# Chip Design for Turbo Encoder Module for In-Vehicle System

Majeed Nader
Electrical and Computer Engineering
Wayne State University
Detroit, Michigan
Email: majeed@wayne.edu

Yunrui Li
Electrical and Computer Engineering
Wayne State University
Detroit, Michigan
Email: yunrui.li@wayne.edu

John Liu
Electrical and Computer Engineering
Wayne State University
Detroit, Michigan
Email: john.liu@wayne.edu

***Abstract*—This paper studies design and implementation of the Turbo encoder to be an embedded module in the in-vehicle system (IVS) chip. Field programmable gate array (FPGA) is employed to develop the Turbo encoder module. Both serial and parallel computations for the encoding technique are studied. The two design methods are presented and analyzed. Developing the parallel computation method, it is shown that both chip size and processing time are improved. The logic utilization is enhanced by 73% and the processing time is reduced by 58%. The Turbo encoder module is designed, simulated, and synthesized using Xilinx tools. Xilinx Zynq-7000 is employed as an FPGA device to implement the developed module. The Turbo encoder module is designed to be a part of the IVS chip on a single programmable device.**

***Index Term:* Turbo encoder module; field programmable gate array; emergency call; in-vehicle system chip.**

## I. Introduction

The European emergency call (eCall) system is a telematics system designed to save more lives in vehicle accidents. It is a governmental mandatory system that is to be implemented by March 2018 [1][2]. The EU eCall system provides an immediate voice and data channel between the vehicles and an emergency center after car accidents. The data channel provides the emergency center with the necessary data for emergency aids.

The EU eCall system main parts includes the in-vehicle system (IVS), the public safety answering point (PSAP), a cellular communication channel. The IVS activates the data channel automatically when a car accident occurs. The IVS collects the minimum set of data (MSD) that includes GPS coordinates, the VIN number, and all required data for an emergency aid. It sends the MSD to the closest PSAP through a cellular channel in up to 4 seconds [1]. The PSAP sends the emergency team to the location of the accidents.

The IVS modem employs multiple modules for the MSD signal processing. The modules of the IVS are shown in Figure 1. The IVS employs a Turbo encoder as a forward error correcting (FEC) [1]. The Turbo encoder implements the digital data encoding technique in data transmissions. Turbo coding is one of the most popular and efficient coding technique to improve bit error rate (BER) in digital communications [3] [4]. The cyclic redundancy check (CRC) [5], the modulator [6], the demodulator-decoder [7] modules are projected and implemented on an FPGA device. They are developed to be embedded modules of the IVS chip.

978-1-5386-4649-6/18/$31.00 

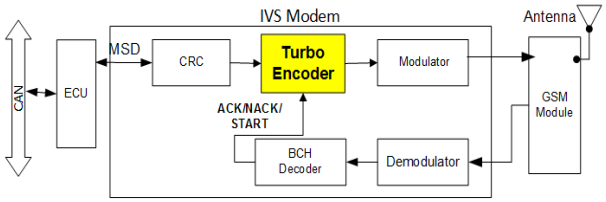


Fig. 1: The IVS block diagram.

This work studies the hardware development of the Turbo encoder. It employs FPGA technologies to develop the Turbo encoder to be an embedded module in the IVS modem. It discusses serial and parallel computation techniques for the Turbo encoder. It does not only design and implement the Turbo encoder module, but also proposes a better solution for the turbo encoder implementation. The improvement of the chip size and processing time are exhibited by developing the parallel computation technique for the Turbo encoder.

## II. Turbo Encoder Module

The turbo encoder technique is one of the most powerful FEC techniques in digital communication [8]. The IVS employs a Turbo encoder module with $1/3$ code rate. The Turbo encoder functionalities are detailed in the third generation partnership project (3GPP) standards. The 3GPP Turbo encoder is illustrated in Figure 2 [8]. The input signal of the turbo encodes is the MSD data appended with the CRC parity bits in binary. The block length of the MSD data is 1148 bits. The output of the module is the MSD encoded data in binary. Implementing the turbo coding technique with $1/3$ coding rate and thrills bits, the length of the output is 3456 bits. The thrills structure has an impact of the Turbo encoder [9].

