Improving the performance and reliability of systems which employ the 'CONTROLLER AREA NETWORK' protocol through low-level changes to the controller implementation

Thesis submitted for the degree of
Doctor of Philosophy
at the University of Leicester

by

Imran Sheikh

Embedded Systems Research Group
Department of Engineering
University of Leicester
Leicester, UK.

JANUARY 2011

Abstract

The CAN (Controller Area Network) protocol provides one of the cost-effective methods to network current generations of distributed embedded systems. Although it is a robust protocol with short messages and simple priorities, it is largely thought of as only being suitable for soft real-time, event-triggered systems. Safety critical applications require highly predictable behaviour with strict bounds on worst-case message transmission times; the next-generation mechatronic systems also requires a high level of information throughput. In its current form, CAN lacks most of these requirements principally due to its medium access scheme and physical-layer design. This thesis presents a frame work which aims to enhance the capabilities of CAN, in order to push the boundaries of the protocol's current operation. In particular, the main research question to be addressed is the exploration of the extent to which low-level modifications can enhance CAN suitability for use in the next generation of critical systems. In order to answer this question, it is first necessary to develop a flexible and robust platform to implement these modifications using a novel facility made up from custom soft-core CAN controllers. This novel facility was then employed to implement and experimentally investigate three small but conceptually significant protocol modifications as follows:

Increasing the effective data rate from 1 to 10 Mbps whilst doubling the effective payload from 8 to 16 bytes; Reduction of unwanted transmission jitter by compensating for bit stuffing; Enabling a windowed transmission scheme to provide optimal trade-off's between transmission reliability and real-time behaviour in noisy environments.

The thesis describes the results obtained from these experiments and summarizes the main pros and cons that appear. The thesis then concludes with observation that the modified CAN protocol may be suitable for use with certain classes, of the next generation time-critical distributed embedded systems.

Dedication

To My Late Parents
May Allah rest there soul in peace

Acknowledgements

All praises to the Almighty 'ALLAH' who is the most beneficent who has given me the strength to achieve what I have in my life and whatever I will in future.
The work presented in this thesis would not have been possible without the help of my supervisors Michael Short and Michael Pont. Michael's thank for believing in me, you have always supported my ideas and work, and inspired me throughout the last four years.
Also, I must thank my colleagues at ESL and TTE systems, especially Dr Ayman Gendy for listening patiently and giving me some inspirational ideas.
Specially i would also like to thanks Dr K.M. Yahya and Dr Tariq Jadoon who have always acted as my mentor and provided with every support in my carrier.
I would also like to thank NWFP UET Peshawar, Higher Education Commission and people of Pakistan for funding my PhD studies, it is a such a great honour to represent my country who although been in dire financial crisis have funded my studies.
Finally I would like to thanks, my wife Sana, my beautiful and *lovely* three daughters Sara, Areesha and Momina who have extended their love and cooperation in achieving this goal. I cant forget here my Mother and Father whom I miss dearly on this day, when i am so near to achieve my doctorate for which they had worked so hard in their life time. I also want to thanks my three sister and brother Zeeshan for their support to continue my studies during the tough time, when we lost both of our dear parents.

My home, Leicester — Imran Sheikh
August 12, 2010.

Related Publications and Contributions

A number of papers were published during the course of the work described in this thesis. Please note that the contents of some of these papers have been adapted for presentation in this thesis:(in reverse chronological order):

1. Imran, S. and Short, M.J. (2010), **Conformance Testing of Soft-Core CAN Controllers: A Low-Cost And Practical Approach**, In: J.A. Cetto et al. (Eds), Informatics in Control, Automation and Robotics: Lecture Notes in Electrical Engineering, Vol. 85, Part 2, pp. 129-141, Springer-Verlag, Berlin, ISBN: 978-3-642-19729-1, 2011.

2. Short, M.J. and Imran, S.(2010), **Computing Optimal Window sizes to enforce dependable communications in time-triggered Controller Area Networks**, Proceedings of the 9th International workshop on Real time Networks RTN 2010, Brussels, Belgium, 6th July, 2010.

3. Imran, S., Short, M.J. and Hanif, M. (2010), **Improving information throughput and transmission predictability in Controller Area Networks**, Proceedings of IEEE Symposium on Industrial Electronics, ISIE 2010, Bari, Italy, pp. 1736-1741,,July 4-7, 2010. A revised version of this paper has been recommended for the special section of IEEE Transaction on Industrial Electronics and has been submitted for review with the title **Analysis of an Enhanced Protocol Controller for Next-Generation CAN Networks**.

4. Imran, S., Short, M.J. and Yahya, K.M. (2010), **Analysis of Overclocked Controller Area Network**, Proceedings of the 7th IEEE International Conference on Networked Sensing Systems (INSS 2010), Kassel, Germany, June 15 - 18, 2010, pp. 37-40, Published by IEEE Industrial Electronics Society, ISBN 978-1-4244-7910-8

5. Short, M.J. and Imran, S. (2010), **Dual-rate overclocking in CAN networks: a soft-core controller prototype**, Proceedings of the 7th IEEE International Conference on Networked Sensing Systems (INSS 2010), Kassel,

Germany, June 15 - 18, 2010, pp. 314-317, Published by IEEE Industrial Electronics Society, ISBN 978-1-4244-7910-8.

6. Imran, S. and Short, M.J. (2010), **Employing Integrated Logic Analysers and Virtual I/Os to Verify Soft Core Protocol Implementations**, IAENG International Journal of Computer Science, Vol. 37(1), pp. 36-49.

7. Imran, S. and Short, M.J. (2009), **A Low Cost and Flexible Approach to CAN Conformance Testing**, Proceedings of the 6th International Conference on Informatics in Control, Automation and Robotics, Intelligent Control Systems and Optimization, ICINCO 2009, Milan, Italy, July 2-5, 2009, INSTICC Press 2009, ISBN 978-989-8111-99-9.

8. Imran, S., Short, M.J. and Athaide, K.F. (2009), **Using Virtual I/O for CAN Bit Timing Conformance Tests**, Proceedings of the World Congress on Engineering, WCE 2009, London, U.K, July 1 - 3, 2009, Vol. 1, pp. 480-485, ISBN: 978-988-17012-5-1. (Best Student Paper Award).

9. Imran, S. and Short, M.J. (2009), **Improving Information Throughput in CAN Networks: Implementing a Dual Speed Approach**,Proceedings of the 8th International workshop on Real time Networks, RTN 2009, Dublin ,Ireland, June, 2009.

10. Imran, S., Short, M. and Pont, M.J. (2008), **Hardware implementation of a shared-clock scheduling protocol for CAN: A pilot study**, in Proceedings of the 4th UK Embedded Forum (September 2008, Southampton, UK), pp.72-78. ISBN 978-0-8634-1949-2. ISSN 0537-9989.

Techical Report

1. I. Sheikh and M. Short. **CAN conformance testing-A new approach** Technical Report, Tech-Report ESL-09-01, ESL, Engineering Department, University of Leicester, Feb 2009.

This technical report include the details of all the conformance tests performed on the CAN IP core during this research work.

Contributions

This next section summarizes the main contributions of the author of this thesis, in the papers listed above. The number in brackets represents the percentage contribution of the author in each case.

Conformance Testing

Paper 1 (75%), 6 (80%), 7 (80%) and 8 (85%) are principally adopted for Chapter 5 on CAN conformance testing.
Paper 1: This paper is an extension of the paper 8 (conference paper) to a book chapter.
Paper 6: This paper is an extension of the paper 7 (conference paper) to a journal paper.
The ideas contained within these works were conceived by the thesis author, with appropriate guidance and suggestions from the supervisor. Formation of the Conformance test bench, running of the tests and documentation were all done by the thesis author. In paper 7 the third author helped in the installation of the Xilinx tools for the running of the tests.

Overclocking

Paper 4 (65%), 5 (45%) and 9 (70%) are adopted in the Chapter 6 on CAN overclocking. The basis of this work is an idea published in previous works by Cena & Valenzo, However the idea has never been implemented or experimentally evaluated before.
Paper 9: The implementation was carried out on the CAN IP Core exclusively by the author of this thesis. The experimentation and the case studies presented in this paper is principally the work of the author, with appropriate reviews provided by the supervisor.
Paper 4: This paper is related to the analysis of Overclocking (which was implemented and published before in paper 9). The idea and implementation of the experimental analysis is done by the author of the thesis, given in section III, A of the paper. The section III, B is adapted from the 2nd author's previous work, the mathematical adaptation as Equation 6 and 7 in the paper is a combined work by the 1st and 2nd author.

Paper 5: The implementation and experiment done in section III of this paper is contributed by the author of the thesis.

Jitterless Communication

Paper 3 (80%) has been adapted for Chapter 7 on Jitterless communication. The work was conceived following detailed discussions with the supervisor.
The thesis author carried out the implementation exclusively on the CAN IP core, all the case studies and analysis has been done by the author of the thesis. The 3rd author of the paper assisted with the timer configuration of the ARM board used in the experiment for the case study presented in section 7.3 of the thesis and section IV of the paper. The paper and work was done in the guidance and suggestion of the supervisor.

Windowed Transmission

Paper 2 (30%) is adapted for the work shown in Chapter 8 and 9.
The idea of this work and the ideas contained in section 8.4.3. are principally the work of the first author of paper 2. The contributions of the thesis author to this work is to assist in the solution for the main theorem given in section 8.4.2. Section 8.5, 8.7 and 8.8, which covers the implementation of the idea on the IP Core (in its entirety), and the setting up of the experiment and presenting the results and analysis is contributed by the author of the thesis.
The work shown in Chapter 9 is now under review for a publication in IEEE TIE (70%), this is an enhanced version of paper 3 containing the main results of the case study. All the implementation, compilation of results and analysis has been done by the thesis author, under the guidance of the supervisor.

CAN Test Bench

Paper 10 (80%) has been partially (only section 2) adapted in Chapter 3 of this work. The section 2 of this paper discussed an early implementation of the CAN IP core, early versions of the HDL coding, the setup of the test bench and execution of the experiments is principally the work of the author. The author 2 and 3 (supervisor's) provided appropriate guidance and review on the paper.

Contents

List of Tables

List of Figures

Chapter 1
Introduction

Embedded electronics have been an integral part of modern electro-mechanical systems and the last quarter of the century has seen a tremendous growth in this field. The use of electronics as a replacement of conventional hydraulic and mechanical systems has seen an enormous growth in recent years. The use of embedded processors is now an integral part of the automobile and aircraft industry.

With the varying complexity of computing requirements in these applications, the embedded systems do not remain confined to a single computer or node. In some cases the overall system is itself distributed in to several nodes hence requiring separate computing units at each node. For data exchange in a communication network, these distributed processors need to be linked. Just as a LAN connects general purpose computers [Fli83], the control network made of these embedded processors are connected through a communication protocol designed specifically for distributed embedded systems.

The design of a communication protocol for distributed embedded systems requires certain key questions to be answered. The general requirement of a typical communication protocol design are the topology, bit rate, arbitration scheme and length of the network. Certain requirements, although are part of other data networks but have stricter benchmarks when an embedded communication protocol is designed. Some of these requirements have been listed in [Par07] and [UK93].

1. The protocol's predictability to transmit a message in a given time or worst

case response time.

2. Does the protocol support time triggered or event triggered messages or both?

3. What are the error confinement and correction methods? Unlike other networks, some embedded applications have zero tolerance in case of an error or unsuccessful transmission.

4. The sequencing and synchronisation of data messages is therefore required to make an interrelated logical sequence.

5. One of the major requirements of an embedded network protocol is to support periodic and short messages, in contrast to, computer communication where the trend of the data traffic is mostly aperiodic and spasmodic.

6. With the expansion of the internet and high speed data networks, modern embedded communication protocols must be robust enough to communicate and keep in pace with these protocols, as a typical factory communication setup is made of diverse communication mediums and protocols [RTE07].

The motivation of this work is to study and improve one of the widely used embedded protocol, known as Controller Area Network (CAN). Controller Area Network [11893] has been the most widely used protocol in embedded networks. Since its advent in the early 1980's CAN has been a huge success especially in event triggered applications, and is still being used in large numbers. (An estimated 800 million CAN chips are to be sold in 2010 [CiA06].) CAN has been used in different applications but its most popular utilisation has been in automobiles where it is used in body electronics (SAE Class B) and in industrial applications without any strict timing

requirements. Some of the features of the CAN protocol which make it a popular choice for embedded networks are discussed below:

1. It is a serial bus multi-master communication protocol, which broadcasts to all the participating nodes.

2. It has a non-destructive bit wise CSMA/CA medium access scheme.

3. It uses NRZ encoding for representing the information at the physical layer.

4. Shorter messages with low overheads.

5. Acknowledgement based guaranteed message delivery.

6. Automatic re-transmission of failed messages and also messages which have lost arbitration.

7. Effective error detection and confinement scheme at the physical and data layers; distinction between intermittent and permanent failures.

8. Priorities in transmission can be set by using the low binary ID's for the nodes to be given priority.

9. At the physical layer, CAN uses a differential wire high-speed network system that can reach a throughput of up to 1 M bits/sec.

10. Formal ISO standards are available for the protocol specification and its conformance hence making it a popular choice for manufacturer's.

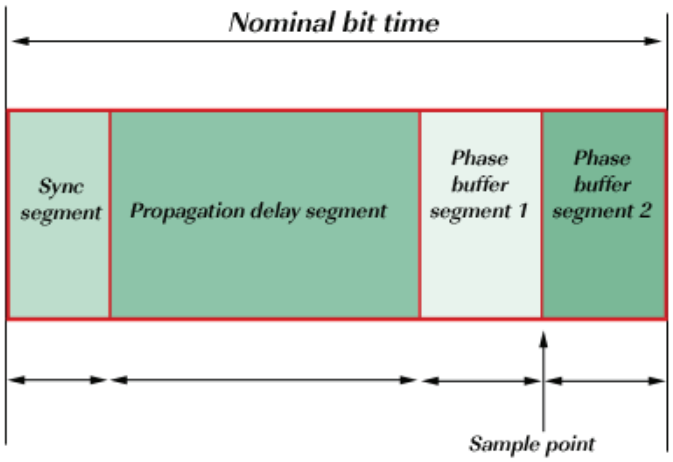


Figure 1.1: Nominal CAN bit segments. (This image have been used with permission from CiA [CIA])

1.1 Problem Formulation

CAN provides features which meet the requirements of a distributed communication protocol and priority-based transmission, with an excellent error confinement mechanism, but it fails mainly to provide features such as time-triggered communications or supporting hard real-time systems.

Recently, much work has been done towards improving CAN to adapt for real-time applications; rather an enhanced version of CAN for time-triggered applications has been developed known as TTCAN [LH02] which adds an extra session layer to the already existing CAN physical and data link layer specification. There will be a detailed look at the working of the TTCAN and other protocols in the coming chapters, but the focus, here is on the limited capability of CAN to attain higher speeds and also its limited capacity to support time-triggered applications.

1. CAN is a serial bus protocol, hence multiple nodes may compete to get access to the channel at the same time. The CAN bus contention scheme is based on

a non-destructive bit wise arbitration scheme using simultaneous transmit and receive mechanisms to check for the bit level (a 0 is represented by a dominant level and a '1' by recessive) prevailing on the bus. During arbitration, when more than one node is competing to access the bus, if any of the transmitted bit is recessive and it receives a dominant bit, then the node seizes to transmit and changes to receiver mode. Hence a lower identifier means a high priority and the node with the highest priority will win the arbitration. The nodes which lost the arbitration then wait for the current transmission to finish and will then attempt again to access the bus. The advantage of the non-destructive arbitration which is CSMA/CA based, is that at least one CAN node is always successful to gain access to the bus. This is unlike the CSMA/CD MAC protocol [TBW95], in which the intended transmitters listen for the carrier on the bus and then try to transmit. In the case of simultaneous transmissions a collision occurs and the transmission is seized for a random amount of time.

The use of a non-destructive bitwise arbitration scheme brings some inherent flaws which limits the maximum data rate and length of the CAN bus. According to CAN specifications the bit time is divided into four segments; the Synchronisation Segment, the Propagation Delay Segment, Phase Segment 1, and Phase Segment 2 as shown in Figure 1.1. The sampling point lies between the two phase segments. The sampling point is used to detect the edges so as to synchronise between the sending and receiving nodes. This limits achieving higher transmission rates on the CAN bus, as the bit time needs to be shortened for higher bit rates. A shorter bit time limits the bus length due to longer propagation delay. This is an active area of interest to overcome this limitation.

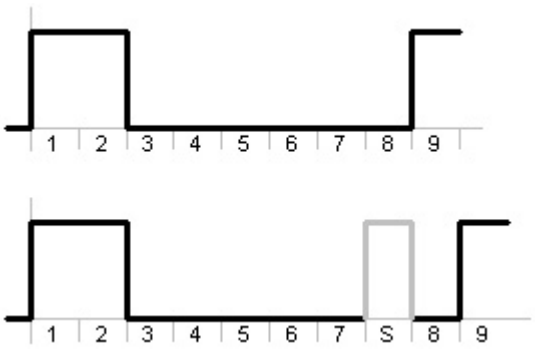


Figure 1.2: Bit stuffing.

2. The encoding scheme used by CAN for bit interpretation is NRZ-L which is a unipolar line encoding scheme where the voltage level remains the same throughout the bit duration. This uses the bandwidth efficiently in comparison to differential encoding schemes like Manchester encoding, which needs almost double the bandwidth to represent the same number of bits [Sta07]. Also NRZ- L constant bit level fulfils the requirements of the CAN protocol. The major drawback related to NRZ-L is of DC component and lack of synchronization capability [Sta07]. Also NRZ-L constant bit level fulfils the requirements of the CAN protocol.The major draw back related to NRZ-L is of a lack of synchronization capability [Sta07]. To counter this problem, bit stuffing is a method used with NRZ-L to overcome the loss of synchronisation. CAN standard specifies that if there are more than five bits with the same polarity then the sixth bit should be of opposite polarity. This is known as a stuffed bit denoted by 's' in Figure 1.2. At the receiver end this stuffed bit is removed from the received message. Bit stuffing can introduce a jitter of up to 24 bit time (worst case). This can cause severe problems when designing fixed priority scheduling in hard

real-time systems [TBW95], this problem needs to be addressed and a solution to increase the predictability of CAN message transmission times is required.

3. CAN message identifiers determine the priority a message gets on the CAN bus. A lower identifier message gets a higher priority than other messages. Although this ensures fixed priorities to the messages it may cause unfair assignment of bus time to the lower message identifier. Even in the case of networks, where it has been tried, transmission volumes should be distributed evenly or time scheduled so nodes do not have to arbitrate for the bus access [TBW95]-[DBBL07]. Induction of any error (especially burst errors of longer duration) in the transmission will introduce retransmissions which will disturb all the calculated worst case transmission times and will also introduce a knock-on effect on the message delivery times. In this scenario messages with high priority which might be running time-critical applications will not be able to access the CAN bus and the message delivery is delayed so much that it becomes useless to transmit those particular messages. This is an interesting area of research for us which has been worked on by several researchers, but will be looking at the very simplest of ideas to introduce time guarantees in message delivery over CAN networks.

4. Conformance of any implementation of a protocol to its standards is necessary for the verification of this implementation. CAN is a protocol with functionality distributed in multiple layers especially on the data link and physical layers. The existing ISO standards [ISOa] provides a conformance specifications for the data link layer. The existing conformance methods are used for industrial-level manufacturing and not only have a quite complex procedure but are quite

expensive to conduct. Research is required on developing a robust, inexpensive and simple technique to perform conformance testing of CAN protocol without the use of complex hardware and test benches.

1.2 Main Research Question

The main research question to be addressed in this thesis is as follows. Given the inherent problems that have been identified with the CAN protocol, what is the extent to which low-level modifications of the protocol itself, implemented at the hardware level, can enhance its suitability for use in the next generation of critical systems? How close will such a modified protocol be to the original CAN protocol, and what advantages and disadvantages will it have over related (and perhaps newer) protocols? An exploration of these concepts, forms the main focus for the remainder of this thesis.

1.3 Thesis Contribution

The contributions of this thesis are listed as follows:

- Develop a IP core implementation of the Controller Area Network using a Hardware Descriptive Language HDL, to explore the extent of the modifications and enhancements that is achievable on this facility.

