ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 2, February 2016

Scalable Flexray Communication Controller using CAN Protocol

M. Sowmiya, N. Kanagaraj

PG Scholar, Department of Electronics and Communication, Mookambigai College of Engineering, Pudukottai, India

Assistant Professor, Department of Electronics and Communication, Mookambigai College of Engineering, Pudukottai, India

ABSTRACT: Modern vehicles incorporate an increasing number of distributed compute nodes, resulting in the need for faster and more reliable in-vehicle networks. Time-triggered protocols such as FlexRay have been gaining ground as the standard for high-speed reliable communications in the automotive industry, marking a shift away from the event-triggered medium access used in controller area networks (CANs). Advanced applications can benefit from tight coupling of the embedded computing units with the communication interface, thereby providing functionality beyond the FlexRay standard. Such an approach is highly suited to implementation on reconfigurable architectures. The CAN Protocol can be used to provide the high speed communication between two or more devices.

KEYWORDS: Distributed Compute Nodes, Time-Triggered Protocols, Reliable Communication, Control Area Networks, FlexRay.

I. INTRODUCTION

Modern high-end vehicles incorporate 100 or more embedded computing units that implement advanced capabilities such as automated parking, pedestrian detection with automatic braking, and other safety or comfort features. These algorithms perform complex processing on data gathered from a network of sensors, to produce control sequences for distributed actuators. The communication bandwidth and quality of service required for such advanced electronic control units (ECUs) exceed the capabilities of the event-triggered controller area network (CAN) protocol, which has been pervasive in automotive systems until now. Moreover, next-generation in vehicle systems, specifically in electric vehicles that have a high level of automation, demand higher determinism, leading to the widespread adoption of time-triggered communication schemes and protocols such as FlexRay and time-triggered Ethernet.

FlexRay is gaining ground as a de facto communication standard for safety-critical functions such as drive-by-wire, cruise control, and adaptive braking systems, while also facilitating communication for noncritical ECUs. Although time-triggered networks such as FlexRay provide higher determinism and communication bandwidth, increasing proliferation of embedded computing units increases the associated communication overheads and power consumption, which can degrade overall system performance.

Typically, each ECU has a discrete communication controller (CC) to manage its access to the network. We show that, by closely coupling the controller with the ECU and extending the predefined communication framework, advanced and intelligent embedded compute units with enhanced capabilities such as fallback and fault tolerance can be designed. This scheme enhances the overall quality and performance of the system. However, such evolutions and extensions of the protocol cannot be implemented using off-the-shelf controllers or platform-agnostic solutions, and they require a modular flexible implementation, which is ideally implemented in reconfigurable logic.

Moreover, reconfigurable technology enables us to merge the controller and multiple applications on the same device, while preserving the necessary isolation between them, and partial reconfigurability can be exploited to reduce power consumption further.

A. Optimized FlexRay CC Architecture

In this system, we present an architecture-optimized FlexRay CC, which integrates configurable extensions that augment the CC's capabilities beyond those defined by the FlexRay standard. The controller provides enhancements to the data path, such as programmable width time stamping, data filtering, header insertion, and processing functions,

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 2, February 2016

which are abstracted away from the host function. Our flexible architecture can be used to design advanced ECUs on reconfigurable hardware, which consume less power and offer increased consolidation, while providing enhanced capabilities that are impossible to implement using standard controllers or IP cores.
We also quantify the potential of the proposed controller using case studies based on existing and evolving automotive applications that are safety critical and data intensive. Our experiments show that advanced features, such as high-speed mode switching for fault-tolerant ECUs, low-latency data handling for high-performance gateways, timeliness, and security for messages can be efficiently achieved by integrating such extensions within the controller data path, rather than offloading them to the processing logic.

B. Time-Triggered Network Standards

The move toward time-triggered network standards in automotive systems has been driven by the more advanced requirements imposed by advanced mission-critical and comfort features in future vehicles. Widespread event-triggered networks such as CANs fail to address the requirements of such applications. Time-triggered CAN (TT-CAN) is an extension of the CAN protocol that enables time-triggered operation by enforcing a slot-based structure, while retaining backward compatibility with the standard CAN.
However, TT-CAN suffers from dependability issues and limited bandwidth; thus, it did not gain widespread adoption. Some research sought to overcome these limitations through hardware extensions on the network controller. In recent years, FlexRay has emerged as the standard for time-triggered communication in the automotive domain. However, most recently, new hardware developments have seen time-triggered Ethernet emerge as a possible replacement for FlexRay, although standard communication protocols are still under development. The enhancements we present in this system can be similarly applied to other time-triggered standards, although we use FlexRay to demonstrate the concepts within a realistic certifiable environment.

