TURBO CODING INTERLEAVING

Progettazione di un interleaver turbo-code compatibile

AA 2015/2016

Chiara Caiazza

INDICE

1 INTRODUZIONE

1.1 Descrizione dell'algoritmo 3
1.2 Possibili applicazioni
1.2.1 Trasferimento tramite la rete 4
1.2.2 gestione dei dischi 5
1.3 Architettura 6

2 CODICE

2.1 Codice VHDL 7
2.2 Codice test in c++ 9
2.3 Codice test bench 11

3 SCELTA DEI VALORI DI TEST

3.1 Esempio di esecuzione in c++ 15

3 SCELTA DEI VALORI DI TEST 14

4 SINTESI 17

1 Introduzione

Un interleaver è un dispositivo che elabora un set di dati in modo da disporli in maniera non contigua. Tutto ciò viene effettuato al fine di migliorare il tasso di successo degli algoritmi di FEC (Forward Error Correction) nel caso di errori multipli consecutivi.

1.1 Descrizione dell'algoritmo

Esso prende in ingresso un set di valori di ingresso e li riordina secondo una data relazione che deve essere nota sia in fase di interleaving che in fase di de-interleaving.

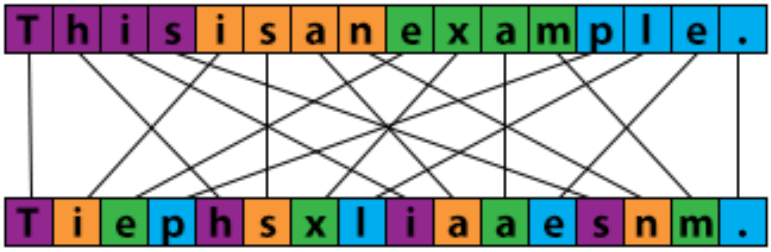


Figura 1: interleaving

Per riottenere l'informazione iniziale la sequenza deve essere sottoposta al processo inverso (de-interleaving).

