Prova Finale (Progetto di Reti Logiche)

Prof. Gianluca Palermo – Anno 2019/2020

Michele Veroni (Codice Persona 10579458 – Matricola 889479)
Lorenzo Zane (Codice Persona 10577011 - Matricola 890375)

Indice

1 Introduzione

1.1 Scopo del progetto

La specifica è ispirata al metodo di codifica a bassa dissipazione di potenza denominato "Working Zone": lo scopo del progetto è implementare un componente hardware, descritto in VHDL, che, preso in ingresso un address (8 bit con indirizzi validi tra 0 e 127) ed un pool di 8 working zone definita come un intervallo di indirizzi di dimensione fissa che parte da un indirizzo base (anch'esso ad 8 bit), calcola se l'address appartiene ad una working zone.
Nella versione da implementare, il numero di bit da considerare per l'indirizzo da codificare è 7 e la dimensione delle working zone è 4 indirizzi incluso quello base.
Se durante l'esecuzione non venisse trovata nessuna working zone compatibile con l'address fornito allora sarebbe modificato e ritrasmesso semplicemente concatenando davanti uno 0. Invece, nel caso si fosse trovata una compatibilità si andrà a tradurre l'indirizzo attraverso un offset rispetto alla working zone codificato in one-hot (cioè il valore che vogliamo rappresentare è equivalente all'unico bit a 1 nella codifica). Successivamente trasmesso concatenandolo con un 1 in testa.

1.2 Interfaccia del componente

Il componente da descrivere deve avere la seguente interfaccia.

```
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_clock* 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.

1.3 Segnali interni

Abbiamo definito i seguenti segnali per poter implementare correttamente il componente:

```
-- State machine signal
signal r1_load : std_logic;
signal r2_load : std_logic;
signal rDone : std_logic;
signal rCounter_load : std_logic;
signal rCounter_sel : std_logic;
signal o_end : std_logic;
signal temp : std_logic_vector (15 downto 0);

-- Datapath signal
signal o_reg1 : std_logic_vector (7 downto 0);
signal o_reg2 : std_logic_vector (7 downto 0);
signal mux_regCounter : std_logic_vector (15 downto 0);
signal add_regCounter : std_logic_vector (15 downto 0);
signal o_regCounter : std_logic_vector (15 downto 0);
signal sub : std_logic_vector (7 downto 0);
```

In particolare:

- *r1_load* è il segnale che definisce se il registro `o_reg1` debba essere caricato;
- *o_reg1* è il registro utilizzato per memorizzare l'`ADDRESS` da analizzare;
- *r2_load* è il segnale che definisce se il registro `o_reg2` debba essere caricato;
- *o_reg2* è il registro utilizzato per memorizzare le working zone;
- *sub* è il registro contenente il risultato della sottrazione tra `o_reg2` ed `o_reg1` necessaria per verificare l'eventuale appartenenza dell'`ADDRESS` ad una working zone;
- *rDone* è un segnale clone di `o_done` utilizzato in lettura durante la verifica di alcune condizioni;
- *rCounter_load*, *rCounter_sel*, *mux_regCounter*, *add_regCounter*, *o_regCounter* e *temp* sono registri utilizzati per l'implementazione di un contatore.

1.4 Dati e descrizione memoria

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

- Gli indirizzi dallo 0 al 7 contengono ciascuno una working zone;
- L'indirizzo 8 è usato per salvare l'address di input;
- L'indirizzo 9 è usato per salvare l'address di output.

