Jurnal
Teknologi

Full Paper

UTILIZATION OF HIGH-SPEED DSP ALGORITHMS OF CYCLIC REDUNDANCY CHECKING (CRC-15) ENCODER AND DECODER FOR CONTROLLER AREA NETWORK

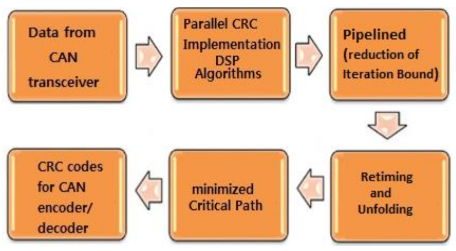
Ronnie O. Serfa Juan, Hi Seok Kim*

SLSI Laboratory, Electronic Engineering, College of Engineering, Cheongju University, Cheongju City, South Korea

*Corresponding author
khs8391@cju.ac.kr

Article history
Received
21 July 2015
Received in revised form
12 November 2015
Accepted
4 January 2016

Graphical abstract


Data from CAN transceiver
Parallel CRC Implementation DSP Algorithms
Pipelined (reduction of Iteration Bound)
Retiming and Unfolding
minimized Critical Path
CRC codes for CAN encoder/ decoder

Abstract

Advanced driver assistance system (ADAS) performs an increasing improvement in active road safety and driving convenience. Controller Area Network (CAN) is now getting popular because of its expanding applications and widely utilizations in low-cost embedded systems from automation to medical industry. While implementing an effective and efficient mechanism for clock synchronization, serial operation causes the reduction of CAN transmission rate can have an adverse impact on the real-time applications of systems employing this protocol. Also, maintaining the reliability of this technology especially in safety services, a reliable system needs certain requirements like glitches management and troubleshooting in order to avoid certain occurrences of transmission error. In this paper we present a simulated Cyclic Redundancy Checking (CRC) encoder and decoder that perform high speed error detection for CAN using CRC-15. Digital Signal Processing (DSP) algorithms were used, namely pipelining, unfolding and retiming to attain the feasible iteration bound and critical path that is appropriate for CAN system. The source code for Encoder and Decoder has been formulated in Verilog Hardware Description Language (HDL) from actual simulation to implementation of this CRC-15 for CAN system.

Keywords: CRC-15. CAN encoder and decoder, pipelining, unfolding, retiming

1.0 INTRODUCTION

A system like Controller Area Network (CAN) needs a real-time approach in correcting certain defects in its node like errors and glitches during transmission and reception. Aside from other CAN protocol's requirements: safety and security are needed to give a proper attention. Because the aim of road traffic safety systems are to reduce or totally eliminate the harm, certain fatalities or even in damage to property from resulting collisions of road vehicles. Research and development in CAN applications like the Advanced Driver Assistance Systems (ADAS) is rapidly increasing. Today, the requirements for better performance of this system and process flow are raised significantly. Fortunately, CAN itself has a self-correcting method specification that used for error checking on each frame's contents which is called as the Cyclic Redundancy Checking (CRC) code. CRCs are used in a wide variety of computer networks and data storage devices to provide inexpensive and effective error detection capabilities [1]. On transmission mode as information transfer rates and the amount of the data stored increases, the need for an uncomplicated but powerful error detection codes increases as well. Whenever high speed transmission rate is required, serial implementation does not meet this requirement.

However, CRC hardware operation is based on Linear Feedback Shift Registers (LFSRs), which utilizes serial transmission. LFSR is built from a simple shift-registers with a small number of XOR gates and this is being used for random number generations, counters and especially for error checking and correction. While Galois Fields is the theory behind LFSRs; a finite field named after Évariste Galois that contains a finite number of elements. Serial usage for CRC codes cannot achieve a high throughput. Likewise, CRC generation can be implemented using parallel techniques. Parallel CRC code computation can significantly increase its throughput. Common polynomial representations of CRC polynomials for automotive controller network applications are CRC-11 and CRC-24 for FlexRay utilizations [2], CRC-15 for CAN applications while CRC-17 and CRC-21 are for CAN-FD [3]. Equation (1) shows the standard implementation using CRC-15 generating polynomial P(x) for CAN:

$$P(x) = X^{15} + X^{14} + X^{10} + X^{8} + X^{7} + X^{4} + X^{3} + 1 \qquad (1)$$

The generating polynomial P(x) can be transform into binary form as 1100010110011001, which has 16 bits in total. The left most bit is the most significant bit and corresponds to the coefficient of X^{15}, the highest order of P(x).

