PROGETTO DI RETI LOGICHE AA 2018/19

Professore: Gianluca Palermo

Gruppo formato da:

Andrea Tresoldi

Codice Persona: 10535801

Matricola: 847387

Simone Zani

Codice Persona: 10502938

Matricola: 846664

POLITECNICO
MILANO 1863

SPECIFICA DI PROGETTO (testo fornito dal professor Gianluca Palermo)

Sia dato 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<=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 valido (punto dal quale calcolare la distanza) 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.

Dati

I dati ciascuno di dimensione 8 bit sono memorizzati in una memoria con indirizzamento al Byte partendo dalla posizione 0.

- L'indirizzo 0 è usato per memorizzare il numero di centroidi (Maschera di ingresso: definisce quale centroide deve essere esaminato);
- Gli indirizzi dal 1 al 16 sono usati per memorizzare le coordinate a coppie X e Y dei centroidi:

1- Coordinata X primo Centroide
2- Coordinata Y primo Centroide
3- Coordinata X secondo Centroide
4- Coordinata Y secondo Centroide
……..
15- Coordinata X ottavo Centroide
16- Coordinata Y ottavo Centroide

- Gli indirizzi 17 e 18 sono usati per memorizzare le coordinate X e Y del punto da valutare
- L'indirizzo 19 è usato per scrivere alla fine il valore della maschera di uscita.

Note ulteriori sulla specifica Dati

- I centroidi nella maschera vengono elencati dal bit meno significativo posizione più a destra (Centroide 1) al più significativo (Centroide 8);
- Il valore della maschera risultante dall'identificazione del/dei centroide/i più vicino/i segue la stessa regola vista al punto precedente. Il bit meno significativo rappresenta il Centroide 1 mentre quello più significativo il Centroide 8;
- Il modulo partirà nella elaborazione quando un segnale START in ingresso verrà portato a 1. Il segnale di START rimarrà alto fino a che il segnale di DONE non verrà portato alto; Al termine della computazione (e una volta scritto il risultato in memoria), il modulo da progettare deve alzare (portare a 1) il segnale DONE che notifica la fine dell'elaborazione. Il segnale DONE deve rimanere alto fino a che il segnale di START non è riportato a 0. Un nuovo segnale start non può essere dato fin tanto che DONE non è stato riportato a zero.

OBIETTIVO

Il nostro obiettivo è quello di descrivere (in VHDL) e sintetizzare il componente hardware che implementa la specifica richiesta, interfacciandosi con una memoria dove sono salvati i dati e dove dovrà essere scritto il risultato finale.
Lo strumento di sintesi da noi usato è XILINX VIVADO WEBPACK e la FPGA target utilizzata è xc7a200tfbg484-1.
Inoltre è compito nostro assicurarci che il componente descritto funzioni con un periodo di clock di 100ns.

DESCRIZIONE STATI DELLA MACCHINA

In totale gli stati della nostra macchina sono 15, da S0 a S13 e lo stato di RESET
Abbiamo deciso di modellare la macchina a stati finiti utilizzando un unico processo che avesse come parametri della lista di sensitività il segnale di clock e il segnale di reset, in modo che ogni variazione di quest'ultimi risvegliasse il processo stesso.
La nostra macchina è sensibile al fronte di salita del clock; ad ogni evento di salita la nostra macchina effettuerà tutte le operazioni stabilite all'interno dello stato in cui si trova ed evolverà nello stato successivo.
Vogliamo sottolineare il fatto che ogni assegnamento fatto all'interno di uno stato non è istantaneo, ma il valore dato ad un segnale verrà effettivamente aggiornato solamente al termine dell'esecuzione del processo; il nuovo valore del segnale sarà quindi osservabile solamente nello stato successivo allo stato in cui è avvenuto l'assegnamento.

Il reset è stato modellizzato come un segnale asincrono:
da specifica questo segnale in ingresso al nostro componente verrà portato alto per inizializzare la macchina, in modo che sia pronta per ricevere il primo segnale di start.
Nella nostra implementazione, se in ingresso al nostro componente arriva un segnale di reset, la macchina si porterà nello stato di RESET indipendentemente dal segnale di clock.

RESET → è lo stato iniziale dove inizia l'elaborazione.
La macchina torna nello stato di RESET quando il segnale i_rst viene portato a 1 indipendentemente dal segnale di clock (i_rst è asincrono). In questo stato la macchina si occupa di inizializzare tutti i segnali per poter iniziare una nuova computazione.
Se il segnale di reset viene portato alto durante una computazione già avviata, l'inizializzazione dei segnali fatta in questo stato permette di iniziarne una nuova senza avere inconsistenza dei dati dall'elaborazione precedente.
La macchina si sposta nello stato successivo soltanto quando arriva un segnale di start e si verifichi un evento di clock che porti il segnale da basso ad alto.

Da S0 a S4 → vengono salvati in due segnali differenti le coordinate X e Y del punto di riferimento (contenute nelle celle 17 e 18 della memoria) e la maschera in ingresso (contenuta nella cella 0 della memoria) nel segnale corrispondente.
In particolare, nello stato S0 viene anche impostato il segnale di enable a 1 per poter comunicare in memoria, e write_enable a 0 per poterci accedere in solo lettura.

Da S5 a S11 → viene eseguito il calcolo delle distanze di ogni centroide dal punto di riferimento per costruire la maschera di uscita corretta (il calcolo verrà effettuato solamente se il bit associato a tale centroide nella maschera di ingresso vale 1 poiché se pari a 0 si rimarrà nello stato S5 e si passerà a controllare il bit successivo).

S5 (la macchina in questo stato può compiere 3 differenti di azioni) →

- se la macchina, bit per bit, ha studiato tutta la maschera di ingresso, allora si sposterà in S12 occupandosi di impostare tutti i segnali necessari per poter scrivere in memoria, nella cella 19, la maschera d'uscita finale.
- se il bit della maschera in ingresso considerato, associato al centroide analizzato in quel momento, è pari a 0, allora quel centroide non andrà preso in considerazione; la macchina rimarrà in questo stato andando ad analizzare il bit successivo della maschera in ingresso;
- se il bit della maschera in ingresso considerato, associato al centroide analizzato in quel momento è pari a 1, allora ci sposteremo in S6 dove la macchina inizierà la procedura per il calcolo della distanza di Manhattan tra il punto di riferimento e il centroide considerato.

Da S6 a S8 → vengono salvati in due segnali diversi le coordinate X e Y del centroide da studiare.

S9 → la macchina salva in due segnali differenti il risultato della sottrazione tra la X del punto di riferimento e la X del centroide e il risultato della sottrazione tra la Y del punto di riferimento e la Y del centroide.
Prima del calcolo delle differenze però la macchina si occupa di verificare, sia per le X sia per le Y, quale tra i due operandi in gioco sia maggiore, in modo tale da avere sempre un numero positivo come risultato dell'operazione.

S10 → La macchina calcola la distanza di Manhattan come somma delle due distanze parziali precedentemente calcolate.

S11 → La macchina confronta la distanza di Manhattan appena calcolata (relativa al centroide considerato) con la distanza minima trovata fino a quel momento della computazione (il valore della distanza minima all'inizio dell'elaborazione è 510, valore che coincide con la distanza massima ottenibile tra due punti all'interno di uno spazio bi-dimensionale 256x256).
Il valore della distanza minima ha una funzione di soglia e verrà continuamente aggiornato durante la computazione per tenere traccia di quale sia la distanza più piccola trovata fino a quel momento. Il valore di questa soglia è fondamentale per costruire correttamente la maschera di uscita.
In questo stato la macchina compirà azioni diverse a seconda del valore della distanza di Manhattan appena calcolata: se risulta

- minore della distanza minima corrente, allora aggiorniamo la distanza minima e assegniamo il valore 1 al bit della maschera d'uscita corrispondente al centroide considerato, impostando tutti gli altri bit a 0; questa operazione è possibile assegnando alla maschera di uscita il valore del segnale UpdateMask.
- uguale alla distanza minima corrente, assegniamo 1 al bit della maschera di uscita corrispondente al centroide considerato senza però modificare il valore degli altri bit; ciò è reso possibile sommando bit a bit il segnale UpdateMask e la maschera d'uscita costruita fino a quel momento.
- maggiore della distanza minima corrente, allora la macchina tornerà in S5, ricominciando con il calcolo di una nuova distanza associata al centroide successivo.

S12 → In questo stato la macchina scrive il risultato in memoria sfruttando l'assegnamento dei segnali fatti nello stato precedente (ricordiamo che le azioni svolte all'interno di uno stato sono eseguite in corrispondenza di un fronte di salita del clock).

La macchina si occupa inoltre di impostare il segnale di o_done a 1 per segnalare la fine dell'elaborazione; come già detto in precedenza, ogni assegnamento fatto ad un segnale sarà osservabile nello stato successivo, quindi il segnale di o_done avrà valore 1 solamente in S13 quando, il risultato scritto in memoria (la maschera di uscita), sarà effettivamente contenuto nella cella 19.

S13 → La macchina attende che il segnale di i_start torni a 0 per poter riportare il segnale di o_done anch'esso a 0 (quindi si rimarrà in questo stato fin quando il segnale di i_start non tornerà basso).
Nel momento in cui il segnale di i_start varrà effettivamente 0, la macchina tornerà nello stato di RESET assegnando a tutti i segnali utilizzati i valori iniziali per poter ricominciare una nuova elaborazione.