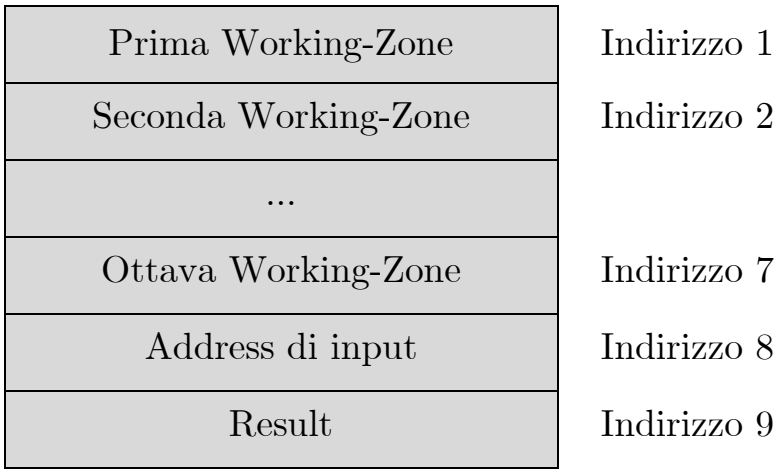


Figura 1: Rappresentazione indirizzi significativi della memoria

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 FSM nel quale si occupa di estrapolare il valore address di input che dev'essere analizzato. Una volta memorizzato l'address d'interesse per il caso di test, si sposta nel secondo stato. Nel secondo stato viene avviata la computazione che permette al componente di individuare se l'address appartiene ad una delle working zone presenti in memoria; in caso negativo, l'indirizzo viene tramesso così come è, ed un bit addizionale rispetto ai bit di indirizzamento (`WZ_BIT`) viene mezzo a 0. In caso positivo, viene computato un nuovo indirizzo secondo il seguente schema: il bit addizionale (`WZ_BIT`) viene posto a 1, i bit di indirizzamento vengono divisi in 2 sottocampi, il numero della working zone al quale l'indirizzo appartiene che sarà codificato in binario, l'offset rispetto all'indirizzo di base della working zone codificato come one-hot.
Una volta terminata l'analisi il componente riporta a 0 `o_done` per poi tornare nello stato di `IDLE`, in attesa di un nuovo segnale di `i_start`.
Il componente risponde inoltre ad un segnale di `i_rst` che, insieme agli altri, ci porta a definire una FMS con *data path*, che combina una normale FSM con tipici circuiti sequenziali. Segue la descrizione della FSM.

2.1 Stati della macchina

La macchina costruita è composta da 5 stati.

2.2 `IDLE` state

Stato iniziale della macchina a stati. Attende il segnale di `i_start` per passare allo stato successivo. Nel momento in cui viene alzato il segnale di `i_rst` la fsm torna in questo stato.

2.3 `GET_ADDRESS` state

Stato in cui viene memorizzato in un registro, il valore `ADDRESS` da analizzare per individuarne l'eventuale appartenenza ad una working zone.

2.4 `CHECK_WZ` state

Stato in cui vengono memorizzati uno alla volta, i valori delle 8 working zone, ed in cui viene calcolata l'eventuale appartenenza dell'`ADDRESS` ad uno di essi.
Se la differenza tra l'indirizzo della working zone in analisi e quello dell'`ADDRESS` è compreso tra 0 e 3, l'`ADDRESS` appartiene alla working zone correntemente in analisi.
Se la differenza risulta non essere compresa tra 0 e 3, l'`ADDRESS` non appartiene alla working zone in analisi e si passa a verificare la successiva.
Nel momento in cui viene verificata l'appartenenza dell'`ADDRESS` ad una delle 8 working zone, o, nel caso in cui si verifica che non appartiene a nessuna delle 8 working zone, la fsm passa allo stato successivo per la scrittura del risultato finale.

2.5 `WRITE_OUT` state

Stato in cui viene scritto l'`ADDRESS` finale.
Nel caso in cui quello di partenza non appartenga ad una working zone, l'address risultate viene scritto concatenando l'`ADDRESS` iniziale con uno 0 davanti. Nel caso opposto, invece, l'address risultante si ottiene traducendo l'indirizzo attraverso un offset rispetto alla working zone codificato in one-hot, successivamente trasmesso concatenandolo con un 1 in testa.
In questo stato viene posto ad 1 `o_done`.

2.6 `DONE` state

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

2.7 Scelte progettuali

La principale scelta progettuale risiede nel tentativo di sviluppare una FSM più lineare e semplice possibile a costo di ottenere condizioni leggermente più elaborate. Questa scelta di raggruppare in pochi stati tutte le condizioni ci ha permesso di ottenere risultati più prestazionali in termini temporali, senza sprecare cicli di clock.