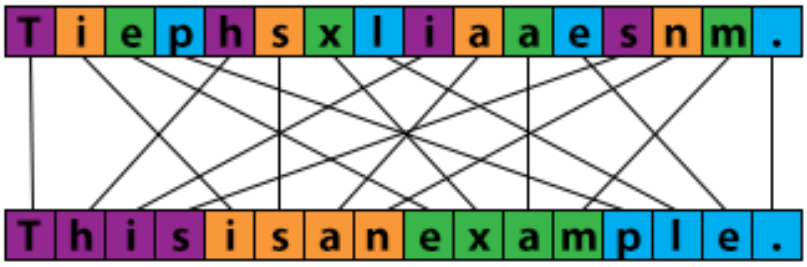


Figura 2: de-interleaving

1.2 Possibili applicazioni

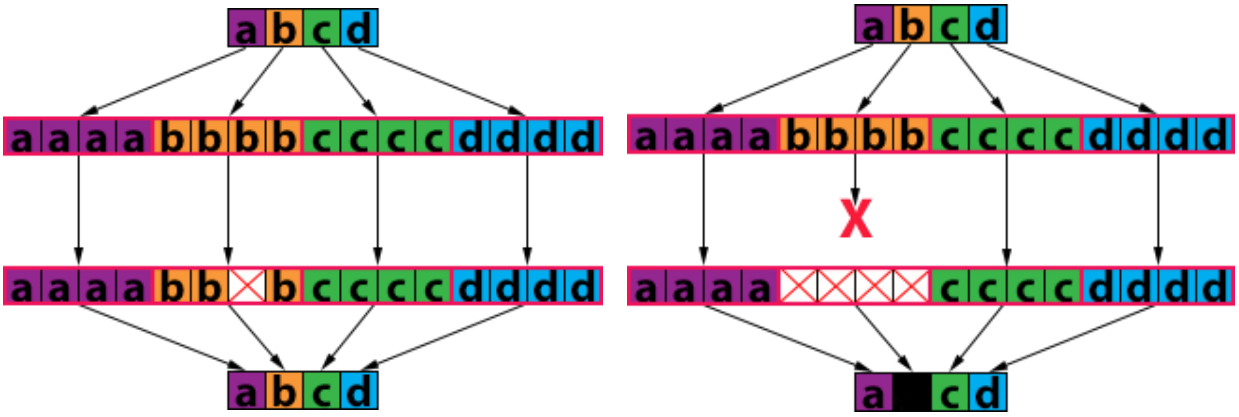
1.2.1 Trasferimento tramite la rete

L'interleaver ha utilizzi nelle comunicazioni dati tramite collegamenti Internet, radio o satellitare.

In questi ambiti la possibilità di avere corruzione di dati all'interno di un pacchetto, od addirittura la perdita totale del pacchetto stesso, non sono trascurabili. In ricezione vengono applicate algoritmi FEC per tentare di correggere l'errore introducendo ridondanza all'interno del messaggio, tuttavia tali tecniche sono limitate dalla quantità di bit errati presenti sul pacchetto.

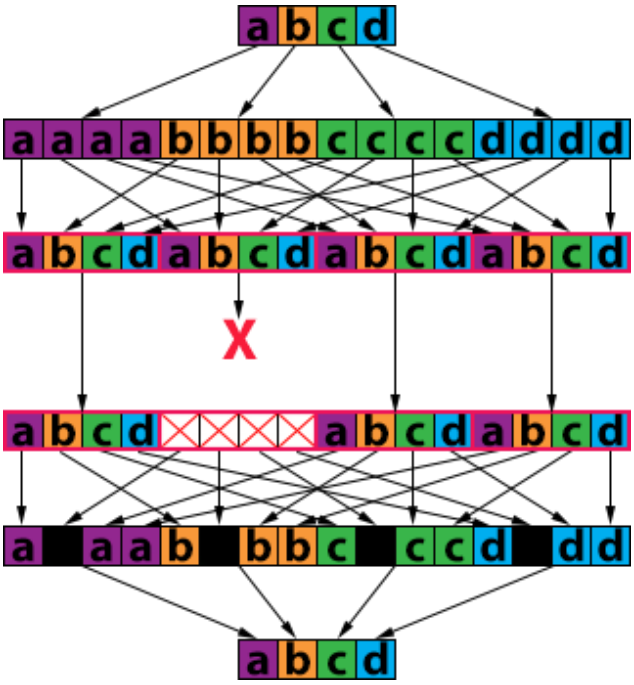
Grazie all'interleaver invece i dati vengono redistribuiti sull'intero set di valori soggetto al processo, diminuendo la quantità di errori presente in ogni singolo pacchetto ed aumentando le probabilità di riuscire nel ripristino dell'informazione.

Analizziamo un esempio banale. Nell'immagine seguente possiamo notare che se non ci avvaliamo di interleaver possiamo ripristinare l'informazione se un carattere viene corrotto, ma se un intero pacchetto viene perso non posiamo fare nulla.



Ora invece possiamo notare cosa capita nel caso il mittente utilizzi un interlaver.

Anche se perdiamo un pacchetto l'errore viene suddiviso nell'intera sequenza e quindi il recovery diventa più semplice.



È importante anche notare che la maggior parte dei canali di comunicazione non è memoryless, dunque la perdita di pacchetti all'interno della rete non è indipendente. L'utilizzo di un interleaver contribuisce a distribuire l'errore in maniera più uniforme all'interno dello stream in uscita, ma se le perdite diventano troppo elevate, il ripristino del dato iniziale rimane ugualmente irrealizzabile.

Infine notiamo che, sfortunatamente, l'utilizzo di un interleaver aumenta la latenza in quanto è necessario aver ricevuto un certo numero di pacchetti prima di effettuare il de-interleaving. Dunque e necessario scegliere con attenzione la quantità di dati da interlacciare, cercando un trade-off tra il livello di integrità della comunicazione e la latenza massima che è possibile sopportare.

