International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-3 Issue-6, August 2014

FPGA Implementation of Frame Decoding Behaviour of Flex Ray Communication Protocol

Deepa M Raju, Abraham C G, V Suresh Babu

Abstract — This paper has highlighted the concept of Frame decoding behaviour of Flex Ray Communication Protocol. The VHDL model of Flex Ray frame decoder of Flex Ray Communication Controller is designed. The design is simulated using ModelSim Altera Edition 13.0 and synthesized using Quartus II 13.0.0.156. The frame decoding behaviour is implemented using Stratix IV GX FPGA. This project design is made with the intention of development of low power; high performance FPGA for decoding the data transmitted which will be a basic for the development of Flex Ray communication controller.

Index Terms—Area Efficient, FPGA, Low power, VHDL Language

I. INTRODUCTION

Safety-critical driver assistance functions with electronic interfaces to the chassis place extremely stringent requirements on the reliability, safety and real-time capability of the communication system [5]. An important property of Flex ray is composability, whose core function is to guarantee deterministic and fault tolerant data communication independent of bus load. CAN (Controller Area Network), the communication technology that has become established in the automotive field, cannot fulfill this challenging set of requirements, since CAN is based on an event-driven communication approach, which means that every bus node of a communication system must be able to access the common communication medium at any time. Event-driven communication systems enable quick reaction to asynchronous processes, but they are non-deterministic. Event-driven communication systems do not exhibit the property of composability.

CAN communication technology cannot fulfill the high requirements for fault tolerance due to its lack of redundant structures and mechanisms and can also only deliver a maximum data rate of 500 Kbit/s in series production. That is why BMW and DaimlerChrysler in 1999 agreed to work together to advance the specification and development of a future, uniform, time-triggered and fault tolerant communication technology. This cooperative effort resulted in the first rough requirements specification for Flex Ray. The Flex Ray Consortium comprises seven core partners: BMW, Bosch, Daimler, Freescale, General Motors, NXP Semiconductors and Volkswagen.

Manuscript published on 30 August 2014.
***** Correspondence Author (s)
Ms. Deepa M Raju, Electronics and Communication Engineering, St.Joseph's College of Engineering and Technology, Palai.
Mr. Abraham C G, Electronics and Communication Engineering, St.Joseph's College of Engineering and Technology, Palai.
Dr. V Suresh Babu, Electronics and Communication Engineering, College of Engineering Trivandrum, Palai.

Over time, many Premium Associate Members and Associate Members have joined the Flex Ray Consortium.

The frame decoder uses the input clock of 50 MHz. The design is simulated using Altera's ModelSim and synthesis using Quartus II 13.0.0.156. The Stratix IV GX FPGA [1] is used for the design configuration.

II. FLEX RAY COMMUNICATION PROTOCOL

The growing interplay between electronic systems and replacement of mechanical components by electronic components heightens the complexity of automotive electronic systems. Flex Ray, the communication technology support both TDMA and Event Triggered Communication. The core property of Flex Ray technology is to guarantee the main fields of application of Flex Ray are extremely safety-critical and time-critical automotive applications. Another important property of Flex ray is composability, whose core function is to guarantee deterministic and fault tolerant data communication independent of bus load. To minimize the risk of failure, Flex Ray provides for redundant layout of the communication channel. Each of the two communication channels may be operated at a data rate of up to 10 Mbit/s. However, as an alternative this redundant channel may be used to increase the data rate to 20 Mbit/s. The choice between fault tolerance and increased transmission rate can be made individually for each Flex Ray message. Due to its high data rate of 10 Mbit/s, plans also call for establishing Flex Ray as a data backbone in the automobile [10].

A. Communication Controller

The Communication Controller relieves the host of all communication tasks. The Communication Controller is connected to the host via the so-called CHI (Controller Host Interface). On-board the CHI there is user-configurable buffers for TX and RX messages. The core of the Communication Controller is the protocol engine. It consists of several communication components. The media access control (MAC) component for bus access [2]. The coding component handles coding of the bytes obtained from the MAC. The decoding component handles decoding of the logical bit stream received by the Flex Ray transceiver. The frame and symbol processing component checks for conformance to the communication cycle upon which the Flex Ray cluster is based. It also checks the RX messages for transmission errors.