L'algoritmo determina il risultato finale utilizzando come operazioni logiche solamente la SUB tra due registri e tenendo conto dei passi grazie ad un COUNTER.
Vengono inoltre memorizzate pochissime informazioni, strettamente necessarie. L'`ADDRESS` iniziale viene memorizzato per permettere la sottrazione necessaria al confronto negli stati successivi; gli indirizzi delle working zone vengono invece salvati per i pochi cicli di clock necessari ad eseguire la sottrazione ed il controllo sull'eventuale appartenenza dell'`ADDRESS` alla corrente working zone.

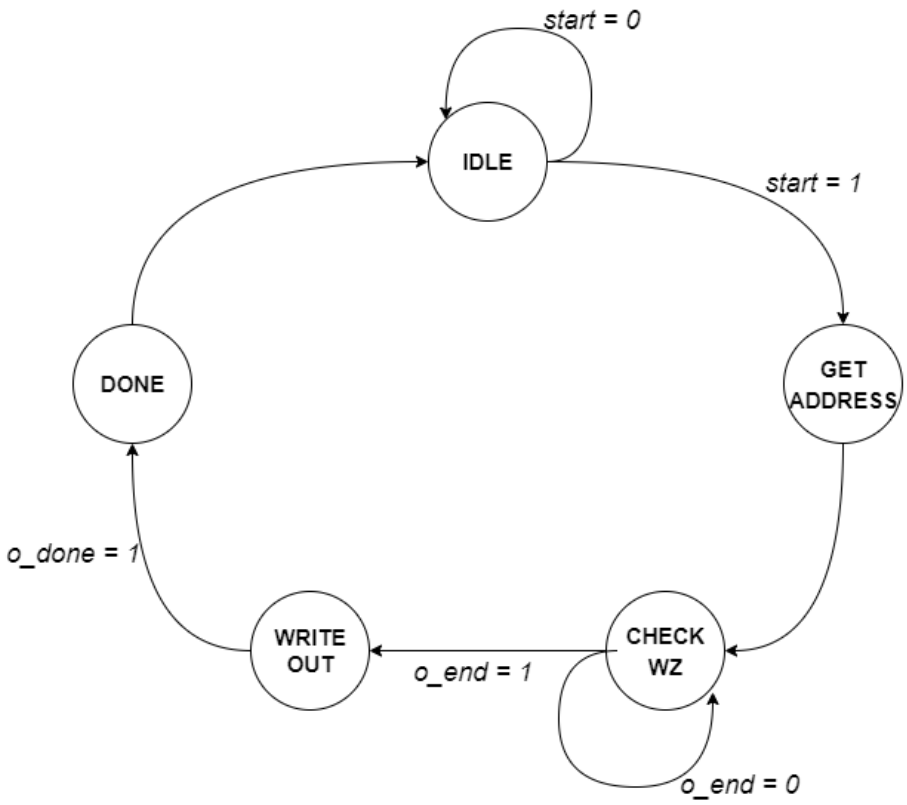


Figura 2: Macchina a Stati con le principali condizioni di transizione

Le transizioni della macchina a stati qui riportata risultano essere una semplificazione di quelle reali. Sono state omesse le transazioni lanciate dal comando di reset ed alcuni degli autoanelli necessari alla corretta sincronizzazione dei cicli di clock con i segnali per una corretta lettura da memoria.

3 Risultati

3.1 Sintesi

Il componente risulta correttamente sintetizzabile ed implementabile con l'utilizzo (in implementazione) di 53 LUT e 99 FF.

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF
synth_1	constrs_1	synth_design Complete!								56	83
impl_1	constrs_1	route_design Complete!	93.640	0.000	0.166	0.000	0.000	0.131	0	53	99

3.2 Test

Il corretto funzionamento del componente sintetizzato è stato testato a partire dai test bench di esempio, sono stati poi eseguiti vari test nel tentativo di coprire la maggior parte di casi limite possibili.
Abbiamo eseguito test che presentano reset multipli durante l'esecuzione, test con reset multipli asincroni lanciati con timing di 1ns dall'istruzione precedente ed una serie di test generati automaticamente con migliaia di combinazioni di test contenenti singoli segnali di start, doppi segnali di start durante lo stesso test set, reset lanciati dopo 4 cicli di clock e doppi segnali di start con configurazioni diverse, in modo da verificare la rilettura del nuovo address dopo il secondo start.

