Implementation of EEDC for Trailer Segment in Enhanced FPGA-based FlexRay Controller

Ronnie O. Serfa Juan[1,2], Hi Seok Kim[1]
[1]*Department of Electronic Engineering, Cheongju University, Cheongju City 28503, South Korea*
[2]*ECE Department, College of Engineering, Technological University of the Philippines - Manila 1000, Philippines*
ronnieserfajuan@cju.ac.kr

***Abstract*—FlexRay is a time-triggered protocol that is a standard for reliable high-speed communications network especially for automotive applications such as Advanced Driver Assistance Systems (ADAS). This paper implemented using field-programmable gate array (FPGA)-based communication controller with a reconfigured trailer segment for a faster data rate. The FlexRay's trailer segments that use enhanced error detections and corrections codes that provide better functionality. This paper is implemented and verified using a Xilinx Spartan 6 FPGA. Experimental results show that this proposed implementation performs better regarding data rate and reduction of resource utilisation. This implementation represents an advancement in the FPGA-based system for vehicular applications and other time-triggered devices.**

***Index Terms*—Configured FPGA; Enhanced Error-Detection and Correction Code; FlexRay; Time-Triggered Devices.**

I. INTRODUCTION

Modularity in hardware architectures are the latest trends in automotive electronic systems [1], it aims to reduce the number of electronic control units (ECUs). The advanced driver-assistance systems (ADAS) domain as an example sets the fusion of central sensor along with its controller architecture for various applications. With all of these demands, a high-level protocol system like the FlexRay technology is required to cater the next-generation in-vehicle systems [2]. In Fujitsu of Next Generation Car Network, FlexRay features perform better against the event-triggered controller area network (CAN) and the set-up on network configurations is possible because the ECU can be easily accessed in a typical communication network set-up [3]. A versatile communication controller unit allows us to reconfigure, modify and enhance to resolve certain pitfalls and to maximise the hardware resources. A set-up like this can helps us to improve the overall quality and performance of any system. A reconfigurable platform like a field-programmable gate array (FPGA) based system can make it possible to improve the controller that is suitable for any applications but maintaining the required protocol to function the system accurately.

We present an enhanced FlexRay communication controller by a reconfigured trailer segment in the protocol engine module. This controller still functioning in regular mode but has an added features that augment beyond the capabilities of a conventional system. The trailer segment uses enhanced error detection and correction code instead of the original cyclic redundancy check (CRC) codes that hold a fixed 24-bits frame because of the CRC-24 polynomial generator.

With the field-programmable gate array (FPGA)-based system, the architecture can be flexible and can help us to reconfigure the hardware that provides an enhanced quality to compare with the existing communication controllers. Our experimental results show better features, such as faster data rate, low-latency data handling, and error detection performance that can be more efficient against with the standard FlexRay communication controllers.

The remainder of this paper is presented as follows. Section 2 is the brief discussion of the FlexRay protocol and related works. Section 3 discusses the proposed enhanced FlexRay controller. Experimental results and implementations are provided in Section 4. Then, Section 5, concludes this paper and outline of future work.

II. REVIEW OF RELATED WORKS

FlexRay has the combined advantage of the time-triggered, and event-triggered protocol [4] superseded the CAN protocol regarding higher data rates, versatile in topology, and fault-tolerant operation that needs for today's automotive industry.

A. FlexRay Protocol

The FlexRay protocol is developed and standardised by the FlexRay Consortium [5]. Also, it is designed to be? An efficient network between ECUs in vehicles embedded systems. The automobile manufacturers have developed the Automotive Open Systems Architecture (AUTOSAR) [6][7] to ECU architecture and software platform. Thus, the automotive industry depends on the reliability of its communication to be free from any flaws that may lead to inevitable fatalities.