The Turbo encoder employs a parallel concatenated convolutional code (PCCC). The PCCC uses two constituent encoders with eight states as it is shown in Figure 2. The initial status of the register are zeros. The first constituent takes the MSD bits and implements the employed convolutional technique. It takes one bit at a time and generates one bit of

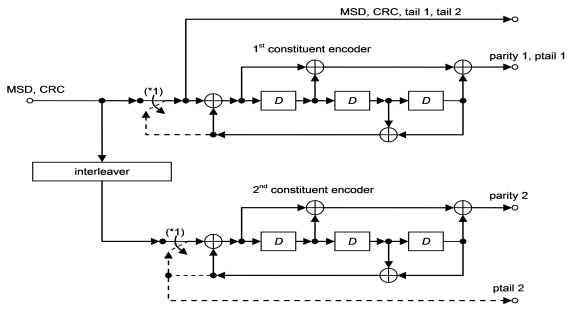


Fig. 2: The structure of the Turbo encoder.

parity1 bits. The second constituent implements an identical technique of the first constituent, but it calls for the MSD bit after they are interleaved with a 3GPP designed interleaver technique [8].

The length of the input data, parity1, and parity2 are 1148 bits. There are 12 bits of the tail bits. They are driven from the shift register feedback. The tail bits are applied for end points between the encoded data blocks. The output structure of the Turbo encoder is illustrated in Figure 3.

| MSD+CRC | $tail_1$ | $tail_2$ | Parity 1 | $ptail_1$ | Parity 2 | $ptail_2$ |
|---|---|---|---|---|---|---|

Fig. 3: The output buffer of the Turbo encoder.

*Interleaver*

Denote the transfer function of the employed PCCC as:

$$G(D) = \left(1, \frac{g_1(D)}{g_0(D)}\right) \tag{1}$$

where

$$g_(D) = 1 + D^2 + D^3$$

$$g_(D) = 1 + D + D^3$$

And denote the input bits to the encoder as $x_1, x_2, ..., x_K$, the output of the interleaver as $x'_1, x'_2, ..., x'_K$, and the output bits of the first and second constituents as $z_1, z_2, ..., z_K$ and $z'_1, z'_2, ..., z'_K$, respectively; where $K$ is the number of input bits to the Turbo encoder.

The encoder output is expressed as:

$$d_K^{(0)} = x_K, d_K^{(1)} = z_K, d_K^{(2)} = z'_K$$

where $K = 0, 1, ..., K-1$.

The three code blocks of the output, $d_K^{(0)}$, $d_K^{(1)}$, and $d_K^{(2)}$ are separated by trellis bits. The trellis bits are generated from the tail bits of the shift registers after encoding of all the input bits. In figure 2, when the upper switch is lowered and the second constituent is disabled, the three tail bits are used to terminate the first constituent. The output bits of the Turbo encoder, including the trellis bits can be expressed as:

$$d_K^{(0)} = x_K, \ d_{K+1}^{(0)} = z_{K+1}, \ d_{K+2}^{(0)} = x'_K, \ d_{K+3}^{(0)} = z'_{K+1}$$

$$d_K^{(1)} = z_K, \ d_{K+1}^{(1)} = x_{K+2}, \ d_{K+2}^{(1)} = z'_K, \ d_{K+3}^{(1)} = x'_{K+2}$$

$$d_K^{(2)} = x_{K+1}, d_{K+1}^{(2)} = z_{K+2}, d_{K+2}^{(2)} = x'_{K+1}, d_{K+3}^{(2)} = z'_{K+2}$$

where $K = 0, 1, ..., K-1$.

The internal interleaver of the 3GPP Turbo encoder is designed to generate a systematic relationship between $x_K$ and $x'_K$ for any $40 \leq K \leq 5114$ [8]. There is a specific approach to design an internal interleaver for the employed Turbo encoder that is detailed in [8]. This work employs the 3GPP standard approach to design the internal interleaver for the employed Turbo encoder.

