Final Project Reti Logiche 2019

Lichinchi Federico, Lorenzi Davide

25 aprile 2019

Indice

Qui di seguito verranno descritti nel dettaglio il problema da risolvere, le scelte progettuali relative alla macchina, i test creati e i relativi risultati e infine i risultati della sintesi.

1 Descrizione del problema

Viene fornito uno spazio bidimensionale definito in termini di dimensione orizzontale e verticale, e siano date le posizioni di N punti, detti "centroidi", appartenenti a tale spazio. Si vuole implementare un componente HW descritto in VHDL che, una volta fornite le coordinate di un punto appartenente a tale spazio, sia in grado di valutare a quale/i dei centroidi risulti più vicino (Manhattan distance). Lo spazio in questione è un quadrato (256x256) e le coordinate nello spazio dei centroidi e del punto da valutare sono memorizzati in una memoria (la cui implementazione non è parte del progetto). La vicinanza al centroide viene espressa tramite una maschera di bit (maschera di uscita) dove ogni suo bit corrisponde ad un centroide: il bit viene posto a 1 se il centroide è il più vicino al punto fornito, 0 negli altri casi. Nel caso il punto considerato risulti equidistante da 2 (o più) centroidi, i bit della maschera d'uscita relativi a tali centroidi saranno tutti impostati ad 1. Degli N centroidi K $\leq$ N sono quelli su cui calcolare la distanza dal punto dato. I K centroidi sono indicati da una maschera di ingresso a N bit: il bit a 1 indica che il centroide è da valutare, mentre il bit a 0 indica che il centroide non deve essere esaminato. Si noti che la maschera di uscita è sempre a N bit e che i bit a 1 saranno non più di K.

2 Specifiche del progetto

Nella progettazione della macchina si sono tenute conto le seguenti specifiche.

2.1 Interfaccia del componente

Il componente da descrivere deve avere un'interfaccia composta dai seguenti segnali:

- `i_clk` è il segnale di CLOCK in ingresso generato dal TestBench;
- `i_start` è il segnale di START generato dal Test Bench;
- `i_rst` è il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START;

- `i_data` è il segnale (vettore) che arriva dalla memoria in seguito ad una richiesta di lettura;
- `o_address` è il segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- `o_done` è il segnale di uscita che comunica la fine dell'elaborazione e il dato di uscita scritto in memoria;
- `o_en` è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- `o_we` è il segnale di WRITE ENABLE da dover mandare alla memoria (=1) per poter scriverci. Per leggere da memoria esso deve essere 0;
- `o_data` è il segnale (vettore) di uscita dal componente verso la memoria.

2.2 La RAM

L'architettura della RAM, su cui si trovano le coordinate dei centroidi e la bit-mask, non viene istanziata nella macchina, dato che è già presente nel Test Bench. Le coordinate e la bit-mask sono salvate nei primi 18 indirizzi di memoria e codificati a 8 bit.

Questi valori non vengono modificati per tutta l'esecuzione del codice. La macchina scriverà il risultato della computazione all'indirizzo 19 della RAM. Ulteriori indirizzi della RAM non possono essere utilizzati per la computazione.

2.3 Il Clock

La macchina deve funzionare con un clock di 100 ns.

2.4 Sintesi e simulazione

Il codice VHDL non deve presentare errori di compilazione. Il componente dev'essere almeno descritto e simulabile correttamente in pre-sintesi. Per raggiungere l'eccellenza nella risoluzione del progetto si richiede che il componente sia sintetizzabile e simulabile anche in post-sintesi.

3 Scelte progettuali

Per la risoluzione del problema assegnato si è deciso di procedere con l'ideazione e l'implementazione di una macchina a stati. La macchina consta 12 stati, numerati da 0 a 10 e uno stato di reset. Per il calcolo della distanza è stata utilizzata la formula della "Manhattan distance".

La macchina è classificabile come macchina di Moore in quanto le uscite dipendono solamente dallo stato corrente.

