IEIE Transactions on Smart Processing and Computing, vol. 5, no. 1, February 2016
http://dx.doi.org/10.5573/IEIESPC.2016.5.1.29

Using DSP Algorithms for CRC in a CAN Controller

Ronnie O. Serfa Juan[1,2] and Hi Seok Kim[1]

[1] Department of Electronic Engineering, College of Engineering, Cheongju University / Cheongju, South Korea
{ronnieserfajuan, khs8391}@cju.ac.kr
[2] Electronics Engineering Department, College of Engineering, Technological University of the Philippines/Manila / Philippines engr_serfs@yahoo.com

***** Corresponding Author: Hi Seok Kim

Received February 16, 2016; Accepted February 24, 2016; Published February 29, 2016

** Extended from a Conference: This paper was presented at ICEIC 2016. This paper has been accepted by the editorial board through the regular review process that confirms the original contribution.*

***Abstract*:** A controller area network (CAN) controller is an integral part of an electronic control unit, particularly in an advanced driver assistance system application, and its characteristics should always be advantageous in all aspects of functionality especially in real time application. The cost should be low, while maintaining the functionality and reliability of the technology. However, a CAN protocol implementing serial operation results in slow throughput, especially in a cyclical redundancy checking (CRC) unit. In this paper, digital signal processing (DSP) algorithms are implemented, namely pipelining, unfolding, and retiming the CAN controller in the CRC unit, particularly for the encoder and decoder sections. It must attain a feasible iteration bound, a critical path that is appropriate for a CAN system, and must obtain a superior design of a high-speed parallel circuit for the CRC unit in order to have a faster transmission rate. The source code for the encoder and decoder was formulated in the Verilog hardware description language.

***Keywords*:** Parallel CRC, Pipelining, Retiming, Unfolding, CRC-15

1. Introduction

Acontroller area network (CAN) is a system that needs a real-time approach to correcting certain problems in its nodes, like errors and glitches. The aim of road traffic safety systems is to reduce or totally eliminate harm, certain fatalities or damage to property from collisions between road vehicles, especially in real-time scenarios. CAN applications like an advanced driver assistance system (ADAS) are rapidly increasing in number and serve an important role in embedded systems. Today, the requirements for better performance by systems and for process flow have been raised significantly. The CAN itself has a self-correcting method that is used for error checking each frame's contents, called cyclical redundancy checking (CRC) code. CRC codes are used in a wide variety of computer networks and data storage devices to provide inexpensive and effective error detection capabilities [1]. Common polynomial representations of CRC polynomials for the algebraic representations of the polynomials for automotive controller network applications are CRC-11 and CRC-24, both for FlexRay utilizations [2], CRC-15 for CAN applications, and CRC-17 and CRC-21 for CAN-FD [3]. Eq. (1) shows the standard implementation for CAN using CRC-15 generating polynomial $P(_X)$ [4]:

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

This 15-bit CRC segment in a data or remote frame contains the frame check sequence from the start of the frame through the arbitration field and the control field to the data field [4]. Stuffing bits are included.

The general hardware set-up for CRC calculation is serial implementation using modulo-2 division [5]. The common design approach is accomplished using the linear feedback shift register (LFSR), which is built from simple shift registers with a small number of XOR gates. This is used for random number generation, counters, and especially for error checking and correction. The Galois

field is the theory behind LFSRs, a finite field named after Évariste Galois, which contains a finite number of elements. Also, CRC generation can be implemented using parallel techniques.

This paper is organized as follows. Section 2 describes the principle behind CRC codes, from serial implementation to parallel CRC architecture. Section 3 discusses general related works on CRC design and utilization. Then, in Section 4, a test bench for evaluation is presented for CRC encoder and decoder utilization for both serial and parallel implementations. Finally, Section 5 concludes this paper.