First, the input bits of the Turbo encoder is re-arranged in a matrix form that has column, C, and row, R. The rows are labeled as $0, 1, ..., R-1$ and the columns are organized as $0, 1, ..., C-1$. The numbers of rows and columns are determined according to the 3GPP standard for Turbo encoder interleavers [8].

Then the input bits $x_1, x_2, ..., x_K$ are re-organized in a matrix where $y_k = x_k$ for $k = 1, 2, ..., K$, and $y_k = 0$ for the elements that $R \times C > K$:

$$\begin{bmatrix} y_1 & y_2 & y_3 & \cdots & y_C \\ y_{(C+1)} & y_{(C+2)} & y_{(C+3)} & \cdots & y_{(C+C)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_{((R-1)C+1)} & y_{((R-1)C+2)} & y_{((R-1)C+3)} & \cdots & y_{(R\times C)} \end{bmatrix}$$

Then an intra-row and inter-row permutation is performed on the $R \times C$ matrix. This work employs the 3GPP standard approaches for the intra-row and inter-row.

Denote,

$$s(j) = (v \times s(j-1)) \mod p \tag{2}$$

where $\left\langle s(j) \right\rangle$ for $j \in 1, 2, ..., p-2$ and $p(0) = 0$ is the inta-row permutation sequence and $v$ is associated primitive root for the specified $p$ from table **??**, and use table I to choose the appropriate pattern to compute the inter-row permutation, $\left\langle T(i) \right\rangle$ for $i \in 0, 1, ..., R-1$. $i$ is the row number index of $R \times C$ matrix, and $j$ is the column number index of the matrix.

Also the minimum prime integer $(q_i)$ is determined in the sequence $\left\langle q(i) \right\rangle$ for $i \in 0, 1, ..., R-1$ such that $q_i > q_(i-1)$, $q_i > 6$ and g.c.d$(q_i, p-1) = 1$, where g.c.d is the greater common divisor.

Then one can build a sequence of the permuted prime integers $\left\langle r(i) \right\rangle$ for $i \in 0, 1, ..., R-1$ such that,

$$r_T(i) = q_i, i = 0, 1, ..., R-1$$

TABLE I: 3GPP inter-row permutation pattern

| $K$ | $R$ | Inter-row permutation patterns <$T(0)$, $T(1)$, ..., $T(R-1)$> |
|---|---|---|
| $(40 \leq K \leq 159)$ | 5 | <4, 3, 2, 1, 0> |
| $(160 \leq K \leq 200)$ or $(481 \leq K \leq 530)$ | 10 | <9, 8, 7, 6, 5, 4, 3, 2, 1, 0> |
| $(2281 \leq K \leq 2480)$ or $(3161 \leq K \leq 3210)$ | 20 | <19, 9, 14, 4, 0, 2, 5, 7, 12, 18, 16, 13, 17, 15, 3, 1, 6, 11, 8, 10> |
| $K$ = any other value | 20 | <19, 9, 14, 4, 0, 2, 5, 7, 12, 18, 10, 8, 13, 17, 3, 1, 16, 6, 15, 11> |

Denote the pattern of $i-th$ row Intra-row permutation as,

$\left\langle U_i(j) \right\rangle$for $i \in 0, 1, ..., R-1$,
one can perform the intra-row permutation such that the position of the $i-th$ permuted bit of the $j-th$ row ($U_i(j)$) is calculated based on equation 3, 4, or 5.

if $(C = p)$ then,

$$U_i(j) = s((j \times r_i) \mod (p-1)) \tag{3}$$

where $j = 0, 1, ..., (p-1)$ and $U_i(p-1) = 0$.

if $(C = p+1)$ then,

$$U_i(j) = s((j \times r_i) \mod (p-1)) \tag{4}$$

where $j = 0, 1, ..., (p-1)$, $U_i(p-1) = 0$, and $U_i(p) = p$.

if $(C = p-1)$ then,

$$U_i(j) = s((j \times r_i) \mod (p-1)) - 1 \tag{5}$$

where $j = 0, 1, ..., (p-1)$.

