MULTI-PROTOCOL GATEWAY FOR EMBEDDED SYSTEMS

B Abdul Rahim[1] and K Soundara Rajan[2]

[1]Department of Electronics & Communication Engineering,
Annamacharya Institute of Technology & Sciences, Rajampet, A.P, India
[2]Department of Electronics & Communication Engineering,
JNTUA College of Engineering, Anantapur A.P, India

ABSTRACT

The embedded systems are highly optimized to perform limited duties of particular needs. They can be control, Process, medical, signal, and image processing applications. The challenges faced by embedded systems are security, real-time, scalability, high availability and also performance based interoperability as more and more different devices are added to the systems. These complex ubiquitous systems are glued together with layers of protocols. Networking of these is a task to look for with minimum flaws in manageability, synchronization and consistency. We have attempted to design a gateway to interconnect UART with SPI, I2C and CAN Protocols. The design can be adopted for various embedded real-time applications and gives the flexibility of protocol selection.

KEYWORDS: *Real-Time Systems; Communication Protocols; Gateway and Embedded Systems.*

I. INTRODUCTION

Embedded systems perform limited duties as they are highly optimized for a particular need. More complex applications can be solved by embedded systems with the integration of different kinds of peripherals. The range of hardware used in embedded systems reaches from FPGAs to full blown desktop CPUs which are accompanied by special purpose ICs such as DSP Processors. On the software side, depending on the needs, everything from logic implementation to systems with own operating system and different applications running on it can be found. The grand challenge is design of integrated system architecture for ultra-reliable systems demanded by the society. Rechtin [1] defines ultra-reliability as a level of excellence so high that measuring it with confidence is close to impossible. Yet measurable or not, it must be achieved otherwise the system will be judged a failure.

The fast growth of electronic functions has led to many insular solutions that prevented comprehensive concepts from taking hold in the area of electrical/electronic architectures. Now a phase began with a marked development of electrical/electronic structures and associated networking topology from a comprehensive perspective. This meant that electrical/electronic content and its networking could claim an undisputed position in the complex systems. The recognition that many functions could only be implemented sensibly with the help of electronics also prevailed. So the image of electronics transformed from being a necessary evil to being a key to new, interesting and innovative functions. These functions must communicate with one another over a complex heterogeneous network. These networks typically contain multiple communication protocols including the industry standard Universal Asynchronous Receive/Transmit (UART), Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C), Controller Area Network (CAN), Local Interconnect Network (LIN), TTP/C and the recently developed FlexRay.

Previously chip-to-chip communications used many wires in a parallel interface, often ICs to have 24, 28, or more pins. Many of these pins were used for inter-chip addressing, selection, control, and data transfers. In a parallel interface, 8 data bits are typically transferred from a sender IC to receiver ICs in a single operation. The introduction of serial communication has led to reduction in real estate required on the board i.e., saving both cost and space.

The UART is a circuit that sends parallel data through a serial line. UARTs are frequently used in conjunction with the EIA (Electronic Industries Alliance) RS-232 standard, which specifies the

electrical, mechanical, functional, and procedural characteristics of two data communication equipments. Some interconnects require their own voltage levels and format of digital data like communication to some flash memories, EEPROM, sensors and actuators. The efficient protocol for the particular IC has to be used with interface.
The basic principle and format of protocols used in the gateway are presented in the next section. In the section 3 we describe the board over which the gateway is designed and the results obtained and finally in section 4 the paper is concluded.

II. ON-BOARD PROTOCOLS

The protocols used for making a gateway are discussed in brief about their principle and formats.

2.1. Universal Asynchronous Receive/Transmit (UART)

UART is used along with industry standard RS-232. Because of the voltage levels defined in RS-232 are different from that of IC I/O on the board, a voltage converter chip (MAX232) is needed between a serial port and an IC I/O pins as illustrated in figure 1.