1.2.2 gestione dei dischi

In passato il meccanismo di interleaving è stato utilizzato anche per motivi che non sono correlati al ripristino di informazioni corrotte, ma per permettere la memorizzazione di dati sequenziali in un insieme di blocchi non contigui in modo da migliorare le performance dei dischi, diminuendo le latenze negli accessi.

Quando un blocco di dati viene letto dal disco viene posto in un buffer in attesa di essere spostato altrove. Nel caso i dati fossero stati salvati in locazioni contigue della memoria il tempo necessario a spostare il dato sarebbe sufficiente a far passare sotto la testina il blocco successivo, costringendo il driver ad aspettare un'intera rotazione prima di effettuare la lettura del secondo blocco.

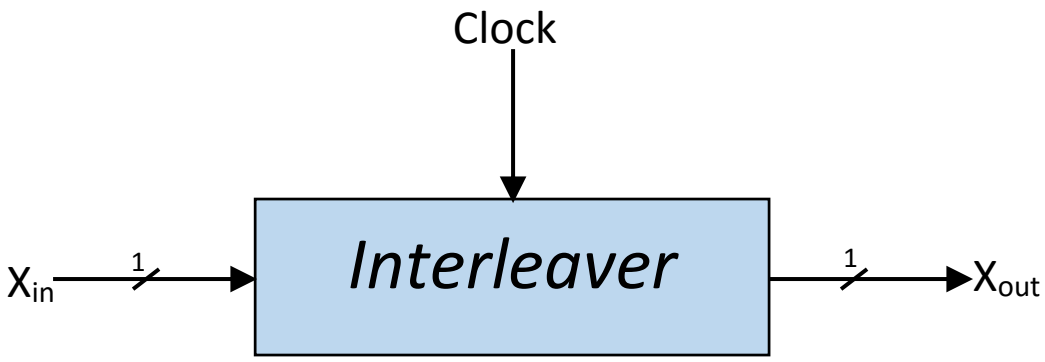
Per comprendere in quale modo l'interleaving può ridurre la latenza in accesso supponiamo di avere 9 settori in una traccia, e supponiamo che per elaborare il dato letto sia necessario un tempo sufficiente a far passare sotto la testina 3 blocchi (il quarto blocco dopo la lettura è il primo blocco a cui il driver ha accesso). Possiamo pensare di riordinare le scritture come segue:

1 8 6 4 2 9 7 5 3

In lettura come prima cosa si accede al blocco n°1 (nel tempo necessario all'elaborazione i blocchi 8,6 e 4 passano sotto la testina). Non appena il driver è pronto a leggere il secondo dato il blocco n°2 si trova sotto la testina ed è quindi possibile leggerlo immediatamente.
Il medesimo ragionamento è applicabile a tutti gli altri blocchi.

Attualmente l'utilizzo dell'interleaving è stato abbandonato in favore dell'utilizzo di buffer di dimensioni maggiori e di cluster di settori.

1.3 Architettura



Lo schema proposto mostra la struttura dell'interleaver implementato dove X_{in} rappresenta il bit d'ingresso ed X_{out} il bit di uscita.

Durante i primi 1024 cicli di clock [0, 1023] viene preso in bit i-esimo, durante i successivi 1024 [1024, 2047] viene restituito l'i-esimo bit di uscita. La relazione che lega bit di ingresso con i bit di uscita è la seguente:

$$X_{out}(i)=X_{in}(|45+i*3|_{1024})$$

2 Codice

2.1 Codice VHDL

```
---------------------------------------------------------------------------- -
--Interleaver
--
--File name : interleaver.vhd
--
--Author: Chiara Caiazza
---------------------------------------------------------------------------- -


Library IEEE;
Use IEEE.STD_LOGIC_1164.ALL;
Use IEEE.STD_LOGIC_ARITH.ALL;
Use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.ALL;


---------------------------- Entity declaration  --------------------------------
entity interleaver is

        port(
                clk     : in   std_logic;           -- Processing clock
                reset   : in   std_logic;           -- Asynchronous active high reset
                bit_in  : in   std_logic;           --  input bit
                bit_out : out std_logic             --  output bit
   );

end interleaver;

------------------ Architecture begins here  ----------------------------------------
architecture Behavioral of interleaver is

----------------------- signals declaration  ----------------------------------------
Begin

        process(clk, reset, bit_in)

                --a implements the relation (actualIndex*3+45)mod 1024
                variable a: integer;
                --actual cycle#
                --          [0-1023]    -> 1024 input cycle  (one  input bit for each cycle)
                --          [1024-2047] -> 1024 output cycle (one output bit for each cycle)
                --          >=2048      -> reset
                variable actualIndex: integer;
                -- store the input bit at the index a
                variable memory: std_logic_vector(1023 downto 0);
```

```
begin

        interleaver:

        -- high active reset
        if reset = '1' then
                -- initialize actual index
                actualIndex:=0;
                -- initialize a
                a:=0;
                bit_out<='Z';

                --all the bits in the array are equal to zero
                for i in 0 to 1023 loop
                        memory(i):='0';
                end loop;


        elsif clk = '1' and clk'event then
        -- In this case I read the input bit and i store it into the array
        -- or I show an input bit depending on actualIndex

                if actualIndex<1024 then
                -- I store the input bit into the array in the a-th element


                        --set the value for a
                        a:=actualIndex*3;
                        a:= a+45;

                        for i in 0 to 2 loop
                                if a>1023 then
                                        a:= a-1024;
                                end if;
                        end loop;

                        --I want to store an input bit...
                        --the value on out is not a significant one
                        bit_out<='Z';
                        -- I store the input bit
                        memory(a):=bit_in;


                elsif actualIndex<2048 then
                        -- I shown the output bit. actualIndex belongs to [1024-2047]
                        -- interval. If I subtract 1024 then it belongs to [0-1023] interval!
                        bit_out<=memory(actualIndex-1024);


                elsif actualIndex>=2048 then
                -- I reset all the index and the memory!

                        -- reset actual index
                        actualIndex:=0;
                        -- reset a
                        a:=0;
                        --output bit is not significant anymore
                        bit_out<='Z';

                        --reset all the bits in the array
                        for i in 0 to 1023 loop
```

```
                                        memory(i):='0';
                                    end loop;

                                end if;

                                --i inctrease actual index
                                actualIndex:=actualIndex+1;

                        end if;
                end process;
        end behavioral;
```

2.2 Codice test in c++

Per il testing è stata scritta la seguente classe c++:

Tester.h

```
#include <string>
#include <iostream>
#include <stdio.h>
#include <bitset>
#include <fstream>
#include <cstdlib>
#include <cstdio>

using namespace std;

class Tester
{
        private:
                ifstream inputStream;
                ofstream outputStream;
                ofstream testStream;
                string nameofInputStream;
                string nameofOutputStream;
                string nameofTestStream;
                int turn;
                void storeATest(bitset<1024>, bitset<1024>);
                void openAStream(string);
        public:
                Tester();
                ~Tester();
                string getAnInputLine();
                int checkInput(string);
                string hexToBits(string);
                bitset<1024> computeOutputBitSet(bitset<1024>);
                void wroteAResponse(bitset<1024>, bitset<1024>);

};
```