The communication cycle or timing hierarchy of FlexRay protocol is shown in Figure 1. The static segment guarantees of service based on ID and assurance of real-time transmission. Every ECU can send a frame of data in one or more slots assigned to it [8]. On the dynamic segment, the communication works similarly to a CAN protocol, and the mini-slot is adjustable based on the transmitted data frame. The symbol window is a time slot with fixed duration, and it is used to transmit special symbols that can be transmitted on the network such as the "wake-up" pattern to start-up the communication. The network idle time is a specific time window that maintains the clock synchronisation.

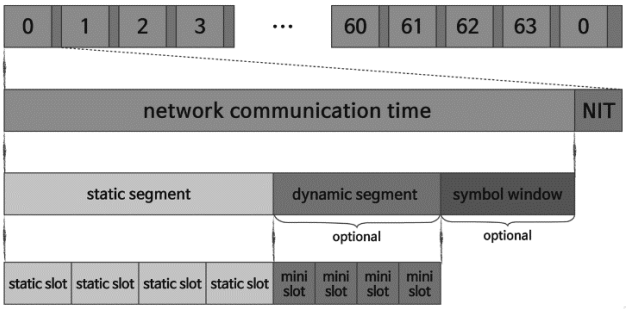

Figure 1: FlexRay Timing Hierarchy.

The FlexRay communication frame is responsible for exchanging information. This frame format is composed of three segments: the header, the payload and the trailer segment [9] as shown in Figure 2. It shows that the header section serves for status information such as bits indicating if the frame is null or if the frame needs to synchronise [10]. It also indicates the variation of the information to be transmitted in this payload segment that varies from 0 to 254 bytes. The payload contains the data to be transmitted. Finally, the trailer section contains the 24-bit CRC uses for error detection, and it is calculated between the payload and header sections [11]. In this scheme of the communication cycle, it leads to efficient bandwidth utilisation because the mode of transmission uses time division multiple access (TDMA). FlexRay can operate not only as a single channel system but also in dual channel mode [12].

B. *FlexRay Scheduling*

A configured design can arrange the divided subsystem into scheduling hierarchy. One approach for FlexRay hierarchical scheduling has been presented in [13]. Also, an example of hierarchical scheduling techniques is shown in [14]. A time-triggered schedule was generated analogously to the FlexRay protocol in [15]. Various approaches have been published to optimise the FlexRay's static and dynamic segment [16]-[18]. These papers presented different strategies about FlexRay scheduling. Moreover, a proposed model for the Distribution Data Service Middleware and its various entities that can exchange Data Objects on the communication network is presented [19]. This proposed algorithm presented a new scheduling method by combining the two old scheduling methods dealing with the Time-Triggered and the Event-Triggered Scheduling for FlexRay network.

C. *FlexRay Communication Controller Implementation*

The implementations on both [20],[21] described the application of FlexRay communication controller and description of language as the platform and translated into hardware. However, the developed hardware is not described in this paper. Another hardware set-up has also been discussed in reference [22]. Its communication controller node together with the finite state machine, including the simulation and the synthesised results on Xilinx tool, is presented here, but the discussion about the hardware configuration is not included. This paper provides a reconfigurable FlexRay communication controller, which features a modified version of the communication controller. The proposed method optimised the FlexRay communication controller regarding resource utilisation reduction includes the efficiency of the data rate by minimising the transmitted frame bits.

