CAN bus based systems teaching using electronic rapid prototyping

Abdoulaye Diakite, Gordana Collier, Andrzej Ordys & Andy Augousti

Kingston University
London, England, United Kingdom

ABSTRACT: Development and consequent implementation of controller area network (CAN) bus made revolutionary changes in the automotive industry design. Coupled with electronic rapid prototyping (ERP) hardware, it is possible to design, implement and test control algorithms of electronic control unit (ECU) on real-time controllers with real input/output devices. This new approach provides flexibility and speed when developing new features. This article describes one such example where experimental approach to teaching CAN bus based system was developed using an electronic rapid prototyping platform and CAN bus protocol to control a radiator system using sensors and actuators.

INTRODUCTION

CAN Bus and Benefits

CAN bus, introduced by Robert Bosh in the 1980s, is a serial communication bus that uses the broadcast method to transmit messages across all devices and microcontrollers interconnected to it without the interaction of a host computer. In addition to the benefits from standardisation and simplification, introduction of the CAN bus protocol in the automobile industry has reduced the amount of wiring by two kilometres; thus, benefiting from significant reduction in car weight [1].

CAN protocol is a half-duplex system with two wires that can provide different data rates from 1 kbps to 1 Mbps for a bus length of 40 metres. However, when using bus length greater than 40 metres, the speed of the bus must be reduced; and in case the length of the bus is greater than thousand metres, specific drivers must be incorporated to keep the speed at the desired rate [2].

CAN communication protocol is a message base protocol that operates by using the media access control method called Carrier-Sense, Multi-Access with Collision Detection and Arbitration on Message Priority [3][4]. CAN bus protocol, being a message-based protocol, has a major advantage in that it allows additional nodes to be added to the bus independently from the nodes already connected to the bus, without the requirement of reprogramming all the nodes to recognise the additional one. The additional node will, therefore, receive messages from the network and will decide whether to use or discard the received data [5].

Furthermore, each node is embedded with error detection, signalisation and self-checking features to ensure reliability. They are able to determine fault conditions occurring to the bus and transit to different modes according to the level of severity of the fault. Moreover, the CAN protocol provides nodes with sophisticated error detection and error handling mechanisms, such as cyclic redundancy check (CRC) with high immunity against electromagnetic interference. It has the capacity to differentiate between temporary errors and permanent failure of other nodes in order to modify their functionalities accordingly [6].

The first CAN protocol was defined by the ISO standard as a standard 11 bit identifier that provided signalling rates from 125 kbps to 1 Mbps with an identifier field that provided for 211 or 2,048 different messages identifiers known as version 2.0A. This version has later been upgraded to version 2.0B by extending the 11-bit identifier to 29-bit identifier with an identifier field that provides for 2^{29} or 537 million identifiers [7]. Figure 1 illustrates both standards. The difference between both versions is that the extended version has a longer identifier length that the previous version as its identifier is made up of an 11-bit identifier plus an 18-bit extension identifier.

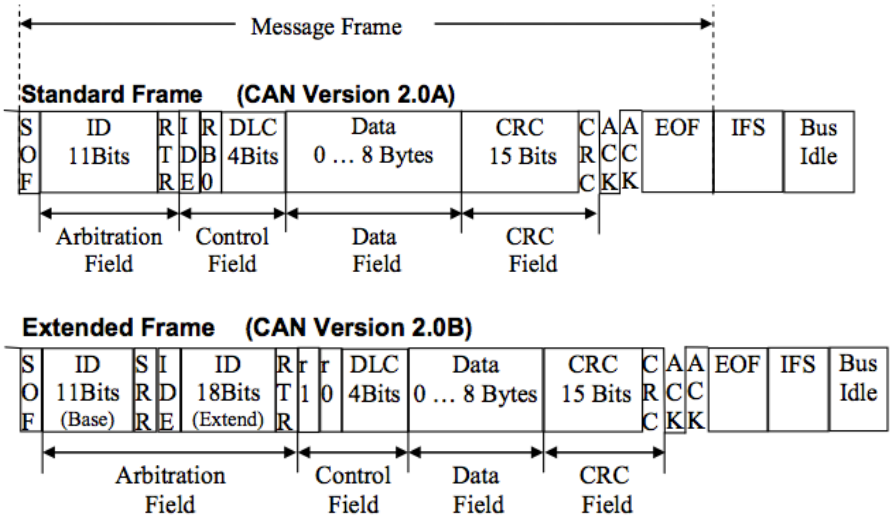

Figure 1: Extended CAN frame (Version 2.0A and 2.0B) [6].