- Explore the conformance testing issues as they arise after the protocol level implementation is done; this is to provide sanctity to the IP core to the given standards.

- Perform an in-depth investigation to identify the existing limitation of the standard CAN protocol or the modifications already proposed.
- Explore the extent to which the IP core solutions can be used to implement any enhancements which are required to make the Controller Area Network more viable for time triggered and predictable communication.
- To experimentally evaluate the proposed enhancements
- To compare and contrast the existing and proposed approaches.

1.4 Thesis Structure

The thesis is divided into 11 chapters; the first chapter provides the introduction and motivation behind this thesis and a brief description about the contribution of this work. An outline of the remaining chapters is as follows.

Chapter 2: Current approach: predictable and time-triggered communications

This chapter will review the current trends and works done in the field of embedded protocols, especially with the point of view of predictable and time-triggered communications.

Chapter 3: Controller Area Network

This chapter will look briefly on the history of CAN, and also summarises the standard CAN protocol specifications.

Chapter 4: CAN Test Bench

This chapter discusses the formation of the design, synthesis and implementation of the CAN IP core on the FPGA and also a simple two node CAN test bed for testing

of the CAN based network.

Chapter 5: CAN conformance testing

This chapter takes the discussion from the previous chapter one step forward by elaborating the need for conformance testing of the CAN soft core developed. This chapter also features a new scheme to practise conformance testing for the CAN protocol. A few of the conformance tests conducted using this scheme are also mentioned.

Chapter 6: Overclocking to increase throughput

Chapter 6 discusses the implementation of an idea to increase the information throughput of the CAN protocol by using dual rate transmission scheme where few of the CAN fields are over clocked during a CAN frame transmission.

Chapter 7: Jitterless communication in CAN Networks

This chapter will discuss the theory and implementation behind removing the unpredictability of a CAN message transmission times by using a uniform size messages by introducing a new field in the CAN frame.

Chapter 8: Windowed transmission in CAN networks

Chapter 8 is about a windowed transmission scheme for CAN networks; this specifically deals with time bound periodic messages which needs to be received in a given time window and not outside it. Few additions to the protocol are also introduced, to deal with the time bound messages.

Chapter 9: Case Study

Chapter 9 features a case study about all the implementations proposed in the previous chapters.

Chapter 10: Discussion and Conclusion

This chapter will feature the final discussion of the goals mentioned and achieved in

this work and the improvements achieved from the proposed work in previous chapters. This chapter also presents the conclusion of this work and proposed further work.

Chapter 2

Current Approach: Predictable and Time-Triggered Communications

In about two decades, extensive work and effort has changed distributive computing from a mere research topic into a working technology. The use of distributed systems is growing at a rapid pace. The concept of distributed computing has grown into a gigantic area of interest, applications such as command and control, automotive sectors, industrial process linkages and robotics, where specifically embedded distributed computing is at high volumes.

Field buses [IS] are low-cost network infrastructures, used to connect embedded devices. The major goal of these field buses is to interconnect sensors and actuators and information transmission with optimal utilisation of resources such as cabling and mechanical infrastructures. Field bus protocols provide two types of communication: i) asynchronous and ii) synchronous. Since these protocols handle real-time information, they should be able to provide some form of determinism in network operation [Ruf02], which is necessary to ensure the real-time properties of the system. However, most of them do exhibit serious shortcomings with regard to: resource allocation, for example, the bus access to provide timely communication; the provision of high bandwidth for rapid communication; the provision of predictable communication with known message transmission times; the provision of reliable and quick testing

procedures to verify the implementation of these protocols. The availability of such services is particularly related to highly dependable, fault-tolerant and more robust distributed systems. The technique to provide these services is not an easy task but it requires solving a comprehensive set of non-trivial problems.

As discussed in the introduction chapter, we have analysed that the CAN is the prime field bus protocol has almost all the shortcomings or limitations in providing the services as discussed in the last paragraph. In this chapter, review of some of the field bus protocols like TTP, and FlexRay is presented. The focus will be to study their capabilities to provide the services, discussed above, and the modifications done to the CAN protocol is also reviewed, as this is the principal research area. Additionally a brief literature review of CAN conformance testing is given, since CAN conformance testing has been a substantial part of this work.

Therefore, this chapter is organised as follows: in section 2.1, an analysis of the traditional designs used in the embedded protocols which provide time-triggered communication. A review of determinism in message transmission delays is presented in section 2.2; section 2.3 discusses the effective through-put and data rates attained by the embedded protocols; section 2.4 will consider previous techniques related to conformance testing of protocols, especially CAN.

2.1 Time Triggered Communication in Field buses

In a real-time, control/automation system, messages are either sporadic i.e. event-triggered or periodic i.e. time-triggered. For a periodic message to be sent from a node monitoring a continuous process (engine revolution), a fixed time slot and an a priori schedule are required to accommodate these messages. Also, their messages

usually contain known data fields which contain the state of the system. Any message which can occur at any unknown instance in time with random data set/contents is an event- triggered message; for example, a brake press in an automotive system is a sporadic event and can happen at any time [Jam04].

In this section, there will be a detailed look at the work done to support time-triggered communication in embedded networks. Two of the most current and robust time-triggered protocols will be discussed first; as these protocols are capturing the market from the CAN or its derivatives. The major focus of this work is to develop CAN for time-triggered and predictable communication, so a review is also presented of the previous work done to enhance CAN for time-triggered communication. The protocol level changes, which provide a complete solution, are discussed in this chapter. CAN modifications at the application level or based on a single parameter to support time-triggered communication have also been discussed in the relevant chapters.

One of the main protocols,to support time-triggered operation is TTP [KG93], which is developed by Professor Herman Kopetz of the Vienna University of Technology. This work was then adapted by the Time Triggered Technology Company together with several alliances. TTP is generally used in the automobiles and aircrafts, although it has several versions, only TTP/C (C stands for SAE class C) which supports time-triggered communications is discussed.

TTA [MH92] is the design principle of TTP, and uses the TDMA medium access method. Time is divided into different slots, and allocated to the smallest replaceable units (SRUs) [Kar02]. A replaceable unit is an electronic module that is the smallest unit that is replaceable in case of a fault. It is connected to the TTP/C network, and

it can only access the bus for transmission in a time slot exclusively allocated for it. A sequence of SRU slots forms a TDMA round, it uses MFM (Modified frequency modulation) [KB03] for data encoding which supports bit synchronisation. Several TDMA rounds are executed repeatedly, and is called a cluster cycle. The cluster cycle is repeated throughout the life of the system. The TDMA slots are available universally to all the nodes, and message replication is performed using redundant channels, this increases the reliability of the system. The speed attained by TTP is larger than the conventional CAN. TTP can achieve higher speeds depending upon the medium of propagation; the average transmission speed is 2 Mbps. TTP uses static table-based scheduling, in this case the communication requirements are fixed in the pre-run time, and a specific schedule runs a timely exchange of messages. Notice that the communication requirements cannot be changed by the application at run time [APF02].

Another major protocol which is also used in high volume and is also designed to provide time-triggered communications is FlexRay [Fle04]. FlexRay originated from the formation of a group/consortium whose initial task was to plan a exhaustive technical analysis of the protocols existent at that time to be used in the car industry, for example CAN, TTP/C, TTCAN e.t.c. The group came out with the following concerns related to these protocols: The available bandwidth for e.g. 500 kbps will not support future applications and also the non-availability of redundant channels, global time base and bus guardians in TTCAN and CAN networks. Although TTP/C provides a solution for most of the above mentioned concerns it has no flexibility for asynchronous transmissions or multiple slots for a single node in its TDMA cycle. The other concerning matter was that the contents of the TTP frame were found to

be too small.

The major enhancement in FlexRay is that the time of a message round is divided into static and dynamic segments. In the static segment only messages already scheduled can be sent. A message where the time is crucial to the operation (such as orders to the braking system) is sent in this segment. The dynamic segment is for occasional burst transmissions, diagnostic information and adhoc messages in general. The dynamic part is limited both in time and bandwidth consumption. The FlexRay system has support for both synchronous and asynchronous frame transfer by dividing the time into these two segments. The time is externally synchronised using a fault tolerant time synchronisation algorithm at every node. FlexRay uses a TDMA scheme for accessing the communications channel, during a static segment. A bus guardian is used to prevent nodes from accessing the bus, in the time slots not assigned to it. The bus guardian is supposed to keep track of the valid access times and only allow traffic to be sent at those times. A bus guardian may only block the traffic on one channel; hence two bus guardians must be used when a node wants to use both channels; a 10 Mbps of speeds is attained in FlexRay.

The major disadvantage of the FlexRay protocol is complexity. As the infrastructure is difficult to implement with a large protocol stack, small embedded processors with 8 bit architecture and 512 code space would never consider FlexRay as an option. The cost to implement is quite high as compared to other field buses. History dictates that for the most part, committee designed protocols and stacks end up being caught up in self-serving financial entities. For that matter, if you want the official CAN specification now you also need to pay for it compared to a product

with data sheets developed by a single company (In the past, Bosch and CAN). Another disadvantage of the FlexRay protocol, is the necessity of the start-up procedure and non-reconfigurability during the runtime [War09]; also the missing support for a simple configuration of a network cycle is a bottleneck of this protocol [FA09].

FlexRay is also considered as a combination of features from byte-flight, a protocol developed by BMW [GBP00]. The byte-flight protocol is based on synchronous and asynchronous methods. It ensures a deterministic transmission delay for a known number of high priority messages and the efficient use of transmission bandwidth for less prioritised messages. This protocol is quite flexible in that if required it can work in fully synchronous or asynchronous mode. The transmission rate is quite high i.e. 10 Mbps.

2.1.1 Enhancements in CAN to support TT communications

The major work on CAN to support time-triggered communication is TTCAN [LH02]. Time Triggered CAN (TTCAN) provides higher layer functionality above the CAN protocol, as the standard CAN protocol [11893] is limited to the physical and data layers. TTCAN was designed to provide time-triggered functionality to the CAN protocol by synchronising the communication between different CAN nodes. TTCAN provides a global time-base so the messages can be sent on a given time-line. In CAN the nodes arbitrate to gain access to the physical media while TTCAN synchronised communication provides predictable access to the medium. Another basic feature of the TTCAN is that the worst case transmission times can be calculated off-line as they are not affected by the temporal conditions on the network.

To achieve synchronous scheduling in a distributed system, all activities are on a

common time base. All TTCAN nodes hold a counter, that is updated once every NTU time. TTCAN implements this time-triggered architecture and the communication schedule is calculated a priori to the system start. TTCAN uses TDMA access method for sending or receiving messages [Kar02]. The Time-Division Multiple Access (TDMA) bandwidth allocation scheme divides the time domain into smaller time slots, or time windows. In a TDMA cycle one or more of these slots are assigned to a participating node on the network to access the shared medium. In the case of TTCAN this TDMA cycle in which each node is given a time slot/s to transmit is called a basic cycle(BC). Each basic cycle starts with a reference message and finishes with the start of the next reference message. Several basic cycles are combined to make a matrix cycle. All basic cycles are the same size in the time domain, but may differ in construction. When a matrix cycle is ended the transmission scheme starts again by repeating the matrix cycle.

This can be further explained by the presence of three different types of time windows in TTCAN which make flexible time scheduling possible. (i) Exclusive time windows: these windows are assigned to the periodic messages, only one message is assigned per time window. (ii) Arbitrating time windows: this is a time window to send sporadic messages and can be assigned to node in demand. (iii) Free time windows: these are for future expansion.

TTCAN is deterministic (compared to CAN), since the limit of the (worst case) time behaviour can be determined in advance due to its pre-determined time slots [AG]. Higher bandwidth utilisation up till 60-70 % [QPFM05] can be achieved using TTCAN while CAN bandwidth utilisation is limited to a maximum of 35 % as the latency behaviour will be indefinite for higher bus loads.

From the above comparison it can be seen that apart from a few problems which are related with CAN still exists in TTCAN; although it claims to be highly time based, it does not allow retransmissions in case of failures or errors. In high EMI environments such as industrial setup or engines/ignition noise can induce high BER's in the range of 10^{-7} to 10^{-11} [FOFF04]; also in some extreme cases, BERs as high as 10^{-3} have been reported for vehicles operating in close proximity to sources of large electromagnetic disturbances [GN04], although an accurate level of EM interference cannot be predicted specially in vehicle networks. This high EMI causes loss of valuable bandwidth and critical messages can be delayed to the next time slot. If the message that is delayed is a safety-critical message, then the transmission error can compromise the safety of the system in TTCAN [AM05]. According to some authors [BBRN04, NP03, FOFF04] the effort to make CAN suitable for safety-critical applications, is not successful in TTCAN. In fact, according to these authors, the standard CAN is more robust than TTCAN. Also the physical layer has not been addressed therefore there is no improvement in the level of jitter due to bit stuffing which is the same as in CAN, and also in the time domain it fails the fail silent assumption without a bus guardian.

Another piece of work done on CAN to counter its inherent drawbacks is FlexCAN [PF04] which claims to be a combination of FlexRay and CAN; this is a time-triggered system over CAN, with an extension for safety-critical systems. FlexCAN utilises a synchronisation message to control the communication cycle sent by a primary node. The other nodes synchronise on this message and then run their local task. Also it has an embedded protocol, safeCAN, to enhance the fault tolerant features. The major problem with this architecture is scalability and also large values of jitter around 187

μsec in message transmission as shown by the authors of FlexCAN [PP07].

Several software and hardware based clock synchronisation scheme's have also been developed which include the shared clock algorithm [APSP07] to synchronise clocks on the network with the use of time-triggered ticks which are propagated on the network. It is a simple implementation with no additional hardware. This is advantageous in the case of time-triggered applications but has lot of overhead in the case of keeping alive messages send at regular intervals known as tick intervals. Another recent effort to design a hardware based clock synchronisation scheme for CAN networks; it uses an orthogonal [RNRP08] clocking scheme to the rest of the system. This implementation again addresses only a single problem associated with CAN networks, this is useful to work with another suitable application that supports TDMA based communication over CAN.

The FTT-CAN [APF02] protocol combines both event and time-triggered systems into a single repeated elementary cycle (EC). Each EC is triggered by the master, by sending a specific message. In response to this message, nodes required to transmit in the synchronous window send their traffic; message collisions are handled by the built-in arbitration mechanism of CAN. In the synchronous window, event- triggered messages can be sent as required. This protocol also allows on-line modification of the time-triggered schedule to meet varying traffic conditions in the network. The protocol was implemented in software but it will require a co-processor which is based on FPGA to have an effective implementation [MAF05]. Another problem is that the synchronous window which is dedicated for time-triggered operation of the nodes arbitrates using the CSMA/BA. Although the transmission schedule is sent in the trigger message of the EC there is a possibility that nodes with lower priorities will

not be able to transmit. FTT-CAN is also vulnerable to a single point of failure in case the master holding the traffic scheduler, hence no more trigger messages with elementary cycle schedule can be transmitted. This can be overcome by replication as stated in [RNRP+04]

CANELy is an enhanced software layer over CAN to provide node failure detection and group communication [RVA03]. Although clock synchronisation (with precision at the tens of microsecond level) is employed for these services [MP10], the system does not directly support time-triggered communication. However it has been proposed that this feature may be built on top of the enhanced layer as a system-level service.

The modifications of the CAN protocol discussed in this section, are not suitable for bandwidth-demanding automotive and other applications without attaining higher speeds [CV06].

2.2 Jitter in Transmission Times

There has been a lot of research focus on adapting CPU scheduling and feasibility analysis techniques to create systems which provably deliver all messages in a timely fashion in CAN networks. As with CPU scheduling, these topics can be broadly divided into two sub-categories; those based on static (table-driven) techniques, and those based on dynamic (non-idling) techniques.

In a statically-scheduled system, all message transmissions are created off-line and stored in some form of table (a "message map") that each node makes use of to control the instants in time that messages are allowed to be transmitted. In order to be employed successfully, each node must have accurate knowledge of the release time and the transmission times.

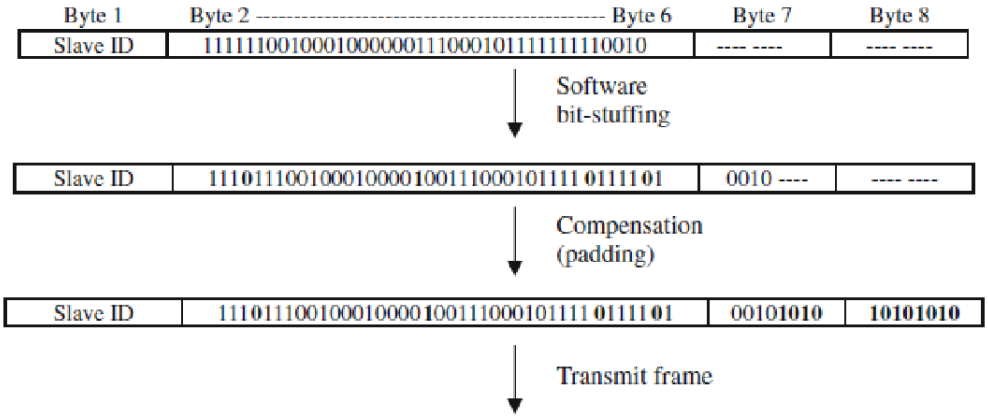


Figure 2.1: Software bit stuffing

Much work has been done and several techniques have been used to minimise the release jitter in embedded control messages. But as discussed, one of the other phenomena which can disturb the computation of the static schedule is the transmission time jitter which is a common factor in embedded protocols. The presence of such jitter can have a harmful impact on the performance of many distributed embedded systems, particularly those involving period sampling and/or data generation (e.g. data acquisition, data playback and control systems) [Tr98]. In [DS95], it was discussed that in any data acquisition tasks jitter rates of greater than or equal to 10% of the task period can introduce errors which are so significant that any subsequent translation of this signal can be considered either as an error or as useless information. Similarly, [Jer77] has discussed the serious impact of jitter on applications such as spectrum analysis and filtering. There are several factors which can effect the transmission jitter such as the bus access time, error conditions and synchronisation issues, but we will discuss another problem specially encountered by CAN due to the use of the NRZ coding scheme. CAN uses a "Non Return to Zero" (NRZ) coding technique. Since there is no synchronisation signalling involved in NRZ a drift in the receiver's

clock can occur when a long sequence of identical bits has been transmitted; this can result in message corruption. To avoid the possibility of this scenario, the CAN communication protocol at the physical level uses a bit-stuffing mechanism which operates as follows: after five consecutive identical bits have been transmitted in a given frame, the sending node adds, an additional bit, of the opposite polarity. When five or more consecutive bits of the same polarity are to be transmitted, a "stuff" bit of the opposite polarity is inserted by the transmitting hardware, and subsequently removed by the receiver hardware. Six consecutive bits of the same type 111111 or 000000 are considered as an error else an error or overload frame is signalled.

Although this mechanism is effectively transparent to the application software, a side-effect is that it causes the total CAN frame length i.e. the number of bits actually transmitted to become (in part) a complex function of the data contents. For example, for a CAN 2.0A message with an 11-bit identifier and 8-bytes of data, the minimum and maximum message lengths are 111 and 135 bits respectively. This translates to a possible variation in transmission time ("jitter") of approximately 22% of the total message length. It has previously been argued that the bit-stuffing inducement of message transmission jitter at these levels can lead to detrimental behaviour, even in statically-scheduled real-time systems ([NHN02, NSP05, NPS09]). Additionally, since message jitter is cumulative for successively transmitted frames, in non-idling priority driven systems, low priority messages can suffer from largely unacceptable jitter magnitudes.

Several software-based protocols to help ameliorate this phenomenon have previously been proposed. Nolte and colleagues ([NHN02, NHNP01]) described a simple

scheme based on applying an XOR mask to the transmitted data, which is subsequently re-applied by the receivers to recover the original frame. Although the original results seemed to indicate a measurable reduction in jitter, it was subsequently argued by [NSP05] that this effect was somewhat illusory, and results of a similar positive nature cannot be guaranteed for arbitrary datasets. Nahas [NP05] also showed that if the transmission data is screened on-line, such that an XOR mask can be selectively applied on a byte-by-byte basis, then a 20% reduction of jitter can be observed in the general case. However this comes with a penalty of increased software complexity in the embedded system nodes, and a slight penalty in terms of information throughput (a payload byte must be reserved to hold XOR encoding information).

