Prova Finale

(Progetto di Reti Logiche)

Prof. Fabio Salice

A.A. 2021/2022

Letizia Grassi

Sommario

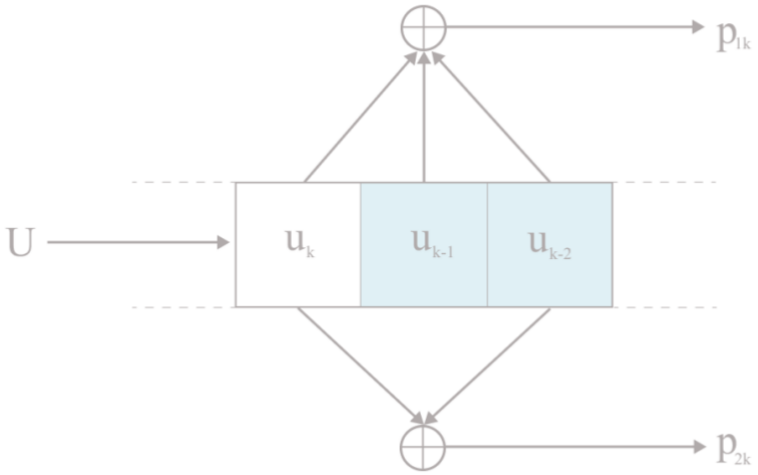
1. Introduzione

Lo scopo del progetto è di realizzare in linguaggio VHDL un componente HW che descriva il comportamento di un *codificatore convoluzionale con tasso di trasmissione* $\frac{1}{2}$ e gestisca la traduzione di un flusso U di bit in ingresso in un flusso Y in uscita mediante l'ausilio di una memoria.

1.1 Codificatore Convoluzionale

Un codificatore traduce una generica sequenza di informazione in un'altra tramite una codifica, nel caso in esame una codifica convoluzionale, che trasforma *m* bit in ingresso in *n* bit in uscita e il cui rapporto $\frac{m}{n}$ viene chiamato tasso di trasmissione.

La seguente rappresentazione raffigura il funzionamento del componente realizzato:



Codificatore Convoluzionale con tasso di trasmissione 1/2

Dalla sequenza U, un bit in ingresso u_k viene tradotto nei due bit p_{1k} , p_{2k} i quali vengono concatenati per creare il codice in uscita y_k . Il modulo si compone di tre registri che contengono l'informazione del bit corrente u_k ed i due bit precedentemente letti u_{k-1}, u_{k-2} (si suppone che all'inizio dell'elaborazione siano uguali a zero).

L'output è generato attraverso sommatori che realizzano i polinomi:

$$p_{1k} = u_k \ xor \ u_{k-1} \ xor \ u_{k-2}$$

$$p_{2k} = u_k \, xor \, u_{k-2}$$

Infine, il codice concatenato generato da ogni bit y_k: p_{1k}, p_{2k} è serializzato per comporre il flusso Y in uscita.

1.2 Memoria

Il componente si interfaccia con una memoria con indirizzamento al Byte dalla quale legge la sequenza U in ingresso divisa in parole di 8 bit e salvate in indirizzi consecutivi a partire dall'indirizzo 1_{10}. Similmente la sequenza Y in uscita viene scritta in memoria divisa in parole di 8 bit e salvate in indirizzi consecutivi a partire dall'indirizzo 1000_{10}.

Il numero di parole N contenute in memoria è scritto nell'indirizzo 0 dal quale si deduce che il massimo numero di parole codificabili è $2^8 - 1 = 255_{10}$ (11111111_2).

1.3 Esempio di funzionamento

Byte in ingresso: **01001011** => Sequenza U: **0 1 0 0 1 0 1 1**

(il primo bit a sinistra è il più significativo)

Sequenza Y: 0 0 1 1 0 1 1 1 1 1 0 1 0 0 1 0 => Byte in uscita: 00110111 11010010

In particolare, il componente al tempo:

t=0: u_k=**0** u_{k-1}=0 u_{k-2}=0 => p_{1k}=0 p_{2k}=0 y_k= 00

t=1: u_k=**1** u_{k-1}=0 u_{k-2}=0 => p_{1k}=1 p_{2k}=1 y_k= 11

t=2: u_k=**0** u_{k-1}=1 u_{k-2}=0 => p_{1k}=0 p_{2k}=1 y_k= 01

t=3: u_k=**0** u_{k-1}=0 u_{k-2}=1 => p_{1k}=1 p_{2k}=1 y_k= 11

primo byte in uscita 00110111

t=4: u_k=**1** u_{k-1}=0 u_{k-2}=0 => p_{1k}=1 p_{2k}=1 y_k= 11

t=5: u_k=**0** u_{k-1}=1 u_{k-2}=0 => p_{1k}=0 p_{2k}=1 y_k= 01

t=6: u_k=**1** u_{k-1}=0 u_{k-2}=1 => p_{1k}=0 p_{2k}=0 y_k= 00

t=7: u_k=**1** u_{k-1}=1 u_{k-2}=0 => p_{1k}=1 p_{2k}=0 y_k= 10

secondo byte in uscita 11010010

2. Architettura

2.1 Interfaccia del componente

Il componente descritto ha 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;
```

2.2 Scelte di design

Il componente è stato realizzato attraverso una macchina sequenziale sincrona composta da un `process` che descrive l'intero funzionamento della macchina a stati finiti e la cui *sensitivity list* è composta dai segnali di clock (`i_clk`) e reset (`i_rst`). Lo stato e tutti i segnali del modulo vengono aggiornati sul fronte di salita del clock. Il reset, che permette alla macchina di fermare l'esecuzione indipendentemente dallo stato in cui si trova, è asincrono e si esegue quando il segnale `i_rst` è posto a 1.

L'algoritmo lavora su una parola alla volta seguendo l'ordine della sequenza. Ad ogni periodo di clock viene estratto un bit partendo dal più significativo mentre la codifica viene registrata in locale per un certo intervallo di tempo prima di essere salvata in memoria. Il processo è composto dalla `procedure encode_and_shift` che si occupa di compiere la codifica convoluzionale di un bit e realizzare lo scorrimento di u_k, u_{k-1}, u_{k-2} rappresentati dai rispettivi segnali `uk, uk1, uk2`. La procedura è adoperata per ogni bit della sequenza U e quindi viene richiamata in ogni stato che si occupa della traduzione (8 stati descritti in seguito).

Si può dedurre che ad ogni 4 bit letti in input corrisponda una parola di output ragion per cui la richiesta di scrittura in memoria verrà inviata ogni 4 codifiche.

La comunicazione con la memoria è gestita dai segnali `o_en` e `o_we` la cui configurazione determina il tipo di accesso:

- `o_en`=1 `o_we`=0 richiesta di lettura il cui risultato è trasmesso sul segnale `i_data`;
- `o_en`=1 `o_we`=1 richiesta di scrittura di `o_data` nella cella con indirizzo `o_address`.