- **Indirizzo non appartenente ad una working zone:** Ingresso 42, uscita 42.
  Il test verifica che il componente gestisca correttamente l'eventualità in cui l'indirizzo in analisi non appartenga a nessuna working zone.

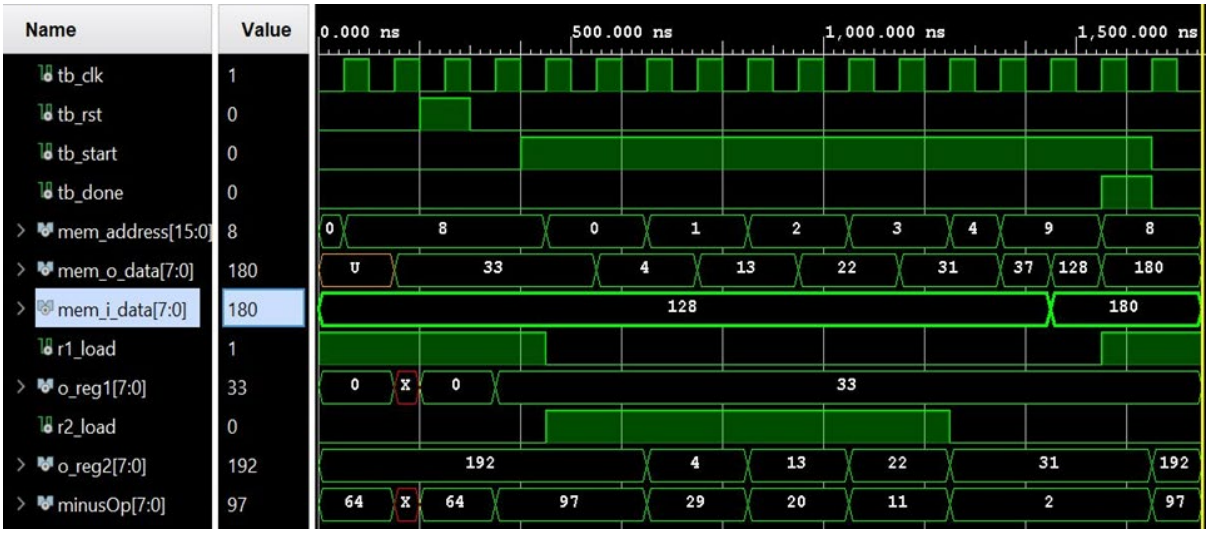


Figura 3: Simulazione con indirizzo non appartenente ad una working zone

- **Indirizzo appartenente alla working zone numero 4:** Ingresso 33, uscita 180.
  Il test verifica che il componente gestisca correttamente l'eventualità in cui l'indirizzo

in analisi appartenga ad una working zone in memoria (diversa dalla prima).

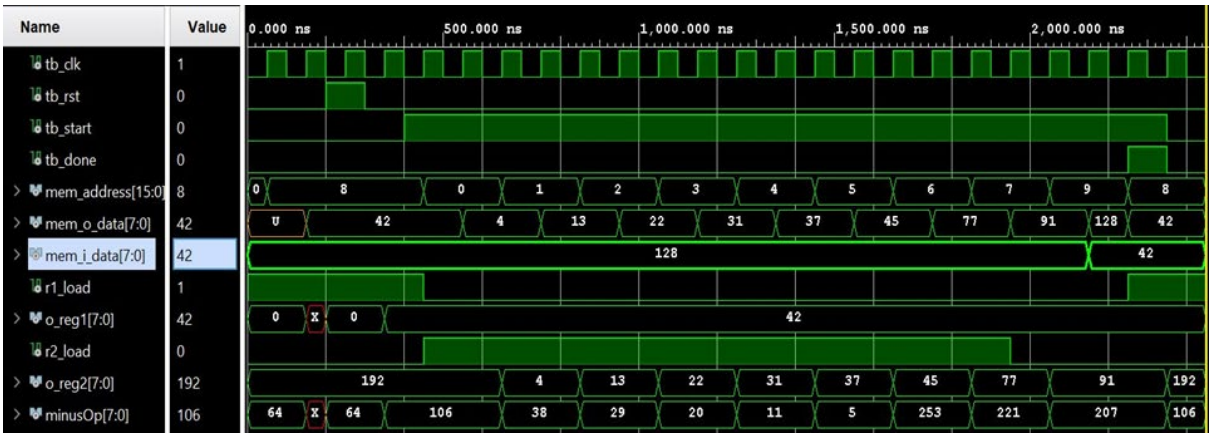


Figura 4: Simulazione con indirizzo appartenente ad una working zone

- **Test con reset asincrono dopo 4 cicli di clock dello start:**
  Questo test lancia un segnale di reset a distanza di 4 cicli di clock dal momento in cui viene alzato il segnale di start.

