Utilization of DSP Algorithms for Cyclic Redundancy Checking (CRC) in Controller Area Network (CAN) Controller

Ronnie O. Serfa Juan
Electronic Engineering Department, College of Engineering
Cheongju University
Technological University of the Philippines
Cheongju City, South Korea
ronnie71@naver.com

Hi Seok Kim
Electronic Engineering Department, College of Engineering
Cheongju University
Cheongju City, South Korea
khs8391@cju.ac.kr

***Abstract*—Controller Area Network (CAN) controller is an integral part of Electronic Control Unit (ECU) particularly in Advanced Driver Assistance System (ADAS) application, its characteristics should always be advantageous in all aspects of functionality. Primarily, the costing should be low but maintaining the reliability of this technology. However, CAN protocol is implementing serial operation resulting to a slow throughput. In this paper, we utilized the Digital Signal Processing (DSP) algorithms, namely pipelining, unfolding and retiming to CAN controller in Cyclic Redundancy Checking (CRC) unit particularly for Encoder and Decoder section in able to attain the feasible iteration bound, critical path that is appropriate for CAN system and to obtain superior design of a high speed parallel circuit for CRC. The source code for Encoder and Decoder has been formulated in Verilog Hardware Description Language (HDL).**

Keywords—parallel CRC, pipelining, retiming, unfolding, CRC-15

I. Introduction

Controller Area Network (CAN) is a system that needs a real-time approach in correcting certain problems in its node like errors and glitches. The aims of road traffic safety systems is to reduce or totally eliminate the harm, certain fatalities or damage to property from resulting collisions of road vehicles. CAN applications like Advanced Driver Assistance Systems (ADAS) is rapidly increasing. Today, the requirements for better performance of system and process flow are raised significantly. CAN itself has a self-correcting method 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]. Common polynomial representations of CRC polynomials for the algebraic representations of the 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 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 SOF, through the arbitration field, control field to data field [4]. Stuffing bits are included.

The general hardware set-up for CRC calculation is the serial implementations using modulo-2 division [5]. The common approach designed is accomplished using the linear feedback shift register (LFSR); in which it 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. Also, CRC generation can be implemented using parallel techniques.

This paper is organized as follows: In Section II, shows the principle behind the CRC codes from serial implementation to Parallel CRC architecture. Section III, discussion of general related works about CRC designs and utilizations. Then in Section IV, a testbench for evaluation is presented for CRC encoder and decoder utilization both for serial and parallel implementation. Finally, Section V concludes this paper.

II. Principle and Algorithm of CRC Codes

CAN protocol utilizes a very sophisticated error handling technique. This network implements CRC method for checking errors in data that has been transmitted on a communications link. CRC can be implemented into two techniques; the Serial and Parallel CRC generation. It provides a capability of detecting burst errors which are commonly encountered in digital transmission. In CRC method, a certain number of bits are appended to the message being transmitted. The receiver can verify with a certain degree of probability if an error occurred in transmission.

A. 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

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

remainder will be appended after applying modulo-2 division and sent with the message. Then it will recalculated the remainder and compares it to the transmitted remainder upon receiving the transmitted information. These group of error control bits is appended to the end of the block of transmitted data and it is called a syndrome [6].

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

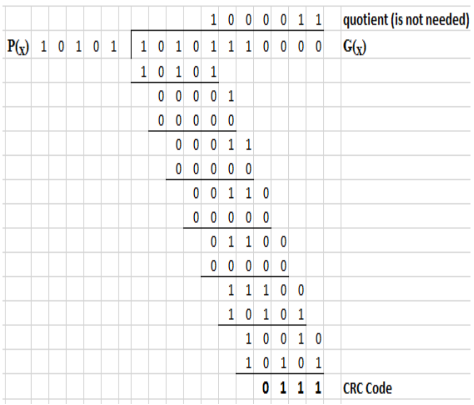

Fig. 1. Procedures using modulo 2

B. *LFSR Theory on CRC Coding*

Generally, CRC arithmetic uses XOR operation and shifting technique based on LFSR theory. This shifting technique is needed in order to determine a CRC code. LFSR are widely used in BCH codes and CRC operation [7]-[9]. Also, LFSR 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. 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 Kth order generating polynomial in *Galois Field*. In Equation (2), K denotes the length of the LFSR, i.e., the number of delay elements and $P_0, P_1, P_2, P_3, \ldots, P_k$ represent the coefficients of the characteristic polynomial of this LFSR 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 indicated the k-bits of CRC while the $X^0 = 1$ term corresponds to connecting the feedback directly to the D flip-flop (FF) input of FF_1.

