Prova finale (Progetto di Reti Logiche) [051228]

Marini Gabriel Raul [10575543]

Morreale Federico [10561624]

15/03/2019

In breve

Lo scopo del progetto è quello di realizzare un componente HW in VHDL che data una matrice di dimensione 256 x 256 e le coordinate di 8 punti, detti “centroidi”, appartenenti a tale spazio, calcola quale/li centroide/i siano quelli più vicini a un punto assegnato. La vicinanza viene espressa tramite una maschera di 8 bit che vengono posti a 1 se il centroide corrispondente alla posizione è il più vicino al punto, 0 altrimenti. In ingresso viene inoltre fornita una maschera che determina quali centroidi considerare, di conseguenza il risultato dell’elaborazione è un “AND” tra la maschera in ingresso e il vettore prodotto dall’elaborazione del circuito.

1 Introduzione

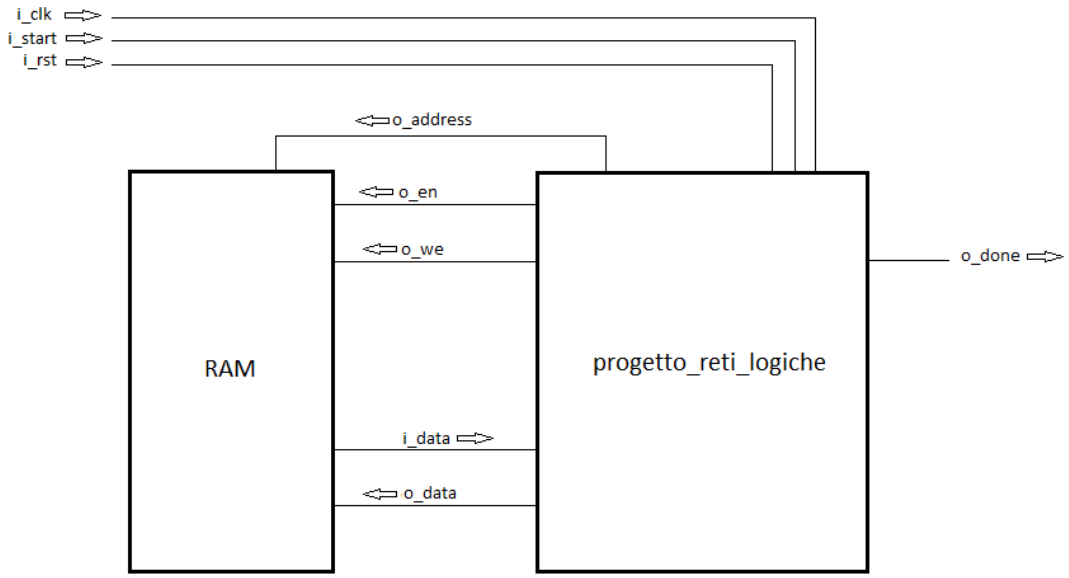
Come da specifica, l'interfaccia del componente è definita come segue:

```
entity project_reti_logiche is
port (
    i_clk : in std_logic;
    i_start : in std_logic;
    i_rst : in std_logic;
    i_data : in std_logic_vector(7 downto 0);
    o_address : out std_logic_vector(15 downto 0);
    o_done : out std_logic;
    o_en : out std_logic;
    o_we : out std_logic;
    o_data : out std_logic_vector (7 downto 0)
) ;
end project_reti_logiche;
```

Il modulo comunica con una memoria **RAM** al cui interno sono precaricati la maschera dei centroidi da considerare, le coordinate dei centroidi stessi e le coordinate del punto da cui calcolare la distanza. La memoria viene abilitata tramite il segnale **o_en** che viene posto a 1 per le operazioni che richiedono di leggerla o scriverla; nell'ultimo caso (scrittura) deve essere posto a 1 anche il segnale **o_we** di **WRITE ENABLE**. **i_data** e **o_data** sono i vettori da e verso la memoria che leggono/scrivono la parola all'indirizzo **o_address**. I segnali **i_clk, i_start** e **i_rst** sono generati internamente dal Test Bench; **i_clk**, come suggerisce il nome, rappresenta il segnale di clock con il quale opera il componente mentre **i_rst** inizializza la macchina per ricevere il primo segnale di **i_start** che avvia la computazione. Infine **o_done** segnala il termine della computazione dopo aver scritto il risultato in memoria.