And then the inter-row permutation is implemented on the $R \times C$ matrix by using the sequence pattern $\left\langle T(i) \right\rangle$for $i \in 0, 1, ..., R-1$.

After the permutations, the elements of the $R \times C$ matrix is denoted by $y'_k = y_k$ such that:

$$\begin{bmatrix} y'_1 & y'_{(R+1)} & y'_{(2R+1)} & \cdots & y'_{((C-1)R+1)} \\ y'_2 & y'_{(R+1)} & y'_{(2R+2)} & \cdots & y'_{((C-1)R+2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y'_R & y'_{(2R)} & y'_{(3R)} & \cdots & y'_{(R\times C)} \end{bmatrix}$$

The output of the interleaver, $x'_2, x'_2, ..., x'_K$, is the bit sequence that are read out from the $R \times C$ matrix column by column starting with $y'_1$ and ending with $y'_{(R\times C)}$. The appended zero bits for $R \times C > K$ are removed from the output.

There are 1148 elements in the interleaver matrix. The 1148 bits of the MSD data are the elements of the interleaver matrix. The interleaver reorganizes the MSD bits in a systematic order. The intra-row and inter-row techniques of the interleaver elements organizations.

Denote the MSD bits as $B_1, B_2, ..., B_K$, where $K = 1148$. According to the algorithm that is explained in the above mathematical modeling, the interleaver matrix is a rectangular matrix, where R is the number of rows and C is the number of columns of the matrix. The interleaver matrix is designed for an input data block that consists of 1148 bits. As a result, the size of the matrix is 20 X 58. The following steps are implemented to drive the interleaver matrix:

1. The input bits of the matrix are denoted as $b_1, b_2, ..., b_K$, where $b_K = B_K$ and $K = 1148$. The remaining elements are padded with zeros.

2. The intra-row and inter-row permutation are performed according to 3GPP.

3. The calculated elements of the interleaver matrix, except for the padded bits, are stored in a file in hexadecimal format.

The Turbo encoder is developed in Verilog HDL language. Verilog has ability to read the hexadecimal file to get the data and use it as the interleaver data.

The length of the input data, parity1, and parity2 are 1148 bits. There are 12 bits of the tail bits. They are driven from the shift register feedback. The tail bits are applied for end points between the encoded data blocks. The output structure of the Turbo encoder is illustrated in Figure 3.

The employed interleaver for the Turbo encoder is designed according to 3GPP standards [8]. The interleaver elements are organized in a rectangular matrix. There are 1148 elements in the interleaver matrix. The 1148 bits of the MSD data are the elements of the interleaver matrix. The interleaver reorganizes the MSD bits in a systematic order. There are intra-row and inter-row techniques of the interleaver elements organizations.

Denote the MSD bits as $B_1, B_2, ..., B_K$, where $K = 1148$. According to the 3GPP standards [8], the interleaver matrix is a RxC rectangular matrix, where R is the number of rows and C is the number of columns. The size of the matrix is 20 X 58. The following steps are implemented to drive the interleaver matrix:

1. The input bits of the matrix are denoted as $b_1, b_2, ..., b_K$, where $b_K = B_K$ and $K = 1148$. The remaining elements are padded with zeros.

2. The intra-row and inter-row permutation is performed according to 3GPP [8].

3. The calculated elements of the interleaver matrix, except for the padded bits, are stored in in a file in hexadecimal format.

The Turbo encoder is developed in Verilog HDL language. Verilog has ability to read the hexadecimal file to get the data and use it as the interleaver data.

## III. FPGA Design for the Turbo Encoder Module

FPGA technologies are employed to develop and implement the designed Turbo encoder module. The register transfer level (RTL) of the module is developed in Verilog HDL. There are multiple registers defined for the input, output, and necessary parameters to implement the Turbo encoding technique. This work studies two methods to execute the encoding, which are serial computation and parallel computation.