C. The FlexRay Protocol

The FlexRay protocol is developed and standardized by the FlexRay consortium and has since been adopted by various automotive companies in production vehicles. These vehicles are complaint with the FlexRay AUTOSAR Interface Specification Standard, which is the industry standard for the software specification of FlexRay nodes, by which any controller implementation must comply. The fundamental element of the media access scheme in the FlexRay protocol is the communication cycle, which is repeated over time, as shown in Fig. 1. Each cycle is comprised of four segments.
The static segment uses a static slot-based access mechanism and is used to send critical data in a deterministic manner. Any ECU can send a frame of data in the one (or more) slot(s) assigned to it. The dynamic segment uses a dynamic slot-based access scheme enabling communication of event-triggered data of arbitrary length. The slot width is dynamic, depending on the amount of data that needs to be transmitted, and access to the medium is controlled by priorities assigned to the ECUs. The symbol window is used to transmit special symbols such as the "wake-up" pattern used to wake up sleeping nodes to initiate communication. Network idle time is the idle period used by nodes to make clock adjustments and align and correct the global view of time to maintain synchronization

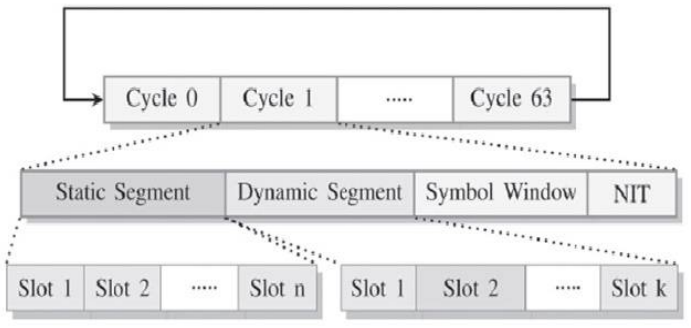


Fig.1 FlexRay Communication Cycle

A typical FlexRay-based ECU integrates a discrete (or embedded) CC and the computational function, which is usually implemented as a software algorithm on a processor to provide flexibility and upgradability. The ECU can

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 2, February 2016

communicate over the bus, through the CC, by transmitting framed data in the slot(s) assigned to it in the static or dynamic segments.

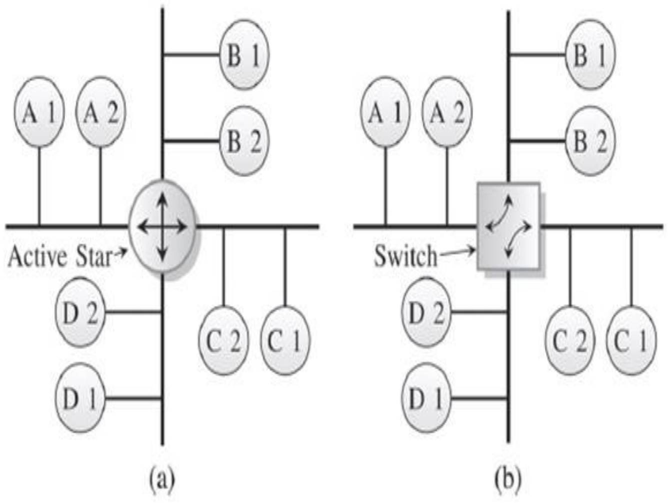


Fig.2 (a) Standard and (b) Switched FlexRay Network Topologies

Multiple nodes may share the same slot in different cycles, as in the case of odd/even cycle multiplexing where one set of nodes is assigned slots in all odd cycles, whereas another set of nodes (which may include some from the first set) is assigned slots in all even cycles. This scheme of cycle-level slot multiplexing can lead to higher overall bandwidth utilization. Fig. 2(a) shows a typical network setup, where node B2 may send data to node A1 in slot 1 of cycle 1, whereas node A1 may reply in slot 1 of cycle 2. The active star is an active repeater that passes information from one branch to all other branches.

The switched FlexRay network is a novel concept that can extend bandwidth without compromising reliability and determinism. The switch architecture allows exploitation of branch parallelism, whereby the switch will repeat frames only on branches that contain the intended recipient. This allows the same slots to be used simultaneously by multiple nodes in the same cycle, and the intelligent FlexRay switch schedules the branch to which information has to be relayed.

Thus, as shown in Fig. 2(b), while node B2 may be sending data to node A1 in slot 1 of cycle 1, node D1 might be sending data to node C2 in the same slot, and the switch, knowing the schedule, connects the corresponding nodes through the switch fabric. By utilizing slot multiplexing and branch parallelism, each slot within a cycle may have different destinations and, thus, different switch configurations. Research on FlexRay networks has been approached from diverse directions in the literature. In Forest highlights challenges such as physical-layer design, cycle and schedule design, and selection of termination, sync, and startup nodes, which were all simpler design considerations for FlexRay's predecessor, i.e., CAN.

