Development and Implementation of Controller Area Network (CAN) Device Driver for a 32-Bit RISC Architecture based RENESAS Micro-Processor Series with AUTOSAR Software Conformance

Ram Prasadh Narayanan
Amrita Vishwa Vidyapeetham
Coimabatore, India

Ahamed Aslam C
Amrita Vishwa Vidyapeetham
Coimabatore, India

Shanthini B M
Amrita Vishwa Vidyapeetham
Coimabatore, India

ABSTRACT
Device drivers abstract away the technical details and the complexity associated with interfacing a peripheral with the controller network. This in-turn increases the simplicity and reliability of the application software, in-turn reducing the product's time to market. Device drivers provide an interface platform for binding the processor-dependent physical layer and processor independent application layer. So it is corroborated that the device driver must communicate with both the layers, there-by reducing or at the most excluding the processor dependency for application development.

The main objective of this work is to provide driver support for Controller Area Network (CAN) communication for a 32-Bit RISC Architecture based Renesas Micro-Processor series, with software conformance for Automotive Open System Architecture (AUTOSAR). AUTOSAR establishes open standards for automotive E/E (Electrics/Electronics) architectures that will provide a basic infrastructure to assist with developing vehicular software, body electronics etc. and management for all application domains like user interfaces, Multi-media etc. This includes the standardization of basic systems functions, scalability to different vehicular platforms, transferability throughout the network and integration from multiple suppliers, software updates and upgrades over the vehicle's lifetime. These features of this particular series of processors provide better interfacing for automotive body applications. The main objective is to provide CAN device driver support to a processor series with AUTOSAR conformance and also includes the support of CAN with flexible data rate or the improved CAN (CAN FD) to improve the bandwidth of the CAN communication network.

Practical Extraction and Reporting Language (PERL) will be used for the Link Layer programming. IDEs or tools are used for (Run-Time Environment) RTE configuration, RTE generation and generation of AUTOSAR Basic Software code, which will be used to generate and validate the conformance of the software with AUTOSAR standard.

General Terms
AUTOSAR, Controller Area Network (CAN)

Keywords
PERL, MCAL, Device Driver, CAN FD, RENESAS micro-processor, 32 bit RISC Architecture.

1. INTRODUCTION
Newer automobiles contain more than 70 ECU's and more than a million lines of code for controlling different applications such as engine control, airbags, antilock braking, cruise control, electric power steering, mirror adjustment, battery and recharging system for hybrid or electric cars and so on. This means that the data traffic is steadily growing and there is a need for reliable and a priority based communication system, which can ease the node-node communication, without compromising the efficiency in data transfer, in spite of increasing data traffic. Controller Area Network (CAN) can provide a homogeneous, priority based data transmission. Due to the scalable feature, a communication environment created with the CAN will be able to meet the scalability requirements imposed by the today's automotive systems and standards, where the number of ECUs' to interconnect steadily increases with time. A brief description on CAN Protocol is given below.

The Controller Area Network (CAN) is a standard for distributed communication with built-in error detection and fault handling capabilities. CAN has been most widely adopted and used for implementing a communication network by the automotive industry due to its inherent strengths of communication mechanism and tremendous flexibility in system design [1]. The CAN bus, developed by BOSCH, is a priority based Multi-master message broadcast system with a maximum signalling rate of 1 Mbps at a maximum bus length of 40 meters or 130 feet. This data holds well when using a twisted wire pair for CAN Communication implementation. Data consistency in communication is maintained in every CAN node of the system, since unlike traditional networking protocols like USB or Ethernet, CAN broadcasts only short messages on the bus which zeroes the necessity for a central bus master supervision and results in low latency between transmission request and start of transmission [1]. Apart from this, CAN provides sophisticated error-detection and error handling mechanisms with dedicated bits assigned for Cyclic Redundancy Checks (CRC). CAN also provides high immunity against electromagnetic interference. It uses a mechanism where erroneous messages are automatically retransmitted if a transmission fails. The data consistency remains guaranteed through-out the system operation when CAN Communication network is implemented.