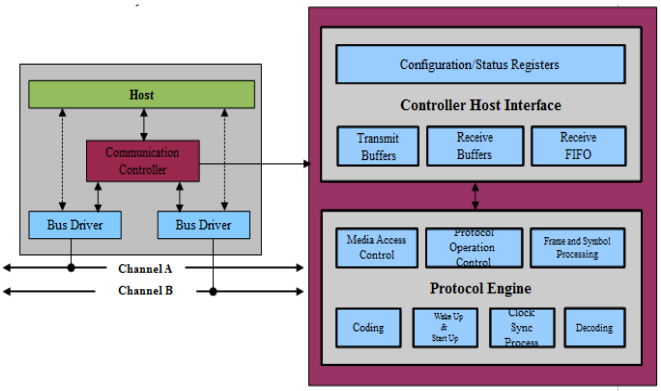

Fig.1: Flex Ray Node

The clock synchronization process component provides for synchronization of the Flex Ray nodes. The wake-up and start up component handles wake-up and start up. The Fig. 1 shows the Flex Ray Node.

B. Flex Ray Bus

Flex Ray technology is designed for data rates up to 10 Mbit/s. It uses unshielded twisted pair wire. The Flex Ray cluster uses the differential signal transmission, so the Flex Ray Bus consists of two lines: Bus Plus (BP) and Bus Minus (BM). Twisting of two lines reduces the magnetic field, so twisted line pairs are typically used in practice.

C. Flex Ray Bus Driver

It is not possible to connect a Flex Ray Communication Controller directly to the physical transmission medium. While the Flex Ray Communication Controller works with binary signals, the physical transmission medium uses differential signal transmission. The Flex Ray Bus Driver converts the logical signal stream received from the Flex Ray Communication Controller into a differential signal stream. In the other direction, it converts the physical differential signal stream received from the Flex Ray bus into a logical signal stream.

D. Flex Ray Data Framing

Data transmission in a Flex Ray cluster is executed using a uniform message frame (Fig.2). Each Flex Ray message is composed of three parts: header, payload and trailer. The header consists of 40 bits [3].

A maximum of 254 user bytes (payload) can be transported by one message. The payload length parameter shows the size of the payload in words. The payload length exhibits the same value for all messages transmitted in the static segment. To protect the payload, the CRC method (CRC: Cyclic Redundancy Check) is used. A CRC sequence is computed based on the header and payload and a generator polynomial defined by the Flex Ray specification. This CRC sequence is appended to the header and payload as the trailer. The CRC sequence for a message corresponds to a multiple of the header and the payload. The receiver of the message can detect any transmission error with very high reliability.

E. Flex Ray Frame Coding in Static Segment

Physical transmission of a message begins with the transmission start sequence (TSS).

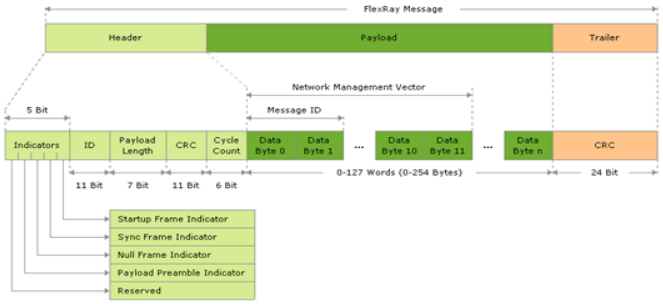

Fig. 2: Flex Ray Message

In the case of a Flex Ray cluster based on an active star topology, this is used to prevent the active star coupler from not being able to transport the first bits of a message from the RX branch to the TX branches. The reason for this is that an active star coupler requires a certain amount of time to reach its operating state (star truncation). In a Flex Ray cluster with an active star coupler, the TSS must at least match the star truncation — a minimum of 3 and a maximum of 15 bits of low level.

