Prova Finale (Progetto di Reti Logiche)

Prof. Gianluca Palermo - Anno 2018/2019

Giorgio Piazza (Codice Persona 10529035 - Matricola 867894)
Francesco Piro (Codice Persona 10534719 - Matricola 870365)

Indice

1 Introduzione

1.1 Scopo del progetto

Si definisca uno spazio quadrato di dimensione 256 x 256 e siano date le posizioni di 8 punti detti "centroidi": lo scopo del progetto è di implementare un componente hardware, descritto in VHDL, che, presi in ingresso i centroidi, una maschera che indichi quali di essi utilizzare per il calcolo e un punto da valutare, calcoli quali siano i centroidi più vicini al punto utilizzando la *Manhattan distance*.

1.2 Specifiche generali

Degli 8 centroidi definiti, K $\leq$ 8 sono quelli da utilizzare nel calcolo della distanza. I K centroidi sono indicati con una *maschera di ingresso* a 8 bit: il bit a `1` indica che il centroide va preso in considerazione nel calcolo della distanza mentre il bit a `0` indica che il centroide non deve essere considerato. I centroidi sono indicati partendo dal bit meno significativo fino al bit più significativo. Segue un esempio di maschera di ingresso:

8	7	6	5	4	3	2	1
1	0	0	1	0	0	1	0

Figura 1: Vengono considerati solo il secondo, il quinto e l'ottavo centroide

La vicinanza del centroide al punto viene espressa tramite una *maschera di uscita* a 8 bit: il bit viene posto a `1` se il centroide è il più vicino al punto fornito, altrimenti viene posto a `0`. I centroidi sono indicati partendo dal bit meno significativo fino al bit più significativo.

Nel caso in cui il punto risulti equidistante da 2 o più centroidi e questa distanza fosse la minima, i bit della maschera di uscita relativi a tali centroidi saranno tutti impostati a `1`. Si noti che i bit a `1` non saranno mai più di K. Segue un esempio di maschera di uscita:

8	7	6	5	4	3	2	1
1	0	0	0	0	0	1	0

Figura 2: I centroidi più vicini al punto sono il secondo e l'ottavo

1.3 Interfaccia del componente

Il componente da descrivere ha un'interfaccia così definita:

```
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;
```

In particolare:

- `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 e il dato di uscita scritto in memoria;
- `o_en` è il segnale di `ENABLE` da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- `o_we` è il segnale di `WRITE ENABLE` da dover mandare alla memoria (=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.

1.4 Dati e descrizione memoria

I dati, ciascuno di dimensione 8 bit, sono memorizzati in una memoria con indirizzamento al byte:

- L'indirizzo 0 è usato per memorizzare la maschera di ingresso;
- Gli indirizzi dal 1 al 16 sono usati a coppie per memorizzare le coordinate X e Y dei centroidi;
- 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 la maschera di uscita.