In a subsequent paper, [NPS09] introduced a further technique known as Software Bit Stuffing (SBS). This technique applies a high-level bit-stuffing algorithm prior to data transmission, such that stuff bits are inserted by the application itself, therefore reducing the levels of hardware-induced stuff bits. The principle of SBS is shown in Figure 2.1. It was found that the technique further reduces the expected levels of jitter by 40%, but again this is at the expense of further increased software complexity and a greater penalty in terms of information throughput (2 payload bytes must now be reserved to hold XOR information). Although both techniques are successful in jitter reduction, for many applications a 25% (or even 12.5%) reduction in the available information throughput capacity of the CAN network along with increased CPU and memory overheads in each node - will be unacceptable.

Another off-line method to control bit-stuffing has been presented in [PKS07], where the identifier fields and DLC field are selected manually and XORed to make an explicit data set not to allow any combination such that more than four similar

polarity bits occurs continuously. A major drawback of this scheme is, of course, the assumption that most CAN control messages are two to four bytes while no mechanism to counter stuffing in data and the CRC field which in most cases greater than 50% is provided.

2.3 Throughput Limitations

LAN's are high speed networks with mostly indeterministic traffic requiring large amount of data to be transmitted over lengths ranging from a few metres to hundred of kilometres, hence the dynamics and design of LAN protocols are entirely different from the field bus communication, which have entirely different application requirements.

One of the major contrast between the field bus protocols is its low data rate/packet size when compared to LAN technologies. The field bus protocols require time and delivery determinism as the mandatory requirement for the design of these protocols.

FlexRay is an example which can transmit up to 10 Mbps. If we compare with CAN which can attain a maximum speed of 1 Mbps, FlexRay has a high bandwidth and low transmission times, hence making it suitable for time-triggered communications. Even in the case of transmission failures it will take almost less time to recover and resend a message as compared to a normal CAN message without a failure. One of the major reasons for the high speeds attained by FlexRay is the lack of an arbitration mechanism which is quite a critical issue in the case of CAN speed limitations. As explained earlier all the CAN nodes have to be in consensus related to bit level; it is necessary that propagation delays on the shared medium be negligible with respect to the bit time [Bos91] . Although if we look at the physical level FlexRay also employs differential NRZ encoding as similar to CAN with a difference of start/stop

bits which keep the nodes in synchronisation; also in the middle there is a byte start sequence to further improve the synchronisation between the nodes.

Previously enhancements have been proposed and implemented in CAN to increase the speed of CAN communication. FastCAN [CV00] suggests a dual channel approach to enhance the speed; the network topology of FastCAN consists of two unidirectional physical channels, here indicated as forward (FC) and reverse (RC) channels. FC is used by the nodes for frame transmission and, in particular, to carry out the arbitration phase. RC, instead, is used to receive frames and to collect the receive status information of the different nodes (i.e. error or overload conditions and acknowledgement bits). Each node is provided with two distinct input ports [forward receive (FRX) and reverse receive (RRX)] and two output ports [forward transmit (FTX) and reverse transmit (RTX)]. Neighbouring nodes are connected by means of two point-to-point unidirectional links, so that the FTX (respectively, RTX) port of a node is connected to the FRX (respectively, RRX) port of the next adjacent node in the network.

The medium access scheme used by FastCAN is very similar to the conventional CAN protocol except that the non-negligible propagation delays involved in this case is taken into account. A node can start transmitting a frame as soon as the forward channel is idle, by detecting a signal on the FRX channel. In comparison to CAN it can send messages at a maximum speed of 16 Mbps with a network length of 150 m. The major disadvantage of this protocol enhancement is the complete change of the CAN infrastructure, hence not just the application layer changes have to be made, but a complete change of physical and data layers is required.

In an another work Cena et al. in a short communication [CV99] floated an

interesting idea to use different transmission speeds for the different fields of a CAN message. A mechanism of overclocking was proposed to increase data rates in those fields of the CAN frame in which only a single node is transmitting. This idea was only suggested and but was not implemented as it requires a physical layer modification of the CAN controller. Ziermann et al. [ZT09] took the basic idea from this technique to implement overclocking to increase the data rate in CAN networks. However this implementation described in [ZT09] differs in several key aspects. Firstly, the S-zone over clocking factor must be restricted to an integer power-of-two multiple of the basic CAN rate. Secondly, although the implementation claims to be backward compatible, the behaviour of the error logic and CRC calculation is not described; brief details of the discrepancies in this implementation are presented:

1. In case of bit errors in any of the additional inserted bit, how will the traditional and overclocked CAN nodes behave? At the moment it seems there is no error management logic involved.

2. The CRC calculation is not explicitly explained; it is not clear whether two separate CRC calculations for the normal and over clocked data fields are carried out, or if the same CRC field is transmitted for both the cases. In the latter case the CRC field will not be a true representation of the over clocked data field, leaving the majority of the over clocked data susceptible to undetected errors.

3. There is no analysis presented to determine the behaviour of BER, due to overclocking. In this work effect of BER on overclocking is discussed in detail in chapter 6.

2.4 Protocol Conformance Testing

As with any complex digital system, the testing of a soft core design can be a complex and costly process [Bot98]; if the core also contains a communication protocol implementation, then conformance testing of the said protocol also becomes a requirement; this may complicate matters further. According to a recent survey by Lai, a conformance test method must be evaluated using three principles [Lai02]. Firstly, the IUT (implementation under test) behaviour must be comparable to the precise protocol definition; secondly a comprehensive test suite must be present covering all the aspects of protocol functionality. Finally, a test system which provides a controlled and reproducible environment for test implementation is required. The techniques to conduct conformance testing of CAN protocol is reviewed in the next section.

2.4.1 CAN Conformance Testing

CAN controllers and transceivers have been implemented at the silicon level, either by dedicated ICs or as on-chip peripherals of embedded devices; many such CAN controllers are widely available, e.g. [SJA00, Mic03]. In practice, as with most other silicon-based protocol implementations, the implementation of CAN conformance testers has been done using dedicated hardware and specially written analysis software, which is a practical approach when testing and verifying conformance prior to high-volume IC manufacture. ISO has developed a standard CAN conformance testing document [ISOa], and any device that wishes to claim CAN conformance is required to demonstrate that the test cases outlined in the standard have been performed and passed without problems. The ISO document not only specifies different types of tests that must be performed for conformance testing, but also specifies a

TP architecture based on the ISO 9646-1.

One of the earliest CAN prototype controllers was named DBCAN [KRWG96]. This implementation was tested using a logic analyser and a pattern generator circuit. As there was no standard for conformance testing at the time the prototype was developed, a commercial basic (as opposed to full) CAN controller was used as a benchmark for verification. A major disadvantage of this scheme was the use of external interface modules to visualise the state of different DBCAN registers, and the testing procedure was somewhat limited in the number of signal channels that could be simultaneously analysed. Since this is a requirement needed in the case of ISO standard conformance testing, the ability to visualise the state of large numbers of CAN registers simultaneously is a prerequisite, such a setup is limited in this respect.

A slightly different verification technique was reported by [NDK+05]. Their technique employed custom design boards with 8051 micro controllers and SJA1000 CAN controllers, but this method involved the design of specialised interface hardware and boards to assist with the testing plan. Specialised verification architecture for testing automotive protocols (including CAN) at both the module and the chip level was proposed by [ZCD+06]. Again, this work requires a specially designed CAN verification component as part of the silicon, while the selection and implementation of actual test sequences, along with the selection of a suitable means of monitoring bus signals, is left open for the tester.

A hardware emulation technique was used to verify a CAN soft core in [WBTMG96]; firstly, the synthesised net list is downloaded into a hardware emulator. This emulator is configured by a PC and the communication between the two is carried via a

specially designed interface card connected to the EISA bus; this emulator is also connected to two commercially available CAN chips. The drawback with this technique is that again, customised hardware along with software especially written to carry out the conformance testing is required. Additionally, to emulate the bus failures and potential error conditions on the bus, a manual technique of connecting the CAN bus to the output of individual nodes is employed, which lacks efficiency and is not robust enough to cover all the scenarios given within ISO DIS 16845.

With respect to soft core CAN implementations, the CAN e-Verification (CANeVC) test bench has previously been described [eVC05]. This commercial test facility requires a CAN specification core to be embedded in the netlist; this core then runs specific tests to verify the behaviour of the CAN soft core. Again, this technique involves time consuming development of a test bench using an expensive commercially available verification IP; additionally, compatibility issues often arise when using CAN implementations other than the proprietary implementation [BACM03], and only a limited number of programmable logic devices are supported. Another implementation to conduct CAN conformance testing has been proposed in [LCZS09]; this approach also uses the normal setup of a PC and a logic analyser with a limited number of signals at hand to monitor the testing.

A very recent and good method to conduct CAN testing has been proposed in [Nov09], although this technique is not specifically designed considering the ISO-16845 standard but can be used for CAN conformance testing with suitable modifications. This technique uses three IP functions: the CAN controller, a CAN test trigger and a CAN generator to control the bit timing and sequence of frames. This is a highly automated technique with large number of inputs variation, but the drawback is that

it needs an embedded PCI FPGA board to implement all the CAN IPs, hence this has limitations on the number of test units connected, the portability of the test suite and the length of the CAN bus.

Finally, several experimental implementations (such as that reported by [FOFF04]) to measure single parameters - such as CAN bit errors - rather than perform complete conformance testing have been described in the literature. Such implementations have typically used complex and non-trivial means, requiring customised hardware and software. In summary then, it can be observed that, to date specialised hardware and / or software has been required to assist with CAN testing plans.

Chapter 3
Controller Area Network

This chapter is dedicated to the CANs protocol specification and describes the background and history of CAN progress over the last 25 years. At the start of the 1980s, communication between embedded nodes was set up based on UARTs, but this was unable to support multi master, secure and high bit rate communication. In 1983 R. Bosch GmbH decided to develop a communication protocol to be used in their automobiles which should meet the real time requirements for its distributed embedded network. As the growth and popularity increased ISO had to start the standardisation of CAN. For the first time in 1987 a CAN integrated circuit prototype was introduced by Intel (82526), while in 1991 the current CAN 2.0 specification was released and Mercedes S series with five controllers communicating at 500 kbps was marketed. In 1995 CAN2.0B standard with high-speed communication and 29 bit identifiers was released. In 1999-2000 the time-triggered version of TTCAN began to develop and was introduced in industries in the mid of 2001. Many application layer protocols such as CANOpen, CAL, CAN Kingdom, Device net and J 1939 have also emerged to support the physical and data link layer functionality of the CAN protocol. An average of 10 to 15 CAN nodes are being used in the 65 to 67 million vehicles produced in 2008, which easily demonstrates the market volume which CAN has captured since its beginning [Par07]. Description of the CAN specifications is given in the next few sections.

	Standard	Identifier
Low-Speed	ISO 11519	11 bit
CAN 2.0A	ISO 11898 (1993)	11-bit
CAN 2.0B	ISO 11898 (1995)	29-bit

Table 3.1: ISO CAN Standards [Cor02]

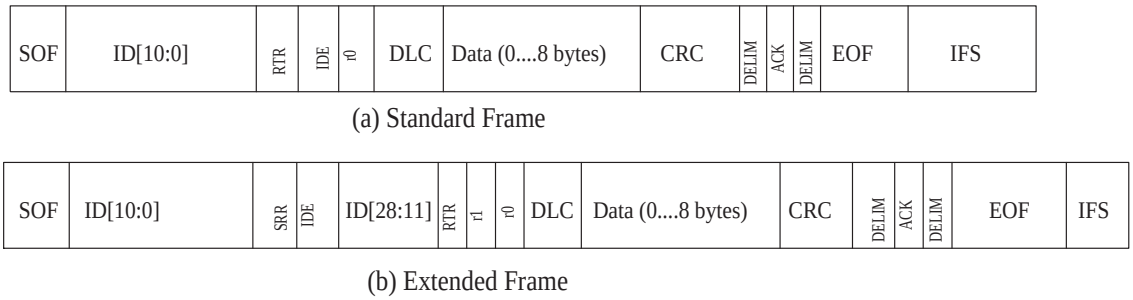


Figure 3.1: CAN frame format (a) Standard (b) Extended

3.1 Protocol

The Controller Area Network (CAN) is a serial bus protocol, which supports to inter-connects sensors, actuators, and other information processing nodes in real-time systems. In this section, a detailed description of the CAN is given, which includes the message formats, bus arbitration scheme, CRC calculation, bit-stuffing and error-handling mechanisms. The different ISO standards describing the three CAN configurations is given in Table 3.1.

3.1.1 Frame Format

The CAN specification [11893] explains the frame format. CAN supports four message types/frames. i) Data frame, ii) Remote frame, iii) Error frame and (iv) Overload frame.

CAN data frames works in two modes and the frame format varies for these two modes, i) Standard CAN and ii) Extended CAN. The meaning of the bit fields of Figure 3.1 (a) are:

- SOF: the starting dominant bit of a CAN frame is called SOF, the receiver node's synchronise with the transmitter on this dominant edge, this is called hard-synchronisation.
- Identifier: the 11 bit CAN standard identifier is used to set the priority of a CAN node on the complete network, a node with lowest identifier will have priority on all other nodes.
- RTR: when any node requires any information the rtr bit is set to dominant, normally this bit is recessive for normal data frame. The identifier of the remote frame determines the intendant node.
- IDE: a dominant IDE bit shows that the message sent has no extended identifier field.
- r0: reserved bit (for possible use by future standard amendment).
- DLC: the 4-bit data length code (DLC) contains the number of bytes of data being transmitted.
- Data: 8 byte data can be transmitted in this field.
- CRC: CRC check of 15 bit is attached in this field.
- ACK: acknowledgement field is two bit, the acknowledgement bit is recessive and the delimiter is over written by the receiver's, if they receive the message

successfully.

- EOF: this is a 7 bit recessive End of Frame field, to correctly process the received message.
- IFS: inter-frame space is 3 recessive bit field and is always inserted between two frames whether data, error, overload or remote frame.

As shown in Figure 3.1 (b), the extended CAN message is the same as the Standard message with the addition of:

- SRR: the SRR bit is a replacement to the the RTR bit of the standard frame, in an extended frame.
- IDE: a recessive ide bit indicates that this message frame has got 29 bit identifier.
- r1: an additional reserve bit in extended frame.

Another type of frame is remote frame. A recessive RTR bit indicates a remote frame, which has no date. The sole purpose of remote frame is to request data from another node represented by the identifier of the remote frame.

Another type of message frame is error frame, which is a special message is transmitted when a node detects an error in a message, the details of error message frame is discussed in the section for error detection and confinement.

3.1.2 Arbitration in CAN Bus

Arbitration is a mechanism to solve the bus access problems. Whenever the CAN bus is free, any node can start to transmit a message. There is a possibility that more than one node can start to transmit simultaneously; this conflict is resolved

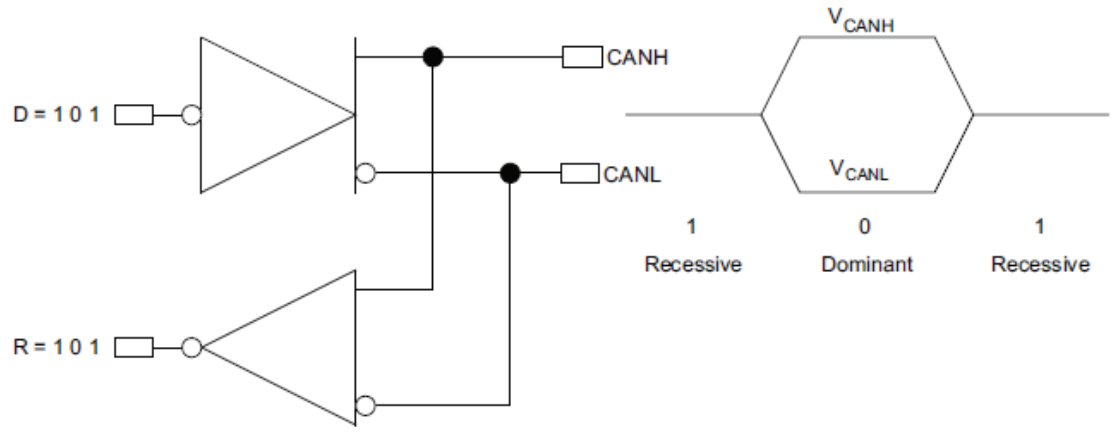


Figure 3.2: Physical signals on a CAN differential bus

by bit-wise arbitration using each nodes unique identifier. During the arbitration phase, each transmitting node transmits the identifier bit and compares it with the level on the bus. If these levels are equal, the node continues to transmit. If the node receives a dominant level, though it has transmitted a recessive level then it lose the arbitration and becomes a receiver. At the end of the arbitration field, only one transmitter is able to access the bus. The arbitration process for three nodes is shown in Figure 3.3.

In this case all three nodes start to transmit their sof signal together. Arbitration process starts after that and all three nodes keep on transmitting their identifiers till the time any node receives a dominant level. Although it has sent a recessive level, therefore according to Figure 3.3, node 1 loses the arbitration first and later node 2 also loses the arbitration, hence only node 3 remains a transmitter and all other nodes are turned to receivers.

Since all the CAN nodes connected to the bus receive every bit of the transmission,

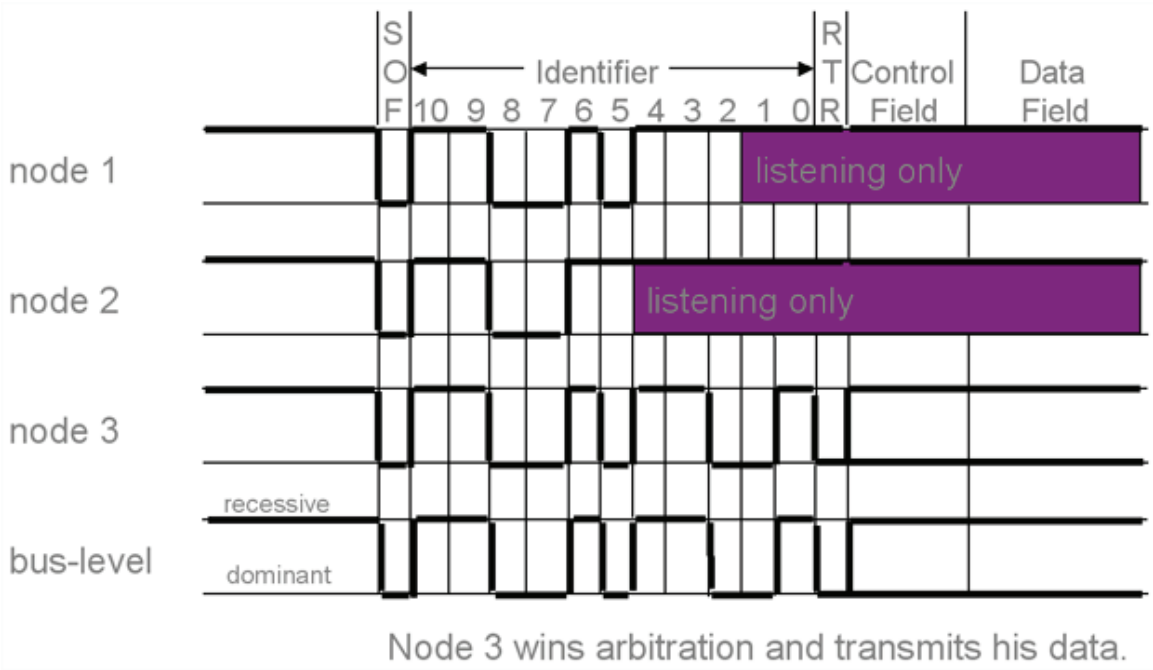


Figure 3.3: Arbitration in CAN network. (This image have been used with permission from CiA [CIA])

there is no concept of a destination addressing, rather the receivers themselves have a filtering mechanism to either accept or discard a message. Message arbitration and physical AND nature of CAN bus is one the causes of speed limitations and unpredictable transmission times. This topic is discussed in length in later parts of the thesis.

