CANNoC: An open-source NoC architecture for ECU consolidation

Kizheppatt Vipin
School of Electrical and Computer Engineering
Nazarbayev University
Astana, Kazakhstan-010000
Email: vipin.kizheppatt@nu.edu.kz

***Abstract*—Modern cars contain more than 50 electronic control units (ECUs) whose number is expected to grow as more and more features are incorporated. The concept of one feature per ECU is contributing to this growth which increases the car kerb weight leading to lower fuel efficiency. Researchers have proposed ECU consolidation to addresses this issue by combining multiple ECUs on the same chip. This paper discusses the implementation of a network on chip (NoC) architecture which enables integrating multiple ECUs on a reconfigurable chip (FPGA) and interfacing it with a standard CAN network. The architecture enables reducing the number of ECUs required for non-critical functions and lower communication overhead to the main bus.**

***Index Terms*—Network on Chip, FPGAs, ECU, CAN**

I. Introduction

Virtually every aspect of modern automobiles are controlled by electronics. Small embedded compute units called electronic control units (ECUs) manage both safety and time critical operations such as ABS, EBD, airbag etc. to driving experience enhancement systems such as infotainment systems and rain-sensing wipers. Today a moderate car incorporates about 50 ECUs and several luxury cars have more than 100 ECUs. The global automotive ECU market size was estimated at USD 33.50 billion in 2016 [1]. This is expected to increase further as more and more features such as automated driving, inter-vehicular communication systems, co-operative traffic management etc. become popular.

Adding new ECUs to manage more and more functionalities increases the overall kerb weight of the vehicle. This leads to lower fuel efficiency and higher battery power consumption. One possible solution to this is ECU consolidation which refers to implementing more than one functionality in a single chip. This consolidation can be at both hardware as well as software levels. Through software consolidation, multiple ECU functions are executed using a single processor using visualization [2]. Consolidation raises other challenges such as meeting the functional safety requirements laid down by standards such as ISO 26262 [3]. Running multiple software on a single processor raises security and interference concerns. The solutions could be at hardware level such as ARM TrustZone implemented in NXP multi-core SoCs, or at an embedded software level using a Hypervisor to create virtual environments for each application [4].

In recent years field programmable gate arrays (FPGAs) are proposed as a platform for ECU consolidation [5]. They have the advantage of providing complete isolation between ECUs at the same time hosting multiple ECUs in the same chip depending upon its logic capacity. It is also possible to modify/upgrade ECUs after deployment due to their reconfigurability. Studies have also shown that they are suitable for time and safety critical application due to the fast response time of hardware implementations compared to their software counterparts [6]. But due to the possibility of single point of failure, initial deployment might be restricted to non-time and safety critical applications.

In this paper we introduce a lite-weight network on chip (NoC) architecture which supports ECU consolidation on FPGA implementations. One of the processing elements of the NoC is a controller area network (CAN) controller, which manages the data movement between the FPGA and the vehicle CAN network. The controller aggregates the traffic from the ECUs and transfer them to the CAN bus. Similarly it accepts data from the CAN bus and broadcasts to the ECUs. ECUs which communicate frequently can be consolidated to the same FPGA thus creating a sub-network. The communication within the subnetwork is not transmitted to the main bus thus reducing the CAN communication overhead. This subnet also enables developers to have custom communication and data encryption methods for added security. The platform is available as an open-source for verification by the community.

The remainder of this paper is organized as, Section II discusses the background, Section III discusses the architecture of the proposed NoC architecture, Section IV discussed the performance metrics and V concludes the paper and gives the future research directions.

II. Background

Electronic control units (ECUs) are generally composed of a microprocessor/micro-controller, associated memory and a network interface to one of the automobile communication buses. They generally run a real-time operating system and manage one of the automotive subsystems. Controller area network (CAN) introduced by Robert Bosch GmbH is one of the most popular bus standards used for interconnecting ECUs [7]. It was later standardized by International Standard Organization as ISO-11898. It is a message-based serial communication protocol with moderate data rate up to 1 Mb/s.