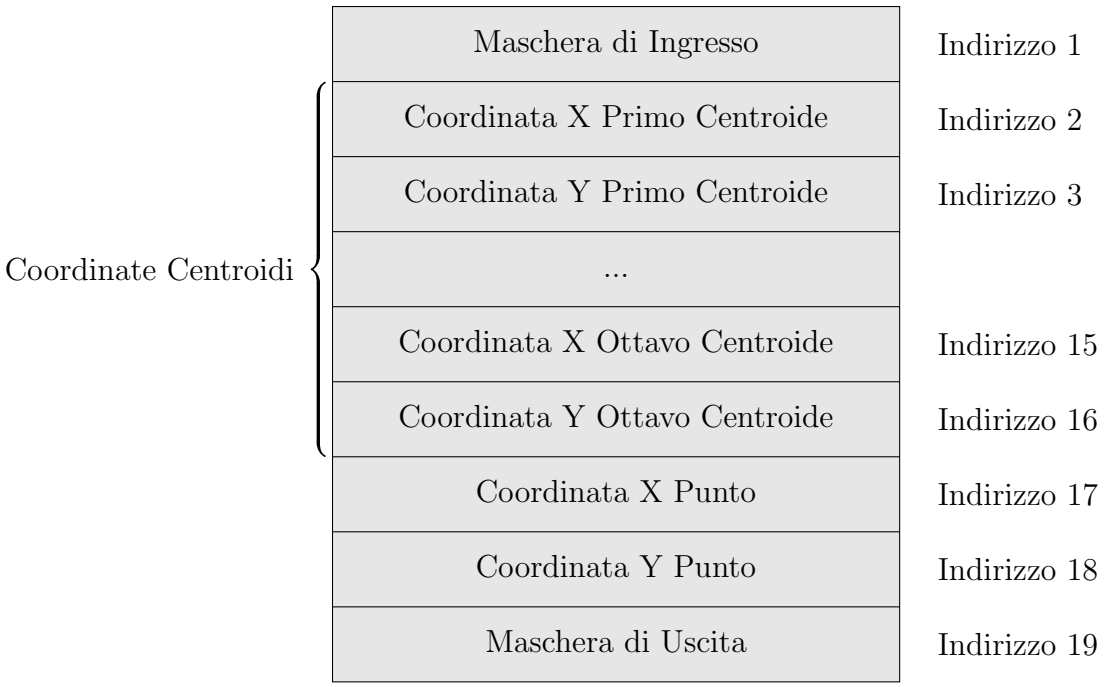

Figura 3: Rappresentazione indirizzi significativi della memoria

2 Design

Quando il segnale `i_start` in ingresso viene portato a 1, il componente sviluppato inizia l'elaborazione spostandosi dallo stato `IDLE` al primo stato della computazione. Una volta terminata la computazione, dopo avere scritto il risultato in memoria, il componente alza il segnale `o_done`. La test bench risponde abbassando `i_start` e, successivamente, il componente riporta a `0` `o_done`; il componente ritorna nello stato `IDLE`, in attesa che il segnale `i_start` torni alto.

Il componente dispone inoltre di un segnale `i_rst` che, insieme agli altri appena elencati, ci hanno portato a definire una FSM(D), macchina a stati finiti con *data path*, che combina una normale FSM con tipici circuiti sequenziali.

Nelle seguenti sezioni troviamo infatti sia la descrizione della FSM sia la descrizione della parte sequenziale della macchina che permette la gestione dei registri utilizzati.

2.1 Stati della macchina

La macchina costruita è composta da 10 stati. Qui di seguito è fornita una breve descrizione per ciascuno di questi.

2.1.1 `IDLE` state

Stato **iniziale** in cui si attende il segnale di `i_start`. In caso venga alzato il segnale `i_rst` si torna in questo stato.

2.1.2 `FETCH_CONST` state

Stato in cui viene richiesta la coordinata X del punto, successivamente la coordinata Y ed infine la maschera in ingresso. Il tutto viene fatto tornando 3 volte in questo stato in quanto non è possibile effettuare queste operazioni in simultanea.

2.1.3 `WAIT_RAM` state

Stato in cui si attende la risposta dalla memoria in seguito alla richiesta di un dato.

2.1.4 `GET_CONST` state

Stato in cui vengono memorizzati, in tre rispettivi registri, la coordinata X del punto, la coordinata Y ed infine la maschera in ingresso.

2.1.5 `FETCH_NEXT` state

Stato in cui viene controllata la maschera. Se è composta da soli `0` si va nello stato di scrittura della maschera di uscita (`WRITE_OUT`) altrimenti si controlla il bit meno significativo della maschera.
Se è un `1` viene richiesta la X del rispettivo centroide se è uno `0` si shifta a destra la maschera di una posizione.

2.1.6 `GET_CENTROID` state

Stato in cui vengono memorizzati temporaneamente, in due rispettivi registri, la coordinata X e la coordinata Y del centroide attuale.

2.1.7 `CALC_DIST` state

Stato in cui viene calcolata la distanza del centroide attuale con il punto da valutare.

2.1.8 `CHECK_DIST` state

Stato in cui viene confrontata la distanza calcolata con la minima calcolata dalle precedenti computazioni.
Se la distanza calcolata è **minore** di quella memorizzata la maschera in uscita diventa composta da soli 0 tranne il bit corrispondente all'attuale centroide che diventa `1`.
Se la distanza calcolata è **uguale** a quella memorizzata la maschera di uscita è uguale a quella memorizzata ma con il bit corrispondente al centroide posto a `1`.
Se la distanza calcolata è **maggiore** non viene eseguito nessun cambiamento.

