An Approach for Redundancy in FlexRay Networks Using FPGA Partial Reconfiguration

Shanker Shreejith[*] [†], Kizhepatt Vipin[*], Suhaib A. Fahmy[*] and Martin Lukasiewycz[†]
[*]School of Computer Engineering
Nanyang Technological University, Singapore
Email: {shreejit1,vipin2,sfahmy}@ntu.edu.sg, martin.lukasiewycz@tum-create.edu.sg
[†]TUM CREATE Centre for Electromobility, Singapore

Abstract—**Safety-critical in-vehicle electronic control units (ECUs) demand high levels of determinism and isolation, since they directly influence vehicle behaviour and passenger safety. As modern vehicles incorporate more complex computational systems, ensuring the safety of critical systems becomes paramount. One-to-one redundant units have been previously proposed as measures for evolving critical functions like x-by-wire. However, these may not be viable solutions for power-constrained systems like next generation electric vehicles. Reconfigurable architectures offer alternative approaches to implementing reliable safety critical systems using more efficient hardware. In this paper, we present an approach for implementing redundancy in safety-critical in-car systems, that uses FPGA partial reconfiguration and a customised bus controller to offer fast recovery from faults. Results show that such an integrated design is better than alternatives that use discrete bus interface modules.**

I. Introduction

Emerging automotive functions like x-by-wire, adaptive driver assistance and occupant safety systems employ a distributed computing paradigm, which requires in-vehicle networks to have stringent and time-bound behaviour as well as high communication capacity. Many of these functions are classified as safety-critical since they have a direct impact on vehicle dynamics and occupant safety. Such functions are are characterised by hard deterministic requirements, since predictability is of paramount importance. Because of their nature, most safety-critical systems have some sort of built-in redundancy either in the shape of a hot stand-by, an isolated backup, or a degraded mode of operation.

Event-triggered communication architectures like the Controller Area Network (CAN) are unable to consistently provide high levels of determinism, as the communication demand increases [1]. Automotive systems are moving towards time-triggered network architectures, such as the FlexRay protocol that can provide high levels of predictability, even when the network is operating at full capacity [2]. Protocols like FlexRay provide multiple schemes of medium access to support safety critical or deterministic systems, as well as non-safety critical systems.

To counter the challenges of increased weight and power incurred by the increasing addition of newer functions, and the increased communication demands of such networks, it is proposed that multiple functions be aggregated onto fewer ECUs. Traditional approaches, using general purpose processors as the computational engine, require real-time operating systems with time-triggered capability to meet the requirements of safety critical systems, when aggregating functions. Moreover, providing sufficient isolation between aggregated functions, would be challenging since processor hardware resources would be shared by the two (or more) functions.

Reconfigurable hardware represents a promising platform for implementing such systems; it is possible to create optimised nodes that integrate ECUs and network controllers on a single piece of hardware, resulting in less power consumption, and weight [3]. Moreover, FPGAs also allow us to instantiate multiple instances on the same hardware function, with little or no interference, proving to be ideal for integrating safety critical or non-safety critical systems.

In this work, we propose a scheme for implementing safety critical ECU systems on reconfigurable hardware, within a time-triggered network like FlexRay. This approach can be adapted to implement an isolated fault-tolerant node or a distributed back-up node which can take over the functionality of multiple failed nodes. The scheme exploits dynamic partial reconfiguration of FPGAs and a custom network controller to provide short turn-around time as required by most safety-critical systems. We present a proof-of-concept implementation on a Xilinx FPGA that integrates a custom FlexRay controller and ECU functionality.

II. Related work

FPGAs are widely used to implement custom architectures to accelerate a range of algorithms. Advanced techniques like dynamic partial reconfiguration (PR), promote re-use of FPGAs hardware resources by allowing non-concurrent functions to use the same hardware. PR is a technique available in recent FPGAs which makes it possible to modify only portions of a circuit, while the remainder of the system continues to function without interruption. This is achieved by selectively altering portions of the FPGA configuration memory, the contents of which determine the function implemented in the FPGA.