Onde evitare errori in lettura o in scrittura dei dati durante la Timing Simulation, si è deciso di attendere un interno ciclo di clock prima di ricevere o scrivere il dato richiesto, ottenendo informazioni solo sul fronte di salita.

3.1 Segnali

Come prima cosa, son stati definiti due tipologie di segnale:

- **STATO**: utilizzato per definire gli stati della macchina;
- **D_ARRAY**: utilizzato per definire un array di 8 segnali definiti da 9 bit, in quanto la massima distanza misurabile tra due centroidi è di 510 bit.

I segnali utilizzati dalla macchina sono stati:

- `distances`(D_ARRAY): usato per memorizzare tutte le 8 distanze calcolate;
- `minDistance`(unsigned): usato per memorizzare la distanza minima calcolata;
- `currState`(STATO): usato per memorizzare lo stato in esecuzione;
- `nextState`(STATO): usato per memorizzare il prossimo stato della macchina;
- `prevState`(STATO): usato per memorizzare il prossimo stato della macchina da eseguire successivamente allo stato di lettura del dato dalla RAM;
- `bitMask`(std_logic_vector):usato per memorizzare la bit-mask che la macchina legge dall'indirizzo 0 della RAM;
- `Xc`(signed): usato per memorizzare l'ascissa del punto da valutare, che la macchina legge dall'indirizzo 17 della RAM;

- `Yc`(signed): usato per memorizzare l'ordinata del punto da valutare, che la macchina legge dall'indirizzo 18 della RAM;
- `Xp`(signed): usato per memorizzare l'ascissa del centroide considerato nel calcolo;
- `Yp`(signed): usato per memorizzare l'ordinata del centroide considerato nel calcolo;
- `maskPtr`(integer): usato come contatore per scorrere lungo la bit-mask;
- `cntrPtr`(integer): usato come contatore per cambiare l'indirizzo di lettura della RAM;
- `outMask`(std_logic_vector): usato per memorizzare la maschera di uscita da scrivere nella RAM.

Tutti i segnali sopraelencati, a eccezione dei segnali `currState, nextState, i_start, i_rst, i_data e i_clk`, hanno un omonimo segnale con prefisso next per mantenere memorizzato il loro valore nel ciclo successivo.

3.2 Stati

La macchina è composta dai seguenti stati:

- `RST`: stato di reset, in cui vengono ripristinati al valore iniziale i segnali coinvolti nell'elaborazione. Si entra in questo stato quando `i_rst` va a '1' oppure prima di iniziare la computazione;
- `S0`: stato di idle, in cui si aspetta che i_start vada a '1' per poter cominciare l'elaborazione. L'elaborazione comincia dallo stato `RST`;
- `S1`: in questo stato si assegnano i parametri per la lettura della bit-mask;
- `S2`: in questo stato si salva la bit-mask appena letta e si assegnano i parametri per la lettura dell'ascissa del punto da valutare;
- `S3`: in questo stato si salva l'ascissa del punto da valutare e si assegnano i parametri per la lettura dell'ordinata del punto da valutare;
- `S4`: in questo stato si salva l'ordinata del punto da valutare, si assegnano i parametri per la lettura dell'ascissa del primo centroide;
- `S5`: in questo stato si salva l'ascissa del centroide considerato e si assegnano i parametri per la lettura dell'ordinata del centroide stesso;

- `S6`: in questo stato si salva l'ordinata del centroide considerato, si calcola la distanza di Manhattan e la si assegna a `distance`, tramite l'ausilio di funzioni matematiche (`abs`). Noi abbiamo calcolato la distanza in un'unica equazione, senza aver progettato alcun blocco logico, in quanto questo porterebbe complicazioni nell'implementazione. Una volta calcolata la distanza, la confrontiamo con la `min_distance`. Se `min_distance` risulta maggiore, allora va aggiornato il suo valore. Successivamente `distance` va salvata in `distances`. Se abbiamo letto tutti i centroidi, si va allo stato `S7`, altrimenti si ritorna in `S5` e si continua con la lettura dell'ascissa del centroide successivo;

