Progetto finale di Reti Logiche

Motta Dennis - Matricola n. 865833

Anno Accademico 2018/2019

Indice

1 Introduzione

Il mio obiettivo per questo progetto, oltre a creare un design funzionante in pre e post sintesi che rispetti le specifiche, è stato quello di creare un componente che arrivasse al risultato il più velocemente possibile, sfruttando quindi ogni ciclo di clock, ma senza dimenticare di scrivere codice semplice, senza ripetizioni e di facile manutenzione.
Queste scelte progettuali portano anche alcuni svantaggi: con un focus sulla velocità totale le caratteristiche di massima frequenza del clock e di area occupata passano in secondo piano. Non si è scelto un focus sull'area occupata in quanto la FPGA scelta ha centinaia di migliaia di Flip-Flop e LUT. Anche un focus su una maggiore frequenza di clock è stato messo in secondo piano in quanto il progetto ha già un periodo di clock di 100 ns dato da specifica.

1.1 Funzionamento in sintesi

In una breve sintesi introduttiva, si può rappresentare il funzionamento di base del componente attraverso un numero finito di step (e ciò verrà rappresentato architetturalmente attraverso una macchina a stati finiti):

1. Reset e attesa del segnale di start.
2. Inizializzazione degli output del componente mandando la richiesta di lettura della maschera d'ingresso alla RAM.
3. Lettura e salvataggio in un registro della maschera di ingresso.
4. Lettura e salvataggio della X del punto da valutare.
5. Lettura e salvataggio della Y del punto da valutare.
6. Lettura e salvataggio della X del 1° centroide.
7. Lettura della Y del 1° centroide. Se il centroide va considerato per la maschera d'ingresso si calcola la distanza tra il punto da valutare e il centroide. Se questa distanza è minore della distanza minima la si salva e si sovrascrive la maschera di uscita temporanea, se essa invece è uguale alla distanza minima si pone il primo bit a '1' nella maschera di uscita temporanea.
8. In modo equivalente 2°, 3°, 4°, 5°, 6° e 7° centroide...
20. Lettura e salvataggio della X del 8° centroide.
21. Lettura della Y del 8° centroide. Se il centroide va considerato per la maschera d'ingresso si calcola la distanza tra il punto da valutare e il centroide. Se questa distanza è minore della distanza minima la si salva e si sovrascrive la maschera di uscita temporanea, se essa invece è uguale alla distanza minima si pone l'ottavo bit a '1' nella maschera di uscita temporanea.
22. Scrittura sulla RAM della maschera di uscita; Segnalazione di fine elaborazione usando il segnale *o_done*; Attesa del segnale di fine (*i_start* posto a '0') che ci permetterà di ritornare allo step 1 per una possibile successiva elaborazione.

1.2 Obiettivo velocità

Per raggiungere l'obiettivo della velocità si è dovuto tenere in considerazione le limitazione della RAM: a ogni ciclo di clock solo una lettura o una scrittura. La RAM quindi ha imposto un limite massimo di velocità raggiungibile. Per raggiungere questo limite si è creato un codice in cui l'elaborazione e la presentazione dei segnali di output avvenisse nello stesso ciclo di clock in cui viene fatta la lettura del dato.
Una volta raggiunto il limite imposto dalla RAM si sono applicate alcune ottimizzazioni per raggiungere la massima velocità nel calcolo della maschera di uscita:

1. Si evita la lettura sia del valore X che del valore Y di centroidi che sono disattivati nella maschera di ingresso
2. Viene presentato il risultato immediatamente quando i bit attivati (bit '1') nella maschera di ingresso sono in numero uguale a 0 o 1 (esempio: "00100000"). In questi casi la maschera di output è necessariamente identica alla maschera di ingresso.
3. Si passa al centroide successivo (tenendo in considerazione l'ottimizzazione n.1) quando alla lettura del valore X del centroide si trova che la distanza sulle ascisse del centroide col punto da valutare

è maggiore della distanza minima fino a quel momento trovata. (esempio: distanza minima = 4; punto da valutare X = 78; centroide X = 12; in questo caso la distanza sulle ascisse è pari a $78 - 12 = 66$ che è già maggiore della distanza minima, si passa quindi al centroide successivo)

