DOCUMENTAZIONE DEL PROGETTO DI RETI LOGICHE

POLITECNICO DI MILANO

ANNO ACCADEMICO 2018/19

DOCENTE: FABIO SALICE

ALUNNO: EVANDRO MADDES

MATRICOLA: 866945; CODICE PERSONA: 10539479

INDICE:

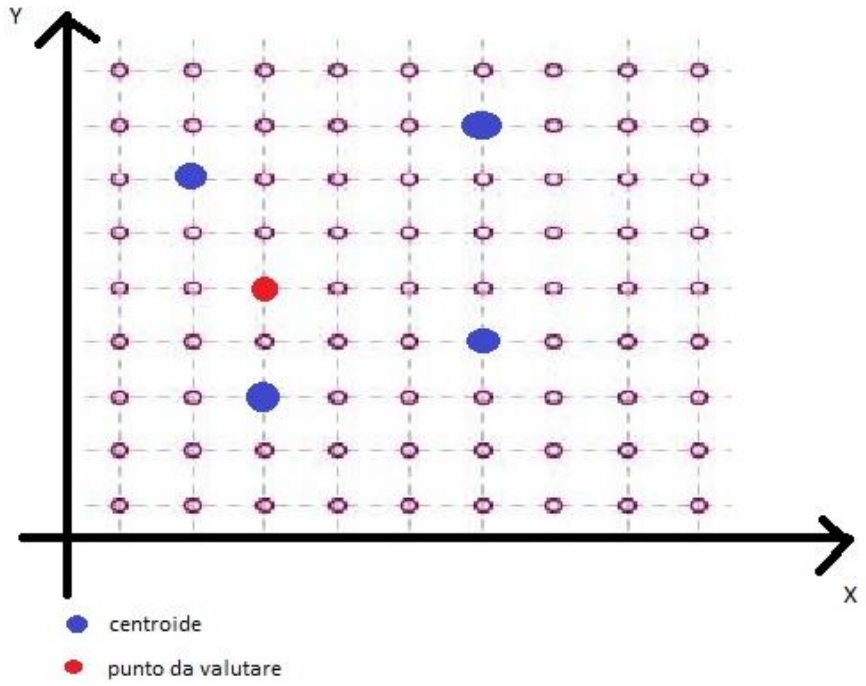
1 .INTRODUZIONE

Il progetto si pone l'obiettivo di implementare un componente hardware descritto in VHDL. Date le coordinate di un punto appartenente ad uno spazio bidimensionale, definito in termini di dimensione orizzontale e verticale, è in grado di valutare a quale/i dei centroidi (punti appartenenti a tale spazio) risulti più vicino (Manhattan distance).

L'FPGA target è xc7a200tfbg484-1.

Le coordinate dei punti, così come il risultato dell'elaborazione, saranno salvati nella memoria.

Si è deciso di impostare il progetto basandosi su un'automa a stati finiti.



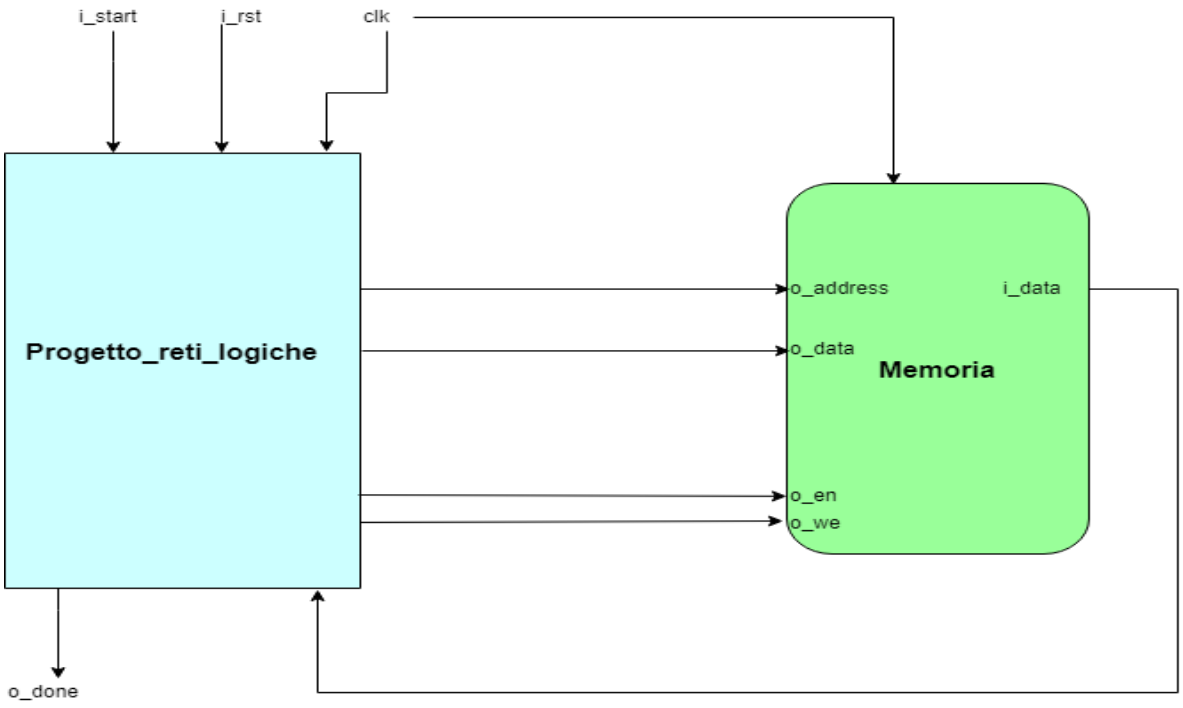
2. ARCHITETTURA

-DESCRIZIONE DEI SEGNALI

Elenco dei segnali presenti nel progetto:

NOME	TIPO DI DATO	DESCRIZIONE
State	State_type	Stato corrente
Current_data	std_logic_vector(7 downto 0)	Vettore contenuto all'indirizzo zero e successivo vettore contenente risultato dell'elaborazione
Current_address	std_logic_vector(15 downto 0)	Indirizzo della memoria da leggere

Distanza_min_	integer	Distanza minima rispetto al punto da valutare
distanze	Array of integer	Distanze dei centroidi rispetto al punto da valutare
i	Integer	Indice per scorrere il vettore
k	Integer	Indice per scorrere l'array
X_centroide	Integer	Coordinata x del centroide corrente
Y_centroide	Integer	Coordinata y del centroide corrente
X_punto_valutare	Integer	Coordinata x del punto da valutare
Y_punto_valutare	Integer	Coordinata y del punto da valutare



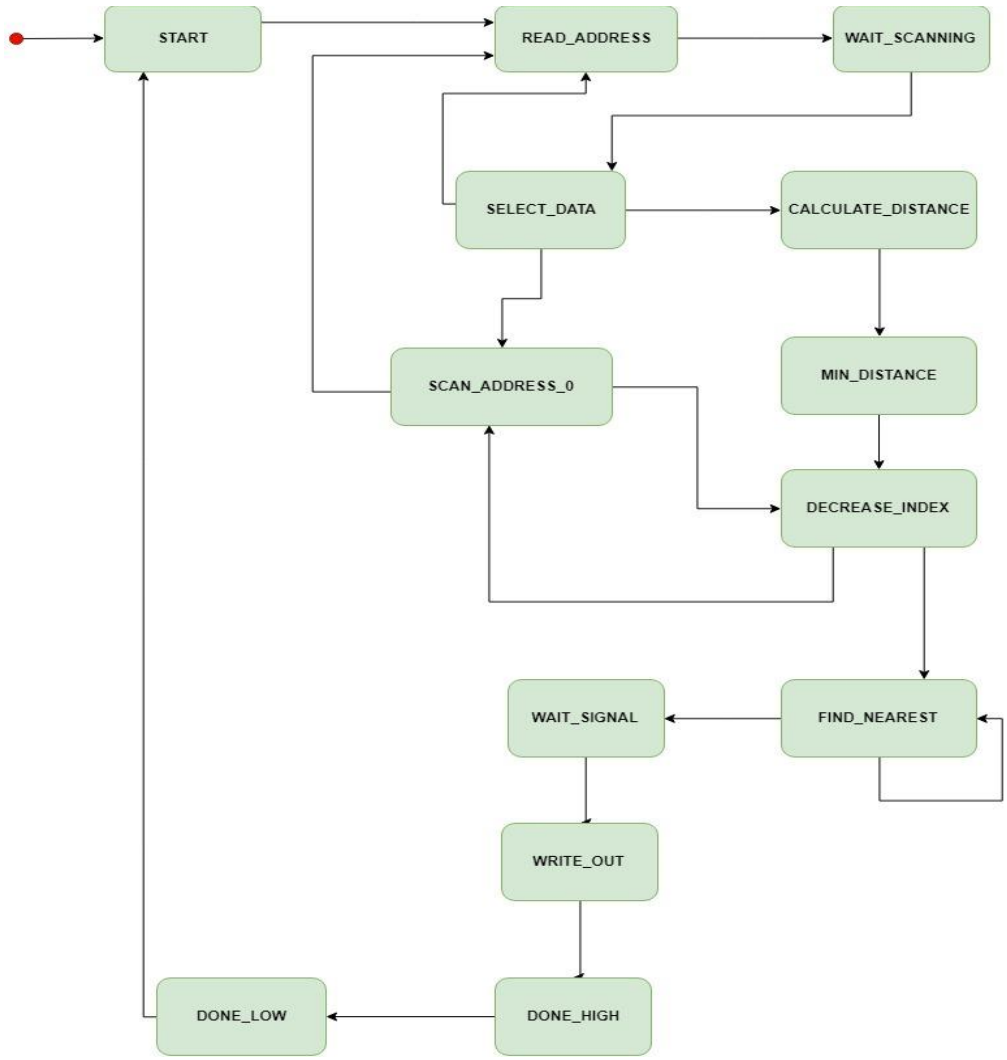
1L'interazione del componente realizzato con la memoria.

-DESCRIZIONE DELL'AUTOMA

L'automa a stati finiti su cui si basa il componente è caratterizzata da 13 stati:

- START: stato di reset; attende il segnale di start e inizializza tutti i segnali;
- WAIT_LETTURA: stato senza operazioni per attendere che la memoria invii i dati;
- READ_ADDRESS: vengono settati i parametri per la lettura dell'indirizzo;
- SELECT_DATA: lettura dei dati in arrivo dalla memoria. L'indirizzo corrente di memoria differenzia quali signal inizializzare;
- SCAN_ADDRESS_0: si analizzano i singoli bit dell'indirizzo zero di memoria;
- CALCULATE_DISTANCE: calcolo dellla distanza tra il centroide corrente e il punto da valutare;
- MIN_DISTANCE: tiene aggiornato il valore della distanza minima;

- DECREASE_INDEX: decrementa il valore dell'indice i;
- FIND_NEAREST: trova i centroidi più vicini al punto da valutare e imposta la maschera che contiene il risultato dell'elaborazione;
- WAIT_SIGNAL: impone i segnali per la scrittura in memoria;
- WRITE_OUT: scrive il risultato in memoria;
- DONE_HIGH: alza il segnale di fine elaborazione;
- DONE_LOW: abbassa il segnale di fine elaborazione;



2 *Automa a stati finiti.*

-DESCRIZIONE DELL' ALGORITMO

Il processo è eseguito a commutazioni del clock con reset asincrono.