2.1.9 `WRITE_OUT` state

Stato in cui viene scritta la maschera di uscita all'indirizzo della memoria 19 e viene posto a `1` `o_done`.

2.1.10 `DONE` state

Stato in cui si attende che la memoria abbassi `i_start` per poter abbassare `o_done` e tornare nello stato `IDLE`.

2.2 Scelte progettuali

La principale scelta progettuale effettuata è stata quella di descrivere il componente con due processi:

1. Il primo rappresenta la parte sequenziale della macchina e serve per gestire l'RT (register transfer) e quindi come vengono manipolati i registri.

2. Il secondo rappresenta invece la FSM che analizza i segnali in ingresso e lo stato corrente per determinare il prossimo stato in cui evolverà il sistema.

L'algoritmo determina il risultato finale utilizzando come operazioni logiche solamente l'`AND` tra due registri e lo `SHIFT` per controllare tutta la maschera di ingresso. Oltre a queste utilizziamo la `ADD` per incrementare l'indirizzo di memoria da richiedere, il `CAST` di un registro al corrispondente intero e infine la `ABS` per il calcolo della distanza.

Abbiamo inoltre deciso di utilizzare un approccio che prevedesse di mantenere memorizzate meno informazioni possibili. Le coordinate del centroide vengono salvate solamente per il calcolo della distanza del centroide corrente e sovrascritte con l'analisi del centroide successivo. Questo approccio offre inoltre maggiore **scalabilità** in quanto, se bisognasse aumentare il numero di centroidi da analizzare, non sarebbero necessarie modifiche al codice se non quella di cambiare le dimensioni dei vettori delle maschere.