The CAN communication officially conforms to the ISO-11898: 2003 and is also recognized by the Society of Automotive Engineers (SAE) as a network communication

standard. CAN protocol can be considered in the context of the seven layer OSI model for communication. Implementation of CAN protocol will depend on the physical layer transceiver, CAN peripheral in the controller, which adheres to ISO 11898 standard and the application layer featuring the CAN software [2]. The standard CAN communication architecture is shown below in Figure 1.

The automotive industry has been pushed to adopt CAN networking in vehicles due to standardization in CAN communication. Advanced software and Micro-Controllers are used by new generation vehicles to introduce the concept of providing communications services in them. A common CAN bus is used by many ECUs for communication due to different purposes.

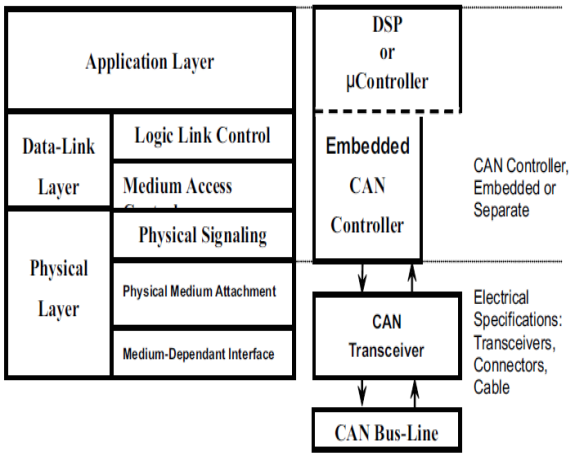

Fig 1: Standard CAN Communication Architecture

The CAN protocol is designed for safe data transfers and provides mechanisms for error detection, signalling and self-diagnostics, which include prediction of fault confinement, there-by preventing faulty nodes from affecting the status of the network. The general measures for error detection are based on the capability of each node of monitoring broadcast transmissions over the bus, whether it is a transmitter or a receiver, and to signal error conditions resulting from several sources. Corrupted messages are flagged by any node detecting an error consequently aborting the message and retransmitting it automatically.
There are 5 different error types:

• **BIT ERROR:** The bus is monitored by the unit which sends a bit on it. When the bit value that is monitored is different from the bit value that is sent, a BIT error is detected.

• **STUFF ERROR**: CAN protocol defines STUFF error in such a way that the 6th consecutive occurrence of the same bit in a message field subjects to the occurrence of bit stuffing error.

• **CRC ERROR**: If the computed CRC and the calculated CRC in the stored message frame differs the CAN protocol throws a CRC error to the ECU associated.

• **FORM ERROR:** The form error occurs when the fixed-form bit field contains one or more illegal bits.

• **ACKNOWLEDGMENT ERROR:** If a recessive bit is found on the acknowledge slot by the transmitter, the acknowledge error is thrown on the bus.
There are some key factors which influence the implementation of AUTOSAR stack in software development for automobiles [3]. They are listed below.

- Creates a generic standard for software development and testing, which makes automobile manufacturers and software developers to adopt the standard and compete on the implementation rather than design.
- It facilitates exchange and update of software and at-most changes can also be brought up to the hardware over the life-time of the vehicle.
- The functionalities of the hardware and software services are de-coupled with the engagement of AUTOSAR stack into vehicular software development.

AUTOSAR (AUTmotive Open System ARchitecture) is an automotive software standard designed by a group of Original Equipment Manufacturers (OEMs), suppliers and tool developers to facilitate a generic automotive and vehicular software development.
The AUTOSAR stack is expressed in Figure 2 and is explained below.

1. The Basic software layer (BSW) is just an abstraction of the underlying hardware. It provides hardware-independent services to the subsequent upper layers. Though this layer is not fully platform independent, it makes the upper layers independent of the implementation platform.