LFSR algorithm for CRC is presented as follows:

1. Determine the appropriate CRC polynomial, for CAN application CRC-15 was selected.
2. To build a 15 bits LFSR, the following specifications from the Galois Field Polynomial of CRC-15.
   a. $G(x) = X^{15} + X^{14} + X^{10} + X^8 + X^7 + X^4 + X^3 + 1$
   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 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. 3 shows the LFSR of CRC-15.
3. Then CRC shift sequence, the initial contents of the LFSR namely, L_0 through L_{14}. Setting an 8-bit data, the first data bit (most significant bit) D_7 is shifted into the shift register.

C. *Parallel Implementation of CRC*

The LFSRs theory, generally serial CRC architecture used LFSR design but the drawback raises on the transmission rate. Parallel architecture overcomes this problem.

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

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 drawback is, it will store the pre-calculating CRC in LUT therefore, every time it required to change the LUT when changing the polynomial.

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.

3. *F-matrix Parallel CRC Generation*

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

4. *Unfolding, retiming and Pipelining Algorithm*

An unfolding algorithm is used to convert the original architecture to parallel. However, this method may lead to a parallel CRC circuit with 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, 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.

III. Related Works

CRC implementations for CRC encoders and decoders are presented in different publications, however no implementations has been made for CRC-15. Also, no DSP algorithm such as pipelining, utilization and retiming had been utilized for CRC-15. In [10] presented the implementation of CRC code based in FPGA discussed about CRC encoder and decoder utilizations.

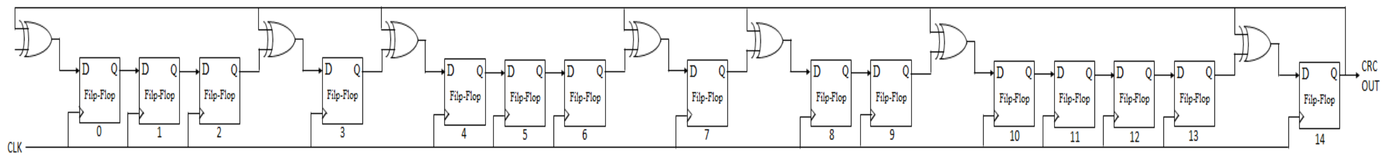

Fig. 3. LFSR of a 15-bits CRC

Although this paper is not intended for any applications like the CAN networks. But, this work has an insufficient discussions, and no synthesized results were presented especially on how to detect any possible syndrome occurrences on its implementation were not conferred in this paper. Also, the data and results presented is not enough for future references. In [11]-[12] 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 II shows the clock cycles, critical path, and the iteration bound of CRC-9 respectively.

Table I and II show the comparison from serial to parallel implementation of CRC-9. As shown in both tables, when the DSP algorithms is implemented, it minimizes the clock cycles and the iteration bound of the original serial architecture.

TABLE I. CLOCK CYCLES OF 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 II. ITERATION BOUND OF CRC-9 ARCHITECTURE

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

IV. CRC-15 ARCHITECTURE SIMULATION

A. *Series Implementation of Encoding and Decoding*

To design the CRC encoder by the Verilog HDL, we can use the algorithm presented above in getting the CRC code. For the simulated Verilog HDL, we selected the generating polynomial as; $P(X) = X^5 + X^4 + X^2 + 1$. The input sides are: data_in [11:0] is the input data, clk is the clock of the system, valid on its rising edge, and crc_en for the enable load signal and on high level and rst for reset. While on the output sides are: the 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. 4.

While the decoding part is similar to the encoding section, at the end of every transmission, we must verify the encoded result of the decoded code if some possible error occurred during transmission. 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 parts, data_decod [11:0] is the decoding original input information and error [4:0] is the slot for the syndrome occurred during transmission.

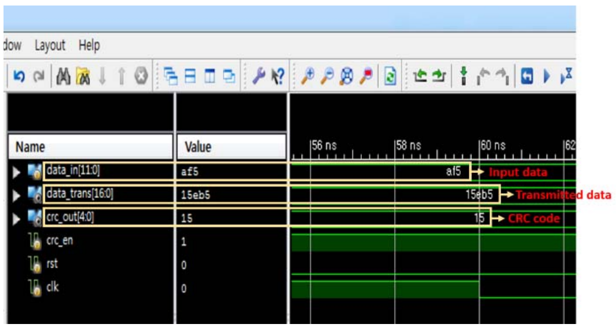

Fig. 4. Simulated CRC Encoder

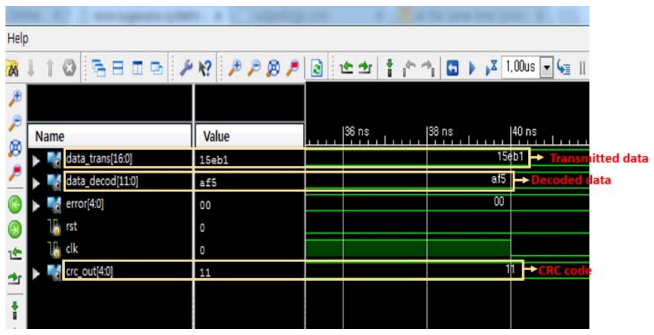

Fig. 5. Simulated result of CRC decoder

Fig. 5 show the data_trans is 15eb1 in hexadecimal and the data_decod af5 in hexadecimal, when converted into binary system 10101111010110001 and 101011110101 respectively. From this, we can identify the CRC code is 10001 in binary form. 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.

