Prova Finale Reti Logiche a.a. 2020/2021

Prof. Fabio Salice

Beatrice Insalata (10708628-936956)
Teka Kimbi Ntimanputu (10673197-932486)

Consegna del 15 Maggio 2022

Indice

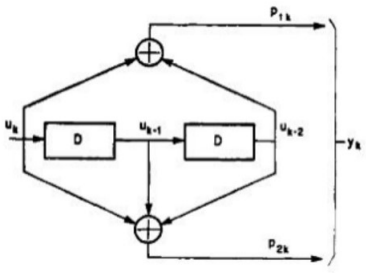
1 Introduzione

1.1 Descrizione del progetto

Scopo della prova finale di reti logiche 2021/2022 è l'implementazione di un modulo hardware tramite descrizione in VHDL. Il componente segue lo schema di un codificatore convoluzionale con tasso di trasmissione $\frac{1}{2}$, ovvero due bit di uscita per ogni bit d'ingresso. Il convolutore lavora su un flusso continuo in ingresso e uscita: un numero W di parole su 8 bit che sarà restituito in sequenza di Z parole su 8 bit. Il modulo realizza quindi una serializzazione dell'input generando un flusso continuo fatto da singoli bit, che concatenati in modo alternato e raggruppati su 8 bit vanno a comporre il flusso Z.

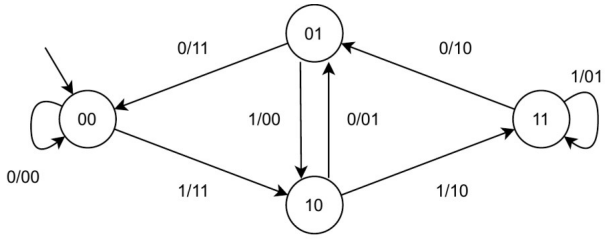
1.2 Struttura del Convolutore

Rappresentazione modello del convolutore:



Codificatore convoluzionale con tasso di trasmissione $\frac{1}{2}$.

Rappresentazione tramite macchina di Mealy:



1.3 Struttura dell'input/output

Il convolutore riceve in ingresso dati da una memoria con indirizzamento al byte. La quantità di parole da memorizzare è all'indirizzo 0, mentre dall'1 in poi si

ha il flusso da elaborare, la cui linghezza è di massimo 255 byte.
In uscita, le parole processate sono scritte partendo dall'indirizzo 1000.

1.4 Interfaccia del componente

Il componente implementa la seguenta 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;
```

- **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 da 8 bit) che arriva dalla memoria in seguito ad una richiesta di lettura;
- **o_address** è il segnale (vettore da 8 bit) di uscita che manda l'indirizzo alla memoria in lettura e scrittura;
- **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 da 16 bit) di uscita dal componente verso la memoria.

1.5 Note ulteriori sulla specifica

I segnali `START` e `DONE`, insieme a quello di `RESET`, regolano il funzionamento del modulo.

1. L'elaborazione inizia quando `START` viene portato a 1, finché `DONE` non è portato alto come segnalazione della avvenuta elaborazione per il singolo flusso. Un nuovo `START` non può essere dato se `DONE` non è a 0: in questo caso, si inizierà nuovamente la codifica;

2. Ad ogni nuova elaborazione, il convolutore torna nello stato 00. La quantità di parole da codificare e l'indirizzo d'uscita sono impostati come 0 e 1000;

3. La prima elaborazione richiede che `RESET` sia dato al modulo, per le successive basta la terminazione della precedente.

2 Soluzione Proposta

Nella realizzazione del componente si è scelto di utilizzare una macchina a stati che ben si adatta al funzionamento richiesto.
E' stato deciso di implementare un accesso sul singolo bit della parola memorizzata al fine di svolgere l'analisi direttamente su di esso, tenendo conto di un indice per individuare la posizione attuale nel byte d'ingresso.
Lo stesso indice viene utilizzato per scrivere nella giusta posizione della parola in output, che poi verrà messa al corretto indirizzo di memoria, aumentato progressivamente.
La computazione parte dalla lettura del numero di parole appartenenti al flusso, per poi procedere con la memorizzazione degli ingressi e convoluzione.

2.1 Segnali utilizzati e Stati della Macchina

La soluzione utilizza diversi segnali ausiliari interni:

```
signal first,second:std_logic;
signal output_index :  integer 7 downto 0:= 7;
signal words:std_logic_vector(7 downto 0);
signal in_address:  std_logic_vector(15 downto 0);
signal out_address:  std_logic_vector(15 downto 0);
signal input_word:  std_logic_vector(7 downto 0);
signal output_word:  std_logic_vector(7 downto 0);
signal state:  integer range 3 downto 0;
signal temp_index :  integer range 7 downto 0 := 7;
signal temp_in :  std_logic_vector(15 downto 0);
signal temp_out :  std_logic_vector(15 downto 0);
signal temp_words :  std_logic_vector(7 downto 0);
signal flux:  std_logic;
```

- I segnali `first, second` contengono i valori generati dalla convoluzione sul bit in ingresso, che verranno successivamente assegnati nella corretta posizione all'interno della parola d'uscita.
- Il segnale `output_index` è utilizzato come contatore per individuare dove accedere all'interno del byte in ingresso e in quale posizione scrivere all'interno del byte in uscita;
- Il segnale `words` è un vettore che memorizza il numero di parole appartenenti al flusso, usato anche per stabilirne la terminazione;
- I segnali `in_address, out_address` tengono rispettivamente conto dell'indirizzo da cui leggere la parola in input e dove scrivere la parola elaborata in output;