2. The Run-Time Environment (RTE) is used for handling the data frame exchange between the upper and the lower Micro-Controller Abstraction Layer (MCAL) and vice versa.

3. AUTOSAR does not define any standard for developing the application layer but it does perform the actual functionality of interacting with the RTE and the user [4].

A basic Functional Structure of the AUTOSAR is shown in Figure 1.

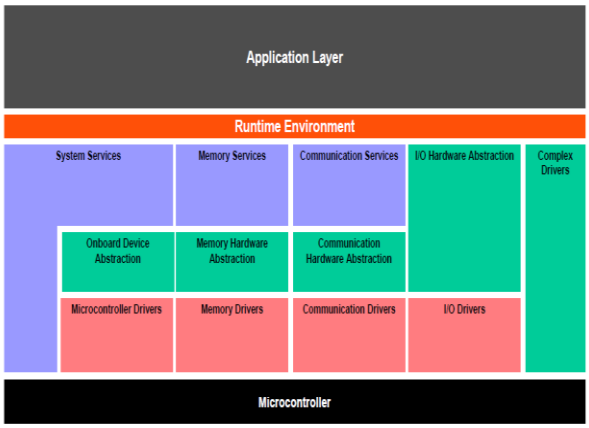

Fig 2: AUTOSAR Layered Software Architecture

The AUTOSAR architecture model is supported by automated methodologies for generating software executable for ECUs, starting from the physical topology of the hardware to the application concerned. It reduces development cost of the software by imposing software re-use.

To reduce cost and time to market of automotive software systems and simultaneously increase the products' quality, the AUTOSAR component paradigm has achieved broad acceptance within the automotive industry. AUTOSAR enables a use of component based system design model for the design of a vehicular system. Standardized interfaces for all the application software components necessary to build the different automotive applications are specified in the AUTOSAR-standards. The developer has the freedom to deploy the functionality by defining only the interfaces to be used.

In AUTOSAR applications, software concerns are covered by standardized software components specified, while infrastructural concerns are handled within middleware layered component design. The so obtained separation of layers leads to an increase in application quality, reusability and maintainability of the software, and hence provokes a reduction of cost and time to market. Implementing a device driver with conformance of AUTOSAR helps in developing a platform independent application, which increases the possibility of software re-use [5]. The Input description, system configuration, ECU Configuration and the generation of software executable are designed with the methodology and architecture described by AUTOSAR.

The main objective of this work is to provide driver support for Controller Area Network (CAN) communication for 32-Bit RISC Architecture based RENESAS Micro-Processor Series, with software conformance for AUTOSAR. The Standard establishes open standards for automotive E/E (Electrics/Electronics) architectures that will provide a basic infrastructure to assist with developing vehicular software, user interfaces, body electronics etc. and management for all application domains. The implementation will be done through Practical Extraction and Report language, AUTOSAR Extensible Mark-up Language (ARXML).

2. LITERATURE SURVEY

Years ago, CAN was far behind to even consider for using in Vehicular software development. But, due to the introduction of stringent norms to adhere for automobiles in-order to reduce the amount of pollutants left by the automobiles, manufacturers were forced to find out mechanisms through which efficient communication happens between ECUs', resulting in proper usage of the mechanical resources in an automobile. It lead to an era where, automobiles were not governed by only mechanical parts, but electronics took the front stage. CAN protocol started playing an important role in control and communication between ECUs' [6]. Newer vehicles contain more than 70 ECU's for different applications such as engine control, airbags, antilock breaking, cruise control, electric power steering, mirror adjustment, battery and recharging system for hybrid or electric cars and so on. This requires an efficient and a reliable communication standard to establish a network between various modules of the system. Can protocol was able to meet the demands and quite sooner, it became a mandatory implementation for communication networks in vehicular software. Since being a Single bus, priority based message transmission protocol, the hardware implementation became much simpler compared to a network implementation of Ethernet or a Flex ray Protocol. At a point, the number of devices connected through CAN network reached the maximum limit and this led to increasing the frame size of the CAN protocol where the 11-bit identifier was replaced with a 29-bit identifier [7].

