PROVA FINALE (PROGETTO DI RETI LOGICHE)

Prof. Gianluca Palermo – Anno 2021/2022

Aurora Tesin (Codice persona: 10652280 – Matricola 913494)

INDICE

1 INTRODUZIONE

1.1 Scopo del progetto

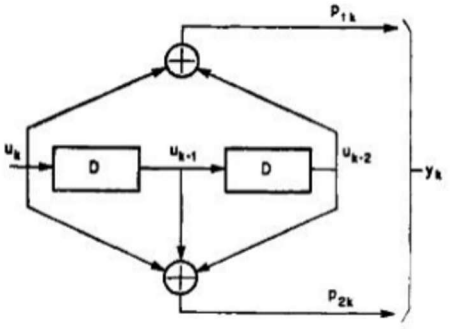
Lo scopo di questo progetto è quello di descrivere in VHDL un componente hardware che applica ad un flusso di bit il **codice convoluzionale** $^1/_2$, ovvero ogni bit ricevuto in ingresso viene codificato con due bit in uscita.

1.2 Specifiche generali

il componente riceverà in input un numero di parole da codificare e in seguito tutte le singole parole da elaborare, mentre fornirà le parole elaborate come output; poiché il tasso di trasmissione è 1/2, ad ogni parola in ingresso ne corrisponderanno due in uscita.

Tutte le parole in input verranno serializzate e trasformate in una sequenza di bit e processate attraverso il *codificatore convoluzionale,* che produrrà come output un flusso di bit, lungo il doppio rispetto a quello in ingresso, che sarà successivamente suddiviso in byte per creare la sequenza di parole in output.

La codifica del flusso di bit dovrà avvenire secondo il seguente schema:



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

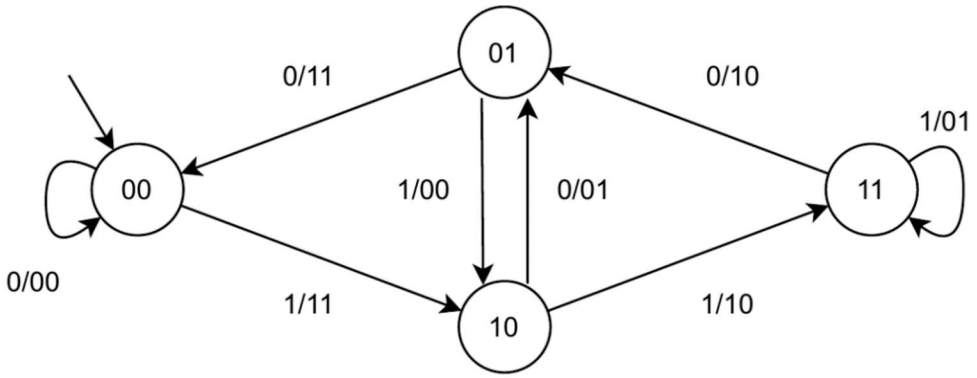
Dove viene rappresentato con ***U*** il flusso di bit in ingresso, con $\boldsymbol{u_{k-2}}$ il bit in ingresso al tempo t - 2, $\boldsymbol{u_{k-1}}$ il bit in ingresso al tempo t - 1 e $\boldsymbol{u_k}$ il bit in ingresso al tempo t.

Lo stream d'uscita ***Y*** è ottenuto come concatenamento alternato $(\boldsymbol{p_{1k}}.\boldsymbol{p_{2k}})$ dei due bit in uscita dal convolutore, ricavati nel seguente modo:
$\boldsymbol{p_{1k}}$ come: $\boldsymbol{u_k}$ **xor** $\boldsymbol{u_{k-2}}$

$\boldsymbol{p_{2k}}$ come: $\boldsymbol{u_k}$ **xor** $\boldsymbol{u_{k-2}}$ **xor** u_{k-1}

La sequenza di parole in uscita ***Z*** sarà la parallelizzazione, in 8 bit del flusso ***Y***.

