International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 11, November-2014

Diagnostic Protocol Stack on LIN Interface

Jeevan Peter Dsouza,
Student
Dept. of PG Studies, UTL,
VTU Extension Center,
Bangalore, India

Ramana Reddy
Assistant Professor,
Dept. of PG Studies, UTL,
VTU Extension Center,
Bangalore, India

Dr. Siva Yellampalli
HOD,
Dept. of PG Studies,
UTL, VTU Extension Center,
Bangalore, India

***Abstract* - This paper deals with design, implementation and testing of ECU diagnostic protocol stack on LIN2.0 interface using Microchip 16F87X controller, sensors, actuators, current drivers and signal conditioning circuits. Protocol stack intended for use in vehicle networks applications, where transferring and processing digital data at low costs and by low power consumption is high priority. The diagnostic protocol stack developed using low cost controller stands as an economical solution which also reduces complexities associated with a typical commercially automotive CAN, MOST or K-line stack network. The implementation of diagnostic protocol stack using CAN, MOST or K-line interface requires extra controller functionality and also increases the area and cost of the system. The ECU diagnostic stack core can be implemented with local interconnect network interface and its suits for various Vehicle network demands. The LIN interface based diagnostic protocol stack core can be reduced to smaller size with optimizing code and speed of execution with an appropriate implementation.**

Keywords - Automotive,ECU,CAN,LIN,Diagnostic,Stack,Network, Protocol, Drivers.

I. INTRODUCTION

LIN is a very popular and commonly used network interface when connecting to an Electronic Control Unit (ECU) with Sensors and Actuators in automotive systems. When connecting an electronic control system to a vehicle network for simple point-to-point communication, additional network circuits and functionality is required, which came at a high cost. Standalone processor or controllers are used to implement the network protocol stack. Now with the existing embedded technology it is feasible to implement an application-tailored subset of a diagnostic protocol stack to achieve a straight-forward and cost-effective connection to a network. Although the Diagnostic protocol stacks can be implemented on various numbers of network interfaces such as CAN and K line, this paper will focus on the implementation of ECU diagnostic protocol stack on local interconnect networks interface [2].

The design handles multiple communicating network node that is both physical addressing and functional addressing. This project work streamlined for diagnostic stacks of automotive embedded network system requirements.

II. BACKGROUND

A. Protocol Stack

The Open Standards Interconnect (OSI) model is theoretical and practical model which is used to describe the behavior of a network stack and related communication issues. The Open Standards Interconnect protocol stack model consists of seven layers, But the ECU Diagnostic protocol stack can be implemented using only physical, data link, network and application Layers [7].

UDS,OBD2,KWP2000 (Application layer)
ISO15765 (Network layer)
LIN2.0 (Datalink layer)
RS232 (Physical layer)

Figure 1. Diagnostic Protocol Stack

B. Local Interconnect Network

The LIN is a serial network protocol used for communication between sensors and actuators in vehicles. The need for a cheap serial network arose as the technologies and the facilities implemented in the car grew, while the CAN bus is too expensive to implement for every component in the car. The LIN nodes are implemented as ASICs or 8 Bit Microcontroller. Supports the Low cost single-wire implementation (Enhanced ISO-9141) and speed up to 20Kbit/s. LIN interface is based on Single Master or Multiple Slave concept. LIN driver is low cost silicon implementation based on common UART or SCI interface hardware [2].

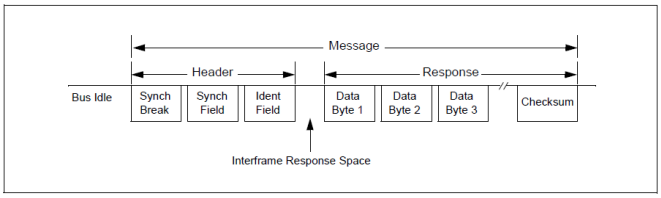

Figure 2. LIN Frame Format

C. Network Level Protocols

The transmission of longer messages is made by segmentation of data to be transmitted. A segmented message is transmitted within a set of frames containing a complete message of a diagnostic service. The first segment is called First Frame, the following segment Consecutive Frame. A flow control is used to adjust the sender to the network layer capabilities of the receiver.