In CAN bit frame, this 15-bit CRC segment in a data or remote frame contains the frame check sequence spanning from start of field (SOF), arbitration field, control field and through the data field [4] including the stuffed bits. The general hardware set-up for CRC calculation is a serial implementations using modulo-2 division [5].

This paper is organized as follows: In Section 2, are divided into three parts: principle behind the CRC codes using serial implementation; utilization using LFSR theory, and Parallel CRC architecture; CRC codes algorithm presentation; Lastly, proposed method using pipelining, unfolding and retiming. Section 3, discussion of general related works regarding CRC designs and utilizations. Then in Section 4, testbench evaluation using FPGA implementation for CRC-15 encoder and decoder utilization in serial operation and presentation of the proposed DSP algorithm for parallel implementation. Finally, Section 5 concludes this paper.

2.0 PRINCIPLE AND ALGORITHMS OF CRC CODES

CAN has very sophisticated error handling implemented as part of the protocol. CRC is a common method for checking for errors in data that has been transmitted on a communications link especially for CAN networks. It can be implemented into two techniques: Serial and Parallel CRC generation.

2.1 Serial Implementation of CRC

Transmitted messages are divided into predetermined lengths that are divided by a fixed divisor also known as generating polynomial. According to the basic calculation, the remainder will be appended after applying modulo-2 division and sent with the message. The remainder is recalculated and compares it to the transmitted remainder upon receiving the transmitted information. If the transmitted data does not match, an error is detected. This group of bits or remainder is called a syndrome [6].

In Standard CAN and Extended CAN structures, the total number of bits of CRC is 16 bits; 15 bits for CRC bit sequences and 1 bit for CRC delimiter [7]. The frame check sequence is derived from Bose, Chaudhuri and Hocquenghem (BCH) best suited for frame lengths of less than 127 bits.

The algorithm for this serial CRC architecture transmission on both encoder and decoder circuit is given as follows:

Initially, we must set the following values that will help us to clearly understand the simple algorithm of CRC code.

Let us denote, Data D(x) = 1010111 in binary form.

In polynomial form is $X^6 + X^4 + X^2 + X + 1$ while the Generating polynomial **P(x)** = 10101 in binary form.

In polynomial form is $X^4 + X^2 + 1$

First Step: Multiple the highest order of P(x) to D(x).

$G(x) = X^4 * (X^6 + X^4 + X^2 + X + 1)$

$= X^{10} + X^8 + X^6 + X^5 + X^4$ (polynomial form)

= 10101110000 (binary form)

Second Step: Divide G(x) to P(x) but using modulo-2 division or XOR operation. Figure 1 shows the modulo-2 division.

Third Step: Upon transmitting the data, the CRC code will be added to G(x). T(x) = G(x) + CRC. The transmitted data T(x) frame will be:

T(x) = 10101110000 + 0111 = 10101110111 (binary form)

Fourth Step: To determine if T(x) frame during transmission was transmitted successfully, it will be evaluated at the receiving end using Step 2 again. If no remainder was obtained, it means the T(x) frame has no error [8]-[9]. However, if an error occurred during transmission the remainder of the modulo-2 division is not zero; syndrome was occurred.

Fifth Step: For CAN application, the result of Step 4 will determine if the ACK Slot is a dominant bit (0) for a non-error or zero-remainder result, while the ACK Slot will be a recessive bit (1) with an error occurrences or the remainder is non-zero.