An architecture for implementing fail-safe safety critical ECU systems on FPGAs, leveraging dynamic reconfiguration (complete reconfiguration), is described in [4]. Their architecture uses FPGA logic as a fail-safe back-up, which is completely reconfigured with one of the back-up modes when

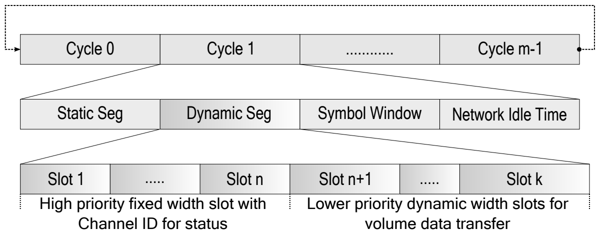

Fig. 1. FlexRay Cycle

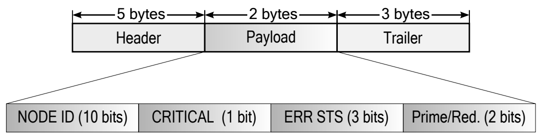

Fig. 2. Dynamic Segment Payload with Message ID

errors are detected. PR has been used in non-safety critical automotive applications such as driver assistance systems [5]. In such cases, using PR can allow a reduction in the size of the required FPGA by time-multiplexing different functionality. In [6], the authors show how PR can be used to dynamically reconfigure a faulty ECU communication controller.

Our work aims to develop a scalable framework for using partial reconfiguration within time-triggered communication networks. We show that a custom network controller combined with an ECU on a single FPGA allows extensions to the standard definitions, allowing for further robustness to failures. We propose to combine the determinism provided by time-triggered scheme with the flexibility of reconfigurable hardware, to develop a system architecture for next generation safety critical systems that are scalable, while offering higher performance and efficiency.

III. Proposed Architecture

A. FlexRay Communication Cycle

The FlexRay communication cycle is the fundamental element of the FlexRay protocol, which is organised as m cycles (up to 64) which repeat over time [2]. Of the four configurable segments, as shown in Fig. 1, our focus is on the flexible *dynamic segment*, primarily used for event-triggered communication on a priority basis. Current systems like adaptive cruise control sparingly use the dynamic segment, meaning they can be used for status communication, without disturbing existing systems. We propose to use the first n slots of the dynamic segment to communicate fixed length system status messages for safety-critical nodes, which will trigger recovery in the case of errors.

We propose to utilise the two-byte *message ID* to indicate the critical status of the device, and whether its functions need to be taken over by a redundant unit, and the dynamic slot filtering scheme, as defined in the FlexRay protocol [2]. The frame structure is shown in Fig. 2. *Node ID* defines the unique ID assigned to the different safety-critical nodes within the network. The critical error status flag describes a critical error condition, and demands immediate attention. The error status flag indicate the consecutive number of non-critical errors encountered by the node, which are tolerable up to a predefined threshold. Within our architecture, each system may have localised fault detection logic or depend on distributed fault detection modules or on a combination of the two. The prime/redundant flag is used to distinguish the prime unit and the redundant unit, which use the same ID.

In each cycle, the node transmits (internal fault-detection) or receives (centralised fault-detection) fault status. Critical errors and high error rates are indicated using the embedded bits. Since only a fixed amount of payload data (2 bytes) is used in the highest priority slots, predictable latency is ensured by this scheme. The lower priority dynamic slots may still be used by other nodes for volume data transfer.

B. Safety-critical nodes on FPGAs

Almost all safety critical systems have a fall-back mode of operation to accommodate unexpected failures. Common strategies are to support either a fail-safe mode or a fault-tolerant implementation. A fault-tolerant implementation is a robust system, which can recover from faulty situations without severe degradation in system performance. Fault-tolerance is ensured by building redundancy for key operations and intelligence to switch to redundant logic, while the primary unit recovers from the erroneous condition [7].

FPGA partial reconfiguration (PR) offers alternative ways of implementing such redundancy. Areas of the FPGA fabric that are designated as partially reconfigurable regions (PRR) at design time can be selectively reconfigured during run-time, without altering the function of static parts of the design. Designating the primary functionality of the node in a PRR and a redundant (or degraded) mode of operation in the static region, fault-tolerance can be efficiently built into a system designed on reconfigurable hardware. If the primary function encounters an error and needs to be reconfigured, the degraded mode can be enabled while the primary function is reconfigured. FPGA based designs are better equipped to handle aggregation of different functions and thus can be designed to provide complete isolation between the two functions.

