

Prova Finale - Progetto di Reti Logiche
Prof. Fabio Salice - Anno 2021/2022

Alberto Nidasio - 10665344

9 giugno 2022

Indice

1	Introduzione	2
1.1	Scopo del progetto	2
1.2	Codice convoluzionale $\frac{1}{2}$	2
1.3	Descrizione della memoria	3
1.4	Specifiche del componente	3
2	Architettura	4
2.1	Macchina a stati	4
2.2	Calcolo del flusso Z	5
2.3	Scelte progettuali	5
3	Risultati sperimentali	5
4	Conclusioni	5

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.

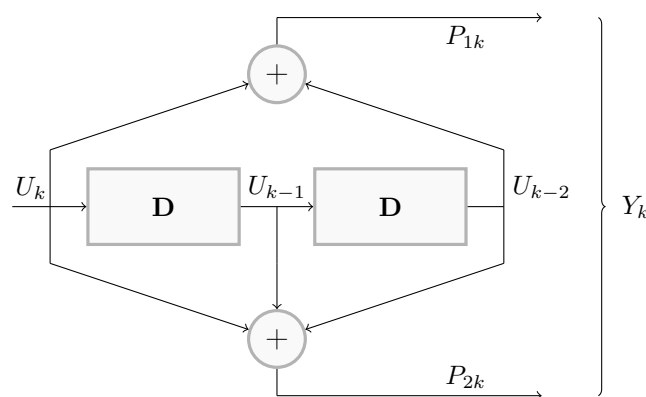


Figura 1: Codificatore convoluzionale con tasso di trasmissione $\frac{1}{2}$

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

$$P_{1k} = U_k \oplus U_{k-2} \quad (1)$$

$$P_{2k} = U_k \oplus U_{k-1} \oplus U_{k-2} \quad (2)$$

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 .

¹Codice convoluzionale.

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

T	0	1	2	3	4	5	6	7	
U_k	1	0	1	0	0	0	1	0	$U = 10100010$
P_{1k}	1	0	0	0	1	0	1	0	
P_{2k}	1	1	0	1	1	0	1	1	$P = 1101000111001101$

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 ingresso è di 255 byte. Il flusso in uscita dovrà essere memorizzato a partire dall'indirizzo 1000.

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_0
1001	P_1
...	...
$1001 + 2 * (W - 1)$	$P_{2*(W-1)-1}$

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

L'interfaccia del componente che deve essere implementata è mostrata in figura 4. In particolare la comunicazione con la memoria avviene tramite i segnali `i_data`, `o_address`, `o_en`, `o_we` e `o_data`.

Per comunicare con la memoria è necessario abilitarla portando a 1 il segnale `o_en`, mentre `o_we` indica la modalità di scrittura, se alto, o la modalità di lettura, se basso.

La memoria, durante il primo rising edge in cui è abilitata, legge o scrive il byte all'indirizzo indicato con `o_address` e riporta il valore sul segnale `o_data`.

2 Architettura

Breve descrizione di come funziona il componente.

2.1 Macchina a stati

La macchina a stati sintetizzata, mostrata in figura 5, è composta da i seguenti 9 stati:

- **IDLE**: La macchina è in attesa del segnale `i_start` e rimane in questo stato finché il segnale non viene portato a 1;
- **REQUEST_W** e **FETCH_W**: Viene attivata la memoria indicando l'indirizzo di lettura 0 di `W`. L'output viene quindi letto da `i_data` e memorizzato, sarà poi utilizzato per contare i cicli di computazione;
- **REQUEST_U** e **FETCH_U**: Viene utilizzata la memoria ancora in modalità lettura per recuperare un byte del flusso `U`, come indirizzo viene utilizzato il numero del ciclo corrente;
- **COMPUTE_P**: In questa fase vengono calcolati due byte del flusso in uscita a partire dal byte precedentemente letto, vedi sezione 2.2;
- **WRITE_P1** e **WRITE_P2**: Vengono scritti in memoria i due byte del flusso `Z` calcolati precedentemente. Se sono stati elaborati tutti i `W` byte del flusso `U`, la macchina passa nello stato **DONE** portando a 1 il segnale `o_done`, altrimenti continua ripartendo da **REQUEST_U**;

- **DONE**: La computazione è terminata e viene atteso il reset del segnale **i_start** per riportare a zero **o_done** e tornare nello stato di **IDLE**.