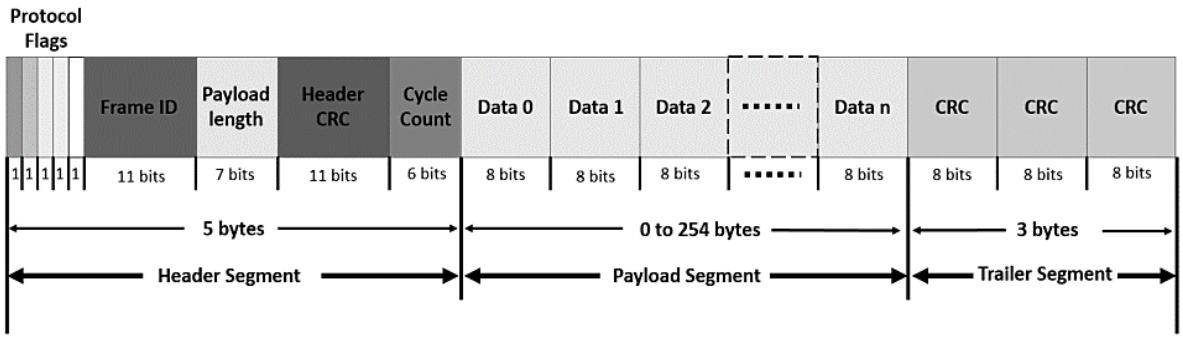

Figure 2: FlexRay Frame Format.

A running ECU hosts a normal FlexRay module on a pre-defined algorithm. The ECU controls and manages the connected sensors and actuators to the networks. Also, it monitors the status of the communication controller and the bus drivers including some conformity when initially sending a data [23]. The bus driver segment provides the control of the bit streams between link levels. The FlexRay core should be configurable to maximise its efficiency. The Xilinx FlexRay controller architecture is shown in Figure 3, comprises of user interface (UI), controller host interface (CHI), and protocol engine (PE) unit. The UI module provides on-chip peripheral bus (OPB) connectivity to the FlexRay controller. Also, it performs OPB Read/Write transactions and interrupt management.

The CHI manages the message to be received and to be transmitted, controls and configures the data flow between the host processor and the PE. This CHI module contains four major blocks namely Memory Map and Registers; Transmit Buffers; Receive Buffers, and Receive FIFO. The memory map and register module contain the control, status, and configuration memory map and registers. It allows read and writes access to control the following register sets such as the protocol and general CHI configuration, including its control and status registers. The transmit (T_X) buffers provide storage and control of message data to be transmitted over the FlexRay physical interface. The FlexRay controller provides up to 128 user-configurable T_X buffers, each of which can store one FlexRay frame with a variable payload length. The Number of T_X Buffers parameter can configure the number of T_X buffers in the controller. The maximum Number of T_X Buffers in the controller depends on the Maximum Payload Size parameter. Then, the receive buffers provide storage for received message data and perform the frame storage and filtering functions. The number of R_X Buffers is user-

configurable up to 128, each of which can store one FlexRay frame of variable payload length. The number of Receive buffers in the controller is configured using the Number of R_X Buffers parameter. Each Receive Buffer has a filter that can be configured based on a set program to accept any combination of Frame ID, Cycle counter and Message-ID values for storage in the buffer. Four acceptance filter pairs are also provided to screen incoming messages from the Protocol Engine for storage into the R_X FIFO. Each pair contains a mask and data pair for Frame ID, Cycle counter, and Message-ID.

Moreover, the FlexRay controller provides a flexible, with a variable length FIFO buffer for storage of the received FlexRay frames under the Receive FIFO. The message storage space allocated to the Receive FIFO is configurable. The PE module composes of all function blocks that perform based on the prescribed specifications. These are the media access control (MAC), bitstream encoder (BSE), bitstream decoder (BSD), frame and symbol processing (FSP), protocol operation control (POC), wakeup and startup (WUS) and clock synchronisation process (CSP).