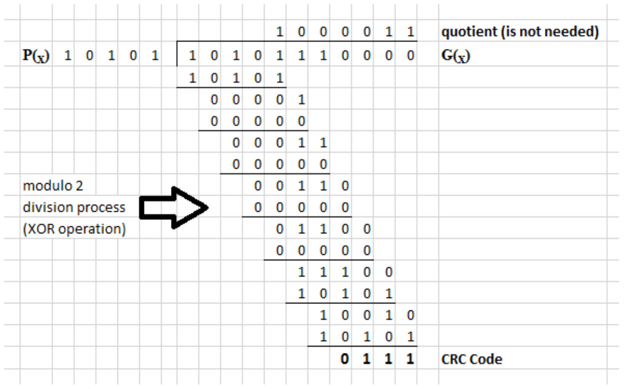


Figure 1 Step 2 procedures using modulo 2

2.2 LFSR Theory on CRC Coding

CRC arithmetic is primarily performs XOR operation in particularly values and using shifting techniques. Linear Feedback Shift Registers (LFSR) theory is needed in order to determine the expected CRC code. LFSR are widely used in BCH codes and CRC operation [10]-[12]. Also LFSR are n-bit counters exhibiting pseudo-random behavior. It is built from simple shift-registers that composes a D flip-flops with a number of XOR gates. Generally, it is used for random number generation, counters and error checking and correction like the Cyclic Redundancy Check. Some of LFSR's advantages are composed of little hardware and high speed in operation. In Equation (2), K denotes the length of the LFSR, i.e., the number of delay elements and P_0, P_1, P_2, P_3, …, P_k represent the coefficients of the characteristic polynomial of this LFSR is

$$P(X) = P_0 + P_1X + P_2X^2 + \dots + P_KX^K \qquad (2)$$

where $P_0, P_1, P_2, P_3, \dots, P_k \in GF(2)$. A 4-bit LFSR is shown in Figure 2.

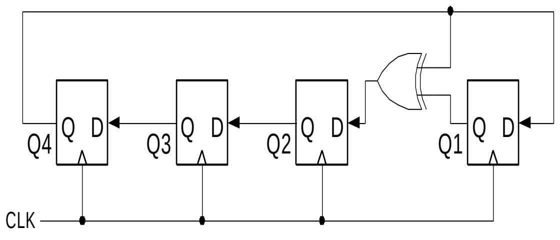


Figure 2 A 4-bit LFSR

The CRC is based on polynomial arithmetic, the remainder of dividing one polynomial in GF(2) (Galois Field with two elements) by another. LFSR algorithm for CRC is presented as follows:

1. Choose the appropriate CRC polynomial, for CAN protocol; CRC-15 is the standard polynomial.
2. To build a 15 bits LFSR, the following specifications from the Galois Field Polynomial of CRC-15.
   a. $G(x) = X^{15} + X^{14} + X^{10} + X^8 + X^7 + X^4 + X^3 + X^0$ (Generating polynomial of CRC-15)
   b. The $X^0 = 1$ term corresponds to connecting the feedback directly to the first FF for general LFSR but for CRC application another XOR gate will be connected before the first FF.
   c. The X^{15} indicates the number of flip-flop; total of 15 flip flops for CRC-15.
   d. These terms X^{14}, X^{10}, X^8, X^7, X^4, and X^3 connects XORs between FF_3 and FF_4, FF_4 and FF_5, FF_7 and FF_8, FF_8 and FF_9, FF_{10} and FF_{11} and FF_{14} and FF_{15} as a required tap of every LFSR. Figure 3 shows the LFSR of CRC-15.
3. Then CRC shift sequence will be the next step in determining the CRC codes. The initial contents of the LFSR are shown in the top row; namely, L_0 through L_{14}. Settings an 8-bit data, the first data bit (most significant bit) D_7 is shifted into the shift register, the new contents of the shift register are function of D_7 and the previous contents. Continue to shift until eight shifts (we used only 8 data bits for explanation purposes).

After the last data bit was shifted to the LFSR, we can be able to set-up the main Verilog program code for the encoder of CRC from the output of these shifted data sequence.