- `S7`: in questo stato si confrontano tutte le distanze calcolate con la distanza minima. Se la distanza calcolata in un centroide è uguale alla distanza minima, viene assegnato il valore '1' in `outMask` in corrispondenza del centroide considerato, altrimenti viene assegnato il valore '0'. Una volta ricavata la `outMask`, si assegnano i parametri per la scrittura della maschera di uscita all'indirizzo 19 della RAM;

- `S8`: in questo stato si setta il segnale di `o_done` a '1' e si assegnano i parametri per rimanere in attesa che il segnale `i_start` vada a '0';

- `S9`: in questo stato si abilita la RAM per la lettura o per la scrittura del dato;

- `S10`: in questo stato si disabilita l'accesso alla RAM e non si cambia stato finché il segnale `i_start` non va a '0'. Nel momento in cui ciò avviene, si ritorna nello stato di idle.

3.3 Processi

La macchina utilizza due processi in modo asincrono:

- `delta_lambda`: in questo processo viene svolta l'intera computazione, e sono descritti tutti gli stati operativi della macchina;

- `state`: questo processo coordina la propagazione dei segnali e la successione degli stati. In qualsiasi stato di elaborazione della macchina, il processo interviene ripristinando la macchina alle condizioni iniziali se il segnale `i_rst` viene innalzato a '1'.

4 Test creati e relativi risultati

4.1 Test creati

Per verificare la corretta descrizione del componente abbiamo proceduto alla creazione di 5 test bench (oltre a quello fornito dal docente) in cui sono stati testati i limiti della macchina:

1. **tb_bitmask_nulla**: la bit-mask è composta tutta da '0';
2. **tb_centroide_coincidente**: un centroide ha le stesse coordinate del punto dal valutare;
3. **tb_centroidi_tutti_uguali**: tutti i centroidi da esaminare hanno le stesse coordinate;
4. **tb_distanza_massima**: un centroide si trova a distanza massima dal punto da valutare (punto e centroide in angoli opposti della "griglia" della RAM);
5. **tb_distanze_incrementali**: ogni centroide dista a una distanza minore dal punto rispetto al successivo centroide da considerare.

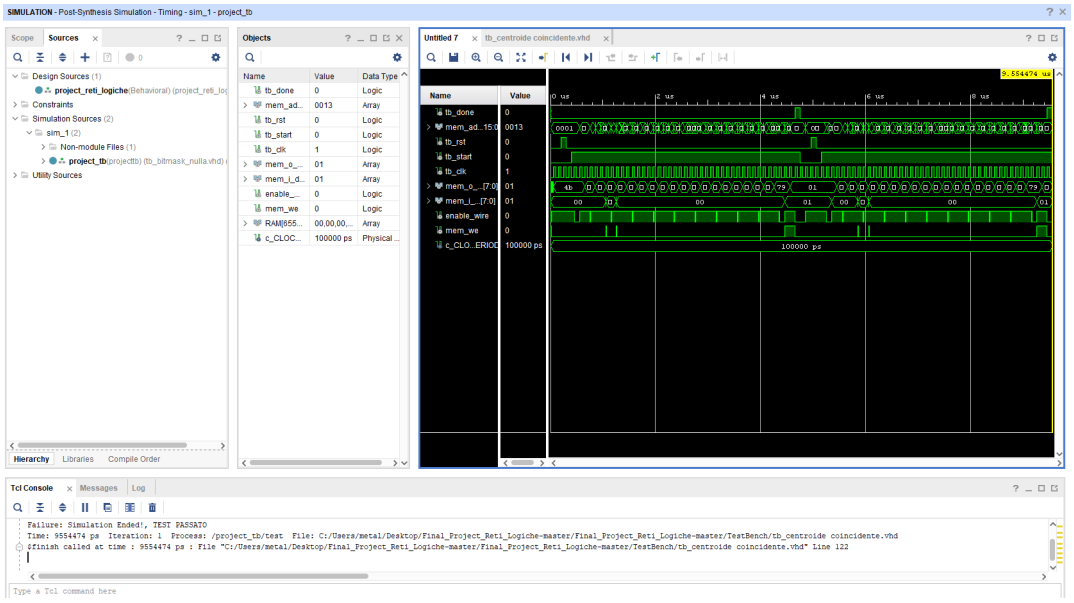
Come modello di riferimento per la creazione dei test bench è stato utilizzato il **tb_salice_reti_logiche** che ha permesso un'efficiente preparazione dei test e una totale assenza di errori nella descrizione della RAM e del componente.