1.3 Obiettivo codice semplice

Per scrivere codice semplice e di facile manutenzione si è deciso di usare il meno possibile funzionalità algoritmiche (process), ciò per cercare di non utilizzare il linguaggio VHDL come se fosse un linguaggio di programmazione software. Il codice è quindi organizzato in diverse funzionalità, tutte inserite in un singolo modulo (entity), questa decisione puramente personale è stata presa per non complicare un codice in sostanza semplice.
Infine si sono create due costanti e un generic per rendere il codice facilmente espandibile a possibili modifiche:

- *MEM_BITS* : costante che indica il numero di bit per un indirizzo di memoria. Per soddisfare la specifica di default è assegnato il valore 16.
- *CELL_BITS* : costante che indica il numero di bit in una cella di memoria che si assume equivalente al numero di centroidi da analizzare. Per soddisfare la specifica di default è assegnato il valore 8.
- *START_ADDRESS* : generic che facilita la modifica dell'indirizzo iniziale di memoria (dove è quindi salvata la maschera d'ingresso). Per soddisfare la specifica di default è assegnato il valore 0.

1.4 Note aggiuntive sulla specifica

Si è assunto che il componente deve essere in grado di eseguire elaborazioni una successiva all'altra anche in assenza di un segnale di reset.

2 Architettura

2.1 Macchina a Stati Finiti

Il funzionamento alla base del componente è stato implementato attraverso una FSM che usa come segnale di ingresso *i_start*. In realtà però il passaggio agli stati successivi è dato da condizioni più complesse che permettono l'ottimizzazione, ciò verrà approfondito nella sezione 2.2 a pagina 5. La FSM non ottimizzata è rappresentata in figura 1, mentre ciò che ogni stato rappresenta è spiegato più in dettaglio in tabella 1.