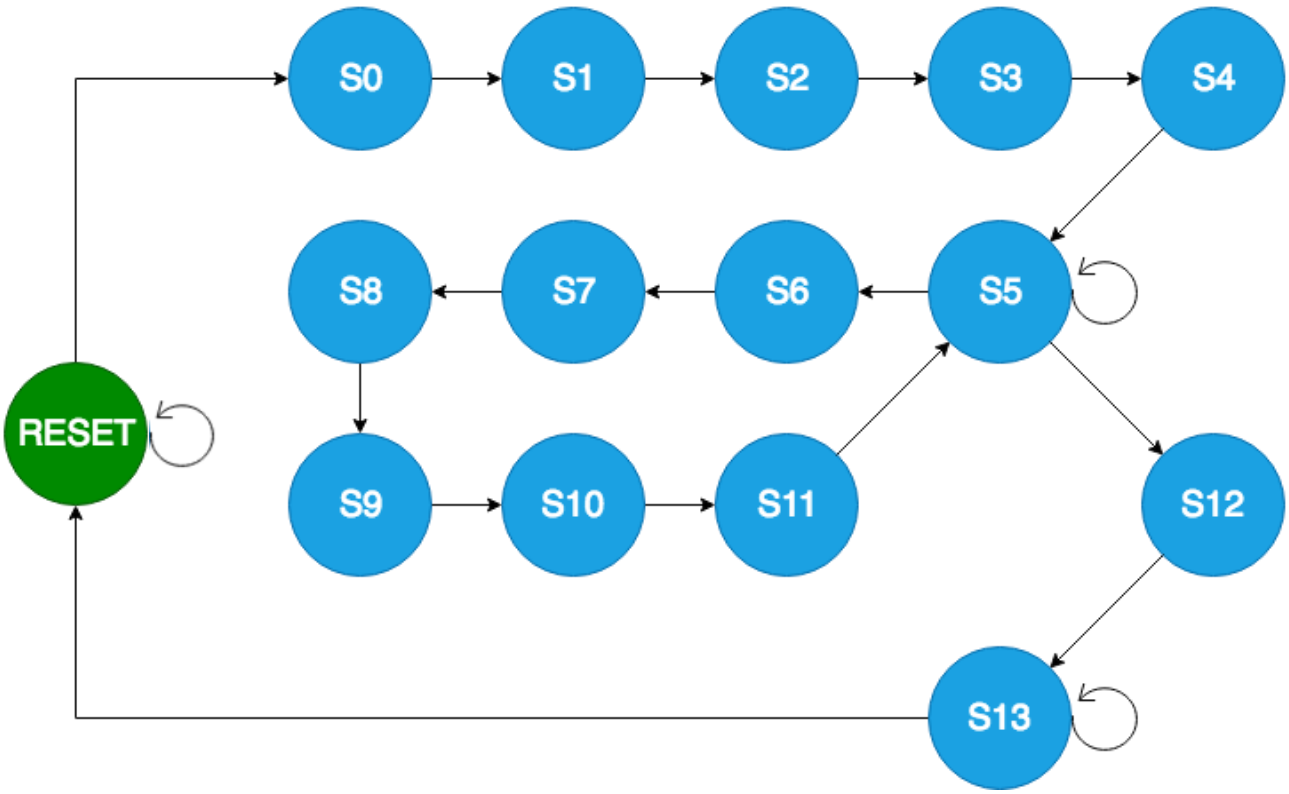
Da notare:

Nello stato S9:
abbiamo scelto di implementare il calcolo in questo modo, per evitare di dover utilizzare la funzione abs (valore assoluto) messa a disposizione nel linguaggio VHDL poiché quest'ultima non ci sembrava avere una diretta implementazione in hardware essendo un costrutto di alto livello.

Segnale UpdateMask:
questo segnale, fondamentale per poter aggiornare la maschera di uscita durante la computazione in modo corretto, viene manipolato in modo tale che, quando viene preso in considerazione l'i-esimo bit della maschera di ingresso e il corrispondete centroide, l'i-esimo bit del segnale valga 1 e tutti gli altri bit valgano 0.

Ulteriori note e commenti che precisano ancora meglio il comportamento della macchina sono presenti all'interno del codice.

GRAFO FSM



Ovviamente visto che il segnale di reset è asincrono e idealmente potrebbe essere portato ad 1 in un qualunque istante della computazione, ogni stato dovrebbe avere una transizione che porta nello stato di RESET; in questo grafico non vengono mostrate tutte queste transizioni che portano in RESET per migliorare la leggibilità della rappresentazione.

TESTBENCH E SIMULAZIONI

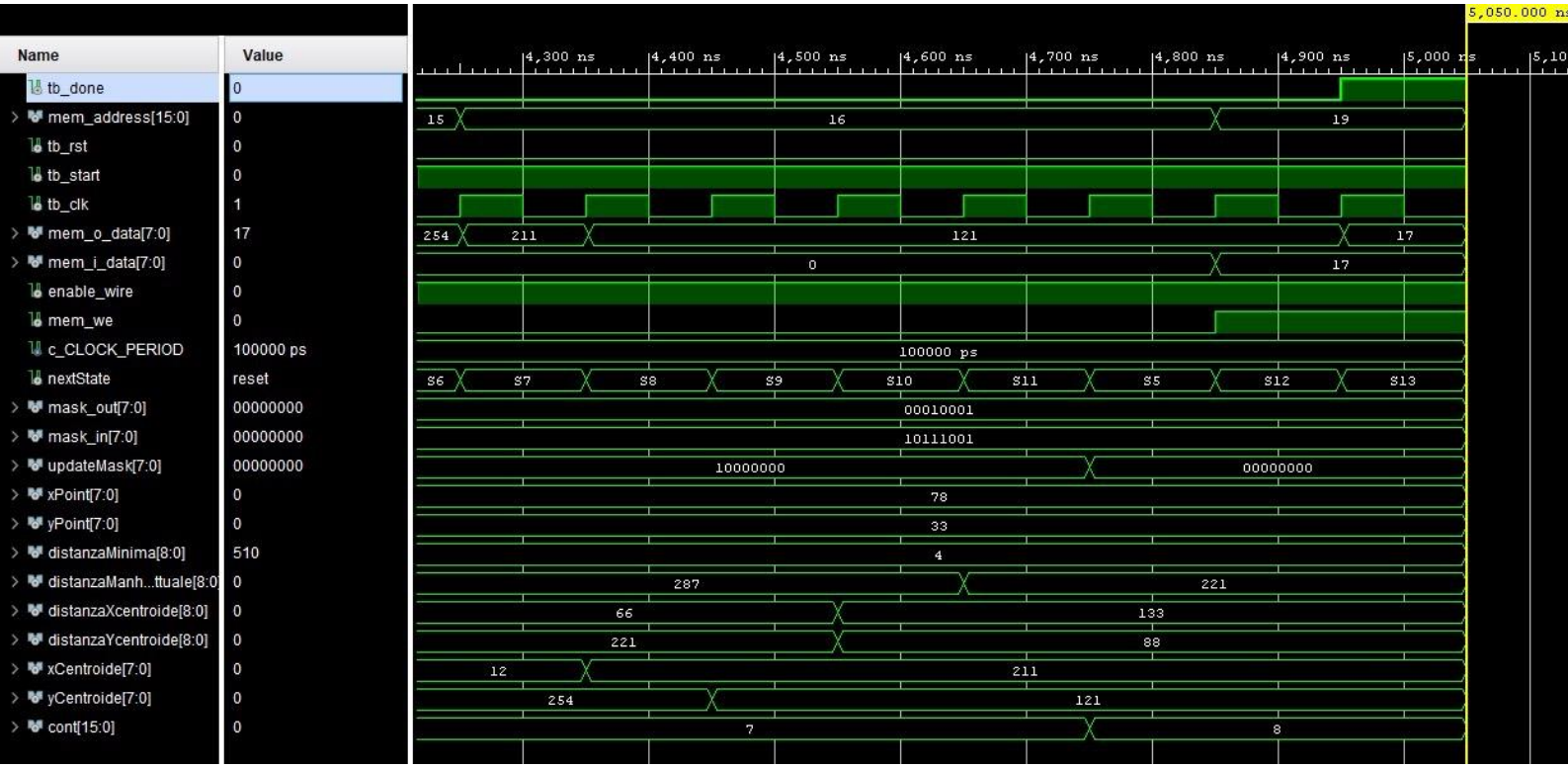
Oltre al Testbench che ci è stato fornito per testare il nostro componente, ne abbiamo creati altri che ci permettessero di verificare la correttezza di quest'ultimo anche in casi particolari della computazione.
Per ogni Testbench utilizzato, abbiamo testato il nostro componente sia in pre-sintesi (Behavioral Simulation) sia in post-sintesi (Fuctional e Timing Simulation) ottenendo sempre una computazione corretta.
In seguito alleghiamo i Testbench utilizzati e i relativi risultati delle varie simulazioni.
Le immagini relative alle simulazioni di post sintesi vengono mostrate solo per il primo Test in quanto i risultati ottenuti sono i medesimi della Behavioral Simulation per tutti i test fatti, tranne che, come ci aspettavamo, per il tempo di elaborazione

TESTBENCH FORNITO:

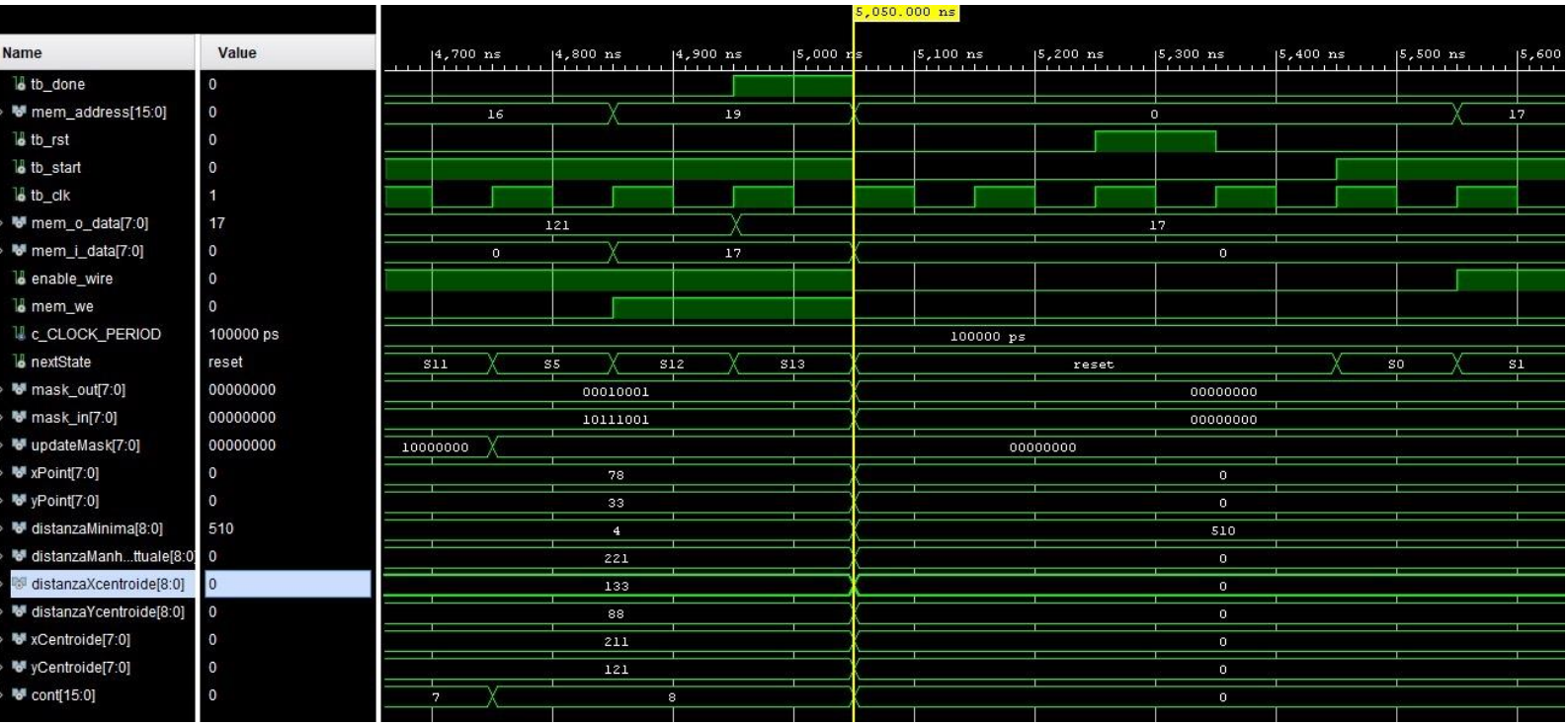
Ci siamo occupati di testare il nostro componente con il testbench fornito dai professori, eseguendo prima una singola computazione e successivamente due simulazioni in sequenza, per verificare se la macchina inizializzasse correttamente i valori dei segnali tra le due computazioni.
I dati presenti in memoria sono specificati all'interno del file del testbench.

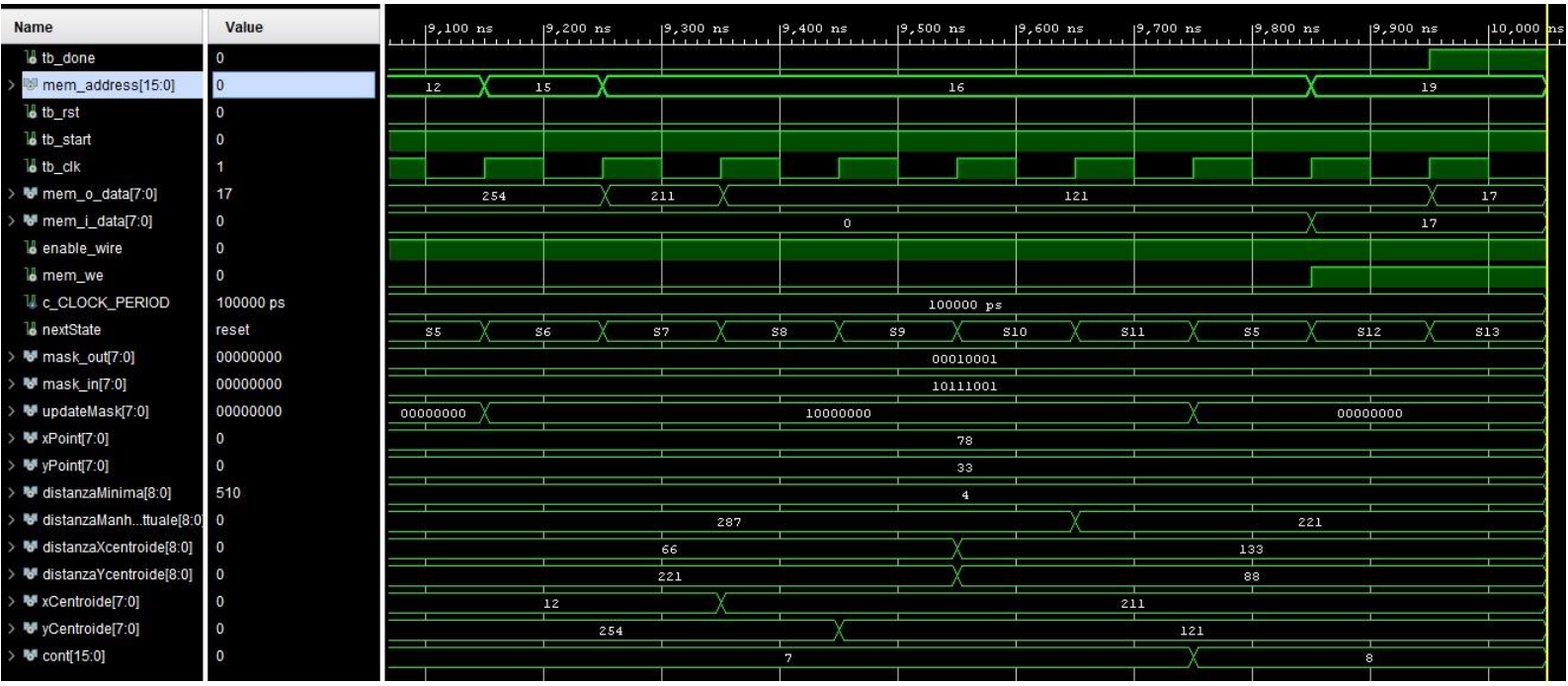
BEHAVIORAL SIMULATION SINGOLA COMPUTAZIONE

```
91          end if;
92      end process;
93
94
95      test : process is
96      begin
97          wait for 100 ns;
98          wait for c_CLOCK_PERIOD;
99          tb_rst <= '1';
100         wait for c_CLOCK_PERIOD;
101         tb_rst <= '0';
102         wait for c_CLOCK_PERIOD;
103         tb_start <= '1';
104         wait for c_CLOCK_PERIOD;
105         wait until tb_done = '1';
106         wait for c_CLOCK_PERIOD;
107         tb_start <= '0';
108         wait until tb_done = '0';
109
110         -- Maschera di output = 00010001
111         assert RAM(19) = std_logic_vector(to_unsigned( 17 , 8)) report "TEST FALLITO" severity failure;
112
113         assert false report "Simulation Ended!, TEST PASSATO" severity failure;
114     end process test;
115
116     end projecttb;
```



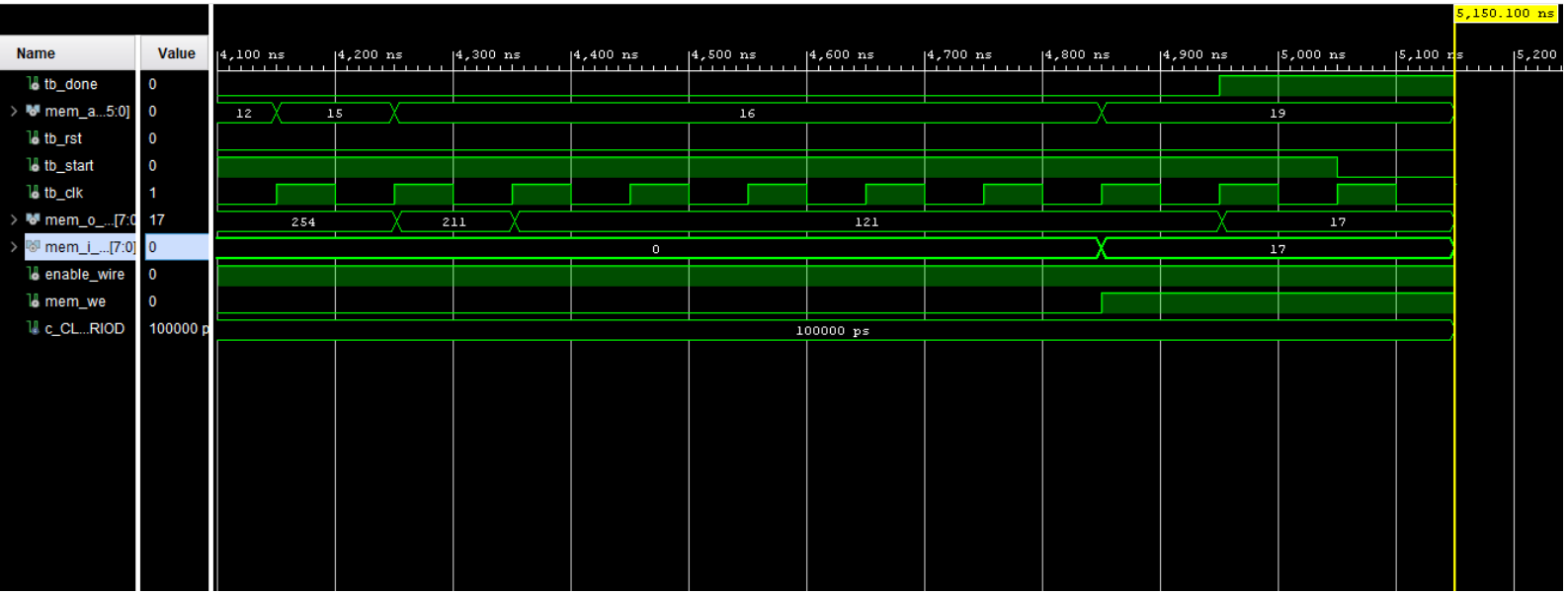
BEHAVIORAL SIMULATION CON 2 COMPUTAZIONI IN CASCATA (inizio e fine seconda computazione)