The TSS is terminated with the frame start sequence (FSS). Following the FSS, the header can be transmitted. The byte start sequence (BSS) precedes each byte to be transmitted. The receiver uses the edge change created by this sequence for re-synchronization. The end of a message is marked by frame end sequence (FES). Eleven recessive bits in the static slot (Channel Idle Delimiter) indicate that the communication medium is available. The Fig. 3 shows a static message that depicts the code elements necessary for physical transmission.

III. FLEX RAY FRAME DECODING METHODOLOGY

The decoding of binary data stream is converting the binary data stream into communication elements such as frames and symbols [4]. Here four types of data frames are considered such as Whole Frame, Broken Frame, Symbol and Raw Data. The process of frame decoding can be divided into three parts:-

- Sampling
- Bit strobing
- Frame decoding

First the signal is sampled in the sampling process to prevent small glitches in the signal. Then the signal is sent through the bit strobing process which synchronizes the bit timing of the output value.

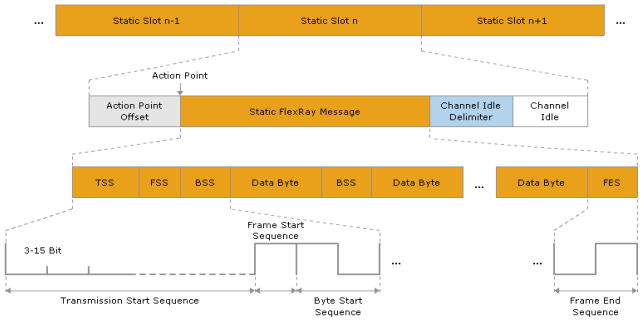

Fig. 3: Frame Coding in static segment

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

Then through the frame decoding process which checks the different parts of the frame and determine the type of data.

A. *Flex Ray Frame Decoding in Static Segment*

The decoding process interprets the bit stream observed at the voted RxD input of the node. The decoding can be mainly classified into frame decoding and symbol decoding. The decoding process does not perform concurrent decoding of frames and CAS/MTS symbols [6], i.e. for a given channel the process shall support only one decoding method (frame or CAS/MTS symbol) at a time.

A frame starts with the first strobed LOW bit after channel idle. The channel idle delimiter refers to the time required by the idle detection mechanism to determine that the channel is idle. In order for idle to occur all bits strobed during the channel idle delimiter must be strobed as high.

The bit stream received can also be a symbol. There are mainly two types of symbol are considered, collision avoidance symbol and media access symbol. The node shall decode the CAS and MTS symbols in exactly the same manner. Since these symbols are encoded by a LOW level of duration cdCAS starting immediately after the TSS, it is not possible for receivers to detect the boundary between the TSS and the subsequent LOW bits that make up the CAS or MTS. As a result, the detection of a CAS or MTS shall be considered as valid coding if a LOW level with duration between cdCASRxLowMin and gdCASRxLowMax is detected.

The data frame of communication element includes different frame sequences while they are encoded. Hence the data decoding involves the decoding of each sequences. The different sequences are shown in fig. 3.

Transmission Start Sequence (TSS)

It is used to initiate a proper connection setup through the Flex Ray communication system. The purpose of the TSS is to open the gates of an active star, i.e., to cause the active star to properly set up branches. During this set up, an active star truncates a number of bits at the beginning of a communication element. The TSS also prevents the content of the frame or symbol from being truncated by the receiving communication module.

Frame Start Sequence (FSS)

It is used to truncate TSS and compensate a possible quantization error.

Byte Start Sequence (BSS)

It is used to provide a bit stream timing information to the receiving ‘communication module’.

Frame End Sequence (FES)

It is used to mark the end of the last byte sequence of a ‘frame’.

IV. Design of Flex Ray Frame Decoder

In the frame decoding process the different parts of the frame are checked and determine the type of data. The data type can be whole frame, broken frame, symbol and raw data. A whole frame is a frame that can be decode from FSS to FES. A broken frame has a correct FSS and is decoded in the same way but do not contain an FES. But instead there is some kind of fault. As a fault is found the already decoded part of the frame is kept but all bits after this point, unless channel is idle, are decoded as RAW data. A symbol is a single high bit after a given number of low bits. Symbol is recognized at the first high bit within the CAS. Raw data are data not recognized as any other type. Data of this type will have a time stamp at the raw data recognition point and simply contains any data found until a channel becomes idle.