2. Principles and Algorithms for CRC Codes

The CAN protocol utilizes a very sophisticated error handling technique. The network implements a CRC method to check for errors in data transmitted over a communications link. CRC can be implemented via two techniques: serial and parallel CRC generation. CRC provides the capability to detect burst errors, which are commonly encountered in digital transmissions. In the CRC method, several bits are appended to the transmitted message so the receiver can determine with a certain degree of probability if an error occurred during transmission.

2.1 Serial Implementation of CRC

Transmitted messages are divided into predetermined lengths that are separated by a fixed divisor, also known as generating a polynomial. According to the basic calculation, the remainder will be appended after applying modulo-2 division and being sent with the message. Then, the receiver will recalculate the remainder and compare it to the transmitted remainder upon receiving the transmitted information. The group of error control bits is appended to the end of the block of transmitted data and is called a syndrome [6].

The modulo-2 division process for a serial CRC architecture transmission is shown in Fig. 1.

2.2 LFSR Theory on CRC Coding

Generally, CRC arithmetic uses an XOR operation and a shifting technique based on LFSR theory. This shifting technique is needed in order to determine the CRC code. LFSR is widely used in Bose-Chaudhuri-Hocquenghem codes and CRC operations [7-9]. Also, LFSR is built from simple shift registers composed of D flip-flops and a number of XOR gates. Generally, it is used for random number generation, counters and error checking and correction, like cyclical redundancy checking. In most cases, CRC calculation is based on LFSR and deals with only one data bit per clock cycle due to serial input.

A basic LFSR architecture for a K^{th} order generating polynomial is a *Galois field*. In Eq. (2), K denotes the length of the LFSR, i.e., the number of delay elements, and

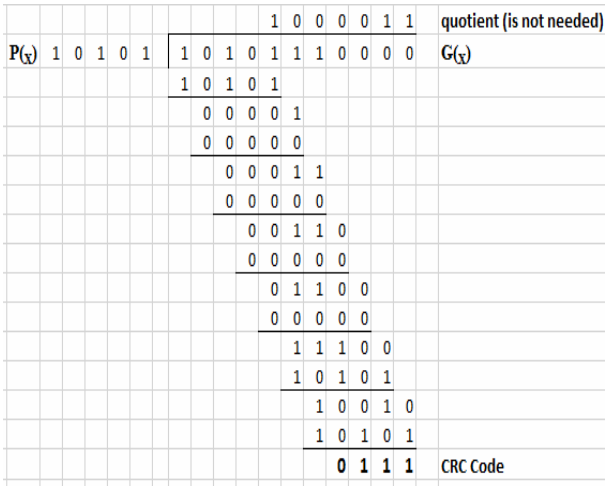

Fig. 1. Division procedure using modulo-2.

P_0, P_1, P_2, P_3, …, P_k represents the coefficients of the characteristic polynomial of this LFSR, which is

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

where $P_0, P_1, P_2, P_3, \ldots, P_k \in GF(2)$.

The Galois field or the primitive polynomial of the form $X^k + \ldots + X^0$ is the proper polynomial in constructing the steps of data shifting for the manipulation of the CRC code. The k exponent indicates the k-bits for CRC, while the $X^0 = 1$ term corresponds to connecting the feedback directly to the D flip-flop (FF) input of FF_1.

An LFSR algorithm for CRC is as follows.

1. Determine the appropriate CRC polynomial; for the CAN application, CRC-15 was selected.
2. In order to build a 15-bit LFSR, the following specifications from the CRC-15 Galois field polynomial must be as follows.
   a. $G(X) = X^{15} + X^{14} + X^{10} + X^8 + X^7 + X^4 + X^3 + 1$
   b. The $X^0 = 1$ term corresponds to connecting the feedback directly to the first FF for general LFSR, but for a CRC application, another XOR gate will be connected before the first FF.
   c. X^{15} indicates the number of flip-flops; a total of 15 flip flops for CRC-15.
   d. The terms X^{14}, X^{10}, X^8, X^7, X^4, and X^3 connect 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 the required tap of every LFSR. Fig. 2 shows the LFSR of a CRC-15.
3. Then, implement the CRC shift sequence, the initial contents of the LFSR, namely, L_0 through L_{14}. Setting eight-bit data, the first data bit (the most significant bit) D_7 is shifted into the shift register.