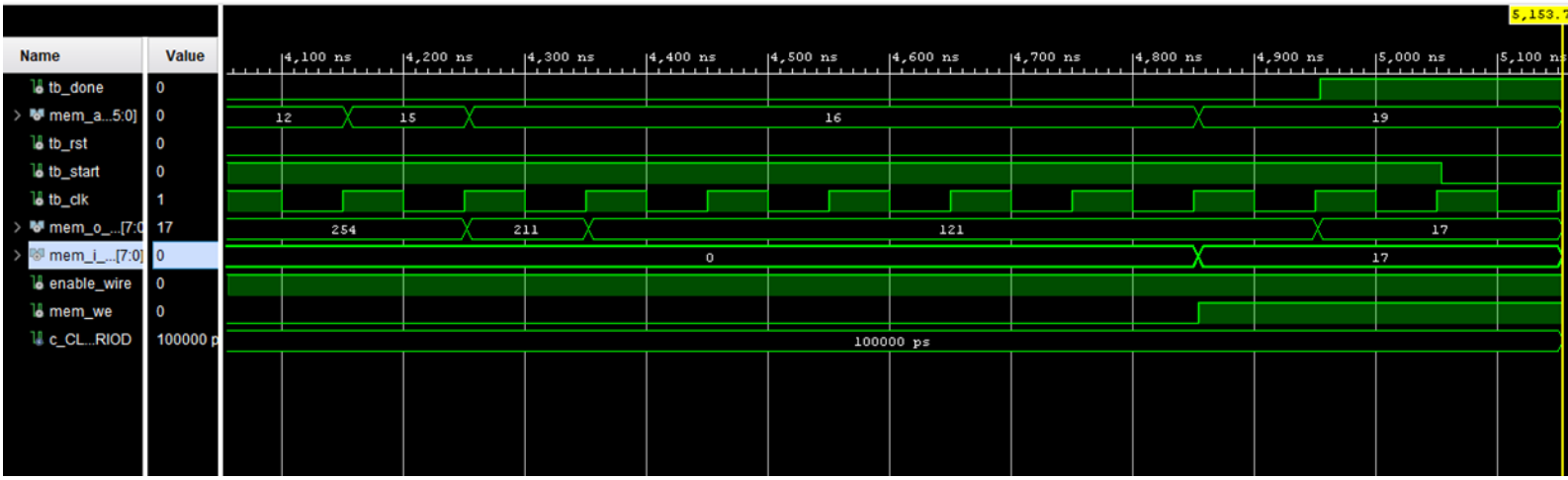


POST-SYNTHESIS FUNCTIONAL SIMULATION E POST-SYNTHESIS TIMING SIMULATION (abbiamo riportato solo i segnali significativi).

FUNCTIONAL:



TIMING:



Entrambi i test sono stati passati con successo come in behavioral simulation.

1) TESTBENCH CREATO

centroidi coincidenti tra di loro e coincidente con il punto di riferimento: mask_in tutti 1 (la macchina studia tutti i centroidi); ci aspettiamo mask_out anch'essa di tutti 1:

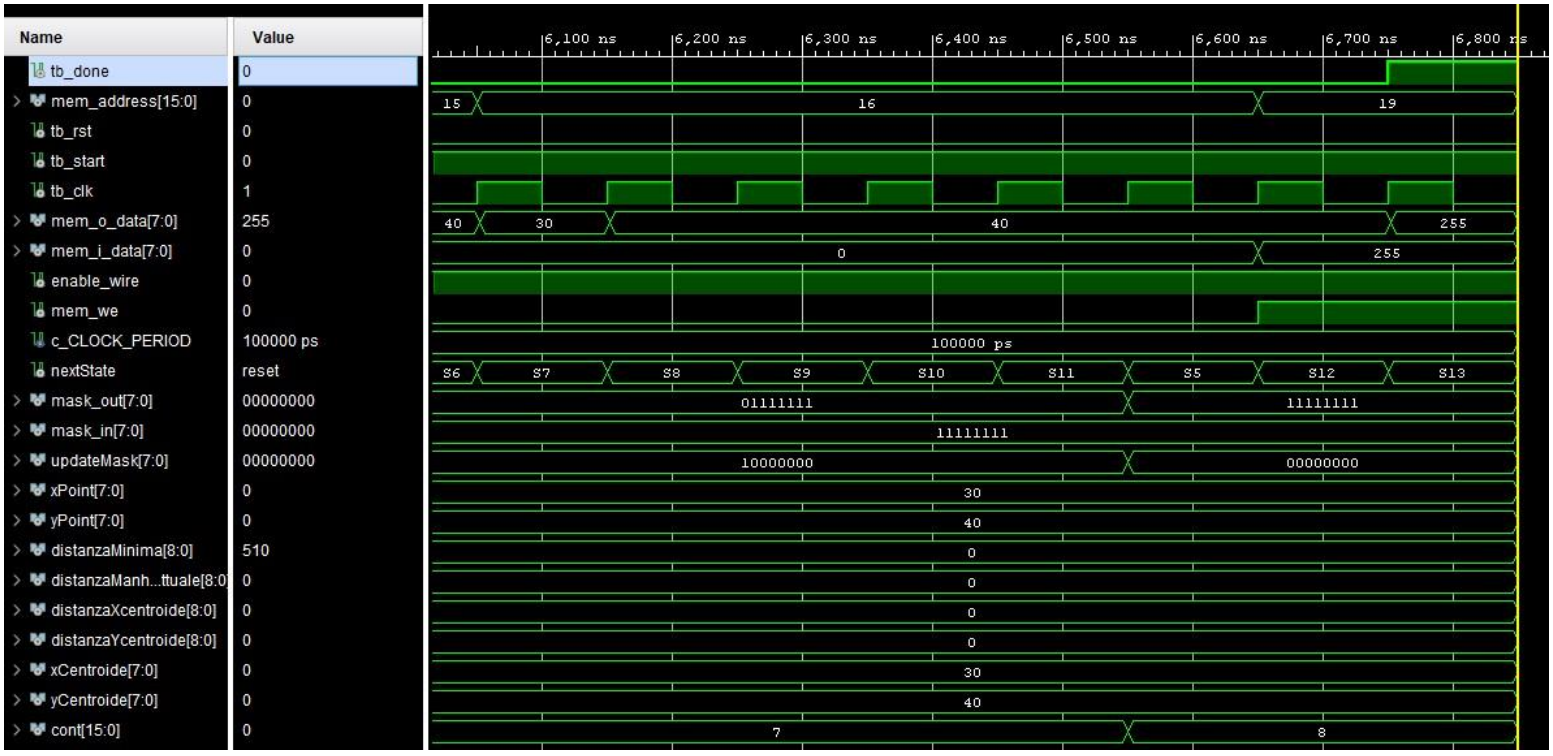
C:/Users/Simone/Desktop/Test Bench di Esempio.vhd

```
20  type ram_type is array (65535 downto 0) of std_logic_vector(7 downto 0);
21
22  -- come da esempio su specifica
23  signal RAM: ram_type := (0 => std_logic_vector(to_unsigned( 255 , 8)),
24                           1 => std_logic_vector(to_unsigned( 30 , 8)),
25                           2 => std_logic_vector(to_unsigned( 40 , 8)),
26                           3 => std_logic_vector(to_unsigned( 30 , 8)),
27                           4 => std_logic_vector(to_unsigned( 40 , 8)),
28                           5 => std_logic_vector(to_unsigned( 30 , 8)),
29                           6 => std_logic_vector(to_unsigned( 40 , 8)),
30                           7 => std_logic_vector(to_unsigned( 30 , 8)),
31                           8 => std_logic_vector(to_unsigned( 40 , 8)),
32                           9 => std_logic_vector(to_unsigned( 30 , 8)),
33                           10 => std_logic_vector(to_unsigned( 40 , 8)),
34                           11 => std_logic_vector(to_unsigned( 30 , 8)),
35                           12 => std_logic_vector(to_unsigned( 40 , 8)),
36                           13 => std_logic_vector(to_unsigned( 30 , 8)),
37                           14 => std_logic_vector(to_unsigned( 40 , 8)),
38                           15 => std_logic_vector(to_unsigned( 30 , 8)),
39                           16 => std_logic_vector(to_unsigned( 40 , 8)),
40                           17 => std_logic_vector(to_unsigned( 30 , 8)),
41                           18 => std_logic_vector(to_unsigned( 40 , 8)),
42              others => (others =>'0'));
43
44  component project_reti_logiche is
45  port (
```

project_reti_logiche.vhd | Test Bench di Esempio.vhd | Untitled 17

C:/Users/Simone/Desktop/progetto_reti/Test Bench di Esempio.vhd

```
92      end process;
93
94
95      test : process is
96      begin
97          wait for 100 ns;
98          wait for c_CLOCK_PERIOD;
99          tb_rst <= '1';
100         wait for c_CLOCK_PERIOD;
101         tb_rst <= '0';
102         wait for c_CLOCK_PERIOD;
103         tb_start <= '1';
104         wait for c_CLOCK_PERIOD;
105         wait until tb_done = '1';
106         wait for c_CLOCK_PERIOD;
107         tb_start <= '0';
108         wait until tb_done = '0';
109
110         -- Maschera di output = 11111111
111         assert RAM(19) = std_logic_vector(to_unsigned( 255 , 8)) report "TEST FALLITO" severity failure;
112
113         assert false report "Simulation Ended!, TEST PASSATO" severity failure;
114     end process test;
115
116     end projecttb;
117
```