La procedura di testing applicata è stata di black-box in quanto i test sono stati generati a partire dalle specifiche fornite e non dal codice descrittivo VHDL prodotto.

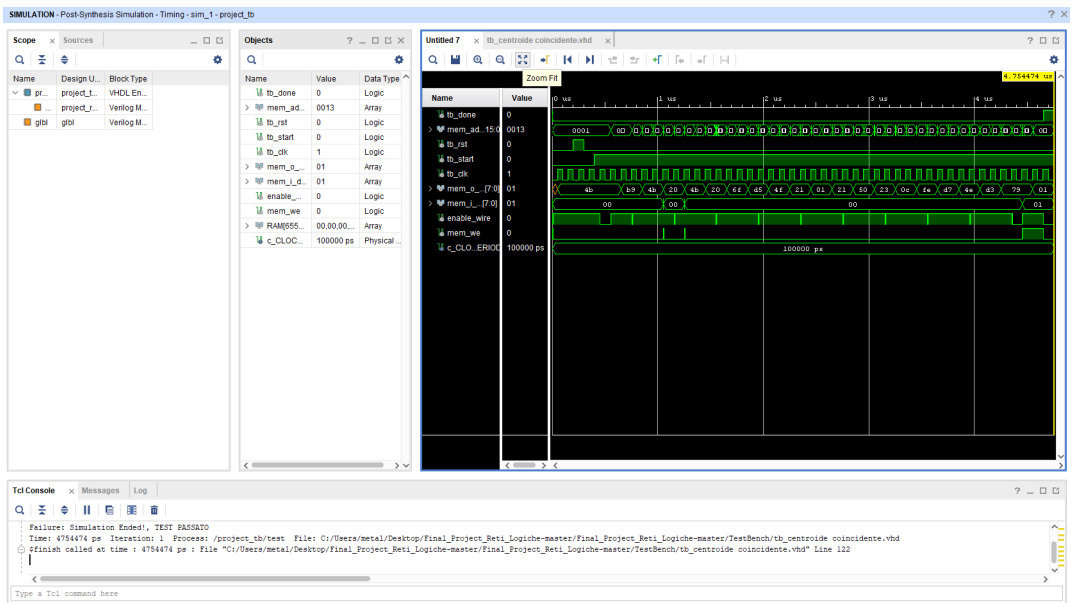
4.2 Risultati ottenuti

Qui sotto sono riportati le schermate delle waveform ottenute e della stampa "TEST PASSATO" nella console al termine della "Run All" del progetto in Timing Simulation, per ogni test bench usato.