B. *Parallel Implementation*

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

1. *Pipelining Algorithm*

It reduces the CP either to increase the clock frequency or sample speed and the iteration bound of the system will be reduces.

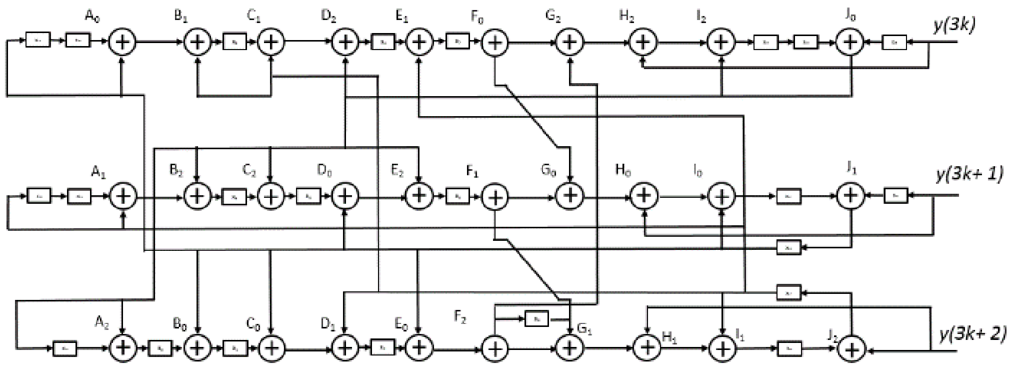

Fig. 6. 3-point Unfolded, 2 factor cutset retiming pipeline and a 4-level pipeline of CRC-15

2. *Retiming Algorithm*

Retiming is used to modify the locations of delay elements without affecting the input/output characteristics of the circuit. This algorithm recues the CP but not changing the latency of the system.

3. *Unfolding Algorithm*

Direct implementation of unfolding may lead to long iteration bound with lowest achievable CP.

Fig. 6, shows the architecture of CRC-15 in 3-point unfolded of 2-factor pipeline-cutset retimed and 4-level pipeline. Tables III and IV shows the output using the DSP algorithms to CRC-15 that is much better output compare with the existing paper serial implementation both for clock cycles and iteration bound respectively.

TABLE III. COMPARISON BETWEEN SERIAL TO OF CRC-15 FOR NUMBER OF CLOCK CYCLES

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 IV. COMPARISON BETWEEN CRC-9 AND CRC-15 FOR ITERATION BOUND

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

V. CONCLUSION

Parallel implementation is preferred for a high speed data transmission which is cannot be executed over serial operation due to its 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 which is appropriate for CAN controller. Fig. 6 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, analyze the design and its effects in different unfolding factors for hardware overhead.

REFERENCES

[1] P. Koopman. (2002). 32-bit Cyclic Redundancy Codes for Internet Applications. *Proc. IEEE International Conference on Dependable Systems and Networks* [Online]. pp. 459-468.Available:http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1028931

[2] FlexRay Consortium. (2010, October). FlexRay Communication System Protocol Specification Version 3.0.1 [Online]. pp. 114-115.Available:https://svn.ipd.kit.edu/nlrp/public/FlexRay/FlexRay%E2%84%A2%20Protocol%20Specification%20Version%203.0.1.pdf

[3] BOSCH. (2012, April). CAN with Flexible Data-Rate Specifications [Online]. pp. 12-13. Available: http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can_fd_spec.pdf

[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

[6] O. Pfeiffer, A. Ayre, and C. Keydel, "Underlying Technology: CAN," in *Embedded Networking with CAN and CANopen*, Copperhill Technologies Corporation, 2008, pp. 224-226.

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

[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.

[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)*. [Online]. pp. 160-162. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6118796&tag=1

[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)*. [Online]. pp. 5254-5258. Available: http://www.ijcsit.com/docs/Volume%203/vol3issue5/ijcsit2012030566.pdf

[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.* [Online]. pp. 1017-1021. Available:http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1715568&tag=1

[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).* [Online]. pp. 66-72. Available: http://iosrjournals.org/iosr-jvlsi/papers/vol2-issue5/J0256672.pdf