Figura 1: Macchina a Stati Finiti implementata

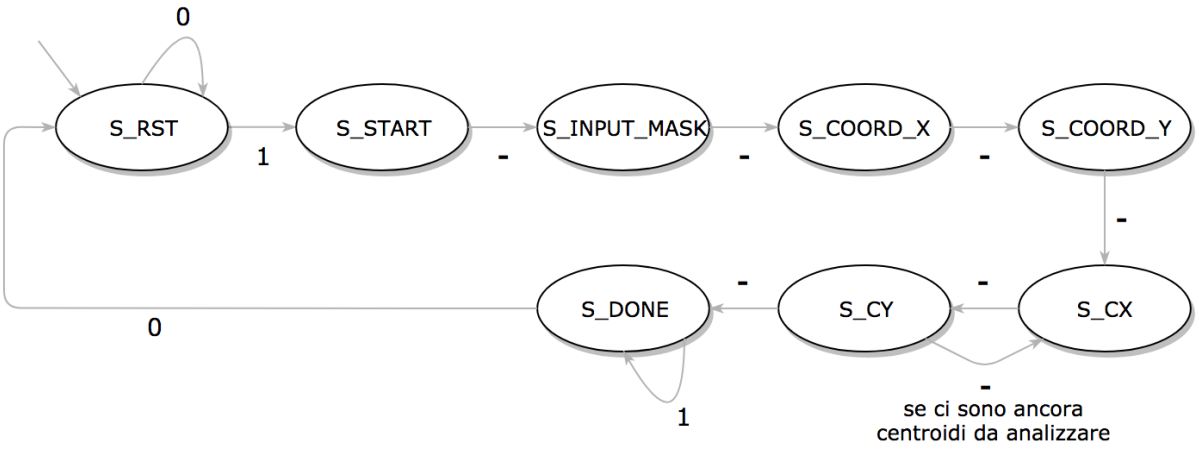


Tabella 1: Stati della FSM

S_RST	Stato di partenza della FSM e stato in cui si andrà in presenza di un segnale *i_rst*. Alla ricezione di un segnale *i_start* si passa allo stato *S_START*.
S_START	Stato iniziale. In questo stato viene fornito alla RAM l'indirizzo della maschera di ingresso specificato dal generic *START_ADDRESS*.
S_INPUT_MASK	Stato in cui il componente legge e salva in un registro la maschera di ingresso che gli è arrivata da memoria.
S_COORD_X	Stato in cui il componente legge e salva in un registro la X del punto da valutare che gli è arrivata da memoria.
S_COORD_Y	Stato in cui il componente legge e salva in un registro la Y del punto da valutare che gli è arrivata da memoria.
S_CX	Lettura e salvataggio delle X dei centroidi.
S_CY	Lettura delle Y dei centroidi. Si calcola la distanza tra il punto da valutare e il centroide, se essa è minore della distanza minima si salva questa nuova distanza e si sovrascrive la maschera di uscita temporanea, se essa invece è uguale alla distanza minima si pone il bit corrispondente a '1' nella maschera di uscita temporanea.
S_DONE	Stato in cui si segnala che il risultato è stato scritto in RAM: *o_done* è portato ad '1'. Alla ricezione di *i_start* uguale a '0' si riporta la macchina in *S_RST* per una possibile successiva elaborazione.

2.1.1 Ottimizzazioni effettuate

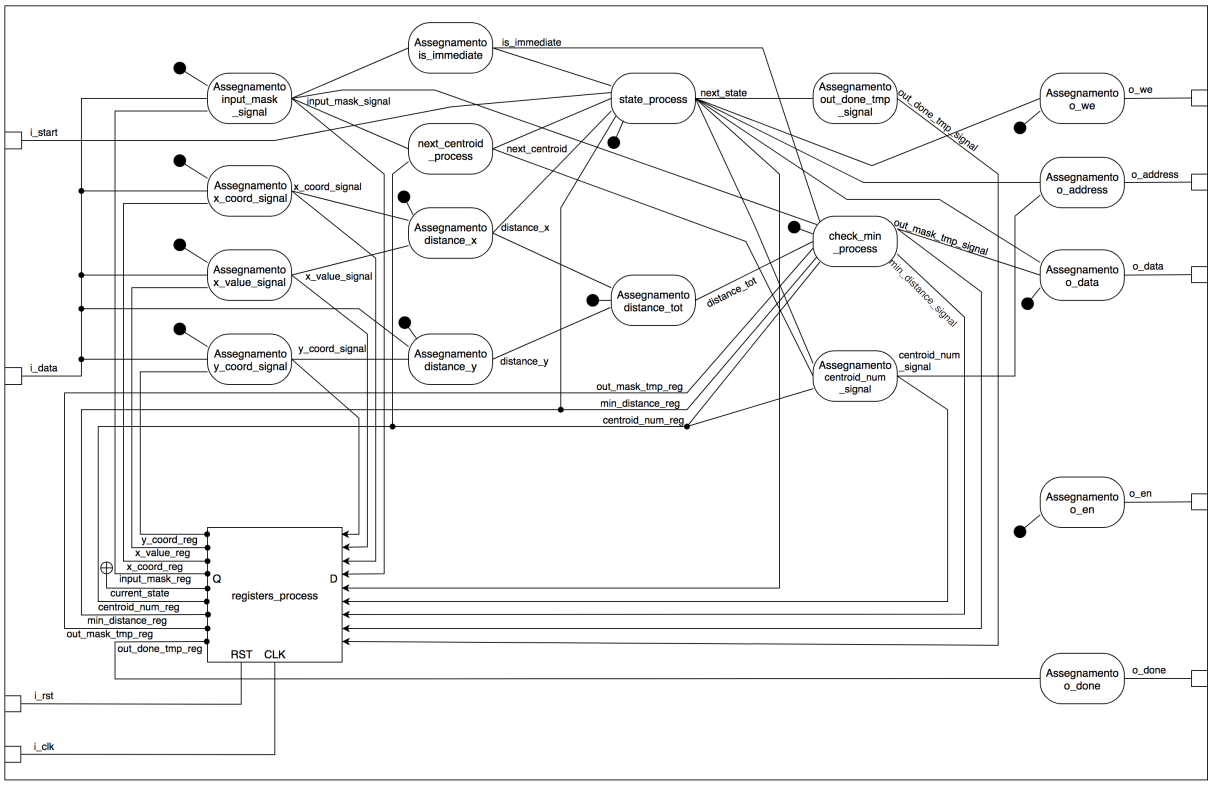
Per arrivare al risultato nel modo più veloce possibile si sono applicate alcune ottimizzazioni già introdotte nella sezione 1.2 a pagina 2. Vediamo più in dettaglio gli effetti sulla FSM:

1. Si evita la lettura di centroidi che sono disattivati nella maschera di ingresso, ciò significa che lo stato *S_CX* verrà percorso un numero di volte pari al numero di bit attivi nella maschera di ingresso.
2. Alla lettura della maschera di ingresso che avviene nello stato *S_INPUT_MASK*, se si scopre che essa ha 0 o 1 bit attivati, si passa direttamente allo stato *S_DONE* scrivendo il risultato nella RAM.