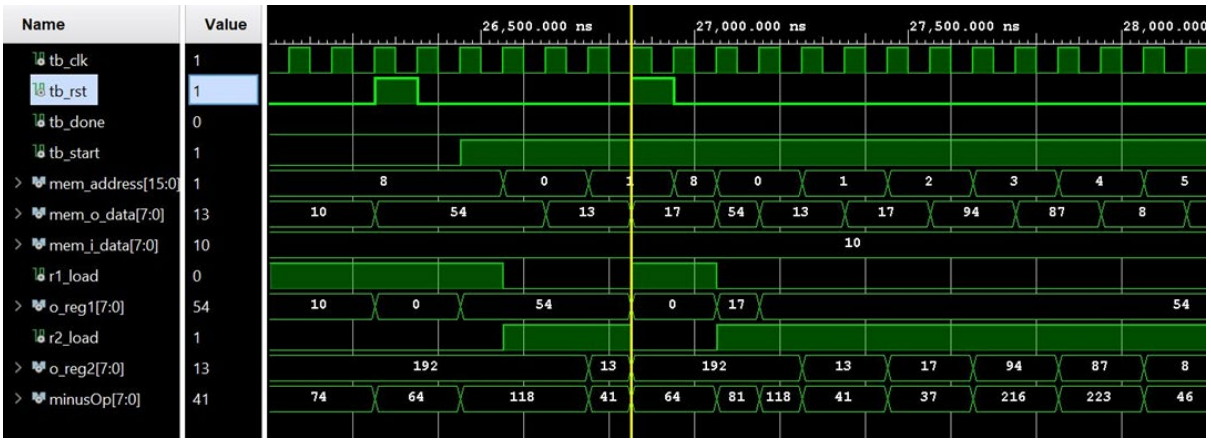


Figura 5: Simulazione con reset asincrono dopo 4 cicli di clock dallo start

- **Test doppio start:**
  Questo test consiste nell'eseguire due codifiche senza la ricezione di un segnale di reset tra le codifiche. Il test mantiene le stesse working zone, alza il segnale di reset per la prima e dopo aver ricevuto il segnale di `o_done`, modifica l'`ADDRESS` da analizzare e riporta `i_start` ad 1. Il componente risulta quindi in grado di codificare nuovamente con le stesse working zone ma address diverso.

