ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 4, July 2014

Design of a high level protocol for Serial Peripheral Interface and its implementation in a CAN controller IC

Shilpa Mayannavar, Ashwini Desai, Dr. U. V. Wali

Abstract — Serial Protocol Interface (SPI), defined by Motorola is a commonly used inter-device communication protocol. SPI defines the lower two layers of ISO-OSI network model, namely physical and data link layer. In SPI, Source, destination and clock are defined by wiring. Hence the data frame does not include any information about the address of data transmission. When the output of SPI needs to be distributed among multiple destinations within a device or multiple devices, defining a protocol for data exchange will help reduce design time. In this paper, we have implemented one such protocol for use in a CAN-bus controller design. We define all the registers required by CAN processor as RAM. The protocol allows us to access these registers programmatically, using an 8-bit instruction set, exchanged over an SPI connection. The technique presented here can be used in any controller design using SPI bus or extended to other serial interfaces. The protocol provides a structural approach to the application layer design while using SPI. Such an approach is expected to help designers to build libraries for communication and control equipment which use SPI bus. The approach used to design the protocol can also be adapted to other design problems.

***Index Terms*—CAN (Controller Area Network), ISO-OSI (International Standard of Organization- Open System Interconnect), SPI (Serial Peripheral Interface).**

I. INTRODUCTION

ISO-OSI network model defines 7 layers of communication, viz., physical, data link, network, transport, session, presentation and application layer. The SPI standard defines only the bottom two layers of this network model. In SPI, the data is exchanged between two devices. It can be daisy-chained so that the data is exchanged between more than two devices as shown in figures 1a and figure 1b respectively.[2]

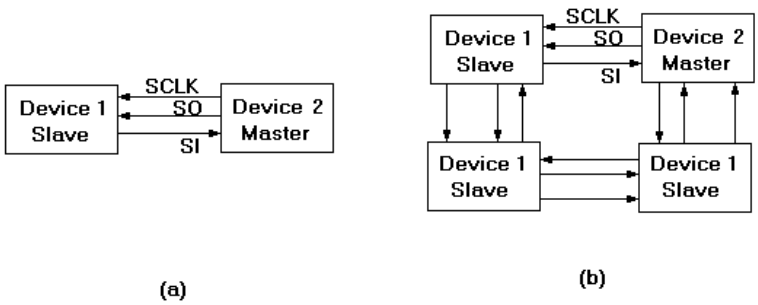

Fig 1: Typical SPI configurations: (a) Peer-to-Peer, (b) Loop

Commonly used configuration shown in the figure 1a, allows exchange of data between two devices. Depending on the capabilities of the devices, either of the communicating devices can be a master, but master should always provide the clock to the slave. In figure1b, the data is exchanged between four devices, device1, device2, device3 and device4, connected in a loop topology. SPI only defines the physical and data link layer of ISO-OSI network model. The connection and media between the devices indicates the physical layer. Data link layer defines the way in which the devices are connected. The connections include the input and output lines such as clock, data in and data out. Data link layer in SPI is implied in the connection itself and no provision is made for location or address information. In the simplest form, two computers exchange data among themselves. In contrast with other serial protocols, communication on SPI is always an exchange of data and not read or write; in other words, read and write occur simultaneously. SPI is a master slave network and hence there are no collisions, arbitration issues, or packet loss, which needs to be handled in other device communication networks.[1]

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 4, July 2014

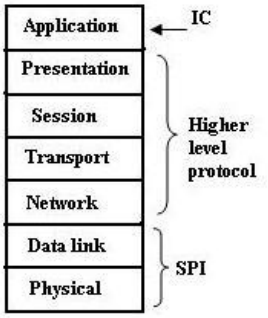

Fig 2: Definition of SPI in ISO-OSI layers

CAN-bus on the other hand defines an explicit packet addressing, arbitration and collision detection mechanism; it needs one because it is a multi-master bus. Most of the equipment manufacturers have developed their own higher level protocols, e.g. CAL (CAN Application Layer) of Philips, or use more widely accepted standards like CAN open.

Network, transport and other higher levels are not addressed in CAN because it is expected to be used within a single equipment or machine. This is not only true of CAN but most of the field bus designs. Communication between devices was not expected to involve large number of devices and hence, the network layer is hardly addressed in most of the device communication systems. This will change over next decade or so as Internet-of-things (IOT) is gaining momentum. Session and presentation layers are also currently not handled in most of the device communication devices for similar reasons.