3. Alla lettura della X di un qualsiasi centroide che avviene nello stato *S_CX*, se si trova che la distanza sulle ascisse tra il centroide e il punto da valutare è maggiore della distanza minima fino a quel momento trovata, si passa al centroide successivo rimanendo quindi nello stato *S_CX* (il centroide successivo è calcolato tenendo presente l'ottimizzazione n.1).

2.2 Schema funzionale

Figura 2: Schema funzionale del componente



Nota: visto la presenza di molte funzionalità che usano il segnale current_state si è usato i simboli: ⊕ ● per rappresentare i collegamenti di questo segnale.

2.2.1 Registri

Come si può vedere dallo schema funzionale, *registers_process* permette di creare 9 registri con lo scopo di salvare il valore di alcuni importanti segnali:

- *current_state* : stato corrente della FSM.
- *centroid_num_reg* : centroide corrente.
- *input_mask_reg* : maschera d'ingresso.
- *x_coord_reg* : X del punto da valutare.
- *y_coord_reg* : Y del punto da valutare.
- *x_value_reg* : X del centroide corrente.
- *min_distance_reg* : distanza minima trovata.
- *out_mask_tmp_reg* : maschera d'uscita temporanea.
- *out_done_tmp_reg* : valore da dare a *o_done* al prossimo ciclo di clock, si è usato un registro per evitare alee statiche sul segnale che avrebbero compromesso il funzionamento.

N.B.: Si è usata la convenzione che i segnali **_signal* rappresentano l'ingresso dei registri mentre i segnali **_reg* le uscite (eccetto per il registro dello stato della FSM che usa *current_state* e *next_state*).

2.2.2 Funzionalità in dettaglio

Vediamo ora più in dettaglio lo scopo di ogni funzionalità:

- Assegnamento *input_mask_signal*, assegnamento *x_coord_signal*, assegnamento *x_value_signal*, assegnamento *y_coord_signal* : assegna all'ingresso del registro il giusto segnale, che può essere l'ingresso *i_data* oppure il valore di uscita del corrispondente registro.
- Assegnamento *is_immediate* : segnala se la maschera di ingresso permette di avere una risposta immediata (cioè se ha 1 oppure 0 bit attivati). Per arrivare al risultato si è utilizzato un famoso "trucchetto": avere un bit attivato significa essere una potenza di 2. Ed N è una potenza di 2 se $(N\&N-1)=0$. Dove & rappresenta l'and bit a bit. Questa funzionalità permette la realizzazione dell'ottimizzazione n.2 della sezione 2.1.1 a pagina 4.
- *next_centroid_process* : trova il valore del possibile prossimo centroide utilizzando il valore del centroide corrente e la maschera d'ingresso, salta quindi i centroidi non da considerare. Questa funzionalità permette la realizzazione dell'ottimizzazione n.1 della sezione 2.1.1 a pagina 4.
- Assegnamento *distance_x* : calcola la distanza sulle ascisse. Usa anche il valore *current_state* per calcolare la distanza solo quando necessario.
- Assegnamento *distance_y* : calcola la distanza sulle ordinate. Usa anche il valore *current_state* per calcolare la distanza solo quando necessario.
- Assegnamento *distance_tot* : calcola la distanza totale, che non è altro che la somma di *distance_x* e *distance_y*. Usa anche il valore *current_state* per calcolare la distanza solo quando necessario.
- *state_process* : gestisce lo stato della FSM. Gli ingressi *current_state* e *i_start* servono alla FSM di base mentre gli altri segnali permettono le ottimizzazioni specificate nella sezione 2.1.1 a pagina 4. Il segnale *next_centroid* permette l'ottimizzazione n.1. Il segnale *is_immediate* permette l'ottimizzazione n.2. I segnali *distance_x* e *min_distance_reg* permettono l'ottimizzazione n.3.
- Assegnamento *out_done_tmp_signal* : assegna all'ingresso del registro il valore da dare a *o_done* al prossimo ciclo di clock. Il segnale è portato a '1' quando il prossimo stato specificato in *next_state* è *S_DONE*.
- *check_min_process* : controlla se il centroide corrente è a distanza minima usando *distance_tot*. Se questa distanza è minore della distanza minima la si assegna a *min_distance_signal* e si sovrascrive *out_mask_tmp_signal*, se essa invece è uguale alla distanza minima si attiva il bit corrispondente al centroide in *out_mask_tmp_signal*.
- Assegnamento *centroid_num_signal* : trova il valore effettivo di quello che sarà il prossimo centroide. Il centroide successivo dipende da *next_state*, cioè lo stato in cui andrà la FSM: se il prossimo stato è *S_CX* si può passare al centroide successivo, specificato dal segnale *next_centroid*. Se il prossimo stato è *S_CY* bisogna tenere il valore del centroide corrente, specificato dal segnale *centroid_num_reg*.