3.1.3 Message Filtering

The CAN Protocol employs a filtering mechanism to distinguish between wanted and unwanted messages. This is accomplished through the use of acceptance code and acceptance masking. The acceptance code and acceptance mask work hand-in-hand

to determine which messages are accepted. Message filtering is applied to the whole identifier. A node can optionally implement mask registers that specify which bits in the identifier are examined with the filter. Identifier acceptance registers define the acceptance patterns of the standard or extended identifier. Any of the bits in the acceptance identifier can be marked 'dont care' in the acceptance mask registers. A message is accepted only if its associated identifier matches one of the identifier filters. The identifier acceptance filter can be programmed in different modes for the CAN IP core, four 8-bit wide acceptance code registers (ACR0, ACR1, ACR2 and ACR3) and acceptance mask registers (AMR0, AMR1, AMR2 and AMR3) are available for a versatile filtering of messages. It operates in one of the three modes: Two 32-bit identifier acceptance filters; four 16-bit identifier acceptance filters; eight 8-bit identifier acceptance filters.

3.1.4 Error Detection and Confinement

Error detection and error confinement are the highlights of the CAN protocol. Error detection in CAN is very robust, because the CAN bus is monitored for all times by all the connected nodes. Hence the probability of missing an error is really small in CAN networks. There are five type of errors defined in CAN:

- Bit errors are easily detectable in CAN due to continuous monitoring, a bit error is one in which the level monitored on the bus is different from the level transmitted. The bit errors are disabled for identifier and acknowledgement delimiter fields.

- if a stuff bit is not detected after 5 consecutive bits, then this is classified as a stuff error (error, overload frames and intermission field are exceptions).

- CRC error: if the calculated CRC on the receiver does not match to the attached CRC check of a transmitted message then this is a CRC error.
- When a different bit pattern of a fixed format data or error frame is detected, this is taken as a form error (e.g. an end of frame is always a 7 bit recessive signal, if a different format is received then an error is signalled).
- In case of a synchronisation failure and unsuccessful reception of a message, the receiver does not send a dominant acknowledgement delimiter; this is taken as an acknowledgement error.

Whenever a node detects an error it starts to transmit an error frame and it does not limit to a previous transmitter, but all the receiver nodes will also signal in case to keep the synchronisation and also to signal any node which has missed the error condition. In comparison to other network protocols, the error recovery mechanism of CAN is quite robust and faulty messages are retransmitted in a short time. This mechanism has certain drawbacks in case of timely message reception applications where continuous errors on the bus will lead to large number of re-transmissions, hence a particular node can occupy the bus for a longer time.

Apart from error signalling CAN also implements an error confinement mechanism, this is implemented using different error counters which increments or decrements with unsuccessful and successful transmissions/reception simultaneously. Threshold values are set to move the node into different states:

- Error Active: if any of the transmit or receiver error counter is less than a passive error state, this allows the node to transmit and receive as well.
- Error Passive: if any of the transmit or receiver error counter is greater than

the error passive threshold and less than the bus off value, the node unit will wait before initiating a further transmission.

- Bus off: if the error counters are equal to a bus off threshold than the bus is off and will seize to transmit.

3.1.5 Physical Signalling

The physical signalling of the CAN protocol is of importance as it not only specifies the actual bit transmission on the bus but also distinguishes CAN as CSMA/Collision avoidance mode of medium access. A complete CAN system comprises of a CAN controller and a transceiver. Connection to the physical layer is made through a line transceiver such as [Phi96b]. The CAN signalling is differential which increases immunity to noise and fault tolerance.

Balanced differential signalling reduces the effect of noise on the transmitted bit levels, as the bits are not determined by a single level but a difference of two levels hence not only providing a medium redundancy but a margin level to detect any distortion of the original transmitted levels. Also balanced differential signalling means that the current in each wire flows in a different direction to keep the noise levels low. The use of the High-Speed ISO 11898 Standard specifications are given for a maximum data rate of 1 M bps for a bus length of 40 m with maximum 30 nodes on the CAN bus. Typical bus lengths for different CAN transmission rates are given in Table 3.2.

The communication cable is either a shielded or unshielded twisted-pair with 120-Ω characteristic impedance. The two signal lines of the bus, CAN_H and CAN_L (Figure 3.2), is normally in recessive state, and are biased to 2.5 V. The dominant

Bit Rate (kbps)	Bus Length (m)	Nominal Bit time μsec
1000	40	1
500	100	2
250	250	4
125	500	8
62.5	1000	20

Table 3.2: Typical bus lengths for different CAN transmission rates

state on the bus switches the CAN_H, 1 V higher to 3.5 V, and switches CAN_L 1 V down to 1.5 V, creating a 2-V differential signal. This explains the major protocol components of CAN. The next chapter discusses the set up of a CAN test bench and will discuss some other aspects/specifications of the CAN protocol.

This explains major protocol components of CAN. The next chapter discusses about the set up of CAN test bench and other aspects/specifications of the CAN protocol.

Chapter 4
CAN Test Bench

The aim of this thesis is to propose and implement enhancements in the existing CAN protocol to provide higher speeds and reliable communication. This chapter describes and discusses the design of a flexible CAN IP core, which is a key requirement if the aims stated above are to be achieved. In addition, the set up of a test bench based upon this developed CAN IP core, with details of a simple case study undertaken to verify the basic functionality of the core, is described.

The CAN protocol is a well established standard and is available in hard wired Integrated Circuits(IC). These ICs are available as stand-alone chips [SJA00, Mic03] [1] which can be used as discrete components in CAN network designs. Alternately, CAN controller circuits are also commonly available as on the chip peripherals of micro controller or DSP cores [Phi04, TMS97]. These off the shelf CAN controllers are made under the CAN specification and comply with the conformance testing standard. Although these CAN controllers are relatively easy to use when setting up CAN networks, they lack the flexibility to allow any modifications or enhancements to be made to the CAN protocol at the hardware level.

The set up of the test bench based on the CAN IP Core described in this chapter provides for a very flexible and novel experimental platform. This platform can be

[1] A comprehensive guide on CAN standalone and on chip controllers can be found at http://www.kvaser.com/en/about-can/can-controllers-and-transceivers/71.html

used to implement, to test and to verify any modifications to the existing CAN protocol under tightly controlled conditions. The changes to the protocol were necessary to make the intent of this project realisable. There was a need to find a flexible and robust solution, which allows not only modifications at the protocol layer but also supports field tests with the help of standard interfaces. A cost-effective answer to all these requirements was to have the protocol implemented in the form of a silicon IP core, which is robust, flexible and easily modifiable. This core can be implemented on a re-programmable device such as a CPLD or FPGA. Hence the formation of the CAN IP core provides us with a basis to experiment and implement the proposed changes. The evolution of the CAN IP core will be discussed in the succeeding sections of this chapter.

4.1 CAN IP Core

The CAN IP core was implemented in Verilog [Ver06] HDL(Hardware Descriptive Language). Two CAN standard documents [11893, Bos91] were used as the main source of reference for this implementation. Verilog was chosen as the implementation Hardware Description Language (HDL) due to its C-like structure which is fairly straightforward in comparison to other languages such as VHDL [Smi96].

Figure 4.1 shows the basic components of the CAN IP core. This section will discuss each of the main functional blocks. These functional blocks are as standard, and can be found in many of the commercially available CAN controllers [SJA00, Mic03]. The blocks have also been discussed in great length by Farsi et al. in their book [FRB99]. The developed IP core was made to be as close a match as possible to a typical silicon implementation, and appears to host CPU as a number of special

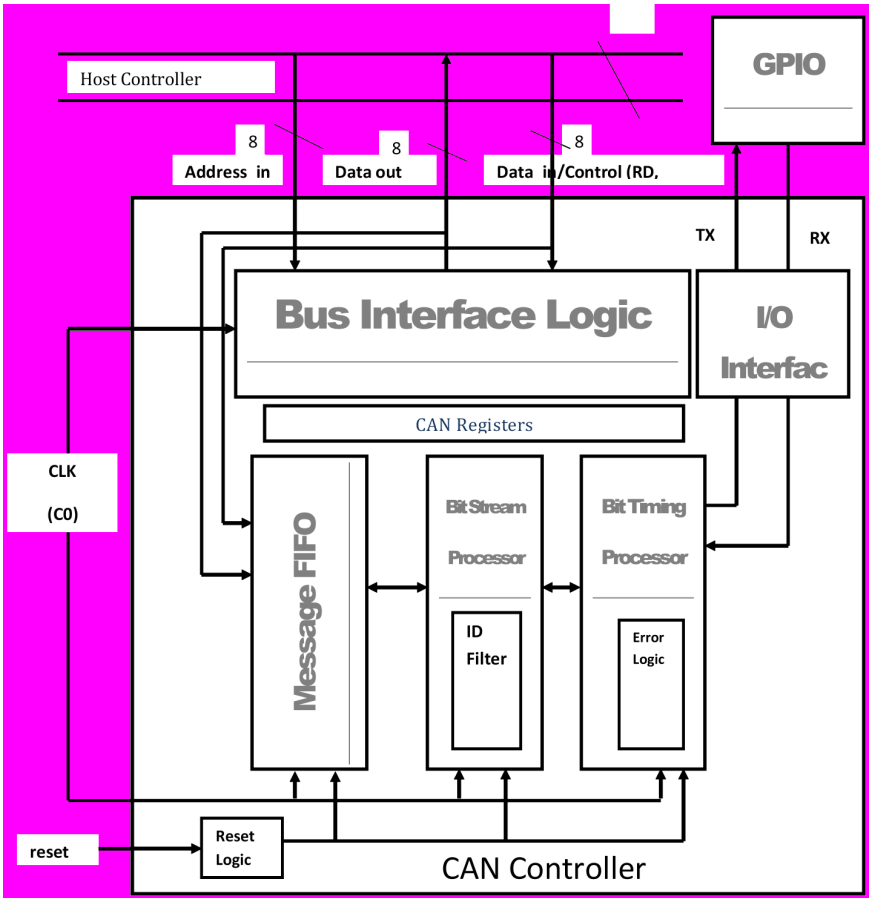


Figure 4.1: CAN block diagram

function registers accessed via a simple bus interface. The following sections present a detailed description of the principal functional blocks making up the CAN controller.

4.1.1 Bus Interface Logic

The bus interface logic interprets commands from the host controller and produces relevant addressing, data input of the CAN registers and reads back information to the host processor.

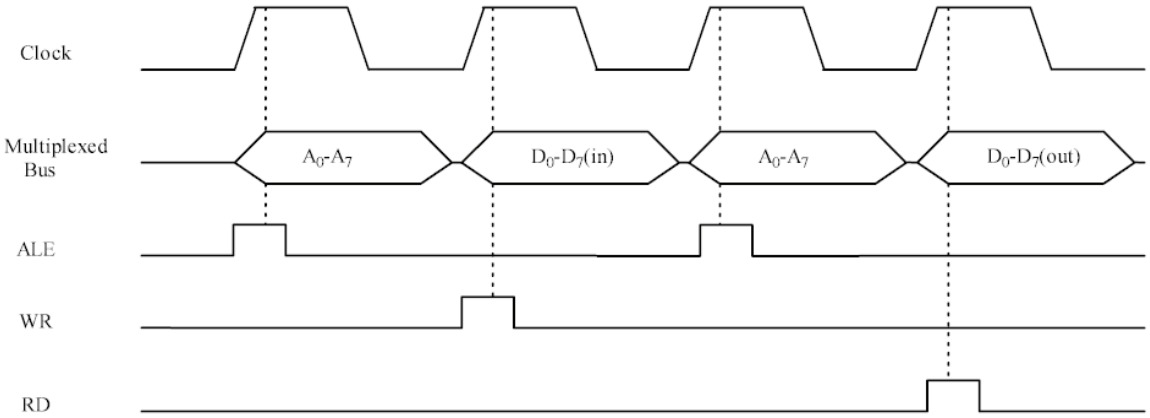


Figure 4.2: Multiplexed bus timing

The bus interface logic acts as the principal interface between the host processor and the CAN controller. Its main duties include the interpretation of commands from the host controller, decoding of addresses and handling data transfers to and from the CAN registers. Two different versions of this silicon IP core were designed. In the first instance a simple multiplexed bus and secondly a fully-featured Serial Peripheral Interface (SPI) logic interface. For clarity of presentation, the thesis will consider only the simple multiplexed bus version which is as shown in Figure 4.1.

The ALE (Address Latch Enable) signal is used to multiplex both address and data on the bus. When data are to be transferred from the host controller to the CAN silicon IP core, the required address is first latched with the ALE signal set high. Data is then written on the multiplexed bus by setting the WR (write) signal high. When data is to be read from the silicon IP core, again the required address is first written on the multiplexed bus with the ALE signal held high. Once the address is latched, the RD signal is set high and the data is placed on the bus by the core, and read by the host controller. The timing diagram of these sequences of operations

Register Name	Function	size (bytes)	Status
Mode	Setting between normal and configuration modes	1	WR only
Command	Transmit request and release receive buffer	1	WR only
Status	Transmit , Receive and Error status is stored in this register	1	RD only
Interrupt	Transmit, Receive and Error interrupts are stored	1	RD only
Acceptance code	Stores the Filtering code for message identifiers	1 to 4	WR & RD
Acceptance Mask	Stores the And mask for Filtering	1 to 4	WR & RD
Timing Register	Stores the value of Baud rate pre scalar	1	WR only
Segment Registers	Store the bit timing segment values	1	WR only
Transmit Buffer	Transmit data and identifier are stored	11 to 13	WR only
Receive Buffer	Received message is stored in this buffer	11 to 13	RD Only
Error Capture	Type of Error and direction of error	1	RD only
Tx Error Counter	Transmit error count	1	RD & WR
Rx Error Counter	Receive error count	1	RD & WR

Table 4.1: CAN registers and their functions

is depicted in Figure 4.2.

4.1.2 CAN Configuration Registers

This functional block contains the configuration and data registers of the CAN IP core. The registers contain configuration information, which the host CPU can use to set up the configuration and run-time operation (the sending and receiving of messages) of the CAN network. The registers declared in this functional block include those employed for command, status, timing, error and transmit/receive operations. Bus interface logic explained in the previous section drives the address translation logic and appropriate signals to read and write data from and to the host controller. The registers are either updated synchronously or asynchronously, according to their configuration and the CAN specification requirements. Table 4.1 lists all of the configuration registers inside the CAN IP core and their functions.

4.1.3 Bit Stream Processor (BSP)

The BSP is the sequencing logic and main state machine which controls the data stream between the transmit buffer, message buffers and the bitwise signals appearing on the CAN-bus itself (via the transceivers). This implementation of the CAN IP

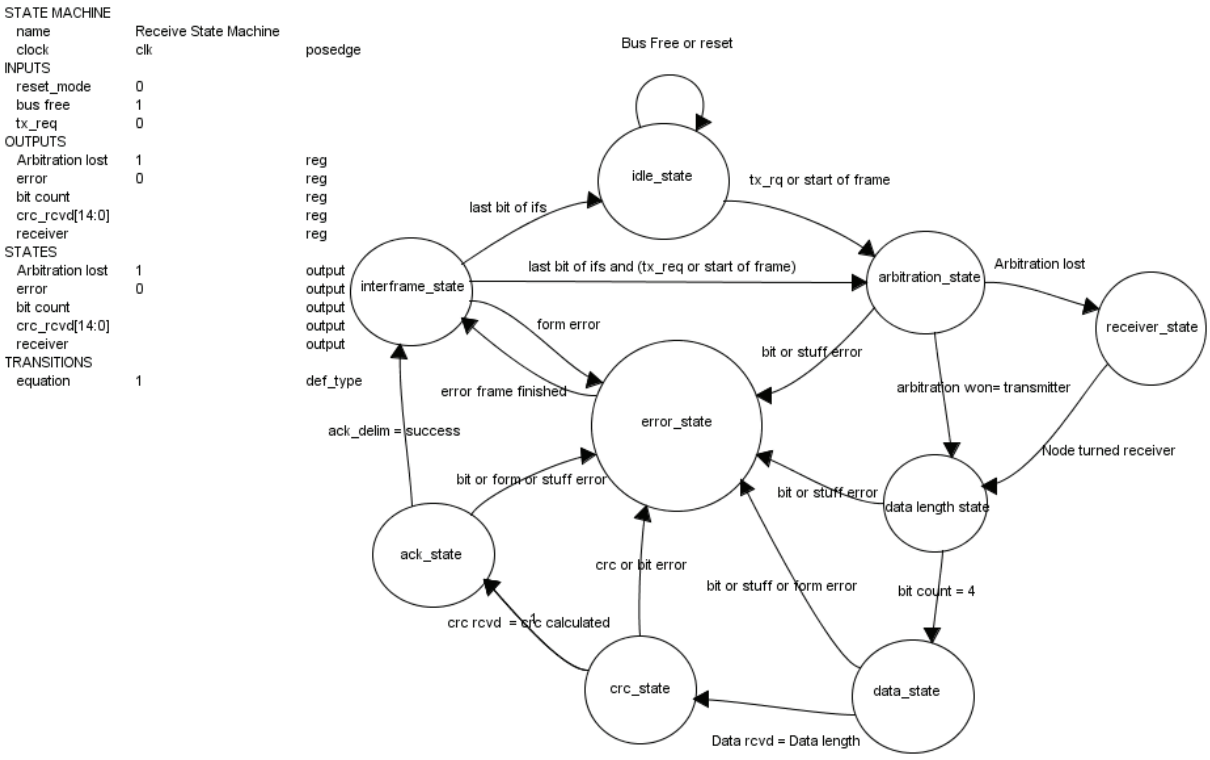


Figure 4.3: CAN message state machine

core follows the reception/transmission state machine according to the CAN bus frame fields as given in Figure 4.3. Section 3.1.1 defines these states in detail. The BSP also contains the required logic and state machines to perform error detection, arbitration, bit stuffing and error handling on the CAN-bus. The next sub-sections discuss the main elements of the BSP functionality in more detail.

Bit Stuffing and De-stuffing

This block performs bit stuffing according to the CAN specification [Bos91], i.e. preventing the transmission of more than five bits of similar polarity in a data or a remote frame. When such a situation is detected in the master bit stream, a bit of opposite

polarity is automatically inserted (stuffed) to break the sequence, in order to keep the CAN nodes synchronised, as CAN uses NRZ encoding for data communication.

The implementation of bit stuffing in the CAN IP core is done on the complete frame. First of all, the fields from the Identifier to the CRC field are serialised and are then checked for stuffing. Where applicable, stuffed bits are appended to the message in the appropriate places and it is then considered to be ready for transmission. On the receiver side, the bits are serially checked for the stuffed bit, and where detected they are removed. The process to remove stuffed bits is called bit de-stuffing. The stuffed message is first stored in a temporary register before de-stuffing. Bit de-stuffing starts with the CRC state. From this temporary register, the message is shifted serially to another register *de_stuff_reg*. Once a stuff bit is detected, this bit is overwritten by the next incoming bit. Once complete, the de-stuffed message is then forwarded to the message filtering module for further processing.

Error Logic

The Error Logic block is responsible for the error confinement of the transfer-layer modules. In the silicon IP core, the error logic module is a dedicated part of the BSP. The BSP detects the errors and in turn increments the error counters in accordance with the rules set out in the CAN specification. There are separate registers for transmission and reception error counters (TEC and REC). The counters help the CAN controller to switch between the different error states (discussed in chapter 3), according to the state diagram shown in Figure 4.4. The error passive and bus off states are a measure for the error confinement. Once a transmit error count is greater than 255, the bus goes into off state. The error status is also updated in the error

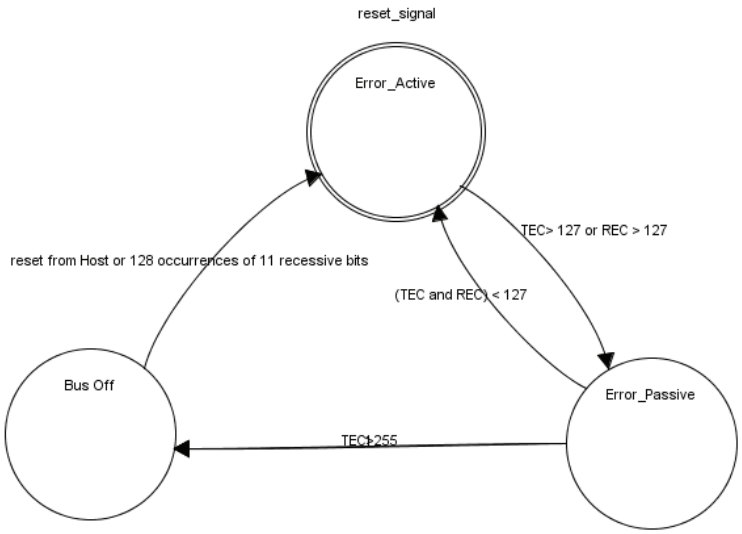


Figure 4.4: CAN error state machine

status register and can be read by the host controller. As shown in the Figure 4.10 'interrupt' signal can be used to interrupt the host controller in case of an error or change in error state, such as bus off. The host controller can send a 0 count to the TEC, to switch the CAN node from bus off to active error state.

