A Novel Bit Stuffing technique for Controller Area Network (CAN) Protocol

Tapas Ranjan Jena
Dept. of ECE
NIT Rourkela, Odisha, India
tapasranjanjena@gmail.com

Ayas Kanta Swain
Dept. of ECE
NIT Rourkela, Odisha, India
swain.ayas@gmail.com

Kamalakanta Mahapatra
Dept. of ECE
NIT Rourkela, Odisha, India
kkm@nitrkl.ac.in

***Abstract*—Control area network (CAN) is a two- wired, half duplex, high-speed network system that is far superior to conventional serial communication protocol such as RS232 in terms of functionality and reliability. CAN implementation are also cost effective. All CAN controllers in a network operate at the same frequency for safe and proper data transfer. As CAN uses Non Return to Zero (NRZ) bit encoding, long sequence of same polarity bit will cause the losses of synchronization among the CAN controller. CAN provides an effective mechanism for clock synchronization known as "bit-stuffing". It is very difficult to predict the precise transmission time of message which leads to an adverse impact on many time critical applications. To mitigate above problem different "bit-stuffing" techniques such as XOR masking and Software Bit Stuffing (SBS) is available in the literature. In this paper a novel alternative method known as Eight-to-Eleven Modulation (EEM) technique is used for "bit-stuffing" and a comparison is brought with existing SBS technique and the superior features of EEM is established. The proposed technique is validated through FPGA implementation.**

Keywords—Controller Area Network, Bit stuffing, EEM

I. Introduction

Controller Area Network (CAN) [1] is a serial communication protocol technology that was originally designed for the automotive industry, especially for European cars, but has also become a popular bus in industrial automation as well as other application [1] – [5].

CAN protocol is designed for short message, no more than eight bytes long [6]. CAN afford a maximum transmission rate of 1 Mbps. CAN uses "Non Return to Zero" (NRZ) coding. In Manchester encoding for sending one symbol it required two bit of information whereas NRZ encoding required only one bit. As compared to Manchester coding, NRZ coding decreases the signal transition for every "0" or "1" bit transmission and also increases the band width utilization. But in NRZ coding there is no easy way to tell where each bit starts or ends when there is continuous "0" or "1". When this type of situation arises it may happen that transmitter and receiver loses synchronization. That means whatever the data bit send by the transmitter would not be properly received by the receiver. To avoid this type of situation CAN provides a mechanism called "bit-stuffing", in which no more than five consecutive bits (with the same polarity) to transmit on the bus [4]. Transmitted bits are checked before transmission, if a long sequence of same polarity bit come it add an opposite polarity bit with a regular interval. The added bit will give an extra transition for the synchronization of receiver and transmitter.

Fig.1 shows the basic operation of the bit-stuffing mechanism carried out in the sending end of CAN controller.

Original frame:11001111110000000001111111111111.......
Transmitted frame: 110011111**0**100000**1**000011111**0**11111**0**111.
(Bold digits are indicated as Stuffed bits)

Fig. 1.The basic operation of bit-stuffing in the sending CAN controller

In this bit-stuffing process the number of stuffed bit required completely depends on the incoming bit pattern. So the output frame length is a complex function of incoming bit pattern. Consequently, it is difficult to predict the precise transmission time of bit pattern when we have a time critical design.

It may be noted that once transmission starts, a CAN message cannot be interrupted, and the variation in transmission time therefore has the potential to cause a significant impact on the real-time behavior of systems employing this protocol. A number of studies have considered ways for bounding the response time of message frames in CAN-based networks [7] – [9]. The variation in message response time is referred to as "jitter". Jitter has a key impact on the performance of many applications, particularly those involving data acquisition, data processing and control. Impacts of jitter on various applications are given in [10] – [12]. Further possible sources of jitter and proposed solutions to limit this jitter in CAN networks are presented in [13] – [15]. This paper only deals with jitter caused by bit stuffing.

In this paper our aim is to reduce the timing variation due to variable message length causes by bit-stuffing in CAN protocol. Initially we develop a module by using "Software Bit Stuffing" (SBS) and then by using a new technique called "Eight-to-Eleven Modulation" (EEM). For both the logic we made a logic utilization comparison as well as frequency of operation.