A bit-wise arbitration mechanism using identifiers is used to determine the priority of messages in order to avoid data collision, when all nodes are sending data to the bus at the same time. 11 bits to 29 bits are the length usually associated with the identifier, while 0 to 8 bytes is the length given to the data [6][8]. However, to support the non-destructive bitwise arbitration, two setups are required. The first setup is the use of logic states required to determine the nodes states, which can either be dominant or recessive. The second setup is that the node transmitting must monitor the bus state. This will enable to see if the logic state in the transmission process is currently appearing on the bus. In regard to CAN bus, the binary logic is used to determine the state of the node. The logic *0* represents the dominant bit while the logic *1* represents the recessive bit [9].

There are four types of frames used to constitute CAN, as follows:

- Data frame: contains node data for the transmission;
- Remote frame: requesting data from a specific node;
- Error frame: error detection ensured by a node within the bus;
- Overload frame: resole delay between the data and/or the remote frame.

The data frame is the most important field of CAN frames and consists of seven subcategories that are: star of frame, arbitration field, control field, data field, CRC field, ACK field, End of Frame. Figure 2 illustrates the data frame and the remote frame.

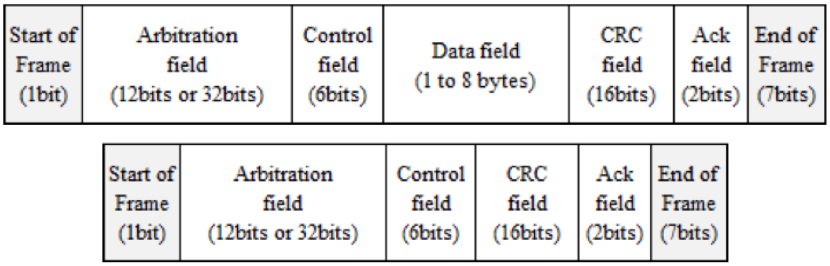

Figure 2: Data frame and remote frame [6].

Teaching CAN Bus

Due to its complexity and abstract nature, it is often a challenge to deliver the information described even at Master's level teaching, especially, if the learners do not have good grounding in computer science or programming experience; for example, students on mechanical or automotive courses. Thus, the experimental approach and graphical programming environment are expected to provide quick gains and better engagement of the students, whilst equipping them for similar work in industry based on real-time computing to create an embedded system with on-board FPGA. The proposed experiment was successfully developed by such a student as a part of his capstone project to be used in the next academic year for teaching embedded systems to automotive Master's level students.

STUDENT EXPERIENCE

The National Instruments (NI) LabVIEW graphical programming system was used as the design environment to develop the embedded in-vehicle network electronic system for radiator control. Digital, analogue input/output

industrial modules, including a specialised CAN bus module, were connected to a reconfigurable field programmable gate array (FPGA) chassis within a cRIO real-time controller. A graphic user interface (GUI) was created using LabVIEW programming environment, which supports establishing the communication between the two ports (CAN0 and CAN1) within the same 2-port CAN module: CAN0 port was used as the sender frame and CAN1 as the receiver. A write permission cluster with several input fields was used to constitute a message from CAN0 frame.

A similar cluster was used for CAN1, but in read only mode. The virtual instrument window called FPGA CAN bus. VI was, then, compiled and downloaded on the cRIO memory to be executed. This established the CAN bus communication on the real-time controller. A transmit button was used to trigger the communication between both ports; the data computed from CAN0 frame was transmitted and, then, received by CAN1 frame. The flowchart in Figure 3 illustrates the different sequences required from creating a message in CAN0 to displaying the message in CAN1, whilst Figure 4 illustrates the implementation program and associated GUI (LabVIEW front panel).