2.3 Parallel Implementation of CRC

In LFSR theory generally, a serial CRC architecture uses an LFSR design, but a drawback arises with the transmission rate. A parallel architecture overcomes this

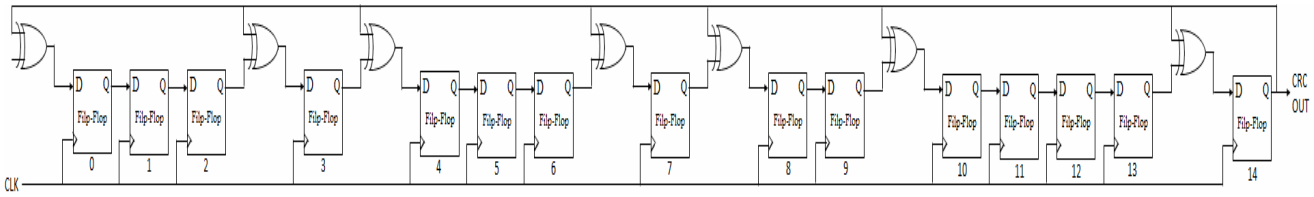

Fig. 2. LFSR of a 15-bit CRC.

problem.

There are different techniques for parallel CRC generation, given as follows.

2.3.1 Table-Based Algorithm for Pipelined CRC Calculations

This algorithm provides a lower memory look up table (LUT) and a high pipelining table architecture, and can obtain higher throughput. The main drawback is that it will store the pre-calculating CRC in the LUT. Therefore, it is necessary to change the LUT every time the polynomial changes.

2.3.2 Fast CRC Update

This parallel algorithm does not need to calculate CRC code each time for all the data bits. Instead, it calculates CRC code for only those bits that change, and it needs a buffer to store the previous CRC code and data.

2.3.3 F-matrix Parallel CRC Generation

This parallel algorithm is not complex, compared with the other structure. It compresses a long sequence of data bits.

2.3.4 Unfolding, Retiming and Pipelining Algorithm

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

3. Related Works

CRC implementations for CRC encoders and decoders were presented in different publications, however, no implementations have been made for CRC-15. Also, no digital signal processing (DSP) algorithms, such as pipelining, unfolding, and retiming, have been utilized for CRC-15. Reddy et al. [10] presented implementation of CRC code in a field-programmable gate array and discussed CRC encoder and decoder utilization.

Table 1. Clock Cycles of a CRC-9 Architecture.

Architecture	Number of Clock Cycles
Original Architecture (Serial)	9
2-level pipelined	10
4-level pipelined	12
Unfolding the 4-level pipelined	4
Retiming the unfolded architecture	5

Table 2. Iteration Bound of a CRC-9 Architecture.

Architecture	Iteration Bound
Original Architecture (Serial)	$2T_{XOR}$
2-level pipelined	T_{XOR}
4-level pipelined and retimed	$7/8T_{XOR}$

Although the paper was not intended for applications like CAN networks, this work has insufficient discussion. No synthesized results were presented, and in particular, ways to detect possible syndrome occurrences with its implementation were not conferred in the paper. Also, the data and results presented are not enough for future references. Cheng and Parhi [11] and Singh et al [12] showed simulated results using DSP algorithms for CRC-9 using a generator polynomial of $X^9 + X^8 + X + 1$.

Tables 1 and 2 show a comparison of serial-to-parallel implementation of CRC-9. As shown in both tables, when the DSP algorithm is implemented, it minimizes the clock cycles and the iteration bound of the original serial architecture. The number of clock cycles (9 in the original serial architecture) is reduced to 5 when retiming is utilized on the unfolded architecture.