CRC Block

The Cyclic Redundancy Check (CRC) is a common method of detecting errors in a block of data [RG88], which operates by appending some additional (redundant) information to the data being protected. This redundant information is then used to calculate the integrity of the received data by the receiver. A generator polynomial is employed by the transmitter to create the redundant information, and this operates by dividing the message by a specified shift polynomial, with the remainder appended to the original data frame as the CRC. At the receiver side, the same procedure is

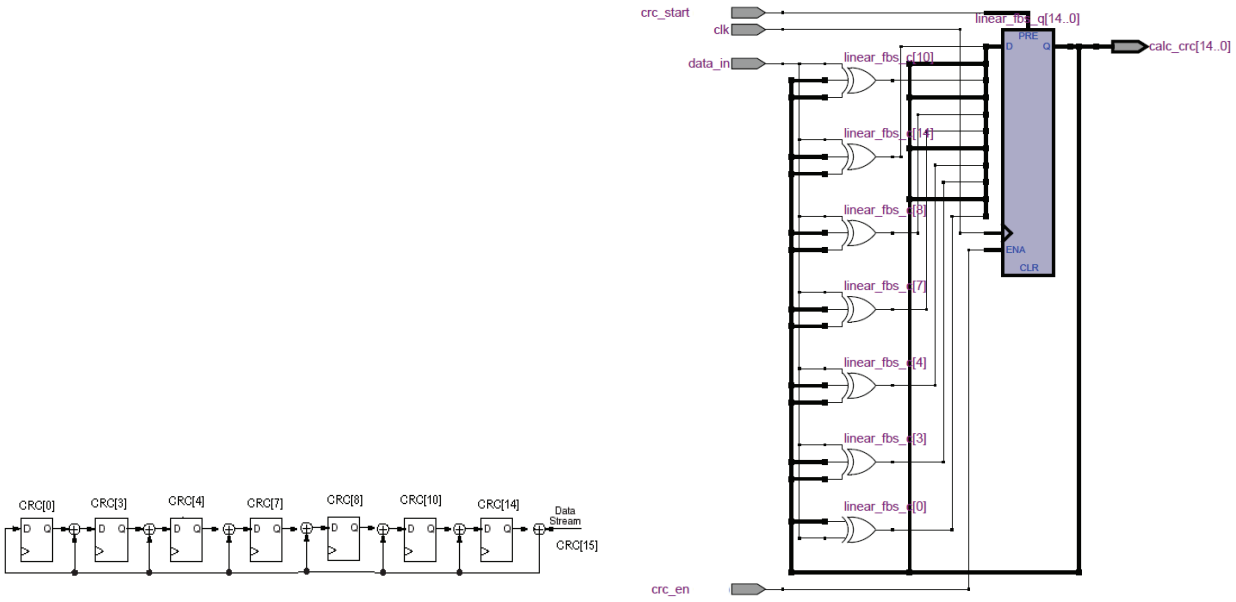


Figure 4.5: (a)15 bit LFBSR block (b) CRC RTL schematic

applied on the received data and compared with the calculated CRC value. In CAN, a message is accepted if and only two values match, otherwise an error flag is generated to inform the transmitter of an error condition. No error recovery is attempted. CAN employs a 15 bit CRC code, which is generated by the following shift polynomial:

$G(x) = 1 + X_3 + X_4 + X_7 + X_8 + X_{10} + X_{14} + X_{15}$

The CRC check is a logical process which can be implemented by a dividing circuit, this can be realised by an XOR gate and a shift register. A Linear Feed Back Shift Register (LFBSR), is mainly used for implementing a dividing circuit. A LFBSR has two main parts: serial data is shifted with feedback from the output, using a given function. A 15 bit LFBSR block is shown in Figure 4.5(a). When calculating the CRC, the input to the shift registers (which is given by the polynomial generator) are XORed with the output of the last shift register. The RTL schematic of the CRC calculation for the CAN silicon IP core is given in Figure 4.5(a). When calculating

the CRC, the input to the shift registers (which is given by the polynomial generator) are XORed with the output of the last shift register. The RTL schematic of the CRC calculation for the CAN silicon IP core is given in the Figure 4.5(b). A verilog code listing of the CRC implementation is given below:

```
assign calc_crc = lbfs_q;
 // Shifting and XOR operation using the Generated polynomial G(X)
    lbsf_q [0] = lbfs_q[14]^data_in;
    lbfs_c(1) = lbfs_q(0);
    lbfs_c(2) = lbfs_q(1);
    lbfs_c(3) = lbfs_q(2) ^ lbfs_q(14) ^ data_in;
    lbfs_c(4) = lbfs_q(3) ^ lbfs_q(14) ^ data_in;
    lbfs_c(5) = lbfs_q(4);
    lbfs_c(6) = lbfs_q(5);
    lbfs_c(7) = lbfs_q(6) ^ lbfs_q(14) ^ data_in;
    lbfs_c(8) = lbfs_q(7) ^ lbfs_q(14) ^ data_in;
    lbfs_c(9) = lbfs_q(8);
    lbfs_c(10) = lbfs_q(9) ^ lbfs_q(14) ^ data_in;
    lbfs_c(11) = lbfs_q(10);
    lbfs_c(12) = lbfs_q(11);
    lbfs_c(13) = lbfs_q(12);
    lbfs_c(14) = lbfs_q(13) ^ lbfs_q(14) ^ data_in;

always @ (posedge clock)
begin
```

```
  if(crc_start)
    lbfs_q <= 15'h7FFF; // initialize the lbfs_q register
  else if (crc_en)
    lbfs_q <= lbfs_c; // store the calculated crc in a the lbfs_q register
end
```

The CRC used in CAN protocol is of type ARQ (Auto Retransmit Request) [RG88]. When a CRC error is detected, a retransmission request is automatically forwarded to the transmitter using the error frame. CRC is used in data communication networks because of its capability to detect burst errors. A n bit CRC check is capable of detecting all burst errors that effect odd no of bits, all burst errors of $n-1$ bits and a high probability of detecting all burst errors of length more than n [Sta07]. CAN uses CRC-15, which is capable of detecting patterns of 1, 2 and 3 bit errors and detecting burst errors of 14 or more bits.

Message Filtering block

CAN bus is a shared medium, and all the bus activity is monitored by all the nodes connected on the bus. A message transmitted is received by all, but in most situations it is not necessary that every received message is of interest to every participating CAN node, and therefore some messages need not be forwarded to the application layer. The message identifier not only signifies the priority of a message, but also indicates its data contents. A filter block at each node is used to decide which of the received messages is of interest, and should be buffered and forwarded to the application layer. The CAN filter block performs the message filtering, and consists of match and mask filter registers. The messages are filtered by the combination of

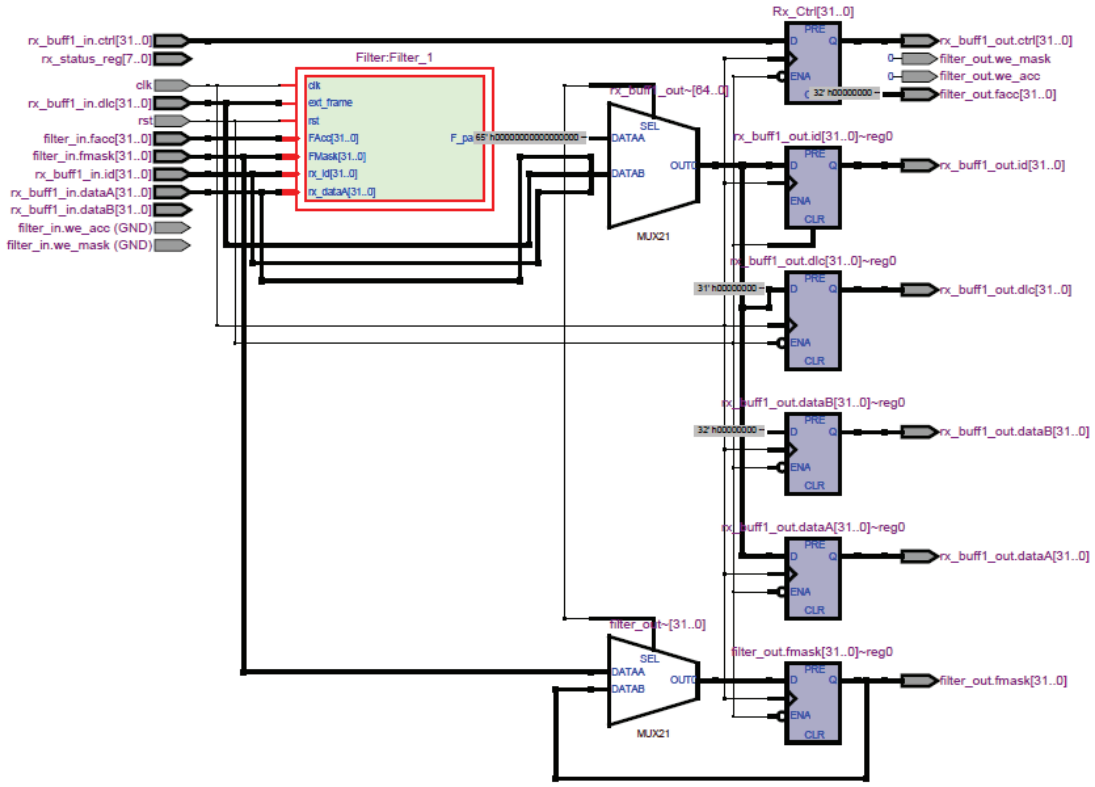


Figure 4.6: RTL schematic of filtering block

match and mask registers. The mask registers specify which bits in the identifiers are to be examined, and following this masking, only those bits of the identifiers which match with one of the match registers, are retained.

The filtering process evaluates the following binary expression when accepting a message. The equation given below is for a n_{th} bit of the filter:

$$Filter_{Pass}(n) = ((NOT(id(n)XOR(FAcc(n)))or(NOT(FMask(n))))$$

where

id(n)= nth bit of the identifier.

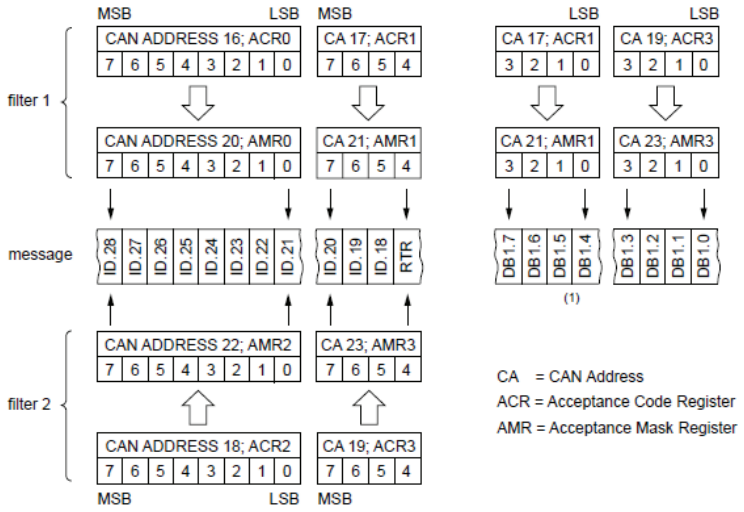


Figure 4.7: Dual Message filtering, adapted from [SJA00]

FACC(n) = nth bit of the filter match.

FMASK(n) = nth bit of the filter mask.

All the $Filter_{pass}(n)$ bits)(for all n) are then ANDed together to complete the filtering process. Figure 4.6 is a schematic of the RTL implementation of the CAN IP Filtering block. The CAN filtering process supports three different configurations. These filtering modes use the four 8-bit wide Acceptance Code Registers (ACR0, ACR1, ACR2 and ACR3) and Acceptance Mask Registers (AMR0, AMR1, AMR2 and AMR3) in different combinations to make the message filtering. It operates in one of the three modes:

1. A Single 32 filter for a standard identifier: 11 bit identifier and 2 data bytes have to pass the combination of filters.

2. A Single 32 filter for an extended identifier: 29 bit identifier and RTR bit have

to pass the combination of filters.

3. Two 16 bit filters: The filtering process is elaborated in the Figure 4.7. The received message is passed through both the 16 bit filters, and if it passes any of the filters, the message is accepted.

Message FIFO

When a message has been received error-free and accepted by the filter, they are stored in the message FIFO. The purpose of this FIFO is to buffer messages in order of receipt, such that they can be read by the host controller asynchronously i.e. as the controller is still handling the receipt or transmission of other messages. In the CAN IP core, the implemented FIFO length is 128 bytes. This can be used to store up to 9 extended CAN messages. Note that the FIFO can potentially be extended arbitrarily and that the size is only effectively bounded by the available RAM. The CAN IP core uses Xilinx 8 x 8 RAM to implement the FIFO. Figure 4.8 shows the block diagram of this storage device. The code listing to follow shows the initialisation of the RAM module.

```
RAMB4_S8_S8 FIFO
  (
    .DOA(data_out), // data out from FIFO
    .ADDRA({2'h0, wr_pointer}),
    /* wr_pointer is 7 bits to support 128 addressing locations */
    .CLKA(clk),
    .DIA(data_in), // One byte data in
    .ENA(1'b1), // Only RAM in system, enable = 1
```

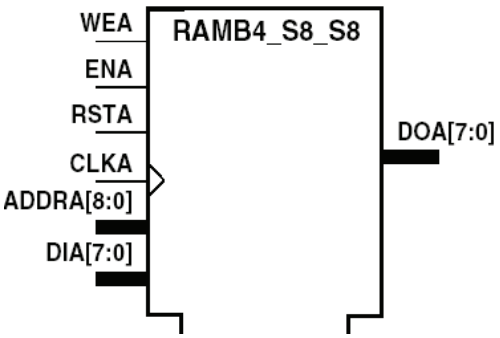


Figure 4.8: Xilinx 8x8 RAM [Xil08]

```
    .RSTA(1'b0),
    .WEA(wr & (~fifo_full)),//Write enable, to write data if FIFO not full
  );
```

4.1.4 Bit Timing Logic (BTL)

The BTL is employed to convert the logical output of the BSP into the logical and temporal (sub-bit) stream required of the CAN-bus, and vice-versa. This block is the last and direct interface between CAN controller and transceiver. The node synchronises to the message bit stream on dominant edges at the start of a message (hard synchronisation), and intermittent resynchronisation takes place on dominant edge transmissions during the reception of the message. The hard synchronisation and resynchronisation identifies dominant edges and then synchronises the node to the rest of the CAN nodes, with the bit stuffing mechanism ensuring that resynchronisation occurs at regular intervals. Verilog code listing to handle *recessive-to-dominant* edge is given below:

The highlighted code shows the phase segment starts immediately in case of hard

```
assign hard_synchronize  = (idle_state | ifs_state)&(~RXCAN)& sampled_bit;
assign resynchronize     =  (~idle_state) & (~ifs_state) & (~RXCAN) & sampled_bit;

// phase_seg starts as soon as there is a dominant edge, synchronization window is skipped.
// synchronization or hard synchronization

assign phase_seg = (resynchronize_q | hard_synchronize) | sync_seg));
```

Figure 4.9: Synchronisation on dominant edges(Code listing).

or resynchronisation, which will synchronise the receiving node to the transmitting node. The BTL also provides programmable phase and propagation segments. As discussed further in chapter 6, these segments provide compensation for the propagation delay and any phase shifts occurring in the physically distributed network. The CAN bus is sampled once a bit time, at the so-called sample point which lies at the intersection of the two programmable phase segments. The calculation of bit timing parameters is vital for the working of the CAN network. An example calculating these parameters is given below.

Example of Bit Time Calculation

Calculation of CAN Bit time parameters for the following system constraints:

Bit rate = 1 M bit per second and Bus length = 20 m

Bus propagation delay = $5 \times 10^{-9} m/s$.

Physical Interface (PCA82C250)[Phi96b] transmitter plus receiver propagation delay = $150ns$.

MCU oscillator frequency = 48 MHz (Clock Frequency of FX2 on the KNJN board).

1. Total Physical Delay = $2 \times (propagationdelay + transceiverdelay)$

   $2 \times (20 \times 5 + 150) \times 10^{-9} = 500$ ns.

2. Assuming baud rate pre-scalar as 2.

   The CAN clock = $\frac{48}{2} = 24Mhz$.

   Time Quanta = $\frac{1}{24 \times 10^{-6}} = 41.66ns$

   No of time Quanta in one CAN bit time = Round_up(CAN Transmission Rate /Time Quanta).

   = Round_up($\frac{1000}{41.66}$) = 24.

3. Time Quanta assigned to propagation delay = 500 / 41.66 = 12 TQ.

   1 TQ is assigned to synchronisation window.

   No of remaining TQ = 24-(12+1) = 11.

4. Since the sampling point is the intersection of the phase segment 1 and phase segment 2.

   A good sample point is 66% of the CAN Bit time.

   For a total of 24 TQ, 66% is 16, hence

   Phase Segment 1 = (16-13)=3 TQ.

   Phase Segment 2 = 24-16 = 8 TQ.

4.2 CAN Test Bench hardware

The functionality of the CAN IP core described in the previous sections closely resembles the most common industrial CAN controllers currently available on the open market, such as the MCP2515 and the SJA1000 [SJA00, Mic03]. There have previously been CAN IP core implementations, principally for research or educational

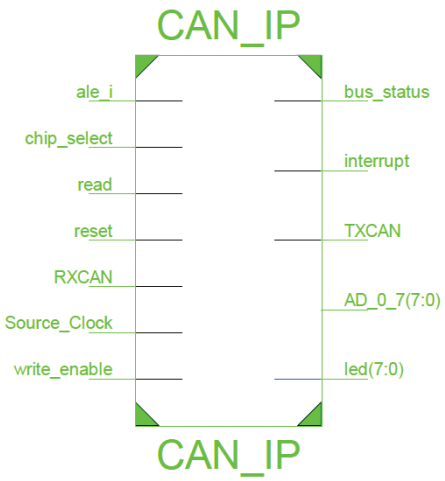


Figure 4.10: CAN IP core pin diagram

purposes such as [OAF05]. This implementation had a simple parallel interface to the host controller. Figure 4.10 shows the pin diagram of the CAN IP core. Multiplexed address, data and control signals are connected to the host controller. The RXCAN and TXCAN pins connect the 'rx' and 'tx' pins of the CAN transceiver. The CAN status register is connected to the LED pins; this helps to demonstrate the CAN internal status.

