# Prova Finale - Progetto di Reti Logiche

Prof. Fabio Salice - Anno 2021/2022

Alberto Nidasio - 10665344

5 giugno 2022

## Indice

# 1 Introduzione

## 1.1 Scopo del progetto

Lo scopo del progetto è quello di implementare un modulo hardware, descritto in VHDL, che applichi il codice convoluzionale $\frac{1}{2}$ ad un flusso di bit salvato in memoria. Il flusso di bit generato dovrà essere a sua volta memorizzato.

## 1.2 Codice convoluzionale $\frac{1}{2}$

Un codice convoluzionale è un tipo di codifica nel quale l'informazione, composta da $m$ bit, viene trasformata in un flusso di $n$ bit, dove $m/n$ è il rapporto del codice o tasso di trasmissione.[1]

Nel caso in esame, il codice convoluzionale ha un rapporto $\frac{1}{2}$, quindi per ogni bit di informazione vengono generati due bit. Per il calcolo del flusso in uscita viene seguito lo schema riportato in figura 1 dove i nodi rettangolari rappresentano dei flip flop e i nodi rotondi delle somme.

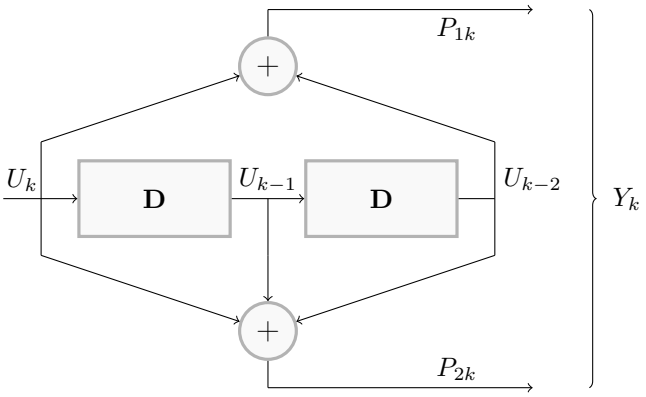


Figura 1: Codificatore convoluzionale con tasso di trasmissione $\frac{1}{2}$

Considerando il flusso $U$ di bit in ingresso e indicando con $U_k$ il bit arrivato nell'istante $k$, la codifica produce due bit, $P_{1k}$ e $P_{2k}$, calcolati come mostrato in eq. 1 e 2. È importate sottolineare che per produrre i bit P viene applicato l'operatore XOR ($\oplus$) così da ottenere una somma senza riporto.

$$P_{1k} = U_k \oplus U_{k-2} \tag{1}$$

$$P_{2k} = U_k \oplus U_{k-1} \oplus U_{k-2} \tag{2}$$

Il flusso in uscita sarà la concatenazione dei bit $P_{1k}$ e $P_{2k}$. Il convolutore è quindi una macchina sequenziale sincrona con un clock globale che scandisce l'ingresso dei bit del flusso $U$ e il calcolo dei bit $P$.

[1] *Codice convoluzionale*. URL: `https://it.wikipedia.org/wiki/Codice_convoluzionale`.

In figura 2 è riportato un esempio in cui vengono codificati 8 bit. Il flusso in uscita è quindi composto come: $P_{10}, P_{20}, P_{11}, P_{21}, ..., P_{17}, P_{27}$.

| T | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $U_k$ | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| $P_{1k}$ | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| $P_{2k}$ | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |

$$U = 10100010$$

$$P = 1101000111001101$$

Figura 2: Esempio di codifica di 8 bit

## 1.3 Descrizione della memoria

Il modulo da implementare deve leggere il flusso $U$ da una memoria con indirizzamento al byte. All'indirizzo 0 è presente la quantità di parole $W$ da codificare e, a partire dall'indirizzo 1, i byte $U_k$. La dimensione massima della sequenza di ingesso è di 255 byte. Il flusso in uscita dovrà essere memorizzato a partire dall'indirizzo 1000.

Di seguito, in figura 3 è rappresentato il contenuto della memoria.

| Indirizzo | Contenuto |
|---|---|
| 0 | $W$ |
| 1 | $U_0$ |
| 2 | $U_1$ |
| ... | ... |
| $W$ | $U_{W-1}$ |
| ... | ... |
| 1000 | $P_{10}$ |
| 1001 | $P_{20}$ |
| ... | ... |
| $1000 + (W-1) * 2$ | $P_{1(W-1)}$ |
| $1001 + (W-1) * 2$ | $P_{2(W-1)}$ |

Figura 3: Contenuto della memoria

## 1.4 Specifiche del componente

Il componente da descrivere in VHDL deve implementare l'interfaccia indicata in figura 4 e rispettare le seguenti caratteristiche:

- L'elaborazione parte quando il segnale `i_start` viene portato a 1;
- Il segnale `o_done` deve essere portato a 1 per indicare il termine della computazione;
- Il modulo deve essere in grado di ripartire ogni qual volta il segnale `i_start` viene portato a 1;
- Il modulo deve resettare il proprio stato ogni volta che il segnale `i_rst` viene portato a 1.

Inoltre il segnale `i_start` rimarrà alto finché `o_done` rimane basso e il segnale `i_rst` verrà inviato solamente una volta, successive elaborazione dovranno ripartire solamente con `i_start`.

```
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;
```

Figura 4: Interfaccia del modulo

# 2 Architettura

# 3 Risultati sperimentali

# 4 Conclusioni