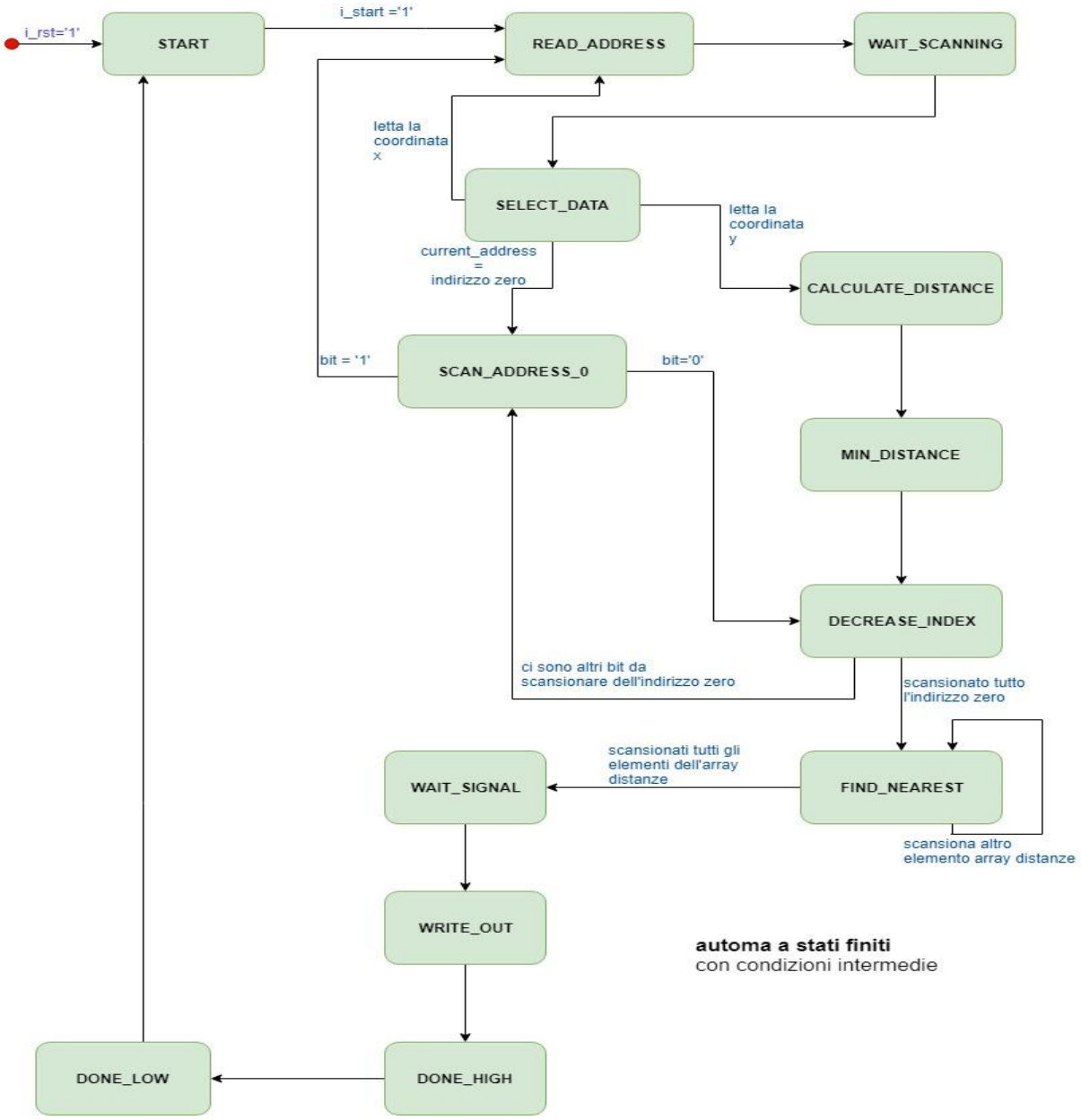
La macchina a stati aspetta il segnale di start per inizializzare i segnali, in particolare lavora sul fronte di salita del clock dove si inizializza la lettura dell'indirizzo 17 della memoria per impostare le coordinate del punto da valutare.

In seguito si legge l'indirizzo zero e in base al valore dei singoli bit che lo costituiscono si prende in considerazione o meno il rispettivo centroide.

Lette entrambe le coordinate del centroide se ne calcola la distanza dal punto da considerare e la si salva nell'array *distanze,* si aggiorna la distanza minima e si passa alla lettura del centroide successivo.

Finita la scansione dell'indirizzo zero, si confronta il contenuto dell'array *distanze* con *distanza_ min* e si riportano i risultati in *current_data* (in *current_data* è presente il contenuto dell'indirizzo zero di memoria, i bit a 1 andranno portati a 0 se il centroide corrispondente non sarà alla minima distanza dal punto da considerare).

Infine si scrive in memoria il risultato e si riporta la macchina allo stato di START pronta per un'altra elaborazione.



3 Automa a stati finiti con condizioni.

-SCELTE IMPLEMENTATIVE

Partendo da carta e penna, ho pensato subito ad una macchina a stati, dopo aver realizzato un primo "modello", l'ho arricchito con specifiche condizioni relative ad ogni singolo stato.

Poi sono passato alla progettazione in VHDL, fin da subito ho scelto di implementare un singolo processo che si occupa di svolgere tutte le funzioni.

