POLITECNICO
MILANO 1863

PROVA FINALE (Progetto di Reti Logiche)

Filisetti Rossana (Codice Persona 10744562 - Matricola 960175)

Anno Accademico: 2022/2023

Docente: Fabio Salice

INDICE

1. INTRODUZIONE

1.1 Descrizione del progetto:

Lo scopo del progetto è l'implementazione di un modulo hardware descritto in VHDL che indirizza verso un preciso canale di output i dati contenuti in un determinato indirizzo di memoria. L'indirizzo di memoria da cui recuperare i dati viene acquisito tramite un canale d'ingresso seriale da 1 bit, in cui viene anche specificato in quale canale, tra i 4 a disposizione, indirizzare i dati ricevuti. In particolare i primi due bit in input nel canale seriale indicheranno il canale di output e gli N restanti (estesi a 16 bit) l'indirizzo della memoria. Il segnale in ingresso viene considerato solo se è attivo un segnale di START.
I canali di uscita mostreranno il messaggio solo per un ciclo di clock quando è attivo un segnale DONE, in tutti gli altri momenti conterranno il valore "0000 0000".

1.2 Esempio funzionamento:

Se avessimo in input un segnale di START pari a "01110011110..." e un ingresso "10110100101..." verrebbe considerato per una prima elaborazione solo il segmento di input "011", cioè quello corrispondente al primo segnale di start alto. Questo input viene suddiviso nei due bit per selezionare il canale d'uscita ("01", i dati saranno quindi mostrati sul canale Z1) e gli N bit (estesi poi a 16 bit) dell'indirizzo di memoria da cui recuperare i dati ("0000 0000 0000 0001"). Alla fine dell'esecuzione, per un ciclo di clock, avremo il segnale DONE alto e vedremo il valore contenuto nell'indirizzo di memoria indicato sul canale Z2.
In una seconda elaborazione in cui verrebbe considerato il secondo segmento corrispondente ad un segnale di START alto avremmo Z0 come canale di uscita e "0002" (exa) come indirizzo di memoria di riferimento. Nel ciclo di clock in cui DONE diventa '1', vedremo il nuovo dato estratto sul canale Z0 e il vecchio dato nel canale Z2.

1.3 Descrizione dell'interfaccia:

L'interfaccia del componente è:

```
entity project_reti_logiche is
    port (
        i_clk   : in std_logic;
        i_rst   : in std_logic;
        i_start : in std_logic;
        i_w     : in std_logic;
        o_z0    : out std_logic_vector(7 downto 0);
        o_z1    : out std_logic_vector(7 downto 0);
        o_z2    : out std_logic_vector(7 downto 0);
        o_z3    : out std_logic_vector(7 downto 0);
        o_done  : out std_logic;
        o_mem_addr : out std_logic_vector(15 downto 0);
        i_mem_data : in std_logic_vector(7 downto 0);
        o_mem_we   : out std_logic;
        o_mem_en   : out std_logic
    );
```

```
end project_reti_logiche;
```
Dove:

- `i_clk` è il segnale di CLOCK in ingresso generato dal Test Bench;
- `i_rst` è il segnale di RESET che inizializza la macchina per ricevere il primo segnale di START;
- `i_start` è il segnale di START generato dal Test Bench;
- `i_w` è il segnale da cui vengono recuperate le informazioni ed è generato dal Test Bench;
- `o_z0, o_z1, o_z2, o_z3` sono i quattro canali di uscita;
- `o_done` è il segnale di uscita che comunica la fine dell'elaborazione;
- `o_mem_addr` è il segnale di uscita che manda l'indirizzo alla memoria;
- `i_mem_data` è il segnale che arriva dalla memoria in seguito ad una richiesta di lettura;
- `o_mem_en` è il segnale di ENABLE per la comunicazione con la memoria;
- `o_mem_we` è il segnale di WRITE ENABLE per poter scrivere in memoria. (In questo particolare progetto, non essendoci la necessità di scrivere in memoria, questo segnale resterà costante e pari a 0)

