Design of CAN Transmitter with an I2C Interface

Anupama K Benachinamardi
Assistant Professor, Dept. of Electrical and Electronics Engineering, SDM College of Engineering & Technology, Dhavalagiri, Dharwad,580002 INDIA.

U V Wali
Professor, Dept. of Electronics & Communication, K.L.E. Society's College of Engineering & Technology, Udyambag, Belgaum, 590008 INDIA

ABSTRACT
This paper describes development of a CAN transmitter with an I2C interface. Such a device could be interfaced with any existing microcontroller that supports I2C bus. I2C bus definition is simple enough to be implemented on any micro controller. All the development is done using public domain software, viz., Icarus Verilog and related tool set**.**

General Terms
Digital control network and Communication Networks.

Keywords
CAN (Controller Area Network), I2C (Inter IC) bus, IVerilog (Icarus Verilog), sda (serial data line), scl (serial clock line).

1. INTRODUCTION
A device Control Network consists of sensors, actuators and controllers designed to consume the data generated by sensor, historic data and user information and .produce output that can drive actuators and give feedback to user about the status of the system under control [1]. The control networks in general, can be electrical, pneumatic, hydraulic, optical, fluidic or a combination of these. Control intelligence can be deterministic, heuristic, stochastic, linear, nonlinear, fuzzy and neural or a mix of many. Electronic control circuits can be based on analog, digital, mechatronic, optical devices, which determine the control algorithms that can be realized. In present time frame, digital networks are far more popular than any other type of control networks. The performance, reliability and maintenance of digital systems depend upon many parameters but are relatively simple compared to analog networks. Digital control networks differ from computer networks in specific aspects. Control information has to be real time otherwise; the system may not work at all. Session management and security are often not present in control networks as the systems run on themselves, without much user intervention. Data transfer on control networks is generally limited to a few bytes to a few kilobytes, at the maximum. Most of the data on control networks is telemetry or remote control information, which is inherently limited in size. Images, multimedia are not transferred on control networks.

CAN bus was developed by BOSCH as a multi-master message broadcast system over a field bus with a maximum signaling rate of 1Mbps [2]. In this work, CAN bus is selected as communication bus as it has advantages over RS232 [3] bus. It defines bus arbitration, collision detection and avoidance at data link layer which are not specified in RS485 [4].

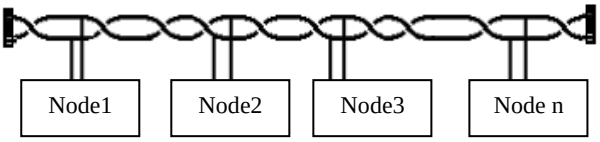

Fig1: CAN Field Bus

The figure 1 represents structure of CAN bus. Its ability to handle collision with minimal data loss is a tremendous achievement in device control networks. Another advantage is its support for multi master. The early CAN development was mainly supported by the automobile industry. The protocol is also widely used today in industrial automation and other areas like networked embedded control with applications in products such as medical equipment and building automation.

2. FEATURES OF CAN BUS AND I2C BUS
The Controller Area Network (CAN) Protocol was developed by Robert Bosch for Automotive Networking in 1982. Over the years, CAN have become a standard for Automotive Networking and industrial control. It is a low-cost, easy-to-implement, differential bus network with powerful error checking and a high transmission rate of up to 1 Mbps [2]. Each CAN packet is quite short and may hold a maximum of eight bytes of data. This makes CAN suitable for small embedded networks which have to reliably transfer small amounts of critical data between nodes. CAN bus is terminated on either ends by a 120 Ohm resistor. CAN will define only physical and data link layers. The CAN physical layer connects the CAN controller to the physical bus wires. A CAN network is made up of group of nodes. Each node can communicate with any other node. In CAN network, higher priority is given lower CAN-ID. Every node that wants to acquire the bus will be continuously monitoring the bus. When the bus becomes free, all the nodes that were monitoring for the bus will be considered. A device (node) with lowest CAN ID will acquire the bus for the transmission without any loss of control or data. Other nodes will pull back and try again when bus is free. This means at least one of the messages will be transmitted without loss of data. This behavior is significantly different than what is seen in popular networks like Ethernet CSMA/CD protocol. Ability to address bus contention allows errors and exceptions to be easily handled on a CAN bus. The Controller Area Network (CAN) efficiently supports distributed real-time control with a very high level of security. Its domain of application ranges from high speed networks to low cost multiplex wiring. In