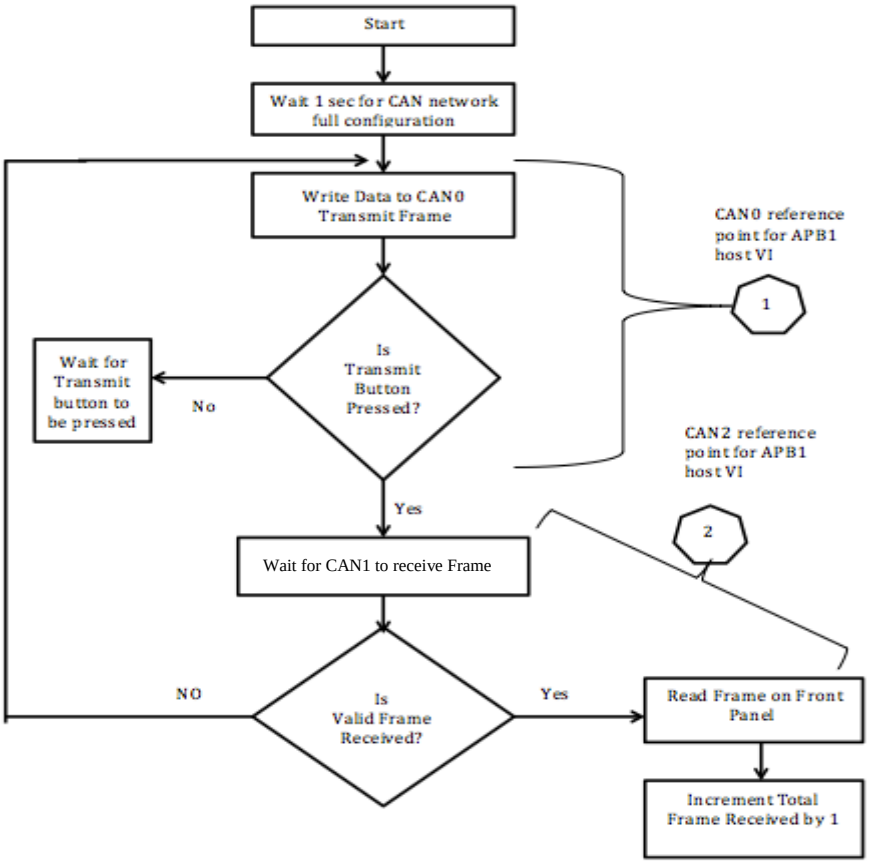

Figure 3: Transmit CAN frame was designed by creating a VI in the FPGA target.

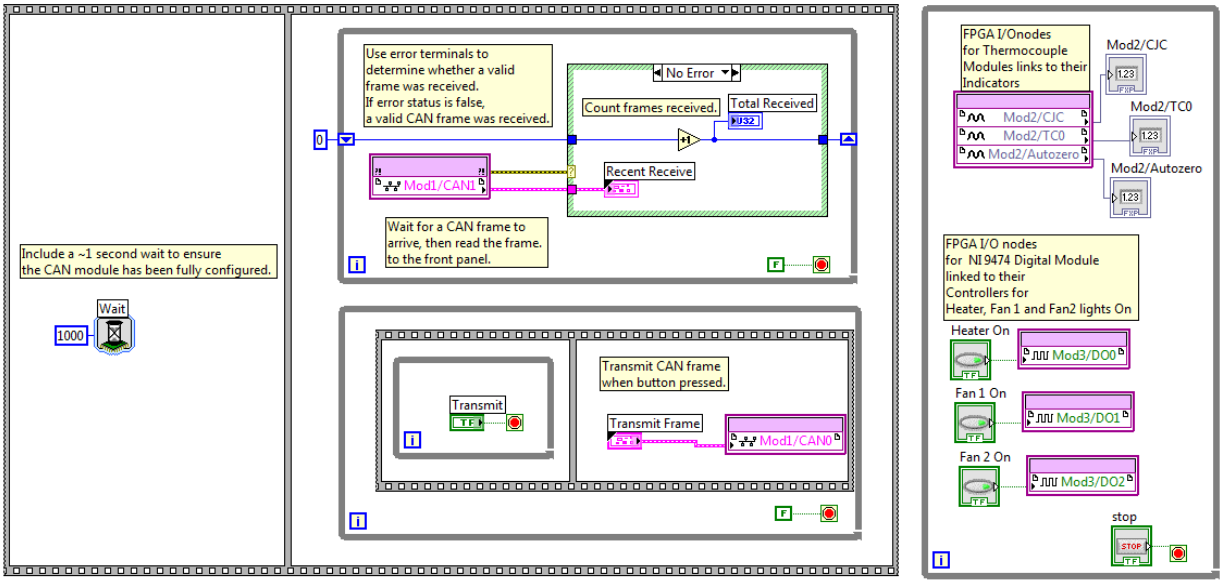

Figure 4: FPGA CAN bus VI - block diagram.

The front panel of the FPGA CAN bus VI is shown in Figure 5.