2) TESTBENCH CREATO

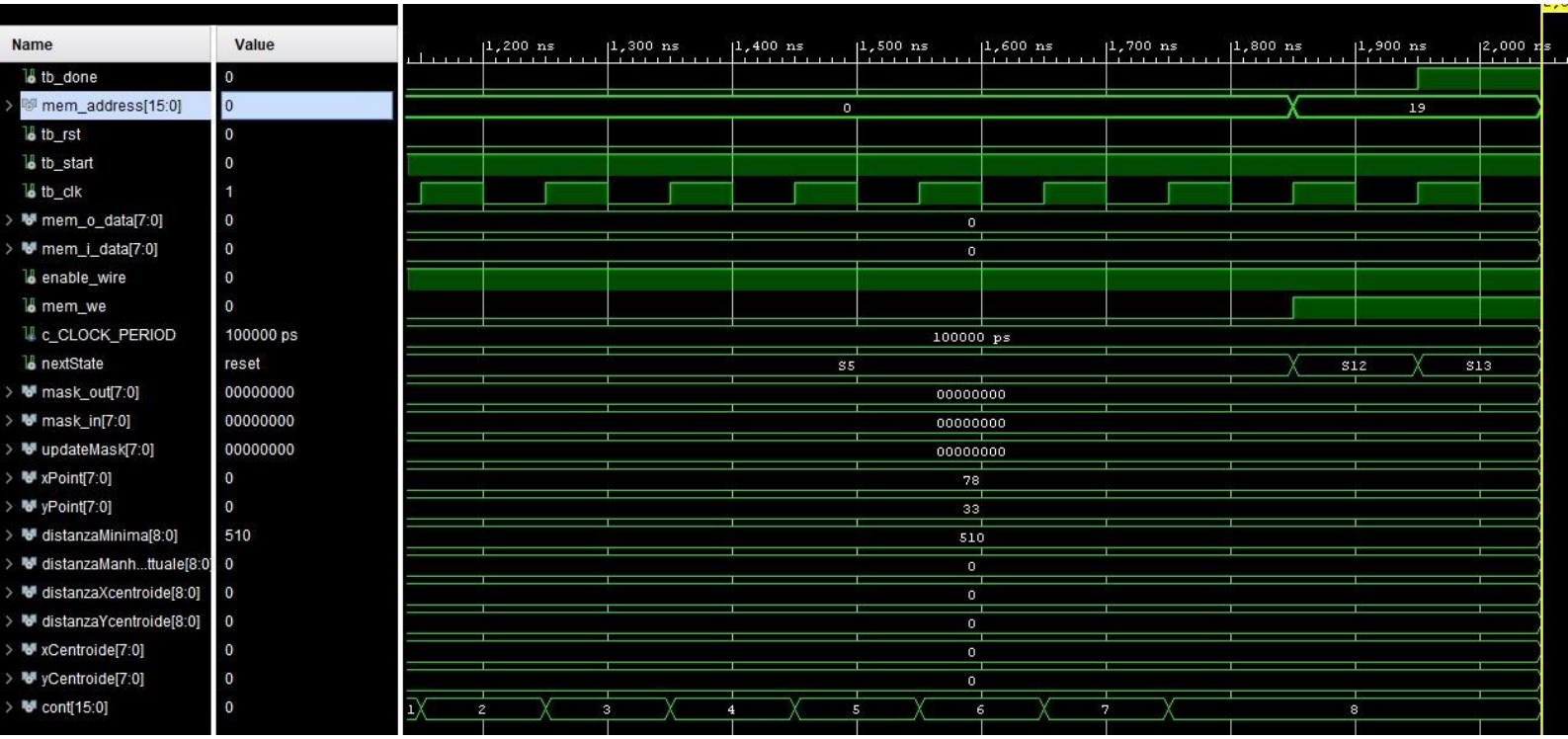
mask_in con tutti i bit che valgono 0 (la macchina non considera nessun centroide per la computazione); ci aspettiamo mask_out anch'essa di tutti 0:

```
19
20  type ram_type is array (65535 downto 0) of std_logic_vector(7 downto 0);
21
22  -- come da esempio su specifica
23  signal RAM: ram_type := (0 => std_logic_vector(to_unsigned( 0 , 8)),
24                           1 => std_logic_vector(to_unsigned( 75 , 8)),
25                           2 => std_logic_vector(to_unsigned( 32 , 8)),
26                           3 => std_logic_vector(to_unsigned( 111 , 8)),
27                           4 => std_logic_vector(to_unsigned( 213 , 8)),
28                           5 => std_logic_vector(to_unsigned( 79 , 8)),
29                           6 => std_logic_vector(to_unsigned( 33 , 8)),
30                           7 => std_logic_vector(to_unsigned( 1 , 8)),
31                           8 => std_logic_vector(to_unsigned( 33 , 8)),
32                           9 => std_logic_vector(to_unsigned( 80 , 8)),
33                           10 => std_logic_vector(to_unsigned( 35 , 8)),
34                           11 => std_logic_vector(to_unsigned( 12 , 8)),
35                           12 => std_logic_vector(to_unsigned( 254 , 8)),
36                           13 => std_logic_vector(to_unsigned( 215 , 8)),
37                           14 => std_logic_vector(to_unsigned( 78 , 8)),
38                           15 => std_logic_vector(to_unsigned( 211 , 8)),
39                           16 => std_logic_vector(to_unsigned( 121 , 8)),
40                           17 => std_logic_vector(to_unsigned( 78 , 8)),
41                           18 => std_logic_vector(to_unsigned( 33 , 8)),
42              others => (others =>'0'));
43
44  component project_reti_logiche is
```

project_reti_logiche.vhd | Test Bench di Esempio.vhd | Untitled 9

C:/Users/Simone/Desktop/Test Bench di Esempio.vhd

```
92      end process;
93
94
95      test : process is
96      begin
97          wait for 100 ns;
98          wait for c_CLOCK_PERIOD;
99          tb_rst <= '1';
100         wait for c_CLOCK_PERIOD;
101         tb_rst <= '0';
102         wait for c_CLOCK_PERIOD;
103         tb_start <= '1';
104         wait for c_CLOCK_PERIOD;
105         wait until tb_done = '1';
106         wait for c_CLOCK_PERIOD;
107         tb_start <= '0';
108         wait until tb_done = '0';
109
110         -- Maschera di output = 00000000
111         assert RAM(19) = std_logic_vector(to_unsigned( 0 , 8)) report "TEST FALLITO" severity failure;
112
113         assert false report "Simulation Ended!, TEST PASSATO" severity failure;
114     end process test;
115
116     end projecttb;
117
```



3) TESTBENCH CREATO

centroidi tutti coincidenti tra di loro ma non con il punto di riferimento.

