Journal of Engineering and Development, Vol. 13, No. 1, March (2009) **ISSN 1813-7822**

FPGA Implementation of Parallel and Pipelined Turbo Encoder for Real -Time Applications

Dr. Raghad. Z. Al-Macdici
Computer & Software Engineering Department, College of Engineering
Al-Mustansiriya University, Baghdad, Iraq

Abstract

The main keywords defining the quality of communication system are the data rate and the data transmission reliability. Error correcting codes are generally employed to achieve the reliability of the data transmission. The present trend is to achieve high data rates on low-cost designs (such as FPGAs), especially in real time application including multimedia transmission. Most of the time, parallel, architectures are required to process error correcting codes with high data throughput.

In this paper, an effective parallel architecture is proposed for the classical Turbo encoder based on parallel and pipelining designing of RSC encoder and the implementation of interleaver using register file. The simulation results shows that the data rates up to 142.918 M bits/s can be achieved on FPGA implementations.

الخـلاصــــــــة

يمثل معدل نقل المعلومات ووثوقية وصول المعلومات إلى المستلم بشكل صحيح عاملين أساسيين يحددان فاعلية أداء أي منظومة اتصالات التي تستخدم فيها مصححات الأخطاء لزيادة الوثوقية بوصول المعلومات بشكل صحيح . إن التوجه الحديث يتضمن الوصول إلى أعلى معدلات نقل البيانات باستخدام تقنيات مجال البوابات المنطقية المبرمجة الرخيصة التكلفة نسبيا والعالية المساحة والقليلة التأخير وخاصة إذا تحدثنا عن تطبيقات الزمن الحقيقي والتي تشمل نقل معلومات وسائط متعددة.ففي اغلب الأحيان يكون الهدف هو عمل تصميم متوازي لكي تتمكن المرمزات من معالجة المعلومات بسرعة عالية.

في هذا البحث تم تقديم نموذج محور للمرمز النفاث التقليدي بحيث إن المرمز الناتج متوازي المعالجة ومتعدد المراحل وذلك بإعادة تصميم النموذج التقليدي للمرمز RSC مع استخدام الـ register file لتصميم المبعثر بدلا من استخدام الذاكرة العادية . بالرغم من الزيادة في حجم المرمز المقدم إلا انه اثبت كفاءة عالية لنقل المعلومات بسرعة تجاوزت 142.918 M bit/sec على قطعة الـ FPGA المستخدمة.

1. Introduction

The standard on 3rd generation wireless communications, for instance, recommends increasing the data rates in order that the delay in which the data is processed remains small enough to be acceptable to the end-user [1,2]. Most of the time, error correcting codes are employed, Real-time processing of such codes when high data rates are considered may be a rather difficult task [2]. Different strategies can be considered to achieve the real-time constraint. Selecting a fast technology can be an adequate solution as it significantly reduces the design process complexity [3]. However, this strategy is not compatible with the usual low-cost constraint on end-user applications. Cost-effective designs usually require improved architectural designing techniques such as parallelization. Pipelining can be a very effective parallelization technique allowing high data rates on low-cost technologies such as FPGA devices (Field Programmable Gate Arrays) [3,4]. This strategy is adopted in this paper for the implementation of RCS (Recursive systematic Convolutional encoder) which is the constituent code in Turbo-encoder [1,5,6]. Therefore**,** the Turbo encoder is a parallel concatenation of two systematic convolutional encoders RSC1 and RSC2 as depicted in **Fig.(1)**.

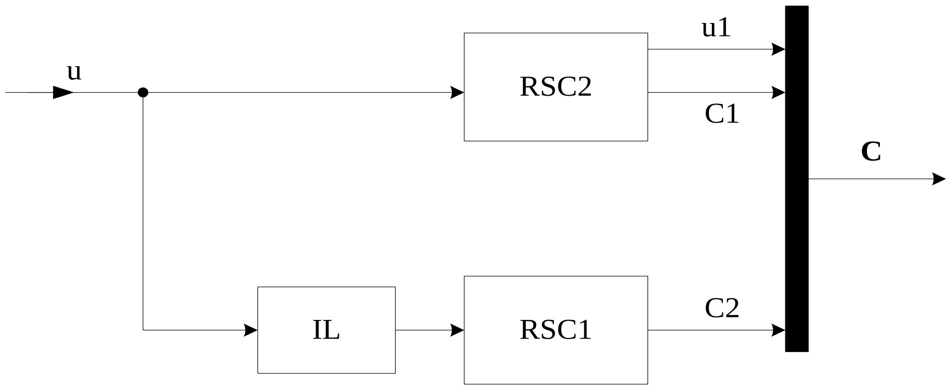

Figure (1) The block diagram of Turbo Encoder

The component encoder RSC2 is preceded by an interleaver (IL), which permutes the data sequence. Thus, both components encoders encode the same data apart from the order. The output from the encoder (C) is the concatenation of bit streams from the two component encoders encoded data is modulated and fed to the channel.