The units that are transported in a network layer frame are called Packet Data Unit.

The PCI (Protocol Control Information) contains the network layer flow control information. The PCI type Single Frame (SF) indicates that the transported message fits into the single PDU, i.e. it contains at maximum five data bytes. Multi-PDU message is continued with a number of Consecutive Frames (CF).If more than 15 CF PDUs are needed to transport the complete message, the frame counter is incremented [4].

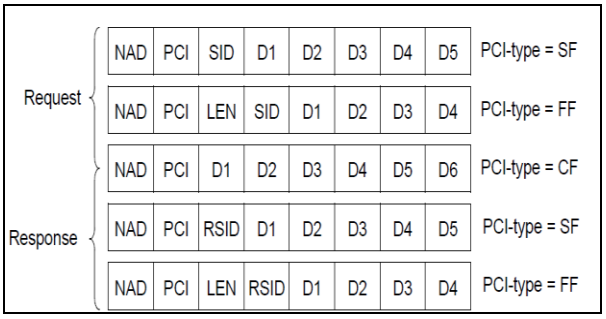

Figure 3. Protocol Data Unit

D. *Diagnostic Services*

Diagnostic services are used by diagnostic tester tools in vehicle communication links for accessing the ECUs information. Services include accessing diagnostic faults codes, read and write parameters, activate diagnostic or test routines, and download or upload software and data. The protocol to be used for accessing the ECUs by the diagnostic tester will be in accordance to ISO 14229 or ISO 14230 specification.

The diagnostic services are grouped into control of diagnostic sessions, Data Transmission, Stored Faults, Input Output Control functional units and identified by Service Identifier [1].

III. DESIGN AND IMPLEMENTATION

The diagnostic stack cores that are described in this paper can be viewed as tailored subsets of the diagnostic stack and one possible configuration is represented related to the OSI model. Diagnostic protocols are normally structured in a layered stack. The stack IS0-15765 at the network layer and LIN driver interface at data-link. The Application Layer may be a software or hardware application that communicates with the used diagnostic core through the network layer. Software application (UDS, KWP200, and OBD2) can be implemented in a separate layer [1].

The Application, Network and Link Layers in the diagnostic stack cores are designed and implemented using Embedded C language. This means that the designs are not restricted to a specific embedded controller.

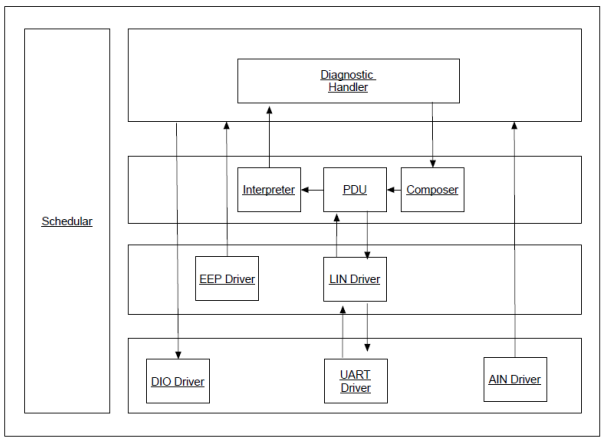

Figure 4. Software Architecture Design

A. *LIN Driver*

The UART implementation of the LIN protocol is the most efficient way of implementing a LIN driver since the UART contains much of the hardware support for generation of LIN bytes, i.e. start bit, 8-data bits, and stop bit format. The first problem to solve for LIN driver implementation using a UART is that of generation of the synchronization break. The maximum number of bits that can be driven to dominant with the UART is 10 bits. Therefore, to generate the synchronization break, a different strategy is used. The synchronization break is generated by changing the value in baud rate register, so that the baud rate is effectively decreased. Therefore writing 0x00 to the UART will result in synchronization break. After the synchronization break is generated, the baud rate is set to the original value for transmission of the synchronization byte and identifier [5].

