POLITECNICO
MILANO 1863

Prova Finale
(Progetto di Reti Logiche)
2018/2019

Sez. Prof. William Fornaciari - Tutor Davide Zoni

Chiudinelli Andrea	**845407**	**10534129**
Capra Paolo	**844235**	**10490184**

Indice

1.Macchina a stati

1.1 Descrizione generale

La macchina a stati che abbiamo implementato legge dalla memoria una maschera di ingresso e le coordinate (x,y) del punto che deve essere valutato. Dopo aver salvato in delle apposite variabili questi dati passa a ricontrollare la maschera appena salvata e ,solo se necessario (1 presente sulla maschera), andrà a controllare le coordinate (x,y) dei centroidi e calcolare la relativa distanza di Manhattan dal punto di riferimento. Tutte queste distanze vengono poi salvate in un vettore e controllata la distanza minore presente su quest'ultimo (possono anche essercene più di una) scrive in memoria la maschera d'uscita risultante.

1.2 Descrizione delle variabili

Abbiamo usato differenti variabili, andiamo ora a descriverle per comprendere il loro utilizzo:

indirizzo_corrente *(std_logic_vector):* viene usata per salvare l'indirizzo della memoria RAM che si vuole avere per leggere/scrivere dati.

distanza_minima *(unsigned):* viene usata per tenere traccia ad ogni controllo di quale sia la distanza di Manhattan minore tra tutte mentre vengono calcolate man mano.

maschera_in, maschera_out *(std_logic_vector):* queste sono le variabili dove salviamo la maschera iniziale e la maschera finale che andremo a scrivere in uscita.

x_riferimento, y_riferimento *(unsigned):* queste sono le variabili dove salviamo le coordinate (x,y) del punto di riferimento che dobbiamo controllate.

x_centroide, y_centroide *(unsigned):* queste sono le variabili dove salviamo le coordinate (x,y) dei vari centroidi per poter calcolare la distanza rispetto al punto di riferimento.

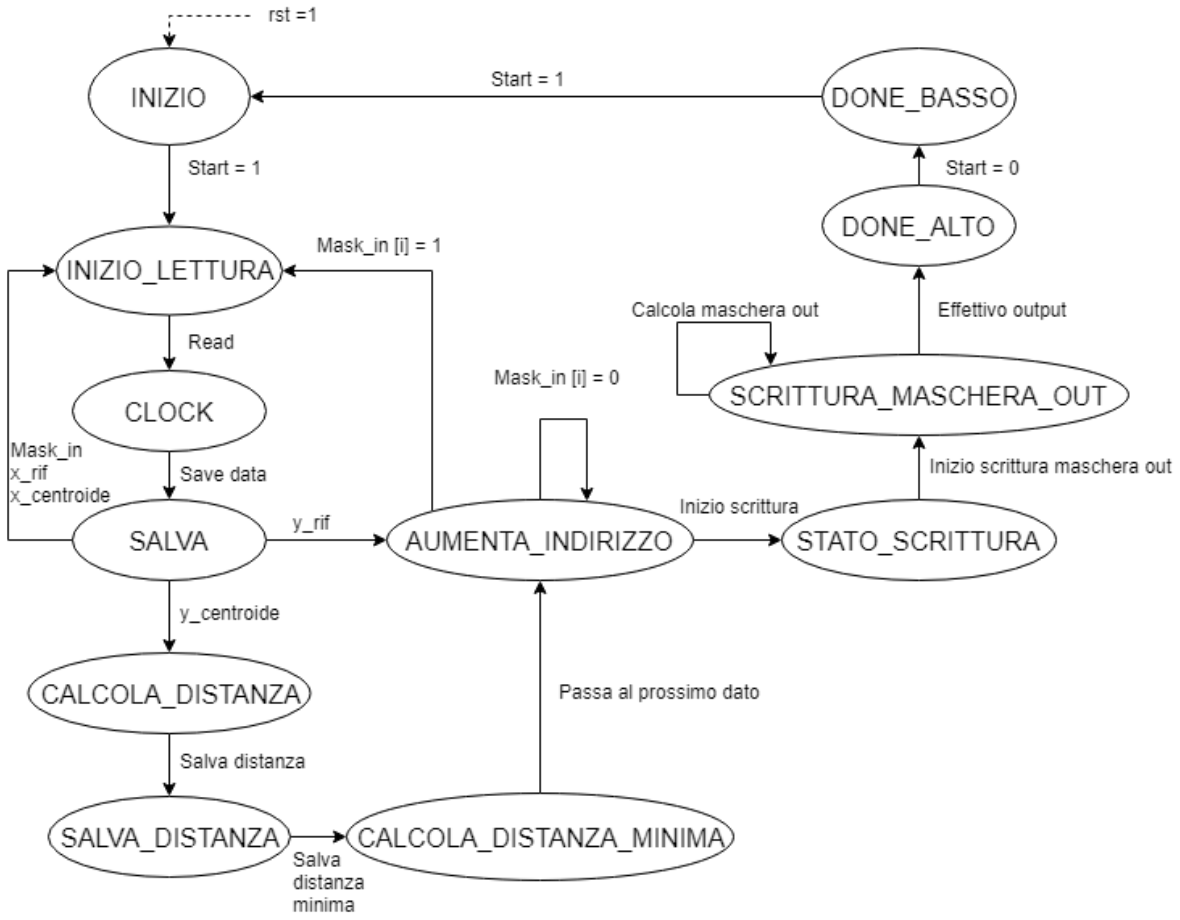
distanza *(unsigned):* viene usata per il risultato di operazione della distanza di Manhattan tra il punto di riferimento e il centroide che si sta controllando in quel momento.