4. CRC-15 Architecture Simulation

The CAN controller is the hardware component that manages physical access to the transmission medium. It provides registers for configuration of the connection to a bus. The controller also implements the functionality for managing and controlling the CAN protocol, including management of the transmission modes and handling of the bus off-state. Parts of the designed controller are the CRC encoder and decoder units. This paper shows the two

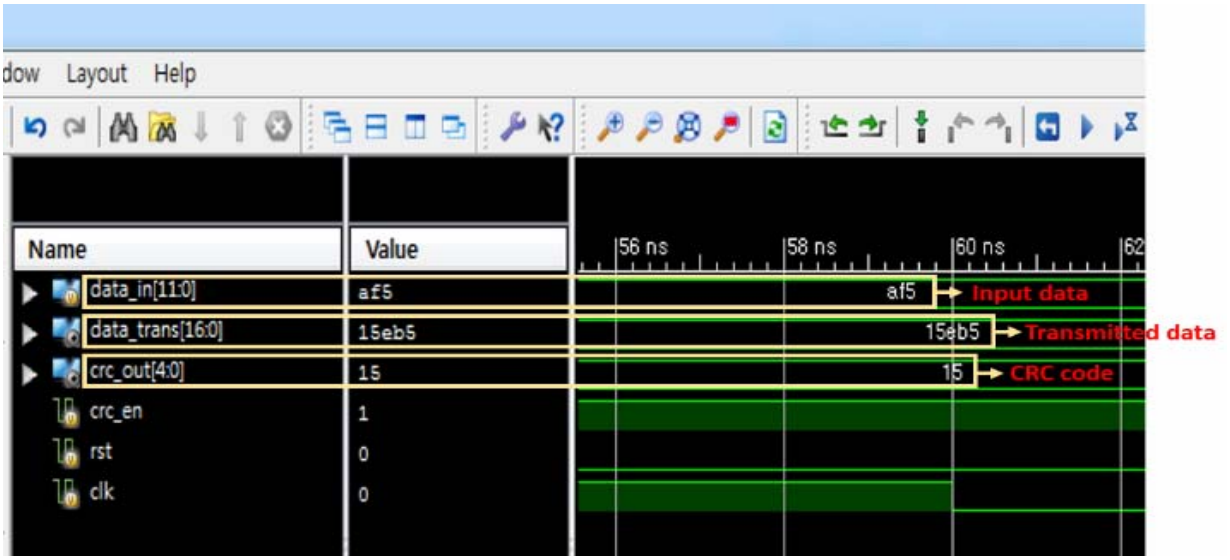

Fig. 3. Simulated CRC encoder.