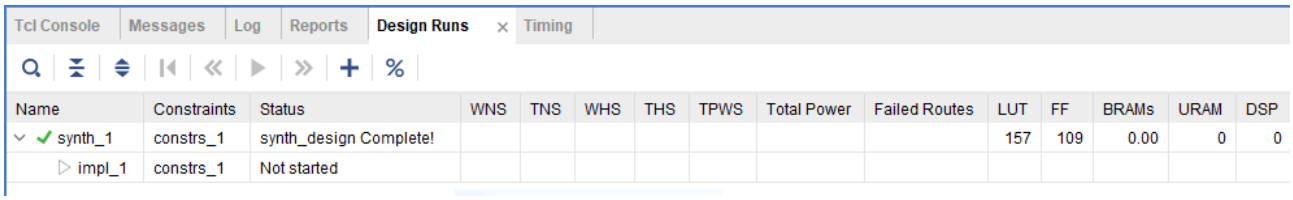
In una prima versione erano presenti meno stati ma più complessi, che portavano ad avere problemi in post-sintesi (legati alla generazione di ritardi); inoltre avevo utilizzato anche della variabili che andavano ad interferirre con il segnale di reset (il quale non era preso in considerazione se messo durante l'elaborazione).

Quindi ho deciso di aumentare il numero di stati rendendoli più lineare e introducendo un nuovo stato di attesa (WAIT_SCANNING) per permettere ai segnali di commutare per tempo.

Gli ultimi problemi sono stati risolti sostituendo le variabili, che si aggiornano nel momento dell'assegnazione, con opportuni segnali, nei quali l'aggiornamento avviene con un certo ritardo.

3. RISULTATI SPERIMENTALI

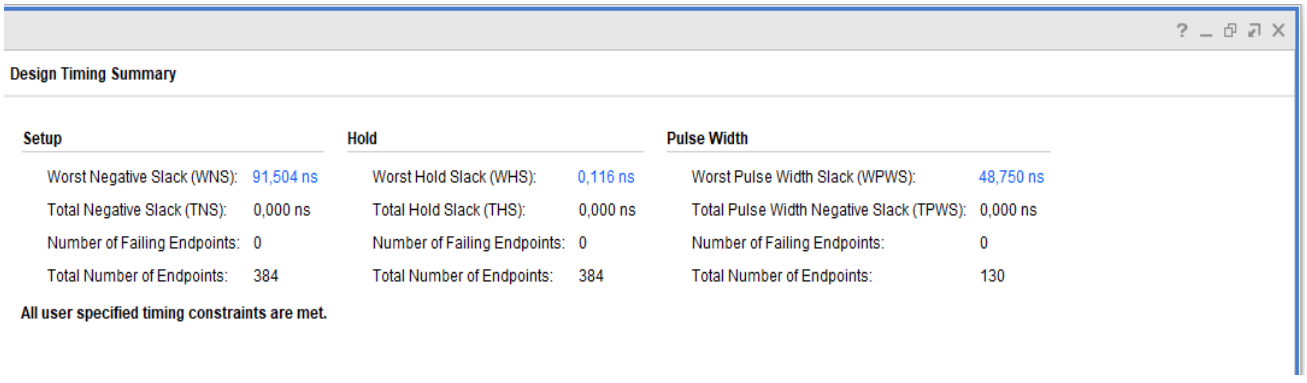
-REPORT DI SINTESI



Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP
synth_1	constrs_1	synth_design Complete!								157	109	0.00	0	0
impl_1	constrs_1	Not started												

4 Report di sintesi.

Impostando un clock di 100 ns e un ritardo in lettura dalla memoria pari a 2 ns (come da testbench messo a disposizione) si ottiene:



5 Report timing.

-REPORT DI SIMULAZIONE: TESTBENCHs

Dopo aver verificato il corretto funzionamento del componenete lo si sottoporà ad alcuni casi limite.

Si riportano prima il codice del testbench e poi la waveform della post-synthesis timing simulation la quale durante la progettazione del componente ha fatto emergere più errori.

1) Testbench messo a disposizione dal docente

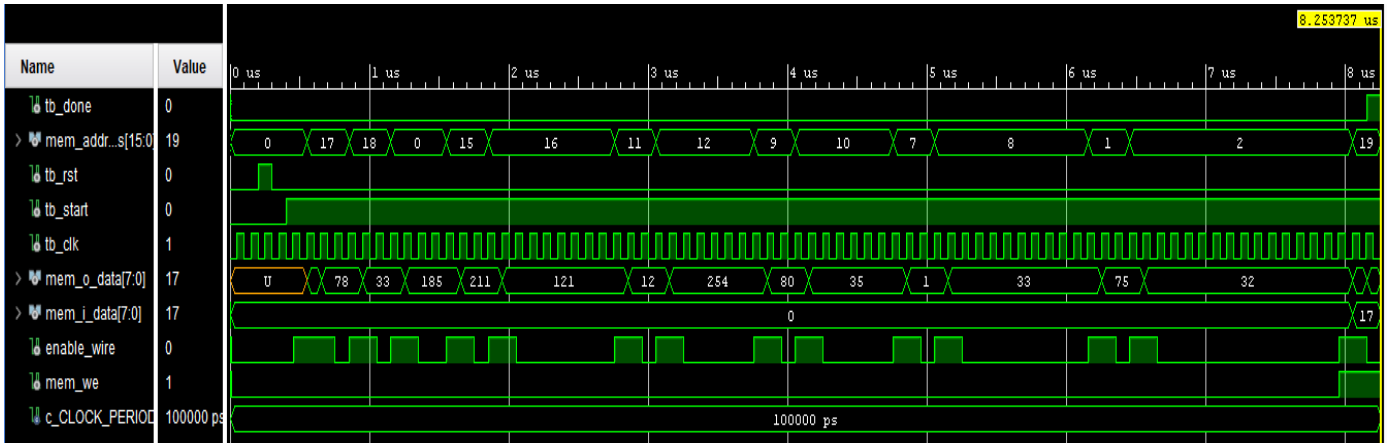
Si verifica che il componente rispetti la specifica, in particolare si considerano cinque centroidi di cui solo due si trovano a distanza minima:

```
test : process is
begin
    wait for 100 ns;
    wait for c_CLOCK_PERIOD;
    tb_rst <= '1';
    wait for c_CLOCK_PERIOD;
    tb_rst <= '0';
    wait for c_CLOCK_PERIOD;
    tb_start <= '1';
    wait for c_CLOCK_PERIOD;
    wait until tb_done = '1';
    wait for c_CLOCK_PERIOD;
    tb_start <= '0';
    wait until tb_done = '0';

    -- Maschera di output = 00010001
    assert RAM(19) = std_logic_vector(to_unsigned( 17 , 8)) report "TEST FALLITO" severity failure;

    assert false report "Simulation Ended!, TEST PASSATO" severity failure;
end process test;
```