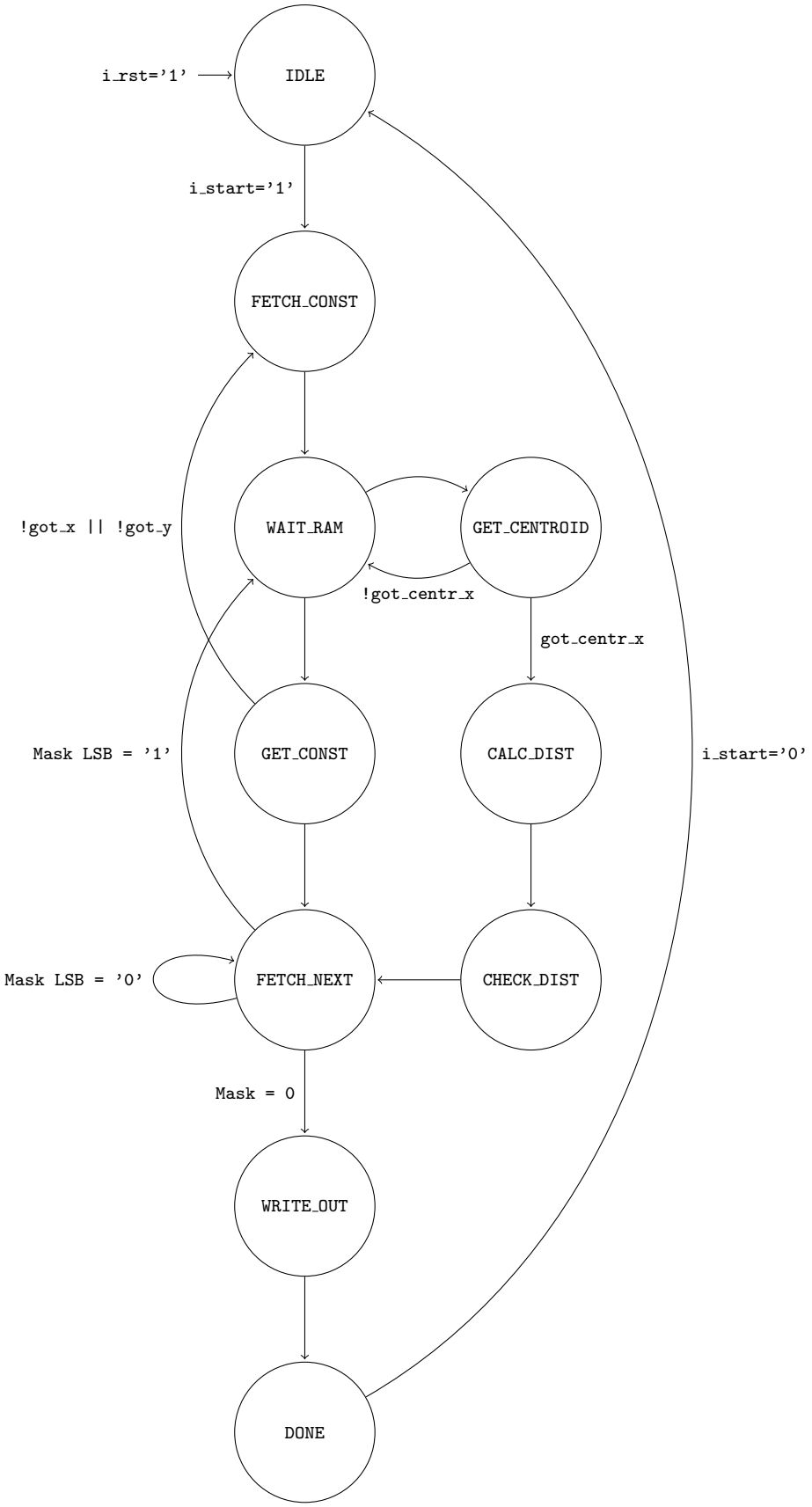

Figura 4: Macchina a Stati con le principali condizioni di transizione

3 Risultati dei test

Per verificare il corretto funzionamento del componente sintetizzato, dopo averlo testato con il test bench di esempio, abbiamo definito altri 11 test (tra i quali anche quelli che spingono la simulazione verso i corner case) in modo da cercare di massimizzare la copertura di tutti i possibili cammini che la macchina può effettuare durante la computazione.

Di seguito è fornita una breve descrizione dei test utilizzati e per quelli più significativi viene anche mostrato l'effettivo corretto funzionamento grazie allo *screenshot* dell'andamento dei segnali durante la simulazione.

I test bench per la verifica dei corner case sono 3:

1. **Maschera di ingresso nulla** ("00000000"): la maschera di ingresso è nulla. Il test verifica che il componente non consideri alcun centroide come più vicino a quello in esame.

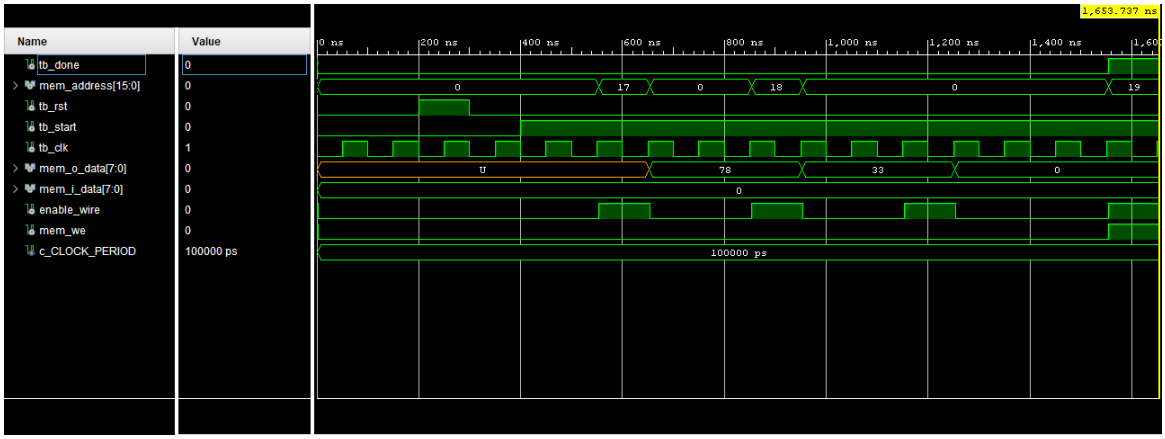

Figura 5: Simulazione con `in_mask` = "00000000"

2. **Maschera di ingresso completa** ("11111111"): la maschera di ingresso è composta da soli `1` e tutti i centroidi coincidono con il punto nella posizione (255, 255). Il test verifica che tutti i centroidi siano considerati come più vicini.