Il *convolutore* è una macchina sequenziale sincrona con un clock globale e un segnale di reset, rappresentabile con il seguente diagramma degli stati che ha nel suo 00 lo stato di reset, con transazioni identificate da $\boldsymbol{u_k/p_{1k}p_{2k}}$ **:**



1.1.1 esempio di codifica

Un esempio di funzionamento:

BYTE IN INGRESSO = 10100010

Il primo bit a sinistra, ovvero il più significativo, è il primo bit da processare. Applicando l'algoritmo convoluzionale si ottiene la seguente serie di coppie di bit:

T	0	1	2	3	4	5	6	7
Uk	1	0	1	0	0	0	1	0
P1k	1	0	0	0	1	0	1	0
P2k	1	1	0	1	1	0	1	1

Il concatenamento dei valori p1k e p2k per produrre la sequenza di uscita segue il seguente schema:

p1k al tempo t=0, p2k al tempo t=0, p1k al tempo t=1, p2k al tempo t=1 … e così via, producendo la sequenza in uscita BYTE IN USCITA = 11010001 11001101.

1.3 Struttura della memoria

Le parole sono rappresentate in byte, ovvero 8 bit ciascuna, con un massimo di 255 parole in input e 510 in output.

I dati in input e output verranno letti e scritti su una memoria di tipo RAM con indirizzamento al byte e bus indirizzi di 16 bit, suddivise nel seguente modo:

- La cella ***(0)*** conterrà il numero di byte da codificare.
- Le celle comprese tra ***(1)*** e ***(num_byte)*** conterranno tutte le parole da codificare.
- Le celle comprese tra ***(1000)*** e ***(1000 + num_byte*2)*** conterranno tutte le parole codificate secondo il codice convoluzionale.
- Le altre celle conterranno il valore 0.

Schema memoria:

Indirizzo	**Valore**
0	Numero byte da elaborare – num_byte
1	Primo byte
….	….
num_byte	Ultimo byte da elaborare
….	0
….	….
1000	Primo byte codificato
1001	Secondo byte codificato
….	….
2*num_byte-1	Ultimo byte codificato
…	0

Il progetto non comprende la progettazione della memoria, quella fornita è rappresentata dal protocollo *"Single-Port Block RAM Write-First Mode"*, derivata dalla user guide di VIVADO.

1.4 Interfaccia componente

Il componente avrà la seguente interfaccia:

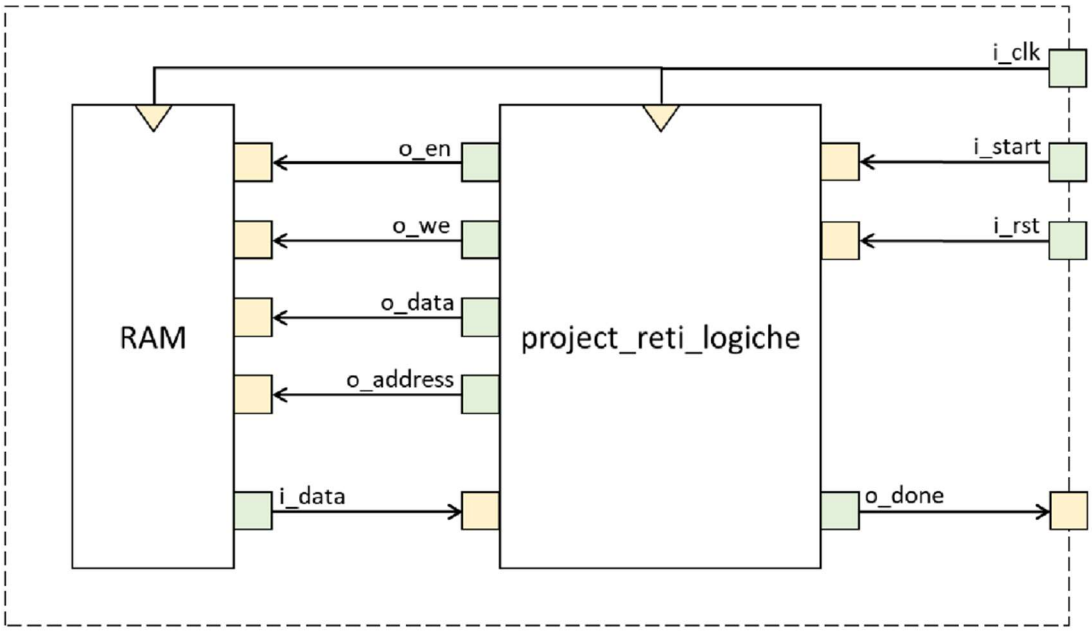
```
entity project_reti_logiche is
      port (
            i_clk     : in std_logic;
            i_rst     : in std_logic;
            i_start   : in std_logic;
            i_data    : in std_logic_vector(7 downto 0);
            o_address : out std_logic_vector(15 downto 0);
            o_done    : 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.

1.5 Specifiche di funzionamento

Il componente è così collegato alla memoria:



All'avvio, prima dell'inizio della prima computazione, al componente verrà inviato il segnale di **i_rst**; esso rimarrà poi in attesa che il segnale di i_**start** venga posto ad 1 per iniziare la computazione, e rimarrà ad 1 per tutto il processo di elaborazione. Al termine della computazione il componente terminerà portando il segnale **o_done** a 1, che rimarrà ad 1 fino a che il segnale di start non tornerà 0.

Il componente aspetterà che il segnale **i_start** ritorni a 0 prima di tornare nel suo stato iniziale da cui potrà iniziare una nuova elaborazione; il modulo è progettato per poter codificare più flussi uno dopo l'altro; perciò, le computazioni successive alla prima non richiederanno il reset del modulo, basterà solamente che **i_start** venga posto nuovamente ad 1.

Inoltre, il componente descritto deve funzionare correttamente con un periodo di clock di almeno **100 ns**.

1.6 Esempi

Esempio3: (Sequenza lunghezza 3)
W: 01110000 10100100 00101101
Z: 00111001 10110000 11010001 11110111 00001101 00101000

INDIRIZZO MEMORIA	VALORE	COMMENTO
0	3	\\ Byte lunghezza sequenza di ingresso
1	112	\\ primo Byte sequenza da codificare
2	164	
3	45	
[...]		
1000	57	\\ primo Byte sequenza di uscita
1001	176	
1002	209	
1003	247	
1004	13	
1005	40	

Questo è un esempio del contenuto che si troverà in memoria alla fine di un'elaborazione di una sequenza in ingresso di 3 parole.

Si noti che i valori fino all'indirizzo 3 sono i dati in input scritti prima dell'elaborazione e rimarranno invariati, mentre i valori dall'indirizzo 1000 fino all'indirizzo (1000 + 3*2 – 1) sono le 6 parole in uscita che il componente avrà scritto in memoria prima del termine della sua elaborazione.

2. ARCHITETTURA

2.1 Scelte progettuali

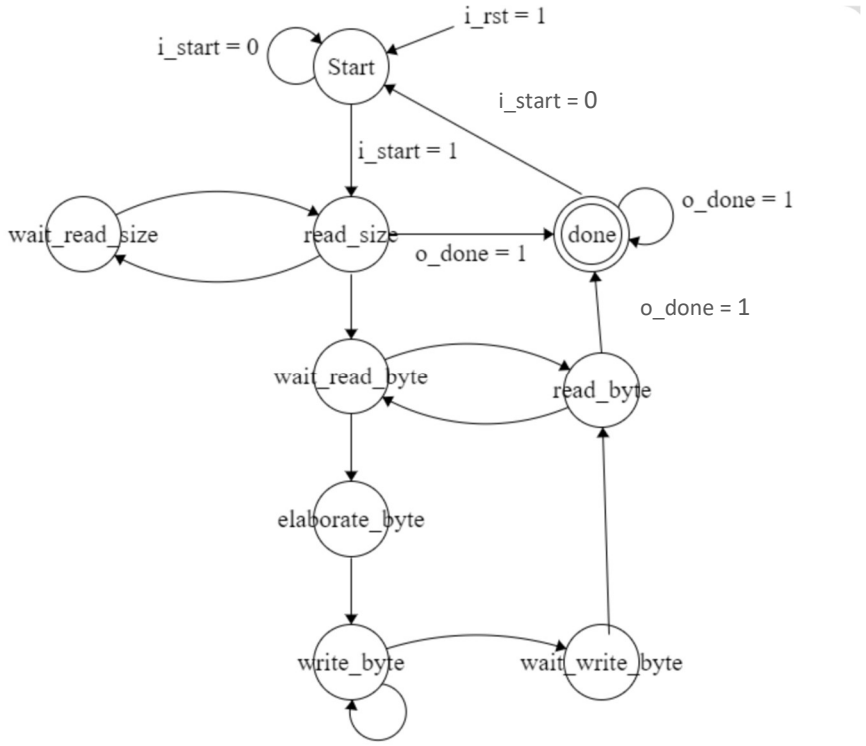
La descrizione del componente in VHDL si basa su una macchina a stati finiti composta da 9 stati. Si è optato per una soluzione con un singolo process che porta la FSM nei vari stati con il supporto di alcuni registri e segnali necessari alla computazione.