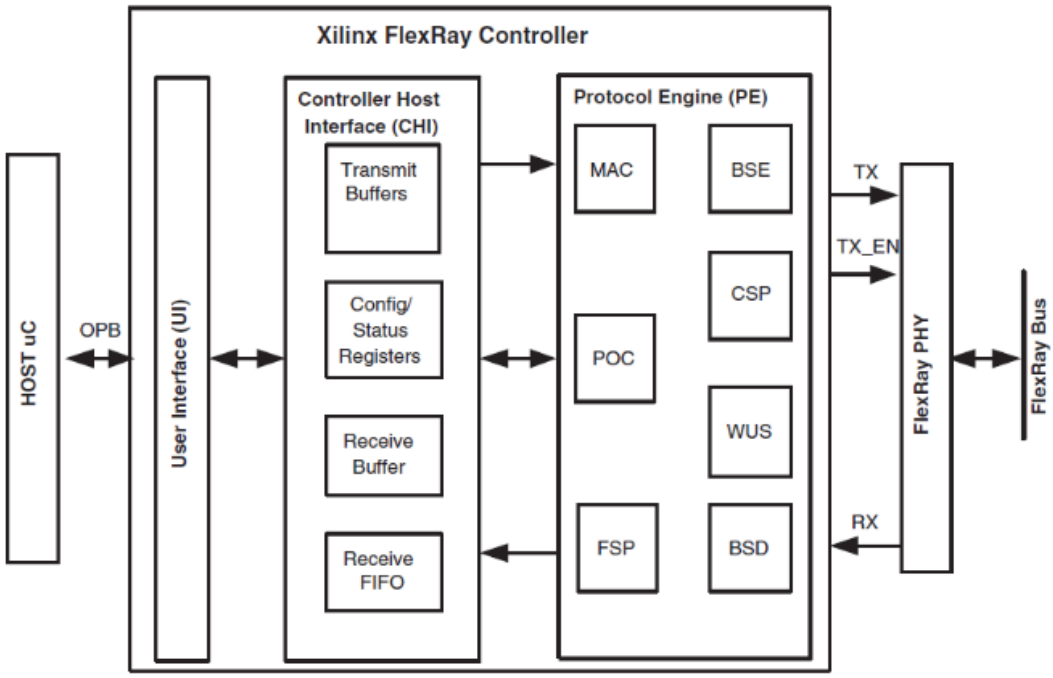

Figure 3: FlexRay Controller Architecture.

The MAC is assigned to maintain the timing within each communication cycle (static, dynamic, symbol window and network idle time). It manages the timing of static, dynamic slot duration and the node transmission operation. It also handles the assembled frames for transmission and asserts signals to other modules to indicate the static segment start, slot boundary, dynamic segment start, symbol window start and network idle start time. The BSE module handles encoding frames and symbols. The FSP is divided into frame and symbol encoder. The frame encoder takes the incoming bytes from the MAC and assembles the FlexRay bitstream like appending a transmit start sequence (TSS) to start of the bitstream or calculating frame CRC and appending it to the end of the frame. The symbol encoder can transmit three types of symbols such as collision avoidance symbol (CAS), media test symbol and wakeup symbol (WUS). The BSD block performs sampling and majority voting, bit clock alignments (BCA) and bit strobing, frame and symbol decoding, channel idle detection decoding error detection. The FSP checks the correct timing of frames and symbols concerning the TDMA scheme. This is done by applying further syntactical tests to received frames and verifies the semantics of the correctness of received frames. The results are then signalled to the host. Moreover, FSP provides status indicators to the host like the valid frame, valid symbol, syntax error, content error, boundary violation and TX conflict.

The Protocol Operation Control (POC) is an interface between the host and FlexRay PE sub-modules. It determines the operational state of the controller. The POC state transitions are controlled synchronously and occur as a consequence of a host command or an error condition occurring. The state transitions indicate a change in the operation of the protocol engine. See Figure 4 for an illustration of the POC state machine. The wakeup procedure describes the transition of a node from sleep mode to operational mode. The startup procedure describes the integration of nodes into the cluster. The CHI triggers both Wakeup and Startup (WUS) procedures. The CSP handles the synchronisation of all nodes in a cluster ensuring a common time. This is done by using sync frames sent by nodes on the network. The two components of time adjustment that are handled by the CSP are Rate correction and Offset correction. Clock synchronisation process has two significant sub-processes that perform the synchronisation functions, namely clock sync processing (CSP) and clock sync startup (CSS).