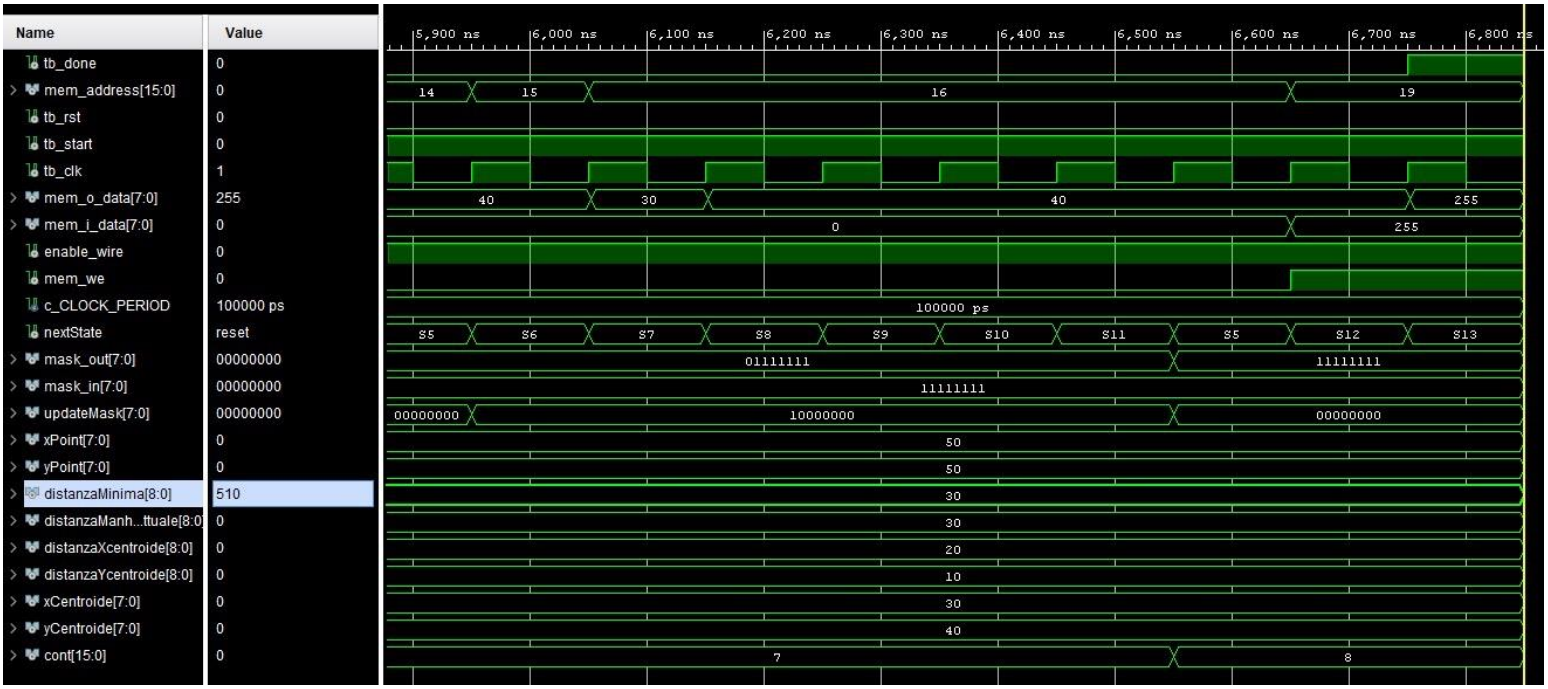
mask_in costituita da tutti 1 (la macchina studia tutti i centroidi); ci aspettiamo mask_out anch'essa di tutti 1:

```
20  type ram_type is array (65535 downto 0) of std_logic_vector(7 downto 0);
21
22  -- come da esempio su specifica
23  signal RAM: ram_type := (0 => std_logic_vector(to_unsigned( 255 , 8)),
24                           1 => std_logic_vector(to_unsigned( 30 , 8)),
25                           2 => std_logic_vector(to_unsigned( 40 , 8)),
26                           3 => std_logic_vector(to_unsigned( 30 , 8)),
27                           4 => std_logic_vector(to_unsigned( 40 , 8)),
28                           5 => std_logic_vector(to_unsigned( 30 , 8)),
29                           6 => std_logic_vector(to_unsigned( 40 , 8)),
30                           7 => std_logic_vector(to_unsigned( 30 , 8)),
31                           8 => std_logic_vector(to_unsigned( 40 , 8)),
32                           9 => std_logic_vector(to_unsigned( 30 , 8)),
33                           10 => std_logic_vector(to_unsigned( 40 , 8)),
34                           11 => std_logic_vector(to_unsigned( 30 , 8)),
35                           12 => std_logic_vector(to_unsigned( 40 , 8)),
36                           13 => std_logic_vector(to_unsigned( 30 , 8)),
37                           14 => std_logic_vector(to_unsigned( 40 , 8)),
38                           15 => std_logic_vector(to_unsigned( 30 , 8)),
39                           16 => std_logic_vector(to_unsigned( 40 , 8)),
40                           17 => std_logic_vector(to_unsigned( 50 , 8)),
41                           18 => std_logic_vector(to_unsigned( 50 , 8)),
42              others => (others =>'0'));
43
44  component project_reti_logiche is
45  port (
```

project_reti_logiche.vhd | Test Bench di Esempio.vhd | Untitled 17

C:/Users/Simone/Desktop/progetto_reti/Test Bench di Esempio.vhd

```
92      end process;
93
94
95      test : process is
96      begin
97          wait for 100 ns;
98          wait for c_CLOCK_PERIOD;
99          tb_rst <= '1';
100         wait for c_CLOCK_PERIOD;
101         tb_rst <= '0';
102         wait for c_CLOCK_PERIOD;
103         tb_start <= '1';
104         wait for c_CLOCK_PERIOD;
105         wait until tb_done = '1';
106         wait for c_CLOCK_PERIOD;
107         tb_start <= '0';
108         wait until tb_done = '0';
109
110         -- Maschera di output = 11111111
111         assert RAM(19) = std_logic_vector(to_unsigned( 255 , 8)) report "TEST FALLITO" severity failure;
112
113         assert false report "Simulation Ended!, TEST PASSATO" severity failure;
114     end process test;
115
116     end projecttb;
117
```

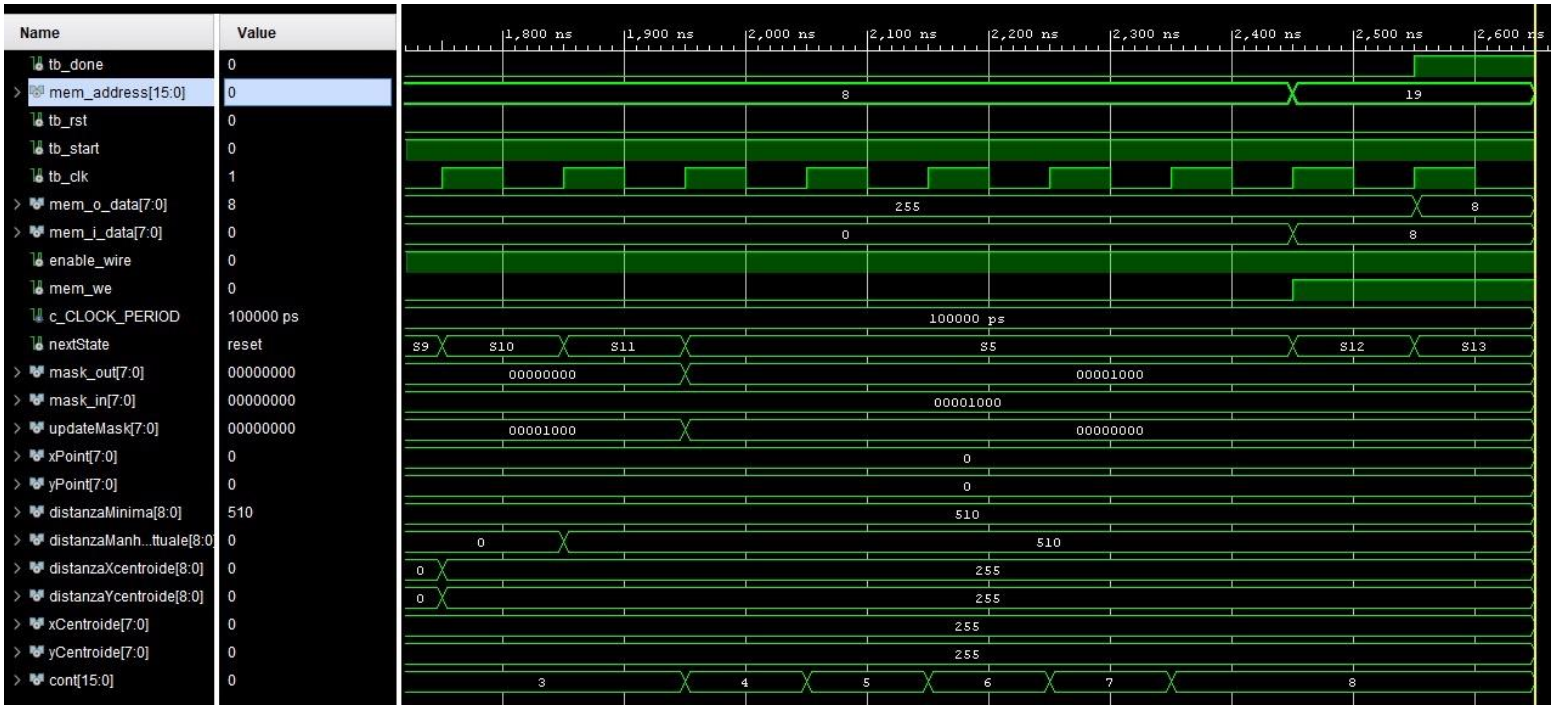


4) TESTBENCH CREATO

mask_in ha valore 8 (in decimale) quindi la macchina studia solamente il centroide in corrispondenza del 4° bit (celle 7 e 8 della memoria); questo centroide ha coordinate (255, 255) e il punto di riferimento ha coordinate (0, 0) quindi la distanza di Manhattan calcolata sarà la distanza massima ottenibile con queste specifiche (spazio bidimensionale 256x256); ci aspettiamo 'mask_out' uguale a 'mask_in' poiché stiamo considerando un solo centroide.
Questo test è stato utilizzato per verificare se la nostra macchina riuscisse a gestire in modo corretto la distanza massima ottenibile.

```
20      type ram_type is array (65535 downto 0) of std_logic_vector(7 downto 0);
21
22      -- come da esempio su specifica
23      signal RAM: ram_type := (0 => std_logic_vector(to_unsigned( 8 , 8)),
24                               1 => std_logic_vector(to_unsigned( 75 , 8)),
25                               2 => std_logic_vector(to_unsigned( 32 , 8)),
26                               3 => std_logic_vector(to_unsigned( 111 , 8)),
27                               4 => std_logic_vector(to_unsigned( 213 , 8)),
28                               5 => std_logic_vector(to_unsigned( 79 , 8)),
29                               6 => std_logic_vector(to_unsigned( 33 , 8)),
30                               7 => std_logic_vector(to_unsigned( 255 , 8)),
31                               8 => std_logic_vector(to_unsigned( 255 , 8)),
32                               9 => std_logic_vector(to_unsigned( 80 , 8)),
33                               10 => std_logic_vector(to_unsigned( 35 , 8)),
34                               11 => std_logic_vector(to_unsigned( 12 , 8)),
35                               12 => std_logic_vector(to_unsigned( 254 , 8)),
36                               13 => std_logic_vector(to_unsigned( 215 , 8)),
37                               14 => std_logic_vector(to_unsigned( 78 , 8)),
38                               15 => std_logic_vector(to_unsigned( 211 , 8)),
39                               16 => std_logic_vector(to_unsigned( 121 , 8)),
40                               17 => std_logic_vector(to_unsigned( 0 , 8)),
41                               18 => std_logic_vector(to_unsigned( 0 , 8)),
42              others => (others =>'0'));
43
44      component project_reti_logiche is
45      port (
```

```
92      end process;
93
94
95      test : process is
96      begin
97          wait for 100 ns;
98          wait for c_CLOCK_PERIOD;
99          tb_rst <= '1';
100         wait for c_CLOCK_PERIOD;
101         tb_rst <= '0';
102         wait for c_CLOCK_PERIOD;
103         tb_start <= '1';
104         wait for c_CLOCK_PERIOD;
105         wait until tb_done = '1';
106         wait for c_CLOCK_PERIOD;
107         tb_start <= '0';
108         wait until tb_done = '0';
109
110         -- Maschera di output = 00001000
111         assert RAM(19) = std_logic_vector(to_unsigned( 8 , 8)) report "TEST FALLITO" severity failure;
112
113         assert false report "Simulation Ended!, TEST PASSATO" severity failure;
114     end process test;
115
116     end projecttb;
117
```



5) TESTBENCH CREATO

valori presenti in memoria uguali a quelli del testbench fornito ma, durante la simulazione, il segnale di reset viene portato alto non appena il segnale di done comunica la fine dell'elaborazione.