- I segnali **input_word, output_word** contengono rispettivamente il valore della parola letta in ingresso e della parola risultata dalla convoluzione e da scrivere in uscita;
- Il segnale **state** memorizza il corretto stato della macchina convolutrice per potervi rientrare in seguito ad una scrittura;
- I segnali **temp_index, temp_in, temp_out, temp_words** sono segnali utili nell'assegnazione dei corretti valori a ciascuno dei precedenti;

INDICE	NOME	AZIONE
1	RESET	Stato di inizializzazione dei segnali ai valori necessari per il corretto funzionamento del programma, vi si ritorna all'inizio di ogni nuovo flusso e quando il segnale RST viene portato alto.
2	START	Stato che fa partire l'esecuzione, abilitando la lettura da memoria con o_en = '1'.
3	WAIT_WORD	Stato che legge da memoria il numero di parole appartenenti al flusso.
4	READ_WORD	Stato controlla se la parola letta è 0.
5	WAIT_DATA	Stato che attende un ciclo di clock per leggere la prima parola del flusso.
6	READ_DATA	Stato in cui si legge la parola da elaborare e si inizializza output_index.
7	ENTER	Stato che assegna il corretto ingresso nella macchina convolutrice e disabilita la lettura da memoria.
CONVOLUZIONE	S0,S1,S2,S3	Stati della macchina convolutrice.