MyTester.cpp

```
#include <string>
#include <iostream>
#include <stdio.h>
#include <bitset>
#include <cstdlib>
#include <cstdio>

#include "Tester.h"

using namespace std;


int main(int argc, char *argv[])
{
	bitset<1024> outputBitSet;
	string inputBuffer, inputbit;
	Tester tester ;

	for (;;)
	{

		inputBuffer = "";

		inputBuffer = tester.getAnInputLine();
		if (inputBuffer == "-1")
			return 1;

		/*check the input value*/
		if (tester.checkInput(inputBuffer) < 0)
		{
			cout << "The input value in input.txt is not a valid one!" << endl;
			cout << "Premere un tasto per continuare\n\n";
			getchar();
			return 1;
		}

		cout << "input value (" << inputBuffer.size() << " hex char):                    " <<
                                                                        inputBuffer << endl;

		inputbit = tester.hexToBits(inputBuffer);
		bitset<1024> inputBitSet(inputbit);
		cout << "\n\nThis is my  input bitset (" << inputBitSet.size() << " bits):
                                                        " << inputBitSet << endl;

		outputBitSet = tester.computeOutputBitSet(inputBitSet);
		cout << "\n\nThis is my  output bitset (" << outputBitSet.size() << " bits):
                                                        " << outputBitSet << endl;

		tester.wroteAResponse(inputBitSet, outputBitSet);
		cout << "Premere un tasto per continuare\n\n\n\n";
		getchar();
	}
	return 0;
}
```

Il programma legge da un file di ingresso una stringa di caratteri esadecimali, ne controlla la correttezza e la lunghezza. Dopo di che la converte in binario ed inizializza la bitset.

La funzione computeOutputBitset è quella che, nel rispetto della relazione data all'interleaver, assegna i bit della bitset di ingresso alla bitset di uscita.

computeOutputBitset

```
bitset<1024> Tester::computeOutputBitSet(bitset<1024> inputBitSet)
{
	bitset<1024> outputBitSet;
	int index;
	bool val;

	for (int i = 0; i < 1024; i++)
	{
		index = i * 3;
		index += 45;
		while (index >= 1024)
		{
			index -= 1024;
		}

		/*value of the i-th bit in input*/
		val = inputBitSet.test(i);
		/*set the index-th bit of the output*/
		outputBitSet.set(index, val);

	}
	return outputBitSet;
}
```

Dopodiché il risultato calcolato viene stampato a video ed un file di output viene generato.

Se in ingresso si trovano più stringhe valide altre iterazioni vengono eseguite, altri dati vengono mostrati a video e nuovi file di output vengono salvati. Il programma termina quando l'EOF viene trovato.

Il programma c++ è stato scritto e testato utilizzando Microsoft Visual Studio 2013 su ambiente Windows.

2.3 Codice testbench

Per evitare di dover produrre a mano tutti i singoli 1024 dati in ingresso al dispositivo, il codice c++ è stato adattato così che uno o più file di testbench siano prodotti ogni qualvolta esso viene eseguito, prendendo come ingresso la stringa (o le stringhe) presente nel file di input.

Qui di seguito è riportato un esempio di file di testbench generato

```
-------------------------------------------------------------------------------
--Interleaver - TestBench
--
--File name : interleaver_test-2.vhdl
--
--Library : IEEE
--Author: Chiara Caiazza


LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE ieee.numeric_std.ALL;

ENTITY interleaver_test IS

END interleaver_test;

ARCHITECTURE interleaver_test_arch OF interleaver_test IS
	COMPONENT interleaver IS
		port(	clk : in std_logic;			--Processing clock
			reset : in  std_logic;			--Asynchronous active high reset
			bit_in : in std_logic;			--input bit
			bit_out : out std_logic			--outputbit
		);
	END COMPONENT interleaver;

	--CONSTANT
	CONSTANT clock_period : TIME := 200 ns;
	CONSTANT len : INTEGER := 2049;


	--INPUT SIGNALS
	SIGNAL clock : std_logic := '0';
	SIGNAL rst : std_logic := '1';
	SIGNAL bit_input : std_logic := '0';


	--OUTPUT SIGNALS
	SIGNAL bit_output : std_logic := 'Z';

	SIGNAL clock_cycle : INTEGER;
	SIGNAL testing: Boolean :=True;

	BEGIN
		I: interleaver PORT MAP(clk => clock, reset => rst, bit_in => bit_input, bit_out =>bit_output);

--Generates clk
		clock <=NOT clock AFTER clock_period/2 WHEN testing ELSE '0';

		--Runs simulation for len cycles
		proc_test: PROCESS(clock)
			VARIABLE count: INTEGER:= 0;
			BEGIN
				clock_cycle <= (count+1)/2;
				CASE count IS
					-- reset
```

```
                WHEN    0       =>      rst<='1';

                -- starting input
                WHEN    1       =>      bit_input<='1';rst<='0';
                WHEN    2       =>      bit_input<='1';
```