2. ARCHITETTURA

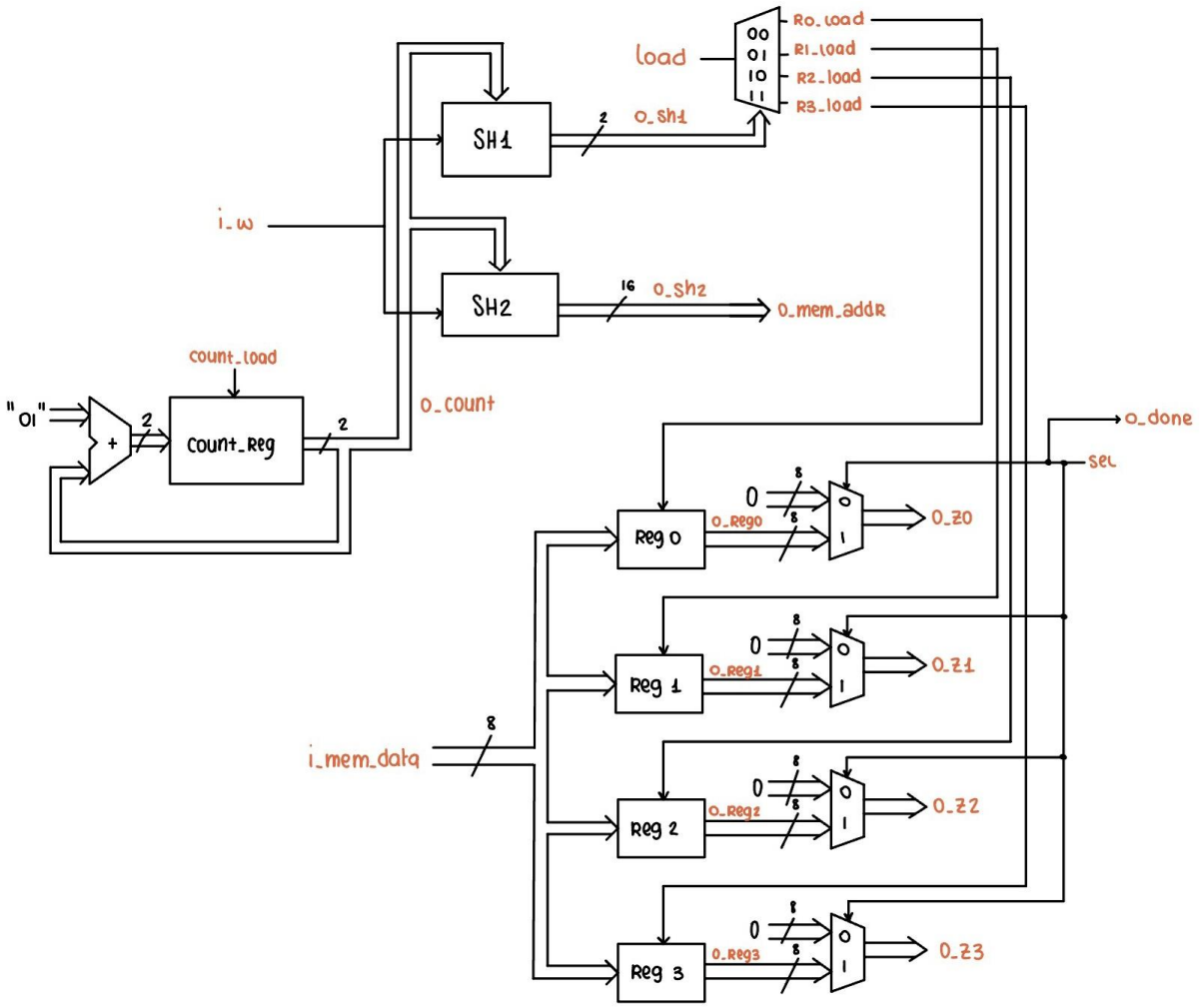
Il componente è composto da due moduli principali che lavorano in contemporanea: un primo modulo gestisce i registri e gli altri componenti del circuito sequenziale corrispondente, mentre un secondo implementa la macchina a stati.

