Progetto di Digital Systems
Turbo Coding Interleaver

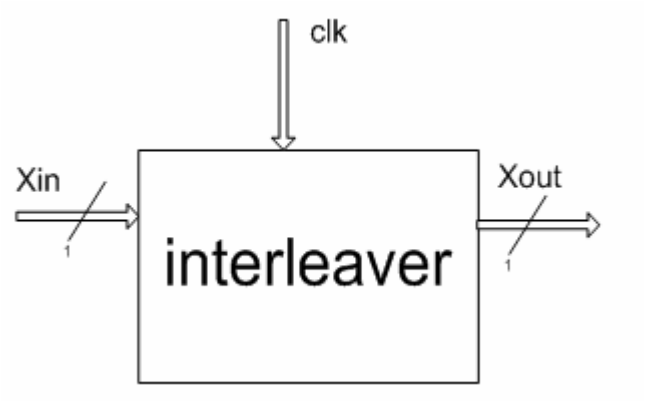
David Costa

Introduzione

L'obiettivo è quello di progettare un RP (relative prime) interleaver per turbo codici compatibile con le seguenti specifiche:

- Lunghezza interleaver = 1024.
- Implementi la relazione $X_{out}(i) = X_{in}(|45 + i \cdot 3|_{1024})$

Schema a blocchi:



Nel primo capitolo saranno presentati, in modo semplificato, i concetti di codifica di canale e di turbo codici, in maniera di poter introdurre il dispositivo nelle sue possibili applicazioni.

Verrà poi mostrato l'interleaver, in particolare il nostro modello, il suo sviluppo in linguaggio *VHDL* ed i test effettuati.

Capitolo 1

Interleaver

1.1 Codifica di canale

In una trasmissione digitale l'utilizzo di canali rumorosi, che introducono errori nella comunicazione dei dati, richiede delle tecniche di elaborazione del segnale informativo volte a garantire l'integrità dei dati per una trasmissione il più possibile ottimale del messaggio.

La prerogativa di rappresentare l'informazione in formato digitale porta, rispetto alla trasmissione analogica, i vantaggi della codifica di canale (integrità dei dati) e della codifica di sorgente (compressione dell'informazione).

Il processo di *codifica di canale* si presta a codificare le informazioni inviate in un canale di comunicazione in modo che, in presenza di rumore, gli errori possano essere individuati e/o corretti. Si distinguono due metodi di codifica:

- Backward Error Correction (BEC)
- Forward Error Correction (FEC)

Il BEC richiede solo l'individuazione dell'errore, mentre nel FEC il decoder deve avere anche la capacità di correggere un certo numero di errori e di localizzare la posizione in cui sono occorsi. Nel resto della relazione prenderemo in cosiderazione quest'ultimo.

Tra i tipi di codici per la correzione richiamiamo brevemente i *codici convoluzionali*.

Nei codici convoluzionali ogni simbolo di informazione a m bit da codificare è trasformato in un simbolo a n bit (con $n > m$), e m/n è il rapporto (*rate*). Il rate esprime la quantità di ridondanza nel codice: più basso è il rate, più ridondante è il codice.

La struttura di un semplice codificatore convoluzionale è mostrata in figura 1.1. La lunghezza k del registro a scorrimento è detta *constraint length* (lunghezza dei vincoli), e la trasformazione è una funzione degli ultimi k simboli di informazione (nella figura $k = 3$). Il codificatore ha n sommatori e tipicamente i k registri di memoria sono inizializzati al valore 0, hanno un bit di ingresso e due di uscita, combinazioni lineari del bit attuale e di alcuni precedenti (due, in figura). Un bit di ingresso è immesso nel registro più a sinistra, il codificatore fornisce il risultato e sposta i bit di tutti i valori di registro a destra e attende il prossimo bit di ingresso. Se non ci sono bit d'ingresso rimanenti, il codificatore continua a fornire risultati fino a quando tutti i registri sono ritornati allo stato zero. I bit codificati sono convoluzioni della sequenza di ingresso.