Il segnale di o_done è posto a 1 quando si conclude l'elaborazione e non viene riportato a 0 fino a quando i_start non è posto a 0. Quando o_done=0 la macchina si trova nello stato iniziale in attesa oppure sta eseguendo la codifica della sequenza U.

2.3 Macchina a Stati Finiti

Si è scelto di implementare una FSM composta da 14 stati ed organizzata in:

a. una prima fase di inizializzazione del componente;
b. una fase di iterazione nella quale avviene la lettura in memoria delle parole, la codifica dei bit e la scrittura in memoria dei risultati;
c. un'ultima fase per la chiusura dell'elaborazione corrente e la preparazione per un'eventuale successiva elaborazione.

Di seguito è fornita una descrizione sintetica degli stati che la compongono e lo *State diagram*.

- SLEEP: rappresenta lo stato iniziale della macchina ed è raggiunto appena dopo un segnale di reset (i_rst = 1) o dopo la fine della computazione (quando i_start = 0 e o_done=0). In questo stato il componente resta in attesa di ricevere un segnale di i_start per procedere con la computazione. Quando i_start=1 viene richiesto alla memoria di leggere il contenuto del primo indirizzo.

- READ_N: stato utilizzato per permettere di leggere da memoria l'informazione del primo indirizzo che rappresenta il numero N di parole da codificare.

- SET_N: stato che si occupa di registrare in locale il numero N e controlla che ci siano effettivamente parole da codificare. In caso affermativo manda una richiesta alla memoria per leggere la prima parola della sequenza e continua l'esecuzione nel successivo stato SET_COUNTERS; in caso contrario porta lo stato nella fase di chiusura dell'elaborazione, ossia nello stato DONE.

- SET_COUNTERS: stato in cui vengono inizializzati contatori e segnali locali utilizzati successivamente per facilitare l'elaborazione, in particolare, c_in e c_out contano il numero di parole lette e prodotte mentre i segnali uk1 e uk2 ad ogni ciclo di clock rappresentano i bit u_{k-1}, u_{k-2} precedentemente descritti.

- READ_FIRST_WORD: stato in cui avviene la lettura della prima parola tramite il segnale di input i_data. Prepara la macchina alla lettura del primo bit della sequenza U avvalendosi del segnale uk che rappresenta ad ogni ciclo di clock il bit elaborato dal codificatore.