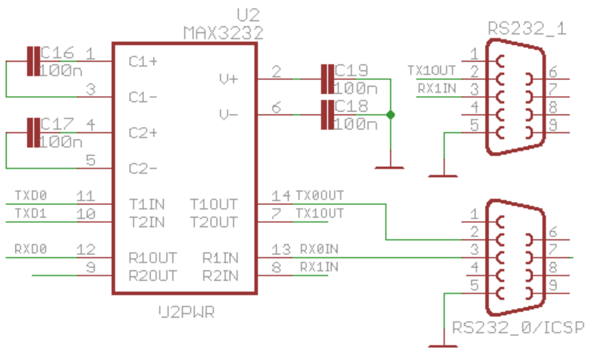

Figure. 1 Converter IC between RS232 and other ICs.

A UART includes a transmitter and a receiver. The basic functions of a UART are a microprocessor interface, double buffering of transmitter data, frame generation, parity generation, parallel to serial conversion, double buffering of receiver data, parity checking, and serial to parallel conversion. The frame format used by UARTs is a low start bit, 5-8 data bits, optional parity bit, and 1 or 2 stop bits. The frame format for data transmitted/received by a UART is given in Figure 2. No clock information is conveyed through the serial line. Before the transmission to start, the transmitter and receiver must agree on the set of parameters in advance, which include the baud rate, the number of data bits, stop bits, and the use of parity bit. The commonly used baud rates are 2400, 4800, 9600 and 19200 bauds. We should always have the same baud rates as in the PC and in the UART.
The baud rates are calculated as follows:

$$\text{Baud rate} = f_{PCLK1} / (16*BRR),\ BRR = f_{PCLK1} / (16*\text{Baud rate})$$

For example, in our application, we used 9600 as baud rate, and the f_{PCLK1} is 8 MHz.

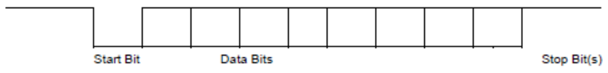

Figure 2 frame format for UART data

2.2. Serial Peripheral Interface (SPI)

So, what is SPI? SPI is a very simple serial data protocol. This means that bytes are send serially instead of in parallel. SPI is a standard protocol that is used mainly in typical embedded systems. It

falls in the same family as I2C or RS232. SPI is primarily used between micro-controllers and their immediate peripheral devices. It's commonly found in cell phones, PDA's, and other mobile devices to communicate data between the CPU, keyboard, display, and memory chips.
The SPI (Serial Peripheral Interface)-bus is a master/slave, 4-wire serial communication bus. The four signals are clock (SCLK), master output/slave input (MOSI), master input/slave output (MISO), and slave select (SS). Whenever two devices communicate, one is referred to as the "master" and the other as the "slave". The master drives the serial clock. Data is simultaneously transmitted and received, making it a full-duplex protocol. Rather than having unique addresses for each device on the bus, SPI uses the SS line to specify which device data is being transferred to or from. As such, each unique device on the bus needs its own SS signal from the master. If there are 3 slave devices, there should be 3 SS leads from the master, one to each slave as shown in Figure 3.