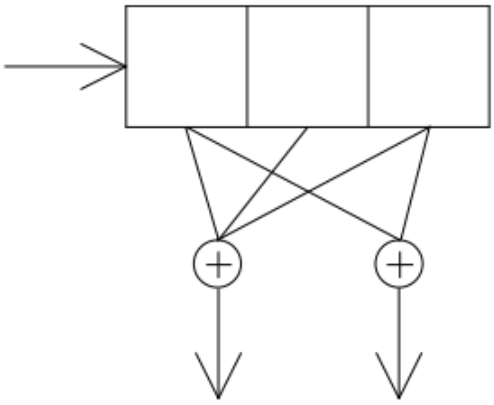


Figura 1.1: Codificatore convoluzionale

A differenza dei codici lineari, i codici convoluzionali sono dotati di memoria in quanto l'influenza di un blocco di bit in ingresso si protrae sulla codifica dei blocchi successivi.

1.2 Turbo codici

I turbo codici sono una classe di codici di correzione degli errori ad alte prestazioni introdotti nel 1993.

Sono una nuova classe di codici convoluzionali che si avvicinano strettamente ai limiti teorici imposti dal (secondo) teorema di Shannon. La capacità di Shannon, nota anche come *limite di Shannon*, di un canale di comunicazione è il massimo tasso di trasferimento di dati che può fornire il canale per

un dato livello di rapporto segnale/rumore, con un tasso di errore piccolo a piacere.

Se una probabilità di errore per bit p_b è accettabile, sono raggiungibili tassi fino a $R(p_b)$:

$$R(p_b) = \frac{C}{1 - H_2(p_b)}$$

$$H_2(p_b) = -[p_b \cdot log(p_b) + (1 - p_b) \cdot log(1 - p_b)]$$

I codici turbo hanno prestazioni che approssimano a meno di 1 dB i risultati teorici di Shannon.

L'idea di base è la concatenazione di due codici: l'informazione viene codificata, tutti i bit ottenuti sono permutati da un *interleaver* e nuovamente codificati da un secondo codice (figura 1.2).

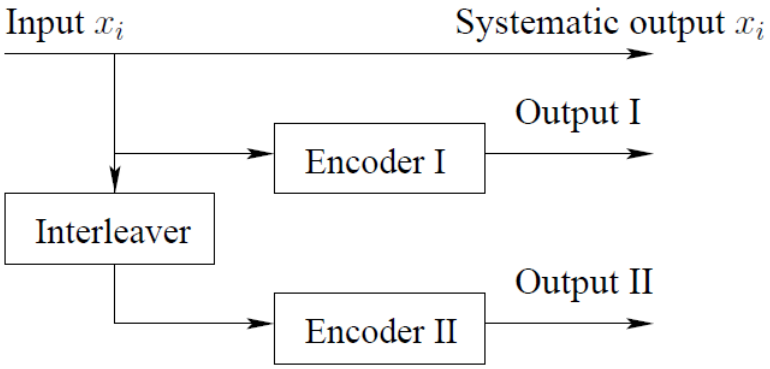


Figura 1.2: Generico turbo encoder

La novità rispetto alle tradizionali tecniche di decodifica dei codici concatenati è la decodifica iterativa: in ricezione si attivano alternativamente i decodificatori dei due codici componenti, in modo da utilizzare ad ogni passo l'informazione prodotta dall'altro decodificatore al passo precedente. Questo offre una complessità di decodifica molto minore rispetto i precedenti algoritmi.

In pratica, gli stessi dati sono codificati con due codificatori disgiunti e la decodifica viene fatta utilizzando per ogni bit da decodificare l'affidabilità di quel bit fornita dall'altro codificatore. In altre parole, prima viene fatta

una decodifica con il codice 1; vengono anche memorizzate le verosimiglianze di ogni bit. Poi, si decodificano gli stessi dati con il codice 2, e si usano le verosimiglianze precedenti per aiutare le decisioni.