The rest of this paper comprises the following. A brief background discussion of the previous different "bit-stuffing" technique developed in [16] and [17] is discussed in section II. "Eight-to-Eleven Modulation" (EEM) is discussed in the section III. Implementation of the EEM algorithm is discussed in the section IV. Result and analysis is discussed in section V. System implementation in FPGA is discussed in section VI. Finally a conclusion is drawn in section VII.

II. Previous work in this area

A. *Selective XOR masking*

XOR masking was first proposed by Nolte et al.[18,19]. In this technique transmitted bytes are XOR-ed with the masking bit "101010…" for removing stuff-bits in particular sets of data (which tend to have long sequences of 1s and 0s). Again the received bytes are XOR-ed with the same masking bit to get the original data bytes. This technique reduces CAN message-length variations (i.e. jitter) to low levels and also reduces large computational or memory overheads. These techniques are described in-detail in [18] and [19].

In a more general case, the transmitted data bytes may not have continuous 0 or 1. It may contain random 0 or 1. When the data sets are random, directly using Nolte XOR masking [16] level of bit-stuffing cannot be reduced significantly. In this case the data bytes are tested and the Nolte XOR masking is applied to the selected data byte. Such a technique was referred as "Nolte C" or "selective XOR masking".

B. *Software Bit Stuffing (SBS)*

In selective XOR masking the levels of message length variation was reduces significantly, but few conditions remain unsolved like the boundaries of the adjacent bytes are processed individually. So the continuous 0 or 1 at the end of one frame and the same continuous 0 or 1 at the start of the next frame give a long sequence of 0 or 1. To eliminate this type of condition "Software Bit Stuffing" (SBS) technique was proposed[17] and was implemented in [22].

In this technique data frames are checked before transmitting. If a sequence of four consecutive bits is detected, SBS adds an additional bit of opposite polarity. Here at most five consecutive bit of same polarity is allowed (one stuff bit and four in coming data bit).

III. Eight-to-Eleven Modulation (EEM)

In SBS data encoding / decoding are performing at run-time (while the system is operating) using a function call approach [17]. Due to run time processing the CPU overheads of the processor increased.

To achieve the high processor utilization as well as to reduce the effect of bit stuffing a new X-to-Y modulation approach is used. In X-to-Y modulation approach "X" signifies the number of original data bit and "Y" signifies the number of encoded data bit. A most known example of X-to-Y modulation is Eight-to-Fourteen Modulation (EFM) which is used in compact discs (CDs) [20].

Eight-to-Eleven Modulation (EEM) [21] is another type of X-to-Y modulation where "X" equals to 8 (the number of bits per byte) and "Y" equals to 11(the number of encoded data bits).

$$No.of\ stuffed\ bits = \left\lfloor \frac{No.of\ bits\ subject\ to\ bit\ stuffing\ -1}{Maximum\ \ No.of\ consecutive\ bits\ allowed\ -1} \right\rfloor \qquad (1)$$

As given in [18], the number of stuffed bits necessary to avoid CAN bit-stuffing is found as follows:

Where the symbol $\lfloor\ \rfloor$ denotes the floor function to make the number of stuffed bit to integer. In our case we have 8 bits that are subjected to bit-stuffing and maximum 4 consecutive bits are allowed for bit-stuffing, then the number of stuffed bits required in each byte is 2 bits.

The worst case "bit-stuffing" for a CAN controller will be five consecutive bits followed by a opposite polarity bit and so on [18]. In software level to avoid hardware "bit-stuffing" consecutive bit with same polarity should not be more than four. The worst case condition for 8 bit of data will be four "0" followed by three "1" and one"0" (i.e. 00001110). From the previous discussed formula the number of required stuff bit will be two. After addition of these two stuff bit we found that the sequence will be like "0000111100". Here the bold bits are the stuffed bits. When two bit pattern of the above type transmitted consecutively, we found five consecutive same polarity bit at the boundary.

To avoid such boundary condition we need to add another extra stuff bit. After adding the extra stuff bit the total number of stuff bit will be three for a set of 8 bit data. So the total number of bit encoded bit will be 11 bit. Fig. 2 shows the 11 bit encoded format of the 8 bit input data, where we have one stuff bit at the middle of the input data sequence, one near most significant bit and one near least significant bit.