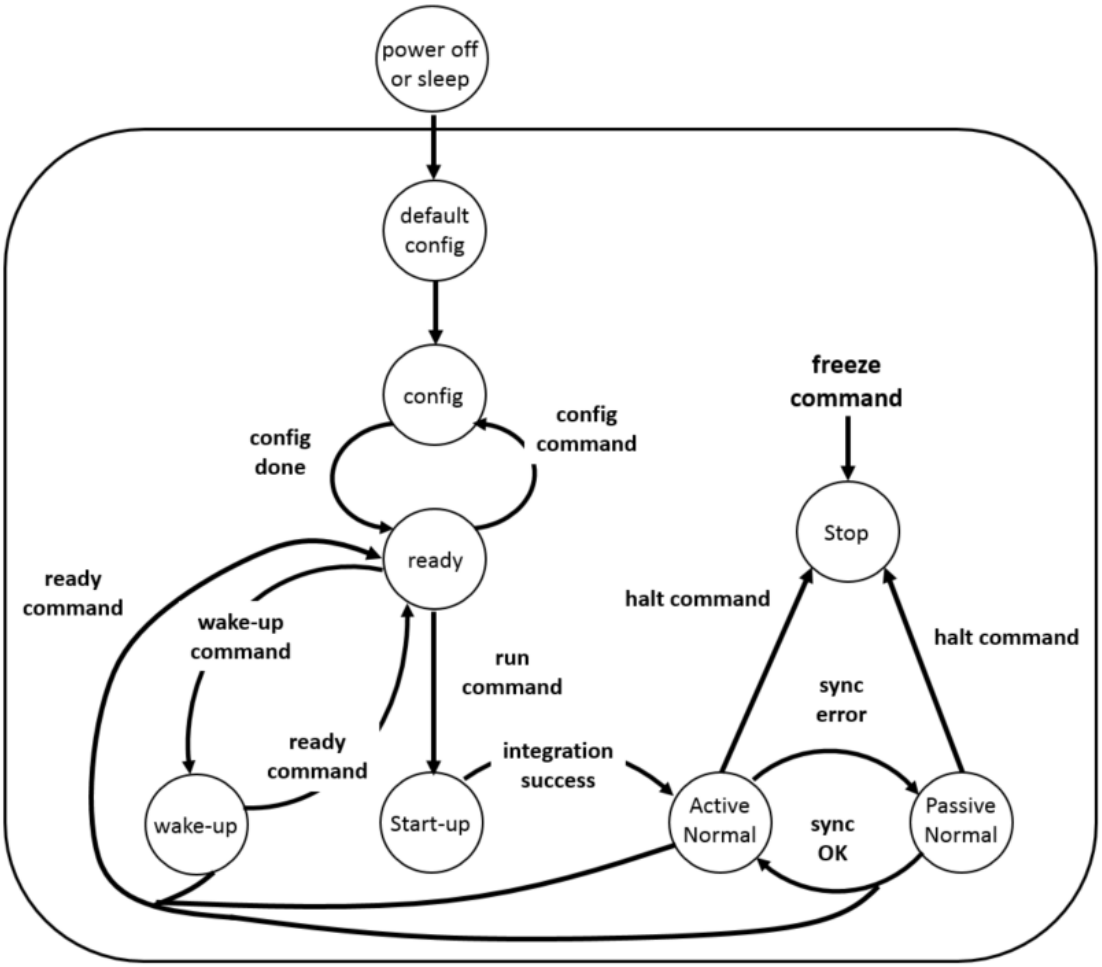

Figure 4: FlexRay Transition State.

III. Proposed Architecture

A. Configured FlexRay Communication Controller

Specific parameters are needed in optimising and configuring the communication controller like the removal of some unused hardware. In this paper, we configured the protocol engine module that contains the BSE. Since the BSE module handles the encoding of frames, it calculates the CRC frame and to be appended it to the end if the frame to be transmitted. We implemented alternative error detection instead of the conventional CRC. This proposed error detection is explained in the next section.