2 Architettura

Come suggerisce la specifica, la soluzione più ovvia è quella di usare una macchina a stati finiti per i vari step dell'elaborazione. Trattandosi di un problema di complessità ridotta abbiamo optato per una soluzione monomodulare con un singolo process che scandisce i vari stati ed esegue le operazioni elementari nel contesto della soluzione.



2.1 Tabella dei signal interni

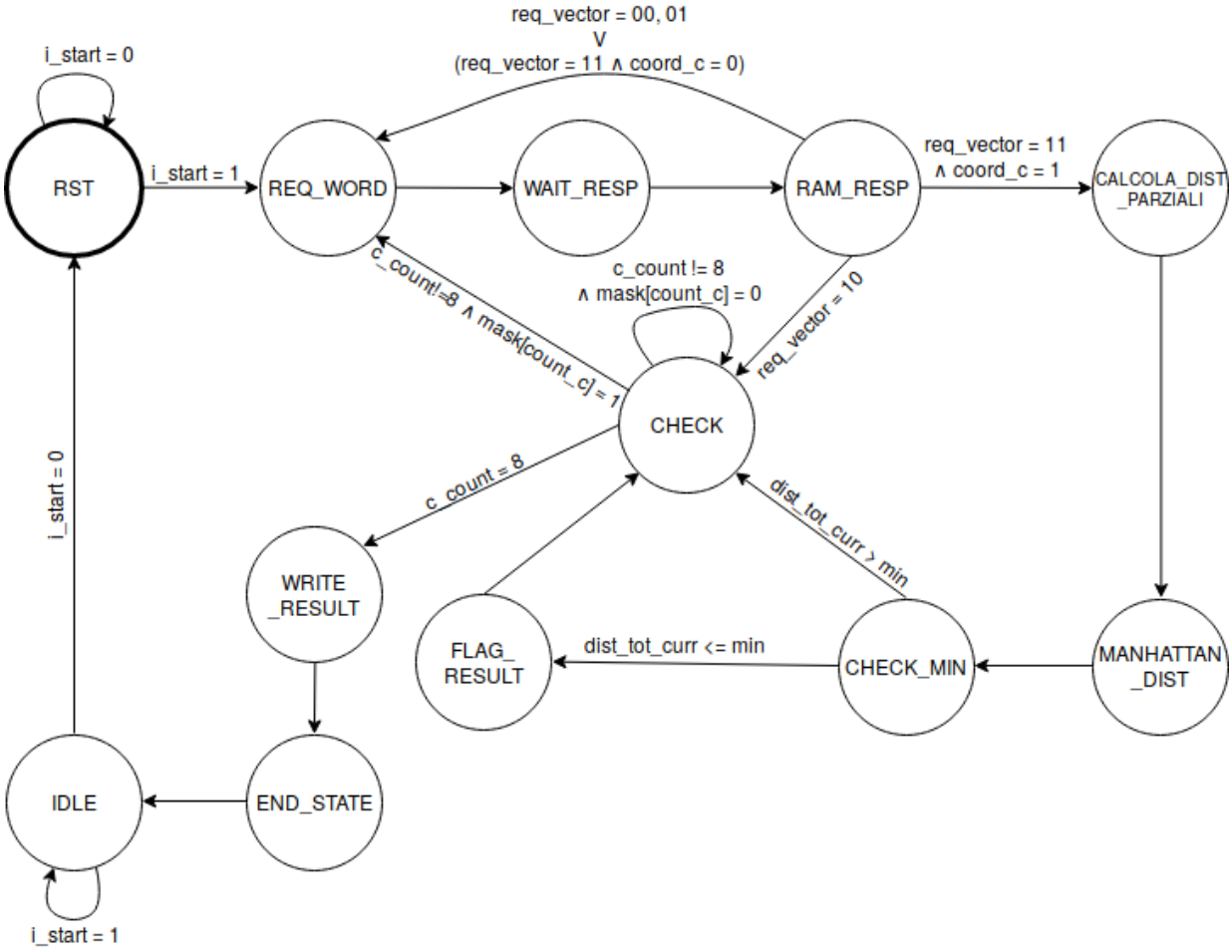
Avendo utilizzato i signal secondo il pattern standard di VHDL per l'implementazione di un flip-flop, tutti quelli riportati nella tabella sono da intendersi come registri.