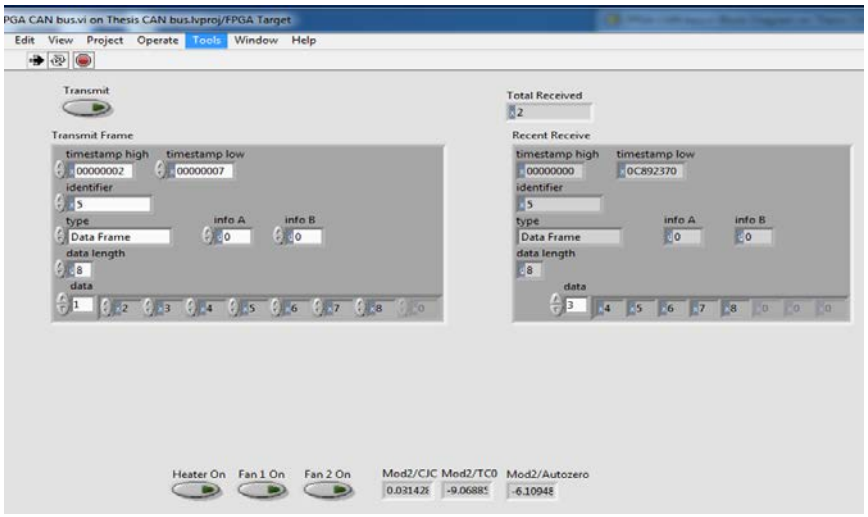

Figure 5: FPGA CAN bus VI - front panel.

Design of API1 (Host) VI

The output signal is a combination of both the linearised temperature and the *Transmit CAN Signal* value that was encapsulated within the array prior to the conversion from signal to Frame Raw. The linear temperature received is checked against the three predefined thresholds. The actuator rules implemented are as follow:

- Rule 1: if temperature sensed is less than or equal to 5°C, it automatically switches ON radiator built in heater.
- Rule 2: if temperature sensed is greater than 25°C, it automatically switches FAN1 ON.
- Rule 3: if temperature sensed is above 35°C, then, it automatically switches FAN1 and FAN2 ON.

The rules were implemented within an external virtual instrument window called application programming interface host (API1). The rules mentioned above are executed once the Read CAN Frame is complete. The application programming interface API1 (Host) VI flowchart in Figure 6 illustrates the different sequences required from acquiring the temperature and sending it from CAN0 to receiving it by CAN1 and implementing the rules accordingly. The hexagram 1 represents sender frame CAN0 from FPGA CAN bus VI and the hexagram 2 represents the receiver frame CAN1 imported into the application programming interface API1 (Host).

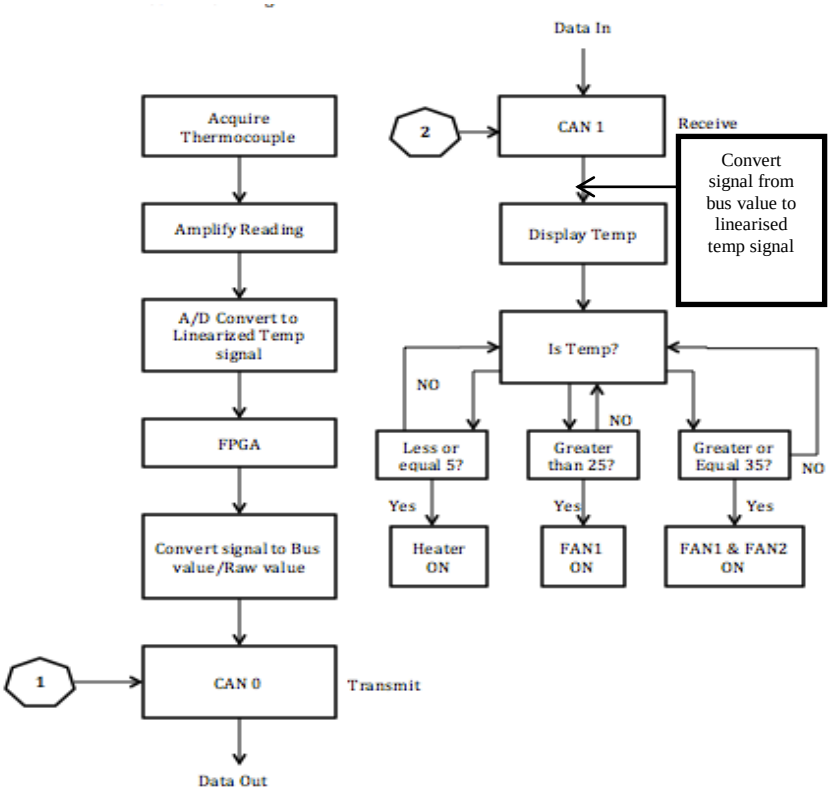

Figure 6: Flow diagram of API1 (Host) VI.