2.1 Datapath

Per realizzare il progetto ho iniziato sviluppando un circuito sequenziale poi descritto in una entity datapath. Questo modulo è composto da vari processi che gestiscono i dati in ingresso e uscita dalla RAM.
Il circuito è descritto nel seguente schema in cui sono presenti:

- un contatore iniziale gestito dalla FSM che permette di dividere il dato in ingresso nelle due informazioni che contiene;
- due shift register (SH1 E SH2) attivati dal counter che permettono di trasformare l'ingresso seriale in parallelo e contengono rispettivamente l'indirizzo di uscita finale e l'indirizzo di memoria da cui prelevare i dati;
- un demultiplexer che permette il caricamento di dati nel registro corrispondente al canale di uscita ed è gestito dalla FSM;
- quattro registri per memorizzare i dati in uscita;
- quattro multiplexer che gestiscono il dato in uscita sui canali `o_z`: se il segnale `sel` è alto sui canali si vedrà il valore caricato nel relativo registro, altrimenti si vedrà il valore '0000 0000'.

Non presenti nel disegno riportato in seguito, ma comunque presenti nel circuito sono i segnali di clock e reset collegati ai registri.



2.2 Macchina a stati

La macchina a stati è implementata in un secondo modulo diviso in due processi: uno per la gestione dei registri e uno per la gestione della logica degli stati.
Dopo vari miglioramenti la macchina finale risulta avere 6 stati:

1. **RESET state**

   Stato di reset in cui la macchina attende l'inizio del processo. Oltre ad essere lo stato iniziale, viene raggiunto anche nel caso in cui `i_rst` diventi '1' in qualsiasi momento del processo, mentre si passa allo stato successivo nel momento in cui il segnale `i_start` sale a '1'.

2. **LOAD state**

   In questo stato viene alzato il segnale `count_load` che permette l'attivazione del contatore all'interno del circuito. La macchina resta poi in questo stato finché il segnale `i_start` torna a '0' e quindi tutti i dati sono stati raccolti dal flusso d'ingresso.

3. **MEM_READ state**

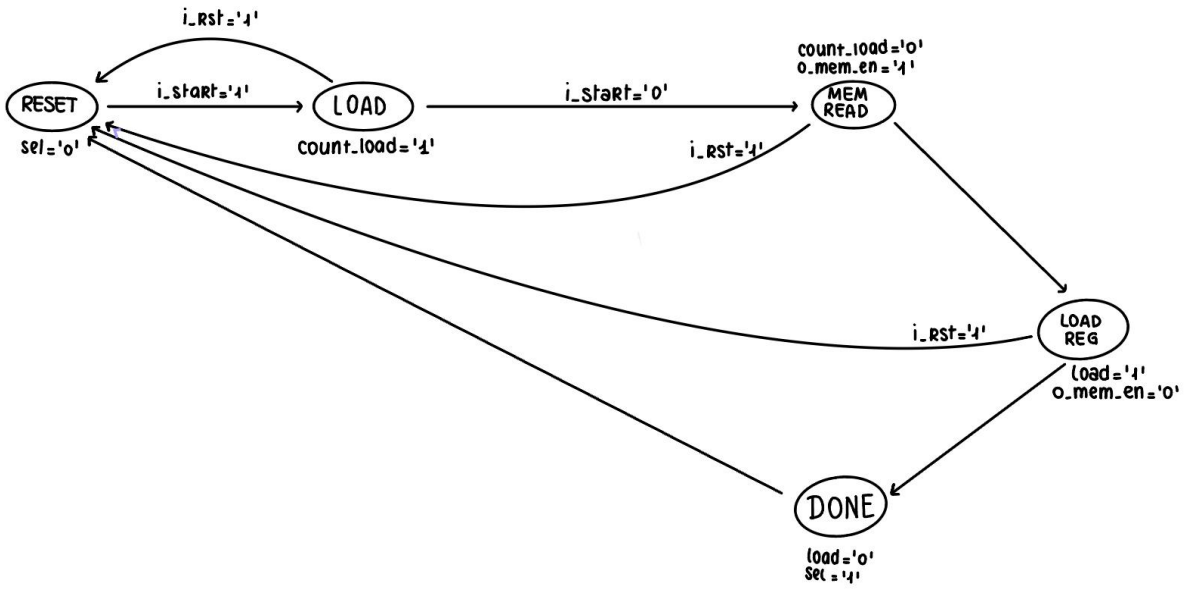
Stato di lettura dell'informazione dalla memoria: `o_mem_en` viene posto = '1' e `count_load` torna a '0'.

4. **LOAD_REG state**

   In questo stato vengono caricati i registri collegati alle varie uscite `o_Zk` alzando a '1' il segnale `load` che, attraverso il demultiplexer, permette il caricamento del dato nel registro relativo al corretto canale di output. Viene inoltre riposto a '0' il segnale `o_mem_en`.