II. Related work

In the existing methodologies high-end vehicles incorporate 100 or more embedded computing units that implement advanced capabilities such as automated parking, pedestrian detection with automatic braking, and other safety or comfort features. These existing techniques perform complex processing on data gathered from a network of sensors, to produce control sequences for distributed actuators. The communication bandwidth and quality of service required for such advanced electronic control units (ECUs) exceed the capabilities of the event-triggered controller area network (CAN) protocol. In which it has been pervasive in automotive systems until now. Moreover, next-generation in vehicle systems, specifically in electric vehicles that have a high level of automation, demand higher determinism, leading to the widespread adoption of time-triggered communication schemes and protocols such as FlexRay and time-triggered Ethernet.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 2, February 2016

The design of electric vehicles require a complete paradigm shift in terms of embedded systems architectures and software design techniques that are followed within the conventional automotive systems domain. It is increasingly being realized that the evolutionary approach of replacing the engine of a car by an electric engine will not be able to address issues like acceptable vehicle range, battery lifetime performance, battery management techniques, costs and weight, which are the core issues for the success of electric vehicles. While battery technology has crucial importance in the domain of electric vehicles, how these batteries are used and managed pose new problems in the area of embedded systems architecture and software for electric vehicles. At the same time, the communication and computation design challenges in electric vehicles also have to be addressed appropriately. This paper discusses some of these research challenges.
Modern vehicles incorporate a significant amount of computation, which has led to an increase in the number of computational nodes and the need for faster in-vehicle networks. Functions range from noncritical control of electric windows, through critical drive-by-wire systems, to entertainment applications; as more systems are automated, this variety and number will continue to increase. Accommodating the varying computational and communication requirements of such a diverse range of functions requires flexible networks and embedded computing devices. As the number of electronic control units (ECUs) increases, power and efficiency become more important, more so in next-generation electric vehicles. Moreover, predictability and isolation of safety-critical functions are nontrivial challenges when aggregating multiple functions onto fewer nodes. Reconfigurable computing can play a key role in addressing these challenges, providing both static and dynamic flexibility, with high computational capabilities, at lower power consumption. Reconfigurable hardware also provides resources and methods to address deterministic requirements, reliability and isolation of aggregated functions. This letter presents some initial research on the place of reconfigurable computing in future vehicles.
The Controller Area Network (CAN) protocol was originally developed for distributed automotive applications in the 1980s, and has previously proved to be extremely popular for the implementation of low-cost distributed systems. Whilst CAN has many features that make it suitable for such applications, it also has a number of well-discussed drawbacks; data transmission is limited to a fixed 8-byte payload at 1 Mbps, and the hardware bit-stuffing mechanism and automatic retransmission scheme can act to severely decrease the predictability of a CAN network. This paper will describe a modified CAN-like protocol controller that can be implemented on a programmable-logic device such as an FPGA. The modified protocol controller can help to ameliorate one of the drawbacks of CAN, whilst operating as closely as is possible to the original protocol specification. Specifically, this paper will discuss an implementation of a simple extension to the original protocol that allows for larger payloads, coupled with modifications to the bit-stuffing mechanism to increase predictability. The effectiveness of this modified CAN controller is illustrated in a series of experiments employing a novel test facility. The paper also discusses the potential drawbacks of the new controller, and is then concluded by suggesting possible areas of further research.
In this system, we present some of the main tasks that have to be addressed and solved for FlexRay to become a reality in series production at Audi. The results will be used for future generations of vehicles in the Volkswagen Group. The methods established and products developed build the base for Audi's FlexRay series development.
This system introduces the concept of switched FlexRay networks and proposes two algorithms to schedule data communication for this new type of network. Switched FlexRay networks use an intelligent star coupler, called a switch, to temporarily decouple network branches, thereby increasing the effective network bandwidth. Although scheduling for basic FlexRay networks is not new, prior work in this domain does not utilize the branch parallelism that is available when a FlexRay switch is used. In addition to the novel exploitation of branch parallelism, the scheduling algorithms proposed in this paper also support all slot multiplexing options as defined in the FlexRay v 3.0 protocol specifications. This includes support for the newly added repetition rates and support for multiplexing frames from different sending nodes in the same slot. Our first algorithm quickly produces a schedule given the communication requirements, network topology and FlexRay parameters, but cannot guarantee an optimal schedule in terms of the bandwidth efficiency and extensibility. Therefore, a second, branch-and-price algorithm is introduced that does find optimal schedules.

III. PROPOSED SCHEME

In the proposed approach the FlexRay protocol is developed and standardized by the FlexRay consortium and has since been adopted by various automotive companies in production vehicles. These vehicles are complaint with the

FlexRay AUTOSAR Interface Specification Standard, which is the industry standard for the software specification of FlexRay nodes, by which any controller implementation must comply.
The fundamental element of the media access scheme in the FlexRay protocol is the communication cycle, which is repeated over time.
Each cycle is comprised of four segments.

- Static Segment
- Dynamic Segment
- Symbol Window
- Network Idle Time

A. Controller Data Path Extensions

Traditional controllers depend on the host processor to read the received data and determine the usefulness of it. The controller issues a data interrupt, to which the processor responds with a status register read followed by a data read request, subsequently receiving the data. These overheads are wasted in the case of frames with irrelevant data (such as obsolete or untimely data) or multicycle data frames where the processor cannot process the received fragment until more data is available. In the case of critical data frames such as error state that require immediate attention, the latency introduced by the traditional scheme limits the performance of safety-critical systems, which rely on host-triggered recovery. With custom extensions, such exceptions can be handled at the controller, which processes the information and informs the host processor (using interrupts).
The host retains absolute control but is not involved in the low level processing, which is handled instead by the configurable extensions. Fig. 11 describes the functioning of such extensions on the receive path of the controller On the receive path, the extensions can monitor the received data for matching FlexRay message ID, application-based custom headers or timestamp information, which are contained in the data segment of the FlexRay frame. The FlexRay message ID can be used for application/user-defined communication in dynamic segment data frames.

