

Progetto finale di Reti Logiche

Roland Reylander [10539438]

1 Aprile 2019

Indice

| | | |
|----------|--|----------|
| 1 | Specifiche di progetto | 2 |
| 2 | Scelte progettuali | 3 |
| | Macchina a stati | 4 |
| 3 | Risultati dei test | 5 |
| 4 | Risultati della sintesi | 5 |
| | Schema <i>Post-Synthesis</i> | 6 |

1 Specifiche di progetto

Lo scopo del progetto è la realizzazione di un componente hardware usando il linguaggio VHDL che risolve il seguente problema: in uno spazio quadrato di 256x256 vengono posizionati 8 centroidi e si vuole trovare il centroide o i centroidi più vicini ad un punto dato.

Il componente dovrà interfacciarsi con una memoria RAM dalla quale legge i dati di input e scrive il risultato della computazione. La memoria RAM di input contiene la maschera dei centroidi da considerare, le coordinate dei centroidi, le coordinate del punto dal quale calcolare la distanza e un indirizzo dedicato al risultato, cioè la maschera di uscita.

| INDIRIZZO RAM | CONTENUTO |
|---------------|---------------------------------------|
| 0 | maschera dei centroidi |
| 1 | coordinata X del centroide |
| 2 | coordinata Y del centroide |
| ⋮ | ⋮ |
| 17 | coordinata X del punto da considerare |
| 18 | coordinata Y del punto da considerare |
| 19 | maschera di output del risultato |
| ⋮ | indirizzi non utili |

Tabella 1: Rappresentazione della RAM

Per risolvere il problema è necessario leggere la maschera e le coordinate del punto e successivamente leggere dalla RAM i centroidi per poi valutare quelli più vicini. Completate queste operazioni bisogna scrivere nell'indirizzo 19 della memoria la maschera che avrà valore 1 sul bit relativo al centroide, considerando il bit meno significativo a destra e quello più significativo a sinistra.

2 Scelte progettuali

Per affrontare la risoluzione di questo problema è stata pensata una FSM con i seguenti stati:

- **reset**: tutti i signal vengono inizializzati;
- **changeAddress**: in base al valore che assume `cnt` (cioè `readMask`, `readXPoint`, `readYPoint`, `readXCoord` e `readYCoord`) viene aumentato l'indirizzo per la prossima lettura della RAM;
- **waitClock**: la lettura dalla memoria richiede 2 ns e quindi non è immediata. In questo stato si aspetta un ciclo di clock affinché il valore `i_data` assuma il valore desiderato che verrà poi usato nello stato successivo;
- **readData**: in base al valore di `cnt` in questo stato vengono salvati su dei signal dedicati i valori della maschera e delle coordinate del punto. Successivamente verranno usati i signal `xAddress` e `yAddress` per memorizzare le coordinate dei centroidi;
- **calcDistance**: viene calcolata la distanza e memorizzata in `tempDistance`;
- **compareDistance**: la distanza calcolata al punto precedente viene confrontata con `bestDistance` in modo da tenere solo i centroidi più vicini;
- **sendMask**: `o_data` assume il valore del risultato finale, l'indirizzo da mandare alla RAM il 19° e `o_we` viene settato a 1 per la scrittura del risultato;
- **load**: segnale di `o_done` viene messo a 1 per indicare la fine dell'elaborazione e l'avvenuta scrittura del risultato;
- **last**: `o_done` torna a 0 e il componente torna allo stato di **reset** pronto per ricevere il prossimo segnale di start.

Tutto il funzionamento avviene all'interno di un process in cui la lista di sensibilità è composta da `i_rst` e `i_clk` in modo tale da poter tornare nello stato di **reset** quando `i_rst` = 1 e osservare il cambiamento del signal `state` di tipo `stateType` dichiarato all'inizio.

Dopo la lettura della maschera dalla RAM nello stato **changeAddress** viene scandita la maschera dei centroidi da considerare tramite il signal `maskPos` in modo da passare alla lettura delle coordinate dei centroidi in caso il bit sia 1 altrimenti verrà letto il prossimo bit della maschera.

I signal `tempDistance` e `bestDistance` sono entrambi `std_logic_vector(8 DOWNTO 0)` dato che la distanza massima delle coordinate X e Y dei centroidi è $255 + 255 = 510$ e necessita quindi di 8 bit. Inoltre `bestDistance` viene inizializzata nello stato di **reset** con `OTHERS => '1'` cioè 511 che è maggiore della massima distanza possibile dei centroidi e permettendo il corretto funzionamento dell'algoritmo.