8	HANDLER	Stato che assegna il risultato della convoluzione alla corretta posizione della parola di output, sfruttando il valore di output_index.
9	DECREASER	Stato che considerando l'output_index attuale assegna al valore temporaneo il giusto indice successivo e stabilisce se procedere nella computazione o andare in stato di WRITE. Può anche modificare i valori del numero di parole elaborate e dell'indirizzo di lettura quando arriva alla fine di un byte.

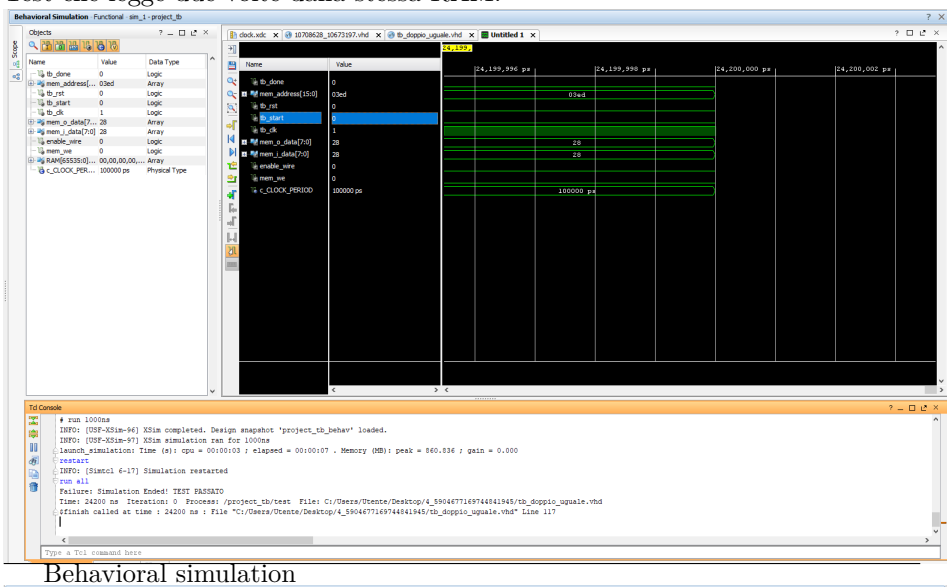
10	STATE_SWITCH	Stato che porta nella giusta posizione all'interno della macchina convolutrice in base all'attuale valore del segnale state.
11	WRITE	Stato che si occupa di impostare l'indirizzo di uscita e scrivere la parola in uscita, aumentando il successivo indirizzo di output.
12	WAIT_OUT	Stato che controlla in quale stato della macchina fare proseguire l'elaborazione, in base al numero di parole rimaste e alla posizione dell'output_index.
13	DONE	Stato che determina il raggiungimento della fine di un flusso.

3 Testing

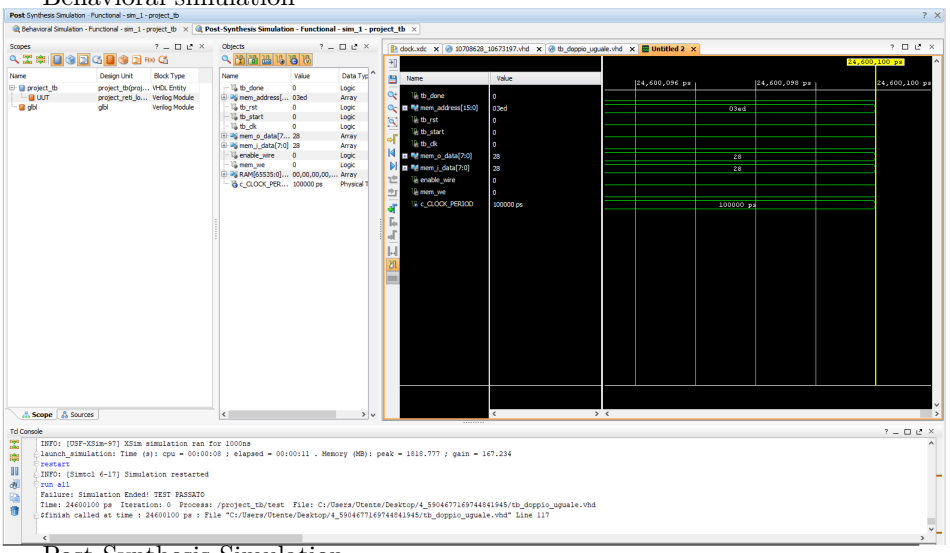
Il componente supera il TestBench fornito dal docente in Behavioral e Post-Synthesis. E' stato inoltre sottoposto ad un ulteriore testing sui file presenti su Git e sui 2000 test autogenerati, passandoli tutti. Si