Nome	Tipo	Valore iniziale	Descrizione
CURR_S	STATUS	U*	Memorizza lo stato corrente.
ADDRESS	std_logic_vector(15 downto 0)	0000000000000000	Memorizza l'indirizzo di memoria che si vuole accedere.
mask	std_logic_vector(7 downto 0)	00000000	Memorizza la maschera dei centroidi validi acquisita dalla RAM.
curr_x	std_logic_vector(7 downto 0)	00000000	Coordinata x del centroide corrente.
curr_y	std_logic_vector(7 downto 0)	00000000	Coordinata y del centroide corrente.

point_x	std_logic_vector(7 downto 0)	00000000	Coordinata x del punto da cui calcolare la distanza.
point_y	std_logic_vector(7 downto 0)	00000000	Coordinata y del punto da cui calcolare la distanza.
dist_x_curr	std_logic_vector(7 downto 0)	00000000	Distanza sull'asse delle ascisse tra il centroide corrente e il punto.
dist_y_curr	std_logic_vector(7 downto 0)	00000000	Distanza sull'asse delle ordinate tra il centroide corrente e il punto.
dist_tot_curr	std_logic_vector(8 downto 0)	000000000	Distanza Manhattan dal centroide corrente al punto.
min	std_logic_vector(8 downto 0)	111111111	Distanza minima corrente(di default assume il massimo valore).
result	std_logic_vector(7 downto 0)	00000000	Memorizza la maschera da salvare in memoria.
c_count	std_logic_vector(3 downto 0)	0000	Contatore che tiene traccia del centroide corrente.
req_vector	std_logic_vector(1 downto 0)	00	Registro utilizzato nello stato req_word per la comunicazione con la RAM.
coord_c	std_logic	0	Indica quale coordinata del centroide si sta acquisendo(0 si riferisce all'ascissa x, 1 all'ordinata y).

(*) Unico valore non inizializzato nel process, tutti i registri interni vengono inizializzati nello stato RST.

2.2 FSM



2.2.1 RST *state*

E' lo stato iniziale in cui vengono effettuate tutte le operazioni di inizializzazione dei registri; attende il segnale di **start** per dare inizio alla computazione ed è lo stato di destinazione ogni qualvolta il segnale **i_rst** viene posto a 1.

2.2.2 req_word *state*

In questo stato si imposta **o_address** all'indirizzo della parola di memoria che si vuole leggere dalla RAM. Il prossimo stato è **wait_resp**.

2.2.3 wait_resp *state*

Questo stage intermedio consiste semplicemente nell'attendere un ciclo di clock per permettere a tutti i segnali di assestarsi. In questo modo si ha la sicurezza che la

comunicazione con la RAM così come l'andamento dei segnali interni presentino un margine di tolleranza a eventuali ritardi non previsti.

2.2.4 RAM_resp *state*

E' lo stato in cui i valori acquisiti dalla RAM vengono memorizzati nei registri interni per le successive fasi elaborative. In base al valore contenuto nei registri **req_vector** e **coord_c** la parola proveniente da **i_data** viene memorizzata nei registri delegati al salvataggio delle coordinate del punto, del centroide corrente o semplicemente della maschera dei centroidi da considerare.