Infine si assegna il valore alle 5 uscite del componente:

- Assegnamento *o_we* : il segnale di write-enable della RAM viene abilitato quando lo stato successivo della FSM è *S_DONE*.
- Assegnamento *o_address* : assegna l'indirizzo di lettura della RAM in base allo stato successivo della FSM e al centroide successivo, il dato sarà quindi poi letto al ciclo di clock successivo.
- Assegnamento *o_data* : assegnamento dell'output di scrittura della RAM. Alla RAM viene mandata in scrittura la maschera di uscita temporanea (*out_mask_tmp_signal*) solo nello stato per cui ciò è necessario, cioè lo stato precedente alla fine, quindi quando il *next_state* è *S_DONE*.
- Assegnamento *o_en* : il segnale di enable della RAM viene abilitato per tutti gli stati in cui si ha una lettura o scrittura della RAM.
- Assegnamento *o_done* : il segnale di fine elaborazione è abilitato quando si è nello stato *S_DONE*. Si assegna però a questo segnale *out_done_tmp_reg*, valore di uscita del registro. Ciò viene fatto per evitare glitch su questa uscita.

3 Sintesi

3.1 Registri sintetizzati

Analizzando il "Vivado Synthesis Report" troviamo che sono stati sintetizzati i registri come descritto nel codice. Viene quindi segnalata la creazione di 9 registri (per un totale di 58 Flip Flop a singolo bit utilizzati):

Num. bit	**Num. registri**	**Contenuto**
9	1	Distanza minima.
8	5	Maschera di uscita temporanea; maschera d'ingresso; X del punto da valutare; Y del punto da valutare; X del centroide corrente.
5	1	Indice del centroide corrente.
3	1	Stato della FSM.
1	1	Registro usato per l'uscita *o_done* al fine di evitare glitch sul segnale.

3.2 Area occupata

Eseguendo un "Report Utilization" vediamo ora l'area occupata dal design sintetizzato. Come già spiegato, non si è cercato di ottimizzare l'area occupata del componente. Questo aspetto è stato lasciato a responsabilità del tool di sintesi.

Tabella 2: Report di utlizzo

Risorsa	**Utilizzo**	**Disponibilità**	**Utilizzo in %**
Look Up Table	154	134600	0.11%
Flip Flop	58	269200	0.02%

Si può notare che i valori di utilizzo hanno svariati ordini di grandezza in meno rispetto alla disponibilità della FPGA. Anche per questo motivo non si è ritenuto proficuo ottimizzare l'area utilizzata.

3.3 Report di timing

Analizzando il report di timing si può vedere quanto è veloce in un singolo ciclo di clock il design sintetizzato. Si è ottenuto con il clock della specifica di 100ns un Worst Negative Slack pari a 90,541ns. Da questo valore, sapendo anche il ritardo di riposta della RAM (T_{RAM}), possiamo calcolare il periodo minimo applicabile al design creato:

$$T_{min} = T_{curr} - WNS + T_{RAM}$$
$$T_{min} = 100ns - 90.541ns + 2ns = 11.459ns$$

Il design creato ha quindi una massima frequenza di clock pari a: $f_{max} = 1/T_{min} \approx 87.3Mhz$.