2.3 Parallel Implementation of CRC

Generally serial CRC architecture used LFSR design but the drawback raises on the transmission rate. It consumes more time to send the information. Parallel architecture overcomes this problem. It is an efficient way to increase the throughput rate. Although parallel transmission increase the data that can be processed in one clock cycle, it can lead to a long critical path (CP).

Though, increasing the throughput rate will reduce the parallel processing, this architecture tends to increase the hardware cost. The proposed design obtains shorter CP for parallel CRC circuits leading to high processing speed than the common serial implementation of CRC.

There are different techniques for parallel CRC generations given as follows:

a. A Table-Based Algorithm for Pipelined CRC Calculations
b. Fast CRC Update
c. F-Matrix Parallel CRC Generation
d. Unfolding, Retiming and Pipelining Algorithm

2.3.1 A Table-Based Algorithm for Pipelined CRC Calculations

This algorithm provides lower memory Look Up Table (LUT) and high pipelining table architecture and can obtain a higher throughput. The main problem is, it will store the pre-calculating CRC in LUT so, every time it required to change the LUT when changing the polynomial

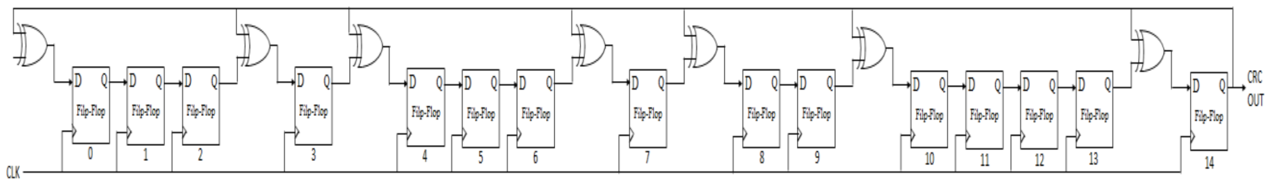


Figure 3 LFSR of a 15-bits CRC

2.3.2 Fast CRC Update

This parallel algorithm does not required to calculate CRC each time for all the data bits, instead of that calculating CRC for only those bits that are change and it requires buffer to store the old CRC and data.

2.3.3 F-Matrix Parallel CRC Generation

This parallel algorithm is simpler and not complex to compare with the other structure. It compresses long sequence data bits.

2.3.4 Unfolding, Retiming and Pipelining Algorithm

An unfolding algorithm is used to convert the original architecture to parallel processing. However, this method may lead to a parallel CRC circuit with high iteration bound, which is the lowest feasible critical path. Hence, Pipelining is needed to minimize this problem. It was developed to reduce the iteration bound of the serial CRC architecture. Then, unfolding algorithm is applied to attain a parallel structure with low iteration bound. Finally, retiming algorithm is essential to obtain the achievable lowest critical path.

3.0 RELATED WORKS

CRC implementations for CRC encoders and decoders are presented in various publications, but no implementations has been made for CAN applications using CRC-15. Also, CRC-15 design that includes significant procedures in achieving an appropriate software utilizations to hardware are not been realized. In addition, no DSP algorithms such as pipelining, utilization and retiming has been deployed for CRC-15.

In [12] presented the implementation of CRC encoder and decoder are based in FPGA utilizations. Although this paper is not intended for any applications like CAN networks, this work has an insufficient discussions, and no synthesized results were presented especially on detecting any possible syndrome occurrences are not conferred on this paper. In [13]-[14] shows the simulated results using the DSP algorithms for CRC-9 that uses a generator polynomial of $X^9 + X^8 + X + 1$. In Table I, and Table 2 shows the clock cycles, critical path, and the iteration bound of CRC-9 respectively.

Table I Clock cycles of CRC-9 architecture

Architecture	Number of Clock Cycles
Original Architecture	9
2-level pipelined	10
4-level pipelined	12
Retiming after pipelining	12
Retiming the unfolded architecture	5

From these works mentioned above, we propose a kind of realization of error-checking for CRC-15 that verified through Verilog language and FPGA implementation. Certain standards were been observed for verifications and comparison purposes.