- EXECUTE_7: stato in cui inizia la fase di iterazione. Viene inizialmente fatto un controllo sulla sequenza di parole ancora da leggere per decidere se terminare l'elaborazione o procedere. In questo stato avviene la codifica del bit alla posizione 7 della parola corrente.

- EXECUTE_6: Esegue la codifica del bit alla posizione 6 della parola corrente.

- EXECUTE_5: Esegue la codifica del bit alla posizione 5 della parola corrente.

- EXECUTE_4: Esegue la codifica del bit alla posizione 4 della parola corrente. Dopo la quarta codifica è stata prodotta una intera parola che viene trasmessa sul segnale o_data per essere scritta in memoria.

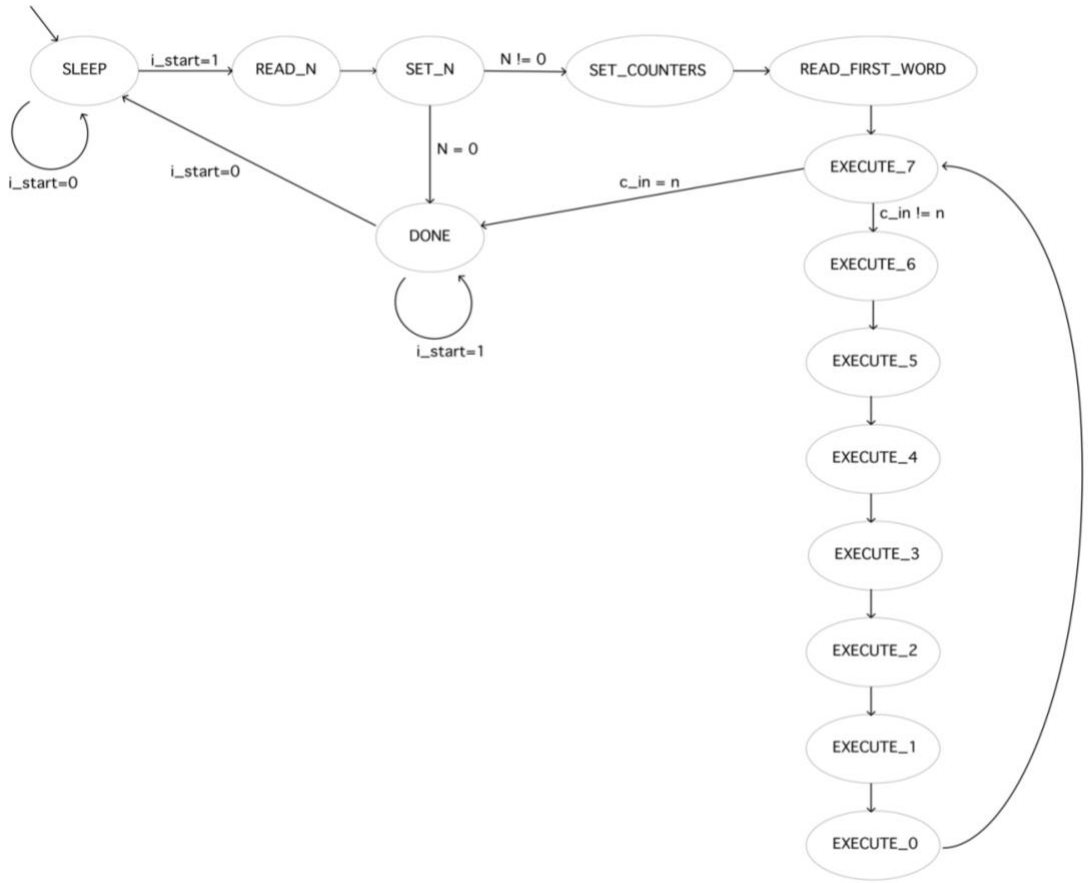
- EXECUTE_3: Esegue la codifica del bit alla posizione 3 della parola corrente.

- EXECUTE_2: Esegue la codifica del bit alla posizione 2 della parola corrente ed inoltra una richiesta di lettura in memoria per ricevere la successiva parola della sequenza U.

- EXECUTE_1: Esegue la codifica del bit alla posizione 1 della parola corrente.

- EXECUTE_0: Esegue la codifica del bit alla posizione 0 della parola corrente e registra la prossima parola da codificare che letta sul segnale i_data. Dall'ultima scrittura in memoria sono stati prodotti altri 8 bit di codice, questi sono trasmessi in o_data per essere scritti in memoria. Infine, riporta lo stato in EXECUTE_7 per procedere con la nuova iterazione.

- DONE: Stato che conclude l'elaborazione. Si raggiunge se la sequenza ha lunghezza nulla oppure se tutte le parole della sequenza U sono state codificate e scritte in memoria. La macchina permane in questo stato finché i_start=1 dopodiché azzera il segnale di fine esecuzione o_done e riporta la macchina allo stato iniziale, pronta per procedere con un'eventuale nuova elaborazione.

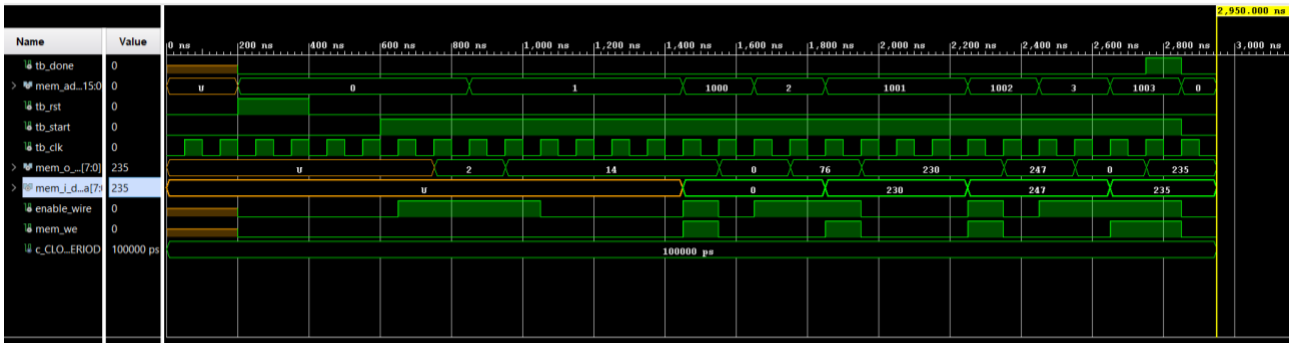


3. Risultati Sperimentali

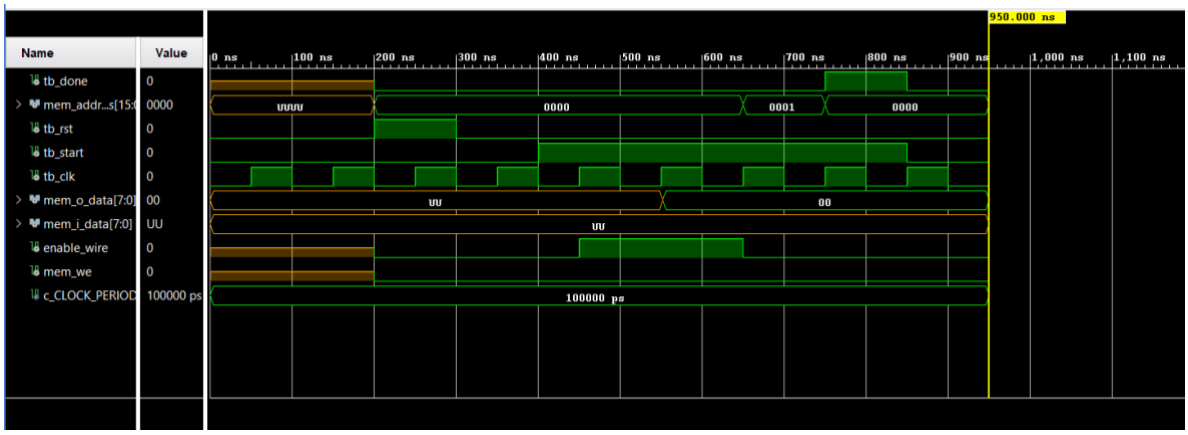
3.1 Simulazione

Il componente è stato testato in pre-sintesi su diverse serie di test con periodo di clock 100 ns. In primo luogo, si è pensato di testarlo su sequenze casuali di bit per controllare la corretta esecuzione della codifica e si è poi proceduto con gli specifici casi limite del flusso U:

- Sequenza minima: 0 bit.
- Sequenza massima: 255*8 bit.



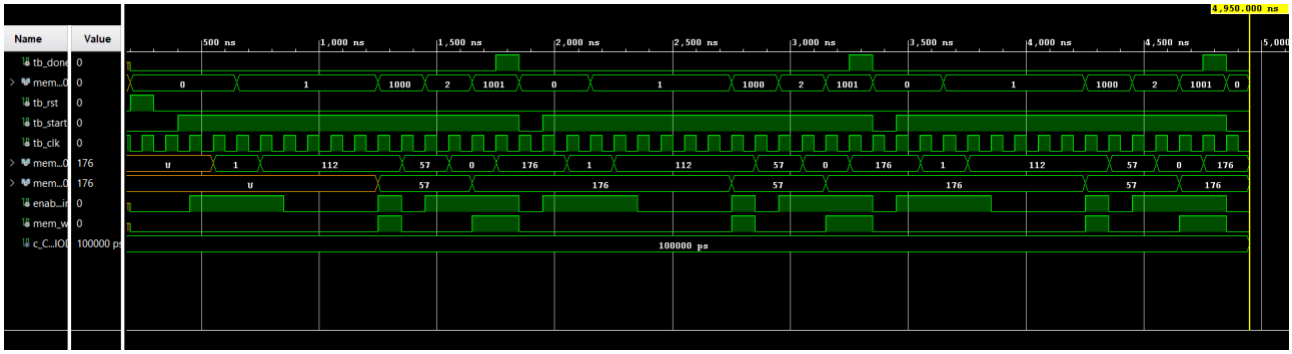
Test: sequenza di 16 bit, senza reset asincroni



Test: sequenza minima, senza reset asincroni