SPI is expected to be a 'super' synchronous communication system: Data transfers occur on clock transition. The packet sequences are also expected to happen in a specified sequence as there is no explicit frame address contained within the frame. On one hand, this open design gives flexibility but on the other hand, this becomes a limitation when developing standard device libraries etc. Absence of such libraries leads to longer development time.

SPI uses three I/O lines, viz., SPIClk, SI and SO, for clock, input and output respectively. SI is a serial input into the device, SO is the serial output of the device and SPIClk is the clock driven by the master. SI of one device is connected to SO of other device, and vice-versa, forming a loop. Hence, when SPIClk is triggered, read and write occur simultaneously within each device. Hence SPI represents a two-wire, full duplex communication protocol, where in data is exchanged instead isolated read/write operations. SPI also allows data transfer to be 'mutually agreed' to occur on rising edge or falling edge (called phase, CPHA). Master can also start transfer on high-active or low-active clock (called polarity, CPOL). Combining the phase and polarity, four different modes can be supported by SPI.

SPI defines only how a byte exchange takes place between the master and slave. However, there is no fixed protocol to understand the content of the exchanged data. In typical telemetry applications, a single data type is exchanged over such an interface. However, in a generic case, there may be many types of data that need to be exchanged between the communicating devices. For example, when SPI is used to interface a CAN controller, microcontroller, acting as a master needs to send specific instructions to the CAN controller to configure which packets need to be read from or written to the CAN-bus. These are generally called Acceptance Filters (AF) and Address mask (AM). Specific memory has to be provided to buffer the data to be transmitted as well as to store received data. Other registers may also be required to store various control information.

As the SPI standard is open about these problems, the responsibility of data recognition, classification and storage, lies with the equipment designer. When several equipments are designed to communicate using a SPI interface, open design, inter-operability becomes a critical issue. Hence, there is a need to standardize some procedures to ensure portability of such equipment.

Our current work attempts to standardize the development process when using SPI. In this paper, we discuss the

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 4, July 2014

general approach and demonstrate a specific implementation using that standard. The technique has been used in designing a re-useable soft IP core.

II. METHODOLOGY

A. Overview of the protocol

The protocol allows data transfer in sets of bytes called **Byte Frames**. Each frame has a header and payload. Header can be one or two bytes depending on the end application. If the application needs short and full address, length of the header should be specified in the first byte. First byte is therefore called Instruction.[2]

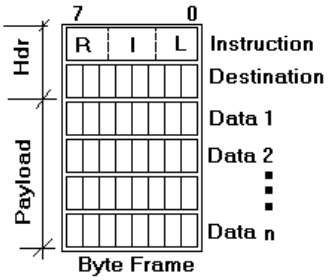

Fig 3: Frame Format

For short addresses, the instruction itself may contain address details. Otherwise, the second byte contains the destination register address. Up to sixteen registers can be accessed using one byte instruction, which is sufficient for most applications. However, when more registers (or devices or ports) are involved, we need to use the two-byte header format. The instruction byte essentially contains three components: Header length L, instruction (op-code) I and destination register(s) R. There is no specific limit on number of bits allocated for these sub-fields or their exact location. Second byte, when used is an unsigned 8 bit integer, indicating an address offset into a register bank. Alternate formats may be required to address specific memory organization when SPI is used to access multiple devices. For example, if SPI communication is used to control multiple CAN-bus controllers, each such controller will have its own set of registers. Even when all such registers are located on a single block of RAM, memory holes may exist. We will need specific address resolution strategies to handle such situations. The example discussed in later part of this paper will present one such situation.

Format for the data bytes is unconstrained. Applications can use any suitable format. Sometimes, it may be a simple ASCII byte set or may represent a floating point number. The number of bytes is usually fixed by the instruction itself.

For example, if the instruction is to receive a floating point number, the following steps are implied:

a) SPI buffer must be cleared before next byte of data is received. Otherwise, existing data in the SPI buffer may be lost. When the instruction is received, the SPI buffer contains the instruction itself and hence needs to be moved to a different location before the next SPI clock (if needed later). Hence, the controller must work at a clock ratio that allows internal transfers to be completed in fraction of SPI clock.

b) 4 Bytes of data will be received in the payload. Each of these bytes needs to be transferred one byte at a time, to a register that can be addressed as a floating point number or as byte array.

