Prova finale Reti Logiche

Stefano Da Silva - 10583319

Anno Accademico 2022/2023

Indice

1 Introduzione

Il progetto consiste nella implementazione di un modulo hardware descritto con in linguaggio VHDL in grado di soddisfare la specifica fornita di seguito riassunta.
Il modulo presenta 2 ingressi principali **W** e **START** e 5 uscite **Z0**,**Z1**,**Z2**,**Z3**, **DONE**.
L'ingresso seriale **W** identifica una sequenza che può variare da un minimo di 2 bit ad un massimo di 18 bit. La sequenza viene ritenuta valida da quando **START** diviene uguale ad '1' fino a che non diviene '0', il segnale di **START** rimane alto per minimo 2 cicli di clock e massimo 18.
La sequenza prelevata identifica 2 valori : i primi 2 bit rappresentano il canale di uscita su cui verrà scritto l'output prelevato da memoria , rispettivamente ('00' **Z0**, '01' **Z1**, '10' **Z2**, '11' **Z3**), mentre i restanti bit descrivono l'indirizzo di memoria da cui verrà estratto il dato da scrivere in uno di questi canali. Gli indirizzi di memoria sono a 16 bit, ciò significa che l'indirizzo verrà esteso a 16 bit qualora la sequenza che rappresenta l'indirizzo non raggiunga tale numero. Quando viene scritto il dato all'interno di uno dei 4 canali **Z** di uscita il segnale **DONE** viene posto ad 1 e resta alto unicamente nel ciclo di clock in cui viene scritto il dato, contemporaneamente a questo evento viene mostrato il valore contenuto all'interno dei 4 canali. In tutti gli altri casi il valore di **DONE** vale 0 così come quello del dato in uscita dai canali.

Il modulo presenta un segnale di reset ed un segnale di clock, unici per tutto il sistema. Prima del primo **START** il modulo viene inizializzato resettandolo, cosa che non sarà necessaria per le altre eventuali eleborazioni.

```
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;
```

Figure 1: Interfaccia del componente

1.1 Interfaccia del componente

- **i_clk** è il segnale di CLOCK in ingresso generato dal Test Bench;
- **i_rst** è il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START;
- **i_start** è il segnale di START generato dal Test Bench;
- **i_w** è il segnale W precedentemente descritto e 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_add** è il segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- **o_mem_en** è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- **i_mem_data** è il segnale (vettore) che arriva dalla memoria in seguito ad una richiesta di lettura;
- **o_mem_we** è il segnale di WRITE ENABLE da dover mandare alla memoria (=1) per poter scriverci. Per leggere da memoria esso deve essere 0.

Di seguito un'esecuzione di esempio. In questo caso i _w è 1 per i primi 3 cicli di clock per poi diventare 0, per giunta il dato da scrivere verrà preso dall'indirizzo di memoria 000000000001 per poi essere riportato nel canale '11' ovvero Z3. Il dato verrà scritto e mostrato nel ciclo di clock in cui done diventa uguale a 1.