With the introduction of AUTOSAR, the vehicular software became more standardized, which led to automotive and third party vendors to follow the standardization, as it became easy for error diagnostics and testing. Though many other protocols like the LIN, Flex Ray, and Ethernet were also used alternatively, with the exception of Flex Ray, none other protocols were able to provide the efficiency and robustness in vehicular software which CAN provided.

3. DESIGN METHODOLOGY

Development of correct requirements at the beginning is considered an important precondition for successful driver development. Rigorous implementation of requirements in driver development is especially critical, since requirements affect both software and hardware. The goal is to identify elements for effective requirements in development and implementation of the driver. The AUTOSAR software requirement specification for CAN Driver, CAN Interface, CAN Transceiver Driver, CAN State Manager, Communication Manger, hardware requirements of the CAN controller is captured. The hardware requirements are captured according to the specifications of the microcontroller [8]. The Embedded driver design in AUTOSAR is entirely different from Linux device drivers.

An overall ECU configuration overview and the automated generation of source and header files from the system description input are shown in Figure 3.

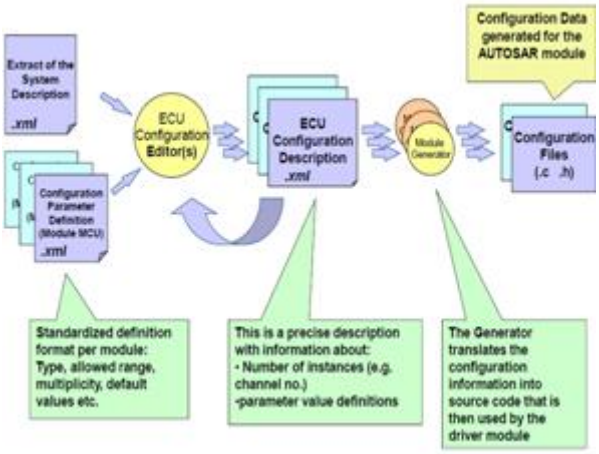

Fig 3: ECU Configuration Structure Overview

The driver development is divided into Static part and Dynamic part. The static part remains fixed, determining the essential non-changeable features needed for the driver, whereas the dynamic part can be changed, depending on the application and the Run Time Environment (RTE). The static part includes API function calls, constants, global variables that shall not be changed once designed. The dynamic part includes configuration parameters for the required module, which may change during the run time. There are standard parameters and vendor specific parameters, where the former one is used by AUTOSAR. It is generic and standardized whereas the latter one mentions the specifications given by hardware.
In this work, an MCAL (Microcontroller Abstraction Layer) generator tool, a PC-based tool, is used. The tool consists of

two parts. One is the Driver configuration part and other is the Code generation part. The input to Driver configuration part called the Parameter Definition File (PDF). The output is the Description file. Both the PDF and definition file are in AUTOSAR Extensible Mark-Up Language (ARXML) format [9]. This description file is one of the inputs to the static code generation part. As mentioned earlier, the static part of the code involves configuration and validation of parameters which remains unchanged throughout the operation of the driver. For Ex., The Controller ID, default and the available baud rates at which the communication can happen, the total number of CAN peripherals associated to the communication etc., are going to be unchanged. So it will be better if these parameters are configured and used unchanged throughout the software operation. Once these parameter values are decided, they are fed as input, the Configuration tool produces a Description file in ARXML Format. The description file contains the values assigned to the static parameters which must be extracted to generate a configuration source and header file. The extraction of the parameter values from the description file can be done with the help of PERL scripting. Module inter-dependency and inter-parameter dependencies must be taken into account while generating the source and header files for the static driver configuration. Here, the description file generated from the Generation tool is manipulated by a series of PERL scripts, which parses, extracts and validated the parameters set by the user in the tool, in an appropriate way. The ECU Configuration description file provided in the command line is parsed and the data structures are created, holding the parsed output. This has to be done in parallel with interaction from the user. The functions which are commonly used and shared by other modules and common for the generation tool are combined into one single entity and included in the process.