3.4 Note sulla sintesi

Si è utilizzato il tool "Vivado 2018.3 WebPACK Edition" impostato con i parametri di default.

4 Simulazioni

Una volta creato il design, esso va anche testato. Per fare ciò si è creato dei test bench appositi al fine di testare il componente sia nei normali casi di utilizzo e sia nei casi limite. L'obiettivo è stato quello di testare tutte le funzionalità e le istruzione del codice. Qui sono riportati i test bench più significativi.

4.1 Test Bench 0 (fornito con la specifica)

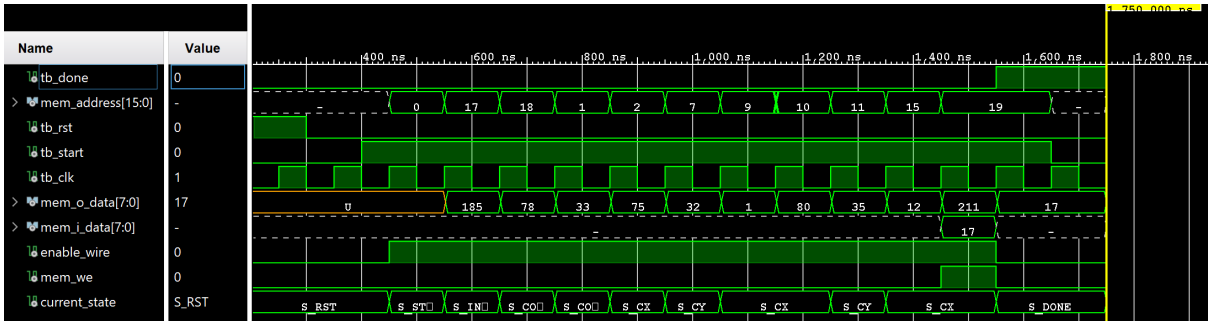
In questo test bench viene provato un caso normale, senza casi limite, però ci permette di vedere alcune delle ottimizzazioni all'opera. Vediamo perciò in dettaglio il funzionamento del componente con questo test bench.

4.1.1 Dati del test bench

Contenuto	Valore	Indirizzo	Da considerare
Maschera d'ingresso	1011'1001 (185)	0	-
X centroide 1	75	1	V
Y centroide 1	32	2	V
X centroide 2	111	3	X
Y centroide 2	213	4	X
X centroide 3	79	5	X
Y centroide 3	33	6	X
X centroide 4	1	7	V
Y centroide 4	33	8	V
X centroide 5	80	9	V
Y centroide 5	35	10	V
X centroide 6	12	11	V
Y centroide 6	254	12	V
X centroide 7	215	13	X
Y centroide 7	78	14	X
X centroide 8	211	15	V
Y centroide 8	121	16	V
X del punto da valutare	78	17	-
Y del punto da valutare	33	18	-

4.1.2 Elaborazione

Figura 3: Test bench 0, waveform dei segnali in Behavioral Simulation



Come si può vedere in figura 3, il componente impiega soltanto 11 cicli di clock dalla ricezione del segnale di start fino alla segnalazione di fine con *o_done*.
Vediamo cosa succede in questi 11 cicli di clock con cui il componente arriva al risultato finale:

I Alla ricezione del segnale di start la FSM del componente passa allo stato *S_START* qui manda come indirizzo di lettura l'indirizzo 0.

II La FSM passa allo stato *S_INPUT_MASK*. Il componente riceve il contenuto della cella di memoria 0: 185. Questa è la maschera d'ingresso che verrà salvata nel relativo registro. Infine manda come indirizzo di lettura l'indirizzo 17.

III La FSM passa allo stato *S_COORD_X*. Il componente riceve il contenuto della cella di memoria 17: 78. Questa è la X del punto da valutare che verrà salvata nel relativo registro. Infine manda come indirizzo di lettura l'indirizzo 18.

IV La FSM passa allo stato *S_COORD_Y*. Il componente riceve il contenuto della cella di memoria 18: 33. Questa è la Y del punto da valutare che verrà salvata nel relativo registro. Infine manda come indirizzo di lettura l'indirizzo 1.