2. Recursive Systematic Convolutional Encoder RSC

In [1] a slightly complex structure was presented over traditional convolutional encoder (non-recursive systematic convolutional encoder NSC).This structure was called a Recursive Systematic Convolutional (RSC) encoder. It is made by including feedback in the encoder (in similar way to the finite impulse response filter IIR), because (NSC) is not systematic that is one of the outputs is not the input itself, the use of a NSC is also unacceptable because of the poor distance properties of the resulting code [1,5,6,7]. Hence, the unit weight input will always produce a low weight codeword at the input of the second encoder (for Turbo-Encoder). The RSC encoder is systematic and recursive since the state of the internal shift register depends on past outputs. Note that the interleaver has no effect on the weight distribution of the overall codeword in the case of NSC encoder .However the RSC (recursive systematic convolutional encoder) due to the IIR property could generates an infinite weight output codeword [1]. The influence of lower weight codeword is to reduce the free distance d_{free} which is the minimum Hamming distance between any two codewords, which leads to lower performance. For rate 1/2 RSC encoder, the input to the encoder at time k is a bit d_k

$$\mathbf{X_k = d_k} \quad \ldots\ldots\ldots \quad \mathbf{(1)}$$

The bit stored in the first delay element, a_k can be found using feedback polynomial g_{1i} and feed forward polynomial is g_{2i}. The feedback variable is given by:

$$\mathbf{a_k = d_k + \sum_{i=1}^{K} g_{1i} * a_{k-i} \bmod(2)} \quad \ldots\ldots\ldots \quad \mathbf{(2)}$$

and RSC encoder output Y_k which called parity data [1,3], is:

$$\mathbf{Y_k = \sum_{i=0}^{K} a_{k-i} * g_{2i} \bmod(2)} \quad \ldots\ldots\ldots \quad \mathbf{(3)}$$

For example let the input data 10011 to the RSC encoder depicted in **Fig.(2)** below with memory (m=2) and generator polynomials, G1=[111], and,G2=[101] or simply $(7,5)_{oct}$. **Table (1)**. Demonstrates the operation of RSC using successive clock periods when the encoder is started by all zero states.

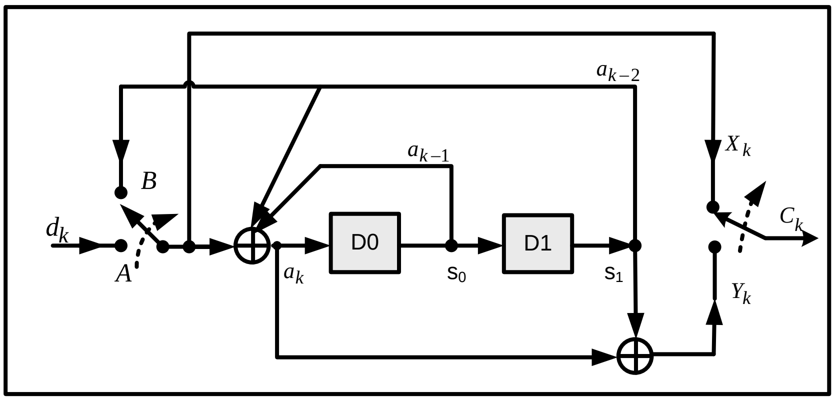

Figure (2) Recursive systematic convolutional (RSC) encoder

Table (1) RSC encoding process for an input sequence (10011)

d_k	a_k	S_0	S_1	X_k	Y_k
1	1	0	0	1	1
0	1	1	0	0	1
0	0	1	1	0	1
1	0	0	1	1	1
1	1	0	0	1	1

Due to the recursive structure of the RSC encoder it is not sufficient to set the last (m)-bits (memory elements) to zero to derive the encoder to all zero state (as the case of conventional convolutional encoder) [7]. Solving a state variable equation at the feedback elements bringing a RSC encoder to all zeros state. For example in the **Fig.(2)**, the state equation at the feedback element is:

$$\mathbf{a_k = d_k \oplus a_{k-1} \oplus a_{k-2}} \quad \ldots\ldots \quad \mathbf{(4)}$$

Solving this equation for d_k yield:

$$\mathbf{d_k = a_k \oplus a_{k-1} \oplus a_{k-2}} \quad \ldots\ldots \quad \mathbf{(5)}$$

Since we want to bring the encoder to all zeros state we set a_k in above to zero and thus:

$$\mathbf{d_k = a_{k-1} \oplus a_{k-2}} \quad \ldots\ldots \quad \mathbf{(6)}$$

This is the input to the encoder required to force the encoder into all-zeros states. Generally the most commonly used trellis termination was first presented in [7]. For the RSC encoder shown in **Fig.(2)**, includes a switch with two positions A and B, the switch is at position A for the first N-clock cycles (where N-is the frame length),and at position B for the additional m-cycles to derive the encoder to all-zero states and hence overcome the problem of termination. Two forms of RSC realization will be presented in the next section.

2-1 Serial Encoder

Figure (3) shows the canonical form of RSC encoder [3]. In this Figure, the encoder memory size, corresponding to the state width is m=3-bits. The following equations describe the encoder operation during one clock cycle:

$$\mathbf{R_{k+1} = R_k * F + (d_k, 0, \ldots, 0)} \quad \ldots\ldots \quad \mathbf{(7)}$$

$$\mathbf{Y_{k+1} = (y_0, \ldots\ldots, y_{n-1})_{k+1} = [(d_{k+1}, 0, 0, \ldots., 0) + R_k * G] * H} \quad \ldots\ldots \quad \mathbf{(8)}$$

Where (y_i) is parity bit , $R_k = (r_0 ,..., r_{m-1})_k$. The matrix H is equal to:

$$\mathbf{H} = \begin{pmatrix} \mathbf{h}_{0,0} & . & \mathbf{h}_{0,n-1} \\ \mathbf{h}_{1,0} & . & \mathbf{h}_{1,n-1} \\ . & . & . \\ \mathbf{h}_{m-1,0} & . & \mathbf{h}_{m-1,n-1} \end{pmatrix} \quad \textbf{(9)}$$