Intermediate data structures are generated, which would aide in generation of C Source and Header files. There are modules to check the correctness of the configuration parameters in the ECU configuration.

There are some applications which require a higher bandwidth communication network implementation. To achieve this, AUTOSAR provides support for higher bandwidth communication protocols like FlexRay, MOST etc. But at times, it becomes an overhead to the software product to include more protocols yet the requirement may insist on a higher bandwidth communication. This led to the inclusion of CAN with Flexible Data rate (CAN FD), where the main difference between an ordinary CAN and a CAN FD is that, more than 8 bytes of data (up to 64 bytes) can be transmitted in a CAN FD frame format as compared to only 8 bytes in a standard CAN [10]. CAN FD also provides an option to switch between different bit rates after deciding the arbitration and it can provide a maximum data transfer rate of 8Mb/s. To be precise, CAN FD offers an adaptable communication speed and provides the facility to transport data at a faster rate. We have researched on ways on implementing CAN FD into the communication mechanism. We will be requiring hardware and MCAL support for CAN FD implementation [11]. Further research and implementation has to be done on providing support to CAN FD in AUTOSAR and its possible application areas must be figured out.

4. RESULTS

The CAN Device Driver for a 32-Bit RISC Architecture based RENESAS Micro-Processor Series with AUTOSAR Software Conformance has been developed and implemented. Test Applications were developed for every module of the drivers and tested for the module's correctness. After successful testing, the AUTOSAR Conformed program is build and the debugged program is flashed into the Processor. From this point, the ECU Module in which the CAN driver was flashed will act as a stand-alone device, successfully participating in a CAN Communication network.

Future work which can be carried out using CAN with flexible data rate (CAN FD) has also been considered. By this, the CAN is implemented through the above-mentioned Micro-Processor series and tested for its working.

5. REFERENCES

[1] Di Natale, Marco, Haibo Zeng, Paolo Giusto, and Arkadeb Ghosal. Understanding and using the controller area network communication protocol: theory and practice. *Springer*, 2012.

[2] Navet, Nicolas, and Hervé Perrault. "CAN in Automotive Applications: a look forward." *13th International CAN Conference*, Hambach Castle. 2012.

[3] Bo, Huang, et al. "Basic concepts on AUTOSAR development." International Conference on Intelligent Computation Technology and Automation (ICICTA), Vol. 1. IEEE, 2010.

[4] AUTOSAR Consortium. AUTOSAR-Layered Software Architecture. AUTOSAR Consortium, Tech. Rep., 2008, Available: http://-www. autosar. org/download/specs_aktuell/AUTOSAR_LayeredSoftwareArchitecture.pdf, 2008.

[5] Bunzel, Stefan. "Autosar – the standardized software architecture." Informatik-Spektrum (2011): 79-83.

[6] AUTOSAR, "Specification of CAN Driver." http://www.autosar.org/download (2007).

[7] AUTOSAR, "Specification of CAN Interface." 2009-03-09). http://www.autosar.org/download

[8] An-juna, W. A. N. G., Jian-chuna JIANG, and C. H. E. N. Pei-ranb. "Device Driver Software Development According with AUTOSAR Specification." International Conference on Computational Engineering (2011).

[9] Robert.S, Dr. Jayasudha J.S, Anurag "TCP/IP Stack Implementation for Communication over IP with AUTOSAR Ethernet Specification" International Journal of Engineering and Innovative Technology (IJEIT) Volume 3, Issue 1, July 2013.

[10] Fürst, Simon, et al. "AUTOSAR–A Worldwide Standard is on the Road." 14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden. 2009.

[11] CAN in automation. [Online]. Available http://www.can-cia.org, June 2008