POLITECNICO
MILANO 1863

PROVA FINALE (PROGETTO DI RETI LOGICHE 2021-2022)

Studente: Carmine Faino
Codice persona: 10696112
Matricola: 932495

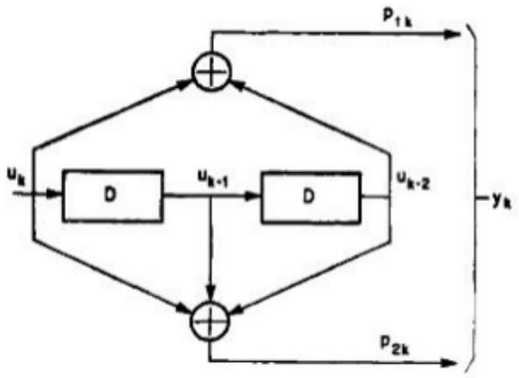
INDICE

1. INTRODUZIONE

1.1 Specifica del progetto

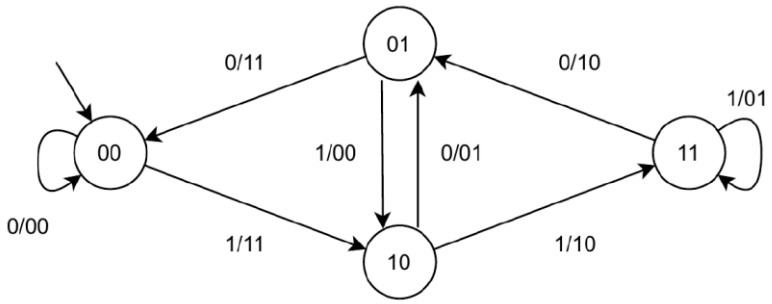
La specifica della “Prova Finale (Progetto di Reti Logiche)” 2021/2022 chiede di implementare un modulo hardware che legga da memoria un certo numero di dati e che poi vada a scrivere sulla memoria stessa un’elaborazione di quest’ultimi.

In particolare viene letto all’indirizzo 0 della memoria il numero di parole N da elaborare, mentre dall’indirizzo 1 all’indirizzo N troveremo le parole codificate su 8 bit. Ogni parola sarà scomposta nei suoi 8 bit che entreranno in ordine in un codificatore convoluzionale così fatto:



Per ogni bit in ingresso avremo 2 bit in uscita, quindi per ogni parola in ingresso avremo 16 bit in uscita. I 16 bit saranno a loro volta divisi in 2 parti da 8 bit ciascuna che verranno scritte in memoria dall’indirizzo 1000 in poi.

Il convolutore è una macchina sequenziale sincrona con il seguente diagramma degli stati:



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_rst : in std_logic;
        i_start : 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:

- il nome del modulo deve essere project_reti_logiche
- i_clk è il segnale di CLOCK in ingresso generato dal TestBench;
- 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_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.

2. ARCHITETTURA

2.2 Descrizione componenti di supporto

Per semplificare la descrizione VHDL del datapath sono stati descritti diversi componenti di supporto, nello specifico:

- Registro W [8 bit] che salva la parola letta
- Registro U [1 bit] che memorizza di volta in volta ogni singolo bit della parola presente in W
- Registro Y [2 bit] che salva l'output generato da U una volta entrato nel convolutore
- 4 registri temp [2 bit] che salveranno a turno Y e che una volta concatenati andranno a comporre gli 8 bit da scrivere in memoria
- Registro Z [8 bit] che contiene la concatenazione dei registri temp
- Registro regContTot [8 bit] che contiene il numero totale di parole
- Registro stato_corr [2 bit] che salva lo stato in cui si trova la macchina a stati

Contatore1 che conta da 0 a 8 e incrementa ogni volta che legge un diverso bit della parola W. Una volta contati i primi 4 bit manda a '1' il segnale quattrobit, mentre se legge tutti gli 8 bit della parola manda a '1' il segnale o_end. Il contatore è composto da:
- MUX1
- Registro regCont1 [3 bit]
- Sommatore sum1