[I case intermedi hanno una sintassi simile al precedente con diversi valori per bit input dunque, per limitare il numero di linee di codice, saranno omessi da questa relazione]

```
                WHEN    1024    =>      bit_input<='0';

                -- starting output

                WHEN    1025    =>      bit_input<='Z';
```

[I case intermedi hanno una sintassi simile al precedente con diversi valori per bit input dunque, per limitare il numero di linee di codice, saranno omessi da questa relazione]

```
                WHEN    2048    =>      bit_input<='Z';
                WHEN    len     =>      testing<=False;
                WHEN    OTHERS  =>      NULL;
        END CASE;
        if clock = '1' and clock'event then
                count:= count + 1;
        END IF;

    END PROCESS proc_test;
END interleaver_test_arch;
```

3 Scelta dei valori di test

Per quanto riguarda la scelta dei valori di ingresso al programma c++, ho scelto di prelevarli in forma esadecimale da un apposito file, convertendoli poi nel corrispettivo binario.

L'uso di caratteri esadecimali permette di gestire meglio l'ingresso da sottomettere al programma di prova, risultando estremamente più maneggevole di una stringa da 1024 bit. Un file di testbench, per l'ingresso sotto analisi, viene automaticamente generato dalla classe stessa.

Input # 1: 7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

Come primo tentativo ho scelto un valore con un numero limitato di 1, per vedere come si comportava l'uscita. La scelta di utilizzare un numero di bit pari ad 1 tali che $X_{out}(i*3+45) = 0 \,|_{i*3+45>=1024}$ ha permesso di verficare che, dopo una traslazione iniziale di 45 bit pari a 0 (dal bit 0 al bit 44) l'uscita seguisse il pattern 100100100…

Input # 2:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

Come secondo input ho scelto altri bit pare ad 1 tali che $X_{out}(i*3+45) = 0 \,|_{i*3+45>=2048}$. In questo modo ho potuto verificare che nei primi 45 bit, che prima erano rimasti vuoti, sia presente il pattern 100100100… Nei bit invece ho potuto verificare la presenza del pattern 110110110110…

Input #3: tutti i bit pari ad 1.