B. Time Awareness for Messages

A major security risk in time-triggered systems such as FlexRay is the lack of time awareness for messages. By monitoring bus transactions, an external agent can easily employ simple replay attacks, flooding the bus with stale data. The FlexRay protocol leaves this vulnerability to the higher layer applications to manage. In our controller, the transmit path allows messages to be optionally time stamped to make the message time aware, at the cost of increased payload size. By inserting the header and timestamp within the data segment of the FlexRay frame, it is transparent to other FlexRay controllers present on the network, ensuring interoperability with off-the-shelf controllers. With timestamps enabled, the receive path can be configured to automatically drop frames that are outside an allowed time window. This creates a basic security layer at each ECU, which can be augmented further by incorporating encryption/decryption logic in the data path. An interesting use case is in high-performance gateways that move data between network clusters.

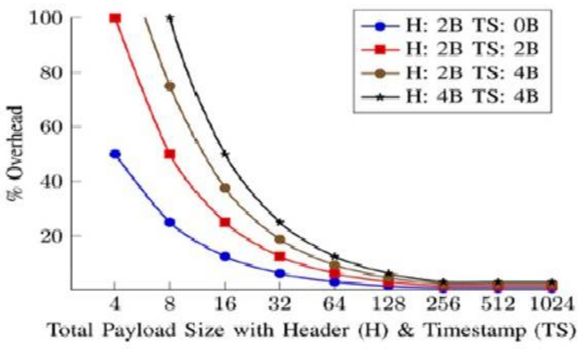


Fig.3 Data repacking for Multicycle Data Transfers

With traditional interfaces, messages arriving from each interface will be forwarded to the switch logic, which decides whether to forward the data to their destination or drop them because they have expired. By building intelligence into the controller, the validity of data can be determined before they are forwarded to the switching logic. We modify the experimental setup described in Fig.3 earlier to model a gateway configured to discard untimely data, either at the processing logic (MicroBlaze), mimicking off-the-shelf interfaces, or at the interface using our enhanced controller extensions. Our tests show that the interface can process the timestamp and discard the message within 180 ns of frame reception.

C. Handling Volume Data for Interfaces

Applications such as radar-based cruise control utilize volume data gathered by the radar sensors to compute distance and relative velocity of other vehicles in the vicinity. A complete data set from a sweep is required by the processing logic to determine these parameters, and these data are received over many data slots.

The processing ECU must reassemble these fragments before the data can be processed. By moving this packing/repacking to the controller interface, the processing logic can overlap the computation with data reception, enabling it to run at lower frequencies and, hence, consume less power. To demonstrate this, we use the experimental setup for the radar-based cruise control ECU. The data from the radar sensor are received over the FlexRay bus in bursts of 256 bytes, which is the maximum payload size defined by FlexRay standard. The MicroBlaze runs software routines on top of the Xilinx Standalone OS. In a normal design, the processor is interrupted each time a block of data is received. The processor responds with the first data read request 12 ms (worst-case) after receiving the interrupt, with the burst read consuming a further 3.84 ms. This is repeated over four cycles to complete the data transfer, cumulatively consuming 63.36 ms.

We then test the same application with an extension that allows the controller to intelligently buffer the entire frame in a buffer, only interrupting the processor at the end of the transaction. This enables the processor to issue back-to-back reads from the controller, completing the entire data movement in 27.36 ms from the reception of the interrupt. To provide a balance between multiscale and single-cycle data, the design has been constrained to handle up to four data cycles at full payload size. To support larger data sizes, larger buffer memory must be added to the controller, resulting in higher device utilization; however, this may be a tolerable cost for some ECUs, and the CC architecture supports it.