V La FSM passa allo stato *S_CX*. Il componente riceve il contenuto della cella di memoria 1: 75. Questa è la X del primo centroide che verrà salvata nel relativo registro. Infine manda come indirizzo di lettura l'indirizzo 2.

VI La FSM passa allo stato *S_CY*. Il componente riceve il contenuto della cella di memoria 2: 32. Questa è la Y del primo centroide con cui calcola la distanza minima col punto da valutare. Infine manda come indirizzo di lettura l'indirizzo 7, **vengono quindi saltati i centroidi 2 e 3 siccome non sono da considerare per la maschera**.

VII La FSM passa allo stato *S_CX*. Il componente riceve il contenuto della cella di memoria 7: 1. **La distanza sulle ascisse tra questo centroide e il punto da valutare è maggiore della distanza minima. Perciò si passa al centroide successivo**. Manda quindi come indirizzo di lettura l'indirizzo 9.

VIII La FSM rimane nello stato *S_CX*. Il componente riceve il contenuto della cella di memoria 9: 80. Questa è la X del quinto centroide che verrà salvata nel relativo registro. Infine manda come indirizzo di lettura l'indirizzo 10.

IX La FSM passa allo stato *S_CY*. Il componente riceve il contenuto della cella di memoria 10: 35. Questa è la Y del quinto centroide con cui calcola la distanza dal punto da valutare. Si trova che questa distanza è equivalente a quella minima, viene perciò attivato il quinto bit nella maschera di uscita temporanea. Infine manda come indirizzo di lettura l'indirizzo 11.

X La FSM passa allo stato *S_CX*. Il componente riceve il contenuto della cella di memoria 11: 12. **La distanza sulle ascisse tra questo centroide e il punto da valutare è maggiore della distanza minima. Perciò si passa al centroide successivo**. Manda quindi come indirizzo di lettura l'indirizzo 15, **viene saltato il centroide 6 disattivato nella maschera**.

XI La FSM rimane nello stato *S_CX*. Il componente riceve il contenuto della cella di memoria 15: 211. **Anche qui la distanza sulle ascisse è maggiore della distanza minima**. Abbiamo perciò raggiunto il risultato finale: 0001'0001 (17). Esso viene mandato in scrittura alla RAM nella cella con indirizzo 19.

4.2 Test Bench 1 (un solo bit attivato nella maschera d'ingresso)

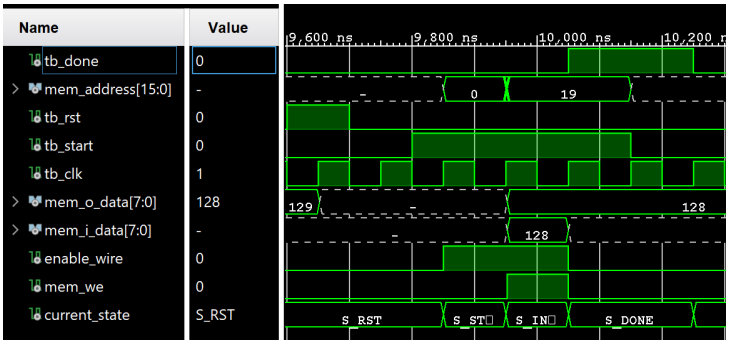
Con questo test bench si è voluto testare l'ottimizzazione n.2 della sezione 1.2 a pagina 2. Si è testato in realtà tutti i 9 casi possibili (0 bit attivati, 1 bit attivato in 8 possibili posizioni) ma vediamo soltanto uno di questi casi in quanto poi gli altri sono equivalenti.

4.2.1 Dati del test bench

Contenuto	**Valore**	**Indirizzo**	**Da considerare**
Maschera d'ingresso	1000'0000 (128)	0	-
...	...	...	X
X centroide 8	25	1	V
Y centroide 8	23	2	V
X del punto da valutare	86	17	-
Y del punto da valutare	129	18	-

4.2.2 Elaborazione

Figura 4: Test bench 1, waveform dei segnali in Behavioral Simulation



Come si può vedere in figura 4, il componente impiega soltanto 2 cicli di clock dalla ricezione del segnale di start fino alla segnalazione di fine con *o_done*.
Vediamo cosa succede in questi 2 cicli di clock con cui il componente arriva al risultato finale:

I Alla ricezione del segnale di start la FSM del componente passa allo stato *S_START* qui manda come indirizzo di lettura l'indirizzo 0.