The implementation of API1 (Host) VI flow diagram in LabVIEW is represented in Figure 7 and Figure 8.

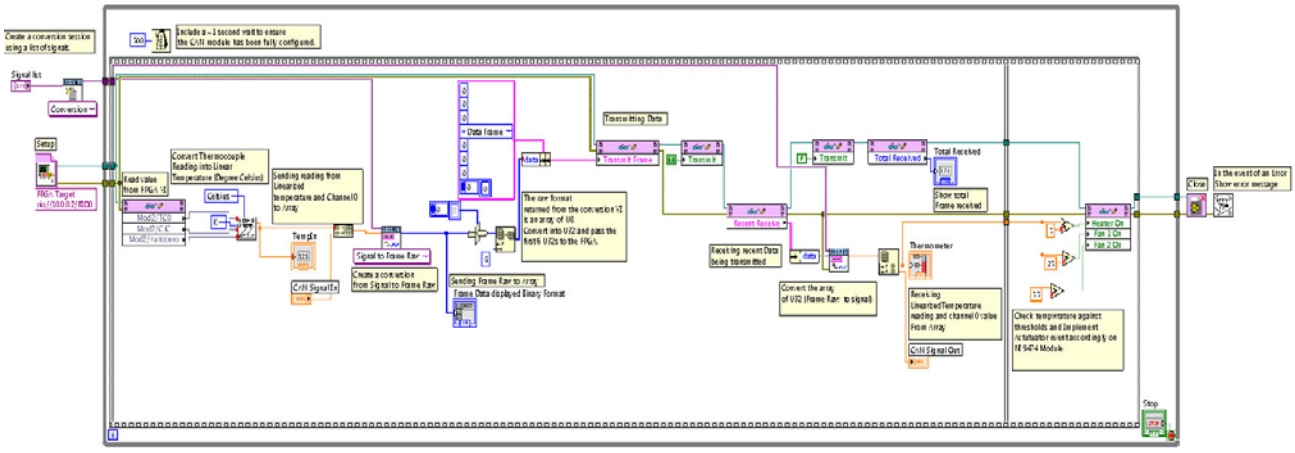

Figure 7: API1 (Host) VI block diagram.

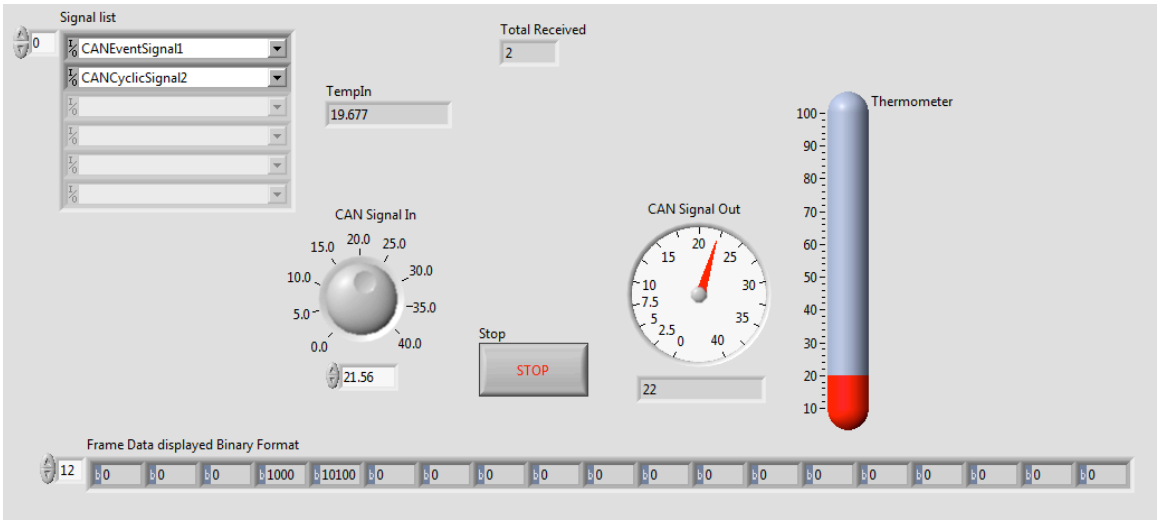

Figure 8: API1 (Host) VI front panel.