automotive electronics, engine control units, sensors, anti-skid-systems, etc. are connected using CAN with bitrates up to 1 Mbps. At the same time it is cost effective to build into vehicle body electronics, e.g. lamp clusters electric windows etc to replace the wiring harness. CAN was developed specifically to realize a number of features suited to embedded systems design including high bit rate allowing real time speeds, high immunity to electrical interference, advanced error detection and confinement capabilities, low cost per node and fewer electrical connections, simple to implement and configure, reducing design time, high reliability and guaranteed transmission times for prioritized messages.

The message transfer process in CAN bus is controlled by four different types of frames; viz., Data Frame, Remote Frame, Error Frame and Overload Frame.

Data frame carries data from transmitters to receivers. A data frame is produced by a CAN bus node when the node wishes to transmit data or if this is requested by another node. Up to 8 byte data can be transported within a frame. Data frame consists of seven bit fields. The figure 2 indicates the structure of data frame.

S O F	Arbitration Field	Control Field	Data Field	CRC Field	ACK Field	E O F

Fig 2: Data Frame

SOF: Start of frame has only one dominant bit. A station is allowed to start transmission when the bus is idle. All stations will synchronize at positive edge of start of frame during the first transmission.

Arbitration Field: Arbitration field consists of 12 bits. It is divided into two sections; as Identifier and RTR bit. The identifier's length is 11 bits. These bits are transmitted in the order from ID-10 to ID-0. The least significant bit is ID-0. The 7 most significant bits (ID-10 to ID-4) may be of any combination but it must not be all recessive bits. In data frame's the Remote Transmission Request bit has to be dominant. Within a remote frame, the RTR bit has to be recessive [5].

Control Field: The control field consists of six bits. It includes the Data Length Code and two bits reserved for future expansion. The reserved bits have to be sent dominant. Receivers accept dominant and recessive bits in all combinations. The number of bytes in the data field is indicated by the data length code, which is of 4 bits wide and it should be transmitted within the control field.

Data Field: The data field consists of the data or message which is to be transferred within a data frame. It can contain from 0 to 8 bytes, which each contain 8 bits which are transferred MSB first.

CRC Field: It contains the CRC sequence followed by a CRC delimiter. CRC sequence is of fifteen bits wide; this CRC sequence is calculated by the polynomial and this sequence is followed by the CRC delimiter which consists of a single recessive bit.

ACK Field: The acknowledgement field is two bits long and contains the ACK slot and the ACK delimiter [5]. In the acknowledgement field, the transmitting station sends two recessive bits. A receiver which has received a valid message correctly, reports this to the transmitter by sending a dominant bit during the ACK slot. All stations having received the matching CRC sequence report this within the ACK slot by super scribing the recessive bit of the transmitter by a dominant bit. The ACK delimiter is the second bit of the ACK field and it should be a recessive bit.

EOF: Data frame ends with End of Frame which consists of seven recessive bits.

I2C was developed by Philips Semiconductors. It allows synchronous transfer of data over two wires, one for data (SDA) and the other for clock (SCL). I2C protocol defines the concept of master and slave devices. The bit transfer takes place as referred in the [6]. Figure 3 shows the status of SDA and SCL during START and STOP conditions during the data transfer. START and STOP conditions are always generated by the master. The bus is considered to be busy after the START condition and free after the STOP condition. Every byte put on the SDA line is of 8-bits long. We can send any number of data bytes. Each byte has to be followed by an acknowledge bit. Data is transferred with the most significant bit (MSB) first. A receiver can't receive another complete byte of data until it has performed some other function, for example servicing an internal interrupt, it can hold the clock line SCL low to force the transmitter into a wait state. Data transfer then continues when the receiver is ready for another byte of data and releases clock line SCL.