5. **DONE state**

   Stato finale dell'elaborazione: il segnale `load` viene riposto = '0' e il segnale `sel` viene posto a '1', permettendo ai canali di uscita di mostrare i dati presenti nei rispettivi registri.



3. *RISULTATI SPERIMENTALI*

Oltre a testare casi generali, per valutare la correttezza del componente lo ho sottoposto a test che ne valutassero il comportamento in casi particolari o casi limite quali:

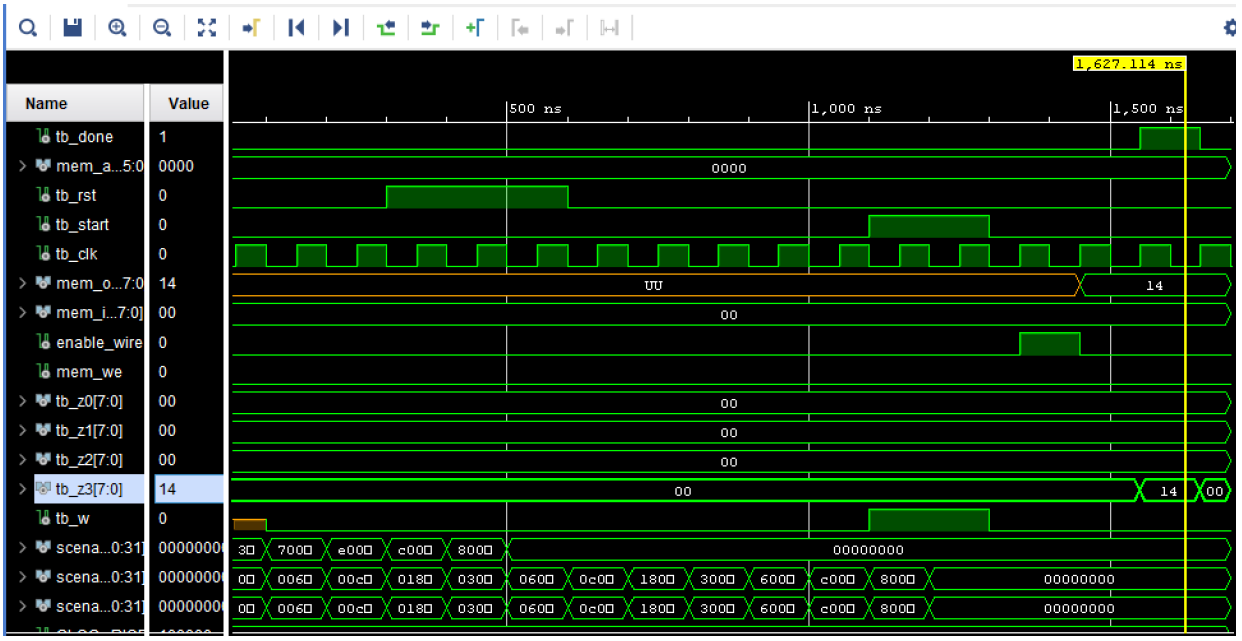
1. Indirizzo vuoto: segnale `i_start` alto per soli 2 cicli di clock. In questo caso il dato dell'indirizzo di memoria da cui prelevare l'informazione è vuoto. Il test controlla che

l'indirizzo venga correttamente esteso in "0000" (exa) e soprattutto che la durata del segnale di start breve non crei problemi per il contatore.
2. Indirizzo pieno: segnale `i_start` alto per 18 cicli di clock. Il test verifica che la lettura del dato venga completata e elaborata correttamente nel caso in cui l'indirizzo di memoria sia di lunghezza massima e quindi non necessiti di estensione.
3. Reset casuale: viene verificato che l'arrivo del segnale di reset in un qualsiasi momento della computazione faccia tornare il componente allo stato di RESET senza creare problemi.
4. Lunghezza done: il test verifica che il segnale done resti alto per un unico ciclo di clock
5. Distanza start-done: il test verifica che sia rispettata la richiesta di mantenere la distanza tra la discesa del segnale start e la salita del segnale done inferiore ai 20 cicli di clock.