There is also a need to address special situations that arise when only a part of the frame is received. If the trailing part of the frame is lost, error should be flagged after appropriate timeout. The situation is more critical if the leading part of the frame is lost. In such a case, there is a chance that a data byte is interpreted as instruction. This could lead to a catastrophic failure of the system as wrong data may be transferred to destination registers.

B. Instruction format for CAN Controller

As an example implementation, we will present the protocol implementation for a CAN controller module. Figure 4 gives a design overview. [3]

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 4, July 2014

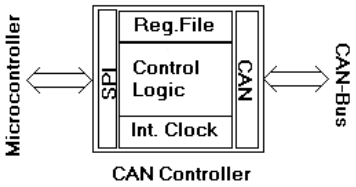

Fig 4: CAN Controller

The module consists of an internal clock generator, a register file, modules for interfacing SPI and CAN-bus. Control logic glues everything together. We have discussed the details of SPI interface here. The CAN controller is used to transfer data between CAN-bus and the microcontroller. When a message is received on the CAN-bus, control circuit buffers it in the register file. An interrupt is generated by the CAN controller. On receiving the interrupt, the microcontroller sends a message on the SPI port to read the contents of the CAN message. CAN controller decodes the request (1 byte) and serializes the data in register file to the SPI buffer. The sequence is reversed when the microcontroller wishes to put some data on the CAN-bus. Microcontroller is SPI master and hence can start the SPI messaging sequence. It transfers the data to be transmitted via SPI interface to the CAN controller. The CAN controller now has to decipher the instruction and transfer the contents of SPI bus to the register file. The CAN transmitter module then picks up the message and puts it on the CAN-bus.

CAN-controller will receive all messages arriving on the CAN-bus. However, this generates too many interrupts to the microcontroller. In order to avoid unwanted interrupts, Microcontroller can setup the CAN controller to receive a limited set of CAN messages. CAN controller uses special registers, generally called Acceptance Filters to filter the incoming messages. Microcontroller has to send specific instructions to CAN controller via the SPI interface to set the acceptance filters.

The instruction format for the SPI module of CAN controller is designed with this data flow in mind. Essentially, there are three types of operations to be performed by the SPI frame: i) Set acceptance filters, ii) Write a message to the CAN-bus and iii) Read messages from CAN-bus. We will need at least two bits to represent these three operations. We have chosen 3 bits: bit 0 for setting the controller, bit 2 for reading and bit 3 for writing. We can write this set as Boolean expression: opcode = 10x0 + 01x0 + 00x1. We have used only three of the possible 8 instructions.

We have used two register banks, i.e, two acceptance filters, two read buffers and two write buffers. Every instruction can access either of these banks: Bit 7 is used to identify the register bank: 1 for higher bank and 0 for lower bank.

CAN message consist of two bytes (11 bits) of device address and up to 8 bytes of data. There may be no data associated with some addresses. Totally, we need 10 bytes of space to store a message (payload). However, all messages will have the first two bytes and hence may not be included in the count. We only need to use 9 possible data sizes – 0 to 8. This requires 4 bits of address space. We have used bits 6 down to bit 4 (3 bits) to indicate length of data from 0 to 7. To identify full CAN message (8 bytes of data + 2 bytes of CAN address), we use bit 1.

Combining these features, we get the following structure for SPI frame instruction. Note that because of small number of registers, we do not need the second byte of header (see fig. 3). The format for instruction byte is shown in figure 5.[4] Description of each bit is given in the Table 1.[4]

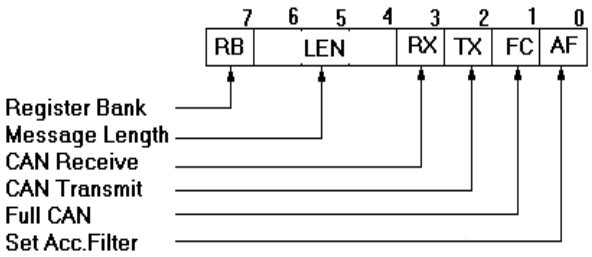

Fig 5: Instruction format

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 4, July 2014

Note that we have tried to maintain the instruction and address orthogonal to each other. For example, RB=1 always refers to upper register bank, irrespective of the instruction. The instruction set is given in Table1.