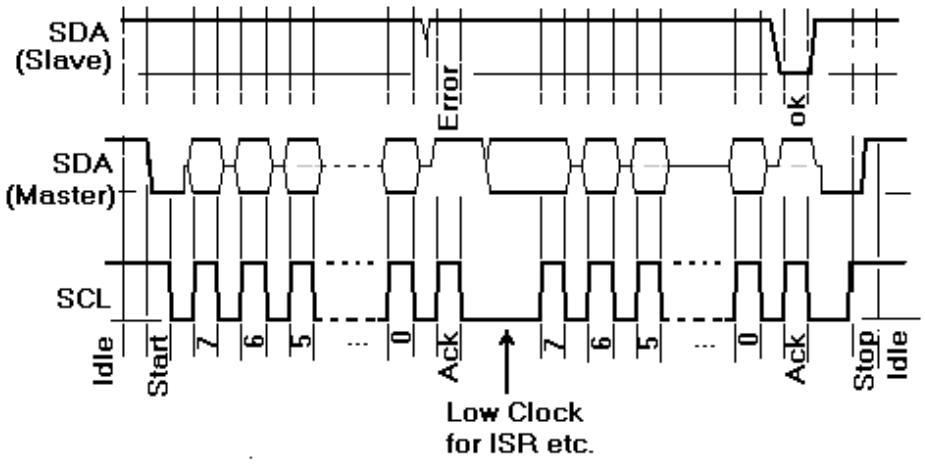

Fig 3: Data Transfer on I2C Bus

The transmitter can be sure that the data was read by the receiver without any error only when the receiver acknowledges the reception. If the data packet and acknowledgement packet are sent separately, there is always a doubt whether the error was in receiving data or the acknowledgement packet, forcing the transmitter to resend the data. So, acknowledgement is often a part of the data packet itself, only that it is not originated by the transmitter but by the receiver. In I2C, the acknowledge-phase handshaking is described. The transmitter releases the SDA line (making it high), clock pulse is generated by the master, the receiver must pulls down the SDA line if there is no error, if there was error in reception, receiver leaves SDA line high, receiver maintains SDA line status till the clock goes low. When a slave-receiver doesn't acknowledge the slave address, the data line must be left high by the slave. The master can then generate a STOP condition to abort the transfer. If a slave-receiver does acknowledge the slave address but, sometime later in the transfer cannot receive any more data bytes, the master must again abort the transfer. This is indicated by the slave generating the not acknowledge on the first byte to follow. The slave leaves the data line high and the master generates the STOP condition. If a master-receiver is involved in a transfer, it must signal the end of data to the slave-transmitter by not generating an acknowledgement on the last byte that was clocked out of the slave. The slave-transmitter must release the data line to allow the master to generate a STOP or repeated START condition.

3. DESIGN AND IMPLEMENTATION

It has been designed in a bottom up approach. Before designing it in Verilog, the various standalone CAN controllers are studied; such as SJA1000 [7] developed by Philips and MCP2515 standalone CAN controllers developed by Microchip. One of the primary reasons for designing in bottom up approach was that a purely behavioral design would take away an opportunity to learn many aspects of digital circuit design. It does not have any ready modules to plug-in into a behavioral design and not uses any existing designs but built. Second reason to take up a bottom up approach was that it would give an opportunity to start building a module library of any designer. All the fundamental concepts of digital blocks [8] are studied and the basic knowledge or concepts related to Verilog [9] [10] is studied.

3.1 I2C slave transmitter

The figure 4 shows the block diagram of I2C transmitter in slave mode. Here data is the 8 bit value or message which is to be transmitted. The data is transmitted through sda line. The active low acknowledgement which is indicated by 'nack' is the input for transmitter and output from the receiver. The I2C slave transmitter is designed based on finite state machine. The figure 5 shows the Finite State Machine of I2C slave transmitter.