As discussed earlier, the RTL design of the CAN IP core is written in Verilog, and as such the design is platform independent. This puts no specific requirements on the use of a particular hardware for implementation. In this work, Xilinx FPGAs were principally employed for implementation purposes [Xil08]. This choice of platform was somewhat arbitrary, but Xilinx has a high device availability and widespread acceptance, coupled with good tool support from both the manufacturer and many third parties. This section describes the hardware and software used to create a test bench based around the developed CAN IP core. The CAN IP core requires 15

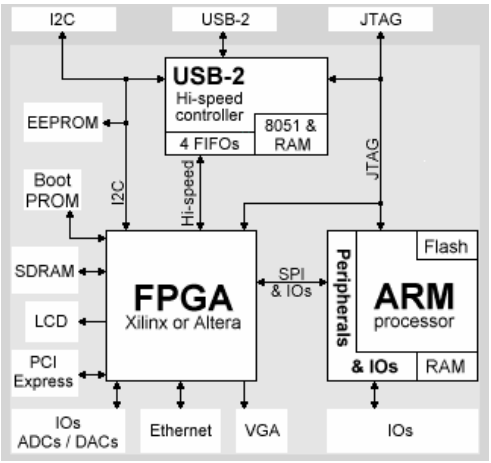


Figure 4.11: Knjn board block diagram

pins to connect the data/address and control signals to the host controller, and some additional pins are used to connect to either the CAN transceiver or to the indicator LEDs as shown in Figure 4.10. A customised solution will fit the requirements of a relatively large pin count, ideally an off-the-shelf board which features an integrated micro-controller and FPGA with appropriately interconnected I/O. If such a device were available, this would solve two issues i.e. there would be no need for external wiring; and both the devices could be driven by a single clock source to remove synchronisation issues. Such a customised solution was available in the form of Knjn development boards [fpg09]. The detail of this board and other hardware used is given in Figure 4.11 and is explained below.

1. The board consists of an integrated Xilinx Xc3s500e FPGA and an LPC2138 ARM7 [ARM05] micro controller. The Xc3s500e is a 0.5 million gate FPGA chip with enough gate count to store the CAN bit map. Table A.1 states the device utilisation. The CAN IP bit map requires only 20% of the device total

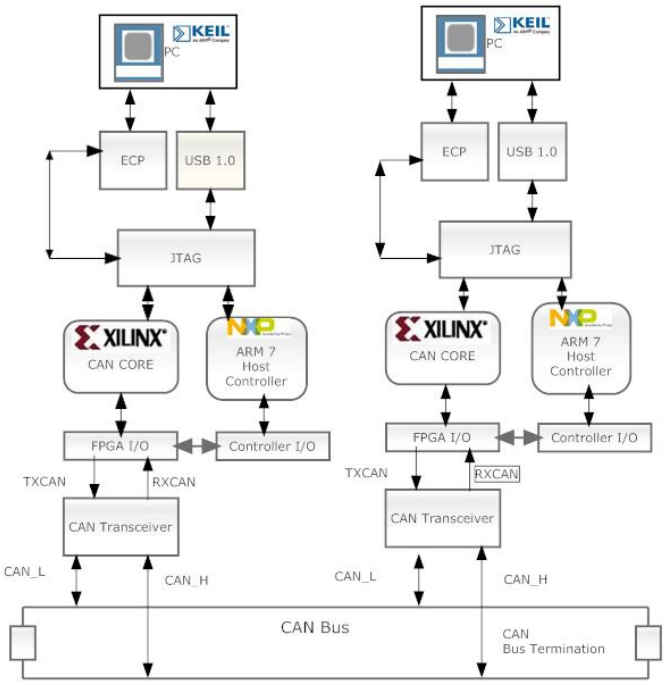


Figure 4.12: An overview of the CAN test bench

resources.

2. The Knjn board is clocked by the usb-2 controller [fpg09] and the source can be configured to work at 12, 24 and 48 MHz. An external oscillator with a DIL-8 package can also be soldered to the board.

3. The board has an openocd usb interface for programming and debugging. A JTAG interface is also provided, which is essential for running of the Chipscope [Xil07].

4. The board also has an Ethernet, SPI and I2C interface for faster communications.

5. The USB interface can be used to power the board, or alternatively a 3.3 or 5 volt external power supply can be employed.

6. An ARM7 LPC2138 micro controller is integrated on the board. The ARM is programmed using the JTAG port over usb, employing an openocd server communicating via a telnet session.

7. A 24 MHz external oscillator clocks the ARM processor. 19 of the port 0 pins are internally connected to the FPGA but need to be configured using the pin assignment tool of the FPGA. These pins are also available to be connected to external signals (e.g. transcievers) if required.

The software tools used for the building of the CAN test bench is described, briefly, below:

1. Xilinx 9.1 toolset generated the net list and routed map, to be downloaded to the Xilinx FPGA.

2. The ARM GNU C toolchain was used to compile and generate the executable for the LPC2138 microcontroller.

3. The Eclipse IDE was used to debug the ARM microcontroller software using the JTAG over USB.

4. The KEIL vision IDE was used to compile software to be downloaded to any accompanying development boards as required for the experiments, for example an MCB2100 [KEI06] and Olimex LPC2129 ARM7-based boards.

The basic structure of the CAN test bench is given in Figure 4.12. FPGA is programmed with the CAN bit map. A JTAG interface is used for downloading and

debugging the design. An ARM LPC238 processor is used as a host controller for the CAN IP core. The Phillips PCA82c250 high speed CAN transceiver [EE96] was used in this design for handling the physical transmission and reception on the physical CAN bus. The transceivers are inter-connected using a standard twisted pair cable with a standard termination of 120 Ω.

4.3 Design Flow and Analysis

In this section the design flow of implementing CAN IP core is discussed, and analysis of resources and timing is presented. A typical IP core design flow is shown in Figure 4.13, a brief description of each section is given below.

Design Start

The specifications are translated into HDL coding; for the CAN IP core this was a translation from the CAN standard document [11893, Bos91] and coding it in Verilog. The basic function block and their implementation in Verilog have been presented in the section 4.2. Next an overview on the feasibility of Verilog HDL for the design of CAN IP core is presented:

1. All the functional blocks had been implemented and are realizable in Verilog; the Verilog coding provides flexibility and simpler design technique. Each of the functional units is defined as a separate Verilog module, this also provide an easy interface for the inter-module communications.

2. The register and wire sizes are parametrizable, hence a single definition of a block of code can be re-used for several implementations; this was helpful when

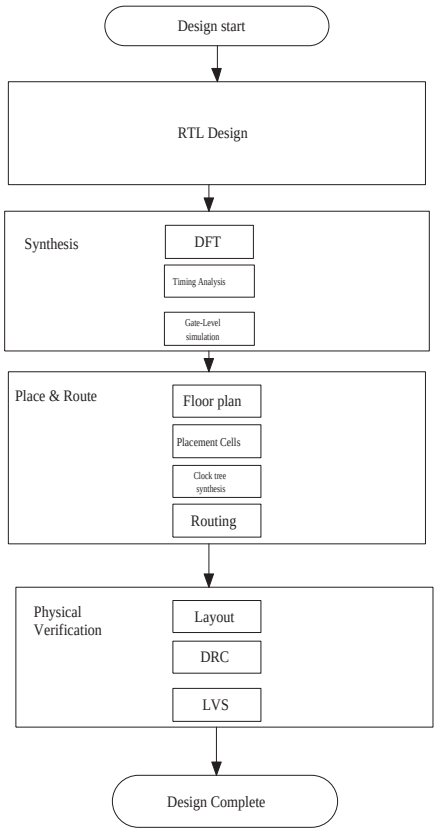


Figure 4.13: An IP core design flow

designing configuration registers for the CAN IP core. The configuration register size varies from a single bit to 8 bits; by using parameters no separate definition was required for each instance of these registers.

3. State machines are used extensively when implementing transmit and receive logic inside the CAN IP core. Verilog provides simple 'always' and 'case' blocks to implement the state machines.

4. In VHDL, use of design libraries are extensive, this provides a large collection

of custom built code for implementation of architecture, packages and configurations. In Verilog, there is no concept of libraries but synthesizable modules are made available by the FPGA vendor's for designs such as ROM, RAM's, DSP and mathematical functions. In the CAN IP core design the Xilinx RAM has been used to implement the message fifo.

The details of few of the issues encountered during the writing of the HDL code and set up of the test bench are listed in Appendix A.2.

Synthesis

Synthesis translates the HDL code into Register Transfer Level (RTL) model. The RTL model produces a net list using the synthesis tool. This model demonstrates transfer of data between the registers and combinational logic, and also specifies the components used such as memory and macro cells. In the synthesis step the design goals and strategies is also specified; design can be set to optimize on the basis of area, power or timing. When a design is set to optimize on the basis of a certain parameter the FPGA synthesis tool will try to put in a higher effort to achieve the design constraints. Since the CAN IP core design does not have any specific area or power requirement, the synthesis was optimized for better timing. The design is further optimized for higher speed using effective Floor planning shown in the respective section.

The statistics for the device utilization with a balanced and time optimized approach is given in Table A.1 and A.2. From the tables, this is evident that there is not a significant difference in resource utilization; a maximum of 1% of gate count has increased. However, the advantage of using the timing optimization is that the

timing constraints given in next section are easily met.

User Defined Constraints

The user constraint file for the CAN IP core design is given in Appendix A.5, there were three major constraints for this design.

1. The knjn board has I/O pins which are shared between the LPC2138 and the FPGA. The interface signals have to be connected on these shared pins to avoid any external wiring. Pin P0.2 to P0.19 and P0.23 of the ARM processor is connected to the FPGA. The CAN IP core has 8 bit address/data and 4 control signals to connect with the ARM processor. Therefore 12 of these signals have been connected together on the shared pins given in the user constraint file.

2. The next constraint was to connect all the input signals to the input-only pins on the FPGA for optimal performance and better isolation. The 'RXCAN' and 'read' signals are connected to the input-only pins, since there were no additional input-only pins available on the inter-connected pins, hence input signals such as write_enable and chip_select are connected to the bi-directional I/O pins. The pin assignment tool then specifies these pins as connected to the input signals.

3. The power supplied to the ARM I/O pins is 3.3 V $\pm$ 10 % [Phi04], to keep it compatible with the FPGA pins, 3.3 LVCMOS I/O standard is used for the CAN IP interface signals.

4. The clock signal to the FPGA is provided by the FX2 usb-clock; this clock can be configured to run at 12, 24 and 48 MHz. For this design 48 MHz clock speed

is used and the timing constraints has been selected in accordance with this clock. The reason to select 48 MHz is faster operation and realizable divisor values (baud rate pre-scalar) for CAN bit rates greater than 1 Mbps. To bind the FX2 clock to produce an exact 48 MHz, a 20.833 ns clock period and a 50% duty cycle is used as a timing constraint. The maximum clock-to-setup path[1] delay observed is 17.263 ns, this allows a maximum clock frequency of 57.927 MHz. The value used for this design is 48 MHz which is inside the maximum limit. Alternate options such as multiple clock sources and clock area partitioning are available to achieve higher frequency operations for this design. The use of 48 MHz clock achieves all the required constraints, hence an alternate clocking mechanism would only increase the complexity of the design. The complete static timing analysis is attached in Appendix A.4.

Floor Planning and Physical synthesis

Floor planning is the process of placing the similar structures close together to increase on the performance and reducing the area. If the structures lie close together then the path delays decrease and timing constraints are relaxed. The area of the FPGA used in this design Xc3s500E is divided into the four equally sliced blocks. With the auto routing option the design is equally divided into the four clocks named as X0Y0, X0Y1, X1Y0 and X1Y1. The DCM unit which provides the clocking to the design is placed in the X1Y1 unit, since by default the design is scattered around all of those 4 blocks, hence there were large clock skews and path delays between sequential and

[1]A clock-to-setup path is a path starting at the Q output of a flip-flop or latch and ending at an input to another flip-flop, latch, or RAM, where that pin has a setup requirement before a clocking signal [xil].

combinational units. Floor planning is used to create 4 new physical (P) blocks for synthesis; each P block holds one of the 4 main functional units of the CAN IP core. The main units as discussed previously are the BSP, BTL , the configuration register block and the clock module. The size of each P block was estimated using the device utilization statistics which is generated by the Xilinx synthesis tool. These new P Blocks are then placed near the DCM unit and to each other, there was reduction in the path delays and the clock delays. With auto routing the setup to clock delay was 17.263 ns and maximum possible clock speed is 57.927 MHz, with manual routing as discussed above the minimum setup to clock delay is reduced to 16.686 ns and a maximum possible clock speed increased to 59.284 MHz.

After floor planning and physical synthesis, a new estimate of the resource utilisation was carried out. The overall slice usage reduced from 1062 to 901, only the use of 4 input LUT's are increased by 1%. The details in the Table A.3 indicate this improvement in resource utilisation.

Bit Generation

The Bit generation is a process to generate the final bit stream file to be downloaded to the FPGA. The Bit generation takes a fully routed NCD (native circuit definition) file to generate a bit map. The configuration used in the CAN IP core design was to generate a bit file and a PROM file (Xc3s500E has a 4 MB of boot ROM). The bit file is used to map the FPGA which is a volatile memory, the PROM file resides in the FPGA ROM, which is copied back after a new power cycle. Chipscope logic core only works if the Read back and reconfiguration is allowed, this option is also enabled when generating the bit stream. The bit stream is downloaded to the FPGA

using either the IMPACT tool provided by Xilinx or the FPGA conf tool provided with the Knjn board. The size of the CAN IP bit stream file is 278 KB.

In this section the design flow for the development of CAN IP core was presented; the statistics given in Appendix A shows the feasibility of the design. All the constraints related to Area, I/O block and timing closure have been met; it is also shown that by using the different design tools the IP core has been optimized for performance with minimal compromise on the size of the design.

4.4 A Simple Two Node Test Network

Once the CAN implementation on the RTL was completed, it was required to run a simple communication test using the CAN IP core. A test bench with two nodes based on the CAN IP core was configured, to verify normal CAN message transmission and reception. Although comprehensive testing and conformance of the CAN IP core is discussed in the next chapter, a straightforward test demonstrates the basic functioning of the CAN IP core. Before a description of the transmission test and its procedure is given, a brief description of Chipscope is presented:

Chipscope

Chipscope [Xil07] is an integrated logic analyser, which can be configured inside an FPGA as an additional core to the main implementation. This core is connected to the Chipscope user interface running on a PC via a JTAG interface. Internal signals can be connected from the implementation module to a Chipscope core, similar to connecting signals between two HDL modules. A Chipscope core is added after the synthesis step and only the synthesized components or nets are available for logic

analysis.

A Chipscope snapshot presents the status of the signals over a sample length. These samples can be taken randomly or with a trigger condition. Different markers are used to signify status of different signals at a specific point in time. Marker T, always shows the trigger instance, while marker X and O are used to calculate the difference between any two events on the snapshot.

4.4.1 Message Transmission

Before a message is transmitted, the CAN IP core needs to be properly configured with appropriate bit timing and message filtering information. Appropriate configuration information needs to be written by the host controller into the corresponding CAN registers. For this purpose, an 8 byte message with random data and identifier values was selected with message filtering disabled, i.e. messages with any possible identifiers will be accepted and placed in the receive FIFO on the receiving node. Bit timing information for transmission speed of 1 M bps was written into the Bit timing register. Once the message has been successfully transferred to the transmit buffer of the transmit CAN controller, the transmit request bit of the CAN command register is set to initiate message transmission on the bus. A code listing written for the host controller to configure the CAN IP core is given in function CAN_Init() in Appendix A.1.

Figure 4.14 shows a Chipscope [Xil07] snapshot illustrating the successful transmission of the message. When a message is transferred to the transmit buffer from the host controller, then a transmit request is set to instruct the controller to attempt transmission. The CAN controller will arbitrate for the bus access, if the bus is free.

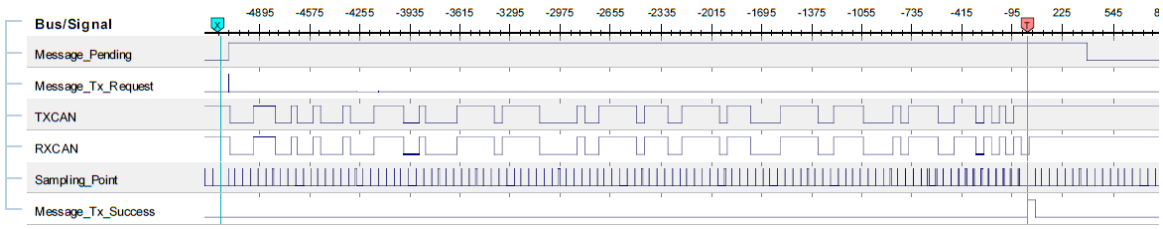


Figure 4.14: CAN message transmission

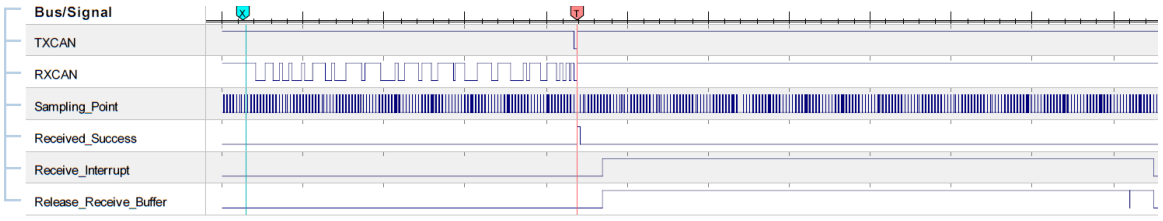


Figure 4.15: CAN message reception

Since this is a two-node network, with one node being in receiver mode, the controller does not have to arbitrate for the CAN bus access. The RXCAN and TXCAN signals specify the message being transmitted and simultaneously received to verify the correctness of the message. The message transmit success indicates a successful transmission of the message.

4.4.2 Message Reception

Similar to a transmitter, the node which is to receive the message must be properly configured to receive the message and notify the host controller of the reception of the message. The host controller may then run an appropriate routine to retrieve the message from the receive buffer FIFO. In this case, the host controller configures the CAN controller receiver interrupt to be enabled, such that after a successful reception

of a message the host controller will be interrupted. Once all messages are read, the host controller then sets the release receive buffer signal. This will initialise the receive FIFO for further reception of CAN messages.

Figure 4.15 shows the successful reception of the message. The RXCAN signal is recessive in idle mode. A dominant bit indicates the start of a CAN frame. The BTL then synchronises the controller to the bit stream and all the frame fields are received. Rx Success signal indicates the successful reception of the message. Once the message identifier goes through the filtering process, the message is stored in the buffer. The Receive_Interrupt indicates the host controller of a received message, which is then read and processed.

4.5 Conclusion

This chapter has discussed the evolution of the basic CAN IP core and a flexible test bench to allow experimentation. The CAN IP core was built in the following steps: i) firstly the specification understanding from CAN documents, ii) a HDL code translation of the CAN specifications, iii) synthesis and implementation on a FPGA chip and iv) setting up of a basic test bench to experiment.

A detailed discussion of the different functional blocks of the CAN IP core, and issues related to its implementation, was presented. Although the results of a simple initial functional test of the IP core were described, its functional verification to the CAN specifications is clearly required [Bos91]. This is the topic of the next chapter, and a new strategy for CAN conformance testing where FPGA devices are employed will be presented.

Chapter 5
CAN Conformance Testing

Conformance testing is an integral part of any protocol development. Conformance testing verifies the behaviour and capabilities of a protocol implementation against the requirements and ideal behaviours as set out in the relevant standard. Traditionally, the implementation of protocol conformance testers has somewhat been a proprietary activity, employing dedicated hardware and analysis software especially written for the protocol and device under test. This is clearly a practical approach when testing and verifying device conformance prior to high-volume IC manufacture. However, recent years have seen resurgence and increased interest in the use of protocol implementations achieved by the use of programmable hardware devices such as FPGAs.

By their very nature, such soft core implementations are often needed in one-off developments; these implementations may even add additional or custom functionality to existing protocols. In these circumstances, cost and availability reasons often dictate that it is not practical for developers to use traditional conformance testing equipment.

When a protocol is implemented in an IP core, it is independent of any specific technology. This is still necessary to test its functionality against the relevant standards. The IP core is implemented using an HDL which in many aspects resembles a traditional programming language. For this reason, many of the features of traditional approaches to conformance testing become redundant.

The motivation of this work was to find a flexible and cheaper technique as a replacement to the traditional ones. Many complex designs, some consisting of multiprocessor clusters, DSPs and communication protocols, have been implemented as IP core on programmable devices. There has been successful implementation of both real-time and non-real time communication protocols in IP core. Examples of non-real time protocols are Ethernet IP core [Raj06]. Embedded and real-time protocols such as Controller Area Network Enhanced Layer (CANELy) [PRA06] and Flex Ray [Gmb07] had also been successfully implemented.