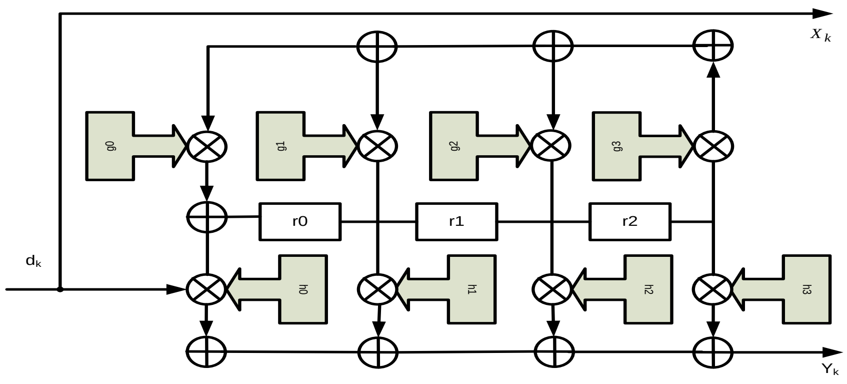

Figure (3) Serial RSC encoder

And matrix F is equal to:

$$\mathbf{F} = \mathbf{G} * \begin{pmatrix} 1 & 0 & . & 0 \\ 0 & 1 & . & 0 \\ . & . & . & . \\ 0 & 0 & . & 1 \\ 0 & 0 & . & 0 \end{pmatrix} \quad \textbf{(10)}$$

where, G is:

$$\mathbf{G} = \begin{pmatrix} g_0 & . & g_1 & 1 & . & 0 \\ g_0 & . & g_2 & 0 & . & 0 \\ . & . & . & . & . & . \\ g_0 & . & g_{m-1} & 0 & . & 1 \end{pmatrix} \quad \textbf{(11)}$$

In serial form of RSC encoder the value of g_0 is always set to one. If R_k is the value of register at time k which represents the state S_i then equation 7 calculates the next state S_{i+1} to be stored in R_{k+1}. Equation 8 calculates the current output Y_k .The serial encoder process a single bit per clock cycle. Consequently, the highest data rate that can be achieved is equal to the maximal clock frequency. One FPGA implementation, this frequency may be insufficient which the case is when the maximal clock frequency is lower than or equal to the device frequency. In order to improve speed without changing technology, it is necessary to design multiple pipeline architecture for the encoder.

2-2 Pipeline Parallel Implementation of RSC Encoder

Parallelization means that more than one input bits are processed in a single clock cycle [8]. Hence, the data rate is calculated as:

$$\mathbf{R_{data}=p*f_{clk}} \quad \ldots\ldots \quad \mathbf{(12)}$$

where, p is the number of input bits simultaneously applied at the input of the RSC encoder in one clock cycle (p-bit are input and p-bits are output). Two main tasks are implemented by classical RSC encoder presented in **Fig.(2)**:

1. Calculation of next state S_{i+1} from present state S_i.
2. Compute the output Y_k

These tasks also implemented by the proposed parallel design, but now p-states and p-output values are to be computed simultaneously. Thus the final generated state is called the future state. The (FSG) future state generation block calculates a future state S_{i+p} from initial state S_i and p-input bits as indicated in the **Fig.(4)**.

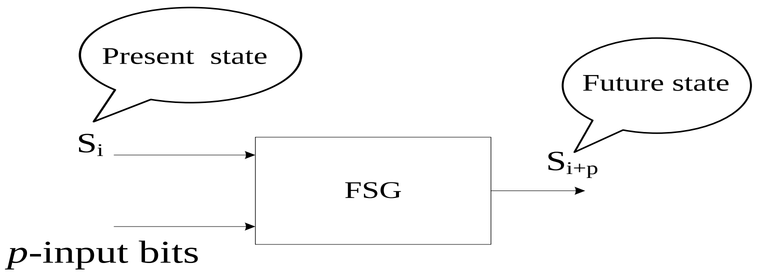

Figure (4) Future state generation block

State S_{i+p} is named future state because it is not the immediate state comes after state S_i like S_{i+1} but the encoder pass through a chain of p-intermediate states calculation internally until it reaches the state S_{i+p} . The mechanism of states generation through a parallel-pipelined approach is illustrated in the **Fig.(5)**.

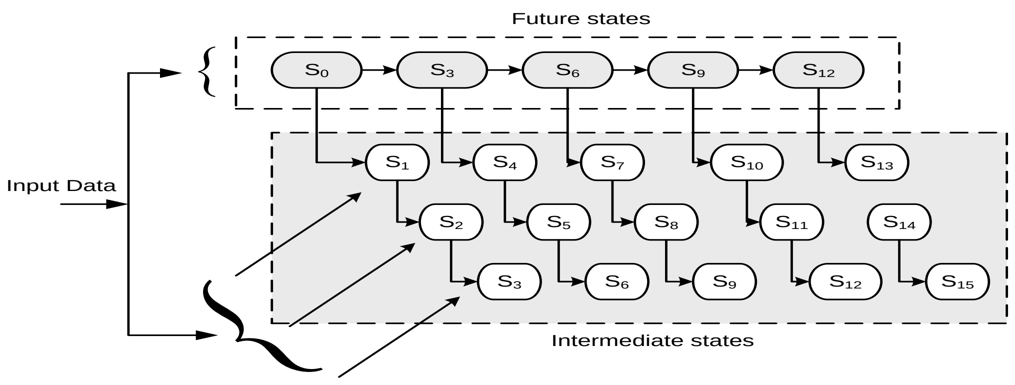

Figure (5) Mechanism of future states deduction p=3

Hence to go forward in deriving future state S_{i+p} ,the intermediate states must be held first, this is possible by construction of the state diagram for RSC encoder which is simply a finite state machine (FSM) [2,3]. For RSC encoder with (m=3) and generator polynomial of $(13,17)_{oct}$ the state diagram is as illustrated in **Fig.(6)**.