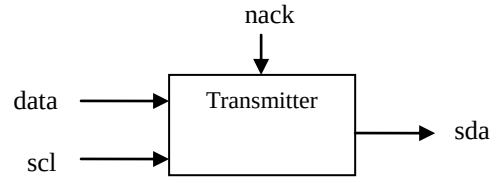

Fig 4: Block Diagram of I2C Slave Transmitter

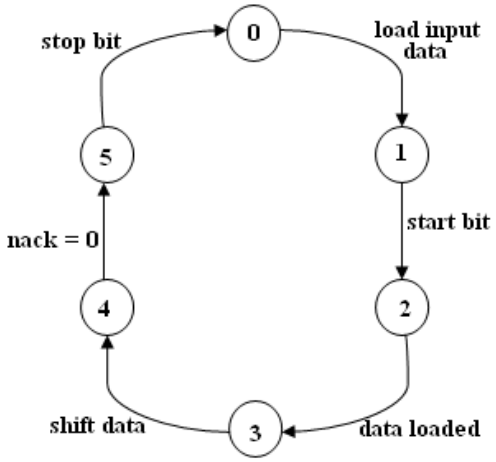

Fig 5: Finite State Machine of I2C Slave Transmitter

Before actual transmission takes place, the data which is to be transmitted is loaded into the register 'trBuff'. This loading of data takes place in fsm 0. In the 1st finite state machine, 'start bit' is enabled and the start bit is enabled when sda=0 and scl=1. This is explained briefly in the literature survey of I2C. Then after the data loading; the 'dataLoaded' becomes high. In 2nd finite state machine, 'busy' variable get set; it indicates that the transmission has started. In 3rd finite state machine; MSB of trBuff get transmitted on the sda line and the remaining bits. At the same time the counter starts counting. When it reaches counting up to 8, finite state machine changes to 4th state. On the 9th clock, acknowledgement bit get set. The 'nack' represents the active low acknowledgement bit; if nack=0 then it indicates that data is transmitted successfully; else it will raise the trErr flag that data is not transmitted correctly. In the 5th finite state machine sda line goes high, which indicates the stop bit. The simulation result of I2C slave transmitter on GTKWave is shown in figure 6.

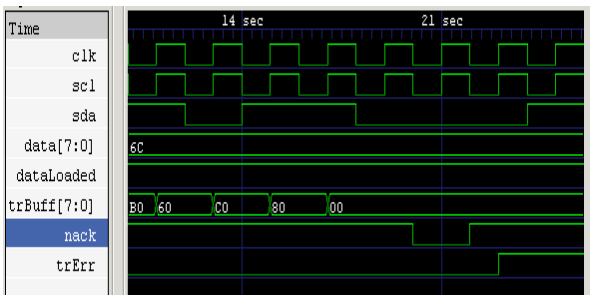

Fig 6: Simulation result of I2C Slave Transmitter.

3.2 I2C slave receiver

The figure 7 shows the block diagram of I2C receiver in slave mode. Here data is the 8 bit value or message which is to be received. The data is received through sda line. The active low acknowledgement which is indicated by 'nack' is the output of the receiver and input for transmitter. The I2C slave receiver is designed based on finite state machine. The figure 8 shows the Finite State Machine of I2C slave receiver. In the finite state machine of zero, both scl and sda line are high i.e. in idle condition. In the first finite state machine sda line goes low, which indicates the start bit. The counter starts counting when scl line goes low. In the finite state machine of 2; buffer

is cleared and data on sda line is taken into buffer. In finite state machine 3; counter reaches ninth clock and gives acknowledgement 'nack', which indicates that transmitted message is received successfully.

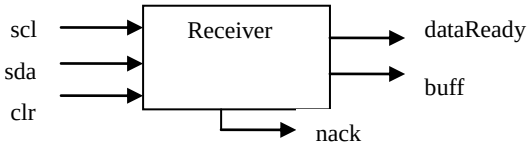

Fig 7: Block Diagram of Receiver

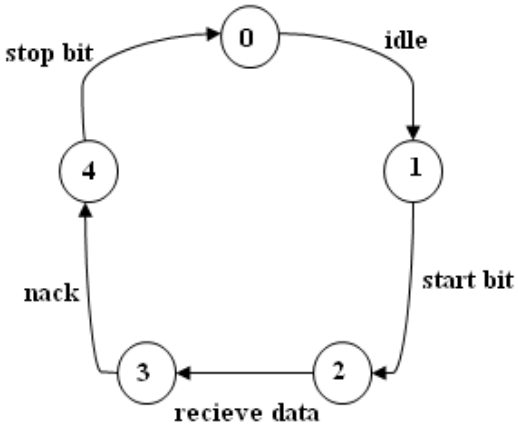

Fig 8: Finite State Machine of Receiver

In 4th finite state machine, sda line goes high, which indicates the stop bit.