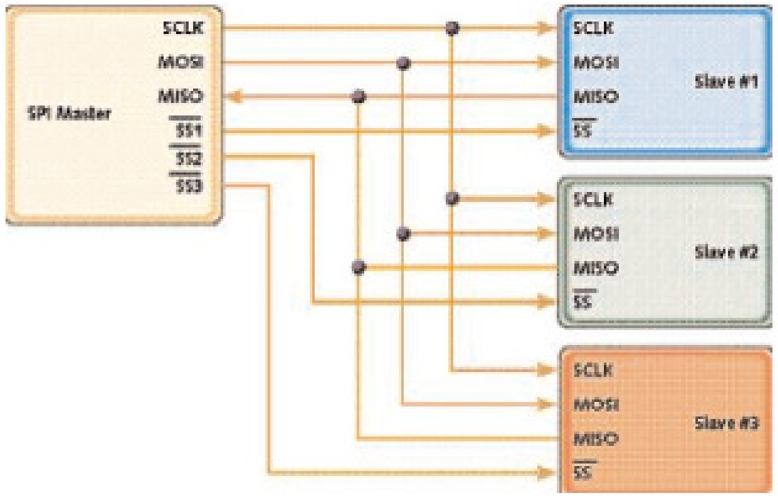

Figure 3: common SPI configuration

This means there is one master, while the number of slaves is limited by the number of chip select lines. When an SPI data transfer occurs, an 8-bit data word is shifted out of MOSI while a different 8-bit data word is being shifted in on MISO. This can be viewed as a 16-bit circular shift register. When a transfer occurs, this 16-bit shift register has shifted 8 positions, thus exchanging the 8-bit data between the master and slave devices. A pair of registers, clock polarity (CPOL) and clock phase (CPHA) determine the edges of the clock on which the data is driven. Each register has two possible states which allows for four possible combinations, all of which are incompatible with one another. So a master/slave pair must use the same parameter values to communicate. If multiple slaves are used that are fixed in different configurations, the master will have to reconfigure itself each time it needs to communicate with a different slave [2].

2.3. Inter-Integrated Circuit (I2C)

Inter-Integrated Circuit (I2C) bus provides good support for communication with various slow, on-board peripheral devices that are accessed intermittently, while being extremely modest in its hardware resource needs. It is a simple, low-bandwidth, short-distance protocol. I2C is easy to use for linking multiple devices together since it has a built-in addressing scheme. Philips originally developed I2C for communication between the devices inside of a TV set. Examples of simple I2C-compatible devices found in embedded systems include EEPROMs, thermal sensors, and real-time clocks. I2C is also used as a control interface to signal processing devices that have separate, application-specific data interfaces. For instance, it's commonly used in multimedia applications, where typical devices include RF tuners, video decoders and encoders, and audio processors. In all, Philips, National Semiconductor, Xicor, Siemens, and other manufacturers offer hundreds of I2C-compatible devices [3].

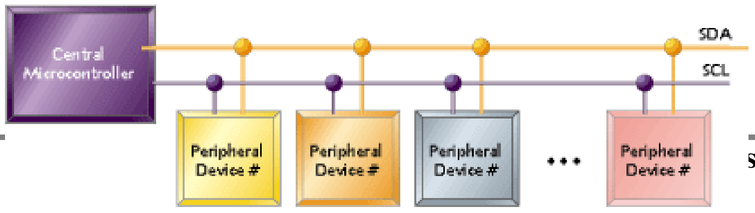

Figure 4: I2C is a two-wire serial bus

The I2C bus uses a bi-directional Serial Clock Line (SCL) and Serial Data Lines (SDA) as shown in figure 4. Both lines are pulled high via a resistor (Rp)(see figure 5). Resistor Rs is optional, and used for ESD protection for 'Hot-Swap' devices. Three speed modes are specified: Standard; 100kbps, Fast mode; 400kbps, High speed mode 3.4Mbps. I2C, due to its two-wire nature (one clock, one data) can only communicate in half-duplex mode. The maximum bus capacitance is 400pF, which sets the maximum number of devices on the bus and the maximum line length. The interface uses 8 bit long bytes, MSB (Most Significant Bit) first, with each device having a unique address. Any device may be a Transmitter or Receiver, and a Master or Slave. Data and clock are sent from the Master; data is valid while the clock line is high. The link may have multiple Masters and Slaves on the bus, but only one Master may be active at any one time. Slaves may receive or transmit data to the Master. V_{DD} may be different for each device, but all devices have to relate their output levels to the voltage produced by the pull-up resistors (R_P).

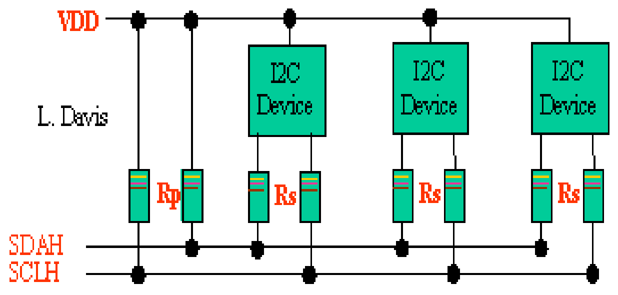

Figure 5: I2C Circuit

As you can see in Figure 6, the master begins the communication by issuing the start condition (S). The master continues by sending a unique 7-bit slave device address, with the most significant bit (MSB) first. The eighth bit after the start, read/not-write (R/ώ), specifies whether the slave is now to receive (0) or to transmit (1). This is followed by an ACK bit issued by the receiver, acknowledging receipt of the previous byte. Then the transmitter (slave or master, as indicated by the bit) transmits a byte of data starting with the MSB. At the end of the byte, the receiver (whether master or slave) issues a new ACK bit. This 9-bit pattern is repeated if more bytes need to be transmitted.