IV. BLOCK DIAGRAM

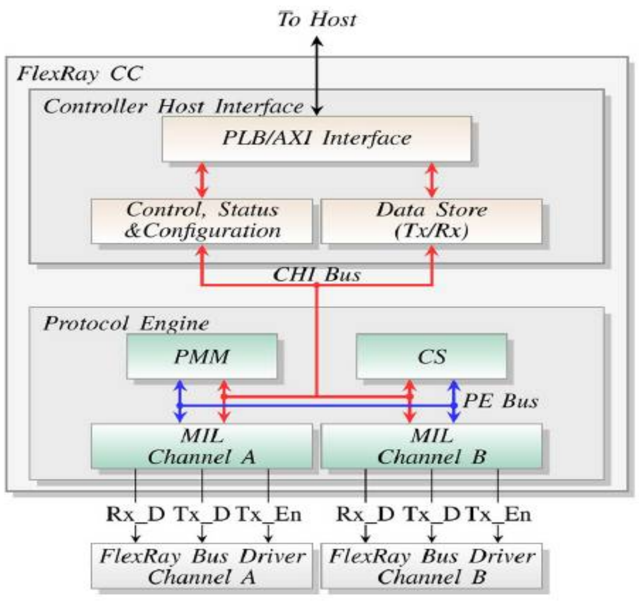


Fig.4 Block Diagram

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 2, February 2016

V. Performance Analysis

A. FlexRay Implementation

The main objective of the integrating CAN with FlexRay is to provide the data bit oriented transferring more efficient manner and with good speed. Advanced applications can benefit from tight coupling of the embedded computing units with the communication interface, thereby providing functionality beyond the FlexRay standard. Such an approach is highly suited to implementation on reconfigurable architectures. The CAN Protocol can be used to provide the high speed communication between two or more devices.

B. FlexRay Processing

FlexRay is gaining ground as a de facto communication standard for safety-critical functions such as drive-by-wire, cruise control, and adaptive braking systems, while also facilitating communication for noncritical ECUs. Although time-triggered networks such as FlexRay provide higher determinism and communication bandwidth, increasing proliferation of embedded computing units increases the associated communication overheads and power consumption, which can degrade overall system performance. Typically, each ECU has a discrete communication controller (CC) to manage its access to the network. We show that, by closely coupling the controller with the ECU and extending the predefined communication framework, advanced and intelligent embedded compute units with enhanced capabilities such as fallback and fault tolerance can be designed. This scheme enhances the overall quality and performance of the system. However, such evolutions and extensions of the protocol cannot be implemented using off-the-shelf controllers or platform-agnostic solutions, and they require a modular flexible implementation, which is ideally implemented in reconfigurable logic

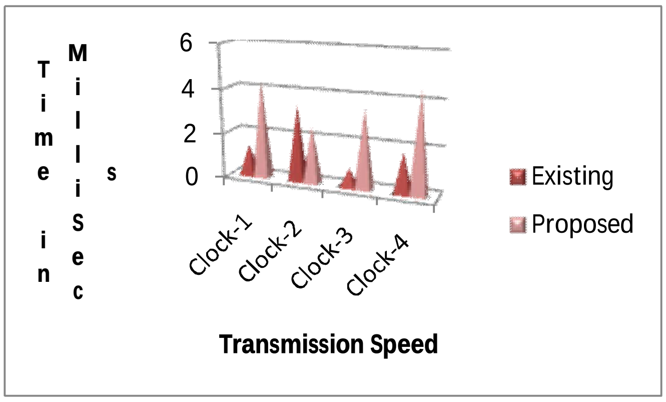


Fig.5 Transmission Speed

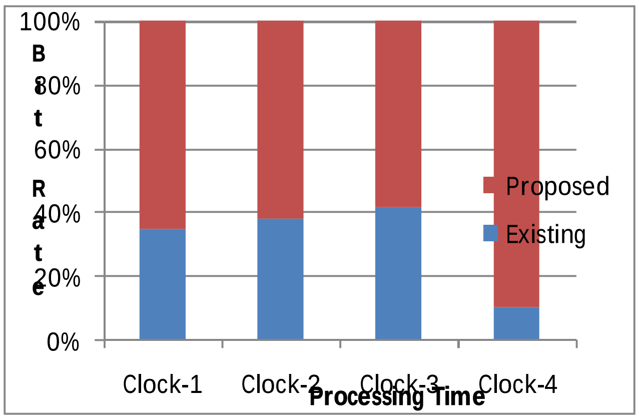


Fig.6 Processing Time

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

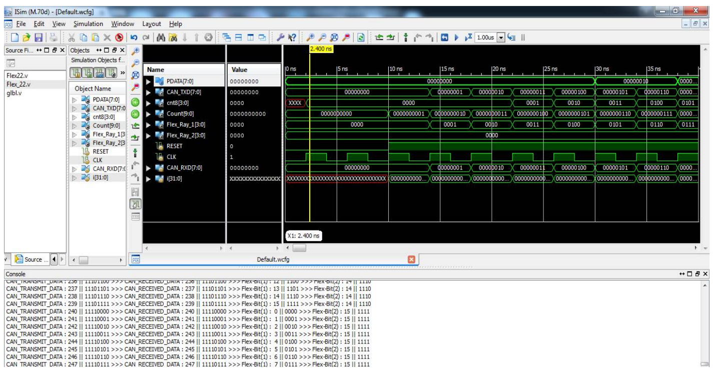
(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 2, February 2016