Table 2 Iteration bound of CRC-9 architecture

Architecture	Iteration Bound
Original Architecture	$2T_{XOR}$
2-level pipelined	T_{XOR}
4-level pipelined and Retiming	$7/8T_{XOR}$

4.0 CRC-15 ARCHITECTURE SIMULATION

4.1 CRC-15 using Series Implementation for Encoding and Decoding

To design the CRC encoder using VHDL, we can use the algorithm presented above in getting the CRC code. For the simulated Verilog program, we selected the generating polynomial as;

$P(x) = X^5 + X^4 + X^2 + 1$

The input data in Figure 4 shows the data_in [11:0], clk is the system clock, and crc_en for the enable load signal rst for reset. While on the output side, the data_trans [16:0] for transmission and crc_out [4:0] is assigned for CRC code. The simulated result of CRC encoding is illustrated in Figure 6.

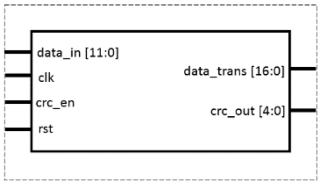


Figure 4 CRC Encoder

From the simulation shown in Figure 5, we can see that data_in as the input information code is 12'hAF5, while the data_trans as the encoded output is 17'h15EB5, we can get the crc_out as 5'h15.

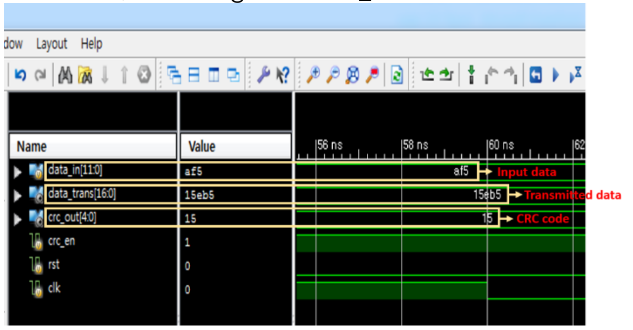


Figure 5 Simulated CRC Encoder

The decoding process is similar to the encoding process, at the end of every transmission, we must verify the encoded result of the decoded code if some possible error occurred during transmission.

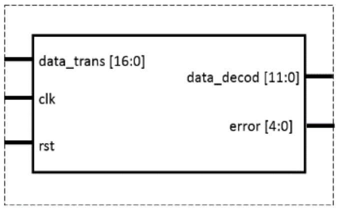


Figure 6 CRC Decoder

From Figure 6, the inputs that we define are the data_trans [16:0] is the received code words from the encoder, the clk is the clock of the decoding program, and rst is the reset signal. While for the output part, data_decod [11:0] is the decoding original input information and error [4:0] is the slot for the syndrome occurred during transmission.

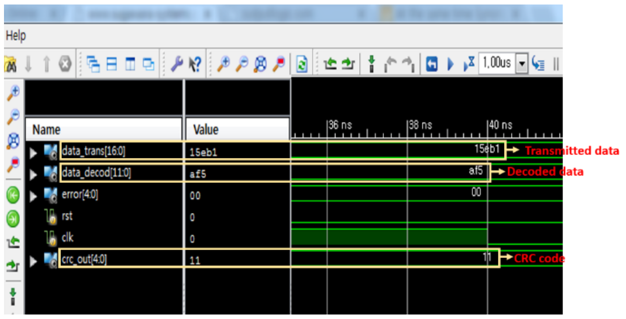


Figure 7 Simulated result of CRC decoder

Figure 7 show the data_trans is 17'h15EB1 and the data_decod is 12'hAF5, we can identify the CRC code as 5'h11. Therefore the encoding and decoding program is correct because the result to the simulated encoding process is the same, and the error output is zero.

4.2 Proposed DPS Algorithm for Parallel Implementation of CRC-15

The discussion presented above is based on serial connection of CRC encoder and decoder, it can be improved more using DSP algorithms of pipelining, retiming and unfolding to minimize the problem arises in transmission rate. Utilization using these DSP algorithms in CRC-15 architectures overcomes this drawback. This proposed architecture should be first pipelined to reduce the iteration bound then retimed and unfolding to design high speed parallel circuits.