Table1: Micro-instruction set

Binary Format	Instruction	Register file (Data Addr)	Description
0xxx00x1	Set Register	AFL	Set Acceptance filter (lower bank)
1xxx00x1	Set Register	AFH	Set Acceptance filter (upper bank)
0xxx01x0	CAN Transmit	TXL	Transmit (write) a message to CAN-bus (lower bank)
1xxx01x0	CAN Transmit	TXH	Transmit (write) a message to CAN-bus (upper bank)
0xxx10x0	CAN Receive	RXL	Receive (read) a message from CAN-bus (lower bank)
1xxx10x0	CAN Receive	RXH	Receive (read) a message from CAN-bus (upper bank)

C. State transition diagram for the micro-instruction set

There are mainly 23 states which include idle, AF_L, AF_H, TX_L, TX_H, RX_L, RX_H for the CAN address in each case and from TX_M1 to TX_M8, from RX_M1 to RX_M8 for the 8 byte of message. Therefore, 5 bit register called state is used which will be varying from 00(H) to 18(H). Depending on the instruction the state transition occurs. When 8 bits are exchanged on SPI, spiBufFull is set. This flag is cleared on receiving Clrspi. Bits [6:4, 1] contain frame length information in msgLen buffer. The state transitions for AF, TX, RX instructions are shown in Figure 6 (a), (b) and (c) respectively.[4]

During Idle state, if AF* instruction is received, two more bytes containing CAN address filters are expected. The first byte contains lower 8 bits and higher byte contains upper 3 bits. On receiving AF instruction, the following state transits occur in next two buffer exchanges on SPI: (a) the state transits from AF_L to AF_H on the spiBufFull signal, and the address is transferred to AFLn register, (b) on the next spiBufFull signal, buffer is copied to AFHn register and the state transits to idle. Register names use n to indicate register bank. For example, AFLn could refer to AFL0 or AFL1, corresponding to bank 0 or bank 1.