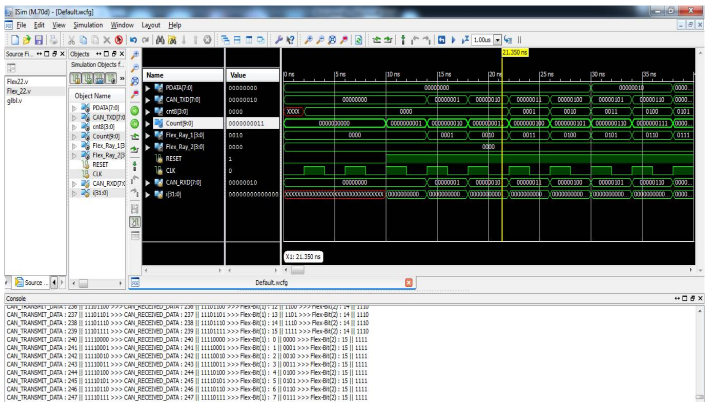
C. Performance Evaluation

The performance of the proposed scheme is defined by means of FlexRay operations as well as successful data bit processing with existing CAN implementation scheme. Extensive Processing and performance analysis shows that the proposed schemes are provably speed and highly efficient while data transferring. We believe all these advantages of the proposed schemes will FlexRay and CAN protocol operations on economies. To understand the practicality of the integration of this kind of approaches, we analyze its strength, evaluate its running time overhead via tested experiments, and conduct monetary cost analysis.

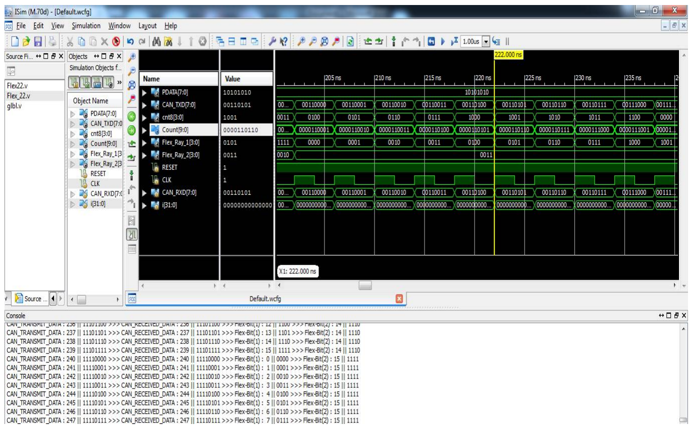
VI. SAMPLE SCREENSHOTS



```
CAN_TRANSMIT_DATA : 236 || 11101100 >>> CAN_RECEIVED_DATA : 236 || 11101100 >>> Flex-Bit(1) : 12 || 1100 >>> Flex-Bit(2) : 14 || 1110
CAN_TRANSMIT_DATA : 237 || 11101101 >>> CAN_RECEIVED_DATA : 237 || 11101101 >>> Flex-Bit(1) : 13 || 1101 >>> Flex-Bit(2) : 14 || 1110
CAN_TRANSMIT_DATA : 238 || 11101110 >>> CAN_RECEIVED_DATA : 238 || 11101110 >>> Flex-Bit(1) : 14 || 1110 >>> Flex-Bit(2) : 14 || 1110
CAN_TRANSMIT_DATA : 239 || 11101111 >>> CAN_RECEIVED_DATA : 239 || 11101111 >>> Flex-Bit(1) : 15 || 1111 >>> Flex-Bit(2) : 14 || 1110
CAN_TRANSMIT_DATA : 240 || 11110000 >>> CAN_RECEIVED_DATA : 240 || 11110000 >>> Flex-Bit(1) :  0 || 0000 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 241 || 11110001 >>> CAN_RECEIVED_DATA : 241 || 11110001 >>> Flex-Bit(1) :  1 || 0001 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 242 || 11110010 >>> CAN_RECEIVED_DATA : 242 || 11110010 >>> Flex-Bit(1) :  2 || 0010 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 243 || 11110011 >>> CAN_RECEIVED_DATA : 243 || 11110011 >>> Flex-Bit(1) :  3 || 0011 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 244 || 11110100 >>> CAN_RECEIVED_DATA : 244 || 11110100 >>> Flex-Bit(1) :  4 || 0100 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 245 || 11110101 >>> CAN_RECEIVED_DATA : 245 || 11110101 >>> Flex-Bit(1) :  5 || 0101 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 246 || 11110110 >>> CAN_RECEIVED_DATA : 246 || 11110110 >>> Flex-Bit(1) :  6 || 0110 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 247 || 11110111 >>> CAN_RECEIVED_DATA : 247 || 11110111 >>> Flex-Bit(1) :  7 || 0111 >>> Flex-Bit(2) : 15 || 1111
```



```
CAN_TRANSMIT_DATA : 236 || 11101100 >>> CAN_RECEIVED_DATA : 236 || 11101100 >>> Flex-Bit(1) : 12 || 1100 >>> Flex-Bit(2) : 14 || 1110
CAN_TRANSMIT_DATA : 237 || 11101101 >>> CAN_RECEIVED_DATA : 237 || 11101101 >>> Flex-Bit(1) : 13 || 1101 >>> Flex-Bit(2) : 14 || 1110
CAN_TRANSMIT_DATA : 238 || 11101110 >>> CAN_RECEIVED_DATA : 238 || 11101110 >>> Flex-Bit(1) : 14 || 1110 >>> Flex-Bit(2) : 14 || 1110
CAN_TRANSMIT_DATA : 239 || 11101111 >>> CAN_RECEIVED_DATA : 239 || 11101111 >>> Flex-Bit(1) : 15 || 1111 >>> Flex-Bit(2) : 14 || 1110
CAN_TRANSMIT_DATA : 240 || 11110000 >>> CAN_RECEIVED_DATA : 240 || 11110000 >>> Flex-Bit(1) :  0 || 0000 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 241 || 11110001 >>> CAN_RECEIVED_DATA : 241 || 11110001 >>> Flex-Bit(1) :  1 || 0001 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 242 || 11110010 >>> CAN_RECEIVED_DATA : 242 || 11110010 >>> Flex-Bit(1) :  2 || 0010 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 243 || 11110011 >>> CAN_RECEIVED_DATA : 243 || 11110011 >>> Flex-Bit(1) :  3 || 0011 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 244 || 11110100 >>> CAN_RECEIVED_DATA : 244 || 11110100 >>> Flex-Bit(1) :  4 || 0100 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 245 || 11110101 >>> CAN_RECEIVED_DATA : 245 || 11110101 >>> Flex-Bit(1) :  5 || 0101 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 246 || 11110110 >>> CAN_RECEIVED_DATA : 246 || 11110110 >>> Flex-Bit(1) :  6 || 0110 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 247 || 11110111 >>> CAN_RECEIVED_DATA : 247 || 11110111 >>> Flex-Bit(1) :  7 || 0111 >>> Flex-Bit(2) : 15 || 1111
```