B. Implementation of Enhanced Error Detection and Correction (EEDC) Code

In the FlexRay frame format, the trailer segment contains a 24-bit CRC that protects the complete frame from any unwanted errors. Cyclic Redundancy Checking code is an example of polynomial codes referring to the corresponding polynomial of a codeword [24]. It represents every codeword $C(x) = C_{n-1}C_{n-2}...C0$ as a polynomial of degree n-1 as shown in Eq (1). The key idea is to ensure that every valid code polynomial is a multiple of a generator polynomial $g(x)$.

$$C(x) = \sum_{i=0}^{n-i} C_i x^i \quad (1)$$

Although this polynomial code is the basis for robust error-correction methods, still it has a fixed number of check bits because based on the nth degree of the generator polynomial that is needed to attach during transmission, like in FlexRay protocol, it uses a polynomial generator of CRC-24. With this, it reduces the transmission rate of the network. Moreover, CRC codes do not implement correction, and it only enforces retransmission whenever an error is detected. The encoding process is to construct a message polynomial $m(x)$ from a given message using the same method as shown in Equation (1).

On the other hand, Hamming code uses redundancy bits 'r' or parity bits that are added to an n-bit data D, it forms a codeword $D + r$ and follows the required number of 'r' in a condition shown in Equation (2).

$$2r \geq D + r + 1 \quad (2)$$

These redundancy bits are inserted at bit positions in power of 2 with the original data bits [25]. Then, all other bit positions are assigned to the data (i.e., 3, 5, 6, 7, 9, 11, etc.). As a result, an increase in overhead because of interspersing the redundancy bits both for the transmitter and receiver parts.

This paper proposes an alternative error detection code that eliminates the drawbacks above which is name as the enhanced error detection and correction (EEDC) code. In EEDC, it shows that for every valid codeword C bits, it contains a valid input data bits Di., for every valid Di entity,

there are *C* bits that can be changed to give an invalid codeword. Therefore, the total number of codewords corresponds to a valid data entity is *C + 1*.

As there are *2Di* valid data patterns, the total number of codewords is *(C + 1)2Di*. In *Di* bit codewords, the possible number of patterns is *2C,* and because of this, it limits the number of valid bits plus the invalid codes that can exist. Thus, Equation (3) shows that

$$(C + 1)2Di \leq 2C \tag{3}$$

then, it can be written as

$$C = Di + r \tag{4}$$

and

$$(Di + r + 1)2Di \leq 2Di+r \tag{5}$$

and the total number of the required *r* should satisfy first the given condition of this inequality shown in Equation 6

$$(Di + r + 1) \leq 2r \tag{6}$$

The data information *Di* together with the required *r* will be constructed into a polynomial form with a degree of n-1 such as a summation of *D(x)* and *r(x)*. The degree of polynomial *D(x)* will increase the nth value of *r* in such that

$$G(x) = D(x) \bullet Xn \tag{7}$$

Therefore, to complete form of this proposed EEDC code, it follows Equation (8).

$$EEDC\ codes = G(x) + r(x) \tag{8}$$

The proposed algorithm of this proposed EEDC code is shown in Figure 5.

IV. Experimental Testing and Results

To validate our design, we implemented the design in Xilinx Spartan 6 (XC7Z020CLG484-1) FPGA. During the testing, we compared the detection of an error in three different error codes. Figure 6 shows the error detection performance of the enhanced error detection correction (EEDC) code is better especially in long runs of bits streams against the conventional CRC of FlexRay protocol and in using Hamming codes. Table 1 shows the resource utilisation using Xilinx Spartan 6 FPGA from the simulation.