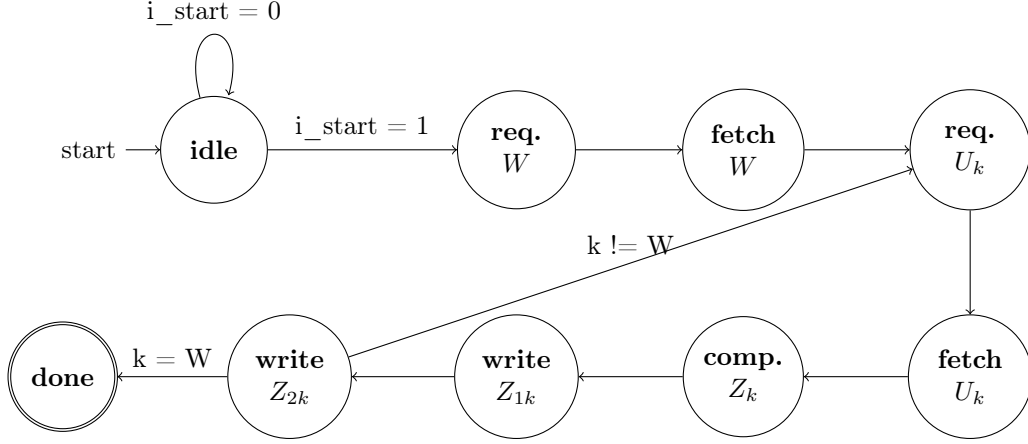


Figura 5: Macchina a stati del componente implementato

2.2 Calcolo del flusso Z

Il convolutore descritto in sezione 1.2 potrebbe essere implementato tramite la macchina a stati illustrata nelle specifiche del progetto², avendo quindi 4 stati e calcolando, ad ogni step, i bit P_{1k} e P_{2k} . In questo modo ci vorrebbero 8 cicli di clock per concludere l'elaborazione di un byte letto da memoria.

Considerando l'indirizzamento al byte, è stato utilizzato un approccio differente per il calcolo di P che permette di elaborare un byte del flusso U in un singolo ciclo di clock.

All'interno del componente viene mantenuto un buffer composto da un vettore di 10 elementi in grado di memorizzare il byte in ingresso e gli ultimi due bit del byte letto durante lo step precedente. Questo ci permette di eseguire le equazioni 1 e 2 su ciascuno degli 8 bit letti in un'operazione sola. Come mostrato in figura 6, in VHDL questo viene ottenuto tramite un ciclo **for** che elabora il byte in ingresso e produce i due byte del flusso in uscita P .

Il risultato è un componente che riesce ad elaborare ciascun byte letto dalla memoria in un singolo ciclo di clock, riducendo così il tempo che il convolutore impiega per elaborare il flusso in ingresso U . Il vantaggio in termini di tempo viene ottenuto a discapito delle risorse hardware utilizzare, infatti vengono sintetizzate le eq. 1 e 2 per 8 volte.

2.3 Scelte progettuali

3 Risultati sperimentali

4 Conclusioni

²PFRL_Specifica_21_22_V3.

```

-- Run the 1/2 convolutional code
for k in 7 downto 0 loop
    -- Compute P1k and P2k
    next_P(k * 2 + 1) <= conv_state(k + 2) xor conv_state(k);
    next_P(k * 2)      <= conv_state(k + 2) xor conv_state(k + 1)
                                xor conv_state(k);
end loop;

```

Figura 6: Calcolo di due byte del flusso K