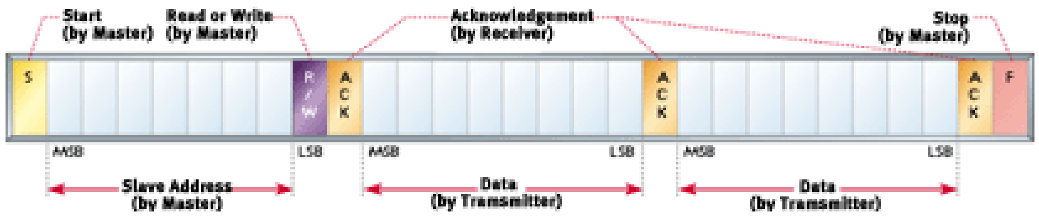

Figure 6: I2C's communication format

In a write transaction (slave receiving), when the master is done transmitting all of the data bytes it wants to send, it monitors the last ACK and then issues the stop condition (P). In a read transaction (slave transmitting), the master does not acknowledge the final byte it receives. This tells the slave that its transmission is done. The master then issues the stop condition. The I2C signaling protocol provides device addressing, a read/write flag, and a simple acknowledgement mechanism. There are a few more elements to the I2C protocol, such as general call (broadcast) and 10-bit extended addressing. Beyond that, each device defines its own command interface or address-indexing scheme. Most often, the I2C master is the CPU or microcontroller in the system. Some microcontrollers even feature hardware to implement the I2C protocol [4].

2.4. Controller Area Network (CAN)

In the mid–1980s, the third party supplier Bosch developed the Controller Area Network (CAN), It was first integrated in Mercedes production cars in the early 1990s. Today, it has become the most widely used network in automotive systems and it is estimated [5] that the number of CAN nodes sold per year is currently around 400 million (all application fields). Today almost every automobile manufacturer uses CAN controllers and networks to control devices such as: windshield wiper motor controllers, rain sensors, airbags, door locks, engine timing controls, anti-lock braking systems, power train controls and electric windows, to name a few. Due to its electrical noise tolerance, minimal wiring, excellent error detection capabilities and high speed data transfer, CAN is rapidly expanding into other applications such as industrial control, marine, medical, aerospace and more.
The CAN bus is a balanced (differential) 2-wire interface running over a Shielded Twisted Pair (STP), Un-shielded Twisted Pair (UTP), or ribbon cable. Each node uses a Male 9-pin D connector. Non Return to Zero (NRZ) bit encoding is used with bit stuffing to ensure compact messages with a minimum number of transitions and high noise immunity. The CAN Bus interface uses an asynchronous transmission scheme where any node may begin transmitting anytime the bus is free. Messages are broadcast to all nodes on the network. In cases where multiple nodes initiate messages at the same time, bitwise arbitration is used to determine which message is of higher priority.
The standard CAN data frame can contain up to 8 bytes of data for an overall size of, at most, 135bits, including all the protocol overheads such as the stuff bits as shown in figure below.

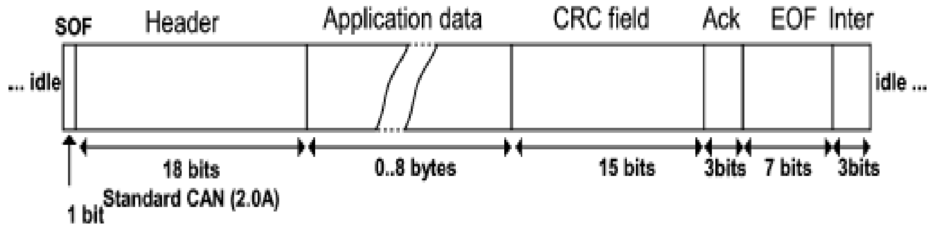

Figure 7: format of CAN data frame

The sections of the frames are:

- The header field , which contains the identifier of the frame, the remote transmission request (RTR) bit that distinguishes between data frame (RTR set to zero) and data request frame (RTR set to 1) and the data length code (DLC) used to inform of the number of bytes of the data field.
- The data field, having a maximum length of 8 Bytes.
- The 15-bit cyclic redundancy check (CRC) field, which ensures the integrity of the data transmitted.
- The Acknowledgment field (Ack). On CAN, the acknowledgment scheme solely enables the sender to know that at least one station, but not necessarily the intended recipient, has received the frame correctly.
- The end-of-frame (EOF) field and the intermission frame space, which is the minimum number of bits separating consecutive messages.