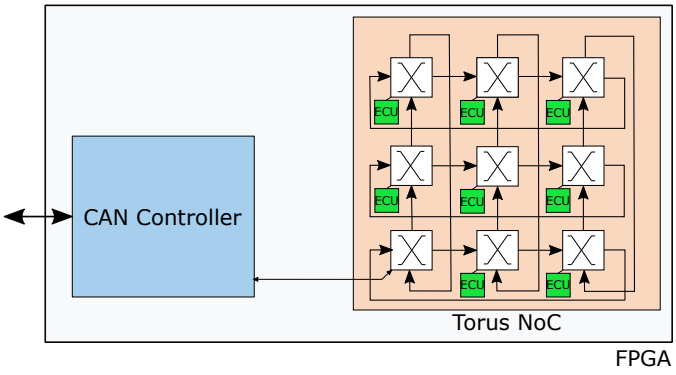

Fig. 1. CANNoC System Level Architecture

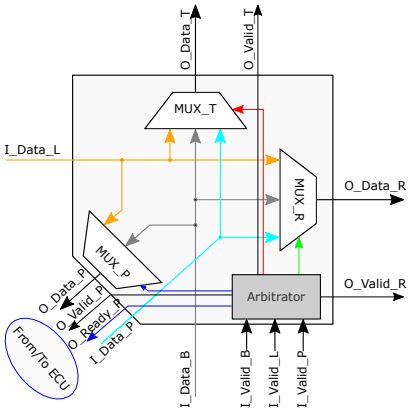

Fig. 2. Internal Architecture of a Switch

Network on chip is an interconnect approach that helps different IPs and subsystems in a system on chip (SoC) to communicate with each other in a scalable manner. In this approach each processing element (PE) is connected to a switch and the switches are interconnected with each other. For ECU consolidation, each PE represents an ECU as shown in Fig. 1. Data transfer happens between the PEs in packet format and the traffic is managed by the switches. Based how switches are interconnected, there are different NoC topologies such as mesh, torus, star, ring, butterfly fat tree (BFT) etc. Torus topology is a two dimensional arrangement of switches with all of them, including the ones along the edges having four connections.

In this work we propose to use a deflection torus based NoC architecture for ECU consolidation called CANNoC. A deflection torus has lower throughput compared to other topologies such as mesh due to the lack to internal buffering [8].. But it is very lite-weight and the performance offered is more than sufficient to meet automotive network requirements. Using a less-resource intensive architecture enables leaving most of the FPGA chip area for ECU implementation. The NoC is capable of communicating with the external world through a CAN interface, which makes the platform ideal for automotive applications.

III. Architecture

The complete system architecture of CANNoC platform is depicted in Fig. 1. It is composed of a NoC packet routing infrastructure with a CAN communication controller interfaced with one of the switches. The number of switches in the NoC (X and Y dimensions) and thus the number of ECUs that can be interfaced with the NoC can be configured.

A. Switch Architecture

The internal architecture of a torus switch is as shown in Fig. 2. Each switch can accept data from its left and bottom neighbors as well as from the ECU. The switches are buffer-less and received data is routed to immediate neighboring switches at the right, top or to the ECU. Communication channels between the switches as well as the ECU conforms to AXI-4 (Advanced eXtensible Interface) stream standard. Flow control is implemented through AXI-stream control signals as shown in Fig. **??**. The *o_valid_p* signal is asserted whenever the switch transmits data to the ECU. Similarly whenever the ECU wants to transmit data, it should assert the *i_valid_p* when the data is placed on the *i_data_p* bus. The switch asserts the *o_ready_p* signal whenever it accepts the data. The ECU should hold the data on the bus until the switch accepts the data.