2.2.5 check *state*

All'interno di questo stato si controllano quanti centroidi rimangono da leggere e se questi vadano considerati come indicato dalla maschera in ingresso. Se il bit corrispondente a un determinato centroide è posto a 0 si passa a quello successivo incrementando **cont_c;** nessun'altra operazione viene effettuata. In caso contrario si ritorna allo stato **req_word** per richiedere la coordinata x del centroide in posizione **cont_c**. Se il contatore dei centroidi indica che tutti i dati utili sono stati considerati si passa allo stato **write_result**.

2.2.6 partial_dist, manhattan_dist, check_min, flag_result

Negli stati **partial_dist** e **manhattan_dist** si calcola la distanza tra la posizione del centroide corrente e il punto usando come metrica la distanza di Manhattan:

$$dist_{Manhattan} = |x_{centroide} - x_{punto}| + |y_{centroide} - y_{punto}|$$

Nello stato **check_min** si valuta se la distanza corrente è minore della distanza minima trovata fino a quel momento della computazione: se è maggiore si ritorna a **check;** se è minore o uguale si va allo stato **flag_result**; nel caso sia strettamente minore prima della transizione si azzera la maschera di uscita e si aggiorna **min** con la distanza del nuovo centroide più vicino trovato. Lo stato **flag_result** semplicemente imposta a 1 il bit di maschera corrispondente al nuovo centroide più vicino trovato.

2.2.7 write_result *state*

In questo stato si procede con la scrittura del risultato finale in memoria. Come da specifica la memoria viene attivata in scrittura attivando il segnale **o_we** e impostando su **o_data** il valore che desidera scrivere. Si passa dunque allo stato **end_state**.

2.2.8 end_state

In questa fase si chiude la comunicazione con la RAM e si alza il segnale **o_done** per indicare la fine della computazione. Si passa dunque allo stato **idle**.

2.2.9 idle *state*

Come da specifica si attende che **i_start** venga messo a 0 per abbassare **o_done**. Se questo accade si ritorna allo stato **RST** in attesa di un altro segnale di start, altrimenti si rimane nello stato attuale.

2.3 Minimalità della FSM

La FSM realizzata non rappresenta la soluzione a numero di stati minimo in quanto è possibile ridurre ulteriormente il diagramma per ottenere una soluzione più compatta. Tuttavia abbiamo ritenuto che fosse la scelta migliore in termini di chiarezza e operabilità del codice, in quanto facilmente modificabile per eventuali ottimizzazioni future. Nel punto **3** la sintesi e i vari test evidenzieranno i risultati della nostra scelta progettuale.

3 Risultati sperimentali

3.1 Sintesi

Il componente è correttamente sintetizzabile e implementabile dal tool con un totale di 137 LUT e 136 FF.

e	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF
synth_2 (active)	**constrs_1**	**synth_design Complete!**								**137**	**136**
impl_2	constrs_1	Not started									
impl_3 (active)	**constrs_1**	**route_design Complete!**	**NA**	**NA**	**NA**	**NA**	**NA**	**5.567**	**0**	**137**	**136**

3.2 Simulazioni

In seguito verranno elencati i casi di test ritenuti significativi ai fini del corretto funzionamento del componente. Essendo stato sottoposto a numerose simulazioni, abbiamo ritenuto che illustrare le più importanti fosse più sensato che proporre molti test bench simili tra loro.

3.2.1 TB generico fornito dal docente

Questo test bench va a studiare il comportamento del componente in una situazione di carattere generale con un clock operativo di 100ns e una memoria implementata tramite un array di byte. La simulazione ha evidenziato il superamento del test in Behavioral, Post-Synthesis Functional e Timing (anche Post-Implementation Functional e Timing hanno ottenuto risultati corretti). I due centroidi validi più vicini sono il primo e il quinto.