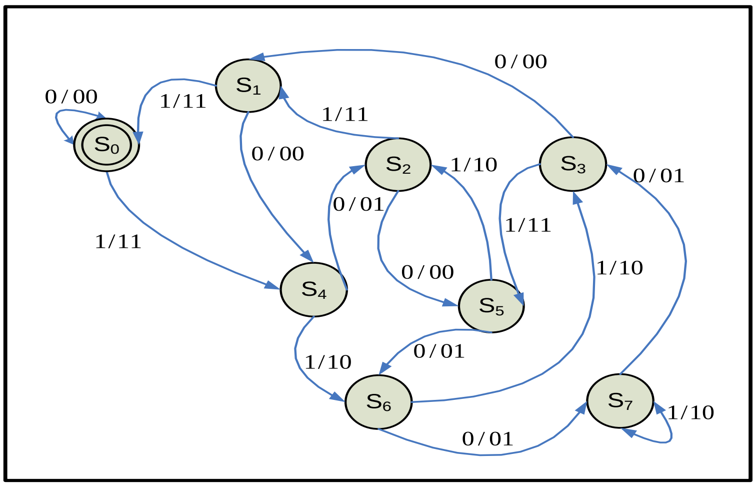

Figure (6) State diagram for $(13,17)_{oct}$ RSC encoder

Following the state diagram presented in **Fig.(6)** with p=3, if the present state is S_6=(110), and the input sequence is [001] then the future state is deduced through the following sequence of intermediate states tracking (the values above arrows are the input bits to the RSC encoder)

$$\begin{tikzcd} S_6 \arrow[r, "0"] & S_7 \arrow[r, "0"] & S_3 \arrow[r, "1"] & S_5 \end{tikzcd}$$

The future state is equal to S_5. In the same way the (FSG) future state generation block is constructed, and the complete results of calculating all possible future states is stored in it. **Table (2)** shows the content of FSG block for case of p=3.

Table (2) The content of FSG block (LUT) for case of p=3

Input / **Pstate**	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
S0	S0	S4	S2	S6	S5	S1	S7	S3
S1	S5	S1	S7	S3	S0	S4	S2	S6
S2	S7	S3	S5	S1	S2	S6	S0	S4
S3	S2	S6	S0	S4	S7	S3	S5	S1
S4	S6	S2	S4	S0	S3	S7	S1	S5
S5	S3	S7	S1	S5	S6	S2	S4	S0
S6	S1	S5	S3	S7	S4	S0	S6	S2
S7	S4	S0	S6	S2	S1	S5	S3	S7

The missing intermediate states S_{i+j} with j=1,…, p are generated through a pipeline from initial state S_i. of course, there is latency in the intermediate states generation path, with correct insertion of delays the p outputs can be picked simultaneously. The outputs are calculated from the intermediate states are generated by a p-stage pipeline as depicted in **Fig.(7)**, (b and c). The input bits are delayed through the shift register array **Fig.(7.a)**. Hence, one clock cycle in the parallel design computes the equivalent p clock in serial design. The state S_i stored in the register R_k in the stage k is generated (applying the equation 7) from state S_{i-1} in stage k+1 or in the FSG block according to S_{i-1} being or not another intermediate state. The corresponding input bits are taken from the shift register array of **Fig.(7.a)**. Hence each stage k generates an output parity bit from the corresponding intermediate state register R_k using equation 8.

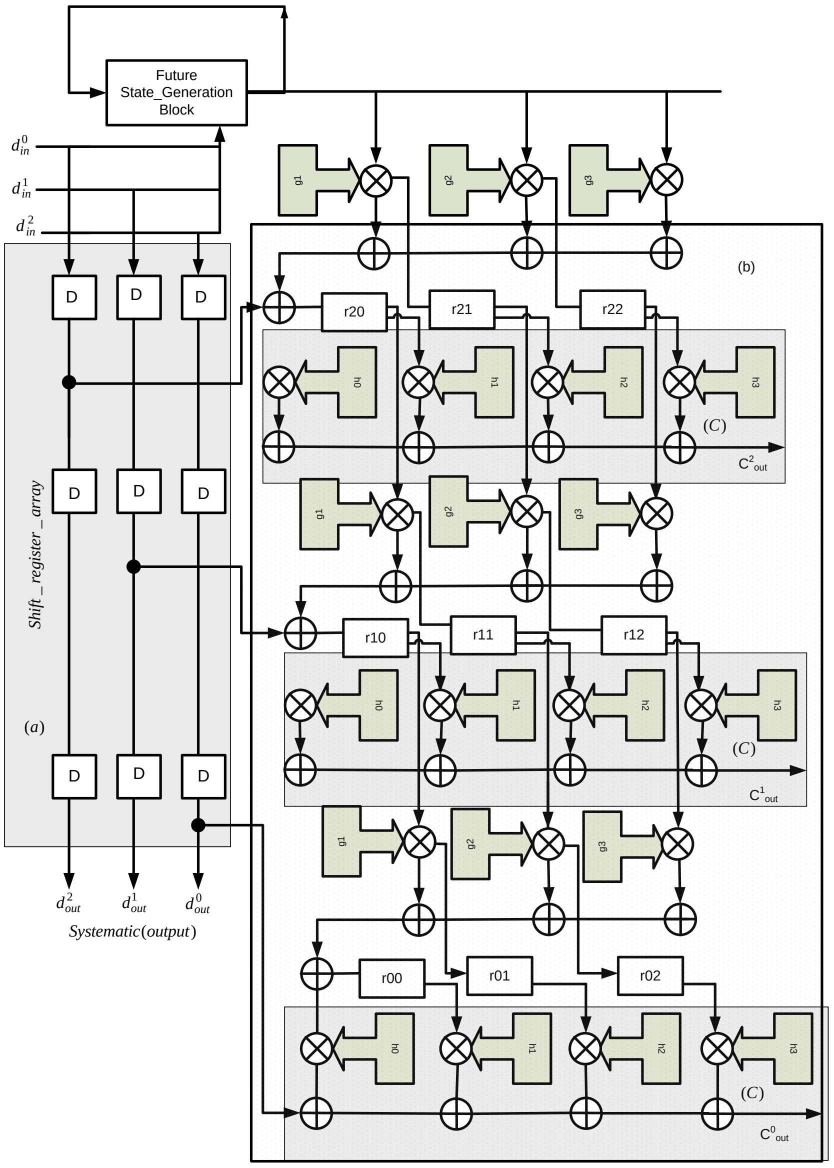

Figure (7) Parallel architecture for the RSC encoder with p=3

3. System Design Consideration

The function of the interleaver is to permute sequences according to static predefined pattern. The main objective with interleaver design is to find a solution that has good performance under the intended operating conditions. Generally well performing interleavers with good distance spectrums has random interleaving pattern [6,9]. The flow chart of the proposed random interleaver is given in **Fig.(8)**.