When ECUs are implemented as custom hardware modules in the FPGAs, they can be customized to have AXI-4 stream interface and directly interfaced with the NoC switches. With minor modifications the AXI interface can be interfaced with standard microprocessor I/O ports also. An asymmetric FIFO is required if the data bus width of the processor is different from that of CANNoC switch data width. Similarly the *o_ready_p* and *i_valid_p* signals need to be polled or combined as an interrupt signal for detecting data transfer and reception. These processors can be soft processors implemented in FPGAs such as microblaze [9] or external microprocessors from TI, NXP etc. to which the signals are routed through the FPGA pins.

For better performance the switch implements a full crossbar using multiplexers, enabling data from any ingress port to be routed to any egress port. Each multiplexer receives data from left, bottom and PE (ECU) interfaces and manages one of the output interfaces. The control signals to the multiplexers are managed by an arbitrator, which also implements the packet routing algorithm. There could be scenarios where data flits are available from all the input interfaces (from left, bottom and PE) of a switch. But there are only two egress ports available to route them (right and top). Since the switches are buffer-less, to avoid deadlock situations a switch is free to back pressure flits from its corresponding PE until one of the output interfaces becomes available.

For the multiplexer controlling the PE (MUX_P in the figure), the highest priority is given to the data coming from the PE connected to that particular switch to minimize latency. For MUX_R, the highest priority is for data coming from left and for MUX_T, the highest priority is for data from bottom. Internal pipelining is implemented between the multiplexers to achieve better clock performance.

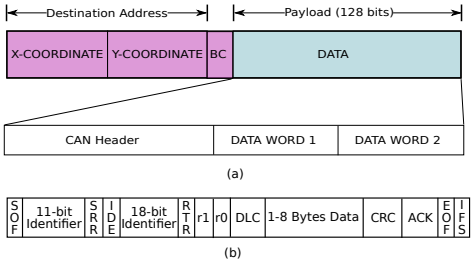

Fig. 3. (a) Packet structure of CANNoC Broad cast packet (b) CAN 2.0 B extended frame format

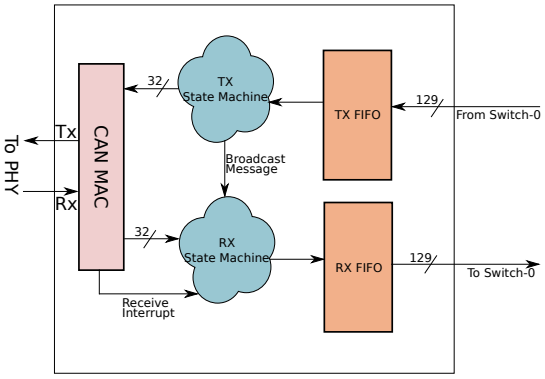

Fig. 4. Architecture of the CANNoC CAN controller

B. Packet Format and Routing

Each switch in the NoC has a unique address based on its X and Y coordinates. The general packet structure used in CANNoC is shown in Fig. 3(a). Data should be sent conforming to this packet format by every ECU in the network. The X and Y coordinate fields are used to route the packets to the appropriate destinations. If the intended communication is within the NoC towards a single ECU (unicast), the corresponding destination ECU coordinates should be entered as the X and Y coordinates. If the data has to be multicasted to every ECU in the NoC, both X and Y coordinates should be set to zero and the BC (broadcast) bit should be set to zero. If the data has to be broadcasted including to the external CAN interface, both X and Y coordinates should be set to zero with the BC bit set to 1. Each packet carries a maximum of 128 bits of data in the payload field.