Fig. 2 Encoding data byte in EEM method (SB stands for "stuffed bit")

Bit 1	SB 1	Bit 2	Bit 3	Bit 4	SB 2	Bit 5	Bit 6	Bit 7	SB 3	Bit 8

In Fig. 2 SB are the stuff bit, bit 1 to 8 are the original 8 bit input data. Stuff bits values are the negation of the previous bit. In this encoding format the worst case data frame will not exceed 4 consecutive same polarities. For example let the input sequence will be "11110001". After encoding the result will be "10111000011" that satisfy the "bit-stuffing" condition and also solve the boundary condition problem.

IV. Implementation of EEM Algorithm

EEM technique can be implemented by Lookup table approach or by Function call approach. Look up table approach again can be implemented by Explicit EEM table or by Implicit EEM table. Function call approach again can be implemented by Algorithmic coding or by Mathematical coding.

Here we are following algorithmic approach to implement EEM technique. The flow chat of the coding is presented in the Fig. 3 and is self-explanatory.

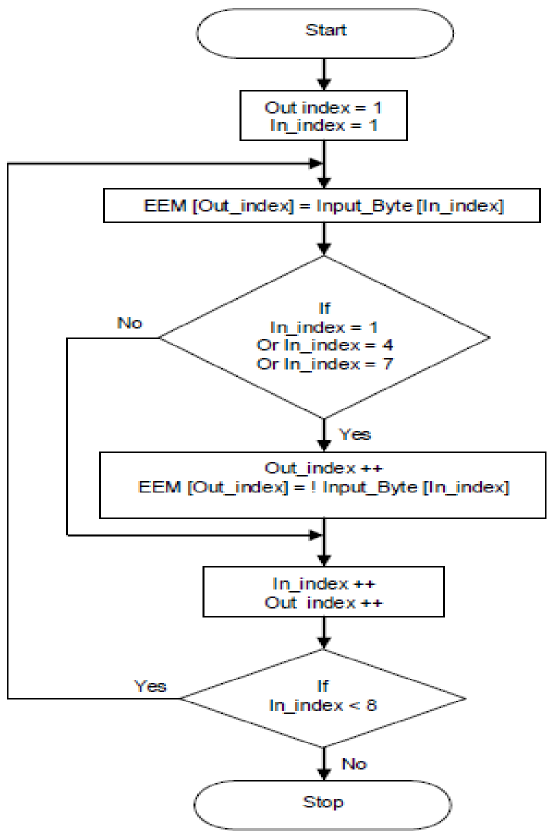


Fig. 3 EEM Algorithm Implementation

V. Results And Analysis

Fig. 4 shows the simulation result of the de-stuffing unit of the CAN Module. From the EEM generation algorithm we found that 8 bit data is converted in to 11 bit encoded data.

While converted in to 11 bit encode data, it copy bit 1 followed by 1 stuff bit, again copy bit 2 to 4 followed by1 stuff bit , again copy bit 5 to t followed by 1 stuff bit and at last copy the bit 8. So the copy sequence is 1 3 3 1 for a particular byte of data. When a sequence of byte is processed then the bit copy sequence will be 1 3 3 2 3 3 2 3 3 .. and so on .In the Fig. 5 we can able to see the state transition, in which state 2 is for copy 2 bit data and state 4 is for copy 3 bit data in between these two state 3 and 5 are stuff 0 and stuff 1 respectably.

After getting the RTL simulation result we dump the code in to Vertex 2 pro board and the result is also verified in hardware level. The result of the FPGA board is analyzed with the help of chip scope pro which is shown in the Fig. 5.

VI. System Implementation In FPGA

The System Implementation and verification were done using Xilinx ISE Simulator with VIRTEX-II Pro FPGA kit. Both SBS as well as EEM architecture implemented and tested. The logic utilization and frequency of operation calculated and listed as in table 1.