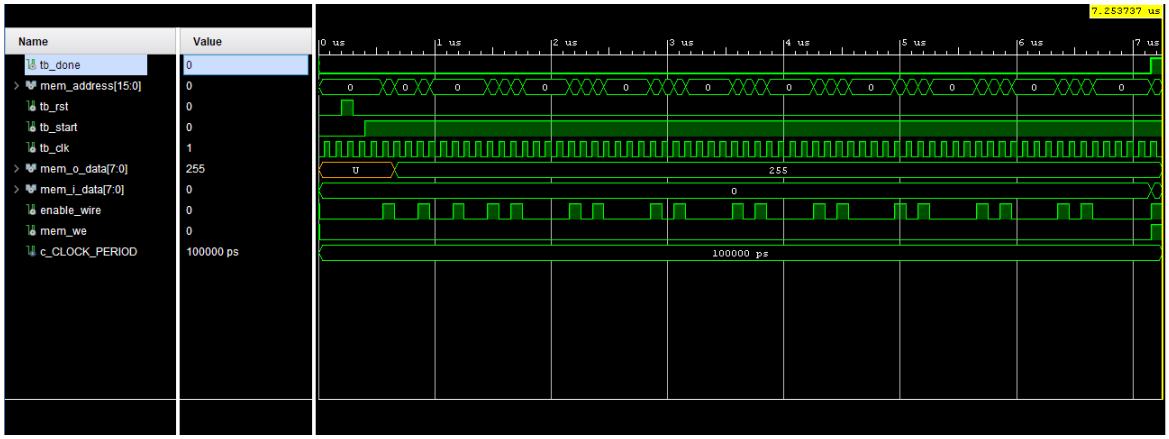

Figura 6: Simulazione con `in_mask` = "11111111"

3. **Distanza Massima**: siccome la macchina aggiorna il valore della maschera di uscita ogni volta che viene calcolata una nuova distanza dobbiamo tenere conto del caso in cui il centroide a distanza minima sia quello che si trova il più lontano possibile da quello in esame (ovvero quando i punti sono distanti 510).

I test bench che verificano il corretto funzionamento dei segnali sono 2:

1. **Reset Asincrono**: il test verifica che il trigger asincrono del segnale di reset non comprometta la computazione e che questa riinizi facendo ritornare la macchina nello stato iniziale `IDLE`.

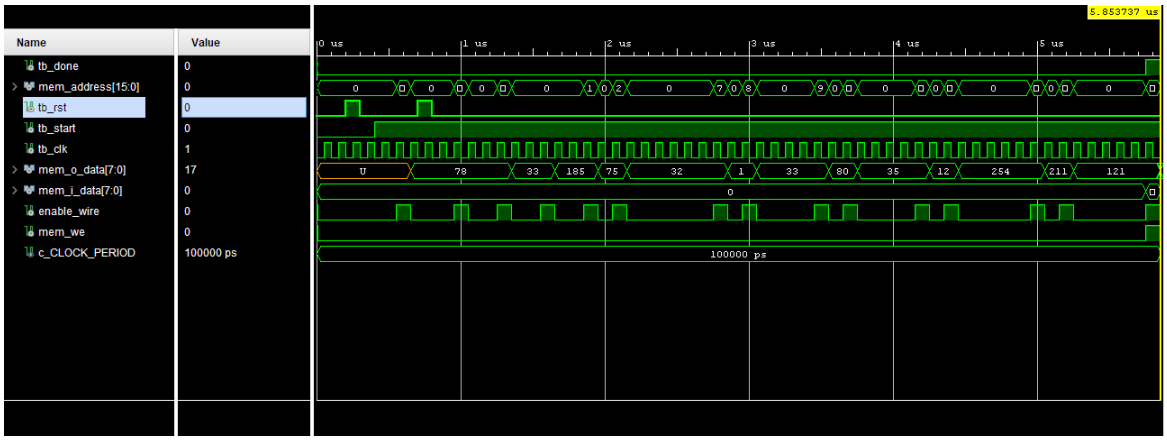

Figura 7: Simulazione con `i_rst` triggerato in modo asincrono durante la computazione

2. **Doppia computazione**: il test verifica la corretta sincronizzazione dei segnali `i_start`, `i_rst`, `o_done` andando a simulare due volte di fila la stessa memoria.