It involves first the TSS_Decoding, Symbol decoding, FSS_BSS_check, Byte_decoding, BSS_FES_check and RAW_data check. The flow chart describing the different states of static frame decoder is shown in Fig.4.

A. *Start State*

In the initial Start state certain data fields are initialized and signals are reset. If the required conditions are met it will move from start state to TSS_Decoding state.

B. *Transmission Start Sequence (TSS) Decoding State*

In this state checks if the TSS is correct. TSS is allowed to have up to *gdTSSTransmitter* (variable between 3 and 15) zero bits followed by a one bit.

In this project 15 bits are considered. If it is more *than gdTSSTransmitter* zero bits, the package is not a frame [9]. But it can be a symbol.

The *gdTSSTransmitter* is set when the Flex Ray system is configured. If the first one bit comes before too many zeroes have passed the process will go to the FSS_BSS_check state. If the first one bit comes after the length of *gdTSSTransmitter* [10] the process will go to Symbol_Decoding state.

C. *Symbol Decoding State*

In symbol decoding state it checks whether the data package is a symbol or a false frame. Here two parameters are considered. They are *gdCASRxLowMax* (variable between 67 and 99) and *cdCASRxLowMin* (constant of 29).

If the number of zeroes are between *gdCASRxLowMax* and *cdCASRxLowMin* followed by a one bit the data package is considered as a symbol and symbol identification is sent out and the process continues in the RAW_data state. If the number of zeroes are greater than *gdCASRxLowMax* and less than *cdCASRxLowMin* [11] the frame is considered as false and the process continues in the RAW_data state. In this case the process sends out a frame false signal.

D. *Frame Start Sequence and Byte Start Sequence Check State (FSS_BSS_check)*

In this state it checks presence FSS, a single high bit and BSS, a high and a low bit. If FSS_BSS_check is OK the process will continue with Byte decoding state. Otherwise the frame is considered as false frame and the process will go to RAW_data state. Also a false frame signal is send out.

E. *Byte Decoding State*

In this state it loops through 8 bits (one byte) and sends out each bit after it arrives from the bit strobing process. Byte decoding state and RAW data state is the only states that send out data bits. After the byte decoding state it continues to BSS_FES_check state.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

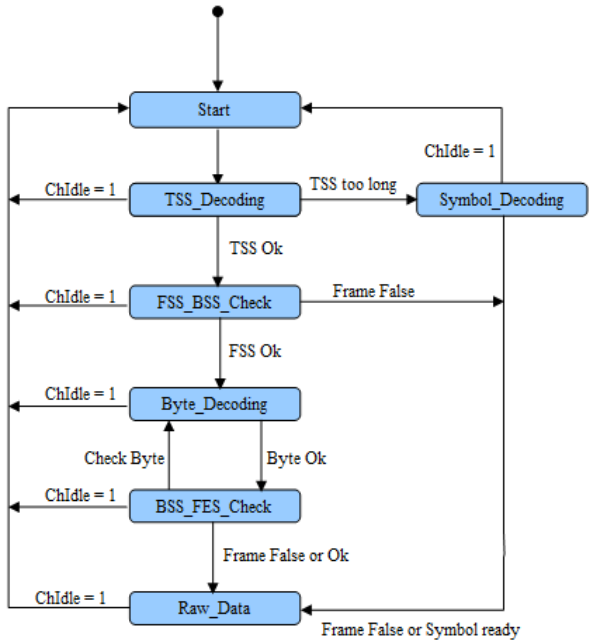

Fig.4: States of the Static Frame Decoder

F. Byte Start Sequence and Frame End Sequence Check State (BSS_FES_check)

After every data byte is send it checks whether the next bits represents BSS or an FES. If the sequence is a high bit and then a low bit it is a BSS. If the sequence is a low bit and then a high bit it is an FES.

