PROGETTO
RETI LOGICHE A.A. 2018-2019

Membri del gruppo:

Marchisciana Matteo – Matricola 870199 – Codice Persona 10586574
Marcer Andrea – Matricola 868629 – Codice Persona 10537040

INDICE

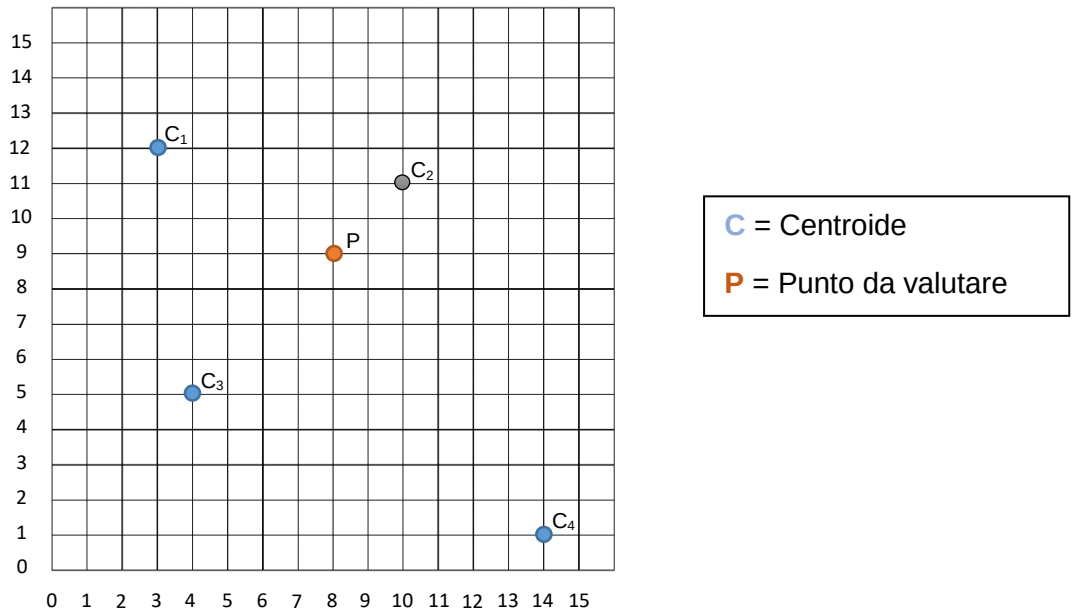
I. Introduzione

1.1 Problema

Sia dato uno spazio $\mathbb{N}$x$\mathbb{N}$ e un insieme di punti, detti "centroidi", appartenenti a tale spazio. Si vuole realizzare un componente Hardware descritto in VHDL che, dato l'insieme di punti e scelto tra essi un punto, restituisca il/i centroide/i ad esso più vicino/i considerando come distanza la distanza di Manhattan.
Nello specifico lo spazio considerato ha dimensione finita pari a 256x256 e l'insieme dei punti è formato da 8 centroidi e dal punto da valutare. Dei primi solo K devono essere considerati nel calcolo delle distanze. L'informazione su quali ignorare e quali considerare viene fornita da una maschera di ingresso a 8bit: se l'i-esimo bit, partendo da quello meno significativo, è posto a 1 allora il centroide deve essere considerato, se è posto a 0 deve essere ignorato. Anche il risultato della computazione dovrà essere una maschera in cui i bit a 1 indicano quale/i centroide/i è/sono più vicino/i.

Di seguito è raffigurato un esempio per comprendere meglio il problema.
Si consideri uno spazio di dimensione 16x16 al quale appartengono 5 punti e la seguente maschera di ingresso "1101".

Figura 1: esempio del problema da risolvere



In arancione è evidenziato il punto da valutare. Prendendo in considerazione la maschera d'ingresso gli unici centroidi da considerare (in blu) sono C_1, C_3 e C_4, mentre C_2 (in grigio) deve essere ignorato nel conteggio delle distanze.
La distanza di Manhattan tra due punti viene calcolata come segue: $d_{12} = |x_1 - x_2| + |y_1 - y_2|$
Ad esempio, la distanza tra P e C_4 è $d_{P4} = |x_P - x_4| + |y_P - y_4| = |8 - 14| + |9 - 1| = 14$.
Una volta calcolate tutte le distanze si può facilmente trovare che i centroidi a distanza minima sono C_1 e C_3.
Di conseguenza la maschera da tornare sarà "0101".
Si noti che il centroide a distanza minima sarebbe C_2, ma questo deve essere ignorato, di conseguenza la sua distanza non deve essere presa in considerazione.

1.2 Specifica

I dati del problema sono reperibili da una memoria a indirizzamento al Byte formata da 20 parole. Sarà possibile recuperare le informazioni tramite l'interfaccia dalla memoria stessa. I dati sono memorizzati in parole di 8 bit disposti nel seguente modo:

Indirizzo di memoria	**Contenuto**
0	Maschera di ingresso
1	X centroide 1
2	Y centroide 1
3	X centroide 2
4	Y centroide 2
5	X centroide 3
6	Y centroide 3
7	X centroide 4
8	Y centroide 4
9	X centroide 5
10	Y centroide 5
11	X centroide 6
12	Y centroide 6
13	X centroide 7
14	Y centroide 7
15	X centroide 8
16	Y centroide 8
17	X del punto da valutare
18	Y del punto da valutare
19	Maschera di uscita

L'elaborazione dovrà partire quando il segnale `START` sarà portato a 1. Esso rimarrà alto finché il segnale DONE non verrà posto alto. DONE rimarrà altro fintantoché anche il segnale di START rimarrà alto.