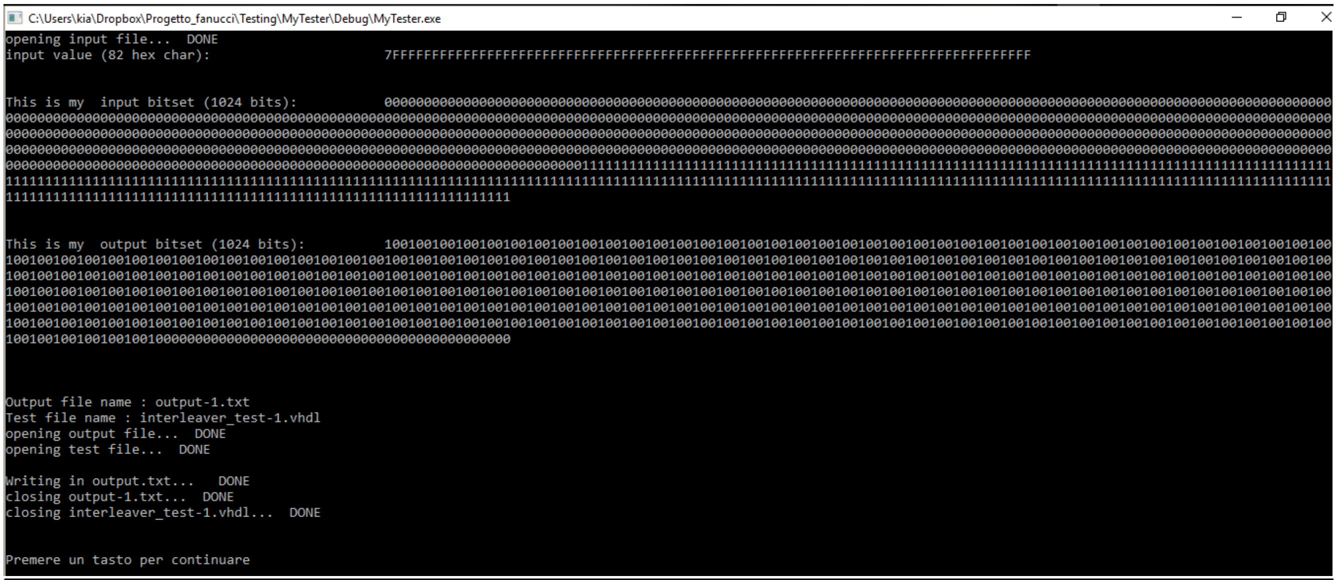
Input #4: tutti i bit pari ad 0.

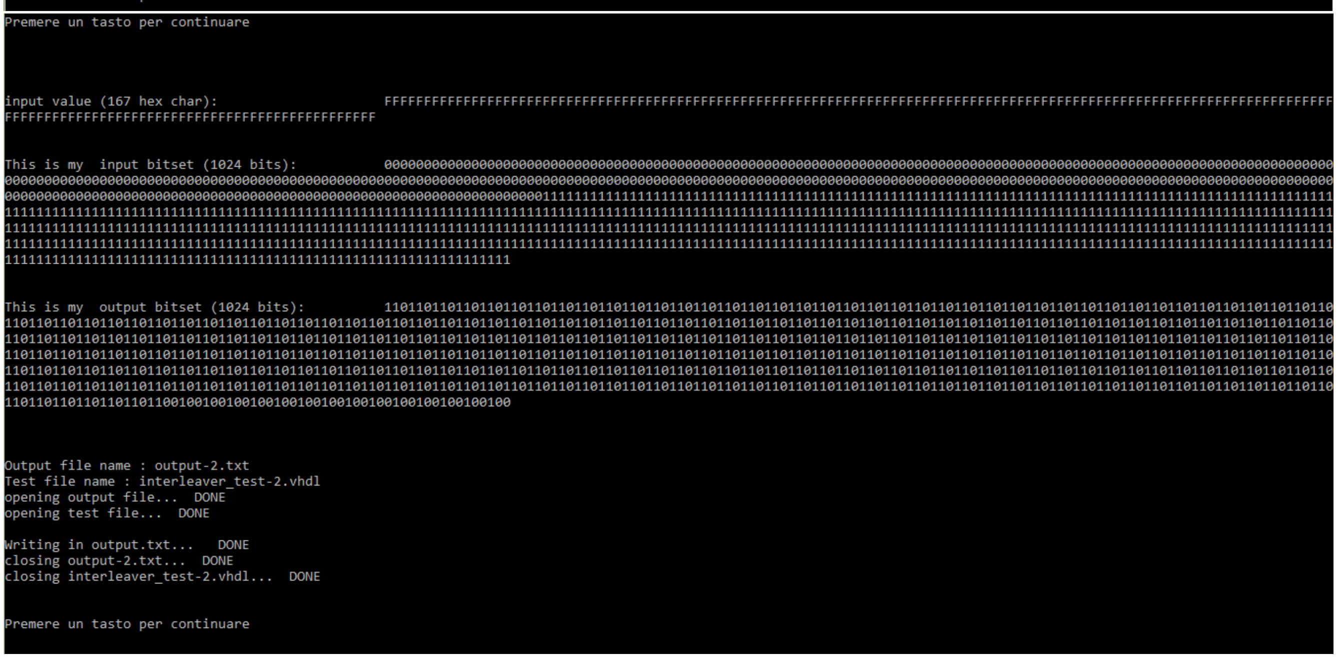
Con questa prova ho voluto verificare che anche il caso limite in cui tutti i bit sono pari ad 1 si comportasse nel modo sperato.