Table I

Bit-Stuffing Technique	**SBS**	**EEM**
No. of Slices	184	191
No. of Slice Flip Flops	139	146
No. of 4 input LUTs	359	374
No. of Bonded IOBs	122	122
Max. Frequency	223.330MHz	237.225MHz

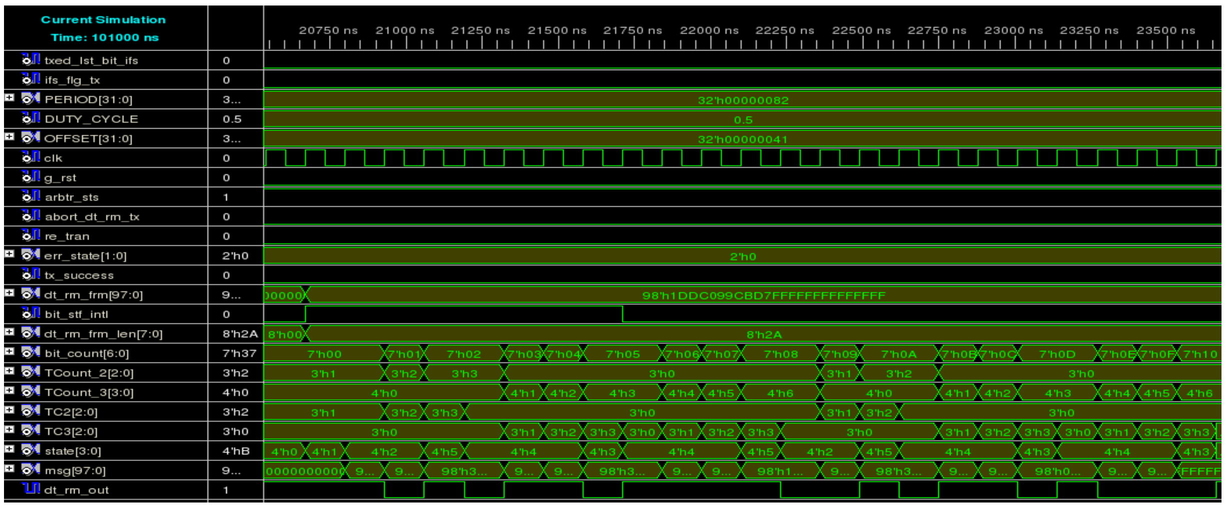
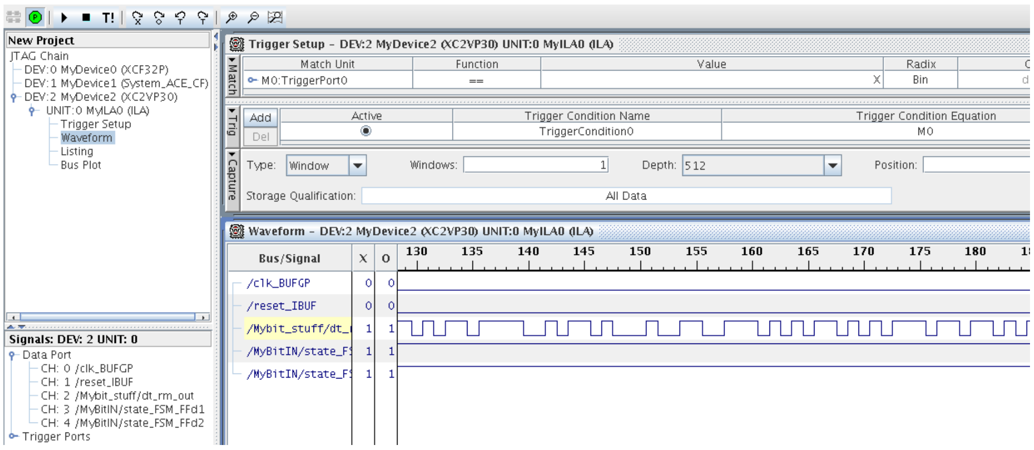


Fig. 4 Simulation Result of Bit-Stuff unit



New Project
JTAG Chain
- DEV:0 MyDevice0 (XCF32P)
- DEV:1 MyDevice1 (System_ACE_CF)
- DEV:2 MyDevice2 (XC2VP30)
  - UNIT:0 MyILA0 (ILA)
    - Trigger Setup
    - Waveform
    - Listing
    - Bus Plot

Signals: DEV: 2 UNIT: 0
- Data Port
  - CH: 0 /clk_BUFGP
  - CH: 1 /reset_IBUF
  - CH: 2 /Mybit_stuff/dt_rm_out
  - CH: 3 /MyBitIN/state_FSM_FFd1
  - CH: 4 /MyBitIN/state_FSM_FFd2
- Trigger Ports

Trigger Setup - DEV:2 MyDevice2 (XC2VP30) UNIT:0 MyILA0 (ILA)