B. *LIN Receiver*

LIN receiver manages incoming packets and check for a new packet. Once a new packet is detected the packet will be saved byte-wise to the Receiver FIFO. Each byte is also sent to the CRC checker, which calculates the LIN receive frame checksum. When the end of frame is signalled from the LIN the CRC check will be completed and the destination address will be verified. Only node addresses that matches the core address and broadcast addresses are accepted. If the packet node address check fails the packet will be rejected by LIN receiver.

C. *LIN Transmitter*

The Transmitter will check for a send flag from one of the packet types. The Transmitter reads data from the Transmitter FIFO and puts out the transmit packet to the physical data bus and sets control signals. Each byte is sent to the frame CRC generator, which calculates the checksum. When the packet maximum length is reached the calculated CRC is sent along with data bytes.

D. FIFO

The Receiver FIFO temporarily stores the received packet i.e. including the whole frame with LIN header, Diagnostic header etc. Packets with incorrect addresses will be filtered out in the Receiver and are not stored in the Receiver FIFO.

The Transmitter FIFO temporarily stores the entire packet, i.e. including the whole frame with LIN header, IP header etc., which shall be transmitted.

E. PDU Composer/Interpreter

The protocol data unit composer and interpreter module are the main control blocks in protocol stack, which are closely related to each other. The composer and interpreter checks that the incoming packet is valid, manages diagnostic frame in both directions, manages diagnostic requests, and generates diagnostic responses. The frame checksum will always be calculated for transmitted packets by the packet composer. The checksum calculation is based on CRC and is executed in parallel with the data copy to the transmitter or receiver FIFO from the application layer respectively from the receiver [4].

F. Core Drivers

Core drivers are very specific to the processor. The Core drivers includes basic software modules like DIO Driver, UART Driver, Analog Driver, PWM Driver, EEP Driver, FLASH driver etc. which are used protocol stack and application handler to implement the customer requirements.

G. Application Handler

The application handler is responsible for implementation of diagnostic services request and response. The application layer also interact with core drivers to process the request from diagnostic tester.

H. Scheduler

The Scheduler is responsible for time-sharing of CPU among tasks. Typical RTOS based on fixed-priority pre-emptive scheduler which assign each process a priority and the scheduler runs highest priority process ready to run. Basic time rate can be selected depending on the time critical specifications and in this project it is 2ms.

IV. EVALUATION AND DISCUSSION

The system is designed and implemented on an embedded system using Microchip PIC 16F87x controller [3].

Physical layers is designed for single-wire or two-wire bus systems. Design provides interface between diagnostic tool and microcontroller. Physical layer is designed to meet diagnostic systems ISO-9141-2 specification. The ISO 9141-2 interface is 19.2 Kbit/s, single-wired interface compatible with UART or SCI. The interface adapter serves as an electrical converter between ISO 9141-2 and RS 232.

The ISO model specifies the requirements for the diagnostic Protocol stack on which one or several on-vehicle Electronic Control Units are connected to an off-board tester in order to perform diagnostic functions. The protocol made by a whole set from the physical layer to the upper application is called communication diagnosis protocol stack in order to realize the communication mechanism and certain function between diagnosis tool and control systems.

Diagnostic stack core consist LIN2.0 driver interface, ISO 15765 network layer, and Diagnostic services. The proposed small point-to-point network architecture diagnostic core system utilizes logic sharing of the PDU Composer and PDU Interpreter parts of the design. In this network protocol system the application layer can be implemented directly in software and thus omitting the need for an external chip [6].

The network layer performs the transmission and reception of longer messages. The transmission of messages is made by segmentation of data to be transmitted. Segmented message is transmitted within a set of frames containing a complete message of a diagnostic service. The Data Link layer is in charge of diagnostic LIN frames. The diagnostic message length is hold in the Protocol Control Information. Handle the transmission requests from the network, load and activate the transmission channels in the LIN driver.