Contatore2 che a partire da 0 incrementa di 1 l'indirizzo di lettura delle parole. Una volta raggiunto il numero presente nel registro regContTot verrà mandato a '1' il segnale i_done. Il contatore è composto da:
- MUX4
- Registro regCont2 [8 bit]
Sommatore sum3

Contatore3 che a partire da 1000 incrementa di 1 l'indirizzo in cui verranno scritte le parole, composto da:
- MUX2
- Registro regAddress [16 bit]
-Sommatore sum2

- MUX3 che in lettura assegna a o_address il valore contenuto in regCont2, mentre in scrittura il valore contenuto in regAddress

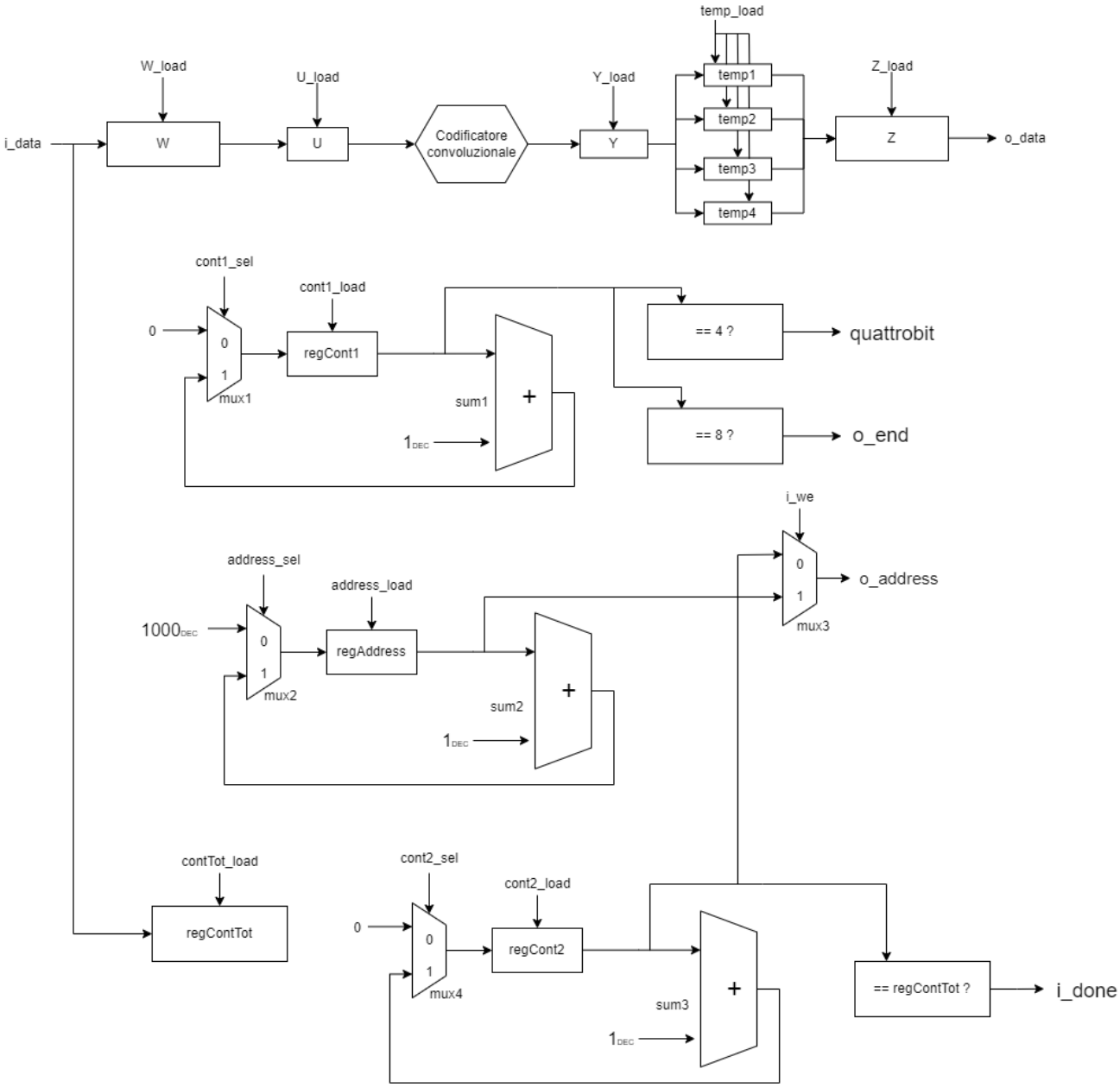
Ogni registro ha il proprio segnale di load che se alzato a '1' caricherà il dato nel registro.
Ogni MUX ha il proprio segnale di sel che indicherà quale dei due input considerare.