Match Unit	Function	Value	Radix
M0:TriggerPort0	==	X	Bin

Active	Trigger Condition Name	Trigger Condition Equation
●	TriggerCondition0	M0

Type: Window Windows: 1 Depth: 512 Position:
Storage Qualification: All Data

Waveform - DEV:2 MyDevice2 (XC2VP30) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O
/clk_BUFGP	0	0
/reset_IBUF	0	0
/Mybit_stuff/dt_	1	1
/MyBitIN/state_F	1	1
/MyBitIN/state_F	1	1

Fig. 5 Board Level Verification Result of Bit-Stuff unit

VII. EEM Implementation in CAN Controller Design

From Table I it is observed that due to the reduction of jitter in EEM, frequency of operation increases as compared to SBS technique. EEM technique is implemented for both bit-stuffing and de-stuffing operation of CAN controller design. EEM implemented design is tested and verified in Xilinx[23] ISE Simulator.

After designing the CAN controller by using EEM technique, four CAN controllers are connected in a network to test how CAN message supports "Carrier Sense Multiple Access with Collision Avoidance" (CSMA/CA). CSMA/CA is known as non-destructive bit-wise arbitration. In CSMA/CA method the node will monitor the network and wait for the bus to ideal. Once the bus is ideal the node can transmit the message. If at a time multiple node will request the bus access the node sending higher priority message will get the access of the bus first.

Fig. 6 shows the result where four CAN nodes are connected in a network through a commonly shared bus. Can_bus_out_0, 1, 2, 3 are indicated the output of the four can connected through the CAN bus. All nodes start transmitting the message frame at the same time but the node sending higher priority message frame will get the bus access first. Here node 2 is in an ideal state. Node 1 sends a message frame having less priority as compared to other so it stops transmitting first. Node 0 and Node 3 both are sending the message frame having same message id but node 0 is a data frame and node 3 is a remote frame. So node 0 get the access of the bus as the data frame has a higher priority as compared to the remote frame.

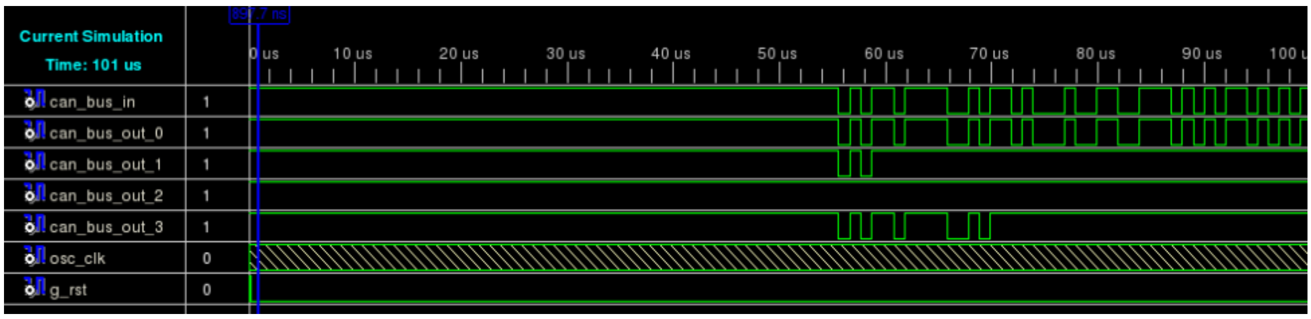


Fig. 6 : CAN wire-anding 4 node structure.

VIII. Conclusion

Bit-Stuffing unit for a CAN controller is designed by both Software Bit Stuffing (SBS) technique and Eight-to-Eleven Modulation (EEM) technique. The verification of the design is done using XILINX ISE simulator. Finally, by using Xilinx Chip Scope Pro the design is verified in hardware level after dumping the code in to Virtex-II Pro kit. From the logic Utilization summery we found that, both the technique uses almost equal logic blocks whereas due to reduction in "jitter" in EEM technique the operating frequency increases.

References

[1] R. Bosch, *CAN Specification 2.0*. Robert Bosch GmbH, Postfach,Stuttgart, Germany, 1991.

[2] L.B. Fredriksson, "Controller Area Networks and the protocol CAN formachine control systems", Mechatronics, Vol. 4, No. 2, 1994, pp. 159-192.

[3] J.P. Thomesse, "A review of the fieldbuses", Annual Reviews inControl, Vol. 22, 1998, pp. 35-45.