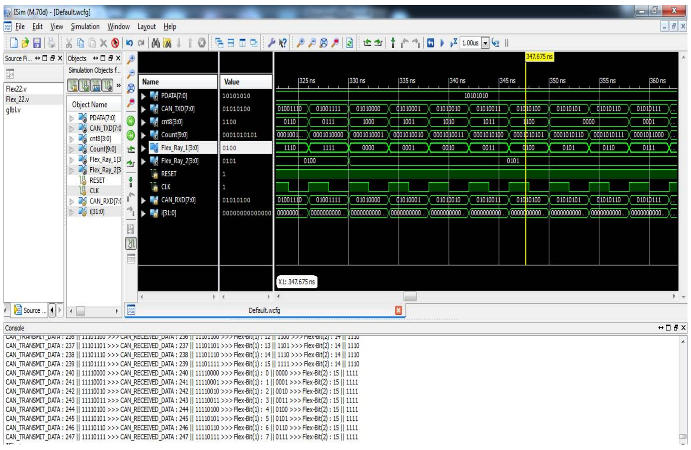
```
CAN_TRANSMIT_DATA : 236 || 11101100 >>> CAN_RECEIVED_DATA : 236 || 11101100 >>> Flex-Bit(1) : 12 || 1100 >>> Flex-Bit(2) : 14 || 1110
CAN_TRANSMIT_DATA : 237 || 11101101 >>> CAN_RECEIVED_DATA : 237 || 11101101 >>> Flex-Bit(1) : 13 || 1101 >>> Flex-Bit(2) : 14 || 1110
CAN_TRANSMIT_DATA : 238 || 11101110 >>> CAN_RECEIVED_DATA : 238 || 11101110 >>> Flex-Bit(1) : 14 || 1110 >>> Flex-Bit(2) : 14 || 1110
CAN_TRANSMIT_DATA : 239 || 11101111 >>> CAN_RECEIVED_DATA : 239 || 11101111 >>> Flex-Bit(1) : 15 || 1111 >>> Flex-Bit(2) : 14 || 1110
CAN_TRANSMIT_DATA : 240 || 11110000 >>> CAN_RECEIVED_DATA : 240 || 11110000 >>> Flex-Bit(1) :  0 || 0000 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 241 || 11110001 >>> CAN_RECEIVED_DATA : 241 || 11110001 >>> Flex-Bit(1) :  1 || 0001 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 242 || 11110010 >>> CAN_RECEIVED_DATA : 242 || 11110010 >>> Flex-Bit(1) :  2 || 0010 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 243 || 11110011 >>> CAN_RECEIVED_DATA : 243 || 11110011 >>> Flex-Bit(1) :  3 || 0011 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 244 || 11110100 >>> CAN_RECEIVED_DATA : 244 || 11110100 >>> Flex-Bit(1) :  4 || 0100 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 245 || 11110101 >>> CAN_RECEIVED_DATA : 245 || 11110101 >>> Flex-Bit(1) :  5 || 0101 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 246 || 11110110 >>> CAN_RECEIVED_DATA : 246 || 11110110 >>> Flex-Bit(1) :  6 || 0110 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 247 || 11110111 >>> CAN_RECEIVED_DATA : 247 || 11110111 >>> Flex-Bit(1) :  7 || 0111 >>> Flex-Bit(2) : 15 || 1111
```