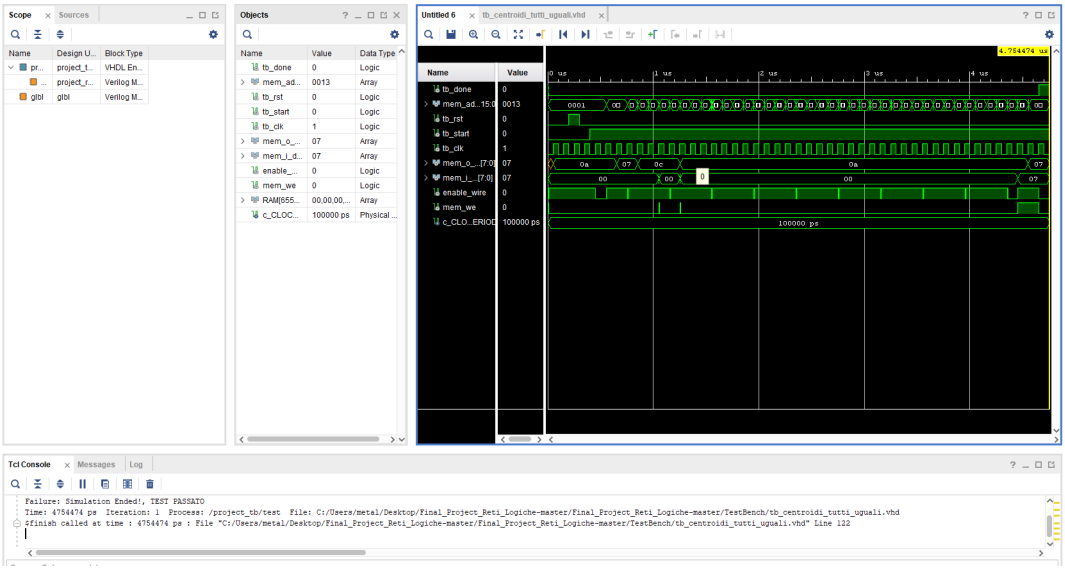
1. **tb_bitmask_nulla**: la bit-mask è composta tutta da '0';



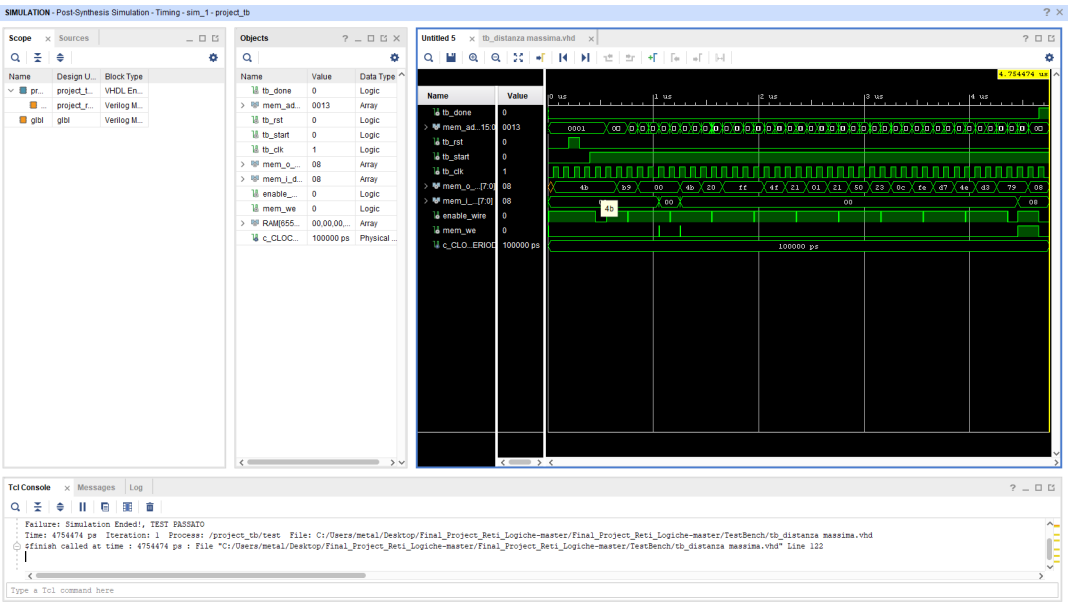
2. **tb_centroide_coincidente**: un centroide ha le stesse coordinate del punto dal valutare;