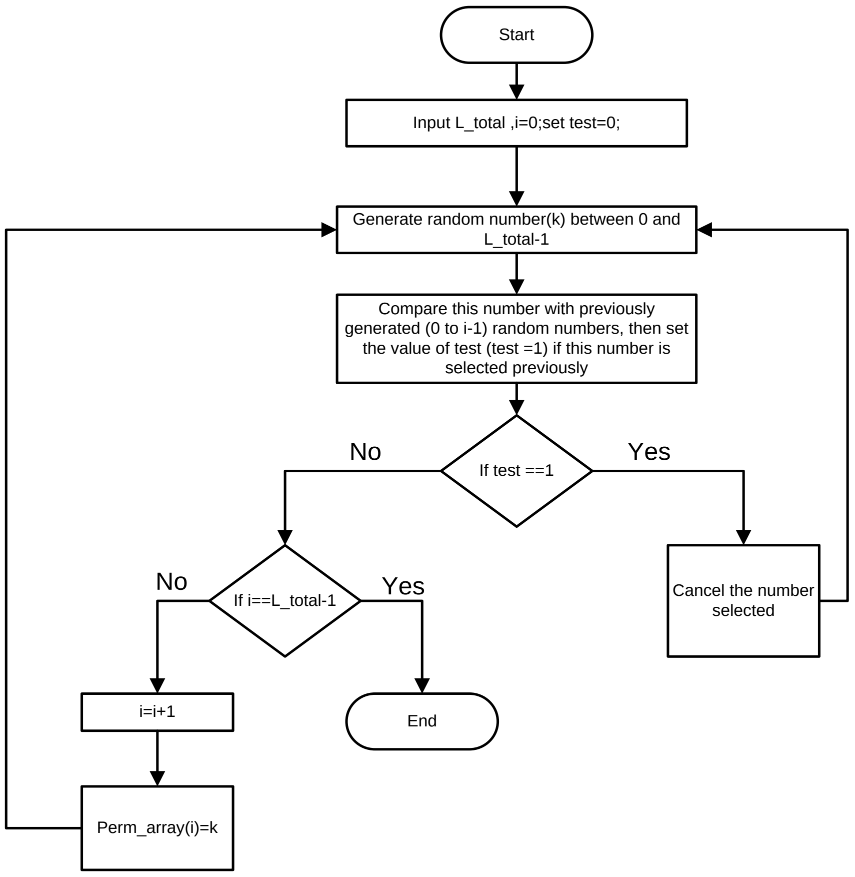
Start
Input L_total ,i=0;set test=0;
Generate random number(k) between 0 and L_total-1
Compare this number with previously generated (0 to i-1) random numbers, then set the value of test (test =1) if this number is selected previously
If test ==1
No
Yes
Cancel the number selected
If i==L_total-1
No
Yes
i=i+1
End
Perm_array(i)=k

Figure (8) Flow diagram of random interleaver

In this flow chart the variable test is used as flag, which is setted if the present selected random number between 0 and L_total-1 is generated previously. Hence, it is canceled and the program return to generate another random number, otherwise if the value of test =0 then the selected random number is stored in permutating array (perm_array). This operation continuous until the variable i=L_total-1, then the program is stopped. From the previous description of turbo encoder it is clear that the interleaver memory is associated with two different addresses, one natural order address and the other is the permutating order address. The natural order address is the output of a simple counter which needs no explanation, while the permutating order addresses are derived directly from the interleaving algorithm described previously using Lookup-table or (mapper) which map natural order input from counter to one of interleaving memory array locations. **Figure (9)** illustrates a block diagram of proposed random interleaver.

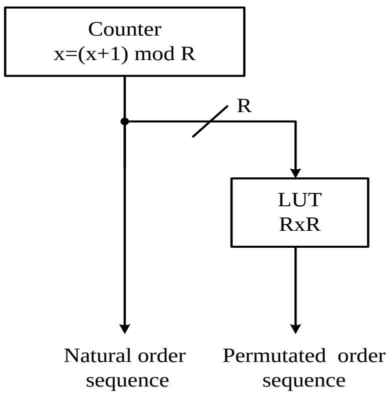

Figure (9) Block diagram of random interleaver

To get minimum delay at the output of the Turbo encoder, and to preserve parallelism obtained with redesigning RSC encoders, the permutated sequence and the natural order sequence must be introduced synchronously at the input of the two RSC encoders, for sake of that the register file circuit is suggested in the design of interleaver instead of a simple one part memory. The basic block diagram of 8-bit width register file cell is shown in **Fig.(10)**.