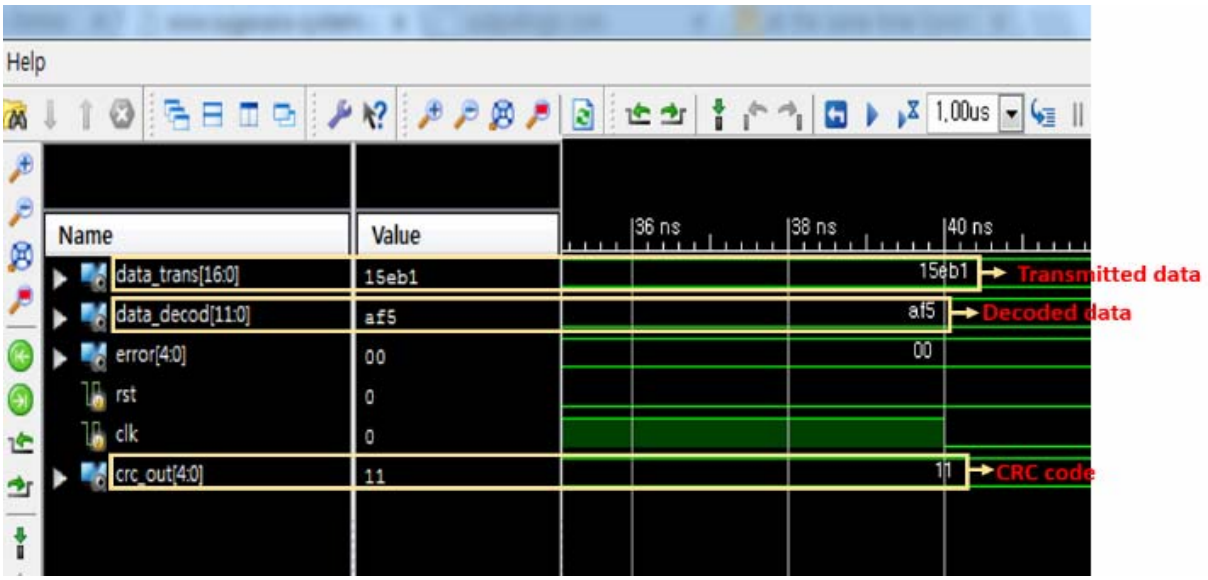

Fig. 4. Simulated result of CRC decoder.

options in implementing an encoder and decoder unit for CRC code.

4.1 Serial Implementation of Encoding and Decoding

In designing the CRC encoder using the Verilog hardware description language (HDL), the algorithm presented above can be used in getting the CRC code. For the simulated Verilog HDL, we selected the generating polynomial $P(_X) = X^5 + X^4 + X^2 + 1$. The input sides are as follows: data_in [11:0] is the input data, clk is the clock of the system, valid on its rising edge, crc_en for the enable load signal on high level, and rst for reset. The output sides are: data_trans [16:0] to be the encoded code words for transmission, and crc_out [4:0] for the CRC code. The simulated result of CRC encoding is illustrated in Fig. 3.

While decoding is similar to encoding, at the end of every transmission, we must verify the encoded result of the decoded code for any possible error that occurred during transmission. The input that we define are data_trans [16:0] for the received code words from the encoder, clk is the clock of the decoding program, and rst is the reset signal. For output, data_decod [11:0] is decoding original input information, and error [4:0] is the slot for the syndrome that occurred during transmission.

Fig. 4 shows data_trans is 15eb1 in hexadecimal and data_decod is af5 in hexadecimal, and when converted into binary: 10101111010110001 and 101011110101, respectively. From this, we can identify the CRC code as 10001 in binary form. Therefore, the encoding and decoding program is correct, because the result of the simulated encoding process is the same, and error output is zero.

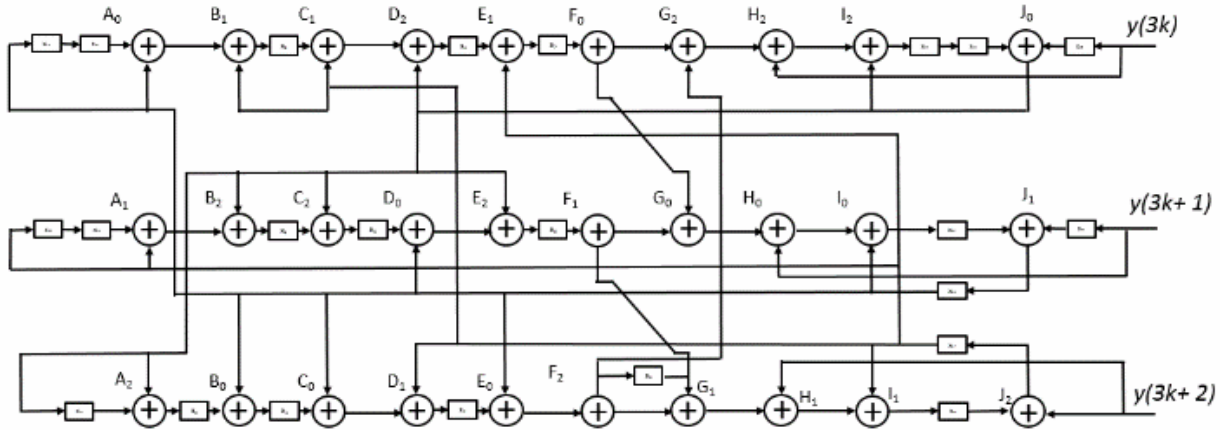

Fig. 5. Three-point unfolded, two-factor cutset retiming pipeline, and a four-level pipeline for CRC-15.