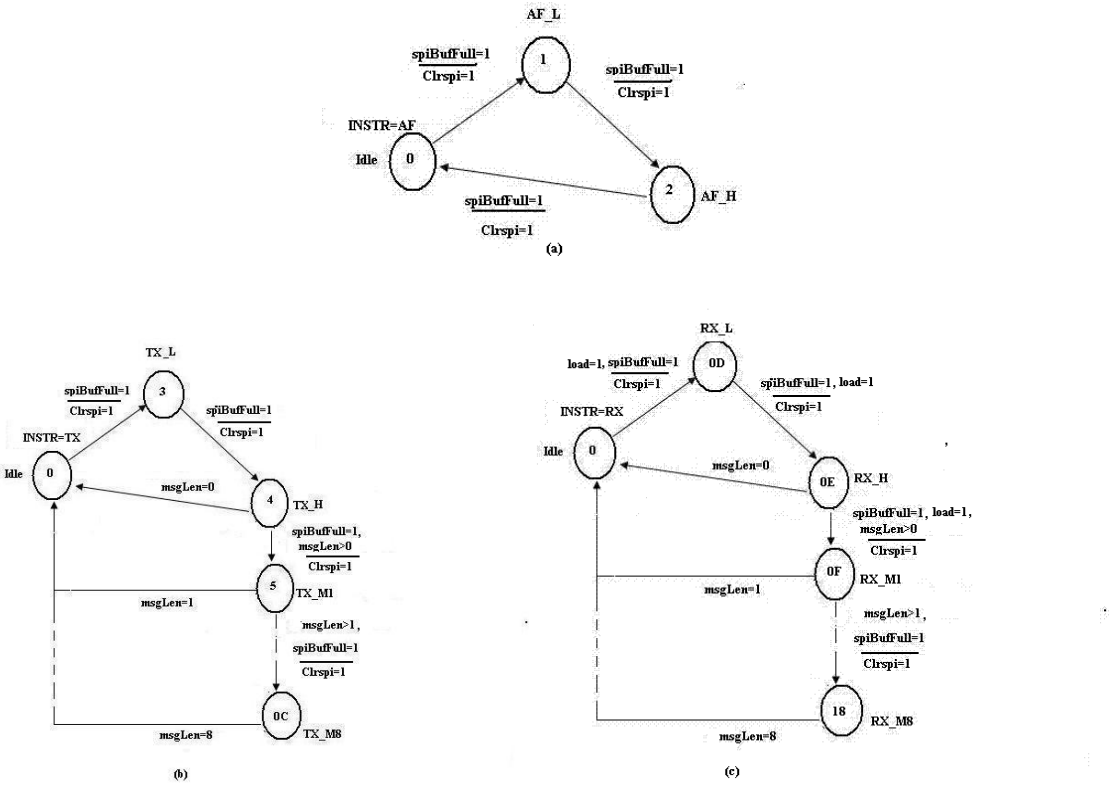

Fig 6: State diagram for (a) AF, (b) TX, (c) RX

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 4, July 2014

State transitions on TX and RX instructions are also similar to those of AF instruction. In both these cases, message length is contained in spiBuffer[6:4, 1]. State transition takes care of returning to idle state when appropriate number of bytes are exchanged.

Currently, we have not shown time-out feature of state engine. It is external to the SPI module. If there is no activity after any state other than idle for a fixed number of clock cycles, the state engine returns to idle state. This is required to handle errors in communication or wrong instruction.

III. IMPLEMENTATION

The implementation of the SPI frame for higher level application involves the SPI with two way serial shift register and the CAN controller. The block diagram is shown in figure 7.

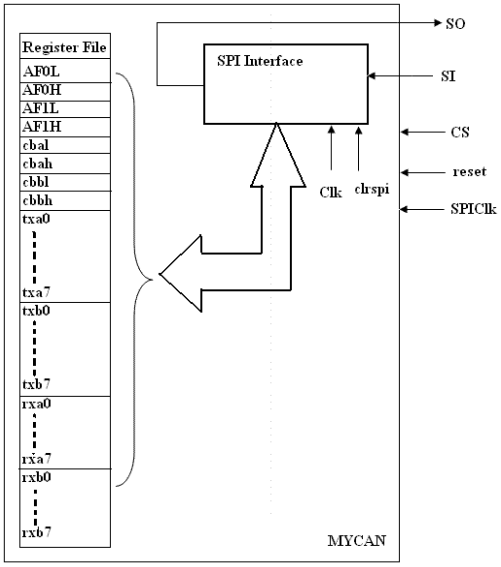

Fig 7: Block diagram of SPI implementation in CAN controller

To keep the model simple we have chosen the SPI interface with CPOL=1 and CPHA = 1 in which, data is latched on the rising edge of the clock and it is transmitted on the falling edge of the clock. Block diagram shown in Fig. 6 represents our implementation of SPI byte frame protocol for CAN-bus controller. Only the SPI side of the controller is shown here for clarity of purpose.

The circuit implements modules like latches and flip-flops, frequency dividers, serial and barrel shift register, up and down counters etc. The code is written in Verilog, using public domain software IVerilog.

IV. SIMULATION RESULTS

The proposed method is implemented in Verilog language using ICARUS Verilog tool and the waveforms are viewed on GTKWave platform.