The data field format is not constrained by the NoC for unicast and broadcast transmissions. It could be proprietary in nature but due to the packet switched architecture the order in which data is received could be different from the order in which it was transmitted. If strict ordering is required, the data field might require a packet number or a time-stamp for reordering at the receiver. When the data is intended for transferring to the external CAN interface, the payload should conform to the CAN *extended frame format* shown in Fig. 3(b). Here the payload field contains both the CAN frame header as well as the data to be sent over the bus. The CAN header field contains the 11-bit identifier, remote transmission request bit (RTR)/substitute remote request bit (SRR), identifier extension bit, 18-bit extended identifier, remote transmission request bit (RTR), and 4 bit data length code (DLC). The maximum data carried by a single frame is 64 bits (8 bytes). The packet uses 64-bits for specifying the header and lower 28 bits are unused. The remaining 64-bits of the payload field is used for carrying the CAN data. The remaining fields in the CAN frame such as SOF (start of frame), EOF (end of frame), CRC etc. are automatically inserted by the CAN controller.

When a switch receives a packet with matching destination address, it is routed to the ECU connected to it. If the address does not match, it is routed based on the dimension order such that if the X-coordinate is not matching, it is routed to right and if the X-coordinate matches and Y-coordinate does not match, it is routed up. If a switch receives more than one packet destined for the same egress port, it is routed based on the multiplexer priorities as discussed in section III-A.

C. CAN Controller

Fig. 4 shows the internal architecture of the CAN controller used in CANNoC. The controller is interfaced with switch-0 of the network architecture. As discussed before, any ECU intending to send data to the system CAN bus or multicast to every other ECU within CANNoC should send it to the CAN controller. The packets received from the network are initially stored in a FIFO (Tx FIFO). The write and read ports of the FIFO are 129-bit wide (128 bit data + multicast bit). The Tx state machine continuously monitor the empty signal from the Tx FIFO to check reception of any new packet. Once a packet is received, it checks the BC bit to determine the transmission type. If it is set, the packet is forwarded to the CAN MAC controller as well as to the Rx state machine. If the broadcast bit is zero, it is forwarded only to the Rx state machine. The write interface of the MAC interface is 32 bit wide and the state machine converts the 128-bit data to four 32-bit transactions.

The RX state machine injects the packets received from the CAN MAC layer as well as the broadcast and multicast packets received from the TX engine to the NoC. The MAC controller is configured to generate an interrupt signal whenever a new packet is received. The RX state machine replicates the received packets and sends to all the ECUs connected to the NoC. The broadcast as well as multicast packets are also duplicated in the similar way. The packets are stored in a FIFO (Rx FIFO) before sending to the network.

Xilinx AXI CAN controller is used to implement the MAC layer [10]. This layer conforms to Conforms to the ISO 11898-1, CAN 2.0A, and CAN 2.0B standards can support CAN bus speed up to 24MHz. The controller accepts the CAN header and data through a 32 bit wide AXI-Lite interface and serialises confirming to CAN standard. The layer also adds the *sof* and *eof* delimiters as well as inserts the 16 bit CRC field. The transmit interface of the MAC layer is connected to the TX state machine and the receive interface is connected to the RX state machine. Before any transaction, the TX state machine configures the controller such as interrupts, inter-frame delay etc. The output of the MAC layer are the CAN serial receive and transmit signals which have to be interfaced with an external PHY chip for interfacing with a standard CAN bus.

TABLE I
CANNoC MODULE-LEVEL RESOURCE UTILIZATION

Submodule (bits)	LUTs	FFs	BRAMs
CAN Controller	936	1567	6
Deflection NoC	2086	1582	0
Total	**3022**	**3149**	**6**

