gli indirizzi 17 e 18 di memoria, ponendo il loro valore a 0, e infine il valore delle coordinate dell’ultimo centroide, poste agli indirizzi 15 e 16 di memoria, settando il loro valore a 255. In questo modo, verrà valutato solamente l’ultimo centroide quindi durante l’esecuzione la variabile ***distance*** rimarrà uguale a “00000000” dato che non dovrà essere considerato nessun centroide ad eccezione di quello finale. Giunti a questo punto il valore della distanza diventerà 510, dato che sono stati scelti due punti posti agli estremi della matrice, motivo per cui è stato necessario settare i segnali ***distance*** e ***new\_distance*** come vettori a 9 bit (per contenere fino al valore massimo di distanza di 510) e non a 8 bit come tutti gli altri segnali

23. signal RAM: ram\_type := (0 => std\_logic\_vector(to\_unsigned( 128 , 8)),

38. 15 => std\_logic\_vector(to\_unsigned( 255 , 8)),

39. 16 => std\_logic\_vector(to\_unsigned( 255 , 8)),

40. 17 => std\_logic\_vector(to\_unsigned( 0 , 8)),

41. 18 => std\_logic\_vector(to\_unsigned( 0 , 8)),

111. assert RAM(19) = std\_logic\_vector(to\_unsigned( 128 , 8)) report "TEST FALLITO" severity failure;

* *test\_bench\_6*: la maschera di ingresso viene settata a 0 (00000000) e di conseguenza il comportamento atteso implica che la maschera di uscita sia uguale a 0 (00000000), non essendoci centroidi da valutare. Scopo del test è quello di valutare la robustezza del componente nel caso in cui non ci siano centroidi da valutare

23. signal RAM: ram\_type := (0 => std\_logic\_vector(to\_unsigned( 0 , 8)),

111. assert RAM(19) = std\_logic\_vector(to\_unsigned( 0 , 8)) report "TEST FALLITO" severity failure;

* *test\_bench\_7*: la maschera d’ingresso viene settata a 255 (11111111) e tutti i centroidi vengono posti coincidenti al punto da valutare. In questo modo la maschera d’uscita attesa sarà 255 (11111111) e durante tutta l’esecuzione ***distance*** avrà valore “000000000”.
* *test\_bench\_8*: la maschera d’ingresso viene settata a 255 (11111111) e tutti i centroidi vengono posti ad un valore uguale, in modo che saranno tutti equidistanti dal punto da valutare. La maschera d’uscita attesa è dunque 255 (11111111).
* *test\_bench\_9*: a seguito del primo segnale di start, impone l’attesa di 10 cicli di clock per poi dare il segnale di reset, ed esegue questo procedimento per 3 volte, per testare la robustezza del componente

113. tb\_rst <= '1';

114. wait for c\_CLOCK\_PERIOD;

115. tb\_rst <= '0';

116. wait for c\_CLOCK\_PERIOD;

117. tb\_start <= '1';

118. wait for c\_CLOCK\_PERIOD \* 10;

**SCELTE ARCHITETTURALI**

La scelta di una macchina stati è stata fatta per la sua compatibilità rispetto alla specifica. Infatti la suddivisione delle operazioni si sposa perfettamente alla suddivisione degli stati che, avendo anche dei nomi esplicativi della loro funzione, mostrano esattamente a che punto si trovi la computazione e cosa venga svolto in quel determinato momento.

L’utilizzo dei segnali di tipo *bit* insieme agli **if-else** è stato necessario per garantire i corretti assegnamenti dei segnali riferiti ai dati e agli indirizzi. Infatti, a causa della concorrenza del vhdl, due istruzioni consecutive in cui uno stesso segnale si trova a destra di un assegnamento e a sinistra di un altro assegnamento non portano al risultato che porterebbero se le due istruzioni fossero sequenziali. Infatti la maggior parte degli **if-else** fanno sì che si entri prima nell’**if**, venga fatto un primo assegnamento rimanendo nello stesso stato e successivamente, passato un ciclo di clock ed effettuato l’assegnamento richiesto, si entri nell’**else** per eseguire il successivo.

L’utilizzo del segnale **address**, spesso coincidente ad **o\_address** dato dalla specifica è volto alla necessità di aggiornare, in determinate situazioni il valore di **o\_address** incrementando di 1 il suo valore, cosa impossibile da fare senza un segnale di supporto, in quanto **o\_address** è un segnale *out* e di conseguenza non può stare alla destra di un assegnamento.