[4] M. Farsi, and M.Barbosa, *CANopen Implementation, applications toindustrial networks*, Research Studies Press Ltd, England, 2000.

[5] M. Short, and M.J. Pont, "Fault-Tolerant Time-Triggered

Communication Using CAN", IEEE Transactions on IndustrialInformatics, Vol. 3, No. 2, 2007, pp. 13-142.

[6] Daniel Mannisto, Mark Dawson, An Overview of Controller Area Network (CAN) Technology, mBus, 2003.

[7] K.W. Tindell, A. Burns, and A.J. Wellings, "Calculating Controller AreaNetwork (CAN) Message Response Times", Control Engineering

Practice, Vol. 3, No. 8, 1995, pp. 1163-1169.

[8] Navet, N. and Song, Y.Q. (1998) "Design of reliable real timeapplications distributed over CAN (controller area network)",Proceedings of INCOM'98, IFAC Symposium on Information Controlin Manufacturing, Metz 22–24 June, 1998, pp. 391-396.

[9] R. Rudiger, "Evaluating the temporal behaviour of CAN based systemsby means of a cost functional", Proceedings of the Fifth International CAN Conference, San Jose, CA, USA, November, 1998, pp. 10.09-10.26.

[10] A.J. Jerri, "The Shannon sampling theorem: its various extensions andapplications a tutorial review", Proceeding of the IEEE, Vol. 65, 1977,

pp. 1565-1596.

[11] S. Hong, "Scheduling Algorithm of Data Sampling Times in theIntegrated Communication and Control Systems", IEEE Transactions onControl Systems Technology, Vol. 3, No. 2, 1995, pp. 225-230.

[12] A. Stothert, and I. MacLeod, "Effect of Timing Jitter on DistributedComputer Control System Performance". Proceedings of DCCS'98 –15th IFAC Workshop on Distributed Computer Control Systems,September 1998.

[13] P. Verissimo, and L. Rodrigues, "A posteriori Agreement for Fault-Tolerant Clock Synchronization on Broadcast Networks", the 22ndInternational Symposium on Fault-Tolerant Computing, Boston, USA,July, 1992.

[14] L. Rodrigues, M. Guimarães, and J. Rufino, "Fault-Tolerant ClockSynchronization in CAN", Proceedings of the 19th IEEE Real-TimeSystems Symposium, Madrid, Spain, December 2-4, 1998.

[15] J. Barreiros, E. Costa, J. Fonseca, and F. Coutinho, "Jitter reduction in areal-time message transmission system using genetic algorithms",Proceedings of the CEC 2000 – Conference of EvolutionaryComputation, USA, July 2000.

[16] M. Nahas, and M.J. Pont, "Using XOR operations to reduce variationsin the transmission time of CAN messages: A pilot study". In:Koelmans, A., Bystrov, A., Pont, M.J., Ong, R. and Brown, A. (Eds.),Proceedings of the Second UK Embedded Forum, Birmingham, UK,October 2005, pp. 4-17. Published by University of Newcastle uponTyne.

[17] M. Nahas, M. J. Pont, and M. Short, "Reducing message-lengthvariations in resource-constrained embedded systems implementedusing the CAN protocol", Journal of Systems Architecture, Vol. 55, No.5-6, 2009, pp. 344-354.

[18] T. Nolte, H. Hansson, C. Norström, and S. Punnekkat, "Using BitstuffingDistributions in CAN Analysis", IEEE/IEE Real-TimeEmbedded Systems Workshop (Satellite of the IEEE Real-Time SystemsSymposium) London, 2001.

[19] T. Nolte, H. Hansson. and C. Norström, "Minimizing CAN responsetimejitter by message manipulation", IEEE Real Time Technology and

Applications Symposium 2002, pp. 197-206.

[20] J. Watkinson, *Introduction to Digital Audio*, Focal Press, 2002.

[21] MouaazNahas "Applying Eight-to-Eleven Modulation to reduce message-length variations in distributed embedded systems using the Controller Area Network (CAN) protocol", Canadian Journal, Vol. 2, July 2011.

[22] SreeramKrishnamoorthy, "Design an ASIC Chip fpr a Controlller Area Network (CAN) Protocol Controller".

[23] Xilinx ISE Tutorial,www.xilinx.com