5.1 Conformance Testing of Protocols Implemented on IP Cores

The test and verification of a IP core as either an individual entity - or as a system level entity - in a SoC design is clearly of paramount importance in complex electronic systems. However, protocol conformance testing requires that both the logic and functionality of a candidate design are verified. Chapter 2 has discussed some general approaches to conduct test and verification of CAN protocol implementations on standard (silicon-based). ISO has not only developed CAN conformance testing standard [ISOa] but also provides documentation for setting up a test architecture [ISOb]. The next few sections will look into the recommendations of setting up a test plan and whether this can be employed to conduct CAN conformance testing.

5.1.1 Conformance Testing Standards

A standard conformance test suite is typically employed to indicate anomalies in a given protocol implementation. The test suite execution procedure to an Implementation Under Test (IUT) is not unique and varies depending on the suite in question. For example, IEEE 1802.3 [IEE94] provides documents outlining the conformance testing standards of a well know protocol suite IEEE 802 [IEE] for LAN (Local Area Network) communication. However, these documents do not specify a unique approach to conduct these tests; it is up to the tester to adopt any such method of convenience to perform these conformance tests under a guideline.

ISO 9646-1 [ISOb] is one such guideline, which states a layered approach to test a protocol. The ISO 9646-1 Test Plan (TP) is flexible to the layer under test, i.e. the same TP can be used to validate different layers. A same TP for a physical- layer testing (such as transceiver functionality or interface standards) can also be adopted to verify the LLC/MAC functionality of a communication protocol. The adaptability of the TP depends upon the use of both appropriate hardware and software. The TP outlined in this standard is shown in Figure 5.1, and indicates that the tester is divided into two testing blocks, a supervisor and an Implementation under Test (IUT).

1. Implementation under test(IUT): IUT module which is to be verified, this implementation can be in form of a IP core, a software module or any hardware component.

2. The first component is the Lower Tester (LT) which provides the test pattern generation and analysis.

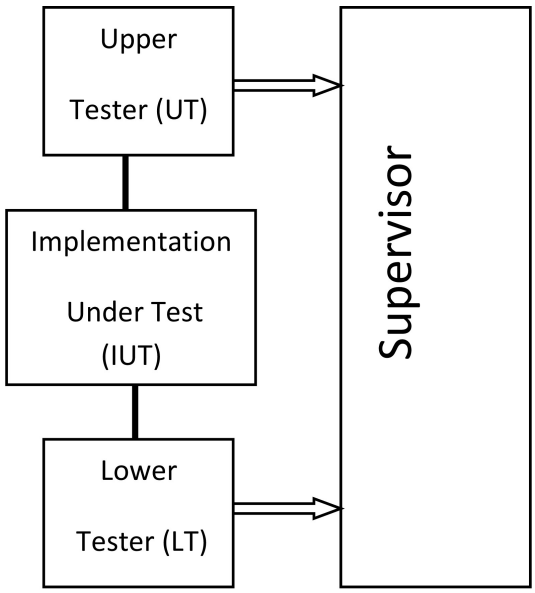


Figure 5.1: ISO 9646-1 test plan architecture.

3. The Upper Tester (UT), has the software to configure and control the IUT.

The Upper Tester (UT) is typically a host processor or a programmable device of some kind, also provides coordination to run the tests between the LT and the IUT [10B01]. The UT receives stimulus from the LT, and generates messages to be passed on to the IUT. The IUT then processes these messages, and both the UT and LT components monitor its behaviour for consistency with the protocol under test. The test supervisor verifies the results; if the result is satisfactory, the test is considered passed and proceeds to the following conformance test.

It should be noted that the CAN testing procedures include coverage of common error conditions, valid frame and bit timing tests. Most tests are critical, and the latter category bit timing contains a number of tests that can be difficult to localise, and suitable means are required to capture and display multiple logic signals an appropriate time-scale. This typically requires the use of dedicated hardware and

logic analysers [LKK98].

5.1.2 Proposed Environment

As discussed earlier, the use of HDLs for soft-core implementations requires verification; generally speaking, pure simulation methods, and also formal verification techniques. If these verification techniques applied solely to the HDL code, it cannot provide a guarantee that the HDL code has been correctly translated and implemented on the target hardware (FPGAs), and is working as desired. The design inside an FPGA is tightly integrated, and most of the signals are inter-wired. These signals are unavailable on the external ports for observation. The two possibilities of testing IUT is explained:

Simulation vs Logic Analysis

Most of the soft-core implementations are tested in simulation [Men07], tools such as Modelsim, Cadence Incisive [Men07, Cad04] provide strong simulation capabilities. Running a simulation, is easy since it does not require synthesis and implementation on a real device (FPGA). This saves time and most of the code are debugged during the design process. However to provide stimulus to the implementation under test, HDL test bench are written; therefore, no actual hardware is created.

Checking the CAN IP core designs in simulation has its own limitations. What if the HDL test bench is written incorrectly, also consider communication protocols such as running in an industrial environment, the limitation of the simulation to model noise and errors is a difficult task. Hence real-time testing needs to be done for a design is considered to be fully verified for its functionality. The conventional logic

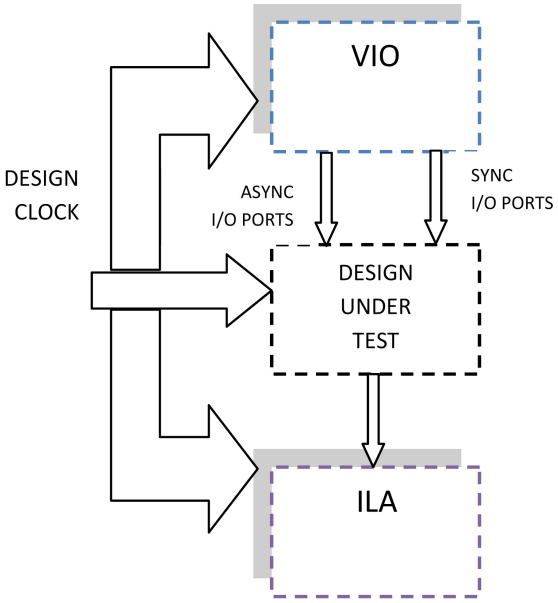


Figure 5.2: Chipscope architecture.

analysers [TLA, Agi08] can analyse the signals in real-time. These logic analysers have their own limitations as discussed in the last section of this chapter.

FPGA manufacturers have designed on-chip debug tools for this purpose. These tools provide full internal visibility using integrated logic analysers (ILAs) such as Signal Tap [Alt] and Chipscope pro [Xil07]. These tools provide small and efficient cores to debug not only I/O but also importantly - internal signals can be captured. Such tools provide real time system debug support for example, by using a JTAG port. The Chipscope pro from Xilinx is one such tool which provides on-chip debugging facility.

Figure 5.2 shows how a design under test can be attached with Chipscope cores. The cores can either be initialised in the HDL source code manually, or by using core insertion tools provided by the package. These cores run at the system clock or a derived clock such as a DCM[XAP03]. The designer can assign different cores to

the design (described below). The three Chipscope cores for conformance testing are listed below:

1. ICON is the controller core and needs to be generated for the Chipscope to work. It provides an interface between the external ports (i.e. JTAG) and the internal cores (ILA, VIO).

2. ILA is the integrated logic analyser core; all input and output signals requiring observation are connected to it.

3. VIO is the Virtual Input Output, which provides Virtual I/Os to the HDL design and can be initiated by the user at run time. These I/Os can be synchronous or asynchronous to the system clock.

The remainder of this chapter will demonstrate the use of such an environment to test a soft core implementation of a CAN controller.

5.2 Test Bed

Real-time testing of a CAN implementation is quite a complicated procedure, and in this case for practical reasons no specialised hardware and software was available to generate the required testing patterns and monitor the behaviour of the CAN IP core. For this reason, only low-cost off the shelf components were used.

In addition to these standard hardware parts, the Chipscope analysis tool [Xil07] was used to visualise and capture the behaviour of the soft core, allowing verification of the testing results. Chipscope is inserted as a separate core onto the device to be monitored, allowing multiple signal channels to be captured via a JTAG interface.

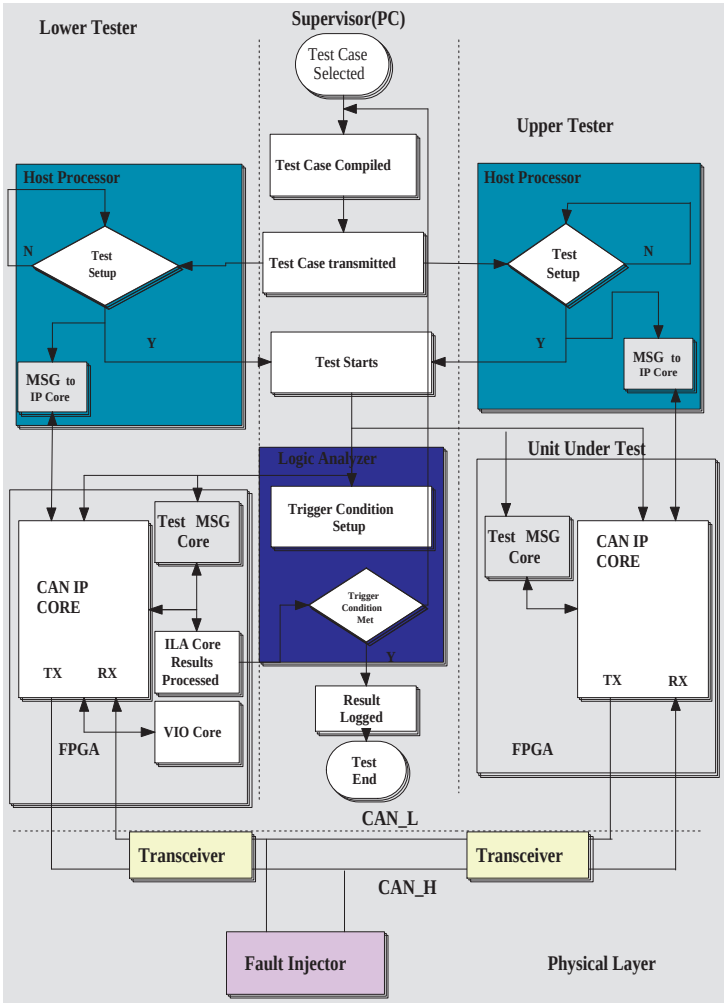


Figure 5.3: Conformance test bed.

Up to 16 internal signal ports can be analysed in a single core, and each port can accommodate up to 256 signals. Multiple cores can be attached to increase the number of signals in a FPGA [OMS05]. In comparison to other resources for capturing multiple FPGA signals, Chipscope retains the key features required but at a fraction of the cost.

In the previous section, the different components of the ISO TP were explained. The test bench for the conformance testing has been designed according to the ISO

9646-1 TP. Figure 5.3 presents the working of the test bench, which is further explained in the following lines.

Supervisor

The PC has been used as a supervisor to coordinate the test suite. Test programme's are compiled and downloaded using the PC to the LT and UT. Also PC runs the Chipscope user interface and trigger conditions are setup to monitor the success of the test. A JTAG interface connects the Chipscope to the ICON core, the integrated logic analyser core resides in both the LT and IUT FPGA.

Lower Tester

The LT is used to run the tests and coordinate with the supervisor to generate specific test patterns on the physical layer. In this project the LT is made up from the Knjn board [fpg09] with a host processor and an FPGA. The FPGA hosts the CAN IP core. This CAN IP core is modified if required to generate test patterns. In many of the test cases, this is required to generate errors, modify bit timings (varying segments or sample points) e.t.c. These changes are done to the standard CAN IP and downloaded to the LT FPGA chip. When the tests run this helps to generate the indifferent condition on the CAN bus; the IUT is then observed for the required behaviour, written in the test specification. VIO core also made part of the LT in test cases where internal signal generation is required. The LT is also responsible to inform the supervisor once a test is completed. The ILA core sends the observation to the PC, to be captured by the Chipscope user interface.

Upper Tester

The upper tester is defined as a consumer of the IUT. In this test setup, the UT and the IUT is part of the same integrated board [fpg09]. The host processor is used to initialise the CAN IP core for a test; also once the test is conducted the host controller consumes the message and monitors for any specific conditions on the internal registers of the CAN IP. This helps to log the output and helps to verify the success of the test in addition to the Chipscope observation.

Implementation Under Test

The IUT is the CAN IP which is developed and explained in Chapter 4. The test plan is designed to help verify the conformance of this IP core. In some cases a test core and a Chipscope ILA core is also connected to the CAN IP. This is required when simultaneous observations are required from both the LT and the IUT. Depending upon the type of tests IUT is either configured as a transmitter or receiver of the message generated by the test suite.

Fault Injector Node

The Fault injector is a COTS ARM board [Phi04], with standard CAN interface, connected to the CAN bus. In this test bench design the board serves two purposes: it helps to inject faults at specified time for test cases related to error detection and management, and also consumes messages to further verify the successful working of the CAN IP core to the standards.

The details of the test bench can be referred back to the basic test facility which was described in Chapter 4. Only a brief account of the hardware and software

components is given below.

5.2.1 Hardware

1. Two Integrated boards with FPGA's (mapped with CAN IP core) and an ARM 7 working as a host controller [fpg09]. These boards works as LT and IUT.

2. A PC with a parallel port to interface with the FPGA for downloading the bit stream and communicating with the Chipscope cores on the chip. A JTAG to parallel cable is required for this purpose [Dig05].

3. An ARM 7 Micro controller boards [Phi04] is connected which works as a fault injector as explained in the previous section.

5.2.2 Software

1. Xilinx ISE [Xil08] for soft-core programming, synthesis, routing and programming the FPGA. The ISE is a complete IDE for FPGA development and contains some additional features like power analysis, optimal routing and timing analysis to name a few.

2. Chipscope ILA is used as a analysis tools with VIO core to generate and control different bit patterns. The bit pattern can be synchronous or asynchronous.

3. The Keil uVision 3 IDE [KEI08] has a open C compiler for ARM. KEIL was chosen for programming and debugging the Micro controller boards.

5.2.3 Use of Virtual I/O

The Virtual Input/output (VIO) core can be used to analyse and drive internal FPGA signals in real-time [Xil07]. The VIO cores have both asynchronous/synchronous signals which are used as both input and output to the system. In the proposed testing method, the synchronous outputs are used either as a sole source of test pattern generator or used in conjunction with HDL modules added with the CAN functionality in the IP core. VIO synchronous signals can output a static 1,0 or a pulse train of successive values [TDT+06]. A pulse train is a 16-clock cycle sequence (logic 1 or 0) driven out of the core on successive clock cycles. Also, different logical trigger conditions can be setup to analyse signals. For example, a trigger can be setup to analyse when an error frame is generated or an ILA counter is set to capture multiple instances of stuff bits [Xil07].

When using pattern generators, test vectors are first stored, and then sent only on the CAN bus when required, thus putting the IUT in different states and allowing its behaviour and responses to be analysed. In this test bed, FPGA based pattern generation is used, which is economical as it adds no extra price to the test setup. The test pattern is generated either by a test and/or a VIO core; these cores work in conjunction with the CAN IP core shown in Figure 5.3. This helps to accurately produce special conditions; for example, in test case 1 (to be reported in the next section) it was needed to delay a sample point by two time quanta on a recessive to dominant edge [ISOa]. This was achieved using VIO core.

This test pattern was easily generated by modifying the delay function in the BTL module of the CAN IP core. With the help of VIO core, a five bit synchronous input delay_add is is used to generate a time quanta delay to the sample point. An example Verilog code is given to illustrate:

```
always @ (posedge clock or posedge reset)
```

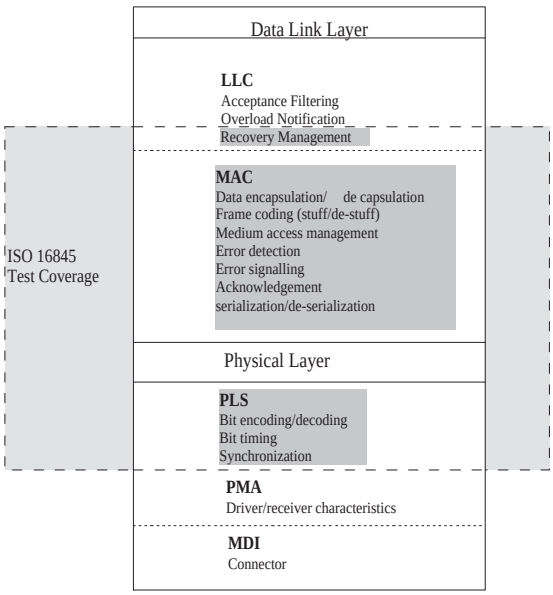


Figure 5.4: ISO 16845 Test Coverage [ISOa]

```
begin
  if (reset)
    delay <= 4'h0;
  else if (Resynchronisation & Phase_Segment1 &
  (~Transmitting | Transmitting & (Next_Bit_to_Tx | (CAN_Tx & (~CAN_Rx)))))
    delay <= (Time_Quanta_Count > {3'h0, Set_Jump})? ({2'h0, Set_Jump} + 1'b1) :
    (Time_Quant_Count + 1'b1);

/* Extra Code Added to set a delay using VIO*/
  else if (delay_add[0] & Transmitting & Next_Bit_to_Tx & (~CAN_Tx))
delay <= delay_add;
  else if (sync_window | start_phase_seg1)
    delay <=# 4'h0;
end
```

5.3 ISO 16845: CAN Conformance Standard

The ISO testing standard provides a complete test suite for the CAN devices. The ISO 16845 test specifications cover the lower part of the data link and the upper part of the physical layer as highlighted in Figure 5.4. The interface to these layers is provided by the CAN configuration registers on one side i.e. data link layer, and by the Tx and Rx (from the CAN transceiver) on the other side, i.e. the physical layer. The tests are defined into different categories based on the frame types and functionality of CAN protocol.

5.3.1 Frame Types

CAN communication is based on three types of frames, therefore the tests are designed based on these three types of frames:

1. **Received Frame types**: These tests are for IUT behaving as a receiver of either data or remote frames.

2. **Transmitted Frame types**: These tests are for IUT behaving as a transmitter of either data or remote frames.

3. **Bi-Directional Frame types**: These tests include testing the frames both received and transmitted by the IUT.

5.3.2 Test Classes

Each of the tests types is then divided into 7 test classes. These classes are distinguished based on the functional components of the CAN implementation.

1. **Valid Frame format Class**: This class includes transmitting/receiving error free frames.

2. **Error Detection Class**: This class includes tests on IUT for correct error detection and responses.

3. **Active Error Management Class**: This class of tests verify the correct management of active error frames.

4. **Overload Frame Management Class**: This class of tests verify the correctness of IUT to operate error free and corrupted overload frames.

5. **Passive Error Management and Bus off Class**: This class of tests verify the correct management of passive error frames and bus off state.

6. **Error Counters Management Class**: This class of tests verify the correct management of TEC and REC by the IUT.

7. **Bit Timing Class**: This class of tests verify the IUT for correct management of bit timing and detection of dominant edges.

The next section will present a series of test cases, one each from the test classes.

5.4 Test Cases

The proposed test facility was employed to test the CAN conformance of the custom created CAN soft core. As the total number of test cases to consider in any single CAN conformance test plan is several, one test case from each class is listed here. Details of several such tests are available in the form of a technical report [SS09].

5.4.1 Test Set up

A test case starts - and ends - in a stable testing state of the IUT [SG00]. The testing is divided into three states.

1. Set up state: it consists of the preamble to take the IUT in to testing mode, for example setting IUT as a transmitter or receiver of a CAN message. Also, in some test cases other preconditions are required to be setup for the main test body.

2. Test State: The main test body is performed after the set up state, this performs the principal test.

3. Verification State: Test state is followed by a checking or verification step, and finally a post amble (if one is required) culminates the test, leaving the IUT into a stable state.

Also, most of the tests require the IUT to be in default state, this state is defined as:

1. The TEC and REC must be 0.

2. No pending transmission is present.

3. IUT must be in idle state.

This process used in the test cases is to be described in the next sections. The first two test cases describe the use of the facility in the verification of CAN bit-timing and overloading test classes, and are detailed in section 5.4.2 and 5.4.3.