Inoltre, il componente da realizzare dovrà utilizzare la seguente interfaccia:

- `i_clk` è il segnale di CLOCK in ingresso generato dal Test Bench;
- `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;
- `o_en` è il segnale di ENABLE della memoria;
- `o_we` è il segnale di WRITE ENABLE della memoria da portare a 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.

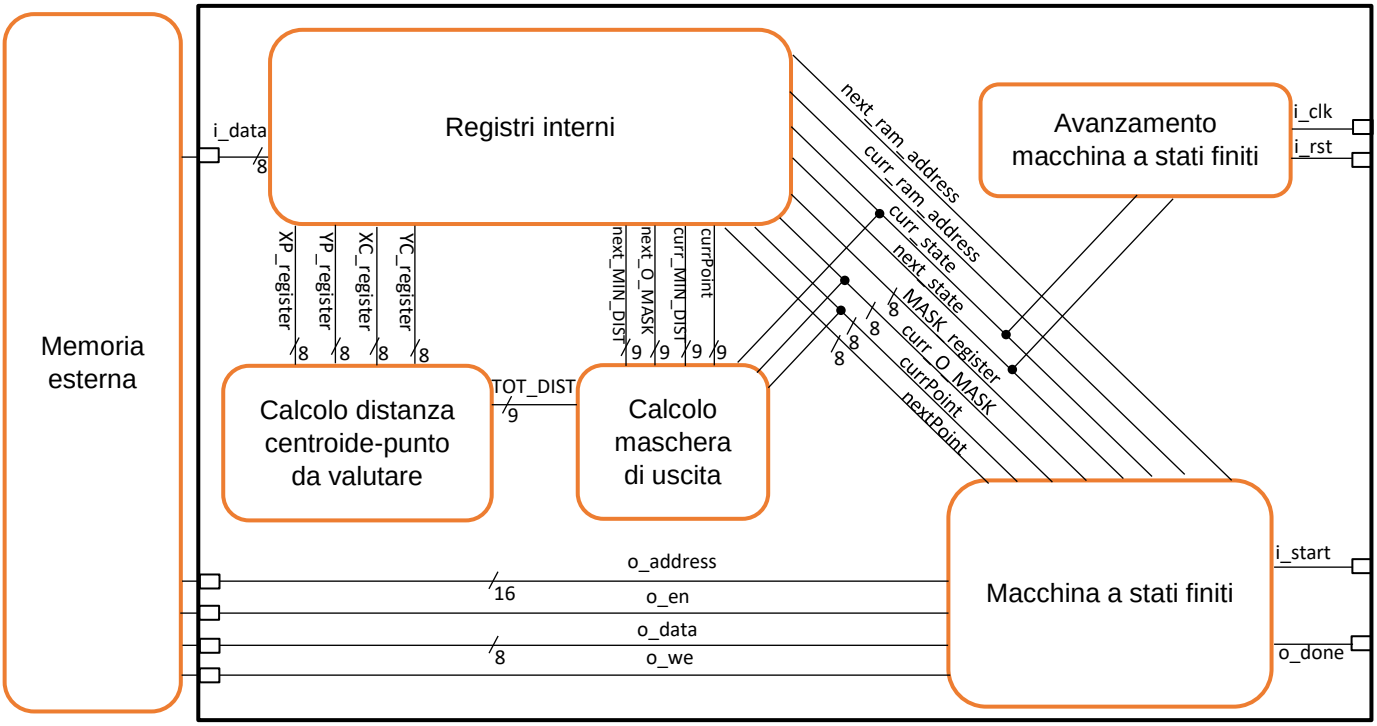
II. Soluzione

Durante la realizzazione del componente si è cercato di individuare un algoritmo capace di trovare la soluzione con l'utilizzo di meno cicli di clock possibili, prestando comunque attenzione all'aggiornamento dei vari registri. È stata quindi tralasciata la metrica d'area ritenuta meno importante data la dimensione della FPGA utilizzata secondo specifica.
Come primo passo per la progettazione del componente si è analizzato il problema con l'intento di individuare un algoritmo veloce e semplice che potesse portare alla soluzione.
Il problema è stato, dunque, suddiviso in tre fasi:

- Lettura da memoria dei dati necessari
- Ricerca del/dei centroide/i più vicino/i
- Presentazione della soluzione

Poiché ciascuna di queste fasi può essere suddivisa in ulteriori sottofasi si è deciso di adottare un algoritmo basato su una macchina a stati finiti. Risulterà così più facile tener conto dell'avanzamento dell'algoritmo e governare i vari segnali interni del componente.
In seguito, sono state individuate alcune funzionalità con l'intento di semplificare il lavoro compiuto dalla macchina a stati finiti che si dovrà occupare solamente di gestire l'interazione con la memoria esterna e delegherà la memorizzazione e il calcolo delle distanze ad altri moduli all'interno del componente.
Di conseguenza sono stati realizzati una serie di moduli dedicati ciascuno a supporto di una specifica funzionalità

Figura 2: schema funzionale del componente



2.1 Segnali interni

I segnali utilizzati per la comunicazione interna sono:

- `MASK_register` [`std_logic_vector(7 downto 0)`]: rappresenta la maschera in ingresso, il suo valore viene letto da memoria solamente una volta durante la fase di lettura dei dati;
- `XP_register` e `YP_register` [`std_logic_vector(7 downto 0)`]: rappresentano le coordinate X e Y del punto considerato, il loro valore viene letto da memoria solamente una volta durante la fase di lettura dei dati;
- `XC_register` e `YC_register` [`std_logic_vector(7 downto 0)`] : rappresentano le coordinate X e Y del centroide attualmente considerato, vengono aggiornati durante la fase di ricerca del centroide appena termina la computazione della distanza del centroide precedente;
- `curr_O_MASK` [`std_logic_vector(7 downto 0)`]: segnale che rappresenta momentaneamente la maschera da ritornare, viene aggiornata ogni volta che viene calcolata una nuova distanza minima;
- `X_DIST` e `Y_DIST` [`std_logic_vector(8 downto 0)`]: rappresentano le distanze lungo l'asse delle X e l'asse delle Y tra il punto considerato e il centroide corrente;
- `TOT_DIST` [`std_logic_vector(8 downto 0)`]: rappresenta la distanza di Manhattan tra il punto considerato e il centroide corrente;
- `curr_MIN_DIST` [`std_logic_vector(8 downto 0)`]: rappresenta la distanza di Manhattan minore finora calcolata;
- `currPoint` [`std_logic_vector(7 downto 0)`]: rappresenta l'indice del centroide attualmente considerato, viene utilizzata una codifica one-hot per facilitare sia la creazione della maschera da ritornare sia l'operazione d'incremento dell'indice, infatti basterà eseguire un semplice *shift left logic* del segnale stesso;
- `curr_ram_address` [`std_logic_vector(15 downto 0)`]: rappresenta l'indirizzo del registro della memoria che attualmente si sta richiedendo;
- `current_state` [`state_type`]: rappresenta lo stato attuale della macchina a stati finiti;
- `i_clk` [`std_logic`]: ad ogni rising edge aggiorna i valori degli altri segnali e fa avanzare la macchina stati finiti;
- `i_rst` [`in std_logic`]: porta la macchina a stati finiti nello stato di `WAIT_START` al ciclo di clock successivo (reset sincrono).

2.2 Registri interni

I dati di interesse vengono memorizzati all'interno del componente tramite l'istanziazione di alcuni flip-flop. Ad ogni dato, infatti, sono associati due segnali: il segnale corrente e il segnale successivo. Ad ogni rising edge del clock al segnale corrente viene assegnato il segnale successivo. Questo garantisce che i segnali utilizzati siano stati correttamente aggiornati e salvati evitando che i dati utilizzati possano commutare durante il loro utilizzo.
I dati che sfruttano questo sistema sono:

- La maschera in ingresso (`MASK`, `MASK_register`)
- Le coordinate X e Y del punto (`XP`, `XP_register`, `YP`, `YP_register`)
- Le coordinate X e Y del centroide corrente (`XC`, `XC_register`, `YC`, `YC_register`)
- La distanza minima finora calcolata (`curr_MIN_DIST`, `next_MIN_DIST`)
- La maschera di uscita (`curr_O_MASK`, `next_O_MASK`)
- L'indirizzo della memoria che si vuole richiedere (`curr_ram_address`, `next_ram_address`)
- L'indice del centroide attualmente considerato (`currPoint`, `nextPoint`)
- Lo stato corrente (`current_state`, `next_state`)