2.2 Codice VHDL dei componenti

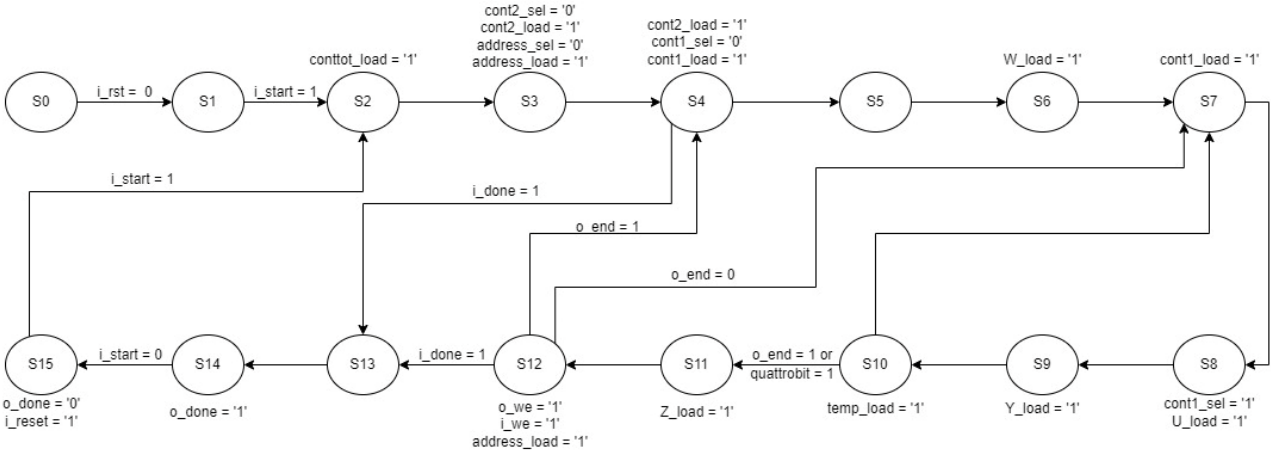
```
architecture Behavioral of project_reti_logiche is
component datapath is
    Port (
        i_clk : in std_logic;
        i_rst : in std_logic;
        i_reset : in std_logic;
        i_data : in std_logic_vector(7 downto 0);
        o_address : out std_logic_vector(15 downto 0);
        o_data : out std_logic_vector (7 downto 0);
        W_load : in std_logic;
        Z_load : in std_logic;
        cont1_sel : in std_logic;
        cont1_load : in std_logic;
        o_end : out std_logic;
        address_sel : in std_logic;
        address_load : in std_logic;
        conttot_load : in std_logic;
        cont2_sel : in std_logic;
        cont2_load : in std_logic;
        U_load : in std_logic;
        Y_load : in std_logic;
        i_we : in std_logic;
        quattrobit : out std_logic;
        temp_load : in std_logic;
        i_done : out std_logic);
end component;

    --datapath signals
    signal W : std_logic_vector (7 downto 0);
    signal Z : std_logic_vector (7 downto 0);
    signal sum1 : std_logic_vector(3 downto 0);
    signal regCont1 : std_logic_vector(3 downto 0);
    signal mux1 : std_logic_vector(3 downto 0);
    signal regAddress : std_logic_vector(15 downto 0);
    signal sum2 : std_logic_vector(15 downto 0);
    signal mux2 : std_logic_vector(15 downto 0);
    signal regCont2 : std_logic_vector(7 downto 0);
    signal sum3 : std_logic_vector(7 downto 0);
    signal mux3 : std_logic_vector(15 downto 0);
    signal mux4 : std_logic_vector(7 downto 0);
    signal regContTot : std_logic_vector(7 downto 0);
    signal U : std_logic;
    signal Y : std_logic_vector(1 downto 0);
    signal temp1 :std_logic_vector(1 downto 0);
    signal temp2 :std_logic_vector(1 downto 0);
    signal temp3 :std_logic_vector(1 downto 0);
    signal temp4 : std_logic_vector(1 downto 0);
    signal stato_corr : std_logic_vector(1 downto 0);
```