In CAN, number of different data rates is defined, with 1Mb/s being the fastest, and 5kb/s the slowest. All modules must support at least 20kb/s. Cable length depends on the data rate used. Normally, all devices in system transfer information at uniform and fixed bit rates. The maximum line length can be thousands of meters at low speeds; 40 meters at 1Mbps is typical. Termination resistors are used at each end of the cable [6].

III. THE GATEWAY

Real-time applications are typically more difficult to design than non-real-time applications. Real-time applications cover a wide range, but most real-time systems are *embedded*. Small systems of low complexity are designed with loops that calls modules to perform the desired functions/operations. Interrupt service routines (ISR) handle asynchronous events and critical operations must be performed by ISRs to ensure that they are dealt with in a timely fashion. Because the execution time of typical code is not constant, the time for successive passes through a portion of the loop is nondeterministic. Furthermore, if a code change is made, the timing of loop is affected [7].
As different protocols have their own advantages and disadvantages to reckon with, the attempt has been made to define a gateway which will suffice the need for a particular system using components suitable to it [8]. The design is implemented and tested by using ARM7 RISC processor. The ARM7 board consist of two CAN nodes, a SPI and an I2C node. The data is fed through the keyboard via PS2 port or through the 'hyper terminal' (UART) and the respective data is displayed on the LCD which is communicating through I2C protocol. However, the available on-chip communication ports of ARM7 are utilized. The block schematic of the design is shown in figure 8.

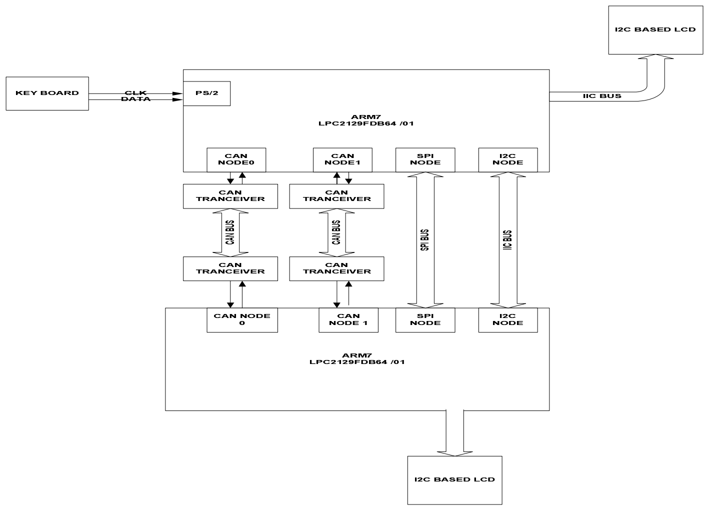

Figure 8: Block schematic of the gateway design

The MCP2551 CAN transceiver is used to serve as interface between a CAN node and the physical bus. The data to be transferred is first loaded into a wrapper, a memory, (LPC2129 16KB SRAM is used), then this is loaded into the data register and the data to be transferred through a particular protocol is selected. The data should be transferred in the format of desired protocol so the frame generator will attach the data in that frame and in the mean time baud rate synchronisation is taken care. For simplicity the baud rate chosen here is 9600. The whole frame is broken down into bytes and then transmitted serially. When frame transmission is complete the receiver takes an appropriate action for checking, analysing and acknowledging the receipt. The data received is stored in message RAM for analysis, in which three control signals are checked for frame start time, ready for reading and end of the frame data. Once the frame reception in complete the mode of frame analysis is changed to write from read. In the second phase the frames are read out of the desired protocol port. For transmission through that port the data can be placed in the frame format of the desired node.