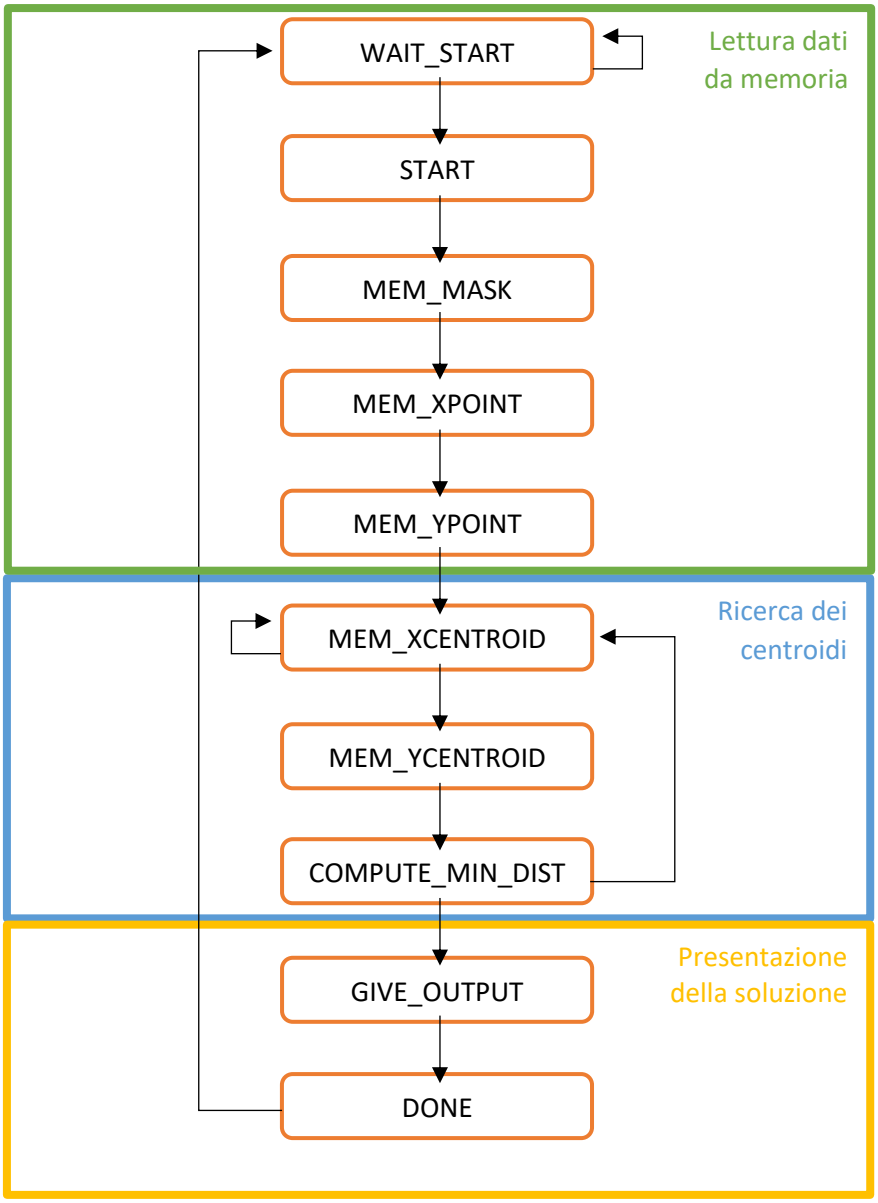
2.3 Avanzamento macchina a stati finiti

L'avanzamento della macchina a stati finiti e l'aggiornamento dei registri è delegato al process `register_process`. Ad ogni rising edge del segnale `i_clk` il process assegna a tutti i segnali correnti dei registri il segnale successivo. In questo modo la macchina a stati finiti avanza e tutti gli altri moduli sono sicuri che i segnali utilizzati rimarranno uguali per tutto il ciclo del clock.
Inoltre, questo process implementa il reset sincrono del componente: ad ogni rising edge del clock assegna a `next_state` lo stato `WAIT_START` e inizializza tutti i registri al loro valore di default.

2.4 Macchina a stati finiti

La macchina a stati finiti è stata realizzata tenendo in mente le tre fasi individuate precedentemente. Per ciascuna di esse sono state identificate una serie di sottofasi successivamente realizzate come stati della macchina. Ad ogni stato è delegato il compito di assegnare ai vari segnali interni del componente il valore corretto e di governare il funzionamento dei restanti moduli. Di seguito viene riportato lo schema della macchina in cui viene evidenziata la suddivisione nelle tre fasi dell'algoritmo.

Figura 3: schema delle fasi della macchina a stati finiti



Nome stato	Descrizione
WAIT_START	inizializza tutti i segnali in attesa del segnale i_start
START	richiede alla memoria il valore della maschera
MEM_MASK	memorizza la maschera di ingresso e richiede la coordinata X del punto considerato;
MEM_XPOINT	memorizza la coordinata X del punto da verificare e richiede la sua coordinata Y;
MEM_YPOINT	memorizza la coordinata Y del punto da verificare e richiede la coordinata X del primo centroide;
MEM_XCENTROID	se il centroide corrente deve essere considerato memorizza la coordinata X del centroide e richiede la sua coordinata Y, se il centroide deve essere ignorato richiede la coordinata X del centroide successivo, se i centroidi sono finiti passa alla fase di presentazione della soluzione
MEM_YCENTROID	memorizza la coordinata Y del centroide corrente
COMPUTE_MIN_DIST	consente di aggiornare la distanza minima e richiede la coordinata X del prossimo centroide
GIVE_OUTPUT	assegna al registro della memoria dedicato la maschera calcolata
DONE	comunica il termine della computazione

2.4 Ottimizzazioni

Durante la progettazione della macchina a stati si è cercato apportare alcune ottimizzazioni per velocizzare la computazione della soluzione:

- Se un centroide non deve essere considerato le sue coordinate non vengono salvate nei registri e la sua distanza non viene calcolata. Questo viene ottenuto aggiungendo un autoanello nello stato MEM_XCENTROID e incrementando immediatamente l'indice del centroide corrente salvato in currPoint;
- Se la maschera di ingresso è "00000000" oppure presenta solo una centroide da considerare la soluzione è banale, la macchina a stati passa direttamente nello stato GIVE_OUTPUT e alla maschera di uscita viene assegnato direttamente la maschera di ingresso;
- Dato che la memoria restituisce il valore richiesto al ciclo di clock successivo invece di adottare uno stato "*WAIT_RESULT*" le richieste per il dato d'interesse sono effettuate nello stato precedente. Ad esempio, nello stato START viene richiesto alla memoria la maschera di ingresso assegnando a o_address l'indirizzo "0000000000000000"; in seguito nello stato MEM_MASK viene assegnato a MASK il valore di i_data, al prossimo rising edge al MASK_register viene assegnato il valore di MASK appena letto. Grazie a questo meccanismo c'è la sicurezza che i dati utilizzati in ogni stato non possano commutare durante il ciclo di clock in cui vengono utilizzati per la computazione.