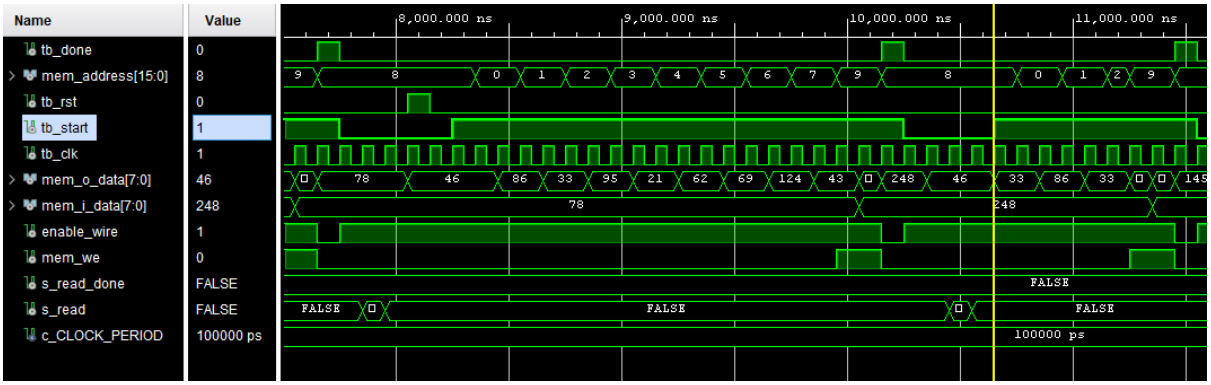


Figura 6: Simulazione con doppio start

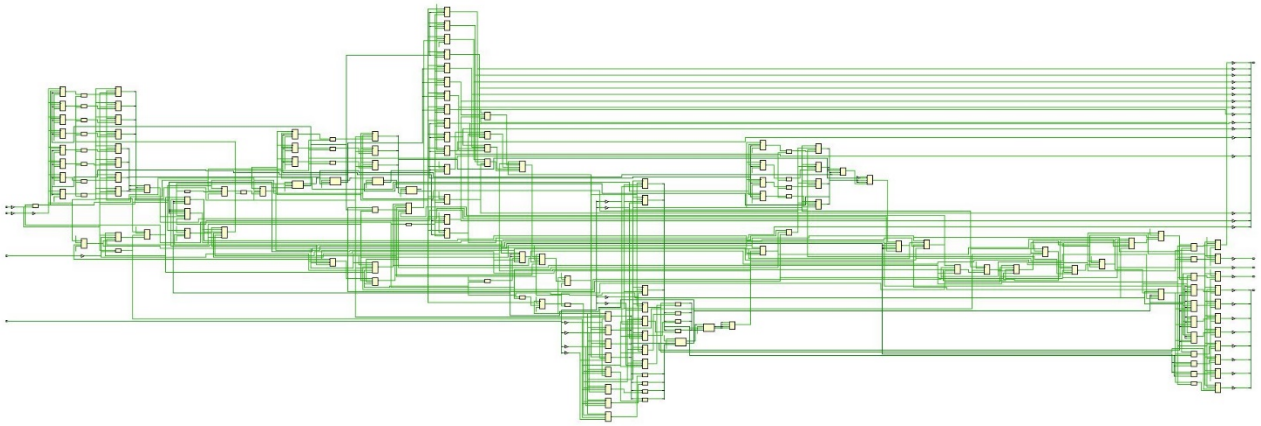
4 Conclusioni

4.1 Risultati della sintesi

Il componente sintetizzato supera senza errori tutti i test precedentemente presentati sia in *Behavioral* che in *Post-Synthesis Functional*.
Inoltre, abbiamo calcolato il minimo e il massimo tempo di simulazione (*Behavioral*) così da fornire il caso peggiore, migliore e medio:

- **2450ns** - verificato nel caso di indirizzo che non appartiene a nessuna working zone
- **1150ns** - verificato nel caso in cui la working zone sia nella prima cella di memoria
- **≃1800ns** - rispetto a test generati casualmente

Il componente sintetizzato ha il seguente schema:



Per quantificare quale sia il tempo del path peggiore, bisogna considerare il *Worst Negative Slack* del componente sintetizzato rispetto al tempo totale a disposizione (di 100ns), riportato nella tabella:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	93.640 ns	Worst Hold Slack (WHS):	0.166 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	168	Total Number of Endpoints:	168	Total Number of Endpoints:	100

All user specified timing constraints are met.

4.2 Ottimizzazioni

Come discusso tra le scelte di progetto, le ottimizzazioni a cui abbiamo lavorato fin dall'inizio sono state principalmente legate al creare una FSM con il minor numero di stati possibile. L'automa possiede oltre allo stato di `IDLE`, un unico stato per la lettura dell'`ADDRESS` da analizzare ed un unico stato per l'analisi di appartenenza a tutte le working zone.

È stata ottimizzato anche ciò che era necessario memorizzare all'interno del componente, esso infatti ha necessità di mantenere solamente il dato `ADDRESS` per permettere di effettuare il confronto con tutte le altre working zone, non è necessaria la memorizzazione di nessun altro parametro.