allegano di seguito alcuni risultati delle prove eseguite, riportando quelli con i casi più critici. In generale, il modulo risulta rispettare ogni constraint assegnato e concludere correttamente la computazione generando il risultato atteso.

3.1 Test Doppio Uguale

Test che legge due volte dalla stessa RAM.



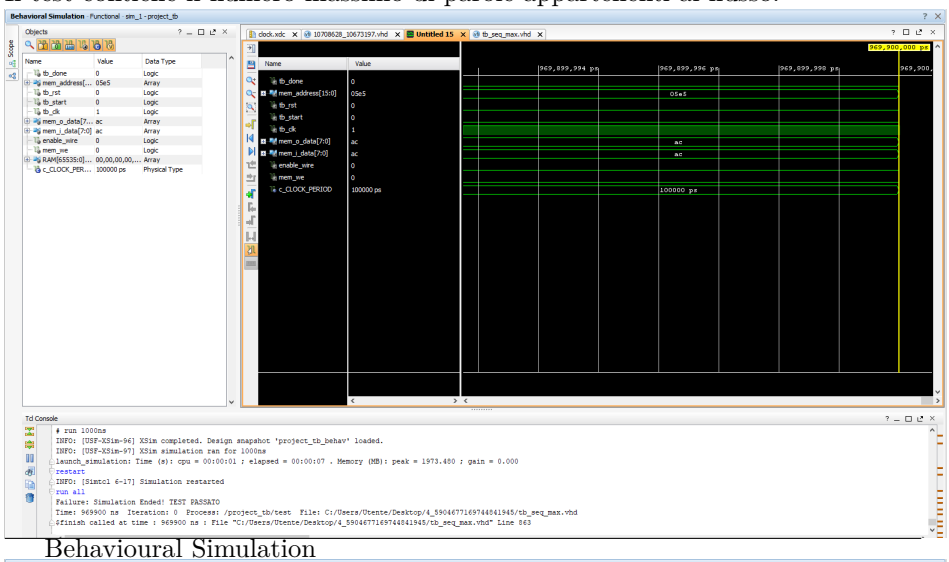
Behavioral simulation



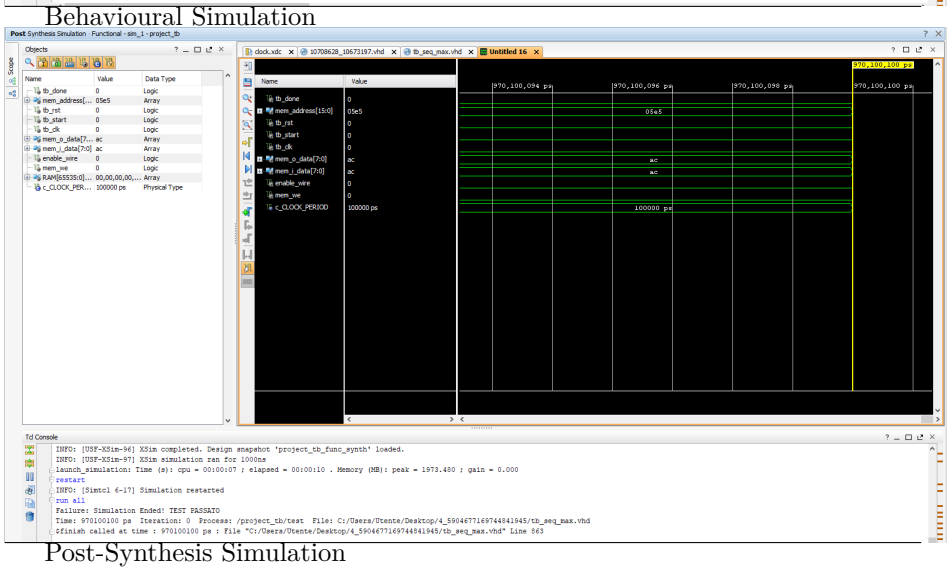
Post-Synthesis Simulation

3.2 Test Sequenza Massima

Il test contiene il numero massimo di parole appartenenti al flusso.



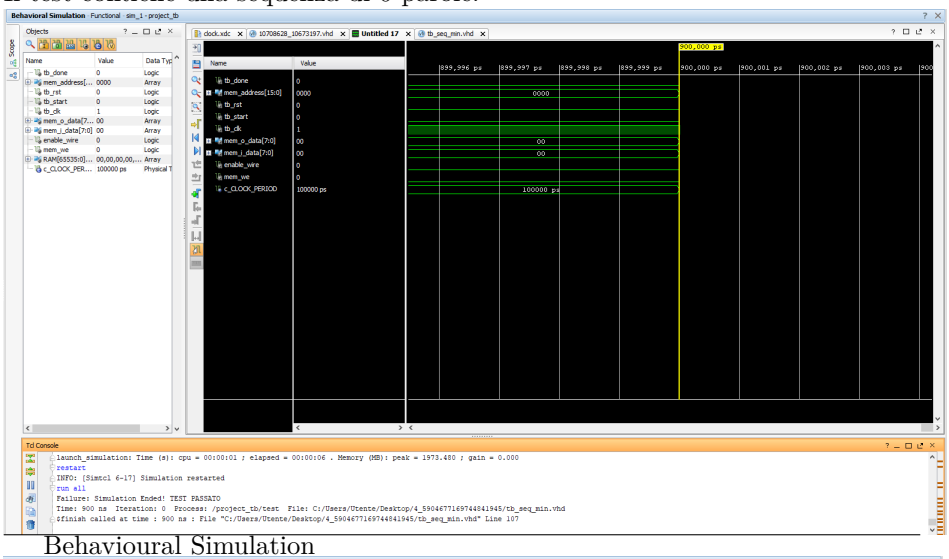
Behavioural Simulation



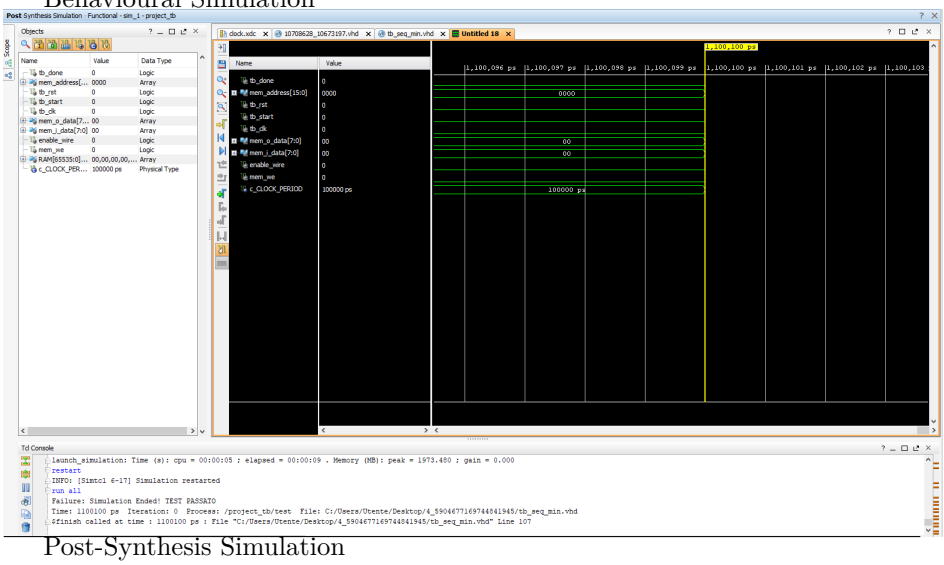
Post-Synthesis Simulation

3.3 Test Sequenza Minima

Il test contiene una sequenza di 0 parole.



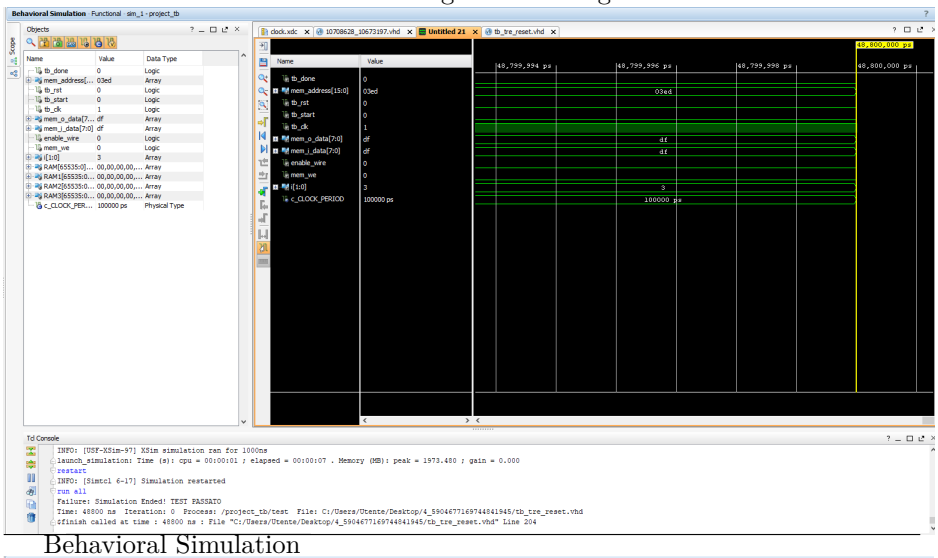
Behavioural Simulation



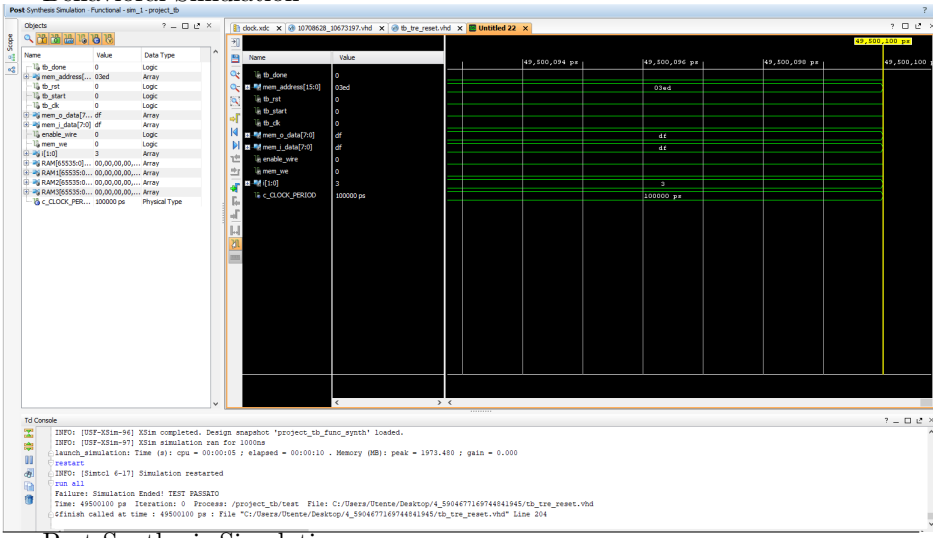
Post-Synthesis Simulation

3.4 Tre Reset

Il test lavora su tre RAM differenti e gestisce un segnale di RESET



Behavioral Simulation



Post-Synthesis Simulation

3.5 Test Generati

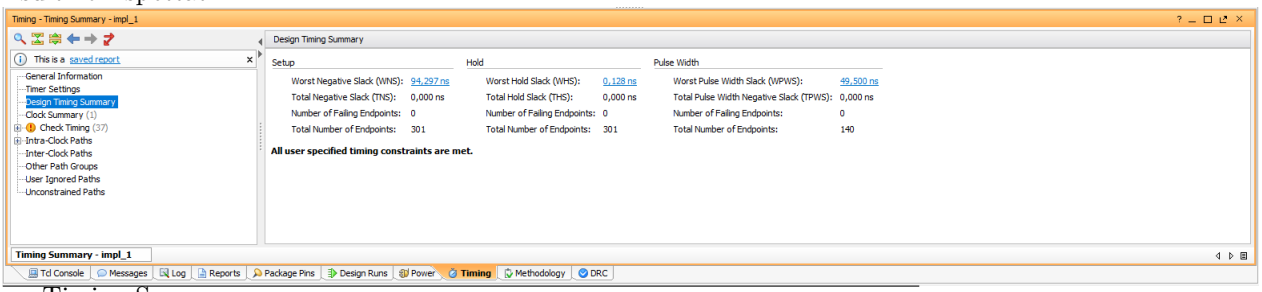
Tutti i test della cartella TestBench risultano superati. Sono stati inoltre da noi generati 2000 casi che vengono anch'essi passati dal componente. Si riporta un estratto del risultato.