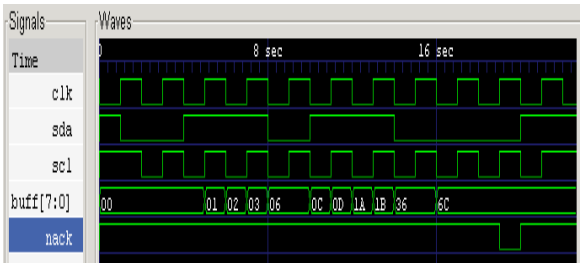

Fig 9: Simulation result of I2C Slave Receiver

3.3 CAN transmitter

The unit originating a message is called transmitter of that message. The unit stays transmitter until the bus is idle or the unit loses arbitration. Transmitter has been designed in Verilog in mixed type of abstraction levels. Transmitter module is the integration of blocks such as counter, d flip-flop, jk flip-flop, frequency division counter, stuff logic module and crc module.

The figure 10 represents the block diagram of CAN transmitter. The dotted line separates the test bench and transmitter module. The data which is to be transmitted is copied into the 'diCopy'. This data is of according to CAN data frame, which is described in section II. Frequency division counter is used to generate the baud rate clock. Before transmitting any message, stuff bit should be added for the security purpose. The bit-stuff area in a CAN bus frame includes the SOF, Arbitration field, Control field, Data field and CRC field. While receiving the message, the stuff bit should be removed and the original message should be received. In the above block diagram of CAN transmitter, s1, s2 and s3 are the three finite state machines.

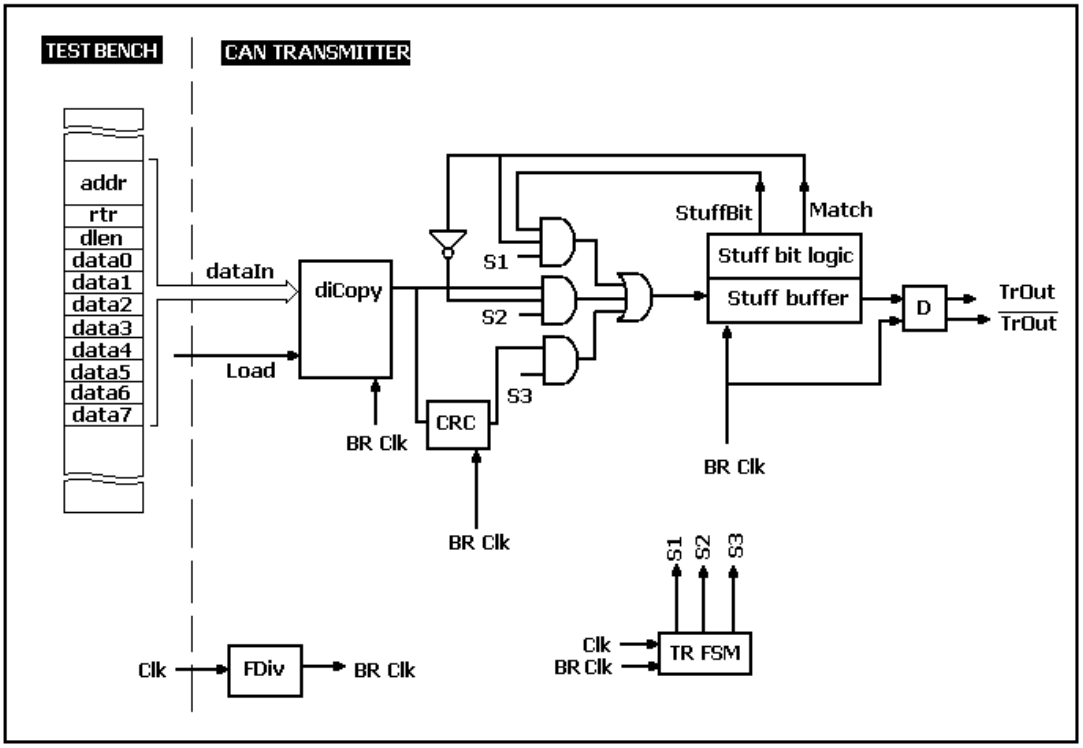

Fig 10: Block Diagram of CAN Transmitter

In the s1 finite state machine, all the variables are initialized and the data of five bits are given to the stuff logic. In second finite state machine, data is copied into the variable 'diCopy' and main counter continue the counting.