i *(integer):* viene usato come contatore per ciclare sull'array tutte_distanze.

passo *(integer):* questa variabile tiene conto dell'andamento del programma, viene usata nello stato di SALVA per poter capire che cosa fare e dove andare come prossimo stato.

tutte_distanze *(array of unsigned):* tutte le distanze che sono state calcolate per ogni centroide vengono salvate in questo array.

1.3 Descrizione degli stati con relativo diagramma



INIZIO: è lo stato iniziale, in cui la macchina resta in attesa che il segnale i_start venga posto a '1'. Quando ciò accade, delle variabili vengono inizializzate con valori di default e 'indirizzo_corrente' viene inizializzato alla posizione della maschera in ingresso, per poi assegnare INIZIO_LETTURA come prossimo stato.

INIZIO_LETTURA: è lo stato dove vengono settati i segnali per la lettura ('o_en' a '1', 'o_we' a '0' e 'o_address' a 'indirizzo_corrente'). Viene assegnato CLOCK come prossimo stato.

CLOCK: è lo stato dove di attesa di un ciclo di clock, in questo stato viene anche incrementata la variabile passo. Viene assegnato SALVA come prossimo stato.

SALVA: questo stato può essere suddiviso in 5 parti, ad ogni parte è assegnato un differente valore di passo che deve essere controllato:

- Se 'passo' è uguale a '1', viene salvata la maschera di ingresso che è contenuta all'indirizzo 0 della memoria RAM nella variabile 'maschera_in', si porta 'indirizzo_corrente' all'indirizzo 17 di memoria RAM dove è contenuta la x del punto di riferimento. Viene assegnato INIZIO_LETTURA come prossimo stato.

- Se ‘passo’ è uguale a ‘2’, viene salvata la x del punto di riferimento in ‘x_riferimento’ aggiungendo un bit davanti a tutto per poter lavorare con 9 bit, si incrementa di ‘1’ la variabile ‘indirizzo_corrente’. Viene assegnato INIZIO_LETTURA come prossimo stato.
- Se ‘passo’ è uguale a ‘3’, viene salvata la y del punto di riferimento in ‘y_riferimento’ aggiungendo un bit davanti a tutto per poter lavorare con 9 bit, si riporta a ‘0’ la variabile ‘indirizzo_corrente’. Viene assegnato AUMENTA_INDIRIZZO come prossimo stato.
- Se ‘passo’ è uguale a ‘4’, viene salvata la x del centroide in ‘x_centroide’ aggiungendo un bit davanti a tutto per poter lavorare con 9 bit, si incrementa di ‘1’ la variabile ‘indirizzo_corrente’. Viene assegnato INIZIO_LETTURA come prossimo stato.
- Se ‘passo’ è uguale a ‘5’, viene salvata la y del centroide in ‘y_centroide’ aggiungendo un bit davanti a tutto per poter lavorare con 9 bit, si setta la variabile ‘passo’ a ‘3’ per poter proseguire con il calcolo della distanza sugli altri centroidi. Viene assegnato CALCOLA_DISTANZA come prossimo stato.

CALCOLA_DISTANZA: è lo stato dove viene calcolata la distanza tra il punto di riferimento e il centroide che si è appena salvato. Calcoliamo la distanza come differenza tra le x sommata alla differenza tra le y e assegnamo questo valore alla variabile ‘distanza’. Viene assegnato SALVA_DISTANZA come prossimo stato.

SALVA_DISTANZA: è lo stato dove viene salvata la distanza appena calcolata nell’array ‘tutte_distanze’ alla posizione i. Viene assegnato CALCOLA_DISTANZA_MINIMA come prossimo stato.

CALCOLA_DISTANZA_MINIMA: è lo stato che controlla se la distanza appena calcolata risulta essere minore della distanza minore tra tutte quelle appena calcolate, se sì allora la variabile ‘distanza_minima’ diventa l’attuale ‘distanza’. Viene incrementata la variabile ‘i’ di ‘1’ per poter salvare nella corretta posizione la prossima distanza. Viene assegnato AUMENTA_INDIRIZZO come prossimo stato.