L'interleaving è alla base dei turbo codici e la scelta dell'interleaver, avendo un effetto significante sulle prestazioni dei codici, è parte cruciale nella loro progettazione.

1.3 Interleaving

Con il termine Interleaving si indica una tecnica di elaborazione di un segnale digitale utilizzata nelle trasmissioni digitali per disporre i dati in maniera non contigua al fine di migliorare le prestazioni in termini di rilevazione e correzione di errori in ricezione. Come mostrato precedentemente, è parte fondamentale dei turbo codici.

Per mostrare come avviene la permutazione dei dati e l'efficacia del metodo utilizziamo un esempio: applichiamo un codice di correzione dell'errore molto semplice (codice a ripetizione) ed effettuiamo il processo di interleaving ad un messaggio, che sarà inviato tramite un canale rumoroso che causa errori a pacchetti, cioè a bit consecutivi.

```
messaggio                       ==>  a  b  c  d  e  f

messaggio codificato            ==>  aaabbbcccdddeeefff

messaggio con interleaving      ==>  abcdefabcdefabcdef

messaggio ricevuto con errori   ==>  abcdefabcdXXXbcdef

messaggio dopo deinterleaving   ==>  aaXbbbcccdddXeefXf

messaggio decodificato          ==>  a  b  c  d  e  f
```

Si nota che, nonostante la presenza di errori (segnati dalle "X"), la particolarità di averli permutati permette di poter ricostruire il messaggio, cosa non possibile senza la tecnica di interleaving-deinterleaving.

L'interleaver può essere definito ed implementato come mostrato in figura 1.3. Il dispositivo legge da un vettore di bit di ingresso v_{in} e scrive su un

vettore di uscita v_{out} i bit *interleaved*, cioè permutati.

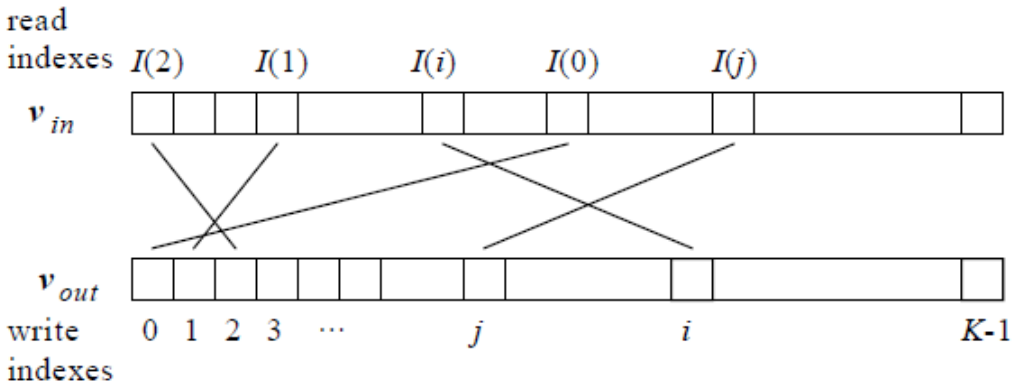


Figura 1.3: Definizione di interleaver

I dati in uscita vengono scritti considerando gli indici $i = 0...K-1$, dove K è la lunghezza dell'interleaver. Il vettore I definisce l'ordine con cui vengono letti i bit in ingresso, quindi l'i-esimo bit di output, scritto nella locazione i del vettore di uscita, è letto dalla locazione $I(i)$ del vettore di ingresso.

Un RP (*relative prime*) interleaver di lunghezza K è definito da

$$I(i) = |s + i \cdot p|_K, \qquad i = 0...K-1$$

dove p e K sono numeri relativamente primi (il loro massimo comun divisore è 1) e s è l'indice di inizio. La caratteristica di essere relativamente primi assicura che ogni elemento venga letto una sola volta.