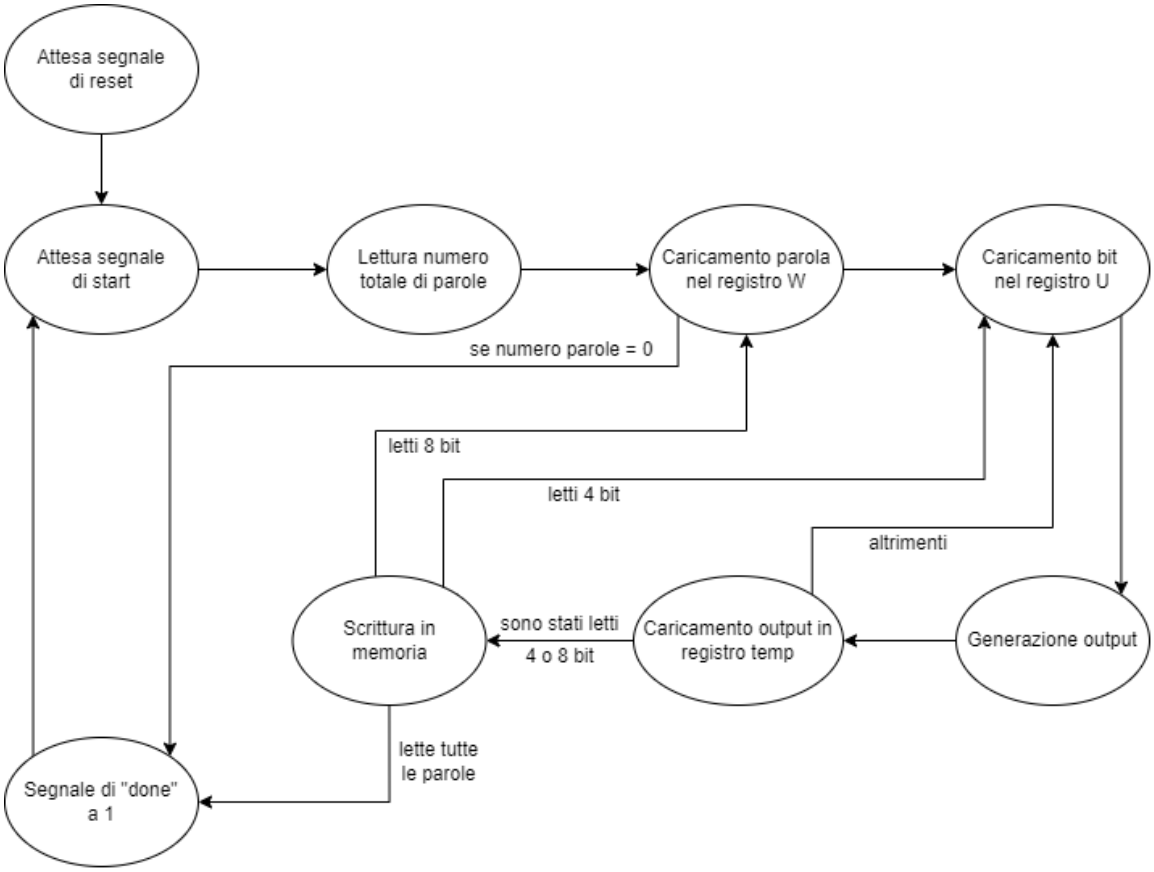
2.3 Rappresentazione del datapath



2.4 Rappresentazione della macchina a stati



-Macchina a stati in una forma semplificata:



3. SIMULAZIONI

3.1 Sintesi

L'operazione di sintesi è stata effettuata per mezzo del software Vivado, considerando come FPGA target il dispositivo Artix-7 FPGA xc7a200tfbg484-1.
Il componente risulta correttamente sintetizzabile e implementabile con l'utilizzo di 83 LUT, 69 Flip Flop. Qui di seguito riportata il report_utilization generato da Vivado:

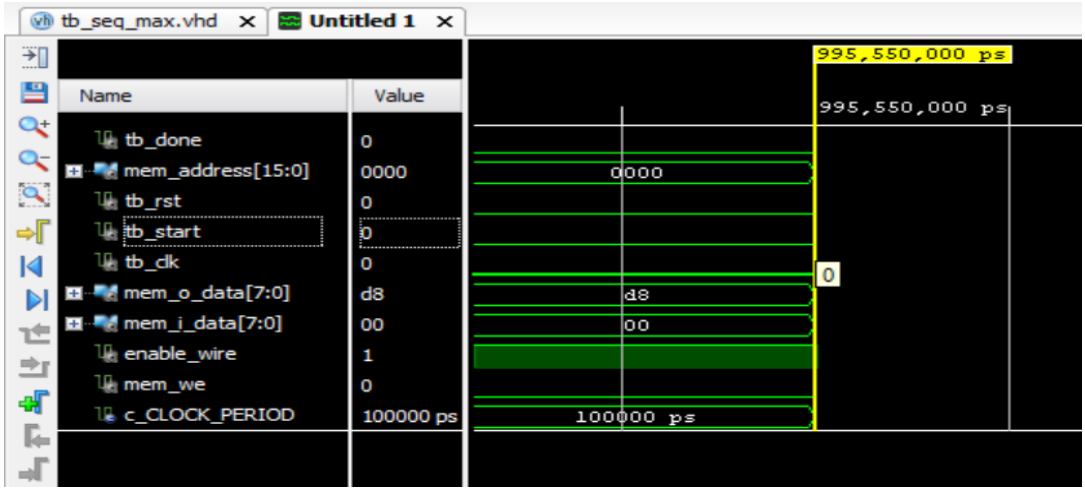
Site Type	Used	Fixed	Available	Util%
Slice LUTs*	83	0	134600	0.06
LUT as Logic	83	0	134600	0.06
LUT as Memory	0	0	46200	0.00
Slice Registers	69	0	269200	0.03
Register as Flip Flop	69	0	269200	0.03
Register as Latch	0	0	269200	0.00
F7 Muxes	1	0	67300	<0.01
F8 Muxes	0	0	33650	0.00

3.2 Test generici e casi limite

Il componente ha superato una moltitudine di simulazione con test generici, ma per verificare la corretta funzionalità sono stati eseguiti alcuni test che andavano a indagare il comportamento in casi particolari