La macchina a stati è sensibile al fronte di salita del clock; ad ogni evento di salita, la macchina eseguirà le operazioni dello stato in cui si trova e si preparerà al cambio di stato: il cambiamento di stato e l'aggiornamento dei segnali verrà eseguito al termine dell'esecuzione del processo.

Il segnale di reset è modellizzato come un segnale asincrono: questo permette di gestire al meglio l'inizializzazione dei segnali ed il caso in cui venga portato alto durante una computazione già avviata, senza avere inconsistenza dei dati dell'elaborazione precedente.

In seguito, viene presentata sia la descrizione della macchina a stati sia la descrizione dei registri che sono stati utilizzati.

2.2 Stati della macchina



START

Stato iniziale: dopo il primo reset la macchina resta in questo stato e rimane in attesa fino a che ***i_start*** viene portato a 1. Ogni volta che il segnale ***i_rst*** viene posto a 1 si ritorna in questo stato.

READ_SIZE

Stato in cui viene richiesto il numero di byte da elaborare e settato il registro last_byte_address contenente l'indirizzo dell'ultimo byte in input. Si fa inoltre un controllo che il numero di byte da elaborare sia diverso da zero, altrimenti si alza il segnale ***o_done***.

WAIT_READ_SIZE

Stato in cui si attende la risposta della memoria in seguito alla richiesta del numero di byte e dell'indirizzo dell'ultimo byte in input.

READ_BYTE

Stato in cui si legge dalla memoria un singolo byte. Questo viene preso dall'indirizzo in memoria assegnato ad ***o_address***.

WAIT_READ_BYTE

Stato in cui si attende la risposta dalla memoria in seguito ad una lettura di un byte.

ELABORATE_BYTE

Stato in cui viene elaborato il byte in input il cui valore è presente in ***i_data***, producendo i due byte in output.

WRITE_BYTE

Stato in cui vengono scritte le due parole in output in memoria. Questo viene fatto ritornando in questo stato una seconda volta, in quanto per ogni byte in ingresso, bisogna scriverne due in memoria.

WAIT_WRITE_BYTE

Stato in cui si attende la risposta della memoria in seguito alla richiesta di scrittura, viene utilizzato anche per aggiornare il registro contenete l'indirizzo a cui si andrà a scrivere il prossimo byte in output.

DONE

Stato in cui si attende che ***i_start*** venga posto ad 1 per poter riportare ***o_done*** a 0 e tornare nello stato inziale (START).

2.3 Registri

Lo stato interno del componente è definito da:

```
signal has_byte_number : boolean := false;
signal set_address     : boolean := false;
signal done_read : boolean := false;
signal write2: boolean := false;


signal last_byte_address : std_logic_vector(15 downto 0) := (others => '0');
signal current_byte      : std_logic_vector(15 downto 0) := (others => '0');
signal out_address: std_logic_vector(15 downto 0 ) := "0000001111101000";
signal outdata: std_logic_vector ( 15 downto 0 ) := (others => '0');
```

In particolare:

- ***has_byte_number*** indica se il numero di parole è stato letto.
- ***set_address*** indica se l'ultimo indirizzo è stato salvato.
- ***done_read*** indica se la macchina ha finito la lettura dei byte.
- ***write2*** flag per la scrittura dei due byte in memoria.
- ***last_byte_address*** indica l'indirizzo dell'ultimo byte da leggere.
- ***current_byte*** registro che indica l'indirizzo del byte in elaborazione.
- ***out_address*** registro che indica a che indirizzo scrivere un byte in output.
- ***outdata*** registro contenente i due byte da scrivere in memoria.

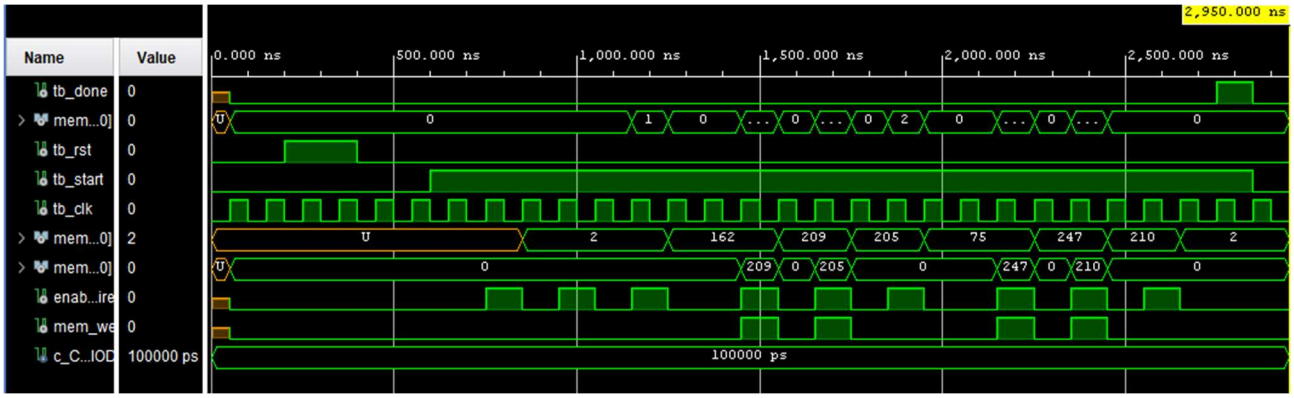
3. RISULTATI SPERIMENTALI

3.1 Simulazione

Per verificare il corretto funzionamento del componente sintetizzato, dopo averlo testato con il test bench di esempio, ho definito altri test che verificano il corretto comportamento del componente durante i corner case.

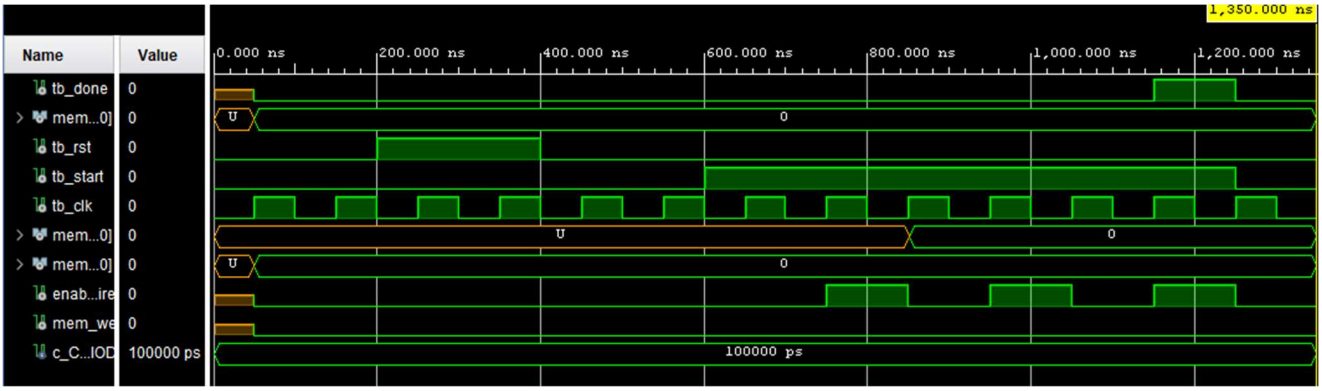
In seguito, verrà mostrata una breve descrizione dei test con le rispettive immagini dell'andamento dei segnali durante la simulazione.