6 Testbench.



7 Waveform della post-synthesis timing simulation.

Per quanto riguarda le altre due simulazioni si è ottenuto: la behavioral simulation in 8.250000 µs e la post-synthesis functional simulation in 8.250100 µs.

Si ottengono i risultati previsti con tutti i segnali inizializzati nel momento della loro istanzazione.

L'elaborazione inizia quando il segnale *i_start* si alza e si conclude quando il segnale *tb_done* si alza.

2) Reset nel mezzo dell'elaborazione

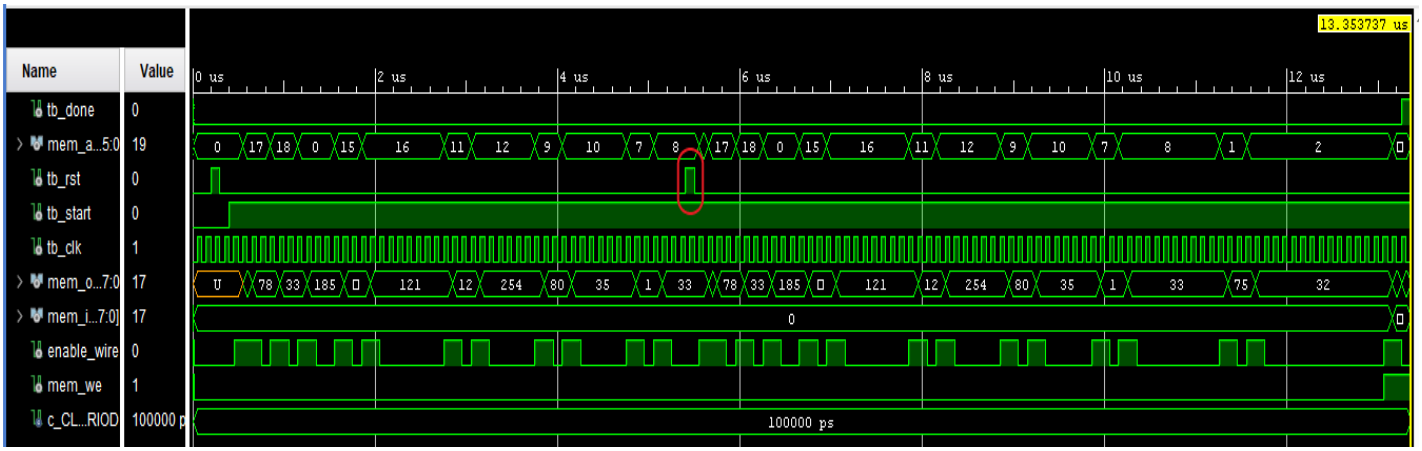
Si alza un segnale di reset dopo 50 cicli di clock, quindi il testbench testa la corretta sincronizzazione con il segnale di reset.

```
begin
    wait for 100 ns;
    wait for c_CLOCK_PERIOD;
    tb_rst <= '1';
    wait for c_CLOCK_PERIOD;
    tb_rst <= '0';
    wait for c_CLOCK_PERIOD;
    tb_start <= '1';
    wait for 50*c_CLOCK_PERIOD;
    tb_rst <='1';
    wait for c_CLOCK_PERIOD;
    tb_rst <= '0';
    wait for c_CLOCK_PERIOD;
    tb_start <= '1';
    wait for c_CLOCK_PERIOD;
    wait until tb_done = '1';
    wait for c_CLOCK_PERIOD;
    tb_start <= '0';
    wait until tb_done = '0';

    -- Maschera di output = 00010001
    assert RAM(19) = std_logic_vector(to_unsigned( 17 , 8)) report "TEST FALLITO" severity failure;

    assert false report "Simulation Ended!, TEST PASSATO" severity failure;
end process test;
```

8 Codice testbench.



9 Wavefrom post-synthesis timng simulation.

Per quanto riguarda le altre due simulazioni si è ottenuto: la behavioral simulation in 13,350000 µs e la post-synthesis functional simulation in 13,350100 µs.

Il componente, ricevuto il segnale di reset, riporta la macchina a stati allo stato iniziale ed avvia l'elaborazione da zero immediatamente, essendo il segnale *i_start* sempre alto.

L'elaborazione termina rispettando i risultati previsti.