4.2.1 Pipelining Algorithm

It reduces the CP either to increase the clock frequency or sample speed or to reduce power consumption at the same speed and the iteration bound of the system will be reduced. The applied four level pipeline algorithm of CRC-15 reduces the iteration bound to $0.5T_{XOR}$ is shown in Figure 8 its four level pipelined structure equation is shown in Equation (3)

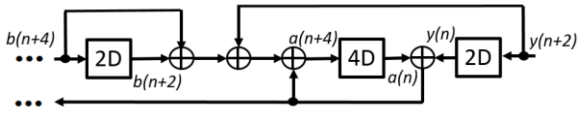


Figure 8 Four Level Pipelined Structure of CRC-15

$$a(n+4) = a(n) + y(n) + b(n+2) + y(n+2) + b(n+4) \quad (3)$$

4.2.2 Retiming Algorithm

Retiming is used to modify the locations of delay elements in a circuit without affecting the input/output characteristics of the circuit. This algorithm reduces the CP but not changing the latency of the system. Retiming is done by applying the Floyd Warshall algorithm [15].

4.2.2.1 Unfolding Algorithm

Direct implementation of unfolding may lead to long iteration bound with lowest achievable CP. In Figure 3 shows the architecture of CRC-15 in LFSR to the 3-point unfolded of a 2-factor pipeline-cutset retimed and 4-level pipeline of CRC-15 architecture as show in Figure 9.

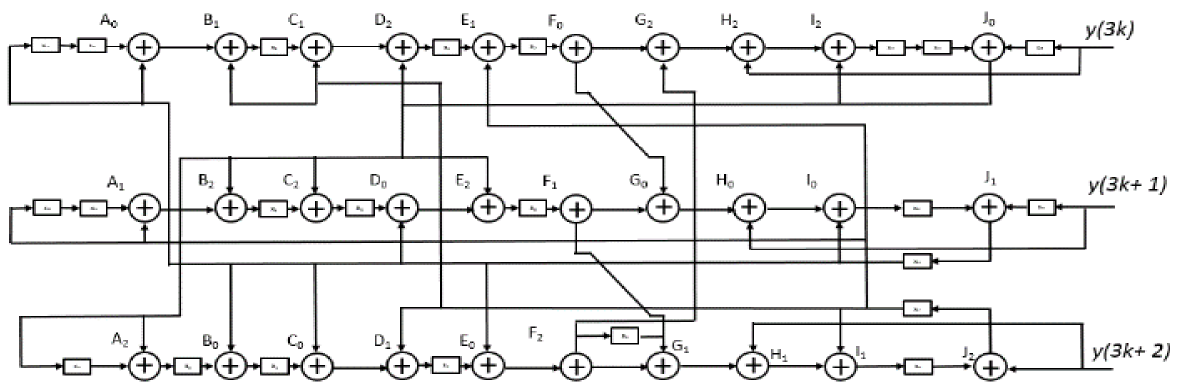


Figure 9 3-point Unfolded, 2 factor cutset retiming pipeline and a 4-level pipeline of CRC-15

Table 3 shows the output of CRC-15 that is much better output compare with the existing paper [13]-[14] using CRC-9 for the number of clock cycles, especially when CRC-15 is subjected to a retimed 3-point unfolded and 4-level pipelined. The clock cycles decreases 20% compares to the original CRC-9 architecture. While Table 4 shows the iteration bound comparison of CRC-9 to CRC-15. It shows a much better output after CRC-15 is subjected to a retimed 4-level pipelined operation, results shows a decrease in iteration bound of 38.09% compare with the original CRC-9.