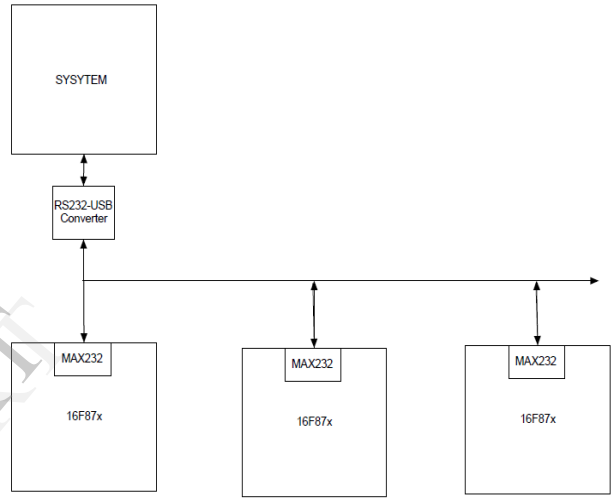

Figure 5. System Block Diagram

V. CONCLUSION

Embedded system design requires decisions regarding simplifications, parallelization, functionality, performance and configuration. The following issues must be considered when designing a diagnostic protocol stack core that is allowed logic utilization, network speed, and number of simultaneous communicating network nodes, limitation of packet loss, supported protocols, duplex mode and size of packets.

1. Physical testing verified that the implementation works correctly in a multi-node LIN network which supports synchronization mode that allows it to adjust its baud rate automatically. The node to node communication process can be handled independently by the LIN module and the complete microcontroller based system, when implemented it supports the maximum LIN bit rate of 19200 bps.

2. Functional testing verified that the network composed of these nodes can communicate with each other and each node correctly raises error conditions. This testing also verifies that the diagnostic service commands and tool interface functions correctly for both physical addressing and functional addressing of nodes.

3. Customer application like UDS, KWP2000 and On-Board Diagnostics can be implemented on diagnostic stack which the reduce costs, and thus presenting a truly reconfigurable system. The core is preferred for point-to-point networks and allows a simple application that offers a cost-effective on-chip solution. The solution gives a reliable and flexible protocol stack core suited for automotive networks.

4. The ECU diagnostic protocol stack fit the design in a cost-effective device and reduces to smaller size by adapting the design to the network system requirements.

5. The Diagnostic stack should be customized when specifying the entire automotive embedded network systems.

REFERENCES

[1] P Dzhelekarski and D. Alexiev " Reading diagnostic data from vehicle OBD2 system " Submitted for publication at 14th International Conference , 2005.

[2] Georgi Krastev ,"Microcomputer Protocol Implementation at Local Interconnect Network " , Computer Systems and Technologies - CompSysTech ,International Conference 2004.

[3] Martin Bates, "Interfacing PIC Microcontrollers - Embedded Design by Interactive Simulation" Second Editon, Copyright @2006 Martin Bates. Page 35, 55.

[4] M Yusairi, B Abu, K Abe " Hardware Design and Implementation of IP-over-1394 Protocol Stack and Its Evaluation " Technical Report of IPSJ, Vol.IAC2002, No.5, Page 51-58, Mar. 2003.

[5] FANG Yi-Yuan , "Design and simulation Of UART serial communication Module Based on VHDL",2011.

[6] K Morita, K Abe, " Implementation of UDP/IP Protocol on FPGA and Its Performance Evaluation " IPSJ General Conferance. Special, Pages 157–158.

[7] Y Izuhara, K Morita, T Tateoka, K Abe , "Specification of TinyIPv6 Protocol Stack for Remote Control and Its Implementation on FPGA " IPSJ Journal,Vol.43, No.11, Pages 3540-3548, 2002

[8] Yongxiang Guo, Wu Deng, "Design of Network device driver in embedded Linux system ", IEEE International Conference on Computer Application , ICCASM 2010.