3) **Nessun centroide da considerare**

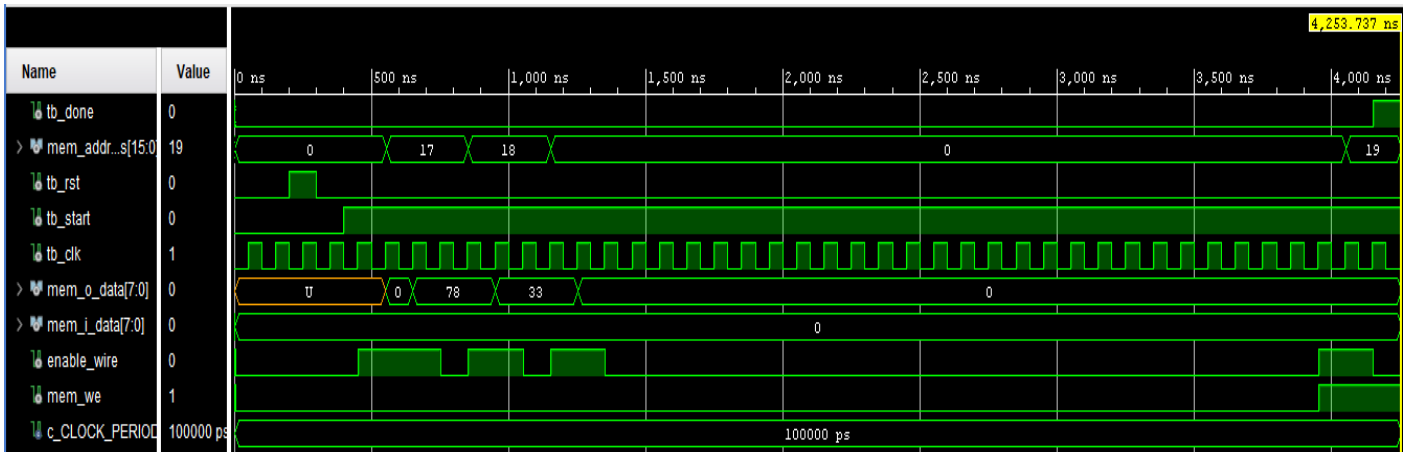
Nessun centroide è preso in cosiderazione, ci aspettiamo una maschera d'uscita composta da tutti zero:

```
test : process is
begin
    wait for 100 ns;
    wait for c_CLOCK_PERIOD;
    tb_rst <= '1';
    wait for c_CLOCK_PERIOD;
    tb_rst <= '0';
    wait for c_CLOCK_PERIOD;
    tb_start <= '1';
    wait for c_CLOCK_PERIOD;
    wait until tb_done = '1';
    wait for c_CLOCK_PERIOD;
    tb_start <= '0';
    wait until tb_done = '0';

    -- Maschera di output = 00000000
    assert RAM(19) = std_logic_vector(to_unsigned( 0 , 8)) report "TEST FALLITO" severity failure;

    assert false report "Simulation Ended!, TEST PASSATO" severity failure;
end process test;
```

10 Codice testbench.



11 Report post-synthesis timing simulation.

Per quanto riguarda le altre due simulazioni si è ottenuto: la behavioral simulation in 4,250000 µs e la post-synthesis functional simulation in 4,250100 µs.

L'elaborazione è molto più veloce rispetto al testbench messo a disposizione dal docente, infatti il componente deve fare molte meno letture in memoria poiché, dopo aver scansionato l'indirizzo zero di memoria, passa direttamente alla scrittura del risultato.

4) Tutti i centroidi da considerare

Si devono considerare tutti i centrodi, massimizzando il numero di accessi alla memoria, dunque si testa come il componente gestisce i ritardi causati dalla lettura in memoria.

```
signal RAM: ram_type := (0 => std_logic_vector(to_unsigned( 255 , 8)),
```

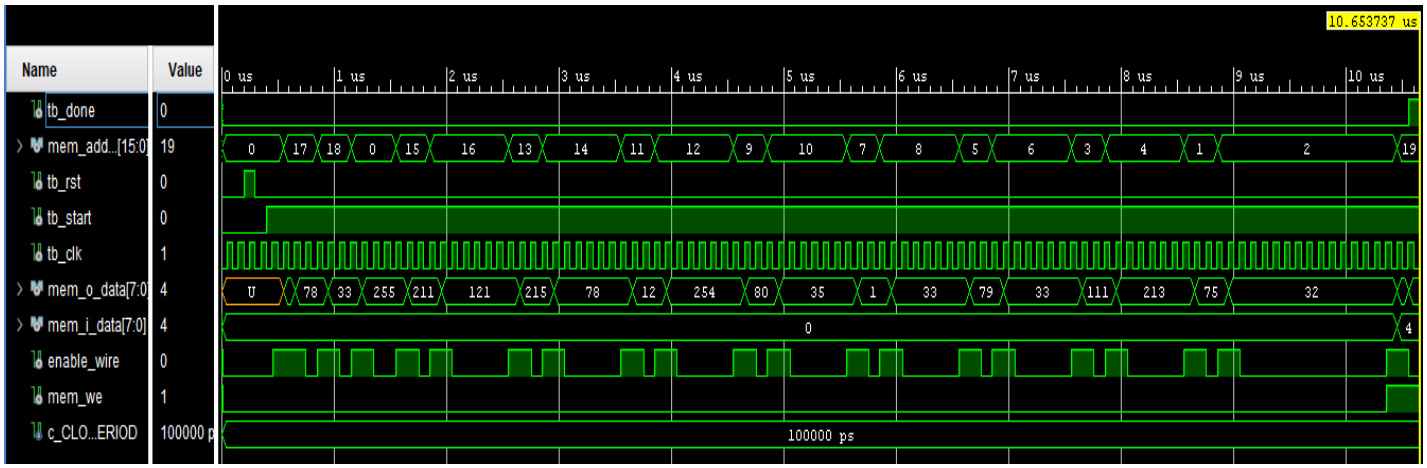
12 Considero tutti i centroidi.

```
test : process is
begin
    wait for 100 ns;
    wait for c_CLOCK_PERIOD;
    tb_rst <= '1';
    wait for c_CLOCK_PERIOD;
    tb_rst <= '0';
    wait for c_CLOCK_PERIOD;
    tb_start <= '1';
    wait for c_CLOCK_PERIOD;
    wait until tb_done = '1';
    wait for c_CLOCK_PERIOD;
    tb_start <= '0';
    wait until tb_done = '0';

    -- Maschera di output = 00000100
    assert RAM(19) = std_logic_vector(to_unsigned(4 , 8)) report "TEST FALLITO" severity failure;

    assert false report "Simulation Ended!, TEST PASSATO" severity failure;
end process test;
```