3.1.1 Test d'esempio fornito dal docente



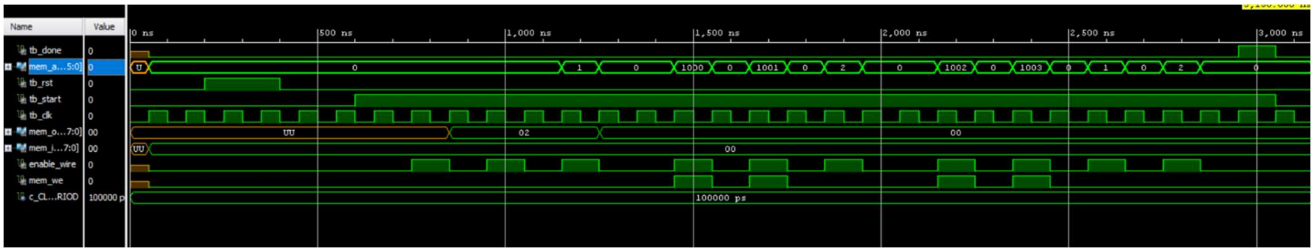
3.1.2 Senza Byte

Test con numero di byte pari a zero; il test verifica che il componente non scriva niente in memoria.



3.1.3 Solo byte di valore 0

Test composto solo da byte di valore 0.

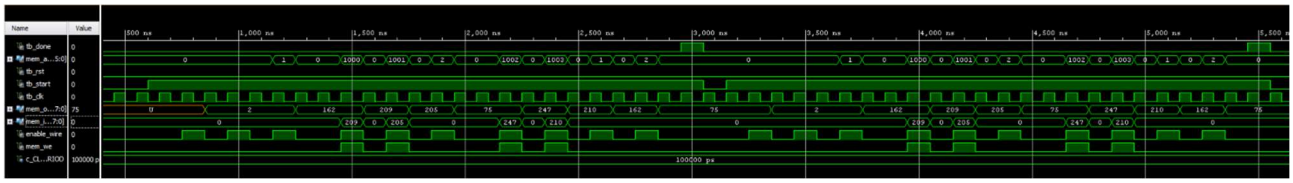


3.1.3 Numero massimo di byte

Elaborazione con la dimensione massima di 255 byte in ingresso. Il test verifica che il componente scriva in memoria 510 byte elaborati correttamente (in allegato solo la parte finale, poiché non ci stava).

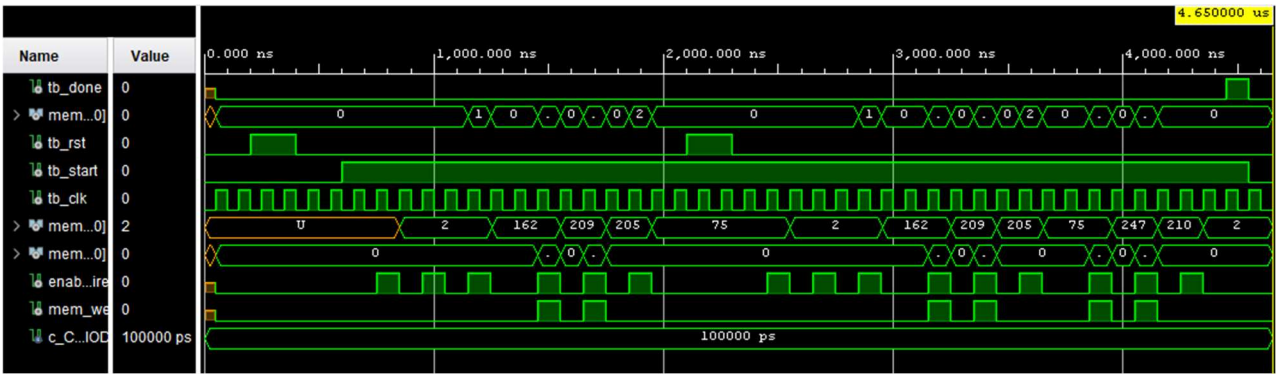
3.1.4 Elaborazione di più flussi

Il test verifica la corretta sincronizzazione dei segnali di i_rst, i_start e o_done e la capacità del componente di codificare un flusso dopo l'altro senza la necessità di resettare il componente.



3.1.5 Reset durante un’elaborazione

Il test verifica che il componente funzioni correttamente anche in caso di reset durante una computazione, facendo ritornare la macchina nel suo stato iniziale.



3.2 Sintesi e implementazione

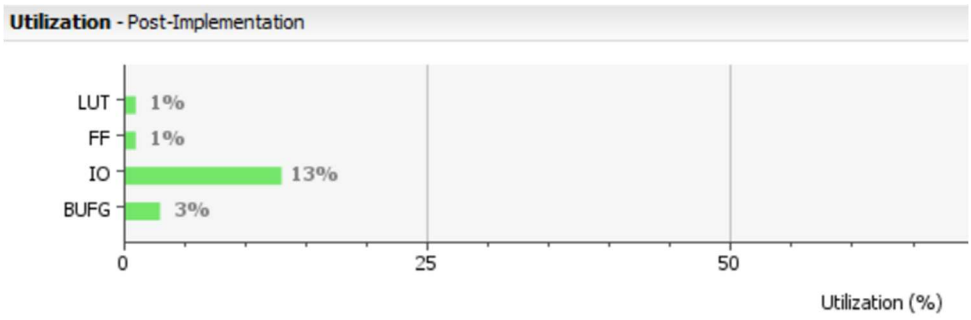
Il componente è correttamente sintetizzabile e implementabile con un totale di 148 LUT e 109 FF.

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	... Failed Routes	LUT	FF	BRAM	URAM	PCIe %
synth_1	constrs_1	synth_design Complete!							148	109	0	0	0.000
impl_2	constrs_1	route_design Complete!	NA	NA	NA	NA	NA	... 0	96	109	0	0	0.000