The BSS indicates the presence of another byte of data. If a correct FES is detected a signal is sent out telling that a complete frame has been received. After an FES is detected the process continues to RAW_data state to see if there are any more bits to store as raw data after the frame. If there are no more bits coming and the signal channel idle goes high the frame decoder process continues to the Start state and waits for the next bit. After a BSS the process increase the byte counter and keeps track of how many bytes the frame contains.

The maximum number of bytes in a frame is 262 and if the counter exceeds 262 bytes the process sends out a signal telling that it is a false frame and continues to RAW_data state for the following bits. If a BSS or FES is not detected the process sends out a signal telling that it is a false frame and continues to RAW_data state for the following bits.

G. Raw Data State

This part takes care of all data bits which follow a correct frame, a false frame or a symbol. The state loops through the bits and sends out the bits until the channel idle is set high. The Table I [10] shows the frame decoder port description table which describes the inputs and outputs of the frame decoder. The Table II [11] shows the parameters used for the design of frame decoder.

V. SIMULATION RESULTS

In the frame decoding process the different parts of the received frame are checked and finally determine the type of incoming data. The data type can be whole frame, broken frame, symbol and raw data.

The output wave forms obtained from ModelSim Altera Edition 13.0 are shown below. The inputs and outputs of the frame decoder are shown in Table I.

TABLE I: FRAME DECODER PORT DESCRIPTION TABLE

Port name	Arribute	Description
clk	Input	Input clock of 50MHz frequency
chIdle	Input	Detect whether channel is Idle
strobed_bit	Input	Input data
strobed_flag	Input	Signal for transfer of data bit
zStrobed	Input	Strobed data
data_out	Output	Output data
data_flag	Output	Signal for retrieval of data bit
data_done	Output	Detects when data is present
data_type	Output	Detects the type of data

TABLE II: PARAMETERS USED

Parameters Used	Description	Value
cChannelIdleDelimiter	Number of high bits that marks start of transition from transmission on the bus to channel idle	11gdBit
cdCASRxLowMin	Minimum number of Low bits preceding a CAS/MTS symbol	29gdBit
gdCASRxLowMax	This is the maximum number of low bits preceding a CAS/MTS symbol. This value can be between 67 and 99. 99 is used in this project	99gdBit
gdTSSTransmitter	This is the number of low bit in the TSS. This value can be between 3 and 15. 15 is used in this project	15
cPayloadLengthMax	This is the maximum number of two byte words in the payload of a frame	127

A. Whole Data Frame followed by Raw Data

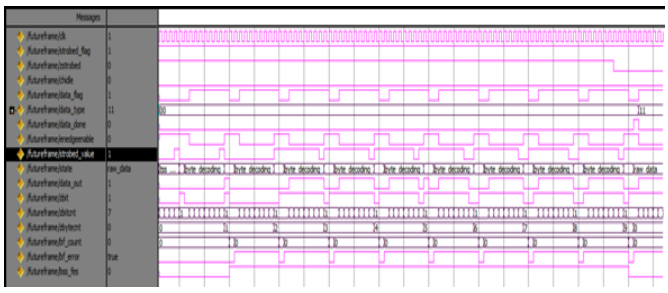

Fig.5: Output wave form of whole data frame followed by raw data

The whole data frame is decoded successfully from FSS to FES. After the data is decoded the state is moved to RAW data state by default.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

B. Broken Data Frame followed by Raw Data

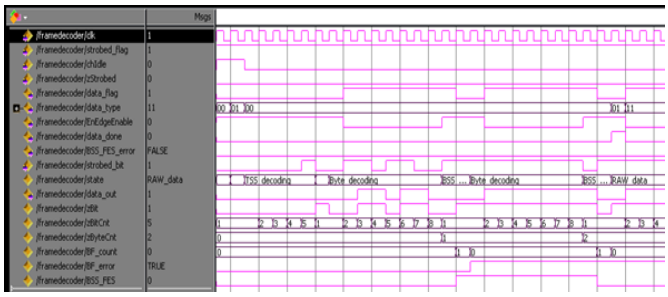

Fig.6: Output wave form of broken data frame followed by raw data

In the broken frame decoding the frame has correct FSS and is decoded in the same way as whole frame but do not contain an FES.