13 Codice testbench.



14 Report post-synthesis timing simulation.

Per quanto riguarda le altre due simulazioni si è ottenuto: la behavioral simulation in 10,650000 µs e la post-synthesis functional simulation in 10,650100 µs.

Il testbench è portato a termine con risultati positivi, si leggono da memoria tutti gli indirizzi dallo zero al diciotto, ciò comporta un tempo di esecuzione massimo per la singola iterazione.

5) **Distanza minima**

Un centroide e il punto da considerare hanno le stesse coordinate, si verifica come il componente gestisce la distanza minima.

```
signal RAM: ram_type := (0 => std_logic_vector(to_unsigned( 255 , 8)),
                         1 => std_logic_vector(to_unsigned( 75 , 8)),
                         2 => std_logic_vector(to_unsigned( 32 , 8)),
                         3 => std_logic_vector(to_unsigned( 111 , 8)),
                         4 => std_logic_vector(to_unsigned( 213 , 8)),
                         5 => std_logic_vector(to_unsigned( 78 , 8)),
                         6 => std_logic_vector(to_unsigned( 33 , 8)),
                         7 => std_logic_vector(to_unsigned( 1 , 8)),
                         8 => std_logic_vector(to_unsigned( 33 , 8)),
                         9 => std_logic_vector(to_unsigned( 80 , 8)),
                         10 => std_logic_vector(to_unsigned( 35 , 8)),
                         11 => std_logic_vector(to_unsigned( 12 , 8)),
                         12 => std_logic_vector(to_unsigned( 254 , 8)),
                         13 => std_logic_vector(to_unsigned( 215 , 8)),
                         14 => std_logic_vector(to_unsigned( 78 , 8)),
                         15 => std_logic_vector(to_unsigned( 211 , 8)),
                         16 => std_logic_vector(to_unsigned( 121 , 8)),
                         17 => std_logic_vector(to_unsigned( 78 , 8)),
                         18 => std_logic_vector(to_unsigned( 33 , 8)),
            others => (others =>'0'));
```

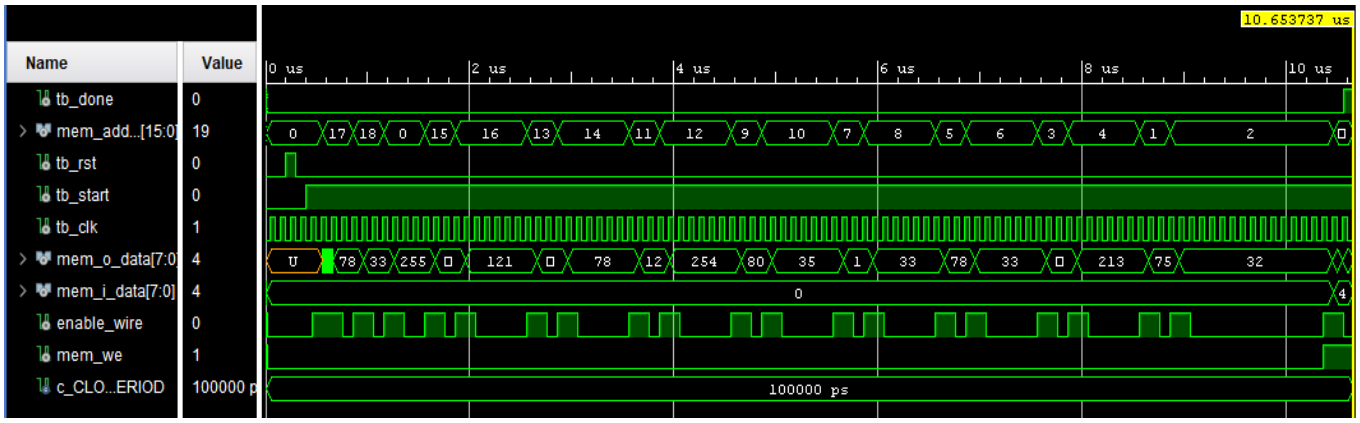
15 Coincidenza tra centroide e punto da considerare.

```
    test : process is
    begin
        wait for 100 ns;
        wait for c_CLOCK_PERIOD;
        tb_rst <= '1';
        wait for c_CLOCK_PERIOD;
        tb_rst <= '0';
        wait for c_CLOCK_PERIOD;
        tb_start <= '1';
        wait for c_CLOCK_PERIOD;
        wait until tb_done = '1';
        wait for c_CLOCK_PERIOD;
        tb_start <= '0';
        wait until tb_done = '0';

        -- Maschera di output = 00000100
        assert RAM(19) = std_logic_vector(to_unsigned(4 , 8)) report "TEST FALLITO" severity failure;

        assert false report "Simulation Ended!, TEST PASSATO" severity failure;
    end process test;
```

16 Codice testbench.



17 Report post-synthesis timing simulation.

Per quanto riguarda le altre due simulazioni si è ottenuto: la behavioral simulation in 10,650000 µs e la post-synthesis functional simulation in 10,650100 µs.