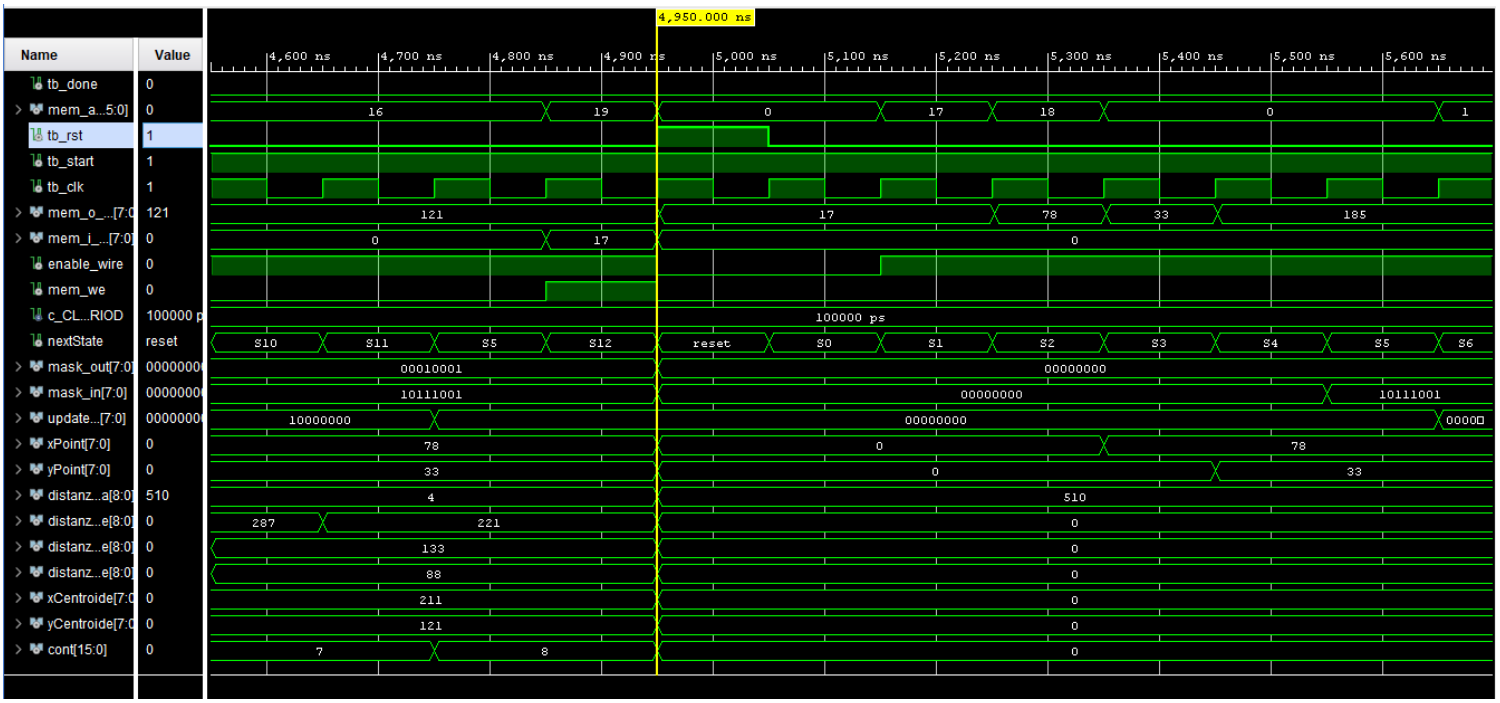
(riportiamo in sequenza le simulazioni relative al momento in cui arriva il segnale di reset e la fine della computazione).

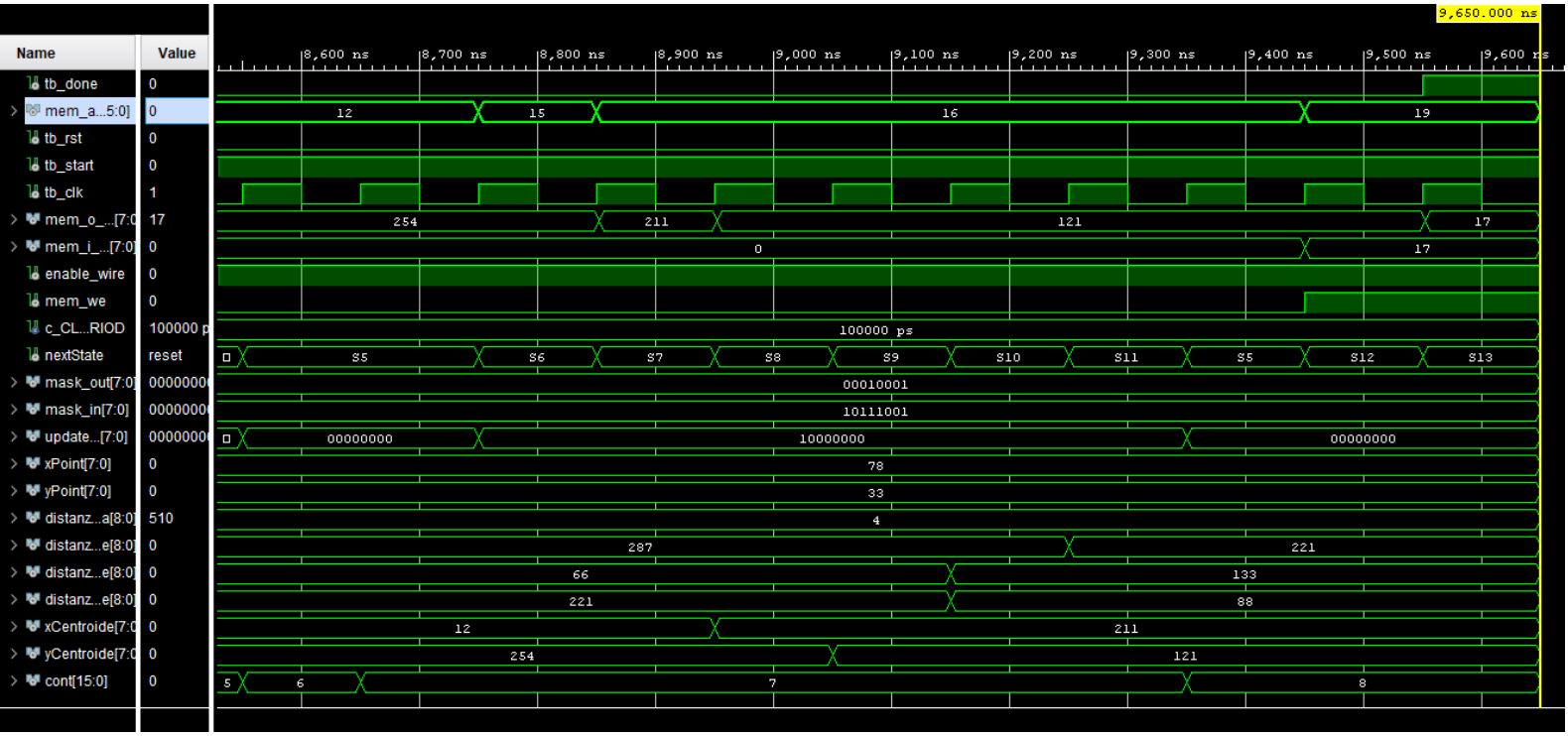
Abbiamo implementato questo test per verificare il corretto funzionamento di una seconda computazione nel caso in cui un segnale di reset interrompesse la prima.

fds.vhd | Test Bench di Esempio.vhd | Untitled 29

C:/Users/Lenovo_/Desktop/PROGETTO RETI LOGICHE/Test Bench di Esempio.vhd

```
95      test : process is
96      begin
97          wait for 100 ns;
98          wait for c_CLOCK_PERIOD;
99          tb_rst <= '1';
100         wait for c_CLOCK_PERIOD;
101         tb_rst <= '0';
102         wait for c_CLOCK_PERIOD;
103         tb_start <= '1';
104         wait for c_CLOCK_PERIOD;
105         wait until tb_done = '1';
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;
112         wait until tb_done = '1';
113         wait for c_CLOCK_PERIOD;
114         tb_start <= '0';
115         wait until tb_done = '0';
116
117         -- Maschera di output = 00010001
118         assert RAM(19) = std_logic_vector(to_unsigned( 17,8)) report "TEST FALLITO" severity failure;
119
120         assert false report "Simulation Ended!, TEST PASSATO" severity failure;
```