È stato inoltre verificato che con un numero maggiore di bit pari ad uno, così come con un carattere non decimale il codice c++ riporta un errore.

Tutti i file di testbench sono stati importati su Active-HDL per verificare il corretto funzionamento del codice VHDL.

3.1 Esempio di esecuzione in c++





```
Premere un tasto per continuare



input value (256 hex char):                     FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF


This is my  input bitset (1024 bits):           111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111


This is my  output bitset (1024 bits):          111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111



Output file name : output-3.txt
Test file name : interleaver_test-3.vhdl
opening output file...  DONE
opening test file...  DONE

Writing in output.txt...   DONE
closing output-3.txt...  DONE
closing interleaver_test-3.vhdl...  DONE


Premere un tasto per continuare
```

```
Premere un tasto per continuare



input value (1 hex char):                       0


This is my  input bitset (1024 bits):           000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000


This is my  output bitset (1024 bits):          000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000



Output file name : output-4.txt
Test file name : interleaver_test-4.vhdl
opening output file...  DONE
opening test file...  DONE

Writing in output.txt...   DONE
closing output-4.txt...  DONE
closing interleaver_test-4.vhdl...  DONE


Premere un tasto per continuare
```

```
Premere un tasto per continuare



input.txt -> EOF

Premere un tasto per continuare
```

4 Sintesi

Infine è stata effettuata la sintesi su Xinlix utilizzado l'ISE tool di cui riporto il summary.

interleaver Project Status (07/03/2016 - 12:28:39)			
Project File:	interleaver.xise	**Parser Errors:**	No Errors
Module Name:	interleaver	**Implementation State:**	Synthesized
Target Device:	xc7z010-3clg400	**• Errors:**	
Product Version:	ISE 14.7	**• Warnings:**	
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	1062	35200	3%
Number of Slice LUTs	2214	17600	12%
Number of fully used LUT-FF pairs	1059	2217	47%
Number of bonded IOBs	4	100	4%
Number of BUFG/BUFGCTRLs	1	32	3%

Detailed Reports					[-]
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	dom 3. lug 12:37:55 2016			

Infine allego due parti estratte dal report file della sintesi.

```
Device utilization summary:
---------------------------

Selected Device : 7z010clg400-3


Slice Logic Utilization:
 Number of Slice Registers:             1062  out of  35200     3%
 Number of Slice LUTs:                  2214  out of  17600    12%
    Number used as Logic:               2214  out of  17600    12%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:    2217
   Number with an unused Flip Flop:     1155  out of   2217    52%
   Number with an unused LUT:              3  out of   2217     0%
   Number of fully used LUT-FF pairs:   1059  out of   2217    47%
   Number of unique control sets:          2

IO Utilization:
 Number of IOs:                            4
 Number of bonded IOBs:                    4  out of    100     4%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:                 1  out of     32     3%

---------------------------
```

```
=========================================================================
Timing Report

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
clk                                | BUFGP                  | 1062  |
-----------------------------------+------------------------+-------+

Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design

Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 3.872ns (Maximum Frequency: 258.233MHz)
   Minimum input arrival time before clock: 1.166ns
   Maximum output required time after clock: 0.844ns
   Maximum combinational path delay: No path found
```