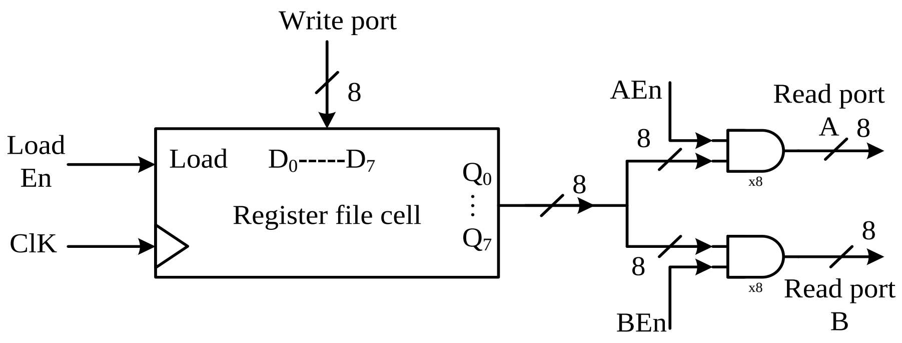

Figure (10) Circuit diagram of single cell in register file

Hence, instead we have single AND gate for single Q-output line (Q_i), we need to connect two AND gates to each output line, one for port A and the other for port B [10]. The read enable lines (AEn, BEn) are used to enable reading word from ports A and B respectively. For each read port, the enable signal is connected in common to one output of all the eight AND gates. The second input of the eight AND gates connect to the eight lines Q_0 to Q_7. For input frame of size 1024 byte we need (1024)x8-bit register file cells. In order to select which register file cell we want to access, three decoders are used to decode the addresses, WA_9……WA_0, RAA_9………RAA_0, RBA_9……RBA_0, one decoder is used for the write address and two decoders are used to read from different locations from the register file cells. Finally, the block diagram of proposed Turbo-encoder is shown in **Fig.(11)**. Note that the input bus width of the RSC encoder indicated in **Fig.(8)**. is 8-bit, not 1-bit width, this indicate that the RSC encoder used in the design of proposed Turbo-encoder is the parallel and pipelined RSC encoder, where the input bits are applied simultaneously at the input of each encoder.

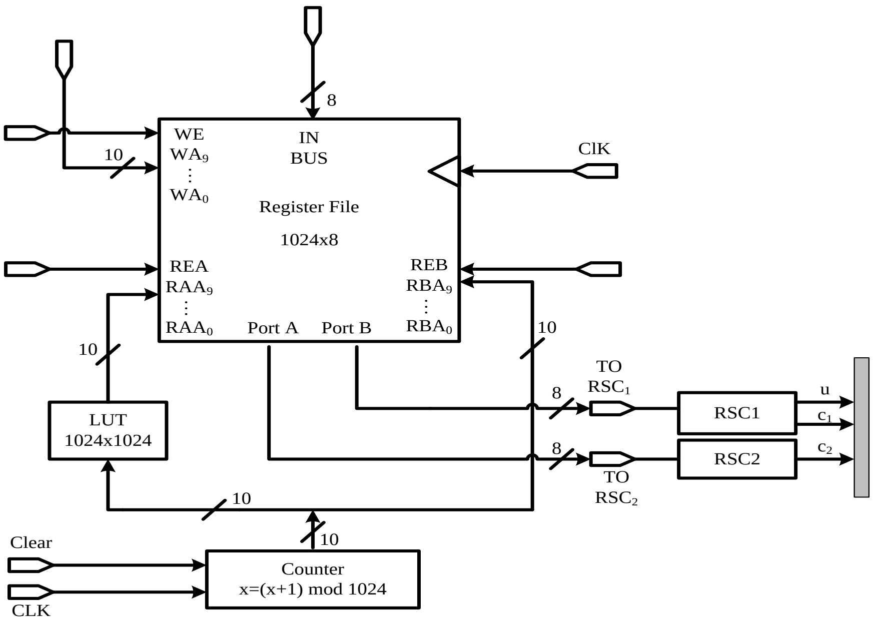

Figure (11) The proposed parallel and pipelined turbo-encoder architecture

4. Results

The architecture of proposed parallel and pipelined Turbo encoder has been synthesized from generic VHDL model (The polynomial generators and the p and m parameters can be selected at synthesis). The simulation and verification of the system consist of the basic elements of the Turbo-encoder circuit coded in VHDL, and a clock generator. The flow chart of the stages of design and verification of proposed system is depicted in **Fig.(12)**. The Modelsim is used for the VHDL simulator. The unit delayed functional simulation is performed for the pre-routed design to verify the internal operation of the Turbo-encoder circuit. After place-and-rout process, the Design pattern matching (DPM) is performed to compare the result and verified the logic as well as timings.

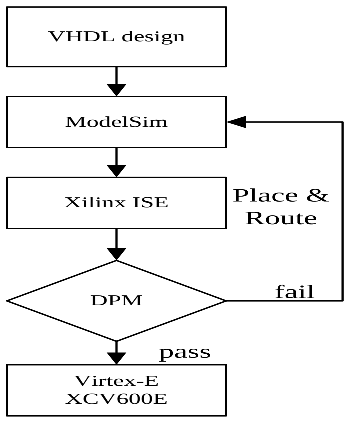

Figure (12) Design flow of proposed system to Xilinx FPGA

The design was synthesized, laid out and routed for specific target FPGA to estimate the size and speed of the Turbo-encoder circuit implemented in FPGA chip. Through the place and route process, the design are targeted and routed successfully into FPGA "Virtex-E XCV600E" [11]. Typical area and timing results from the device are shown in **Table (3)**.

Table (3) FPGA resource utilization for virtex-E

Number of GCLK	1 out of 4	25%
Number of External IOBs	27 out of 158	17%
Number of Slices	4666 out of 6912	67%
Total delay	6.997ns	

The data rate of the system is calculated by reciprocal of the total delay in **Table (3)**. Gving data rate of 142.918 M bits/s. Simulation waveforms are given in **Figs.(13a,b,c)** and **(14)** for $(13, 17)_8$ RSC encoder. The simulation clock in the ModelSim testbeanch is equal to 50ns. The critical path depends, on the implementation style (serial or pipelined), the value of p employed in simulation waveform is p=8. The latency is about p and 2p for respectively the systematic output (out_sys1, out_sys2) and the parity outputs (out_par1,out_par2) as compared with the input data, whereas the latency between the input to the first RSC (inp_intlv) and the input to the second RSC (out_intlv) is about 9ns, which is the time spend in interleaving the input data. The serial implementation of the RSC encoder is given in **Fig.(14)**, it is clear that the signals out_sys_s and out_par_s, which are the systematic and parity outputs from the encoder respectively, appears at the output with latency no greater than 25 ns, (faster that the case of proposed architecture of RSC) but the output is generated bit by bit, not byte by byte which is the case of proposed parallel architecture.