The serial computation method processes one bit in one clock cycle. It reads the input data of the MSD, builds the input and output registers, and calculates the parity1, parity2, and the tail bits in a serial process. After performing the encoding, it generates the output bits. Although the method is designed and implemented, it is noted that there is a long processing time that can be overlapped with the other processes in the module. Figure 4 shows the pseudocode of the serial computation of the Turbo encoder module.

The parallel computing technique is employed to develop the Turbo encoder in Verilog. There are many processes in the serial computation technique that are overlapped by using parallel computing technique. There are two functions developed in the parallel Turbo encoder. The two functions implements almost all the processing time of the encoding technique. The Turbo encoding technique needs the MSD data as a whole package to implement the encoding. Figure 5 shows the pseudocode of the implemented parallel technique

```
Pseudocode code for Serial Computation of Turbo encoder
module TURBO_SERIAL ( inputs, outputs;)
        define REGISERS and PARAMETERS;
      always @(posedge clock, posedge reset)
        begin
         if (reset) output=0;
         else begin
            repeat1 (1148) {
                      Read MSD input data}
           If (repeat1 is done)
              repeat2 (1148) {
                      Build output register for MSDinput;
                      Build output register for Parity1;
                      Build output register for Parity2; }
           If (repeat2 is done)
              repeat3 (1148) {
                      Process Parity1 bits; }
           If (repeat3 is done)
              Repeat4 (3) {
                      Process tail1 bits;
                      Process ptail1 bits;    }
           If (repeat4 is done)
              Repeat5 (1148) {
                      Process Parity2 bits; }
           If (repeat5 is done)
              Repeat6(3) {
                      Process tail2 bits;
                      Process ptail2 bits;    }
           If (repeat6 is done)
              Repeat7(3456) {
                      Generate output bits    }
            end  end
endmodule;
```

Fig. 4: The pseudocode for serial computation of the Turbo encoder.

for the Turbo encoder. Both Pseudocodes of the serial and parallel computing techniques are designed in Verilog and implemented on an FPGA device.

The processing time of the Turbo encoder module (in clock cycles) is denoted by $T_s$ for the serial computation and by $T_p$ for the parallel technique, one has,

$$T_s = T_r + T_b + T_{parity1} + T_{tail1} + T_{parity2} + T_{tail2} + T_w \quad (6)$$

$$T_s = 1148 + 1148 + 1148 + 3 + 1148 + 3 + 3456 = 8054$$

where $T_r$ is the time for reading the 1148 bits of the MSD, $T_b$ is the processing time to build the output register, $T_{parit1}$ is the time of processing parity bits, $T_{tail}$ is the time of processing tail bits, and $T_w$ is the time of generating output bits.

Note that the $T_r + T_b + T_{parity1} + T_{tail1} + T_{parity2} + T_{tail2}$ are processed in one clock cycle in the parallel computing technique.

$$T_p = 1 + T_w = 1 + 3456 = 3457 \quad (7)$$

Then one has,

$$T_p = 0.42 T_s \quad (8)$$

Eq. 8 reveals that the parallel computing can improve the proceeding time of the Turbo encoder by 58%.

### A. Simulation and Verification

Xilinx tools are utilized to simulate the developed modules. A test bench is designed to simulate the Turbo encoder. Verilog

```
Pseudocode code for Parallel Computation of Turbo encoder
module TURBO_PARALLEL ( inputs, outputs;)
        define REGISERS and PARAMETERS;
 function [3455:0]  codedMSD1;
              input [1147:0] MSDdata;
                 begin
                    repeat1 (1148) {
                       Build output register for MSDinput;
                       Build output register for Parity1;
                       Build output register for Parity2; }
                   If (repeat1 is done)
                      repeat2 (1148) {
                       call  CodedMSD2 (codedMSD1) }
                  end
       endfunction
 function [3455:0]  codedMSD2;
              input [3455:0] codedMSD1;
                 begin
                 repeat3 (1148) {
                       Process Parity1 bits; }
                  If (repeat3 is done)
                    repeat4 (3) {
                       Process tail1 bits;
                       Process ptail1 bits;    }
                If (repeat4 is done)
                   repeat5 (1148) {
                       Process Parity2 bits; }
               If (repeat5 is done)
                   repeat6(3) {
                       Process tail2 bits;
                       Process ptail2 bits;    }
                  end
       endfunction
   always @(posedge clock, posedge reset)
      begin
       if (reset) output=0;
       else begin
          repeat1 (1148) {
                    Read MSD input data    }
             If (repeat1 is done) {
                call  CodedMSD1 (MSDdata)   }
             Repeat7 (3456) {
                    Generate output bits    }
           end  end
endmodule;
```