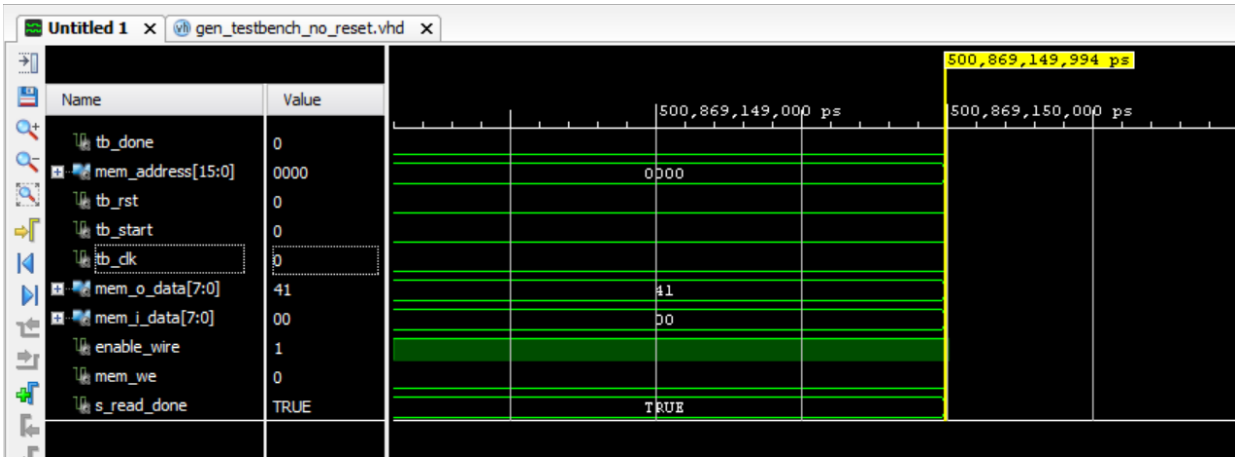
Zero parole da elaborare
Il test effettuato cercava di ingannare il componente dicendo che il numero di parole sarebbe stato 0, ma presentando comunque byte di parole agli indirizzi successivi. Il componente ha risposto correttamente alzando fin da subito il segnale o_done e, quindi, non analizzando alcuna parola.

Massimo numero di parole
Il test effettuato cercava di stressare il componente con una memoria contente il massimo numero di parole ovvero 255. Il componente ha risposto bene allo stress test portando al termine la computazione in 0,99555 Millisecondi.



Tre reset
Il test effettuato consisteva nel far eseguire al componente 4 cicli di lavoro da 3 parole ciascuno, il primo normale e gli altri 3 a seguito di un segnale di RESET. Il componente ha risposto in modo corretto resettando prontamente registri e indirizzi ogni qual volta iniziava un nuovo ciclo.

Stress Test
Per l'ultimo test limite è stato usato uno script in python che ha generato mille test da eseguire in modo sequenziale. Il componente ha dato un output corretto per tutt'e mille, terminando lo stress test in un tempo di circa 0,5 secondi.



3.3 Timing
Dai test è risultato che il componente lavora in modo ottimale anche con un clock di 10 ns, ampliamente sotto i 100 ns richiesti. Come mostra la voce slack del report_timing di Vivado il clock poteva essere ulteriormente ridotto. Qui di seguito il report:

```
(clock clock rise edge)      10.000    10.000 r
                              0.000    10.000 r  i_clk (IN)
net (fo=0)                    0.000    10.000    i_clk
IBUF (Prop_ibuf_I_O)          0.811    10.811 r  i_clk_IBUF_inst/O
net (fo=1, unplaced)          0.760    11.570    i_clk_IBUF
BUFG (Prop_bufg_I_O)          0.091    11.661 r  i_clk_IBUF_BUFG_inst/O
net (fo=69, unplaced)         0.439    12.100    i_clk_IBUF_BUFG
FDCE                                         r  FSM_sequential_cur_state_reg[1]/C
clock pessimism               0.178    12.279
clock uncertainty            -0.035    12.243
FDCE (Setup_fdce_C_D)         0.029    12.272    FSM_sequential_cur_state_reg[1]
-----------------------------------------------
required time                          12.272
arrival time                           -5.802
-----------------------------------------------
slack                                   6.471
```

4. Conclusioni

Questo progetto mi ha permesso di comprendere le mie capacità nell'adattarmi e lavorare in un ambito correlato all'informatica, ma non basato sulla programmazione pura come in altri progetti universitari. Ho potuto mettere in pratica le conoscenze apprese durante il corso di Reti Logiche, come la semplificazione della funzione di uscita della FSM che ho ridotto mediante mappe di Karnaugh.
Durante la realizzazione del progetto ho dovuto affrontare più di un problema, come per esempio la rimozione di un latch che avrebbe potuto generare problemi. Per far ciò mi sono armato di pazienza e tramite debug l'ho individuato e prontamente rimosso.
Mi ritengo soddisfatto del prodotto finale che ha superato ogni test al quale è stato sottoposto anche con ottimi risultati.