IV. RESULTS AND DISCUSSION

The figures 9, 10, 11 and 12 are the snap shots describing the connections made and the results obtained for communication through I2C, SPI and CAN respectively. The figure 9 is setup for connection to the hyper terminal and the figure 10 is the connection of I2C nodes and the data to be transmitted is displayed on the two row LCD display.

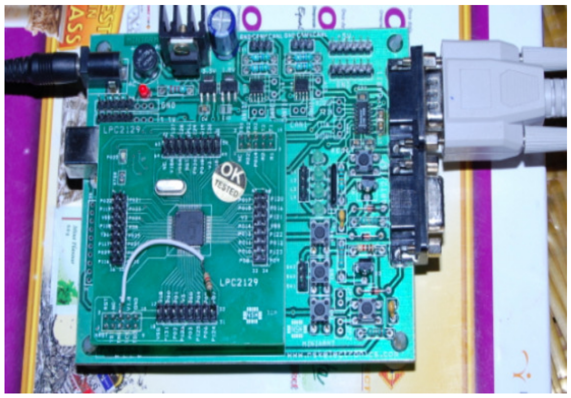
Figure 9: board connections to the UART

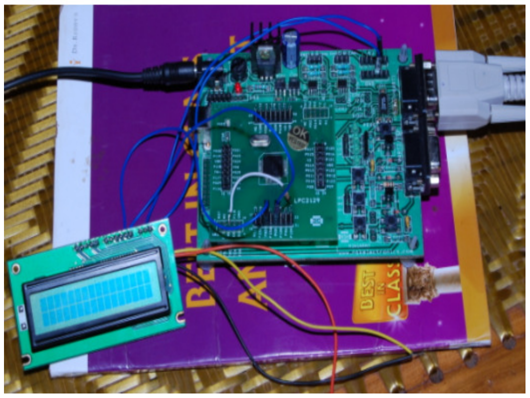
Figure 10: Communication through I2C

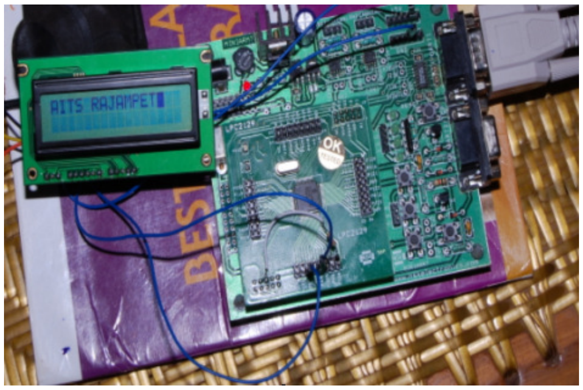

Figure 11: Communication through SPI

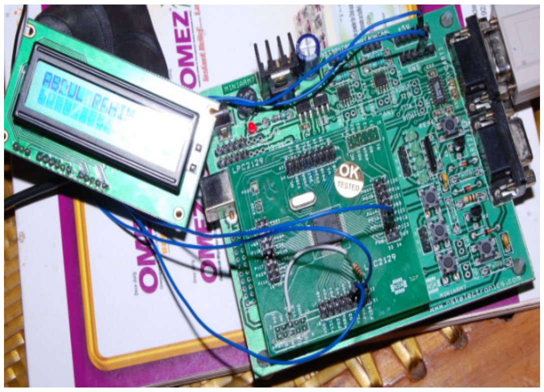

Figure 12: Communication through CAN Bus

The figure 11 shows the connection to SPI node and the data transfer is displayed “AITS RAJAMPET” which was typed in PC, transferred through hyper terminal and from the board 1 to board 2, the transmission is through SPI. Similarly the figure 12 shows the communication through CAN bus and the data transferred “ABDUL RAHIM” is displayed. The selection procedure can be GUI based or in the switch modes.

V. CONCLUSIONS AND FUTURE SCOPE

The multi-protocol integration for an embedded system is developed and tested. The protocols have been serial communicating as they are regularly used in embedded boards. UART is tested by interfacing the main embedded board with the computer. I2C drivers are developed to read the RTC and displayed on LCD. The SPI drivers were developed to interface memory in which the data typed is stored. As this is interfacing the devices of smaller size, power or low I/O count, makes application in the portable systems [9]. And lastly CAN drivers are developed and tested for data transfer from one transceiver to another. The gateway is developed and tested for trans-communication between UART to UART or I2C or SPI or CAN.