2.5 Calcolo distanza centroide-punto da valutare

Il modulo è realizzato tramite una rete combinatoria che ogni qualvolta la coordinata X e la coordinata Y del centroide cambiano calcola la nuova distanza di Manhattan dal punto da valutare. Poiché il modulo è combinatorio quando la macchina è nello stato MEM_XCENTROID nei registri sono memorizzate la coordinata X del centroide corrente e la coordinata Y del centroide precedente, dando così origine ad una distanza errata. Questo, però, non è un problema poiché la gestione della computazione e della memorizzazione della distanza minima è delegata al process compute_min_distance che viene eseguito solamente al momento corretto.

2.6 Calcolo maschera d'uscita

Il modulo è realizzato tramite il process `compute_min_distance`. Durante l'implementazione del process si è cercato di introdurre alcune ottimizzazioni per rendere la computazione veloce e chiara. Come spiegato nella sezione dei segnali, `currPoint` tiene conto dell'indice del centroide attualmente considerato tramite la codifica one-hot, in questo modo:

- Se il centroide è a distanza minore della distanza minima finora trovata è sufficiente assegnare a `next_O_MASK` il valore di `currPoint`;
- Se il centroide si trova a distanza uguale a quella minima finora trovata è sufficiente assegnare a `next_O_MASK` l'OR tra `curr_O_MASK` e `currPoint`;
- Se invece il centroide è a distanza maggiore di quella minima `next_O_MASK` non viene modificato.

III. Risultati della sintesi

3.1. Registri sintetizzati

Facendo riferimento al Synthesis Report di Vivado, troviamo che sono stati istanziati 9 registri.

Numero di bit	Numero registri	Contenuto
16	1	Segnale di indirizzo della RAM
9	1	Distanza minima del centroide dal punto
8	7	Maschera di ingresso; maschera di uscita; X e Y del punto da valutare; X e Y del centroide corrente; `currPoint`.

3.2. Area Occupata

Analizzando l'*Utilization Report* è possibile leggere l'area occupata dal componente sintetizzato, tenendo presente che la sua minimizzazione non è stata prefissata come obiettivo. Di seguito si riporta il numero di risorse utilizzate, insieme al loro utilizzo in percentuale rispetto alle risorse disponibili del FPGA.
Riguardo la scelta del FPGA, si è deciso di seguire la specifica è di utilizzare la *FPGA xc7a200tfbg484-1*.

Risorsa	**Utilizzo**	**Utilizzo in %**
LookUp Table	175	0.13%
Flip flop	91	0.03%

3.3. Report di timing

Analizzando il *Report Timing* è possibile ricavare il periodo di clock minimo per evitare errori nella computazione. Osservando il report si può ricavare che il *Worst Negative Slack* (WNS) è 90,445 ns e considerando che il ritardo di risposta della RAM è di 2 ns, è possibile calcolare il periodo di clock minimo applicabile al componente:

$$\left.\begin{array}{c} T_{clk} = 100\,ns \\ WNS = 90{,}445\text{ ns} \\ \tau_{RAM} = 2ns \end{array}\right\} T_{MIN} = T_{clk} + \tau_{RAM} - WNS = 11{,}555\,ns$$

IV. Simulazioni

Di seguito vengono riportati alcuni risultati derivanti dal testing del componente; data la necessità di testarlo in diversi casi limite si è trovato opportuno creare dei Test Bench appositi, così da osservare il comportamento nei casi di ottimizzazione sopra descritti, ma anche in casi di utilizzo normale.

4.1. Test Bench 1

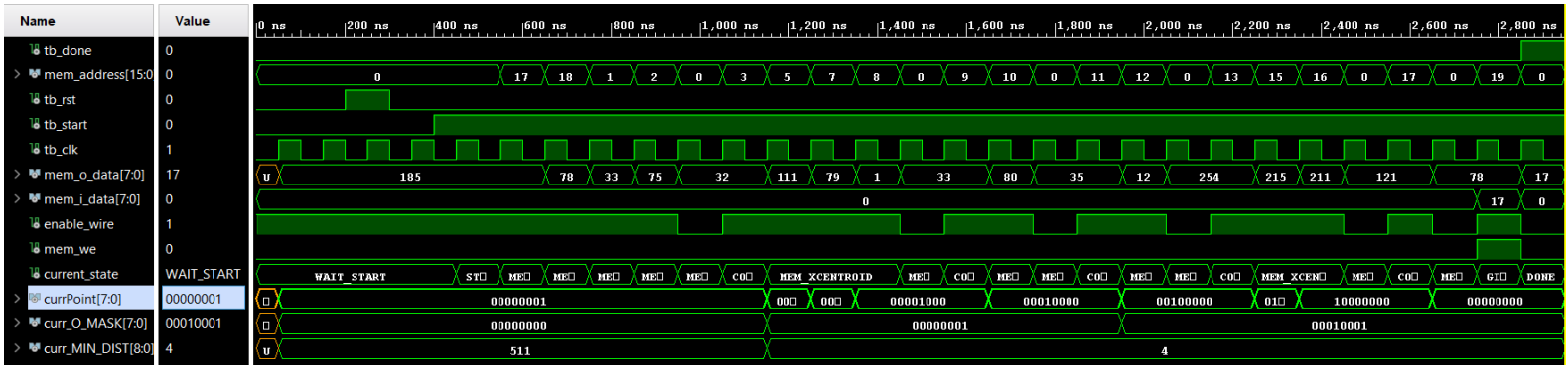
Questo Test Bench è stato fornito con la specifica e non testa nessun caso limite, ma ci permette di capire meglio il funzionamento del componente.