C. Symbol Decoding followed by Raw Data

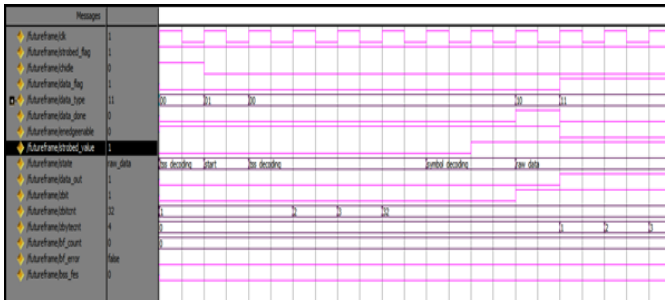

Fig.7: Output wave form of symbol decoding followed by raw data

If the number of zeroes of *zbitcnt* are between *gdCASRxLowMax* and *cdCASRxLowMin* followed by a one bit the data package is considered as a symbol and symbol identification is sent out and the process continues in the RAW_data state.

D. Raw Data Frame

In the raw data frame the frame contain any kind of data until channel idle is reached.

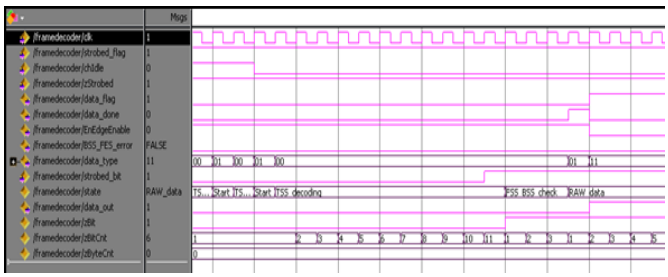

Fig.8: Output wave form of raw data frame

VI. HARDWARE IMPLEMENTATION

The Stratix IV GX FPGA evaluation board is used in this project, which provides a hardware platform for developing and prototyping the design. The board is powered by an external 14V – 20V DC power supply, PCI Express edge connector which interfaces to a PCI Express root port to an appropriate PC motherboard.

The Table III shows the analysis and resources usage summary.

TABLE III: ANALYSIS AND SYNTHESIS RESOURCE SUMMARY

Resources	Usage
Adaptive Logic Modules (ALMs)	66
Combinational ALUTs	94/182,400 (<1%)
Logic Registers	34/182,400 (< 1%)
Total Pins	12/888 (1%)
Combinational with no register ALUT/ Register pair	67
Combinational with register ALUT/ Register pair	27
Register only ALUT/Register pair	7

VII. FPGA CONFIGURATION

The Quartus II Programmer allows you to configure the frame decoder design in Altera Stratix IV GX FPGA device. The Quartus II software is used to generate programming *.sof* file that represent your design, and then use the Programmer to download the programming files to an Altera device over Altera programming hardware known as USB Blaster. The Fig.9 shows the result when the Stratix IV GX FPGA is successfully configured. The Fig.10 shows the Altera Stratix IV GX FPGA Evaluation board.

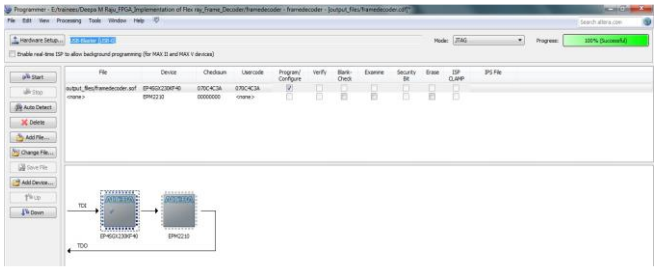

Fig.9: Stratix IV GX FPGA Configuration

Fig.10: Stratix IV GX FPGA Evaluation Board

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

VIII. Design Verification

The evaluation board provides many resources such as DIP Switches, Push button Switches, 16 User LEDs etc that are attached to the Stratix IV GX FPGA are used to test the design.

In this project four DIP Switches and six user LEDs are used for the design verification. The pin assignments of the frame decoder design are given in the pin planner.