C. Proposed Approach to Reconfiguration

Fig. 3 shows an example ECU system on reconfigurable hardware. The *primary function* is the computational implementation of some algorithm like adaptive cruise control, which uses custom hardware accelerators for compute-intensive calculations. It communicates with other ECUs and sensors over the FlexRay bus through the FlexRay communication controller (CC) and the bus driver (BD), which is efficiently implemented as a custom module. The functional unit, comprising the primary function, acceleration and associated memory, are completely implemented in a partially reconfigurable region (PRR-1), so that the functionality can be switched or reconfigured in case of an error. The communication interface comprising the FlexRay CC, BD and the PR controller are implemented in the static region, SR-1. The second static region (SR-2) is designated for implementing redundant logic.

Under normal operation, the node is healthy and SR-2 is configured with the redundant logic, but is clock-gated to

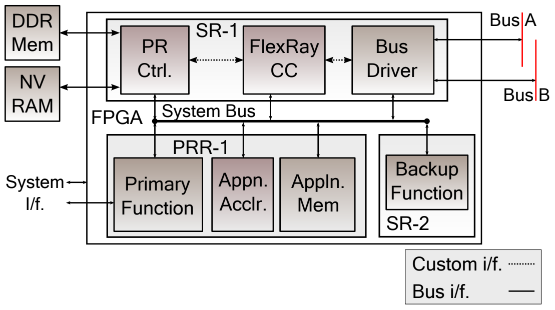

Fig. 3. Fail-safe ECU on FPGA

conserve power. If the node features fault-detection, a local fault initiates redundant mode and the status is indicated to other nodes over the FlexRay bus. Otherwise, a FlexRay frame in the assigned slot with the critical error flag set in the status data would trigger the switch to redundant mode. Disabling clock-gating takes only one clock cycle, resulting in fast turnaround to the redundant mode. Subsequently the partial reconfiguration controller is triggered to reconfigure PRR-1, to enable recovery from the error. PRR-1 is isolated from the system bus by the PR controller, and hence the reconfiguration proceeds without disturbing the functionality of the node. While reconfiguration is in-progress, the CC continues to receive data and the redundant function may use these to provide the required functionality. Once the PRR-1 region is reconfigured and enabled, the node may *optionally* perform a complete reset, by reseting the FlexRay CC and re-integrating to the network. SR-2 is then clock-gated to conserve power.

Implementing this mechanism using a discrete FlexRay controller would not be feasible, since these errors would have to be processed in the ECU processor, adding significant latency. Rather, we use a customised FlexRay controller that incorporate special extensions to enable this functionality. Filters can be programmed to filter dynamic segment messages with a *message ID* matching the ECU's ID (in case of remote fault detection) or to transmit messages with the *message ID* and status, in the priority slot assigned to it (in case of local fault detection). Whenever an error state is received or transmitted, the trigger is passed to the partial reconfiguration controller by these filters. Only by using such a customised controller implementation is it possible for the reconfiguration commands to be acted on immediately by the PR engine. In a normal design, using a fixed standard controller, the FlexRay messages, or local error state would need to be processed by the processor, resulting in noticeable latency before the redundant logic is activated. In our approach, the turn-around time from the point of receiving a reconfiguration command to switching to the back-up mode can be made negligibly small.

Reconfiguration is performed using a special built-in hardware macro called the internal configuration access port (ICAP). Xilinx supports PR in a processor based environment through a vendor-provided controller such as the OPBHW-ICAP, connected as a slave device to the processor bus, but this approach gives low throughput in the region of 4.66-10.1 MBytes/sec [8], [9], resulting in a large reconfiguration latency