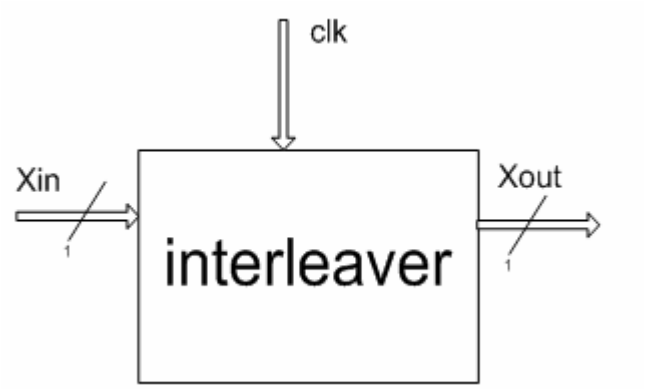
L'interleaving viene principalmente usato nella comunicazione dati, nei file multimediali, nella trasmissione via radio (per esempio, su satellite o sulla TV digitale o su ADSL). L'interleaving viene utilizzato anche nei supporti ottici (CD-ROM, DVD, etc) per proteggere i dati da graffi e danni al supporto di memorizzazione.

Capitolo 2

Architettura

2.1 Diagramma a blocchi

Di seguito è riportato il diagramma a blocchi per l'interleaver.



La rete prevede semplicemente il segnale di ingresso X_{in} (dati da permutare) ed il segnale di uscita X_{out} (dati permutati). Il *reset*, attivo alto, serve a settare il segnale di uscita a zero; non è necessario resettare i dati tra un'operazione e l'altra. Il segnale di *clock*, infine, serve a rendere la rete sincrona. Ad ogni ciclo di clock, se non è settato il reset, viene fornita in uscita una permutazione del segnale di ingresso secondo la relazione specificata.

2.2 Implementazione

Nella descrizione fatta in *VHDL* si è scelto di utilizzare un'architettura di tipo Behavioural in quanto il nostro dispositivo non è una struttura composta da più elementi, ma si presta ad essere descritto attraverso un processo che svolgerà la relazione ingresso-uscita.

I segnali *clock* e *reset* sono `std_logic` in ingresso, X_{in} e X_{out} sono bus `std_logic_vector` di lunghezza 1024, come richiesto nelle specifiche dell'interleaver.

La relazione da implementare nel nostro specifico dispositivo è $X_{out}(i) = X_{in}(|45 + i \cdot 3|_{1024})$, quindi l'indice iniziale è 45, la lunghezza è 1024 e notiamo che 3 e 1024 sono effettivamente *relative prime*. La permutazione avviene all'interno del processo, gestito dal segnale di clock: quando *clock* assume valore 1, all'interno di un ciclo `for` viene assegnato ad ogni bit del vettore di uscita il corrispettivo valore del bit del vettore di ingresso, secondo la relazione. X_{out} mantiene il suo valore finché, ad un nuovo ciclo di clock, non varia il segnale di ingresso.

2.3 Workbench

Per testare l'implementazione dell'interleaver è stata scritta una rete di workbench. Il test dura 50 cicli di clock (dalla durata di 200 ns) e ad ogni cinque di questi viene variato il valore in ingresso. Per semplificare la constatazione i vettori di input sono con tutti i bit settati a 0 escluso uno settato ad 1. I valori stabiliti prevedono in uscita solo il primo bit di X_{out} posto a 1 per il primo intervallo di tempo, il secondo per il secondo intervallo, e così via progressivamente.

```
Xin(45) = 1      ==>     Xout(0) = 1

Xin(48) = 1      ==>     Xout(1) = 1

Xin(51) = 1      ==>     Xout(2) = 1

Xin(54) = 1      ==>     Xout(3) = 1

...
```

Le figure 2.1 e 2.2 mostrano i risultati del test, la prima con la sequenza di ingresso e la seconda con quella di uscita. Per motivi di spazio sono stati inclusi solo i bit che nel corso della prova cambiano il loro valore, escludendo i dati che si mantengono a 0.