Nella figura seguente viene rappresentata la macchina a stati mostrando le condizioni significative per passare dallo stato corrente allo stato prossimo. Il cambiamento di stato avviene al ciclo di clock successivo e inoltre da ogni stato possibile tornare allo stato di **reset** quando `i_rst` = 1 che non viene rappresentato per non appesantire il grafico.

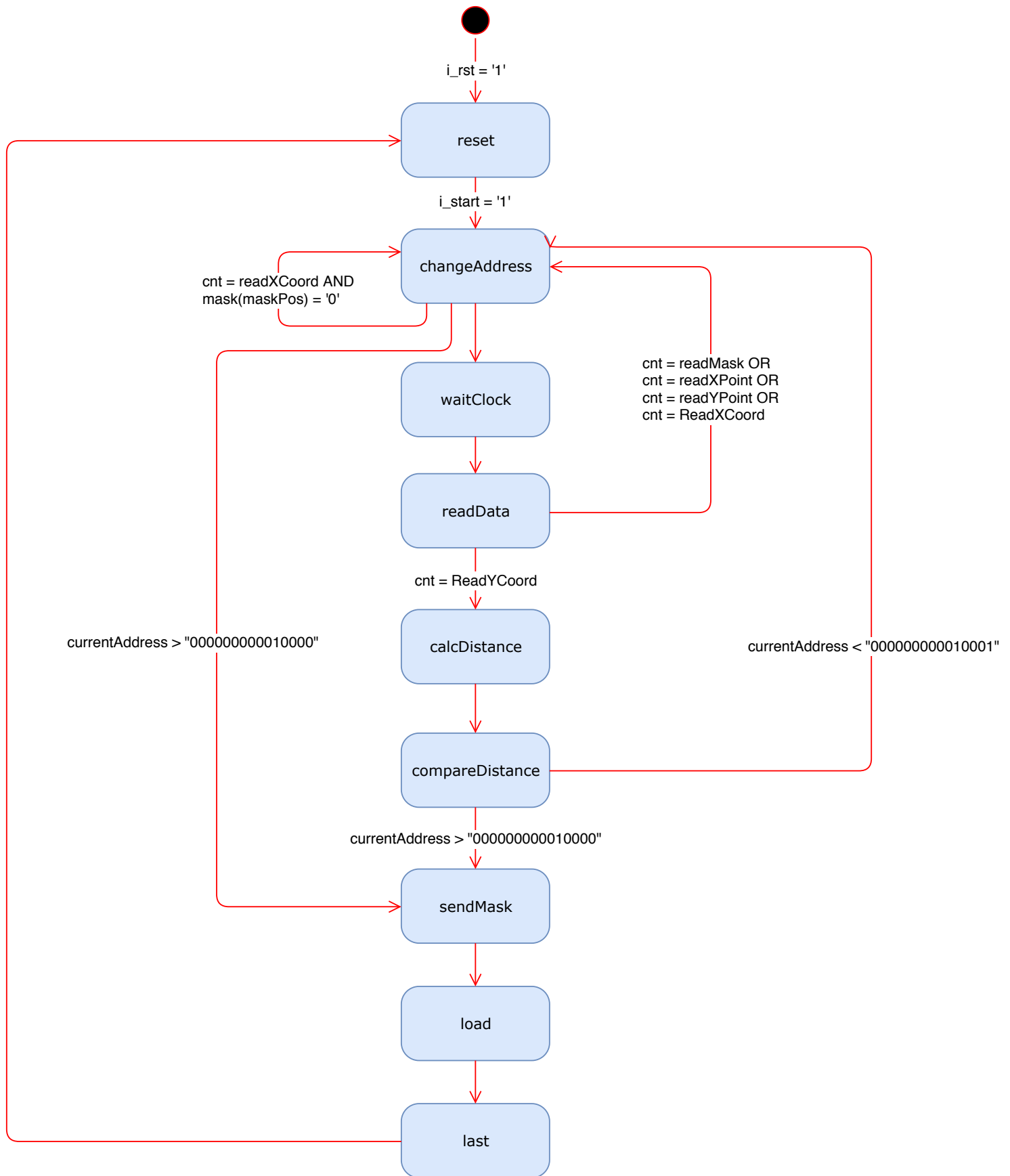


Figura 1: Macchina a stati

3 Risultati dei test

Per controllare il corretto funzionamento del componente sono stati fatti dei test specifici ogniqualvolta si è aggiunto un nuovo stato. Questi hanno permesso di controllare che:

- gli indirizzi della RAM venissero scanditi uno alla volta partendo da 0, proseguendo leggendo gli indirizzi 18 e 19 per le coordinate del punto. Dopodiché leggono le coordinate dei centroidi, saltando eventualmente gli indirizzi delle coordinate i cui centroidi hanno il bit della maschera uguale a 0.
- lo stato `calcDistance` effettivamente calcolasse la distanza giusta, considerando i punti menzionati precedentemente nelle *Scelte progettuali*. Inoltre è stato controllato che la funzione `abs()` inclusa nel package `IEEE.STD_LOGIC_1164.all` fosse anche sintetizzabile, ottenendo quindi questa formula per il calcolo della distanza:

```
tempDistance <= std_logic_vector((ABS(signed('0' & xAddress) - signed('0' & xPoint)))  
+ std_logic_vector((ABS(signed('0' & yAddress) - signed('0' & yPoint))));
```

- nello stato `compareDistance` avvenisse correttamente il confronto tra `tempDistance` e `bestDistance` e il passaggio allo stato `sendMask` dopo il calcolo della distanza dell'ultimo centroide dal punto.
- in `sendMask`, `load` e `last` il risultato venisse passato al signal `o_data` e successivamente settato `o_we` per la scrittura della RAM.

Una volta conclusi i test per verificare la logica del componente si sono aggiunti dei test per il controllo del funzionamento nei casi limite. Il test bench di esempio è stato quindi modificato con i seguenti cambiamenti:

- coordinate casuali e maschere casuali per osservare il funzionamento in generale;
- il punto scelto in (0,0) e i centroidi alla massima distanza cioè (255,255) e il contrario, quindi il punto in (255,255) mentre i centroidi in (0,0). Questo ha permesso di raffinare la formula per il calcolo della distanza.
- la maschera in ingresso con tutti bit a 0 e tutti i bit a 1;
- reset casuali per verificare che torni sempre allo stato di `reset` pronto per iniziare una nuova computazione;

4 Risultati della sintesi

Il componente passa tutti i test menzionati precedentemente incluso il test bench di esempio. Una volta verificato il perfetto funzionamento in *Behavioral Simulation* il componente è stato sintetizzato. Dopo la sintesi il componente è stato sottoposto a tutti i test mostrando risultati sbagliati. Un'analisi attenta ha mostrato che il l'uso di `variabile` al posto di `signal` dava dei ritardi inaspettati. La sostituzione delle variabili e una revisione generale del codice hanno portato a una versione finale funzionante. Infatti il componente viene sintetizzato correttamente e passa tutti i test creati sia in *Behavioral Simulation*, in *Post-Synthesis Funcional* e *Post-Synthesis Timing*. Inoltre, anche se non richiesto, il componente passa gli stessi test anche in *Post-Implementation*.

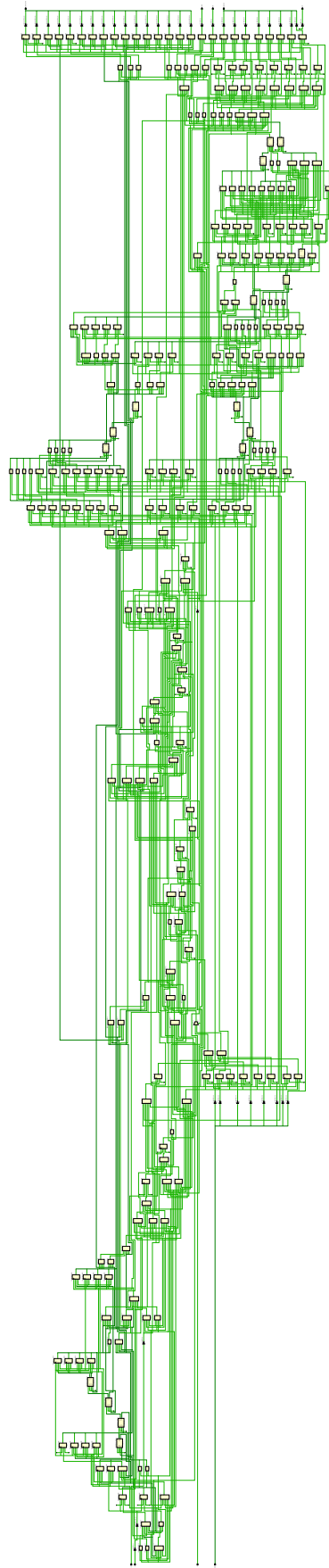


Figura 2: Schema *Post-Synthesis*