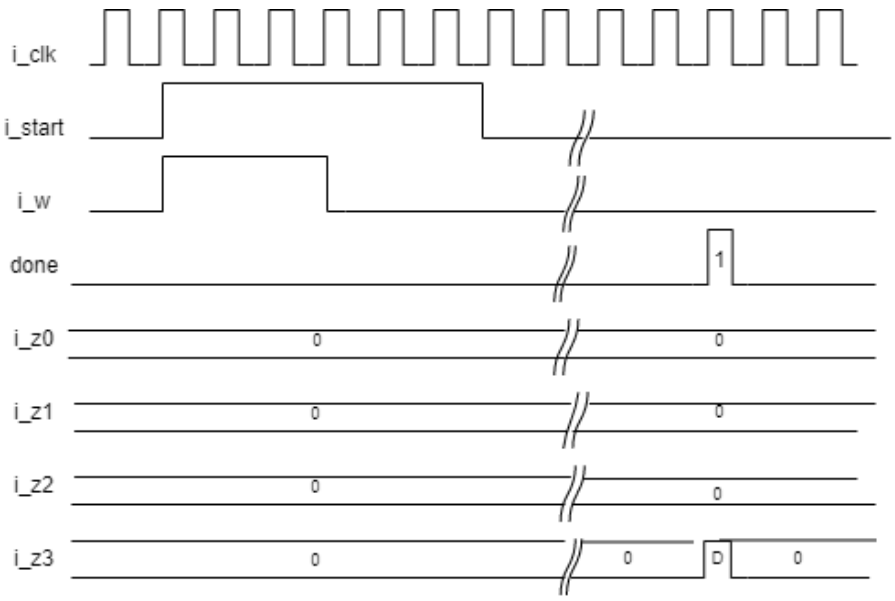

Figure 2: Esecuzione di esempio

2 Architettura

Per risolvere il problema ho deciso di utilizzare un componente che rappresentasse il datapath chiamato **DATAPATH**, e una **FSM** di Moore composta da diversi stati che interagiscono con il datapath .

2.1 Datapath

2.1.1 Lettura della sequenza in ingresso

In figura 3 è presente lo schema del Datapath. Prima di tutto dovevo trovare un modo per salvare la sequenza, vista la natura seriale del dato in ingresso ho deciso di utilizzare due registri a scorrimento di tipologia serie-parallelo, in grado di prendere in ingresso un input seriale e fornire un' uscita parallela. I primi 2 bit della sequenza che rappresentano il canale di uscita vengono salvati nel registro da 2 bit **reg_1**, i rimanenti nel **reg_2**, un registro da 16 bit. Entrambi i registri oltre che essere connessi al reset di sistema possiedono dei reset personali che vengono attivati ogni volta che viene terminata la produzione del dato in uscita. In questa maniera si risolve immediatamente il problema dell'estensione del segno nel caso di un indirizzo fornito a meno di 16 bit, in quanto il registro che salva l'indirizzo ha sempre 16 bit che a fine esecuzione vengono azzerati. Per una corretta sincronizzazione l'uscita di questi 2 registri viene poi connessa rispettivamente ai registri

parallelo-parallelo **reg_mem_addr** e **select_dmux** rispettivamente da 16 bit e 2 bit. L'uscita **reg_mem_addr _out** è collegata poi con il segnale **o_mem_addr** ovvero l'indirizzo della memoria da cui leggere. **Enable** serve a dare il segnale di load al **reg _2**, mentre **input _reg _sel** ha un segnale di uscita che viene mandato alla FSM per specificare quando cambiare stato della macchina, come specificato meglio nella descrizione della FSM.

2.1.2 Produzione dell'output in uscita

Il dato prelevato dalla memoria viene salvato nel registro parallelo-parallelo da 8 bit **mem_data** la cui uscita è connessa all'ingresso del demultiplexer **demux**. Questo ha la funzione di instradare il dato ricevuta dalla memoria in uno dei registri intermedi **reg4**, **reg5**, **reg6**, **reg7**. Quest'ultimi sono tutti registri da 8 bit, compatibilmente al dato che viene prelevato dalla memoria e svolgono la funzione di contenere il dato che poi dovrà essere mostrato in un momento specifico della computazione. Il canale di uscita del demultiplexer viene scelto tramite l'ingresso di selezione **select_dmux** calcolato al punto precedente. I segnali di uscita sono mostrati nella figura 3. Il segnale di uscita di ognuno dei quattro registri è collegato al primo ingresso di un multiplexer, ce ne sono quattro, uno per ogni registro e sono connessi in questa maniera : reg4 $\rightarrow$ mux4, reg5 $\rightarrow$ mux5 ecc. Il secondo ingresso dei multiplexer è la costante '00000000'. L'ingresso di selezione dei 4 mux è comune e viene governato dalla macchina a stati, mentre l'uscita di ogni mux è rappresentata dai vari canali **Z0** , **Z1**, **Z2**, **Z3**. Una volta prelevato il dato ingresso e salvato nel registro corrispondente la FSM imposterà il segnale **mux _selector** ad 1 insieme a **DONE**. Quando **mux _selector** è '1' il valore che i multiplexer producono in uscita è uguale al primo ingresso, cioè il valore di uscita dei registri sopracitati, in caso contrario il valore in uscita è il secondo ingresso, zero. Così facendo si rispetta la specifica secondo la quale il valore dei quattro canali Z sia diverso da 0 solo nel momento in cui **DONE** è 1. I registri reg4..reg7 a differenza degli altri hanno un solo reset che è quello di sistema poichè non devono essere azzerati alla fine di ogni produzione del dato in uscita.