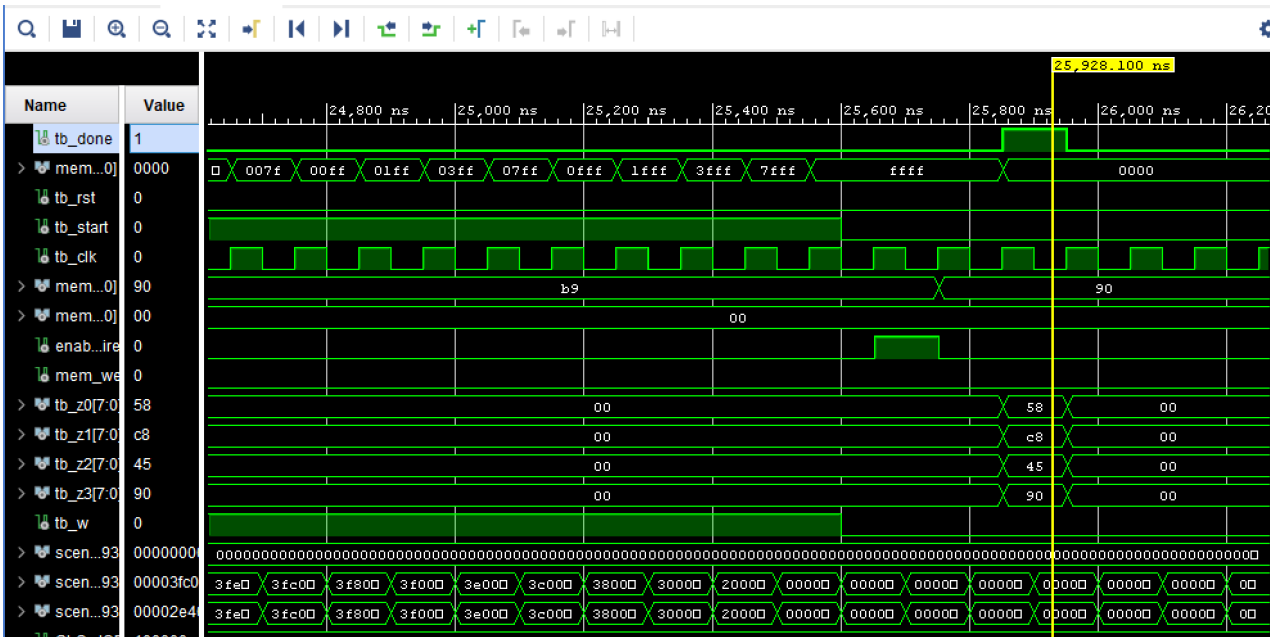
Il componente ha superato con successo tutti i test a cui l'ho sottoposto sia in pre-sintesi, sia in post-sintesi.

Di seguito sono riportate le waveform di alcuni test eseguiti:

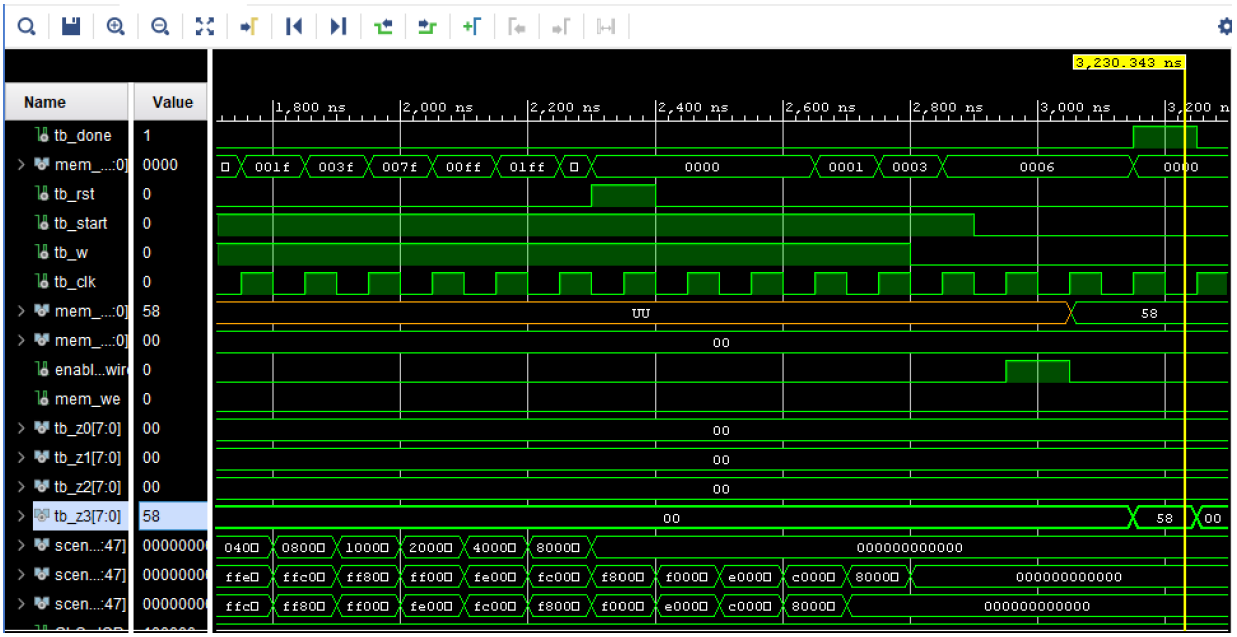
1. Indirizzo vuoto: i_start alto per 2 cicli di clock