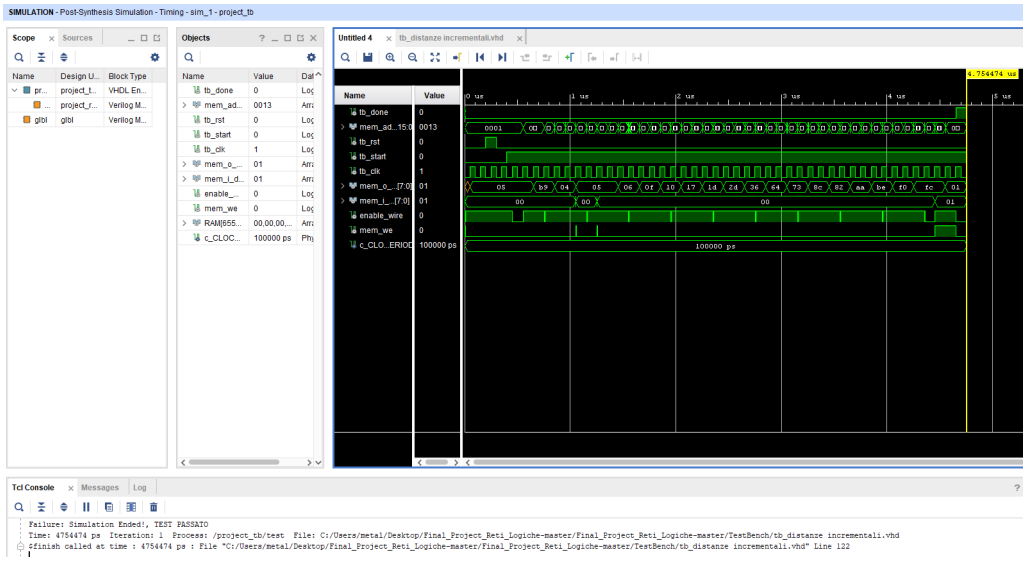
3. **tb_centroidi_tutti_uguali**: tutti i centroidi da esaminare hanno le stesse coordinate;



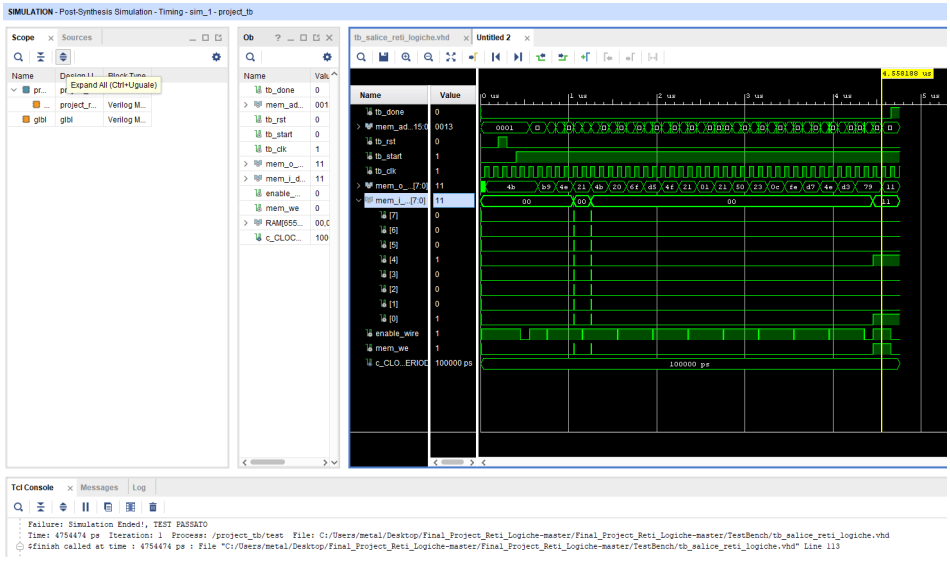
4. **tb_distanza_massima**: un centroide si trova a distanza massima dal punto da valutare (punto e centroide in angoli opposti della "griglia" della RAM);



5. **tb_distanze_incrementali**: ogni centroide dista a una distanza minore dal punto rispetto al successivo centroide da considerare.



6. **tb_salice_reti_logiche**: test bench fornito dal docente.



5 Risultati sintesi

Il componente descritto viene sintetizzato con successo, senza warning significanti che possono compromettere il corretto funzionamento (no inferring latch). Come dimostrato dalla sezione precedente, le simulazioni hanno dato esito positivo sia in pre-sintesi sia in post-sintesi (con timing).