Table 1
Hardware Utilization

Resources	Utilization	Available
LUT	6, 189	53,200
LUTRAM	606	17,400
FF	4,337	106,400
DSP	2	220
IO	152	200
BUFG	2	32

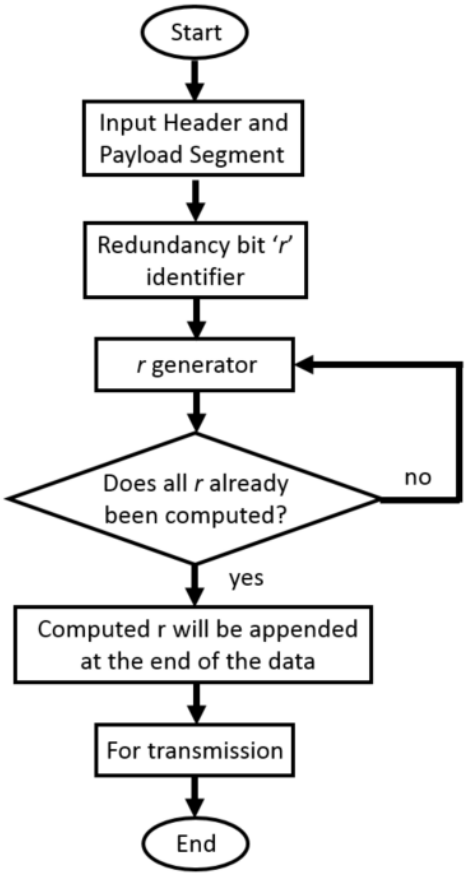

Figure 5: Proposed Algorithm of EEDC.

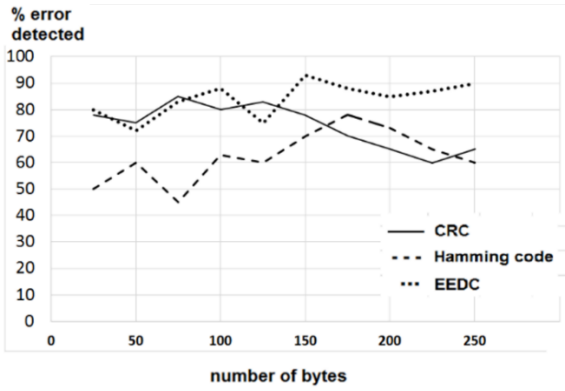

Figure 6: Error Detection Performance.

V. Conclusion

In this paper, we have discussed the FlexRay protocol and some related works on scalable implementations. In order to design a customised communication controller, we determined the contents and functions of each module in the controller. We customised a FlexRay communication controller that has the edge over with other existing controllers since we used a different approach on the trailer segment using using enhanced error detection correction code. The results lead to the reduction of hardware utilisation. It also increases the required transmission rate. Moreover, the detection of errors is much higher compared with the two error codes.

In the future, our enhanced FPGA can be useful to others regarding research about FlexRay. We also intend to use the enhanced error detection and correction code in other reconfigurable hardware for more advanced network setups.

REFERENCES

[1] R. Hedge, G. Mishra, K. S. Gurumurthy. Software and Hardware design Challenges in Automotive Embedded System. International *Journal of VLSI design and Communication System*. 2 (3) (2011) 165-174.

[2] FlexRay Consortium. *FlexRay Communications Systems – Protocol Specifications*. Version 2.1 Revision A. (2005) [Online]. Available: http://www.flexray.com

[3] Fujitsu. *Next Generation Car Network – FlexRay*. [Online]. Available: http://www.fujitsu.com/downloads/CN/fmc/lsi/FlexRay-EN.pdf

[4] J. Kotz, S. Poledna. Making FlexRay a Reality in a Premium Car. SAe *International Conference of the Convergence Transportation Electronics Association*. (2008) 391-395.

[5] M. Grenier, L. Havet, N. Navet. Configuring the communication on FlexRay – The Case of the Static Segment. *4th European Congress on Embedded Real – Time Software*. (2008) 1-18.

[6] R. Shaw, B. Jackman. An Introduction to FlexRay as an Industrial Network. *IEEE International Symposium on Industrial Electronics*. (2008) 1849-1854.

[7] S. Choosang, R. Taburan, S. Gordo. A formal model of an AUTOSAR in vehicle in vehicle communication protocol. *International Conference on Information and Communication Technology for Embedded System*. (2010).