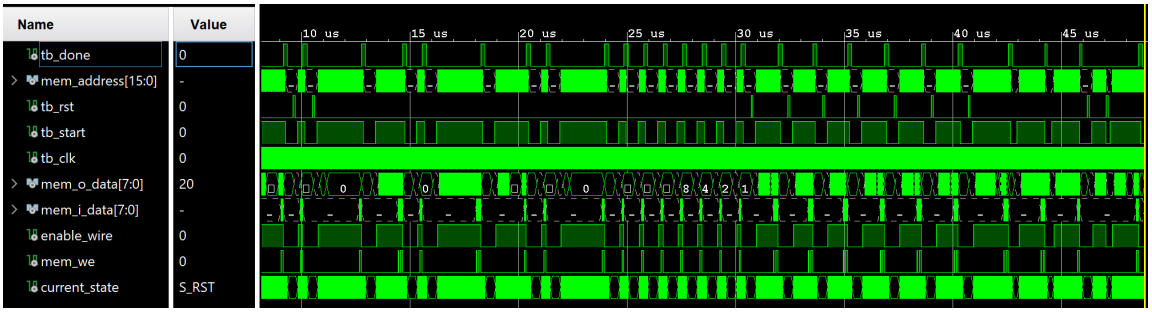
II La FSM passa allo stato *S_INPUT_MASK*. Il componente riceve il contenuto della cella di memoria 0: 128. A questo punto la funzionalità di assegnamento del segnale *is_immediate* riconosce che la maschera d'ingresso è una potenza di due. Ciò viene riconosciuto se $128\&(128-1)$ dà come risultato 0, ed effettivamente si trova che 1000'0000 & 0111'1111 = 0000'0000. Si è quindi raggiunto il risultato finale che non è altro che la maschera d'ingresso stessa: 128. Esso viene mandato in scrittura alla RAM nella cella con indirizzo 19.

4.3 Altri test bench

Sono stati creati altri test bench per testare alcuni casi limite o per controllare la robustezza del componente. Non li vediamo in dettaglio siccome alla fine seguono tutti lo stesso principio di esecuzione.

- Test bench in cui la distanza del punto da valutare coi centroidi è molto alta e tale per cui si devono usare 8 + 1 bit. Si è usato quindi dati in cui il punto da valutare ha ascissa e ordinate dal valore molto basso mentre per i centroidi si è usato sia per la X che la Y valori maggiori di 200.
- Test bench in cui alcuni centroidi sono a distanza 0 con il punto da valutare.
- Test bench in cui tutti i centroidi sono da considerare e hanno la stessa distanza col punto da valutare (maschera di uscita: 1111'1111).
- Test bench in cui si controlla il corretto funzionamento del componente a seguito di elaborazioni successive senza segnali di reset.
- Test bench in cui si controlla il corretto funzionamento del componente quando, durante una elaborazione, viene mandato un segnale di reset.
- Infine sono stati creati una decina di altri test che non vanno a verificare casi particolari ma semplicemente casi normali con dati diversi. Questi test sono stati progettati in modo che ogni bit nella maschera di uscita venga testato almeno una volta.

Figura 5: Esempio di test bench effettuato con elaborazioni successive



5 Conclusione

Tirando le somme si è creato un design con queste caratteristiche:

- Funzionante in pre e post-sintesi.
- Ottimizzato in modo che durante l'elaborazione venga sfruttata al massimo la RAM: a ogni ciclo di clock una lettura o una scrittura.
- Ottimizzato in modo che ogni lettura della RAM venga eseguita solo se strettamente necessaria (non viene letto ciò che non serve).
- Configurabile con le costanti *MEM_BITS* e *CELL_BITS* per adattarsi a differenti tipi di RAM, e con il generic *START_ADDRESS* per eseguire l'elaborazione all'indirizzo desiderato.
- Frequenza massima di clock impostabile a 87.3Mhz.
- Utilizzo di LUT pari al 0.11%.
- Utilizzo di FF pari al 0.02%