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 2, February 2016



```
CAN_TRANSMIT_DATA : 236 || 11101100 >>> CAN_RECEIVED_DATA : 236 || 11101100 >>> Flex-Bit(1) : 12 || 1100 >>> Flex-Bit(2) : 14 || 1110
CAN_TRANSMIT_DATA : 237 || 11101101 >>> CAN_RECEIVED_DATA : 237 || 11101101 >>> Flex-Bit(1) : 13 || 1101 >>> Flex-Bit(2) : 14 || 1110
CAN_TRANSMIT_DATA : 238 || 11101110 >>> CAN_RECEIVED_DATA : 238 || 11101110 >>> Flex-Bit(1) : 14 || 1110 >>> Flex-Bit(2) : 14 || 1110
CAN_TRANSMIT_DATA : 239 || 11101111 >>> CAN_RECEIVED_DATA : 239 || 11101111 >>> Flex-Bit(1) : 15 || 1111 >>> Flex-Bit(2) : 14 || 1110
CAN_TRANSMIT_DATA : 240 || 11110000 >>> CAN_RECEIVED_DATA : 240 || 11110000 >>> Flex-Bit(1) :  0 || 0000 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 241 || 11110001 >>> CAN_RECEIVED_DATA : 241 || 11110001 >>> Flex-Bit(1) :  1 || 0001 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 242 || 11110010 >>> CAN_RECEIVED_DATA : 242 || 11110010 >>> Flex-Bit(1) :  2 || 0010 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 243 || 11110011 >>> CAN_RECEIVED_DATA : 243 || 11110011 >>> Flex-Bit(1) :  3 || 0011 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 244 || 11110100 >>> CAN_RECEIVED_DATA : 244 || 11110100 >>> Flex-Bit(1) :  4 || 0100 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 245 || 11110101 >>> CAN_RECEIVED_DATA : 245 || 11110101 >>> Flex-Bit(1) :  5 || 0101 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 246 || 11110110 >>> CAN_RECEIVED_DATA : 246 || 11110110 >>> Flex-Bit(1) :  6 || 0110 >>> Flex-Bit(2) : 15 || 1111
CAN_TRANSMIT_DATA : 247 || 11110111 >>> CAN_RECEIVED_DATA : 247 || 11110111 >>> Flex-Bit(1) :  7 || 0111 >>> Flex-Bit(2) : 15 || 1111
```

VII. CONCLUSION

We have given an overview of the FlexRay protocol and the generic architecture of the CC, as defined by the specification. By identifying and extracting operations that are mutually exclusive or natively parallel, we have designed a custom controller that takes advantage of the heterogeneous resources on modern FPGAs, resulting in reduced logic footprint and low power consumption, while providing a host of features beyond those described by the standard. Advanced computational capabilities such as fault tolerance and function consolidation can be built into nodes that integrate complex ECU functions with advanced CCs. This approach also improves power consumption compared with the use of discrete controllers. We hope that our flexible and configurable architecture can be leveraged for continued research on intelligent FlexRay nodes and switches on FPGAs, leading to wider adoption of reconfigurable hardware for in-vehicle applications. We aim to investigate extending this controller for use with partial reconfiguration to provide flexible use of the FPGA fabric, enabling further sharing of communication resources between ECUs. We intend to develop intelligent FlexRay nodes and switches on reconfigurable hardware that are energy efficient and that will allow us to explore more advanced network setups. Finally, the principles demonstrated in this approach are also applicable to other time-triggered interfaces, and we hope to explore this for time-triggered Ethernet.

REFERENCES

1. S. Chakraborty et al., “Embedded systems and software challenges in electric vehicles,” in Proc. Des., Autom. Test Eur. (DATE) Conf., Mar. 2012, pp. 424–429.
2. S. Shreejith, S. A. Fahmy, and M. Lukasiewycz, “Reconfigurable computing in next-generation automotive networks,” IEEE Embedded Syst. Lett., vol. 5, no. 1, pp. 12–15, Mar. 2013.
3. I. Sheikh, M. Hanif, and M. Short, “Improving information throughput and transmission predictability in Controller Area Networks,” in Proc. IEEE Int. Symp. Ind. Electron. (ISIE), Jul. 2010, pp. 1736–1741.
4. J. Kötz and S. Poledna, “Making FlexRay a Reality in a Premium Car,” in Proc. Convergence Transportation Electronics Association, SAE Int., 2008, pp. 391–395.
5. Specification of FlexRay Interface Version 3.2.0, AUTOSAR Std. [Online]. Available: http://www.autosar.org
6. FlexRay Communications System, Protocol Specification Version 2.1 Revision A, FlexRay Consortium Std., Dec. 2005. [Online]. Available: http://www.flexray.com
7. P. Milbredt, B. Vermeulen, G. Tabanoglu, and. Lukasiewycz, “Switched FlexRay: Increasing the effective bandwidth and safety of FlexRay networks,” in Proc. Conf. Emerging Technol. Factory Autom. (ETFA), Sep. 2010, pp. 1–8.
8. T. Schenkelaars, B. Vermeulen, and K. Goossens, “Optimal Scheduling of Switched FlexRay Networks,” in Proc. Des., Autom. Test Eur. (DATE) Conf., Mar. 2011, pp. 1–6.