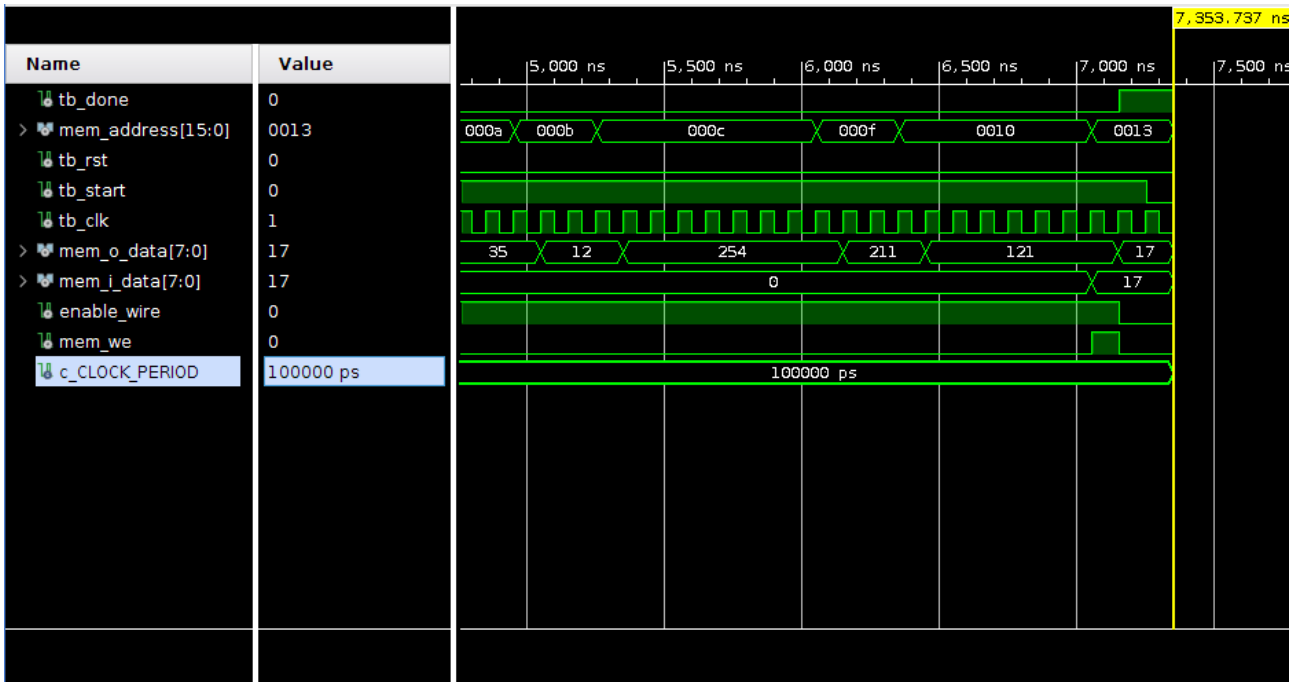
N_Centroide	Posizione	Da considerare
1	(75, 32)	S
2	(111, 213)	N
3	(79, 33)	N
4	(1, 33)	S
5	(80, 35)	S
6	(12, 254)	S
7	(215, 78)	N
8	(211, 121)	S
PUNTO	(78, 33)	--

T_{ck} = 100ns

Maschera centroidi: 185 (10111001)

Risultato atteso: 17 (00010001 – 11 HEX)

Risultato post-synthesis timing:



3.2.2 TB Casi particolari

I seguenti test verranno eseguiti con clock di 100ns e con struttura del test bench coincidente a quella data dal professore. Tutti i test hanno avuto esito positivo in Behavioral, Post-Synthesis Functional e Timing simulation.