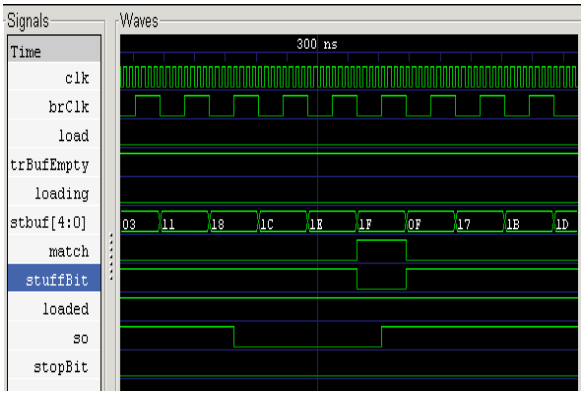

Fig 11: Simulation result of stuff bit

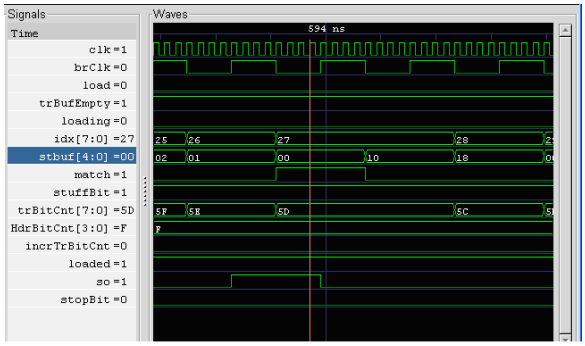

Fig 12: Simulation result of CAN Transmitter

All this process takes place when the transmit buffer is empty. During the third state, generated CRC value is taken. The CRC module works on baud rate clock. Among the above processes whichever is true, that will be given to the stuff logic, the stuff logic module has been described already. The output of stuff logic is given to the D flip-flop, which stores the data from stuff logic and from this, data will be transmitted which is indicated by 'TrOut'. The figures 11 and 12 clearly represent the simulation result of CAN transmitter with stuff bit.

4. CONCLUSION

The CAN bus transmitter soft-core (IP) is successfully designed in Verilog. This can be interfaced with any microcontroller which supports I2C. The CAN protocol is very robust, simple and it can support for multi master. This protocol can be used in industrial automation and other areas like networked embedded control with applications in products such as production machinery, medical equipment and building automation. The CAN transmitter is designed using Verilog which allows modeling of hardware components, which cannot be supported effectively by any other languages. The CAN receiver can be designed in future by integrating some basic modules such as counters, comparators, and stuff logic block along with acceptance filter. To improve the performance, some modifications can be made. A Digital Phase Locked Loop can be used for frequency sampling, which will provide better frequency synchronization and Multiple sampling of CAN bus input can be done. Currently, implementation of a single sample per clock has been done. Reliability can be increased by making several samples within a clock.

5. ACKNOWLEDGMENTS

We are grateful to C-QUAD Computers, a software developing company situated at Rani Channamma nagar, Belgaum, for providing related documents and good atmosphere for research work.

6. REFERENCES

[1] 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.

[2] U. V. Wali, "Plug and Play CAN", Tutorial presented at international Embedded Systems Conference, ESC India 2009, July 2009, Bangalore.

[3] RS 232 specification available: http://www.lammertbies.nl/comm/info/RS-232_specs.htm

[4] RS 485 specification available:
http://www.lammertbies.nl/comm/info/RS-485.html

[5] "CAN specification" Robert Bosch available: http://www.semiconductors.bosch.de/pdf/can2spec.pdf

[6] "I2C specification" available: http://www.i2c-bus.org/fileadmin/ftp/i2c_bus_specification_1995.pdf

[7] "APPLICATION NOTE: AN25.SJA1000 Stand-alone CAN controller" Philips semiconductors.

[8] Charles H. Roth, Jr, "Fundamentals of Logic Design", 5th Ed., Thomson Brooks/Cole, 2004.

[9] Z. Navabi, "Verilog Digital System Design", 2nd Ed., Tata McGraw Hill, New Delhi, 2008.

[10] Sameer Palnitkar, "Verilog HDL, A Guide to Digital Design and Synthesis", Sunsoft Press, 1996.