```
1975) PASSATO
1976) PASSATO
1977) PASSATO
1978) PASSATO
1979) PASSATO
1980) PASSATO
1981) PASSATO
1982) PASSATO
1983) PASSATO
1984) PASSATO
1985) PASSATO
1986) PASSATO
1987) PASSATO
1988) PASSATO
1989) PASSATO
1990) PASSATO
1991) PASSATO
1992) PASSATO
1993) PASSATO
1994) PASSATO
1995) PASSATO
1996) PASSATO
1997) PASSATO
1998) PASSATO
1999) PASSATO
2000) PASSATO
```

4 Implementazione e Design

4.1 Design Timing Summary

Il Design Timing Summary mostra come tutti i constraints assegnati dall'utente risultino rispettati.



Timing Summary

4.2 Report Utilization

Il Report Utilization permette di verificare l'assenza di Latch all'interno del componente sintetizzato.

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs	145	0	134600	0.11
LUT as Logic	145	0	134600	0.11
LUT as Memory	0	0	46200	0.00
Slice Registers	139	0	269200	0.05
Register as Flip Flop	139	0	269200	0.05
Register as Latch	0	0	269200	0.00
F7 Muxes	1	0	67300	<0.01
F8 Muxes	0	0	33650	0.00

Slice

5 Conclusione

Il componente implementato consente di eseguire correttamente ogni test assegnato, rispettando le specifiche assegnate dal documento fornito. Inoltre non oltrepassa i limiti imposti e restituisce il corretto risultato per ogni test su cui è stato provato. Risulta inoltre privo di Latch e non crea problemi nei diversi corner case proposti.