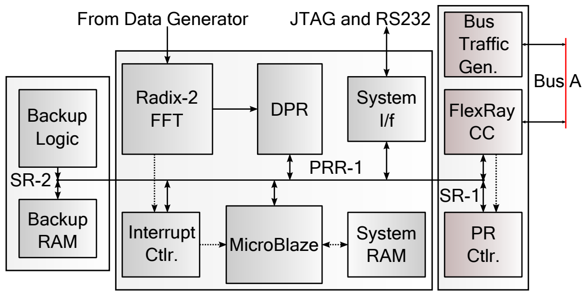

Fig. 4. Prototype for fail-safe Radar Signal Processing node

TABLE I
RESOURCE UTILISATION

	PRR-1	SR-2	SR-1		
			ICAP Cntrlr	FlexRay I/f	Test Logics
Registers	3632	200	672	4607	268
LUTs	3473	138	586	7021	577
BRAMs	24	2	8	8	6
DSPs	6	-	-	3	0
Total Power	1.8 Watts				

of the order of several milliseconds, depending upon the PRR size. However, the ICAP hard macro itself supports much higher speeds such as 400 MBytes/sec. We have developed a custom ICAP controller, that reaches the ICAP's maximum speed [10], using DMAs from external memory, thus enabling faster reconfiguration.

IV. Concept Validation and Results

To validate the architecture and investigate turnaround times, we have implemented a simple radar signal processing node that forms the front-end for driver assistance or adaptive cruise control systems [11]. The system was developed using Xilinx EDK 13.3 and hardware validated on a Xilinx ML605 development board hosting a Virtex-6 FPGA (XC6VLX240T). The application is based on a frequency modulated continuous wave (FMCW) technique with a triangular modulating wave, which can simultaneously determine distance and range-rate of the preceding vehicle. We use a 1 millisecond triangular modulator with a radar cycle of 32 milliseconds. The *primary function* from our node architecture is a MicroBlaze based system, as in Fig. 4, which executes software routines to estimate parameters from the frequency domain data that is generated by the radix-2 FFT hardware module. The estimates are then passed over the FlexRay bus to the central node, which also performs fault-detection and transmits the status using our proposed scheme. The primary ECU functionality is contained within PRR-1, while the redundant unit is contained within static region, SR-2. The FlexRay controller, PR controller and the test logic are in SR-1. The logic utilisation of the design is shown in Table I. PRR-1 has a partial bitstream size of 262 KBytes.

The radar data is received once every 32 milliseconds, on static slot 7 in cycles 32 and 64. The node is configured to monitor slot 24 (first dynamic slot) which relays node status using our proposed scheme. The performance of the framework in the different test cases is measured in terms

TABLE II
TURNAROUND AND RECOVERY TIME

	Turnaround Time		Recovery Time	
Intr. Cntlr./ ICAP Cntlr.	MicroBlaze Intr. Cntlr	Custom Logic	Xilinx ICAP	Custom ICAP
CC Latency	20 ns			
Interrupt Latency	12 to 420 us	0 ns	Custom Logic (0 ns)	
Turnaround/ Recovery	12 to 420 us	30 ns	26.240 ms	0.656 ms

of turnaround time and node recovery time. Turnaround time defines the time to switch to redundant logic from reception of error, while recovery time is the time taken to recover from error and resume normal operation. The turnaround times and recovery times are shown in Table II.

In our first experiment, the data interrupt is processed by the MicroBlaze processor, which checks for the critical error flag or a consecutive error count greater than the threshold and issues a reconfiguration command to the ICAP controller. However, the MicroBlaze also processes FFT data using a high priority interrupt resulting in a worst case interrupt latency of 41038 clock cycles, depending on when the error status flag was received. Thus the turnaround time can vary from 12us in the best case to 422us in the worst case. Alternatively, when the reconfiguration interrupt is instead processed within the controller through the custom extension, the generated interrupt is routed directly to the ICAP controller, resulting in a reduced latency of 20ns. This approach offers deterministic results, achieves a short turnaround of 30ns (400$\times$ over the best case) and is independent of the state of the logic in PRR-1.

To measure the overall recovery time, tests were undertaken with different PR controllers. In the first experiment, we have used the processor based XPS hardware ICAP controller available from Xilinx. The design takes 26,240us to reconfigure the functional block in PRR-1. The node re-integration consumes a further 6 FlexRay cycles, taking the total recovery time to 32.25ms. With our custom PR controller, PRR-1 reconfiguration is completed in just 656us, providing a 40$\times$ improvement over the Xilinx controller. Considering the re-integration time, the node will complete recovery and reintegration to the network in 6.66 ms.