Table 3 Comparison between CRC-9 and CRC-15 for number of clock cycles

CRC Polynomial	CRC-9	CRC-15
Original Architecture	9	15
4-level pipelined	12	20
Retiming after 4-level pipelined	12	20
Retiming the 3-point unfolded and 4-level pipelined	5	4

Table 4 Comparison between CRC-9 and CRC-15 of iteration bound

CRC Polynomial	CRC-9	CRC-15
Original Architecture	$2T_{XOR}$	$2T_{XOR}$
2-level pipelined	T_{XOR}	T_{XOR}
4-level pipelined	$1/2T_{XOR}$	$1/2T_{XOR}$
Retiming after 4-level pipelined	$7/8T_{XOR}$	$1/3T_{XOR}$

5.0 CONCLUSION

High speed data transmission preferred parallel implementation that cannot be executed by serial operation because of slow throughput. The proposed method of applying the DSP algorithm shows a better output from converting the serial CRC-15 to parallel operation that resulted with lower iteration bound and an increased throughput rate.

The result as shown in Figure 9 was subjected first through the utilization of pipelined to minimize the iteration bound, then it was retimed to reduce the CP but not changing the latency of the system and unfolding to obtain a superior design of a high speed parallel circuit.

In our future work, we plan to analyze the effects of higher pipelining level to maximize timing optimization. And, the design can be analyze in different unfolding factors for hardware overhead. In addition, we also plan to develop an alternative error correction-detection method for CRC in able to minimize the bits usage and to increase the frame rate in shortest possible transmission time.

Acknowledgement

This work was supported by the IT and R & D program of Ministry of Trade, Industry and Energy (10049192, Development of a Smart Automotive ADAS SW-SoC for a Self-Driving Car).

References

[1] Koopman P. 2002. 32-bit Cyclic Redundancy Codes for Internet Applications. *Proc. IEEE International Conference on Dependable Systems and Networks.* 459-468.
[2] FlexRay Consortium. 2010, FlexRay Communication System Protocol Specification Version 3.0.1. 114-115.
[3] BOSCH. 2012. CAN with Flexible Data-Rate Specifications [Online].12-13.
[4] Voss, W. 2008. Error Detection and Fault Confinement, in *A Comprehensible Guide to Controller Area Network*, 2nd ed., Copperhill Media Corporation. 117-122.
[5] Janakiram Ch. and Srinivas, K.N.H. 2014. An Efficient Technique for Parallel CRC Generation. *International Journal of engineering and Computer Science*. 3(1): 9761-9765
[6] Pfeiffer O., Ayre, A. and Keydel C. 2008. Underlying Technology: CAN, in *Embedded Networking with CAN and CANopen*, Copperhill Technologies Corporation. 224-226.
[7] Khalifa O., Islam MD R. and Khan S. 2004. Cyclic Redundancy Encoder for Error Detection in

Communication Channels, in *Proc. IEEE RF and Microwave Conference*. 224-226.

[8] Ramabadran T. V., and Gaitonde S.S. 1988. A tutorial on CRC Computations. *IEEE Micro*. 8(4): 62-75.

[9] Glavieux A. 1999. Cyclic Redundancy Checking, in *Data Communications and Computer Networks*, 2nd ed., Publishing House of High Education. 83-86.

[10] Peterson W. W. and Brown, D. T. 1961. Cyclic Codes for Error Detection, in Proc. IRE. 228-235

[11] Ayinala M. and Parhi K. K. 2011. High-Speed Parallel Architectures for Linear Feedback Shift Registers. *IEEE Transactions on Signal Processing*. 59(9): 4459-4469.

[12] Zhang T. and Ding Q. 2011. Design and Implementation of CRC Based on FPGA. *IEEE 2nd International Conference in Innovations in Bio-inspired Computing and Applications (IBICA)*.

[13] Reddy B. N., Kumar B. K. and Sirisha K. M. 2012. On the Design of High Speed Parallel CRC Circuits using DSP Algorithms. *International Journal of Computer Science and Information Technologies (IJCSIT)*.

[14] Cheng C. and Parhi K. K. 2006. High-Speed Parallel CRC Implementation Based on Unfolding, Pipelining, and Retiming. *IEEE Transactions on Circuits and Systems.* 4(2): 1017-1021.

[15] Garfield, N. F. 2011. Floyd-Warshall Algorithm, Anim Publisher