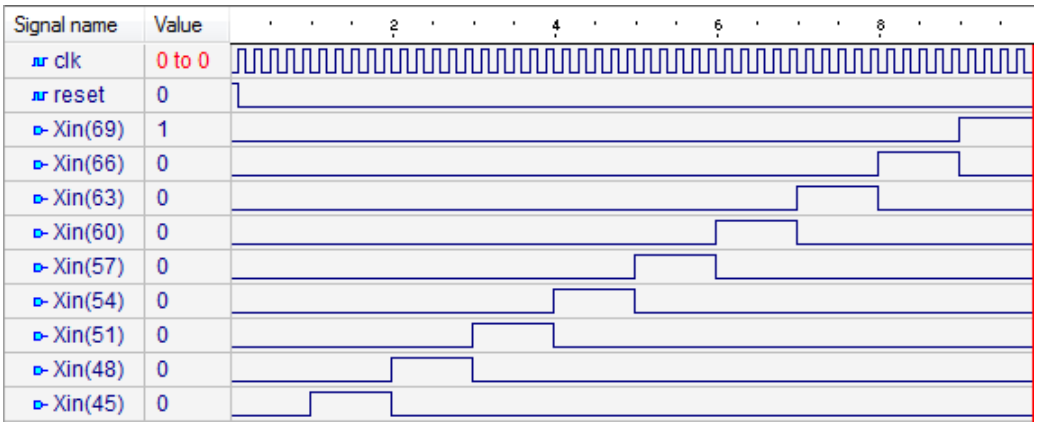


Figura 2.1: Test: segnale di ingresso

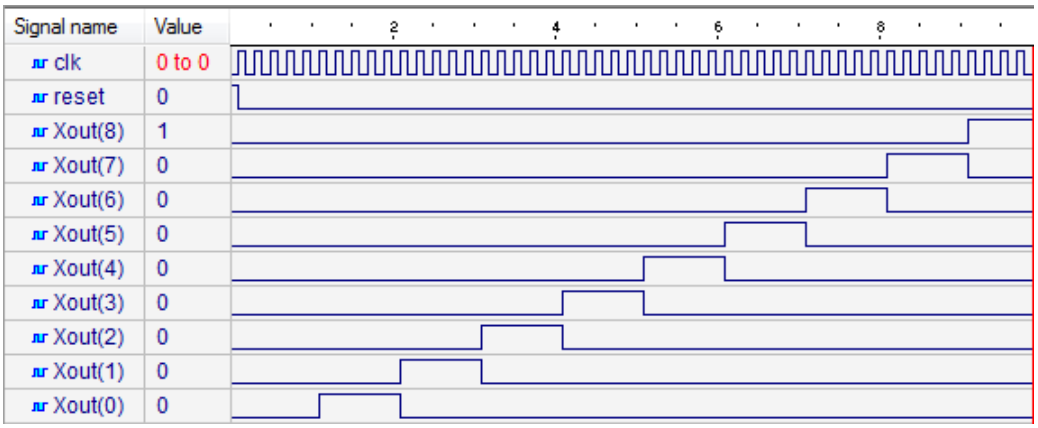


Figura 2.2: Test: segnale di uscita

Il test fornisce risultati corretti ad ogni prova, quindi si può presumere con una certa sicurezza che l'implementazione dell'algoritmo sia corretta.