4.2 Parallel Implementation

Parallel CRC implementation improves slow throughput in serial transmission. Using the following DSP algorithms (namely, pipelining, retiming, and unfolding) helps to minimize the problem that arises in the transmission rate. This proposed algorithm should first be pipelined to reduce the iteration bound, then retimed to reduce the critical path (CP), and unfolded to design a high-speed parallel circuit.

4.2.1 Pipelining Algorithm

This algorithm reduces the CP to either increase the clock frequency or the sample speed, and the iteration bound of the system will be reduced.

4.2.2 Retiming Algorithm

A retiming algorithm relocates registers and delay elements to reduce the cycle time or the register areas without affecting the input/output characteristics of the circuit. This algorithm reduces the CP, but does not change latency in the system.

4.2.3 Unfolding Algorithm

Unfolding is a technique that duplicates the functional blocks to increase the throughput of the DSP program in such a way that the output preserves its functional characteristics and its output. Direct implementation of unfolding may lead to a long iteration bound with the lowest achievable CP.

Tables 3 and 4 shows the output using the DSP algorithms in CRC-15, which is much better output compared with the existing paper on serial implementation, for both clock cycles and iteration bound. Fig. 5 shows the architecture of CRC-15 for three-point unfolded, two-factor pipeline-cutset retimed, and four-level pipeline.

The results show that the CRC-15 architecture was subjected to the following DSP algorithms.

1. It utilized pipelining and was able to minimize the iteration bound.
2. Then, retiming was able to reduce the CP, but not change latency in the system.
3. Finally, unfolding obtained superior design of a high-speed parallel circuit.

Table 3. Clock cycle results of CRC-15 after the implementation of DSP algorithms.

CRC Polynomial	CRC-15
Original architecture (serial)	15
4-level pipelined	20
Retiming after 4-level pipelined	20
Retiming the 3-point unfolded and 4-level pipelined	4

Table 4. Iteration bound results of CRC-15 after the implementation of DSP algorithms.

CRC Polynomial	CRC-15
Original architecture (serial)	$2T_{XOR}$
4-level pipelined	T_{XOR}
Retiming after 4-level pipelined	$1/3T_{XOR}$

5. Conclusion

Parallel implementation is preferred for higher speed data transmission that cannot be executed over serial operation due to its slow throughput. The proposed method of applying the DSP algorithm shows better output from converting serial CRC-15 to a parallel operation that resulted in a lower iteration bound and an increased throughput rate, which is appropriate for a CAN controller, especially for the encoder and decoder unit. Fig. 5 shows the resulting architecture that was subjected to the following algorithms: first, pipelining was able to minimize the iteration bound; then, it was retimed to reduce the CP but did not change latency in the system. It was unfolded to obtain superior design of a high-speed parallel circuit. The clock cycles and iteration bound decreased by 20% and 38.09%, respectively.

In our future work, we plan to analyze the effects of a

higher pipelining level to maximize the timing optimization and to employ the design in different unfolding factors for hardware overhead.

Acknowledgement

This work was supported by the IT and R&D Program of the Ministry of Trade, Industry and Energy (No. 10049192, Development of Smart Automotive ADAS SW-SoC for Self-Driving Car).