Il risultato atteso è stato ottenuto, la distanza nulla o minima è trattata come qualsiasi altra distanza.

6) Distanza massima

Il punto da considerare è posizionato in un vertice dello spazio bidimensionale mentre agli altri estremi ci sono tre centroidi.

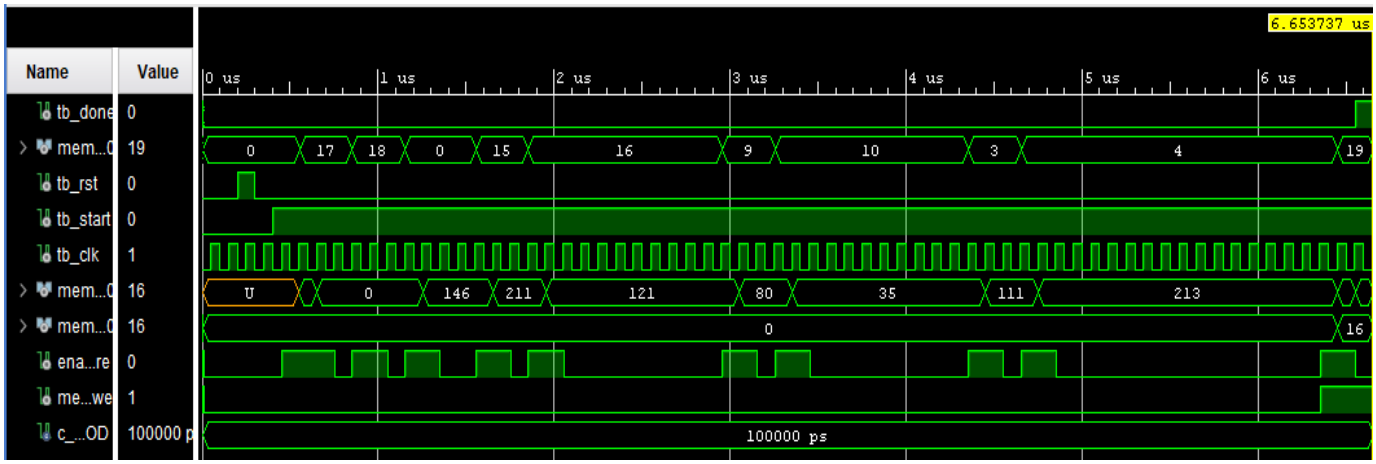
```
-- considero :10010010
signal RAM: ram_type := (0 => std_logic_vector(to_unsigned( 146 , 8)),
                         1 => std_logic_vector(to_unsigned( 255, 8)),
                         2 => std_logic_vector(to_unsigned( 0 , 8)),
                         3 => std_logic_vector(to_unsigned( 111 , 8)),
                         4 => std_logic_vector(to_unsigned( 213 , 8)),
                         5 => std_logic_vector(to_unsigned( 78 , 8)),
                         6 => std_logic_vector(to_unsigned( 33 , 8)),
                         7 => std_logic_vector(to_unsigned( 255 , 8)),
                         8 => std_logic_vector(to_unsigned( 255 , 8)),
                         9 => std_logic_vector(to_unsigned( 80 , 8)),
                         10 => std_logic_vector(to_unsigned( 35 , 8)),
                         11 => std_logic_vector(to_unsigned( 12 , 8)),
                         12 => std_logic_vector(to_unsigned( 254 , 8)),
                         13 => std_logic_vector(to_unsigned( 0 , 8)),
                         14 => std_logic_vector(to_unsigned( 255 , 8)),
                         15 => std_logic_vector(to_unsigned( 211 , 8)),
                         16 => std_logic_vector(to_unsigned( 121 , 8)),
                         17 => std_logic_vector(to_unsigned( 0 , 8)),
                         18 => std_logic_vector(to_unsigned( 0 , 8)),
                         others => (others =>'0'));
```

```
    test : process is
    begin
        wait for 100 ns;
        wait for c_CLOCK_PERIOD;
        tb_rst <= '1';
        wait for c_CLOCK_PERIOD;
        tb_rst <= '0';
        wait for c_CLOCK_PERIOD;
        tb_start <= '1';
        wait for c_CLOCK_PERIOD;
        wait until tb_done = '1';
        wait for c_CLOCK_PERIOD;
        tb_start <= '0';
        wait until tb_done = '0';

        -- Maschera di output = 00010000
        assert RAM(19) = std_logic_vector(to_unsigned( 16 , 8)) report "TEST FALLITO" severity failure;

        assert false report "Simulation Ended!, TEST PASSATO" severity failure;
    end process test;
```

18 Codice del testbench.



19 waveform post-synthesis timing simulation.

Per quanto riguarda le altre due simulazioni si è ottenuto: la behavioral simulation in 6,650000 µs e la post-synthesis functional simulation in 6,650100 µs.

La distanza massima è calcolata correttamente, senza creare problemi al componente.

5. CONCLUSIONI

Il componente esegue le operazioni come da specifica senza accusare problemi generati dai ritardi. A livello di comportamento logico è implementato utilizzando 157 Look Up Table e 109 Flip Flop.

Risulta superare tutti i testbench a cui è stato sottoposto in tutte e tre le simulazioni (behavioral e le due post-synthesis:functional e timing).