3.2.2.1 Test 1

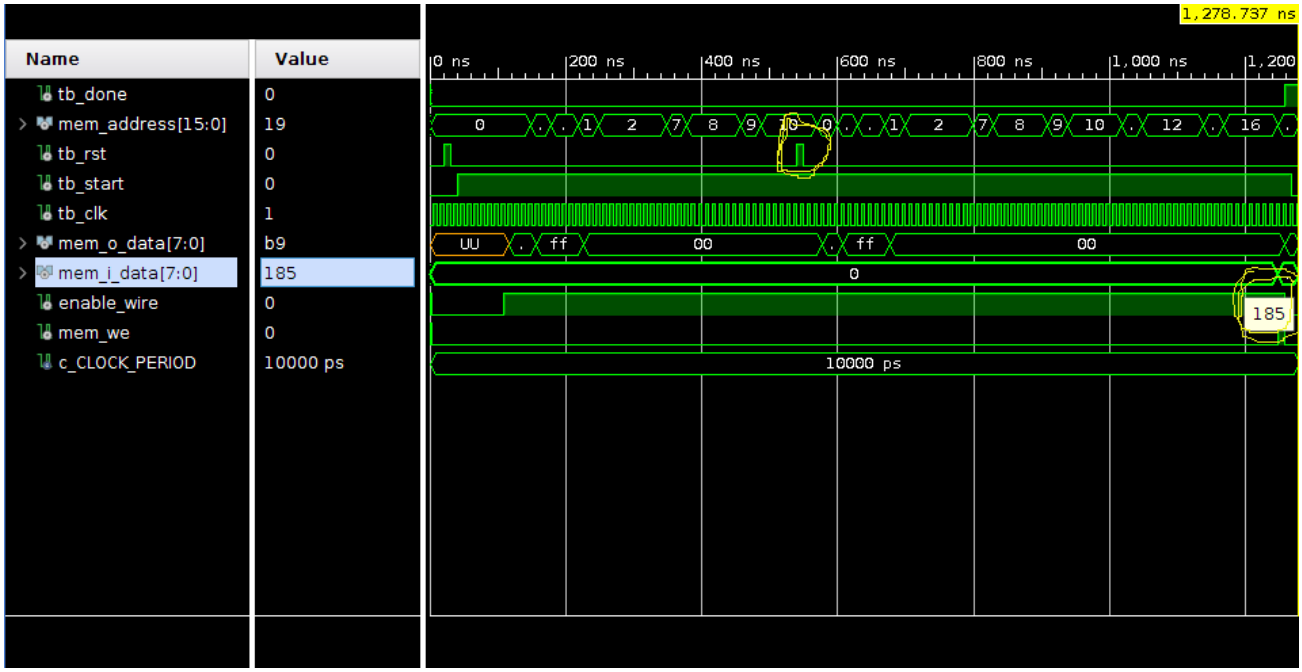
Questo test bench stimola il funzionamento del componente in condizioni insolite. La presenza di N punti coincidenti posti al limite dello spazio (0,0) rispetto al punto (255, 255) ha lo scopo di evidenziare possibili errori quali insufficiente spazio allocato per la memorizzazione delle distanze e corretta elaborazione della maschera d'ingresso. In questo caso il risultato coincide con la maschera in ingresso essendo tutti i centroidi coincidenti e di conseguenza equidistanti dal punto. E' stato inoltre introdotto un segnale di reset nel mezzo della computazione per verificare il corretto riassestamento dei parametri nel caso questo avvenga "inaspettatamente".