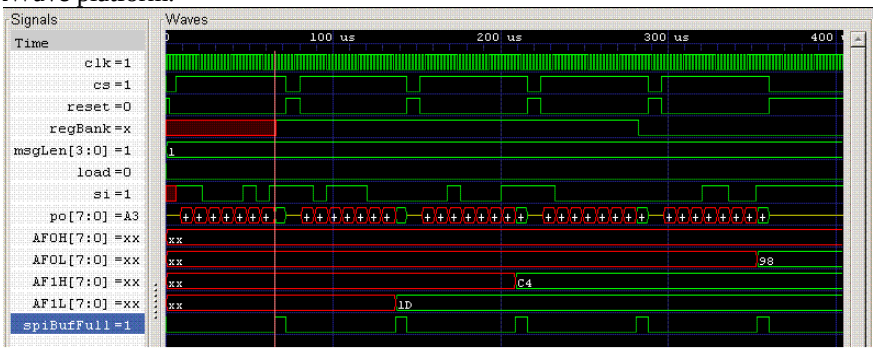

Fig 8: Simulation results for AF

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 4, July 2014

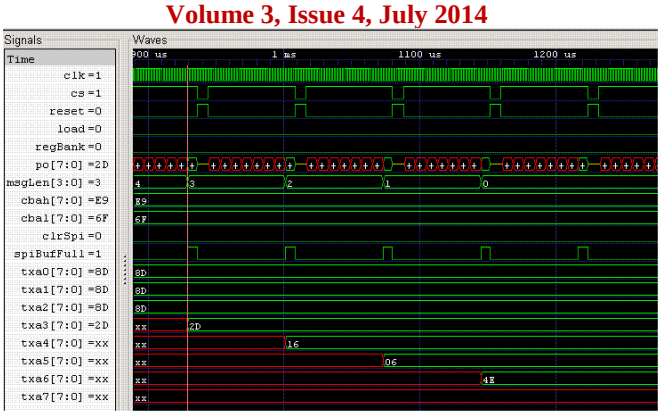

Fig 9: Simulation results of TX

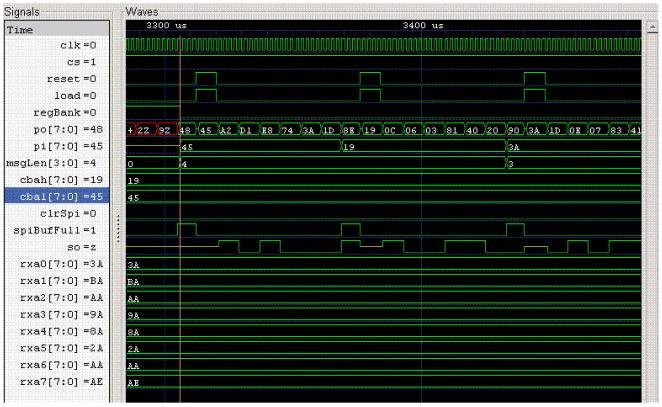

Fig 10: Simulation results of RX

V. CONCLUSION AND FURTHER WORK

A. Conclusion

The paper presents a structured way to design SPI based communication devices. An example implementation is also presented. Some of the results are shown graphically. The proposed SPI byte frame protocol for higher level application provides a structured and faster way to implement SPI in controller designs. The proposed method has been analyzed and implemented successfully on ICARUS Verilog simulation tool.

B. Future work

As discussed, there is scope to improve the protocol. We have only shown one implementation. As we implement more controllers, the protocol may undergo some revisions.

ACKNOWLEDGMENT

The authors would like to the thank C-Quad and KLEDRMSSCET Belgaum for all the support and facilities.

REFERENCES

[1] Anupama Benachinmardi, 'CAN bus Controller IP Design' M.Tech. Project report 2012 KLEDRMSSCET Belgaum.

[2] V. M. Aparanji and U. V. Wali "Evolution of device control networks and their standards" presented in National conference on 'Emerging trends in control, communication, signal processing and VLSI techniques CCSV-09' Oxford college of Engineering, Bangalore.

[3] U. V. Wali 'Plug and Play CAN' Tutorial presented at International Embedded System Conference ESC India July-2009, Bangalore.

[4] Shilpa Mayannavar, "A Novel Solution to the Synchronization problem of CAN" M.Tech. Thesis submitted to VTU June 2014.

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 4, July 2014

AUTHOR BIOGRAPHY

Ms. Shilpa Mayannavar completed her Bachelor of Engineering with major in Electronics and Communication Engineering at SGBIT Belgaum and she has received VTU Gold medal for academic excellence in the field of ECE. Presently she is Pursuing Master of Technology in VLSI Design and Embedded systems at KLEDRMSSCET Belgaum.

Prof. Ashwini Desai completed her BE (E & CE) from GIT, Belgaum affiliated to Karnataka University, Dharwad and was awarded M.Tech. by VTU, Belgaum in the year 2005. She is currently pursuing her PhD, while working as Asst. Professor at KLEDRMSSCET Belgaum. She has presented her work in National and International conferences.

Dr. U. V. Wali is a Professor at KLE DRMSS CET Belgaum and Director at C-Quad computers, Belgaum. He was awarded Ph D from IIT Kharagpur in the year 1986 for his work on He has published papers in 8 International conferences and has also published 6 Journals. His research interests include VLSI physical design, testing and verification, software design and architecture, among others. He is a fellow of Institution of Engineers, India.