The gateway is very useful in communicating between one protocol to another protocol in a heterogeneous systems as the embedded systems are. For example mobile phones, the data typed by using one protocol and transmit & displayed on the LCD this is another protocol. The protocols

selected for implementation are event triggered and they are non deterministic used in non-critical applications. For Safety critical applications like brake-by-wire, steer-by-wire, etc., these protocols are inefficient and hence time-triggered protocols should be used (like, TTP/C, Flexray etc). In future I look forward to implementation with these protocols also.
The known difficulty in time triggered protocols being the clock synchronization of the nodes used as well as the description in scheduling the process in deterministic approach requires more stable clock. Since most of the time triggered protocols adopt TDMA technique the added components increase the size and cost of implementation [10]. The time triggered protocols are designed for hard real-time embedded systems, hence strict designed accuracy is required as compared to the on designed above, which is basically for soft real-time embedded systems.

ACKNOWLEDGEMENTS

We are thankful to Mr. S Narayana Raju, of Atmel R&D (India) Pvt. Ltd, Chennai for the contributions during the programming and development of the board.

REFERENCES

[1] E. Rechtin, "*systems Architecting, Creating and Building Complex Systems*," 2nd ed., Englewood Cliffs, Prentice Hall, 1991.
[2] David Kalinsky and Roee Kalinsky, "*Introduction to Serial Peripheral Interface,"* Embedded Systems Programming, 02/01/2002.
[3] D. Paret and C. Fenger, *The I2C Bus: From Theory to Practice.* John Wiley, 1997.
[4] Phillips Semiconductor, *The I2C-Bus Specification,* version 2.0, Phillips Semiconductor, Dec. 1998.
[5] K. Johansson, M. Torngren, and L. Nielson, *Handbook of Networked and Embedded Control Systems,* Birkhauser, 2005.
[6] Navet et. Al, *"Trends in Automotive Communication Systems"* in Proceedings of the IEEE, vol. 93, No. 6, June 2005.
[7] J J Labrosse, *Embedded Systems Building Blocks*, CMP Books, 2nd Ed, 2005.
[8] B. Abdul Rahim and Dr. K. Soundara Rajan, " *A Gateway to Integrate Communication Protocols of Automotive Electronics"* , Proc of First Intl Conf on Emerging Technologies & Applications in Engineering, Tech & Sciences (ICETAETS), Rajkot, Gujarat, 13-14 Jan 2008, pp 2357-2362.
[9] UART-to-SPI Interface, Application Note AC327, Actel Corp.2009.
[10] B Abdul Rahim and Dr. K Soundara Rajan , *"Fault Tolerance in Real-Time Systems through Time-Triggered Approach"*, CiiT International Journal of Digital Signal Processing, Vol 3. No. 3, April 2011, PP 115-120.

Authors Biographies

B Abdul Rahim born in Guntakal, A.P, India in 1969. He received the B.E in Electronics & Communication Engineering from Gulbarga University in 1990. M.Tech (Digital Systems & Computer Electronics) from Jawaharlal Nehru Technological University in 2004. He is currently pursuing Ph.D degree from JNT University, Anantapur. He has published papers in international journals and conferences. He is a member of professional bodies like EIE, ISTE, IACSIT, IAENG etc,. His research interests include Fault Tolerant Systems, Embedded Systems and Parallel processing.

K Soundara Rajan born in Tirupathi, A.P, India in 1953. He received the B.Tech in Electronics & Communication Engineering from Sri Venkateswara University. M.Tech (Instrumentation & Control) from Jawaharlal Nehru Technological University in 1972. Ph.D degree from University of Roorkee, U.P. He has published papers in international journals and conferences. He is a member of professional bodies like NAFEN, ISTE, IAENG etc,. He has vast experience as academician, administrator and philanthropist. He is reviewer for number of journals. His research interests include Fault Tolerant Design, Embedded Systems and signal processing.