4.1.1. Dati della RAM

Contenuto	Valore	Indirizzo	Da considerare
Maschera di ingresso	10111001	0	-
X Centroide 1 Y Centroide 1	75 32	1 2	Sì
X Centroide 2 Y Centroide 2	111 213	3 4	No
X Centroide 3 Y Centroide 3	79 33	5 6	No
X Centroide 4 Y Centroide 4	1 33	7 8	Sì
X Centroide 5 Y Centroide 5	80 35	9 10	Sì
X Centroide 6 Y Centroide 6	12 254	11 12	Sì
X Centroide 7 Y Centroide 7	215 78	13 14	No
X Centroide 8 Y Centroide 8	21 121	15 16	Sì
X Punto Y Punto	78 33	17 18	-

4.1.2. Esecuzione

Figura 4: Test Bench 1, waveform dei segnali in Behavioural Simulation



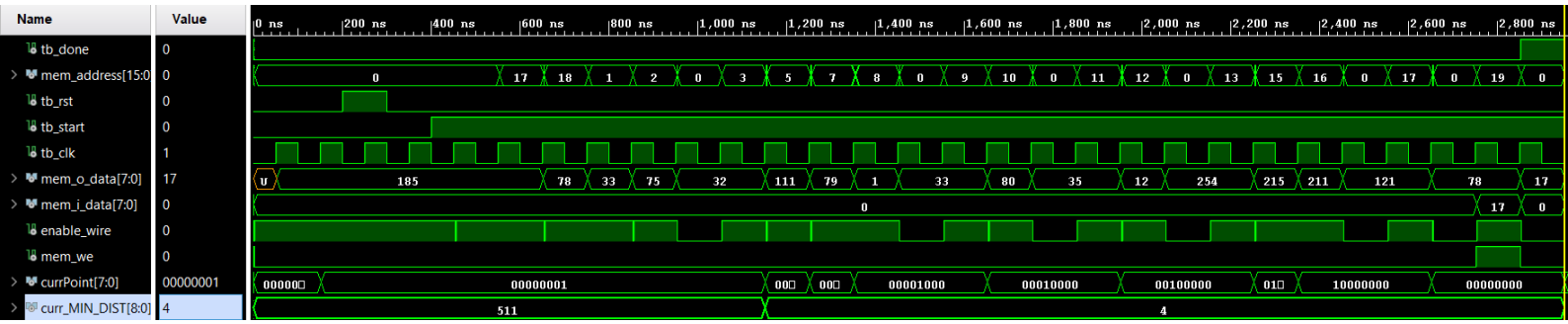
Notiamo che il componente impiega 25 cicli di clock a terminare la computazione.

1. Alla ricezione del segnale `i_start`, la macchina passa allo stato `START`. In questo stato si richiede l'invio del contenuto della RAM alla cella di memoria con indirizzo 0, ovvero la maschera di input.
2. La macchina passa allo stato `MEM_MASK`, in cui si memorizza il valore della maschera di input in un segnale apposito. Si richiede inoltre l'invio della coordinata X del punto da valutare.
3. La macchina passa allo stato `MEM_XPOINT`, in cui si memorizza il valore della coordinata X del punto in un segnale apposito. Si richiede inoltre l'invio della coordinata Y del punto da valutare.
4. La macchina passa allo stato `MEM_YPOINT`, in cui si memorizza il valore della coordinata Y del punto in un segnale apposito. Si richiede inoltre l'invio della coordinata X del primo centroide.
5. La macchina passa allo stato `MEM_XCENTROID`, in cui si memorizza il valore della coordinata X del primo centroide in un segnale apposito. Si richiede inoltre l'invio della coordinata Y del primo centroide.
6. La macchina passa allo stato `MEM_YCENTROID`, in cui si memorizza il valore della coordinata Y del primo centroide in un segnale apposito.
7. La macchina passa allo stato `COMPUTE_MIN_DISTANCE`, in cui si calcola la distanza del centroide in esame dal punto da valutare. Dato che è la prima computazione, la distanza minima attuale è la massima possibile, e quindi quella appena calcolata è minore di essa. Per questo motivo, la distanza minima è settata a quella appena calcolata. Inoltre, la maschera di uscita viene resa uguale al segnale currPoint, e il segnale currPoint viene shiftato a sinistra. Si richiede quindi l'invio della coordinata X del secondo centroide.
8. La macchina passa allo stato `MEM_XCENTROID`. Qui viene eseguita un' operazione AND tra il segnale currPoint e la maschera di ingresso. Siccome il risultato è una stringa di zeri, il secondo centroide viene saltato.
9. La macchina passa allo stato `MEM_XCENTROID`. Qui viene eseguita un' operazione AND tra il segnale currPoint e la maschera di ingresso. Siccome il risultato è una stringa di zeri, il terzo centroide viene saltato.
10. La macchina passa allo stato `MEM_XCENTROID`, in cui si memorizza il valore della coordinata X del quarto centroide in un segnale apposito. Si richiede inoltre l'invio della coordinata Y del quarto centroide.
11. La macchina passa allo stato `MEM_YCENTROID`, in cui si memorizza il valore della coordinata Y del primo centroide in un segnale apposito.
12. La macchina passa allo stato `COMPUTE_MIN_DISTANCE`, in cui si calcola la distanza del centroide in esame dal punto da valutare. Siccome questa distanza è minore della minima distanza attuale, il ciclo continua. Il segnale currPoint viene shiftato a sinistra, e si richiede l'invio della coordinata X del quinto centroide.