2.2 FSM

I segnali di load dei registri, i loro reset personalizzati e gli altri segnali descritti nella sezione precedente vengono governati da una macchina di Moore a 10 stati. In input riceve i seguenti segnali:

- **i_start**
- **input_reg_out**
- **reg4_7_selectors**
- **curr_start**

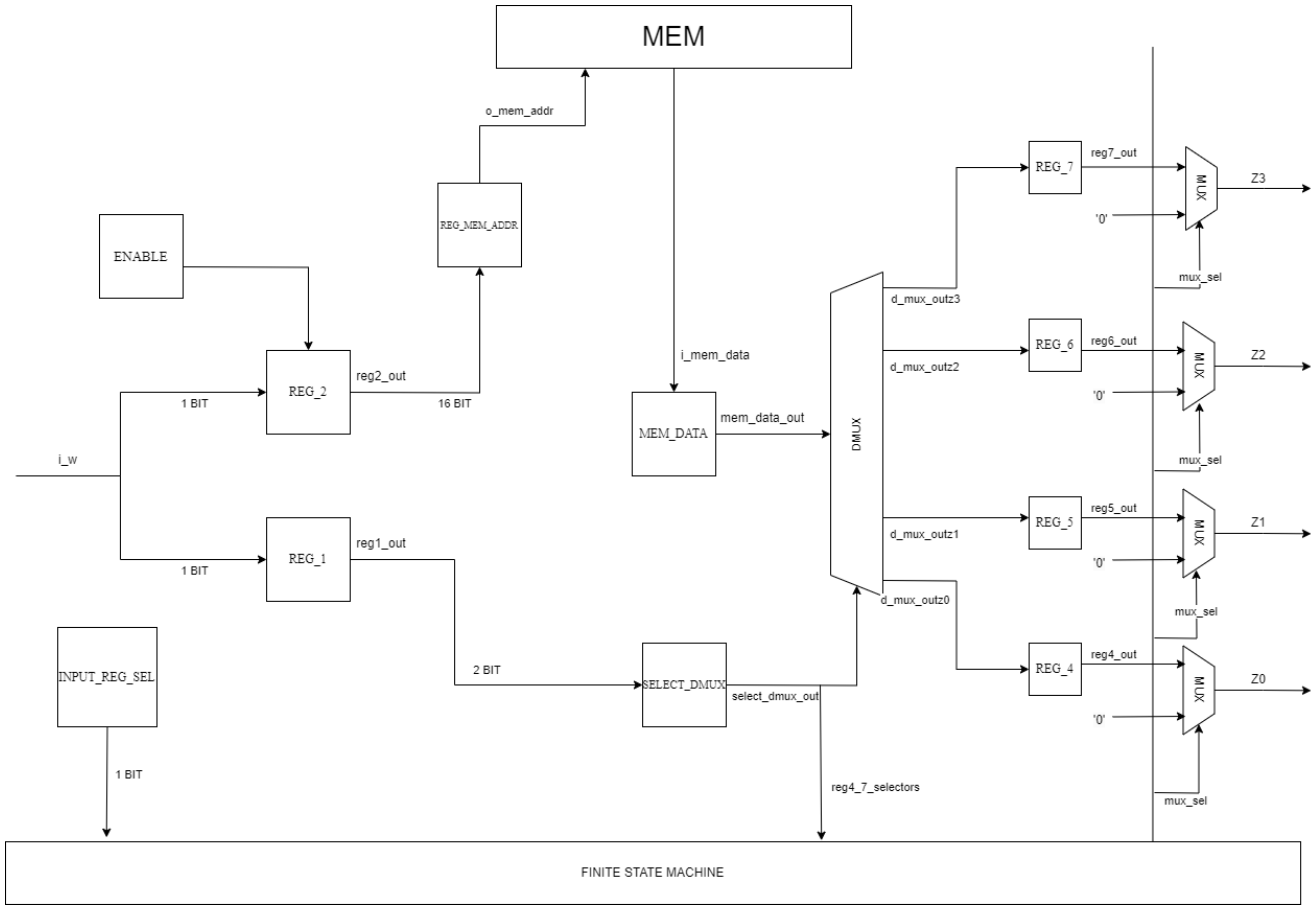

Figure 3: Datapath

La macchina è costituita da tre process, uno per la gestione dell'avanzamento degli stati della macchina, uno per la gestione dei segnali del datapath e infine un process per la gestione dei segnali curr_state e next_state. Di seguito l'elenco dettagliato degli stati :