The results show that the custom FlexRay controller allows us to achieve short and predictable turnaround, compared to a traditional interrupt-based processing technique associated with discrete controllers and processor based designs, which may incur significant and non-deterministic latency. This is because we can process the messages within the controller hardware, rather then require a long roundrip to a processor. Using a custom ICAP controller for PR, the recovery time of a safety-critical ECU on reconfigurable hardware can also be reduced to support higher levels of fault-tolerance. The use of clock gating results in a power efficient architecture, when compared with other alternatives where redundant logic must be active constantly. The scheme can also support failures on the fabric like a LUT failure, by using multiple bitstreams of the same logic which targets a different PR region or has different resource utilisation. The low device utilisation permits aggregation of another function on to the same hardware, which can run in complete isolation. Hence, a single device can be designated as a redundant unit for multiple nodes and the framework can be extended to support multiple redundant modules on the same device, saving power and space.

V. CONCLUSION

Consolidation and time-multiplexing of ECU functions is an important trend in modern vehicles. FPGAs offer exciting possibilities due to their customisation. Dynamic partial reconfiguration can be used to both multiplex functions and offer redundancy. We have presented a scalable scheme for implementing safety-critical systems utilising custom bus controllers and partial reconfiguration on FPGAs. It offers function consolidation and isolated operation while providing higher performance and energy efficiency. Our results show that customised extensions to an on-chip FlexRay controller and a customised PR controller can offer significant performance advantages that make FPGAs ideal for such implementations. We aim to look into extensions of this scheme, along with some formal analysis of the robustness of such methods.

ACKNOWLEDGMENT

This work was supported by the Singapore National Research Foundation under its Campus for Research Excellence And Technological Enterprise (CREATE) programme.

REFERENCES

[1] *CAN Specification, Version 2.0*, R. Bosch GmBh, Std., 1991.
[2] *FlexRay Communications System, Protocol Specification Version 2.1 Revision A*, FlexRay Consortium Std., December 2005. [Online]. Available: http://www.flexray.com
[3] S. Chakraborty, M. Lukasiewycz, C. Buckl, S. Fahmy, N. Chang, S. Park, Y.Kim, P. Leteinturier, and H. Adlkofer, "Embedded Systems and Software Challenges in Electric Vehicles," in *Proceedings of the Design Automation and Test in Europe (DATE)*, 2012.
[4] N. Chujo, "Fail-safe ECU System Using Dynamic Reconfiguration of FPGA," *R & D Review of Toyota CRDL*, vol. 37 no. 2, pp. 54–60, April 2002.
[5] C. Claus, J. Zeppenfeld, F. Muller, and W. Stechele, "Using partial-run-time reconfigurable hardware to accelerate video processing in driver assistance system," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2007.
[6] H.-M. Pham, S. Pillement, and D. Demigny, "Reconfigurable ECU communications in Autosar Environment," in *Proc. International Conference on Intelligent Transport Systems Telecommunications (ITST)*, 2009.
[7] K. Klobedanz, A. Koenig, W. Mueller, and A. Rettberg, "Self-Reconfiguration for Fault-Tolerant FlexRay Networks," in *International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW)*, 2011.
[8] C. Claus, F. H. Muller, J. Zeppenfeld, and W. Stechele, "A new framework to accelerate Virtex-II Pro dynamic partial selfreconfiguration." in *Proceedings of IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, 2007.
[9] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time partial reconfiguration speed investigation and architectural design space exploration," in *Proceedings of International Conference on Field Programmable Logic and Applications (FPL)*, 2009.
[10] K. Vipin and S. Fahmy, "A High Speed Open Source Controller for FPGA Partial Reconfiguration," in *Proceedings of the International Conference on Field Programmable Technology (FPT)*, 2012.
[11] J. Saad, A. Baghdadi, and F. Bodereau, "FPGA-based radar signal processing for automotive driver assistance system," in *Proceedings of IEEE/IFIP International Symposium on Rapid System Prototyping*, 2009.