AUMENTA_INDIRIZZO: è lo stato dove viene aumentato ‘indirizzo_corrente’:
- Se la maschera d’ingresso alla posizione ‘i’ è uguale ad 1 allora si va ad incrementare di 1 ‘indirizzo_corrente’ per leggere la prossima x dei centroidi e il prossimo stato diventa INIZIO_LETTURA.
- Se la maschera d’ingresso alla posizione ‘i’ è uguale a 0 allora si va ad incrementare di 2 ‘indirizzo_corrente’ e aumentare ‘i’ di uno per saltare questo centroide. Il prossimo stato sarà di nuovo AUMENTA_INDIRIZZO.
- Se invece ‘i’ è maggiore di 8 (la lunghezza della maschera) allora vuol dire che i centroidi da leggere sono finiti, la ‘i’ viene resettata e ‘indirizzo_corrente’ viene fissato a 19. Il prossimo stato è STATO_SCRITTURA.

STATO_SCRITTURA: viene utilizzato per settare ‘o_address’ al valore contenuto in ‘indirizzo_corrente’ (19) e per far passare un ciclo di clock prima del calcolo della maschera. Il prossimo stato è SCRITTURA_MASCHERA_OUT.

SCRITTURA_MASCHERA_OUT: è lo stato dove viene calcolata e scritta 'maschera_out', ad ogni iterazione di questo stato viene incrementata 'i' per scorrere la maschera di ingresso e rispetto ad essa e a 'tutte distanze' viene correttamente scritta la maschera di uscita. Vi sono vari casi:

- Se 'maschera_in' alla posizione 'i' è uguale a 0, allora anche su 'maschera_out' verrà scritto 0 alla posizione 'i'.
- Se 'maschera_in' alla posizione 'i' è uguale a 1 e 'tutte_distanze' alla posizione 'i' è diversa da 'distanza_minima', allora su 'maschera_out' verrà scritto 0 alla posizione 'i'.
- Se 'maschera_in' alla posizione 'i' è uguale a 1 e 'tutte_distanze' alla posizione 'i' è uguale a 'distanza_minima', allora su 'maschera_out' verrà scritto 1 alla posizione 'i'.

Quando si avrà finito di scrivere 'maschera_out' allora verranno settati i segnali per la scrittura ('o_en' a '1', 'o_we' a '1' e 'o_data' a 'maschera_out') e si passa allo stato DONE_ALTO.

DONE_ALTO: stato in cui la macchina pone 'o_done' a 1 e, successivamente, aspetta che il segnale di 'i_start' torni a 0 prima di spostarsi su DONE_BASSO.

DONE_BASSO: Viene posto a 0 il segnale 'o_done' mettendosi in condizione di accettare il segnale 'i_start' uguale 1. Se ciò accade, START viene indicato come stato prossimo, in modo da resettare e far eventualmente ripartire l'esecuzione.

In qualsiasi stato si trovi, la macchina a stati è in grado di accettare il segnale i_rst='1', riportando lo stato prossimo a INIZIO.

2. Casi di test

2.1 Maschera d'ingresso 0

- **Specifica:** categoria di test che prevede una maschera di ingresso pari a "00000000"
- **Obiettivo:** sollecitare il caso in cui nessun centroide debba essere considerato, caso che sollecita lo stato AUMENTA_INDIRIZZO
- **Tempo:** 3550 ns

2.2 Maschera d'ingresso 255

- **Specifica:** categoria di test che prevede una maschera di ingresso pari a "11111111"
- **Obiettivo:** sollecitare il caso in cui tutti i centroidi debbano essere considerati, caso che sollecita lo stato AUMENTA_INDIRIZZO
- **Tempo:** 10750 ns

2.3 Tutti i centroidi uguali

- **Specifica:** categoria di test che prevede tutti i centroidi condividano la stessa posizione
- **Obiettivo:** sollecitare la scrittura della ‘maschera_out’, caso che sollecita lo stato SCRITTURA_MASCHERA_OUT
- **Tempo:** 8950 ns

2.4 Test generici

- **Specifica:** categoria di test che prevede l’utilizzo di valori casuali
- **Obiettivo:** sollecita in modo generico tutta l’esecuzione della macchina.
- **Tempo:** proporzionale al numero di centroidi da considerare

3. Ottimizzazioni

Durante lo sviluppo di questo progetto, abbiamo apportato delle modifiche a parti del nostro codice per poter renderlo più efficiente a livello di tempo di esecuzione e delle dimensioni del componente.

Eccone alcune:

- **Lettura dei centroidi:** Al posto di leggere tutti i centroidi e capire poi di quali tenerne conto oppure no, abbiamo preferito capire prima dalla maschera quali fossero quelli da controllare (quando trovavamo ‘1’ sulla maschera) e abbiamo scartato i casi in cui fosse inutile andare a guardarli. in questo modo abbiamo ridotto notevolmente il tempo di esecuzione.

- **Riduzione degli stati:** Il numero di stati è stato ridotto cercando di condensare i salvataggi dei dati, i cicli di clock e i vari aumenti di indirizzo così che il componente risulti dimensionalmente più piccolo.

Ci potrebbero anche essere ulteriori ottimizzazioni:

- **Riduzione degli stati:** Si potrebbero ridurre ancora di più il numero di stati che abbiamo usato per fare occupare ancora meno spazio al componente.

- **Riduzione delle variabili:** Si potrebbero ridurre il numero di variabili che sono state utilizzate per poter rendere meno pesante il programma.