- **WAIT_START** : è lo stato iniziale della macchina , ogni volta che arriva un reset del sistema si ritorna a questo stato. La macchina passa allo step successivo quando **i_start** diventa uguale ad '1', ovvero quando la sequenza è valida.
- **SELECT_CHANNEL**: in questo stato viene salvato in quale canale di uscita instradare il dato prelevato dalla memoria. **Reg1_load** è attivo e l'ingresso **i _w** viene salvato all'interno di **reg_1**. Il reset personalizzato del componente **input_reg_sel**,viene messo a 0. Quest'ultimo ha 2 valori possibili valori nell' uscita da 1 bit **input_reg_sel_out** . Quando è attivo il suo reset personalizzato il valore è sempre 0 mentre se è disattivato inizialmente rimane 0 per poi diventare 1 al ciclo successivo. Fondamentalmente serve alla macchina per capire quando è terminato il prelievo dei bit che rappresentano il canale di uscita e conseguentemente quando deve cambiare stato per passare dal salvataggio del numero del canale in cui instradare il dato (nel reg1) al salvataggio dell'indirizzo di memoria (nel reg2), quindi quando **input_reg_sel_out** vale 1 la macchina passa allo stato successivo.
- **SELECT_MEM_ADDRESS**: è lo stato dove viene prelevata la parte della sequenza che rappresenta l'indirizzo di memoria e salvata nel **reg_2**. La macchina non avanza allo stato successivo finchè **i_start** non torna a 0.
- **FETCH_MEM** : Il valore di **o_mem_enable** diventa 1, questo segnale consente di interagire con la memoria per leggerla.
- **SAVE_MEM_DATA** : in questo stato viene salvato il dato che arriva dalla memoria nel registro **mem_data**.
- **PREP_VALUE4**, **PREP_VALUE5**, **PREP_VALUE6**, **PREP_VALUE7** : sono stati esclusivi, scelti in base all' ingresso che viene dato alla macchina a stati **reg_4_7_selector**. Quest'ultimo è un segnale da 2 bit che rappresenta il canale di uscita a cui instradare il dato prelevato dalla sequenza. Nel singolo stato selezionato si attiva il load del registro corrispondente secondo lo schema seguente (a sinistra della freccia si intende il valore di **reg _4 _7 _selector**) : '00' → **reg4_load** attivato, '01' → **reg5_load** attivato, '10' → **reg6_load** attivato, '11' → **reg7_load** attivato. Quindi viene salvato nel registro il dato proveniente dalla memoria.
- **PROD _OUT** : è lo stato finale . Il segnale **done** viene messo ad 1 contemporaneamente a **mux _selector**, cosìcchè nei canali uscita dei canali **Z0**, **Z1**, **Z2**, **Z3** appaia il valore salvato nei registri intermedi **reg_4**, **reg_5**, **reg_6**, **reg_7**.