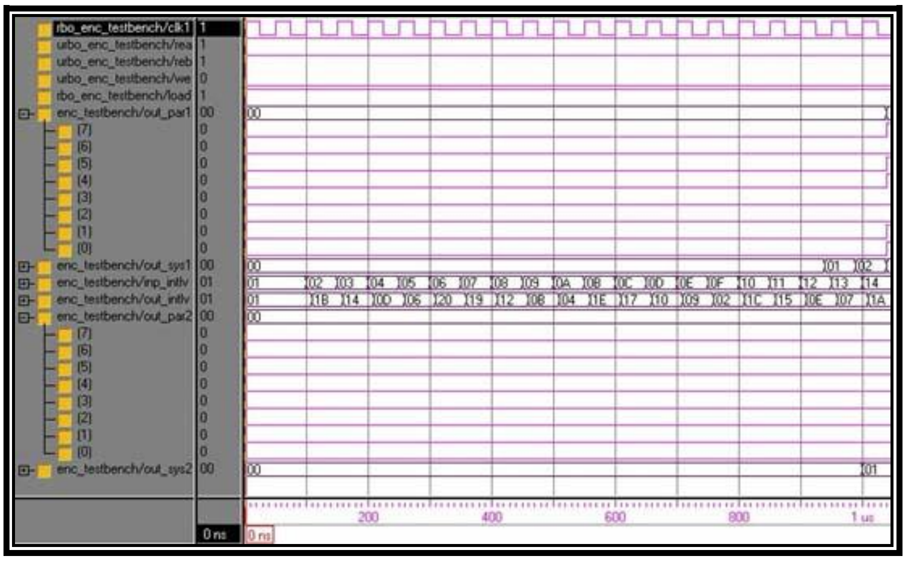

Figure (13a) Parallel and pipelined turbo encoder simulation results (part A)

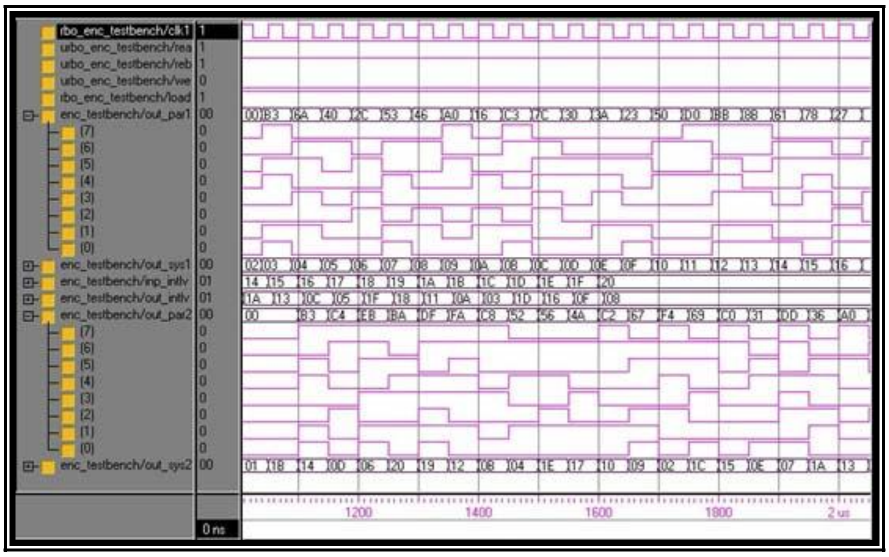

Figure (13b) Parallel and pipelined turbo encoder simulation results (part B)

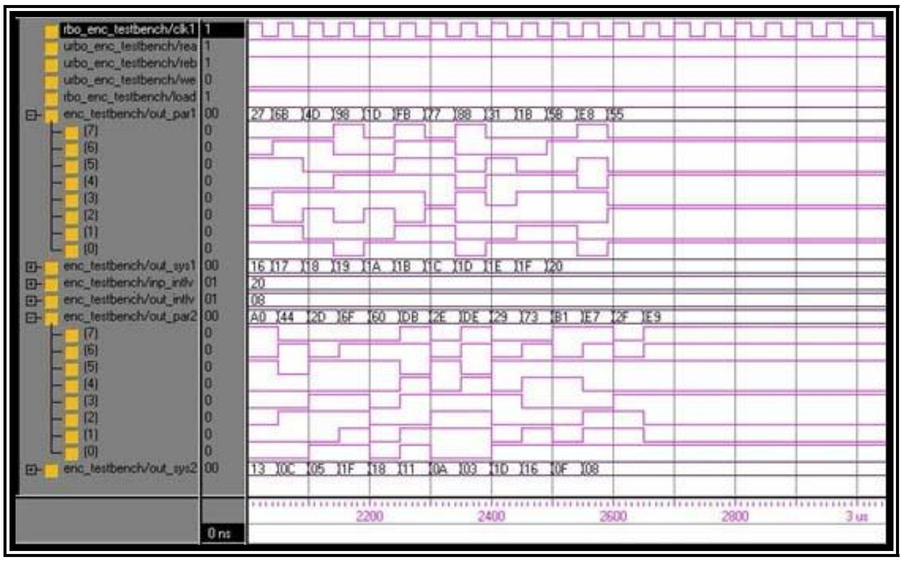

Figure (13c) Parallel and pipelined turbo encoder simulation results (part C)