13. La macchina passa allo stato `MEM_XCENTROID`, in cui si memorizza il valore della coordinata X del quinto centroide in un segnale apposito. Si richiede inoltre l'invio della coordinata Y del quinto centroide.
14. La macchina passa allo stato `MEM_YCENTROID`, in cui si memorizza il valore della coordinata Y del quinto centroide in un segnale apposito.
15. La macchina passa allo stato `COMPUTE_MIN_DISTANCE`, in cui si calcola la distanza del centroide in esame dal punto da valutare. La distanza calcolata è uguale alla distanza minima attuale, quindi viene eseguito un OR tra la maschera d'uscita corrente e il segnale `currPoint`. Il segnale `currPoint` viene quindi shiftato a sinistra, e si richiede l'invio della coordinata X del sesto centroide.
16. La macchina passa allo stato `MEM_XCENTROID`, in cui si memorizza il valore della coordinata X del quarto centroide in un segnale apposito. Si richiede inoltre l'invio della coordinata Y del sesto centroide.
17. La macchina passa allo stato `MEM_YCENTROID`, in cui si memorizza il valore della coordinata Y del sesto centroide in un segnale apposito.
18. La macchina passa allo stato `COMPUTE_MIN_DISTANCE`, in cui si calcola la distanza del centroide in esame dal punto da valutare. Siccome questa distanza è minore della minima distanza attuale, il ciclo continua. Il segnale `currPoint` viene shiftato a sinistra, e si richiede l'invio della coordinata X del settimo centroide.
19. La macchina passa allo stato `MEM_XCENTROID`. Qui viene eseguita un' operazione AND tra il segnale currPoint e la maschera di ingresso. Siccome il risultato è una stringa di zeri, il settimo centroide viene saltato.
20. La macchina passa allo stato `MEM_XCENTROID`, in cui si memorizza il valore della coordinata X dell' ottavo centroide in un segnale apposito. Si richiede inoltre l'invio della coordinata Y dell'ottavo centroide.
21. La macchina passa allo `stato MEM_YCENTROID`, in cui si memorizza il valore della coordinata Y dell' ottavo centroide in un segnale apposito.
22. La macchina passa allo stato `COMPUTE_MIN_DISTANCE`, in cui si calcola la distanza del centroide in esame dal punto da valutare. Siccome questa distanza è minore della minima distanza attuale, il ciclo continua. Il segnale currPoint viene shiftato a sinistra, diventando una stringa di zeri.
23. La macchina passa allo stato `MEM_XCENTROID`. Siccome il segnale `currPoint` è una stringa di zeri, vuol dire che il ciclo è terminato. Resta solo da scrivere l'output nell'indirizzo apposito della RAM.
24. La macchina passa allo stato `GIVE_OUTPUT`. Qui viene segnalato alla RAM che una scrittura è in corso, e viene scritto nell'indirizzo della maschera d'uscita (19) il valore del segnale della maschera di uscita.
25. La macchina passa allo stato `DONE`, che tiene alzato il segnale `o_done` per un intero ciclo di clock. Successivamente, la macchina verrà resettata e sarà pronta per un'altra eventuale computazione.

4.1.3. Funzionamento in Post Synthesis Timing

Figura 5: Test Bench 1, waveform dei segnali in *Post Synthesis Timing Simulation*



Possiamo notare che il comportamento in *Post Synthesis Timing Simulation* non cambia di molto. Appaiono alcune alee: alee statiche di tipo 1 sul segnale `enable_wire`, e alee dinamiche sul segnale `mem_address`. Queste però non sono importanti perché questi segnali sono usati in modo sincrono: il loro valore viene usato solamente nel `rising_edge` del segnale di clock.

4.2. Test Bench 2

Questo Test Bench è stato progettato per testare un caso limite, in particolare il caso in cui la maschera d'ingresso sia banale, dove con maschera banale si intende qualsiasi stringa di bit che abbia un solo 1 oppure sia interamente formata da zeri. Qui possiamo vedere una delle ottimizzazioni in azione.