[8] M. Lukasiewycz, M. Glab, J. Teich, P. Milbredt. FlexRay Schedule Optimization of the Static Segment. *7th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis*. (2009) 363-372.

[9] Rishvanth, D. Valli, K. Ganesan. Design of an In-Vehicle Network (Using LIN, Can and FlexRay), Gateway and its Diagnostics Using Vector CANoe. *Americal Journal of Signal Processing*. 1 (2) (2011) 40-45.

[10] D. C. Liaw, I. C. Liu, K. L. Chang. The FlexRay Implementation of By-Wire System for Electric Vehicle. *World Electric Vehicle Journal*. 5 (2012) 610-616.

[11] A. Joseph. Reliable FlexRay Communication Controller for Automotive Systems. *International Conference on Engineering Innovations and Solutions*. (2016) 107-112.

[12] B. Vermeulen, J. Staschulat, M. Struck, S. Lorenz. FlexRay Switch, *More bandwidth and better robustness in FlexRay networks*. Springer Automotive Media. (2010) 32-36.

[13] A. Easwaran, I. Shin, O. Sokolsky, I. Lee. Incremental schedulability analysis of hierarchical real-time components. *6th AMC & IEEE International Conference on Embedded Software*. (2006) 272-281.

[14] M. Anand, S. Fischmeister, I. Lee. A comparison of compositional schedulability analysis techniques for hierarchical real-time systems. *ACM Transactions on Embedded Computer System*. 13 (1) (2013) 1-37.

[15] P. Mundhenk, F. Sagstetter, S. Chakraborty. Policy-based Message Scheduling using FlexRay. *International Conference on Hardware/Software Codesign and System Synthesis*. (2014).

[16] Texas Instrument. *FlexRay Module Training*. (2015). [Online]. Available: http://www.ti.com/lit/ml/sprt718/sprt718.pdf

[17] K. Schmidt, E. G. Schmidt. Message scheduling for the FlexRay Protocol: The Static Segment. *IEEE Transactions on Vehicle Technology*. 58 (5) (2009) 2170-2179.

[18] E. G. Schmidt, K. Schmidt. Message scheduling for the FlexRay Protocol: The Dynamic Segment. *IEEE Transactions on Vehicle Technology*. 58 (5) (2009) 2160-2169.

[19] W. N. Rabai, R. Bouhouch, H. Jaouani, S. Hasnaoui. Static and Dynamic Scheduling for FlexRay Network using the Combined Method. *International Journal of Information Technology and System*. 1 (1) (2012) 16-24.

[20] Y. N. Xu, Y. E. Kim, K. J. Cho, J. G. Chung, M. S. Lim. Implementation of FlexRay Communication Controller Protocol with Application to a Robot System. *15th IEEE International Conference on Electronics Circuits and Systems*. (2008) 994-997, 2008.

[21] J. H. Park, C. W. Moon. Implementation of an Integrated Controller for a Robot Hand Base on a Vehicle Communication System. *International Journal of Control and Automation*. 7 (11) (2014) 287-298.

[22] M. Khanapurkar, J. Hande, P. Bajaj. Approach for VHDL and FPGA Implementation of Communication Controller. *Second International Conference on Emerging Trends in Engineering and Technology*. (2009) 397-401.

[23] M. Heinz, M. Hillenbrand, P. Brunn, K. Mueller-Glaser. A FlexRay parameter calculation methodology based on electric/electronic architecture of vehicles. *6th IFAC Symposium Advances in Automotive Control*. (2010) 407-412.

[24] P. Koopman, T. Chakravarty. *Cyclic redundancy code (CRC) polynomial selection for embedded networks*. [Online]. Available: https://users.ececmu.edu/~koopman/roses/dsn04/koopman04_crc_poly_embedded.pdf, accessed August 2016

[25] Wang, Hongli. "A kind of performance improvement of Hamming code." *Information and Management Engineering*. Springer, Berlin, Heidelberg, (2011) 315-318.