Fig. 5: The pseudocode for parallel computation of the Turbo encoder.

HDL is employed to design the test bench. There are two MSD data that are simulated for each of the serial computation and parallel computation of the Turbo encoder. Figure 6 shows the simulation results for the serial computation of the Turbo encoder module. The simulation indicates that the structure output data is correct. However, there is a long time that can be overlapped, which is colored in red in the output trace.

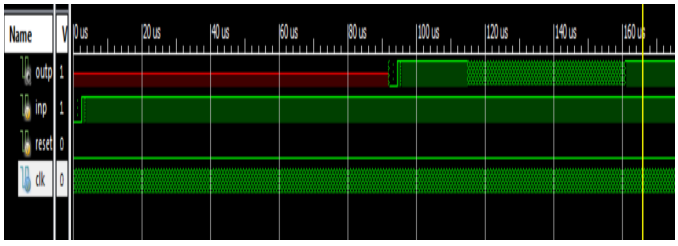


Fig. 6: The simulation result of the Turbo encoder with serial computation.

Figure 7 shows the simulation result of the parallel computing technique. The module starts to generate the output in the beginning. Note that the MSD is encoded in a shorter time compare with the serial computation simulation. This is the

result of parallel computing of multiple process in one clock cycle.

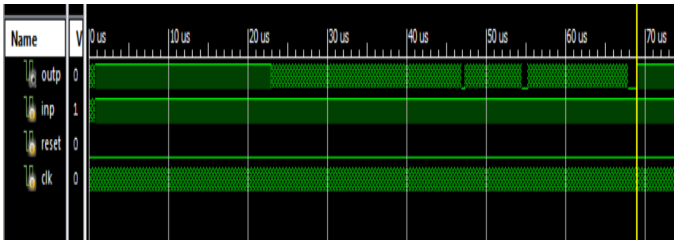


Fig. 7: The simulation result of the Turbo encoder with parallel computation.

### B. Hardware Implementation

FPGA has been widely used in the SoC design [10][11]. Zynq-7000 FPGA device is employed to implement the developed Turbo encoder module. Both serial and parallel computation methods are implemented on the FPGA device separately. The Turbo encoder module is synthesized and loaded on the FPGA device. The synthesis reports show that the parallel computation does not only save processing time, but also reduces the utilized logics and the chip size. Figures 8 and 9 show the utilization flip flops and look-up tables (LTU) for the serial and parallel computation respectively.

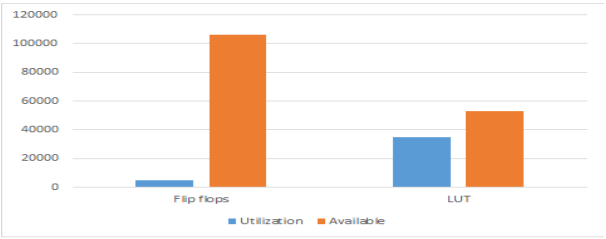


Fig. 8: The logic utilization of the Turbo encoder with serial computation.

The impact of the parallel computation on the logic utilization is shown in Figure 10. Note that the utilized flip flop and LUTs are significantly reduced when parallel computation is employed. The LUT utilization is improved by 73% and the flipflop utilization enhanced by 75%. It is proven that the FPGA technologies strongly supports parallel computation for the Turbo encoder. By reducing the size of the Turbo encoder module, the IVS can be implemented on a single programmable chip with less hardware constrains.