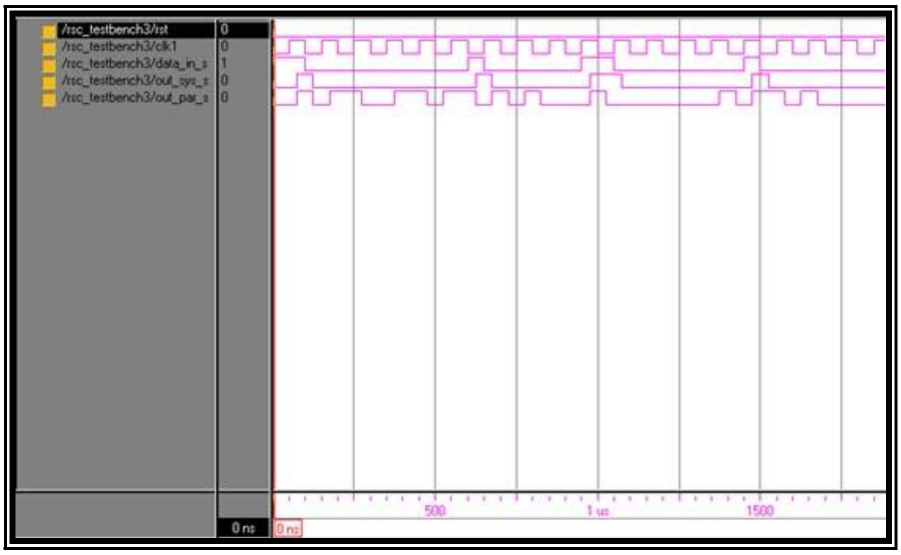
/rsc_testbench3/rst
/rsc_testbench3/clk1
/rsc_testbench3/data_in_s
/rsc_testbench3/out_sys_s
/rsc_testbench3/out_par_s
500
1 us
1500
0 ns

Figure (14) Serial RSC encoder simulation results

5. Conclusion

In this paper ,an Parallel and Pipelined Turbo-encoder Circuit using FPGA technology has been successfully implemented and its preliminary functionality verified, with the development of the VLSI technology, the price of FPGA is cheaper and cheaper. Using FPGA, we can achieve the higher data rates for the Turbo-encode. The architecture proposed in this paper, synthesized from a generic VHDL, allows designing 1/n rate RC systematic encoders, with the employing of register file for parallel and synchronous reading data to both encoders, as a result presenting high data rates (up to 142.918 M bits/s) on low-cost FPGA devices. The p and m parameters (parallelism level and memory width of the code) and the polynomial generators can be selected at synthesis. The FPGA resource utilization table for Virtex-E shows that the trade-off performance/area is favorable to the pipelined architecture which not takes large implementation size, where the size is influenced by the polynomial generator choice.

6. References

1. A., Glavieux, C., Berrou, and P., Thitimajshima, ***"Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes"***, In IEEE International Conference on Communications, Pages 1064-10701, May 1993.

2. Ibrahim, A., Al-Mohandes, ***"Energy Efficient Turbo Decoder for 3G Wireless Terminals"***, Ph.D. Thesis, University of Waterloo, Electrical and Computer Engineering Waterloo, Ontario, Canada, 2005

3. R. M., Bnakar, ***"A Low Power Design Methodology for Turbo Encoder and Decoder"***, Ph.D. Thesis, Dep. of Electrical Engineering, Indian Institute of Technology, Delhi India, July 2004.

4. F., Monteiro, A., Dandache, A., M'Sir, and B., Lepley, ***"A Fast CRC Implementation on FPGA using a Pipelined Architecture for the Polynomial Division"***, IEEE International Conference on Electronics, Circuits and Systems, St Julian, Malta, September 2-5, 2001.

5. J., Kaza, and C., Chakrabarti, ***"Design and Implementation of Low-Energy Turbo Decoders"***, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 12, No. 9, September 2004.

6. S., Benedetto, and G., Montors, ***"Concatenated Convolutional Codes with Interleavers"***, IEEE Communications Magazine, August 2003.

7. Matthew, C. Valenti, ***"Introduction to Turbo Codes"***, Virginia Polytechnic Institute, and State University Publications, 1996.

8. T., Vallino, S., Piestrak, A., Dandache, F., Monteiro, and B., Lepley, ***"Study of a New Parallel Architecture Dedicated to the Family of the DSCC Codes"***, IEEE International On-Line Testing Workshop, Rhodes, Greece, July 1999.

9. M. S. C., Ho, and S. S., Petrson, ***"Interleavers for Punctured Turbo Coders"***, Institute for Telecommunications Research, University of South Australia, 1999.

10. B., Castagnolo, and M., Rizzi, ***"High Speed Error Correction Circuit Based on Iterative Cell"***, International Journal on Electronics, Vol. 14, No. 4, 1993, pp. 529-540.

11. XILINX, ***"The Programmable Logic-Data Book 2000"***.

Appendix (A)

(VHDL program for FSG block):

```
library IEEE;
use ieee.Std_logic_1164.all;
entity  FSG  is
port(Pstate:in  std_logic_vector(2downto0);Input_seq  :in  std_logic_vector(2  down  to  0)
;ClK:in std_logic , Future_state:out  std_logic_vector(2downto0));
   End FSG;
Architecture behavioral of FSG is
  begin
    process(CLK)
Variable control : std_logic;
        begin
         if CLK'event and CLK='1'then
Control =>Pstate & Input_seq
Case (Control)
           .
           .
           .
When 000 001 =>Future_state<=100 after t ns;
When 001 001=>Future_state<= 001 after t ns;
           .
           .
           .
End Case  ;
end if;
End process; End behavioral;
```