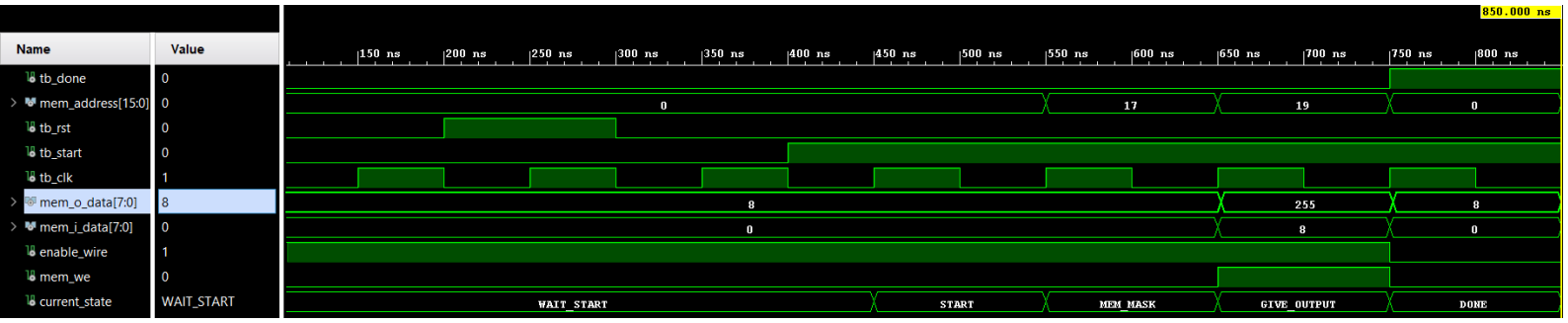
4.2.1. Dati della RAM

Contenuto	Valore	Indirizzo	Da considerare
Maschera di ingresso	00001000	0	-
X Centroide 1 Y Centroide 1	134 100	1 2	No
X Centroide 2 Y Centroide 2	123 101	3 4	No
X Centroide 3 Y Centroide 3	90 0	5 6	No
X Centroide 4 Y Centroide 4	0 0	7 8	Sì
X Centroide 5 Y Centroide 5	255 255	9 10	No
X Centroide 6 Y Centroide 6	0 0	11 12	No
X Centroide 7 Y Centroide 7	254 244	13 14	No
X Centroide 8 Y Centroide 8	140 13	15 16	No
X Punto Y Punto	255 255	17 18	-

Note sui dati della RAM: possiamo notare che l'unico centroide da considerare è il centroide numero 4, che oltretutto presenta la distanza massima dal punto in esame, essendo i due posizionati negli angoli opposti della matrice.

4.2.2. Esecuzione

Figura 6: Test Bench 2, waveform dei segnali in *Behavioural Simulation*



In questo caso di testing, possiamo osservare come il componente impieghi solamente 4 cicli di clock dalla ricezione del segnale `START` alla segnalazione di fine elaborazione tramite il segnale `DONE`. Andiamo a vedere cosa succede in questa elaborazione:

1. Alla ricezione del segnale `i_start`, la macchina passa allo stato `START`. In questo stato si richiede l'invio del contenuto della RAM alla cella di memoria con indirizzo 0, ovvero la maschera di input.
2. La macchina passa allo stato `MEM_MASK`, in cui si memorizza il valore della maschera di input in un segnale apposito. Si richiede inoltre l'invio della coordinata X del punto da valutare. A questo punto, il componente controlla che il valore della maschera appena ricevuto non sia appunto una maschera banale. Siccome è questo il caso (il valore della maschera di input è 8, che corrisponde al numero binario *00001000*), la macchina imposta il prossimo stato a `GIVE_OUTPUT` e la maschera di uscita viene resa uguale alla maschera di entrata.
3. La macchina passa allo stato `GIVE_OUTPUT` in cui la macchina invia, all'apposito indirizzo (19) il valore della maschera di uscita (*8,* che in binario equivale a *00001000)*.
4. La macchina passa allo stato `DONE`, che alza il segnale `o_done` per segnalare alla RAM che il componente ha finito di elaborare.

4.2.3. Note sul Test Bench 2

Non è stato necessario calcolare nessuna distanza, né elaborare la distanza minima. Il componente si limita a passare il valore della maschera di ingresso al segnale che contiene la maschera d'uscita; per questo la posizione del punto e di ogni centroide è irrilevante.

4.3. Altri Test Bench

Come menzionato prima, sono stati creati degli ulteriori Test Bench; questi hanno avuto lo scopo di testare il componente in tutti i casi limiti possibili.
Alcuni casi limite testati sono stati:

- Distanza minima 0, ovvero centroide coincidente con il punto da valutare
- Maschera di ingresso contenente una stringa di zeri. In questo caso l'output è stato una stringa di zeri
- Tutti i centroidi coincidenti
- La distanza minima dei centroidi dal punto da valutare è la distanza massima possibile, rendendo così necessario l'utilizzo di un segnale a 9 bit.

Inoltre, è stata testata anche la robustezza del componente tramite alcuni test "strutturali". In particolare, sono stati testati i seguenti casi:

- Segnale di reset alzato durante l'elaborazione
- Segnale di reset e segnale di start che vengono portati alti contemporaneamente all'inizio della elaborazione
- Elaborazioni consecutive di vari test con varie istanze di RAM

Il componente ha superato tutti i Test Bench da noi creati, sia in *Behavioural Simulation*, che in *Post-Synthesis Functional Simulation* e *Post-Synthesis Timing Simulation*.

V. Conclusioni

In conclusione siamo riusciti a realizzare un componente capace di rispettare tutti gli obiettivi prefissati sia da specifica sia dalle scelte implementative introdotte durante la progettazione. Infatti, è stato realizzato un componente:

- Funzionante sia in pre-sintesi sia in post-sintesi.
- Ottimizzato in modo che l'elaborazione delle maschere banali sia immediata.
- Ottimizzato in modo che, se un centroide non deve essere considerato, le sue coordinate non vengano salvate nei registri e la sua distanza non venga calcolata.
- Con periodo minimo di clock impostabile a 11,555 ns, cioè oltre l'88% più veloce del periodo richiesto dalla specifica.