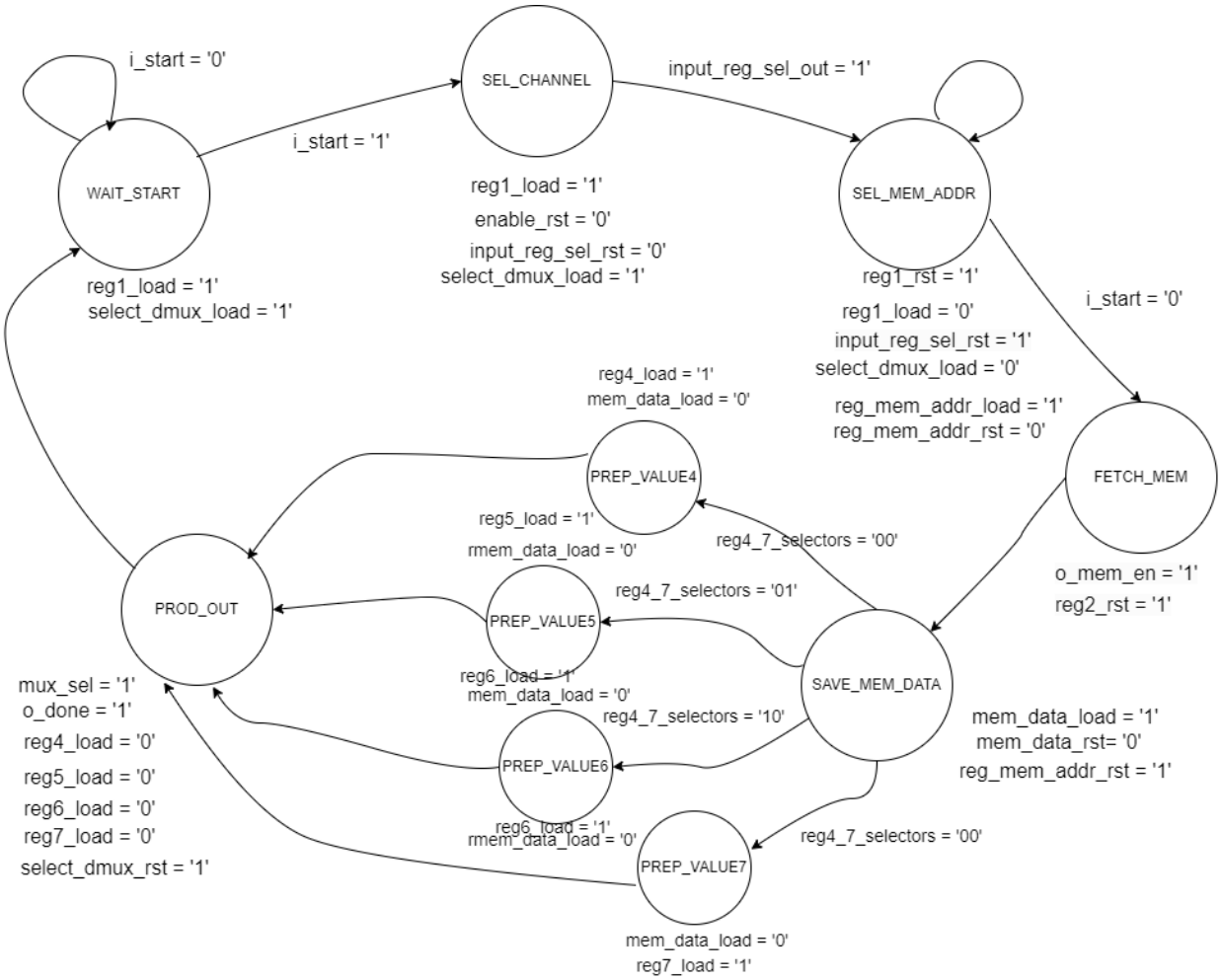
i_start = '0'
SEL_CHANNEL
input_reg_sel_out = '1'
i_start = '1'
WAIT_START
reg1_load = '1'
enable_rst = '0'
input_reg_sel_rst = '0'
select_dmux_load = '1'
SEL_MEM_ADDR
reg1_load = '1'
select_dmux_load = '1'
reg1_rst = '1'
reg1_load = '0'
input_reg_sel_rst = '1'
select_dmux_load = '0'
i_start = '0'
reg_mem_addr_load = '1'
reg_mem_addr_rst = '0'
reg4_load = '1'
mem_data_load = '0'
PREP_VALUE4
FETCH_MEM
reg5_load = '1'
rmem_data_load = '0'
reg4_7_selectors = '00'
o_mem_en = '1'
reg2_rst = '1'
PROD_OUT
reg4_7_selectors = '01'
PREP_VALUE5
SAVE_MEM_DATA
reg6_load = '1'
mem_data_load = '0'
mux_sel = '1'
o_done = '1'
reg4_load = '0'
reg5_load = '0'
reg6_load = '0'
reg7_load = '0'
select_dmux_rst = '1'
reg4_7_selectors = '10'
mem_data_load = '1'
mem_data_rst= '0'
reg_mem_addr_rst = '1'
PREP_VALUE6
reg6_load = '1'
rmem_data_load = '0'
reg4_7_selectors = '00'
PREP_VALUE7
mem_data_load = '0'
reg7_load = '1'

Figure 4: FSM

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	67	0	78600	0.09
LUT as Logic	67	0	78600	0.09
LUT as Memory	0	0	26600	0.00
Slice Registers	82	0	157200	0.05
Register as Flip Flop	82	0	157200	0.05
Register as Latch	0	0	157200	0.00
F7 Muxes	0	0	39300	0.00
F8 Muxes	0	0	19650	0.00

Figure 5: report_utilization

3 Risultati sperimentali

3.1 Sintesi

Il componente passa tutti i testbench forniti sia in presintesi sia in post sintesi utilizzando 82 Flip Flop e 0 Latch come mostrato in figura 5. Dalla tabella 6 si evince che il modulo ha uno slack molto alto di 98,020 ns, ciò significa che soddisfa abbondatamente il requisito di frequenza di clock richiesto.