2. Indirizzo pieno: start alto per 18 cicli di clock



3. Reset casuale: in questo particolare test non è stato verificato il caso in cui il segnale di reset lasci disponibile un unico segnale di start per l'elaborazione in quanto nella specifica del progetto è comunicato che non è necessario gestire il caso in cui il segnale di start rimanga attivo meno di 2 cicli di clock.



3.2 Risultati sintesi

La sintesi viene completata con successo riportando solo due warnings:

- "[Synth 8-3917] design project_reti_logiche has port o_mem_we driven by constant 0" in quanto non essendoci scrittura in memoria il segnale `o_mem_we` rimane costante a 0;
- "[Constraints 18-5210] No constraint will be written out" che risulta essere una conseguenza dell'utilizzo della versione 2018.2 di Vivado (facendo girare il progetto su altre versioni di Vivado questo warning non compare).

Dal ‘report utilization’ risulta in particolare che vengono utilizzati 31 Lookup Table, 55 Flip Flop e nessun Latch.

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	31	0	134600	0.02
LUT as Logic	31	0	134600	0.02
LUT as Memory	0	0	46200	0.00
Slice Registers	55	0	269200	0.02
Register as Flip Flop	55	0	269200	0.02
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

1.1 Summary of Registers by Type

Total	Clock Enable	Synchronous	Asynchronous
0	_	-	-
0	_	-	Set
0	_	-	Reset
0	_	Set	-
0	_	Reset	-
0	Yes	-	-
0	Yes	-	Set
55	Yes	-	Reset
0	Yes	Set	-
0	Yes	Reset	-

Dal ‘report timing’ invece risulta che lo slack è di 97.006 ns, quindi i constraints (100 ns di periodo di clock) sono altamente soddisfatti e il modello funziona a circa 3 ns.

```
Timing Report

Slack (MET) :             97.004ns  (required time - arrival time)
  Source:                 FSM_sequential_cur_state_reg[1]/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@50.000ns period=100.000ns})
  Destination:            DATAPATH0/o_count_reg[0]/CLR
                            (recovery check against rising-edge clock clock  {rise@0.000ns fall@50.000ns period=100.000ns})
  Path Group:             **async_default**
  Path Type:              Recovery (Max at Slow Process Corner)
  Requirement:            100.000ns  (clock rise@100.000ns - clock rise@0.000ns)
  Data Path Delay:        2.407ns  (logic 0.751ns (31.201%)  route 1.656ns (68.799%))
  Logic Levels:           1  (LUT4=1)
  Clock Path Skew:        -0.145ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.100ns = ( 102.100 - 100.000 )
    Source Clock Delay      (SCD):    2.424ns
    Clock Pessimism Removal (CPR):    0.178ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
```

4. CONCLUSIONI

Il problema principale su cui mi sono concentrata durante lo sviluppo del progetto è stato rimuovere i Latch che risultavano presenti nelle prime implementazioni e diminuire gli stati della FSM la quale all’inizio presentava due stati solo per dividere i primi due bit ingresso dal resto dell’informazione. Con l’introduzione del contatore per fare questo lavoro sono riuscita a ridurre il numero di stati e aumentare lo slack da circa 95 ns ai 97 ns finali.