## IV. Conclusion

The Turbo encoder module is designed and implemented to be an embedded module in the IVS modem. FPGA technologies are employed to develop the Turbo encoder module. Xilinx tools and Verilog HDL are employed to design and simulate the module. Both serial and parallel computation techniques are studied for the encoding process. It is shown that the parallel computation can improve the chip size and

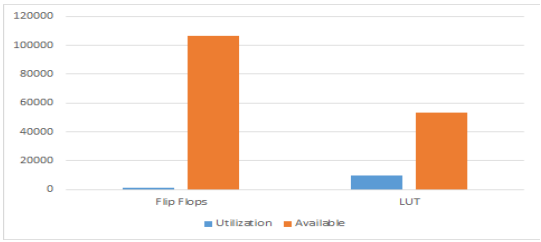


Fig. 9: The logic utilization of the Turbo encoder with parallel computation.

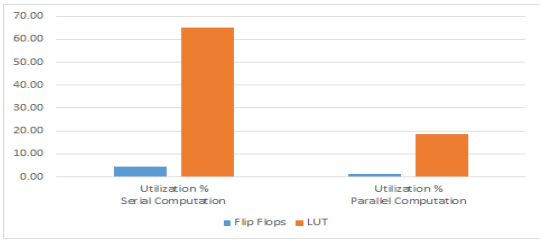


Fig. 10: The logic utilization of the Turbo encoder, serial vs parallel computation.

processing time of the module. Comparing with the serial computation technique, the parallel computation encoding, improves the processing time by 58% and logic utilization by 73%. The processing time enhancement can be seen in both simulation and analyzing the chip processing.

## References


[1] "eCall data transfer; in-band modem solution; general description," 3GPP, Tech. Rep. TS26.267.

[2] Eroupean Commission, "eCall: Time saved = lives saved," Press Release, Brussels, August 21, 2015. website: $http://ec.europa.eu/digital-agenda/en/ecall-time-saved-lives-saved$.

[3] A. Saleem et at. "Four-Dimensional Trellis Coded Modulation for Flexible Optical Communications," *IEEE Journal of Lightwave Technology.,* vol. 35, no. 2, pp. 151-158, Nov. 2017.

[4] C. Studer, C. Benkeser, S. Belfanti, and Q. Huang "Design and Implementation of a Parallel Turbo-Decoder ASIC for 3GPP-LTE," *IEEE Journal of Solid-state Circuits.,* vol. 46, no. 1, pp. 8-17, Jan. 2011.

[5] M. Nader and J. Liu, "Design and implementation of CRC module of eCall in-vehicle system on FPGA," *SAE Technical 2015-01-2844,* 2015, doi:10.4271/2015-01-2844.

[6] M. Nader and J. Liu. "Developing modulator and demodulator for the EU eCall in-vehicle system in FPGAs" in *IEEE, 2016 International Conference on Computing, Networking and Communications (ICNC),* Hawaii, USA, Feb. 15-18, 2016, pp. 1-5.

[7] M. Nader and J. Liu. "FPGA Design and Implementation of Demodulator/Decoder Module for the EU eCall In-Vehicle System" in *International Conference on Embedded Systems and Applications (ESA'15),* Las Vegas, USA, July 27-30, 2015, pp. 3-9.

[8] "Technical Specification Group Radio Access Network; Multiplexing and channel coding (FDD)," 3GPP, Tech. Rep. TS22.212.

[9] D. Viktor, K. Michal, and D. Milan"Impact of trellis termination on performance of turbo codes," in *ELEKTRO,* 2016, pp.48-51.

[10] B. Muralikrishna, G.L. Madhumati, H. Khan, K.G. Deepika, "Reconfigurable system-on-chip design using FPGA," in *2nd International Conference on Devices, Circuits and Systems (ICDCS),* 2014, pp.1-6.

[11] J. J. Rodriguez-Andina, M. J. Moure, and M. D. Valdes, "Features, design tools, and application domains of FPGAs," *IEEE Trans. Ind. Electron.,* vol. 54, no. 4, pp. 18101823, Aug. 2007.