EXPERIMENTAL SETUP

The experiment objective is to create an electronic rapid prototyping network for sending and receiving CAN messages using NI-CRIO9014 and LabVIEW. Furthermore, the ERP system to be used for ECU incorporates real sensor data in the CAN message, which triggered actuator events accordingly.

The experiment setup was done using a Mac book Pro with Windows 7 installed, NI-LABVIEW 2012, NI cRIO 9014 real-time controller, 2-Port High-Speed CAN module (NI9853 with CAN1 powered by 5 V using an external power supply and a CAN cable made in-house), a thermocouple analogue input module (AI, NI 9211) and a high-speed sourcing digital output module (DO, NI 9474). To implement sensor and actuator events, ice and flame were used. Successful implementation and operation of NI DO module within electronic rapids prototyping system, triggers outputs after reaching temperature threshold sent via CAN bus.

CONCLUSION AND FUTURE WORK

This design approach, implementing a CAN bus communication within a rapid prototyping development platform, enables students to become familiar with rapid design, testing and validation of the algorithm. This approach provides conditions similar to that of a modern industrial environment where a proof of concept is prototyped prior to market production of the ECU to use the system. The current Master's level experiment using rapid prototyping was designed with flexibility to incorporate further sensors and actuators to meet today's needs with regard to the automotive industry. If recently released NI powertrain group new six series modules are added to the system, vehicle powertrain could also be monitored and transmitted from the engine control to other devices in the vehicle that requires this data [10].

Moreover, the ERP for ECU developed through this Master's student level experiment could also be beneficial to the KU Racing Team as the main communication means for their next year formula student car. It would enable real-time communication at high-speed and low-speed between the different ECUs, electronic devices, battery management

system (BMS), speed control system and anti-braking system (ABS) depending on the speed required. As the SAE Formula Student car rules allow the use of CAN bus for the accelerator pedal, this could be also implemented. Furthermore, the brake pedal, buzzer and a microcontroller linked to an LED indicating high voltage presence in the car could also be included, and all regrouped under a body control module using the electronic rapid prototyping system. Lastly, the use of on-board diagnostics (OBD) could also be connected to the system, which would provide state-of-health information of the car for the purpose of quickly identifying the nature of the problem.

REFERENCES

1. Kumar, B.P., Raj, V., Satyanaraya, B. and Sarma, D.N, Design of automatic automobile using CAN bus. *Inter. J. of Research in Computer and Communication Technol.*, 1, **7**, 478-482 (2012).
2. Furmanski, P. and Stolarski, Z., Controller area network implementation in microwave systems. *Proc. Inter. Conf. on Microwaves Radar and Wireless Communication*, 3, 869-873 (2002).
3. Goyal, U. and Khurana, G., Implementing MOD bus and CAN bus protocol conversion interface. *Inter. J. of Engng. Trends and Technol.*, 4, **4**, 630-635 (2013).
4. Saadi, I.M, Dynamic message transmission scheduling using can protocol. *Inter. J. of Science and Technol. Research*, 2, **9**, 158-163 (2013).
5. Pazul, K., Controller Area Network (CAN) Basics (1999), 13 January 2014, http://www.cl.cam.ac.uk/research/srg/han/Lambda/webdocs/an713.pdf
6. Provencher, H., *Controller Area Networks for Vehicles.* Hugo Provencher (2012).
7. Chougule, A.K. and Valiya, R.J., Milk dairy automation using CAN protocol: a paradigm for industry automation. *Inter. J. of Advanced Research in Computing Science and Software Engng.*, 3, **9**, 1187-1191 (2013).
8. Singh, V.K. and Archana, K., Implementation of *CAN* protocol in automobiles using advance embedded system. *Inter. J. of Engng. Trends and Technol.*, 4, **10**, 4422-4427 (2013).
9. Goyal, A. and Sharma, N., User driven feedback control system driven using CAN protocol. *Inter. J. of Electronics Communication and Computer Technol.*, 3, **5**, 468-471 (2013).
10. National Instrument, National Instruments Corporation (2013), 30 January 2014, http://www.ni.com/white-paper/4566/en/#toc5