4,950.000 ns
Name
Value
tb_done
mem_a...5:0]
tb_rst
tb_start
tb_clk
mem_o_...[7:0]
mem_i_...[7:0]
enable_wire
mem_we
c_CL...RIOD
nextState
mask_out[7:0]
mask_in[7:0]
update...[7:0]
xPoint[7:0]
yPoint[7:0]
distanz...a[8:0]
distanz...e[8:0]
distanz...e[8:0]
distanz...e[8:0]
xCentroide[7:0]
yCentroide[7:0]
cont[15:0]
100000 ps
reset
S10
S11
S5
S12
S0
S1
S2
S3
S4
S6

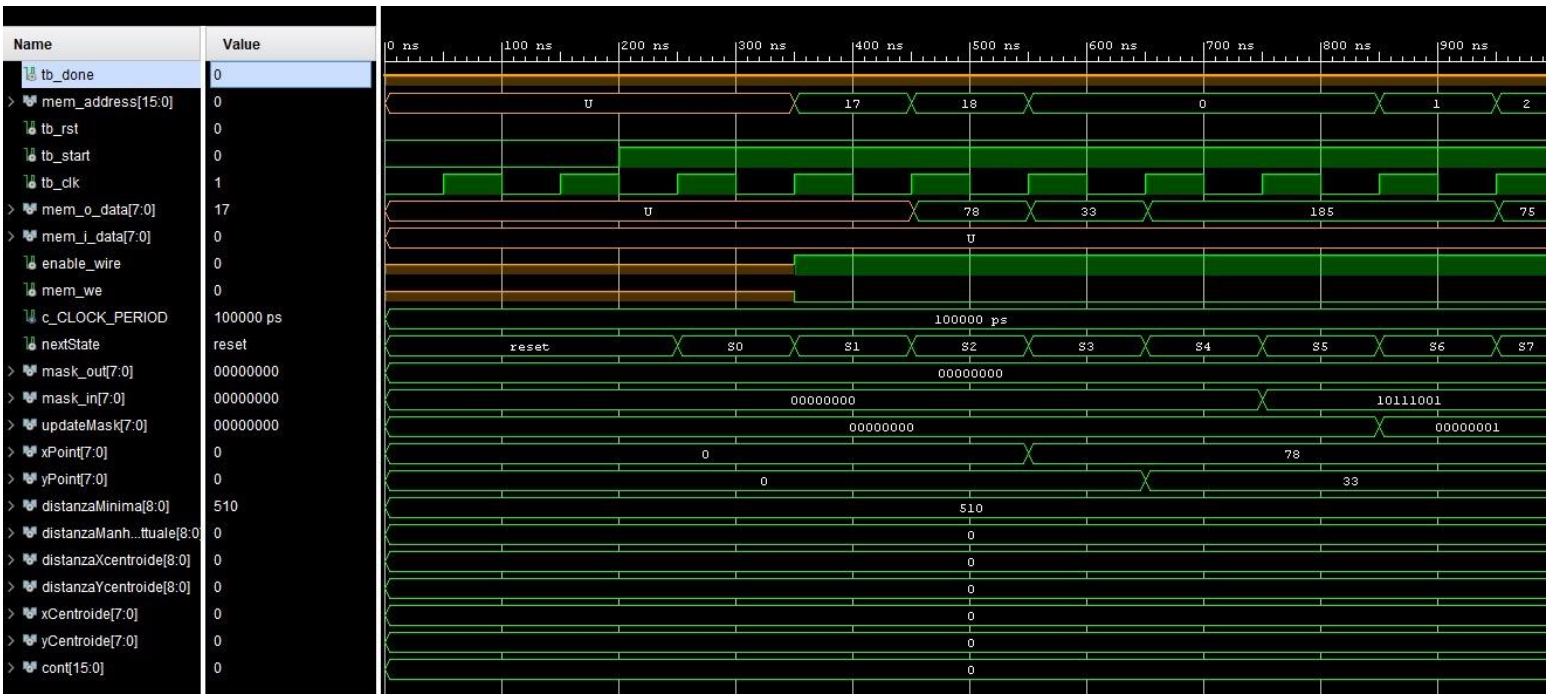

9,650.000 ns
Name
Value
tb_done
mem_a...5:0]
tb_rst
tb_start
tb_clk
mem_o_...[7:0]
mem_i_...[7:0]
enable_wire
mem_we
c_CL...RIOD
nextState
mask_out[7:0]
mask_in[7:0]
update...[7:0]
xPoint[7:0]
yPoint[7:0]
distanz...a[8:0]
distanz...e[8:0]
distanz...e[8:0]
distanz...e[8:0]
xCentroide[7:0]
yCentroide[7:0]
cont[15:0]
100000 ps
S5
S6
S7
S8
S9
S10
S11
S5
S12
S13

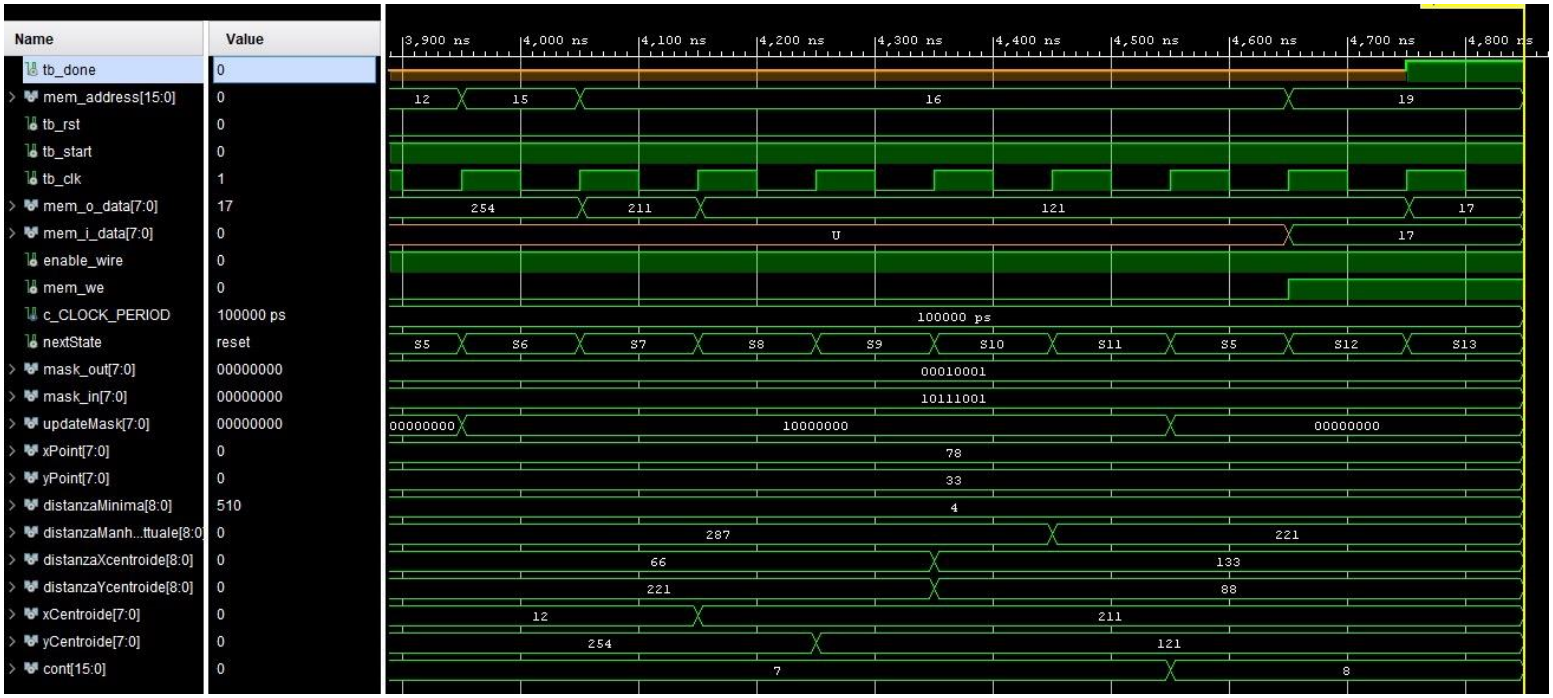
6) TESTBENCH CREATO

valori presenti in memoria uguali a quelli del testbench fornito. Verifichiamo se la macchina riesce a terminare la computazione in modo corretto anche se prima della computazione non arriva un segnale di reset.

(riportiamo in sequenza le simulazioni relative al momento di inizio e di fine dell'elaborazione):

```
88              mem_o_data <= RAM(conv_integer(mem_address)) after 2 ns;
89          end if;
90      end if;
91  end if;
92  end process;
93
94
95  test : process is
96  begin
97      wait for 100 ns;
98      wait for c_CLOCK_PERIOD;
99      tb_start <= '1';
00      wait for c_CLOCK_PERIOD;
01      wait until tb_done = '1';
02      wait for c_CLOCK_PERIOD;
03      tb_start <= '0';
04      wait until tb_done = '0';
05
06      -- Maschera di output = 00010001
07      assert RAM(19) = std_logic_vector(to_unsigned( 17 , 8)) report "TEST FALLITO" severity failure;
08
09      assert false report "Simulation Ended!, TEST PASSATO" severity failure;
10  end process test;
11
12  end projecttb;
13
```





Per tutti i test fatti abbiamo verificato anche il corretto funzionamento in post-sintesi, tutti terminati con esito positivo; abbiamo documentato i risultati di post-sintesi solo per il primo test in quanto, i risultati ottenuti, erano per ogni test uguali alla behavioral simulation.