In secondo luogo, è stato provato il comportamento del componente in risposta a più flussi in ingresso di lunghezza casuale tra [0,255] parole. Infine, si è deciso di incorporare prove di verifica sui reset per testare situazioni di: reset asincroni casuali durante l'esecuzione, reset alla chiusura dell'elaborazione e reset ripetuti.

Nella figura seguente è riportato il risultato di un test bench che compie, in assenza di reset asincroni, tre elaborazioni consecutive di lunghezza 1 Byte.



Test: sequenza di 1 Byte ripetuta tre volte

3.2 Sintesi

La sintesi del componente è stata eseguita con successo utilizzando Xilinx Vivado v2018.3 (64-bit) con Artix-7 FPGA xc7a200tfbg484-1 e ha prodotto i seguenti risultati:

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	95	0	134600	0.07
LUT as Logic	95	0	134600	0.07
LUT as Memory	0	0	46200	0.00
Slice Registers	79	0	269200	0.03
Register as Flip Flop	79	0	269200	0.03
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

Successivamente il componente è stato ulteriormente sottoposto a simulazioni post-sintesi (*Post-Synthesis Functional Simulation* e *Post-Synthesis Timing Simulation*) utilizzando gli stessi test usati nella simulazione comportamentale (*Behavioral Simulation*).

4. Conclusioni

Il componente ha superato tutti i test a cui è stato sottoposto in pre-sintesi ed ha risposto positivamente alla fase di testing post-sintesi. In particolare, nelle due fasi i tempi di esecuzione dei test descritti in precedenza sono:

Test su un unico flusso	Behavioral Simulation (ns)	Post-Synth Functional Simulation (ns)
Sequenza minima	950	1 050
Sequenza massima	205 150	205 250

Test su sequenze ripetute da 1 byte	Behavioral Simulation (ns)	Post-Synth Functional Simulation (ns)
3 ripetizioni	4 950	5 450,1
5 ripetizioni	7 950	8 850,1
10 ripetizioni	15 450	17 350,1

In generale il tempo di esecuzione risulta direttamente proporzionale al numero di reset, al numero di flussi da codificare ed alla loro lunghezza.
Come si può osservare, nel caso di un unico flusso il tempo di esecuzione in pre-sintesi e post-sintesi risulta pressoché identico. Osservando invece i test sulle sequenze ripetute è evidente quanto all'aumentare del numero di flussi aumenti progressivamente il ritardo sul tempo di esecuzione in post-sintesi.
In conclusione, si può dire di aver implementato un modulo HW sintetizzabile e correttamente simulabile che realizza un codificatore convoluzionale con tasso di trasmissione ½ rispettando i vincoli imposti dalla specifica.