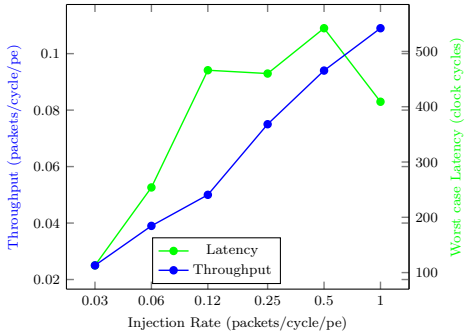

Fig. 5. Throughput, Latency vs injection rate

IV. CHARACTERISATION

CANNoC is implemented in Verilog HDL and characterized using Xilinx FPGAs through Vivado tool flow. It is simulated extensively through Vivado Simulator. Table I lists the total resource consumption of CANNoC when targeting a Xilinx Artix-7 xa7a15tcpg236-2I FPGA (automotive grade). This is a low cost, low power consuming FPGA with limited logic capacity. The NoC is configured as a 2×2 deflection torus which enables interfacing with 3 ECUs (one interface is used for the CAN controller). Overall it consumes about 29% of LUTs, 15% of flip flops and 25% of Block memory. The majority of chip area is available for implementing other ECUs. The bus operates at 20MHz and the NoC subsystem is capable of running up to 206 MHz. The total on-chip power consumption for the design is 0.177W (0.109 dynamic and 0.069 static) at 25C when running at 200MHz NoC clock frequency 20MHz CAN bus clock frequency.

Fig. 5 shows the throughput and worst-case latency for CANNoC NoC subsystem when configured in a 10×10 format. Here packets are sent from every ECU to random destination ECUs. The injection rate decides at what rate PEs try to send packets to the network. The throughput saturates at about 0.15 packets/cycle/ECU when injection rate is 1 packet/cycle/ECU. For higher injection rate, latency increases due to more number of mis-routings. But at 200MHz with 128 bit data width, the throughput of the network is 2.5 GBytes/sec which is several order above the maximum supported speed of CAN protocol.

V. CONCLUSION

In this paper we discussed the design and implementation of an NoC based infrastructure for ECU consolidation and CAN bus interfacing. The proposed architecture is suitable for non-critical ECU consolidation and is very lite-weight leaving most of the FPGA resources for ECU implementations. The NoC architecture can be also interfaced with existing microprocessor-based ECUs by adding a few additional logic. The performance provided by the NoC is more than adequate for the current real-time requirements of automobiles. This platform is provided as an open source enabling other researchers to verify its functionality by interfacing with practical ECUs [11]. Our future research direction will be to develop and integrate FPGA-based soft ECUs with CANNoC. We also intend to integrate other automotive bus controllers such as Flexray and CAN-FD.

REFERENCES

[1] "Automotive Electronic Control Unit (ECU) Market Size, Share & Trend Analysis Report By Application Type (ADAS & Safety System, Chassis Electronics, Powertrain, Infotainment), By Region, And Segment Forecasts, 2018 - 2025," Grand View Research, Tech. Rep., Jan. 2018.
[2] G. Doll, "A smart way to drive ecu consolidation," WindRiver, Tech. Rep., Nov. 2015.
[3] *Road vehicles – Functional safety*, International Standard Organization Std. ISO 26 262, 2011.
[4] "Building a secure system using trustzone technology," ARM Limited, Tech. Rep., 2009.
[5] S. Shreejith, S. A. Fahmy, and M. Lukasiewycz, "Reconfigurable computing in next-generation automotive networks," *IEEE Embedded Systems Letters*, vol. 5, no. 1, pp. 12–15, March 2013.
[6] K. Vipin, S. Shreejith, S. A. Fahmy, and A. Easwaran, "Mapping time-critical safety-critical cyber physical systems to hybrid fpgas," in *IEEE International Conference on Cyber-Physical Systems, Networks, and Applications*, Aug 2014, pp. 31–36.
[7] *automotive subsystem*, Robert Bosch GmbH Std., Rev. 2, 1991.
[8] N. Kapre and J. Gray, "Hoplite: Building austere overlay NoCs for FPGAs," in *Proceedings of International Conference on Field Programmable Logic and Applications (FPL)*, Sept 2015.
[9] *MicroBlaze Processor Reference Guide*, Xilinx Inc., Oct. 2016.
[10] *LogiCORE IP AXI Controller Area Network*, Xilinx Inc., Dec. 2010.
[11] [Online]. Available: https://github.com/vipinkmenon/cannoc