3.2 Simulazioni

Olte ai testbench forniti ho individuato e testato qualche corner case di interesse.

- Sequenze particolari:
  - 2 bit di canale e 0 bit forniti di indirizzo : per la natura del componente progettato non vi è alcun problema nella simulazione in quanto i bit di indirizzo saranno composti da 16 bit tutti a 0.
  - i_w che assume valori diversi da 0 prima che start valga 1: in questo caso il valore in ingresso i_w viene correttamente ignorato.
- Reset di sistema asincrono :
  1. reset durante il salvataggio della sequenza in ingresso: la FSM torna allo stato iniziale **WAIT_START** (come per ogni caso di reset), se start continua ad essere 1 al ciclo

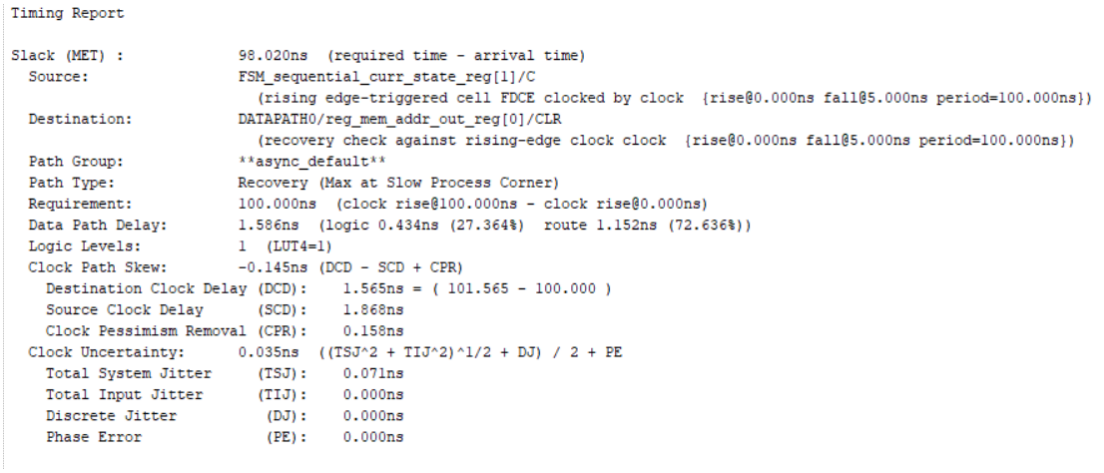

Figure 6: report_timing

successivo a quello di reset sarà ripreso il salvataggio della sequenza. Si presentano poi due sottocasi che spiegherò con esempi:

- **reset durante prelievo canale** : in una sequenza in ingresso "11100" in una normale computazione "11 "rappresenterebbe il canale di uscita e 1000000000000000 l'indirizzo di memoria. Se il reset viene fornito quando sono al secondo bit della sequenza 1 la fsm viene resettata e il terzo bit della sequenza in ingresso rappresenta il primo della nuova. quindi il canale di uscita sarà 10 e l'indirizzo 0000000000000000
- **reset durante prelievo indirizzo** : sequenza in ingresso 001111, in una computazione senza reset 00 è il canale di uscita e 11110000000000000 l'indirizzo di memoria. Se il reset arriva durante il prelievo del terzo bit il quarto bit diventa il primo della nuova e la sequenza diventa : canale 11 e indirizzo 1000000000000000.
- **reset ultimo bit della sequenza** : nel sottocaso particolare in cui il reset avvenga nell'ultimo bit della sequenza al ciclo successivo **i _start** vale 0 , quindi si aspetterà semplicemente la nuova sequenza nello stato di **WAIT_START**.

2. **reset nel momento in cui done dovrebbe diventare '1' e mostrare il contenuto delle uscite** : done rimane a 0 e quindi il valore non viene scritto e le uscite valgono tutte 0.

4 Conclusioni

In conclusione ritengo che il componente progettato rispetti le specifiche, come dimostrato nei testbench si comporta correttamente sia in presintesi che in post sintesi e allo stesso tempo rispetta

ampliamente i costraints di clock non generando alcun latch.