Gli ultimi test bench sono stati definiti per cercare di portare la macchina a compiere il maggior numero di cammini differenti:

1. **Primo centroide**: viene considerato solo il centroide identificato dal bit meno significativo della maschera di ingresso;

2. **Ultimo centroide**: viene considerato solo il centroide identificato dal bit più significativo della maschera di ingresso;

3. **Primo e ultimo centroide**: vengono considerati i centroidi identificati dai bit più e meno significativi della maschera di ingresso nel caso in cui l'ultimo sia più vicino a quello in esame rispetto al primo;

4. **Equidistanti a "stella"**: con maschera di ingresso che considera solo i "centroidi dispari" (01010101) e punto in esame al centro, si considerano solo centroidi situati a due a due negli angoli dello spazio;

5. **Equidistanti a "croce"**: con maschera di ingresso che considera solo i "centroidi pari" (10101010) e punto in esame al centro, si considerano solo centroidi situati a due a due sui bordi dello spazio;

6. **Punto Circondato**: il punto in esame è circondato dagli 8 centroidi dei quali (con maschera di ingresso 255) quelli a distanza minima, quindi 1, si trovano nelle direzioni: N,S,O,E.

Oltre ai test mirati precedentemente descritti, abbiamo deciso di simulare anche numerosi test randomici per testare ulteriormente il nostro componente.

Attraverso un software in C abbiamo generato una test bench con un gran numero di casi di test e li abbiamo simulati in modo automatizzato utilizzando uno script `TCL` fornito a Vivado in modalità `batch`.

L'output dello script viene salvato in un file di log e successivamente, con un comando di ricerca testo, si controlla che i messaggi delle assertions di ogni simulazione siano di superamento per ciascun test.

4 Conclusioni

4.1 Risultati della sintesi

Il componente sintetizzato supera correttamente tutti i test specificati nelle 3 simulazioni: *Behavioral*, *Post-Synthesis Functional* e *Post-Synthesis Timing*.

Qui di seguito è possibile vedere un confronto tra i tempi di simulazione dei due corner case che portano la macchina verso la più breve e la più lunga simulazione:

- **1650ns** - tempo di simulazione (*Behavioral)* con maschera in ingresso "00000000"
- **7250ns** - tempo di simulazione (*Behavioral*) con maschera in ingresso "11111111"

Per avere un parametro che quantifichi rispetto al tempo totale a disposizione quale sia il tempo del path peggiore, bisogna considerare il *Worst Negative Slack* riportato nella tabella *Design Timing Summary* del componente sintetizzato.

Setup

Worst Negative Slack (WNS):	93,322 ns
Total Negative Slack (TNS):	0,000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	241

Hold

Worst Hold Slack (WHS):	0,151 ns
Total Hold Slack (THS):	0,000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	241

Pulse Width

Worst Pulse Width Slack (WPWS):	49,500 ns
Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	132

All user specified timing constraints are met.

Figura 8: Design Timing Summary

4.2 Ottimizzazioni

Le ottimizzazioni che abbiamo attuato sono state principalmente nella riduzione del numero di stati. In prima battuta avevamo steso una macchina che richiedesse le coordinate X e Y del punto, la maschera di ingresso e le coordinate X e Y del centroide in stati differenti. Abbiamo deciso poi di raggruppare la richiesta dei primi tre dati e dei secondi due in due stati differenti risparmiando così qualche stato.

Un'altra ottimizzazione a cui abbiamo pensato è stata quella di richiedere il centroide solamente quando necessario. Se infatti il bit del centroide corrispondente è posto a zero passiamo direttamente al centroide successivo risparmiando tempo di richiesta e di risposta dalla memoria.