Capitolo 3

Codice

3.1 Interleaver

```
1  library IEEE;
   use IEEE.std_logic_1164.all;
3  use IEEE.std_logic_arith.all;

5  entity interleaver is
       generic (
7          N : INTEGER := 1024
           );
9      port (
           Xin  : in std_logic_vector(N-1 downto 0); -- ingresso
11         Xout  : out std_logic_vector(N-1 downto 0); -- uscita
           clk  : in std_logic;       -- segnale di clock
13         reset : in std_logic       -- segnale di reset
           );
15 end interleaver;

17 architecture BEHAVIOURAL of interleaver is
   begin
19     proc : process( clk )
       begin
21         if ( reset = '1' ) then
               for I in 0 to N-1 loop
23                 Xout(I) <= '0';
               end loop;
25         else if ( clk = '1' ) then
                   for I in 0 to N-1 loop
27                     Xout(I) <= Xin((45 + I*3) mod N);
                   end loop;
29             end if;
           end if;
```

```
    end process;
end BEHAVIOURAL;
```

3.2 Workbench

```
library IEEE;
use IEEE.std_logic_1164.all;

entity interleaver_tb is

end interleaver_tb;


architecture interleaver_test of interleaver_tb is
component interleaver
    generic (
        N : INTEGER := 1024
        );
    port (
        Xin  : in std_logic_vector(N-1 downto 0); -- ingresso
        Xout  : out std_logic_vector(N-1 downto 0); -- uscita
        clk  : in std_logic;        -- segnale di clock
        reset : in std_logic        -- segnale di reset
        );
end component;


constant N    : INTEGER := 1024;  -- Bus Width
constant MckPer  : TIME  := 200 ns; -- Master Clock Period
constant TestLen  : INTEGER := 100;  -- No. of Count (MckPer/2) for test


signal clk   : std_logic := '0';
signal reset : std_logic := '0';
signal Xin  : std_logic_vector(N-1 downto 0);


signal Xout  : std_logic_vector(N-1 downto 0);

signal clk_cycle : INTEGER;
signal Testing  : BOOLEAN := True;


begin

    I : interleaver generic map (N => 1024)
```

```
42    port map (Xin, Xout, clk, reset);

44

      clk <= not clk after MckPer/2 when Testing else '0';
46

48    Test_Proc : process(clk)
        variable count : INTEGER := 0;
50    begin
        clk_cycle <= (count + 1) / 2;
52      case count is
          when 0 => for I in 0 to N-1 loop
54                      Xin(I) <= '0';
                    end loop;
56                  reset <= '1';
          when 1 => reset <= '0';
58        when 10 =>  for I in 0 to N-1 loop
                      Xin(I) <= '0';
60                  end loop;
                    Xin(45) <= '1';
62        when 20 => for I in 0 to N-1 loop
                      Xin(I) <= '0';
64                  end loop;
                    Xin(48) <= '1';
66        when 30 => for I in 0 to N-1 loop
                      Xin(I) <= '0';
68                  end loop;
                    Xin(51) <= '1';
70        when 40 => for I in 0 to N-1 loop
                      Xin(I) <= '0';
72                  end loop;
                    Xin(54) <= '1';
74        when 50 => for I in 0 to N-1 loop
                      Xin(I) <= '0';
76                  end loop;
                    Xin(57) <= '1';
78        when 60 => for I in 0 to N-1 loop
                      Xin(I) <= '0';
80                  end loop;
                    Xin(60) <= '1';
82        when 70 => for I in 0 to N-1 loop
                      Xin(I) <= '0';
84                  end loop;
                    Xin(63) <= '1';
86        when 80 => for I in 0 to N-1 loop
                      Xin(I) <= '0';
88                  end loop;
                    Xin(66) <= '1';
90        when 90 => for I in 0 to N-1 loop
```

```
                    Xin(I) <= '0';
92                  end loop;
                    Xin(69) <= '1';
94              when (TestLen-1) => Testing <= False;
                when OTHERS => NULL;
96          end case;

98          count := count + 1;

100     end process Test_Proc;

102 end interleaver_test;
```