IX. Conclusion and Future Enhancements

This paper has highlighted the concept of Frame decoding behaviour of Flex Ray Communication Protocol. The VHDL model of Flex Ray frame decoder of Flex Ray Communication Controller is developed. The design is successfully simulated and implemented using Altera Quartus II tools. The Stratix IV GX FPGA evaluation board is used for the configuration of frame decoder design. Hence this frame decoder design is feasible for the development of a Flex Ray Communication Controller. This work defines a basic for further design process. As the extension of the current project design the frame encoder can be developed.

It can be observed that the implementation of proposed architecture occupies only a small percentage of the corresponding device logic blocks, thus permitting the integration of additional control operations in the same IC. The different components in the communication controller can be incorporated together for the new Communication Controller stack development which includes the features of latest version v3.0 of Flex Ray Protocol Specification.

Acknowledgment

The authors would like to thank Dr. C J Joseph, Prof. Madhukumar S, Dr. Priestly Shan, Mr. Abraham C G, SJCET, Palai, Kerala, India for their support and motivation for this work.

References

[1] S. Shreejith, S. A. Fahmy, and M. Lukasiewycz, "Accelerating Validation of Time-Triggered Automotive Systems on FPGAs," in Proc. of the International Conference on Field Programmable Technology (FPT), 2013.
[2] J. Sobotka and J. Novak, "Flex Ray controller with special testing capabilities," in Proc. of the Conference on Applied Electronics (AE), 2012, p. 269 to 272.
[3] Dominique Paret. "Flex Ray and its Applications: Real Time Multiplexed Network", First Edition. © 2012 John Wiley & Sons, Ltd. Published 2012 by John Wiley & Sons, Ltd.
[4] Michael Gerke, "Flex Ray: Coding and Decoding, Media Access Control, Frame and Symbol Processing and Serial Interface", November 24,
[5] Bernhard Schatz, Christian Kuhnel, "Automotive Embedded Systems Handbook", Technical University of Munich, Michael Gonschorek, Elektrobit Corporation, 2007.
[6] Sergey Kosovo, "Flex Ray Communication Protocol" (Wake Up and Start Up).
[7] "Method of Synchronizing clock of different clusters", US Patent Application, 2009.
[8] Vector Training
[9] Flex Ray Communication System Protocol Specification Version 3.0.1 Flex Ray
[10] Flex Ray Communication System Protocol Specification Version 2.1 Flex Ray Consortium Revision A, Flex Ray Consortium Std., December 2005
[11] Flex Ray Communications System - Electrical Physical Layer Specification, v3.0.1, Flex Ray Consortium.

Ms. Deepa M Raju received the Bachelor of Engineering Degree in Applied Electronics and Instrumentation from Mahatma Gandhi University, Kottayam, Kerala, India, in 2011. She is currently doing M.Tech Degree in VLSI and Embedded Systems from Mahatma Gandhi University. Her areas of interest are FPGA, VLSI Design and Embedded System Design.

Mr. Abraham C G received the Bachelor of Engineering Degree in Electronics and Communication Engineering from M. K. University, Madurai, India, in 2004, the M. Tech. Degree in VLSI Design from Karunya University, Coimbatore in 2007.
Since 2005, he is Assistant Professor in Department of Electronics and Communication Engineering at St.Joseph's College of Engineering and Technology Palai, Kottayam, India. His areas of interest are VLSI System Design and Embedded System Design.

Dr. V. Suresh Babu received the B. Tech. Degree in Electronics and Communication Engineering from Kerala University, Trivandrum, India, in 1989, the M. Tech. Degree in Integrated Electronic Devices and Circuits from IIT Madras in 1996 and the Ph.D. Degree in Electronics and Communication in 2013 from Kerala University. From 1990 to 1991, he was with the Kerala State Electronics Development Corporation Ltd., Trivandrum, India. Since 1991, he has been a member of the faculty of the Department of Electronics and Communication Engineering, in Government Engineering Colleges in Kerala. Presently he is Associate Professor in College of Engineering Trivandrum, Kerala, India. His areas of interest are Analog Circuit Design and VLSI Signal Processing.