References

[1] P. Koopman. (2002). 32-bit Cyclic Redundancy Codes for Internet Applications. Proc. IEEE International Conference on Dependable Systems and Networks. Article (CrossRef Link)
[2] FlexRay Consortium. (2010, October). FlexRay Communication System Protocol Specification Version 3.0.1 pp. 114-115. Article (CrossRef Link)
[3] BOSCH. (2012, April). CAN with Flexible Data-Rate Specifications pp. 12-13. Article (CrossRef Link)
[4] W. Voss, “Error Detection and Fault Confinement,” in A Comprehensible Guide to Controller Area Network, 2nd ed., Copperhill Media Corporation, 2008, pp. 117-122.
[5] Ch. Janakiram, and K.N.H. Srinivas, (2014, December). An Efficient Technique for Parallel CRC Generation. *International Journal of engineering and Computer Science*. pp. 9761-9765. Article (CrossRef Link)
[6] O. Pfeiffer, A. Ayre, and C. Keydel, “Underying Technology: CAN”, in Embedded Netwroking with CAN and CANopen, Copperhill Technologies Corporation, 2008, pp.
[7] W. W. Peterson, and D. T. Brown, “Cyclic Codes for Error Detection,” in Proc. IRE, 1961, pp. 228-235. Article (CrossRef Link)
[8] M. Ayinala, and K. K. Parhi (2011, September). High-Speed Parallel Architectures for Linear Feedback Shift Registers. *IEEE Transactions on Signal Processing*. 59(9), pp. 4459-4469. Article (CrossRef Link)
[9] T. Zhang, and Q. Ding. (2011, December). Design and Implementation of CRC Based on FPGA. *IEEE 2nd International Conference in Innovations in Bio-inspired Computing and Applications (IBICA)*. pp. 160-162. Article (CrossRef Link)
[10] B. N. Reddy, B. K. Kumar, and K. M. Sirisha, (2012). On the Design of High Speed Parallel CRC Circuits using DSP Algorithms. *International Journal of Computer Science and Information Technologies (IJCSIT)*. pp. 5254-5258. Article (CrossRef Link)
[11] C. Cheng, and K. K. Parhi, (2006, October). High-Speed Parallel CRC Implementation Based on Unfolding, Pipelining, and Retiming. *IEEE Transactions on Circuits and Systems*. pp. 1017-1021. Article (CrossRef Link)
[12] S. Singh, S. Sujana, I. Babu and K. Latha, (2013, May-June). VLSI Implementation of Parallel CRC Using Pipelining, Unfolding and Retiming. *IOSR Journal of VLSI and Signal Processing (IOSR-JVSP).* pp. 66-72. Article (CrossRef Link)

Ronnie O. Serfa Juan received his BSc in Electronics and Communications Engineering from Technological University of the Philippines-Manila as a Commission on Higher Education (ChEd) scholar, and he earned his MSc in Information and Telecommunications Studies, majoring in Computer Systems and Network Engineering, at Waseda University, Tokyo, Japan, supported by the Japanese Government under the JICE-JDS scholarship program, in 1999 and 2007, respectively. He is currently working toward his PhD, majoring in Computer and Control, at CheongJu University, CheongJu City, South Korea under the scholarship program of the Korean Government. He was a faculty member for both the undergraduate and graduate programs of Technological University of the Philippines-Manila and some universities in the Philippines. His research interests include radio frequency identification (RFID), ISFET and pH sensor applications and controller area networks for both medical applications and advanced driver assistance system (ADAS) technology.

Hi-Seok Kim received his BSc, MSc and Ph.D. in Electronic Engineering from Hanyang University, Republic of Korea in 1980, 1985 and 1987 respectively. He is currently a Professor in the Electronic Engineering Department, CheongJu University, CheongJu City, South Korea. His research interests include digital video/audio system design, multi-view imaging, 3D image processing, and FPGA design. Dr. Kim has served as General Chair and a committee member of many Korean and international conferences, including the International SoC Design Conference and IEEE ISCAS, He is also one of the General Co-Chairs for APCCAS 2016.