N_Centroide	Posizione	Da considerare
1	(0, 0)	S
2	(0, 0)	N
3	(0, 0)	N
4	(0, 0)	S
5	(0, 0)	S
6	(0, 0)	S
7	(0, 0)	N
8	(0, 0)	S
PUNTO	(255, 255)	--

Maschera centroidi: 185 (10111001)

Risultato atteso: 185(10111001 – B9 HEX)

Risultato post-synthesis timing:



3.2.2.2 Test 2

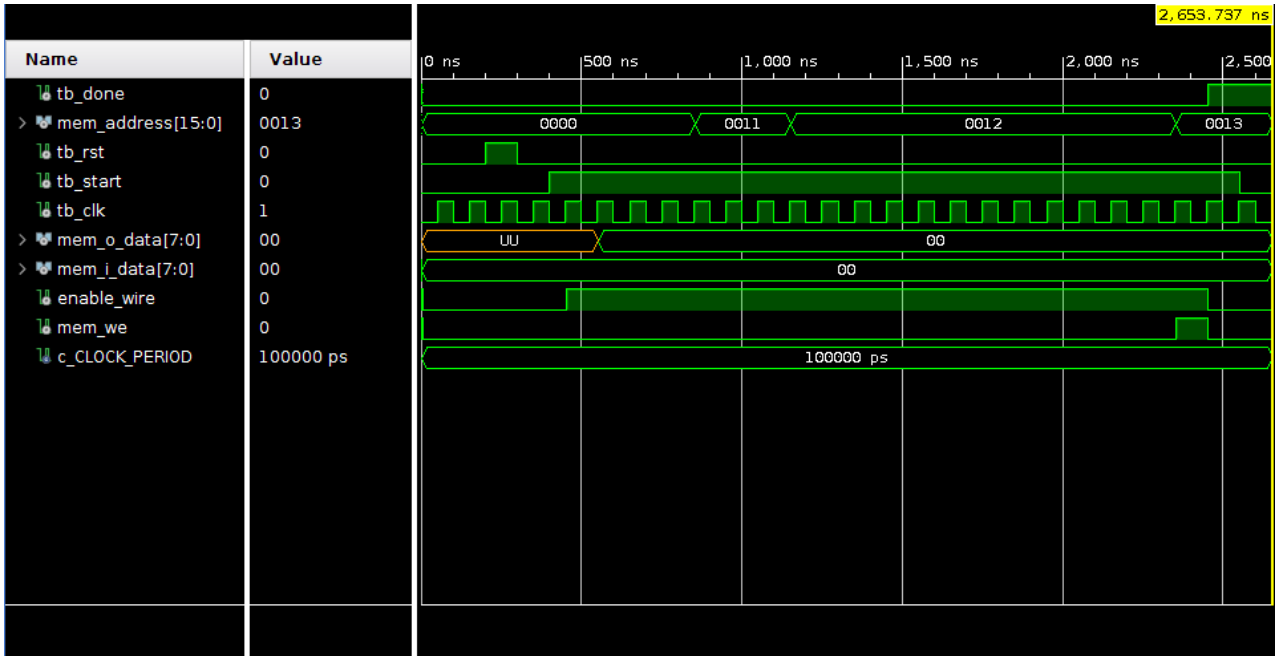
In questo test nessun centroide viene considerato, quindi per qualsiasi valore associato ad ognuno di essi l'output atteso dovrà coincidere con la maschera di ingresso, ovvero 8 bit di '0'. Si può notare una certa velocità di computazione in quanto nessuna operazione di calcolo delle distanze viene effettuata.

N_Centroide	Posizione	Da considerare
1	(0, 0)	N
2	(0, 0)	N
3	(0, 0)	N
4	(0, 0)	N
5	(0, 0)	N
6	(0, 0)	N
7	(0, 0)	N
8	(0, 0)	N
PUNTO	(0, 0)	--

Maschera centroidi: 0 (00000000)

Risultato atteso: 0 (00000000 – 00 HEX)

Risultato post-synthesis timing:



3.2.2.3 Test 3

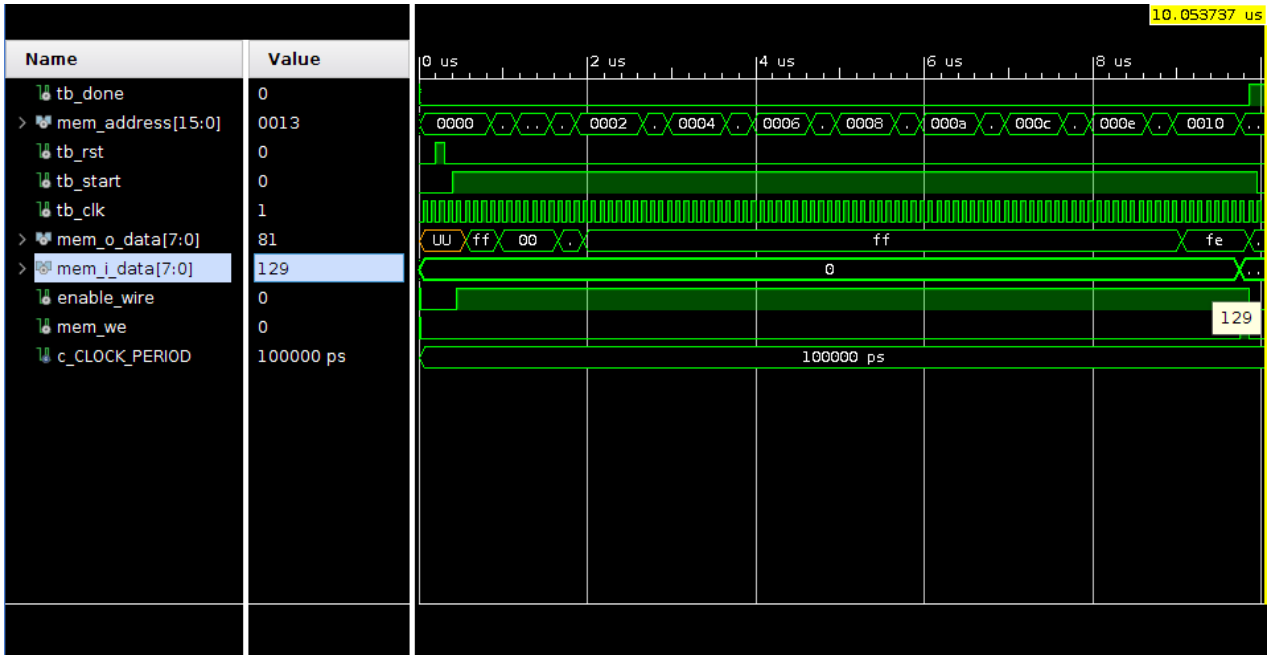
In questo test tutti i centroidi vengono considerati. I centroidi 2, 3, 4, 5, 6, 7 sono coincidenti e hanno coordinata (255, 255) mentre i centroidi 1 e 8 hanno stessa distanza totale ma sono posti rispettivamente in posizione (255, 254) e (254, 255), in modo da andare a testare il corretto funzionamento del calcolo della distanza sulla coordinata x ed y. Il punto da considerare per il calcolo della distanza è invece posto a (0,0). L'output atteso dovrà quindi avere '1' sul primo ed ultimo bit, mentre i restanti saranno posti a '0'.

N_Centroide	Posizione	Da considerare
1	(255, 254)	S
2	(255, 255)	S
3	(255, 255)	S
4	(255, 255)	S
5	(255, 255)	S
6	(255, 255)	S
7	(255, 255)	S
8	(254, 255)	S
PUNTO	(0, 0)	--

Maschera centroidi: 255 (11111111)

Risultato atteso: 129 (10000001 – 81 HEX)

Risultato post-synthesis timing:


Name
Value
tb_done
0
mem_address[15:0]
0013
tb_rst
0
tb_start
0
tb_clk
1
mem_o_data[7:0]
81
mem_i_data[7:0]
129
enable_wire
0
mem_we
0
c_CLOCK_PERIOD
100000 ps
10.053737 us
0 us
2 us
4 us
6 us
8 us
0000
0002
0004
0006
0008
000a
000c
000e
0010
UU
ff
00
ff
fe
0
129
100000 ps