L'operazione di sintesi è stata effettuata, utilizzando il software Vivado, considerando come FPGA target il dispositivo xc7a200tfbg484-1.

Di seguito sono riportati i componenti della FPGA utilizzati e il grafico post-Implementation:

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	148	0	41000	0.36
LUT as Logic	148	0	41000	0.36
LUT as Memory	0	0	13400	0.00
Slice Registers	109	0	82000	0.13
Register as Flip Flop	109	0	82000	0.13
Register as Latch	0	0	82000	0.00
F7 Muxes	0	0	20500	0.00
F8 Muxes	0	0	10250	0.00



4. CONCLUSIONI

Il componente supera correttamente tutti i test specificati nelle tre simulazioni: *Behavioral*, *Post-Synthesis Functional* e *Post-Synthesis Timing*.

Inizialmente si era pensato di creare un'altra FSM a parte per l'elaborazione del flusso di bit, ma veniva troppo complesso, dunque si è optato per uno stato, elabora_byte, che serializza un byte alla volta andando ad elaborare il flusso di bit prodotto con una serie di xor creando così i due byte in uscita salvati in outdata, tenendo traccia degli ultimi due bit con le variabili last1 e last2 e temp.

Ho riportato il confronto tra i tempi di simulazione nel caso dell'elaborazione dei i casi limite da 0 byte e da 255 byte in input.

0 byte

Failure: Simulation Ended! TEST PASSATO (ENCODE_EXAMPLE)

Time: 1453001 ps Iteration: 0 Process: /project_tb/test

$finish called at time : 1453001 ps

255 byte

Failure: Simulation Ended! TEST PASSATO (ENCODE_EXAMPLE)

Time: 230953001 ps Iteration: 0 Process: /project_tb/test File:

$finish called at time : 230953001 ps