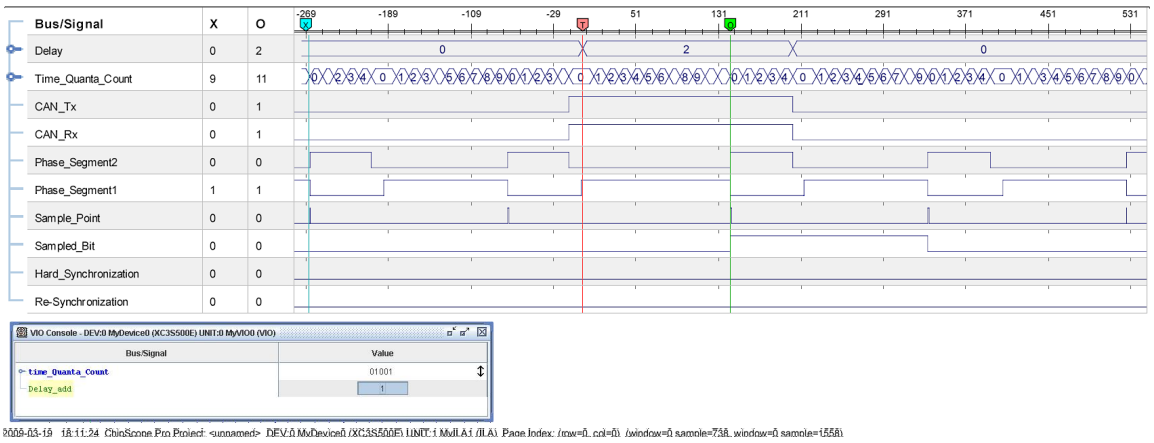


Figure 5.5: Chipscope snapshot for bit timing class.

5.4.2 Non-Synchronisation after a Dominant Bit Transmission

This test is part of Bit timing class, item number 8.7.7 [ISOa].

Purpose of Test: The test is to verify that an IUT transmitting a dominant bit does not perform any resynchronisation as a result of recessive to dominant edge with a positive phase error.

Set up State: The IUT is in the default state.

Test State: The IUT transmits a frame, the LT delay's each recessive to dominant bit by two time quanta. The VIO core provides the stimulus to generate a 2 time quanta extension of the phase segment 1, hence the sample point is delayed. In normal circumstances, the VIO core is sourced by system clock, in this case the system clock is 48 MHz. Since the transmission rate is 250 Kbps, this is too slow in comparison to the system clock. This will not have a synchronised impact as the delay_add trigger will finish well before the CAN bit time. Therefore, a DCM module [XAP03] to generate a clock which is 16 times slower than the system clock is used.

Any synchronous signal generated by VIO will now last for 1 CAN bit time.

Verification State: The IUT must continue transmitting without any resynchronisation.

This test was successful with desired results as stated in the purpose of the test; the observation on the IUT node from the Chipscope - shown in Figure 5.5 is as follows:

1. The default values for phase segment 1 and phase segment 2 are 10 and 5 time quanta, this is a precondition when the "Delay" (VIO) signal is zero.

2. The 'delay_add' signal on the VIO console is the synchronous input and when a user applies the input pulse it generates a delay as shown by the value of 'Delay' bus signal at Marker *T* (Delay=2 was set as a Trigger condition).

3. The observed value of Time_Quanta_Count is 11 (Count starts from 0) at Marker *O*, this is 2 more than the normal phase segment 1 value.

4. By adding this 2 time quanta delay before the dominant to recessive edge means this edge will have a positive phase error of 2 time quanta.

5. The Sampled_bit signal represents the CAN_Rx signal at the sample point. This value for recessive to dominant edge happened on the Sample_Point after Marker *O*.

6. Resynchronisation signal represents if any resynchronisation happens in case of an +ive or -ive phase error on an edge. As this signal remained low, thus proving that no resynchronisation happened.

All these steps verify the success of the test, this test case emphasis the usefulness of VIO core as a complex condition of introducing phase error to verify the behaviour

of CAN Bit timing has been easily achieved without the use of any external input.

5.4.3 TEC Non-Increment on 13 bit Long Overload Flag

This test case is a part of Error Counter Management class, item number 8.6.13 [ISOa].

Purpose of test: This test is to verify that an IUT acting as a transmitter when receives a 13 bit long overload flag should not change the value of its transmit error counter (TEC).

Set up Sate : The test is setup using two instances of CAN IP core, The IUT is set up as a transmitter and to send a message. The system clock for this test case is 12 MHz.

Test State: LT request to IUT, to wait for an overload frame before transmitting the next frame. LT then generates 13 bit overload frame. VIO synchronous input is used to generate an Overload request.

Verification State: IUT after receiving 13 bits of overload frame should not increment the Transmit Error Counter TEC.

All the test states are visible by the snapshots at both LT Figure 5.6 and the IUT, Figure 5.7. In Figure 5.6, a VIO console with a synchronous input overload_request can be observed, this signal request an overload frame (can only be requested by a receiver) between two data frames sent by a transmitter.

1. The signal Overload_Request is also shown on the ILA screen shot, once the data frame is received successfully, an overload frame is sent. These are showing by a high Overload_Frame signal.

2. The overload frame, lasts for 13 bit times as can be counted by the number of

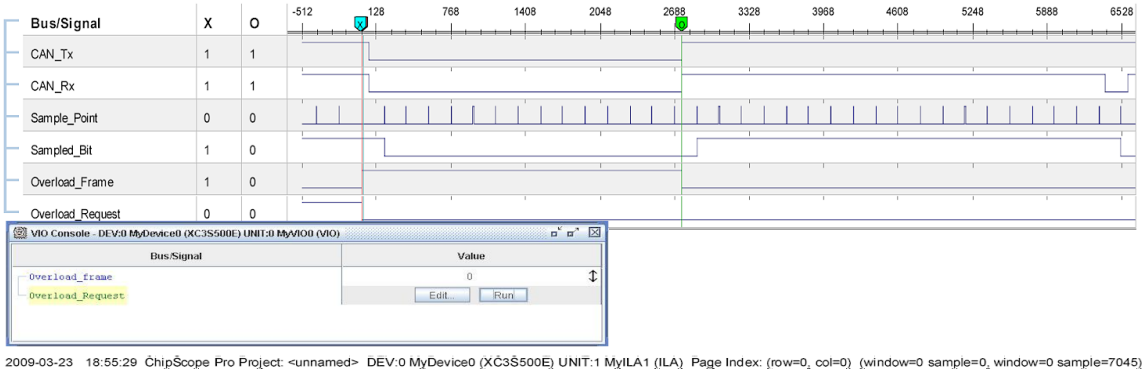


Figure 5.6: Chipscope snapshot at LT for error counter management.

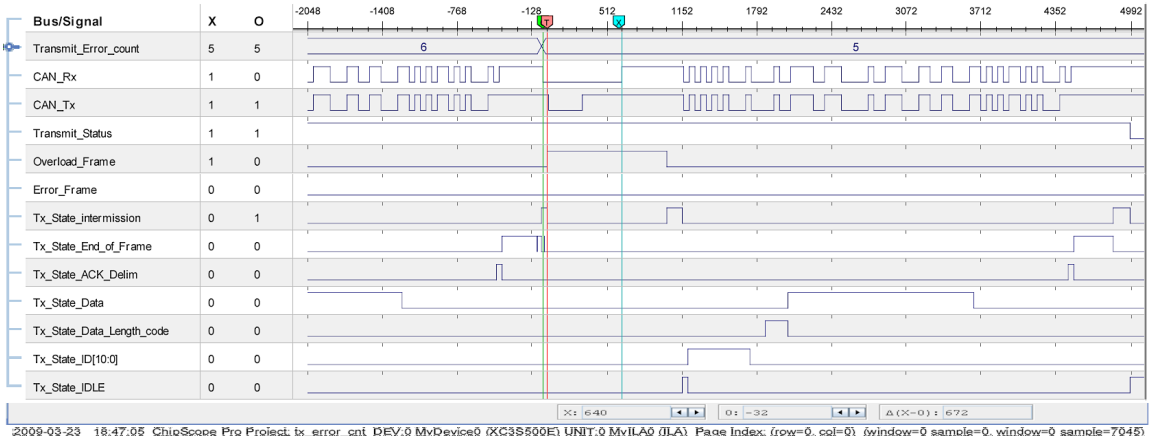


Figure 5.7: Chipscope snapshot at IUT for error counter management.

sample points between Marker X (start of Overload Flag) and Marker O (End of Overload Frame).

The IUT Chipscope snapshot in the test state is showing in Figure 5.7.

1. Before Marker O transmission of a standard data frame is illustrated by different transmission states denoted by Transmit_State_xxxx.

2. At Marker O which is the end of Transmit_State_End_of_Frame, a high Overload_Frame signal is seen demonstrating an Overload Frame being transmitted.

The CAN_Rx remains low between Marker T(Start of Overload_Frame) to the Marker O (End of dominant Overload Flag).

3. The difference between Marker O and X is '672', CAN bit time is given by 12Mhz/250Kbps= 48 clock cycles, hence a figure of 672 is showing 14 CAN bit time.

4. The value of Transmit Error Counter was 6 before the completion of data frame, and changes to 5 after a successful transmission of a data frame. This is illustrated by Transmit_State_End_Of_Frame signal.

5. After receiving 13 dominant bits of Overload Frame, the Error_Frame signal remains low and there is no increment to the Transmit Error Counter.

6. This concludes that the test is successful as the IUT has not considered a 13 bit dominant overload flag as an error. After Marker X i.e., the finish of Overload frame,a successful transmission is showing by the several Transmit_State signals.

The test cases to be described in sections 5.4.4, 5.4.5, demonstrate the use of the facility in showing conformance to another important aspect of CAN conformance testing; the behaviour of the protocol in abnormal (error) and overload conditions.

5.4.4 Error Flag Longer than 6 Bits

This test is a part of the Active Error Frame Management class, item number 7.3.1 in ISO 16845.

Purpose of test: The test is to verify that a CAN transmitter will only tolerate 7 dominant bits after sending its own "error flag". The case described below is for

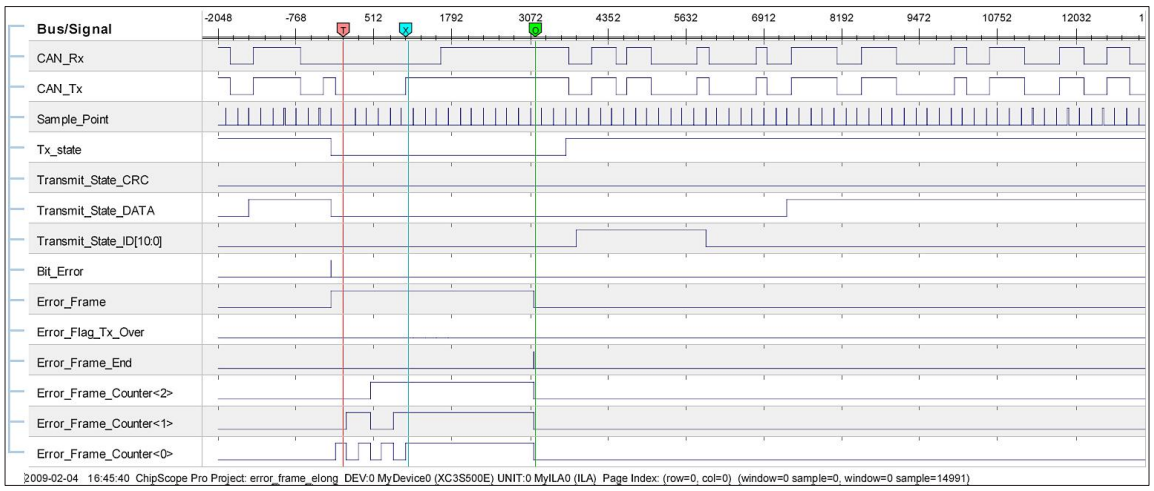


Figure 5.8: Chipscope snapshot at IUT, active error frame management class.

when the error flag is elongated by 4 dominant bits.

Set up State: The IUT is in default active error state and works as a transmitter, the LT (Second CAN IP Core is working as a LT and is in receiver state).

Test State: An error is generated during a transmission by the IUT, the fault injector node is used to inject a bit flip on the CAN bus. The IUT then sends a 6 bit dominant error flag, LT (working as a receiver) sends an 8 bit error flag.

Verification State: The IUT must not take more than 6 dominant bits as an error and should not transmit any extra "error frame", instead it should retransmit the corrupted message.

The methodology employed was to modify the LT code to carry out this requirement. The LT as a receiver is modified to generate an 11 bit "error flag". The

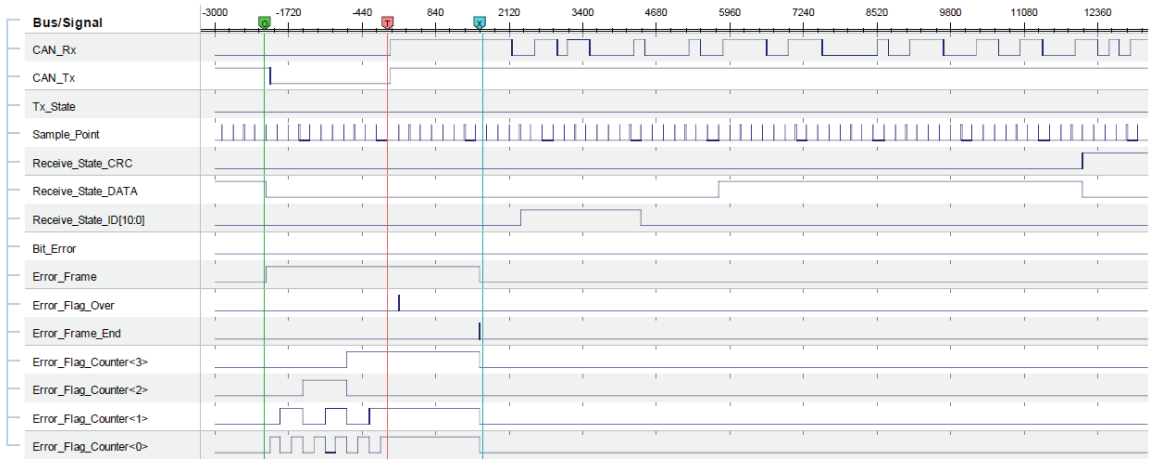


Figure 5.9: Chipscope snapshot at LT, active error frame management class.

snapshot of the events on the CAN bus was captured with the help of Chipscope trigger mechanism shown in Figure 5.8 and Figure 5.9.

1. On the Left of Marker T, the TX_State_Xxxx high indicates an ongoing transmission. During data transmission, a Bit_Error is injected, indicated by the bit inversion as CAN_Tx is recessive while CAN_Rx is dominant.

2. The error frame exists between markers T and O indicated by signal Error_Frame, the Error_Flag is between Marker T and X. The end of error flag is indicated by a high Error_Flag_Tx_Over signal. The Error_Flag_Counter bus is indicating the count of error flag bits sent.

3. Note that at Marker X the CAN_Tx signal has changed to recessive, while the CAN_Rx signal remains dominant for next 4 bits which is due to superimposition of the "Active Error Flag" sent by the LT.

4. The error frame is continued till Marker O and the end is shown by a low Error_Frame Signal and a high Error_Frame_End Signal.

5. On the right of Marker *O*, after three sample points (intermission Field), a new frame transmission has started indicated by different Tx_State_xxx.

The observation at the receiver node (shown in Figure 5.9) is as follows:

1. Marker *O* indicates the start of an Error_Frame, the Tx_State low indicating the node is a receiver.

2. The Error_Flag_Counter is a 4 bit wide bus which counts up to 11 bits i.e. it is sending 4 extra dominant transmitter node. The dominant bits can also be verified by the CAN_Tx and CAN_Rx bits.

3. After the Error_Flag_Over Signal is set high the CAN_Rx and CAN_Tx signals turns to dominant for next seven bits indicating an error frame delimiter.

4. Right of Marker *X* the Error_Frame signal is low, and after three recessive bits (Intermission Field), a new frame is started to be received (Tx_State is low, CAN_Tx is recessive), indicated by different Recieve_State_xxxx signals).

5.4.5 MAC Overload Generation during Intermission Field

This test is part of the Overload Frame Management class, item no is 8.4.1 [ISOa].
Purpose of Test: This test verifies that an IUT will be able to transmit a data frame starting with the identifier field and without transmitting SOF, when detecting a dominant bit on the third bit of the intermission field.
Set up State: The IUT is in a default state and transmitting messages.
Test state: The IUT is set up to transmit two data frames. The LT will request an overload frame after receipt of the first frame. After the completion, of the overload

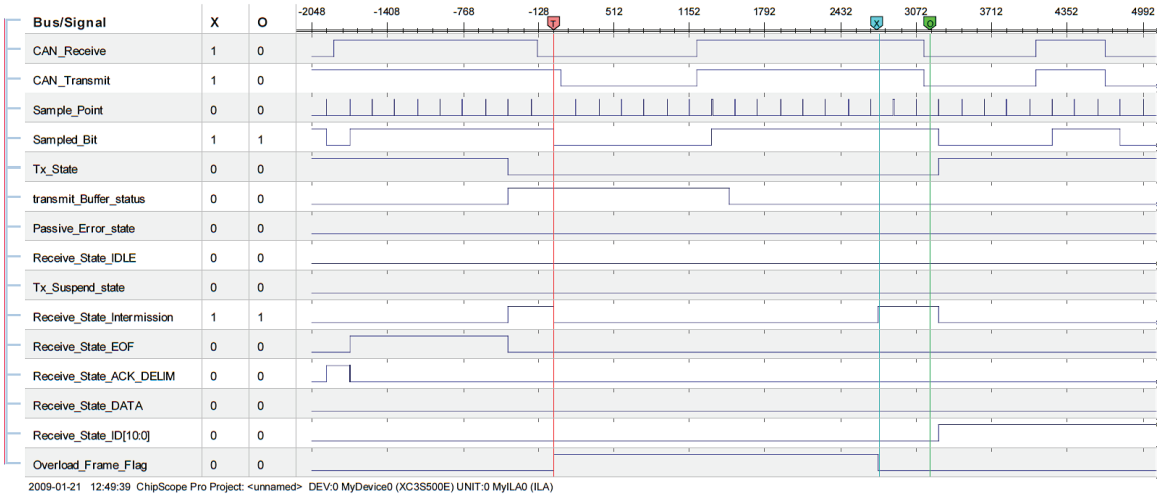


Figure 5.10: Chipscope snapshot, overload frame management class.

frame, the third bit of the intermission field is set dominant by the fault injector node. **Verification state**: The IUT must not consider the the third bit(which is dominant) of the intermission field as a bit error, and should not send a dominant level SOF. The dominant bit of the Intermission field is quite taken as the SOF, and IUT completes the transmission of the second frame.

This test was successful with desired results as stated in the purpose of the test; the observation on the transmitter node from the Chipscope - shown in Figure 5.10 is as follows:

1. Left of Marker T The Tx_state flag is high indicating ongoing transmission, Receive_State_Data and ACK_DELIM indicating a successful transmission while the node is error active.

2. At Marker T, there is an error on the Receive_State_Intermission field generating an overload frame. The Overload Flag is six dominant bits and 8 bits of Overload delimiter this can also be verified by the sample point count.

3. After the overload frame an intermission field signal can be seen at the Marker *X*.

4. The number of sample points can be counted i.e. 2 between markers *X* and *O*, displaying intermission field as recessive, the third bit of intermission field is a dominant bit.

5. Just after the Marker *O*the Receive_State_ID [10:0] goes high without any SOF. The Identifiers first 4 bits are dominant as required by the Test case.

5.4.6 Frame Acceptance after Passive Error Frame Transmission

This test is a part of the Passive Error State and Bus Off class, item number 8.5.2 in DIS [ISOa].

Purpose of Test: The purpose of this test is to verify that a Passive state IUT acting as a transmitter accepts to receive a frame after the second bit of intermission.

Set up State: The IUT is put in error passive state.

Test state: The IUT transmits a frame, and an error is introduced by the fault injector node. The transmitter sends a passive error flag. After the end of the error flag, the LT transmits a frame.

Verification state: The IUT must acknowledge the frames.

The Figure 5.11 shows the complete setup and running of the test.

1. The Tx_State signal is high, on the left-hand side of the snapshot. A bit error is introduced, and the IUT start to transmit Error_Frame_Flag. The error frame

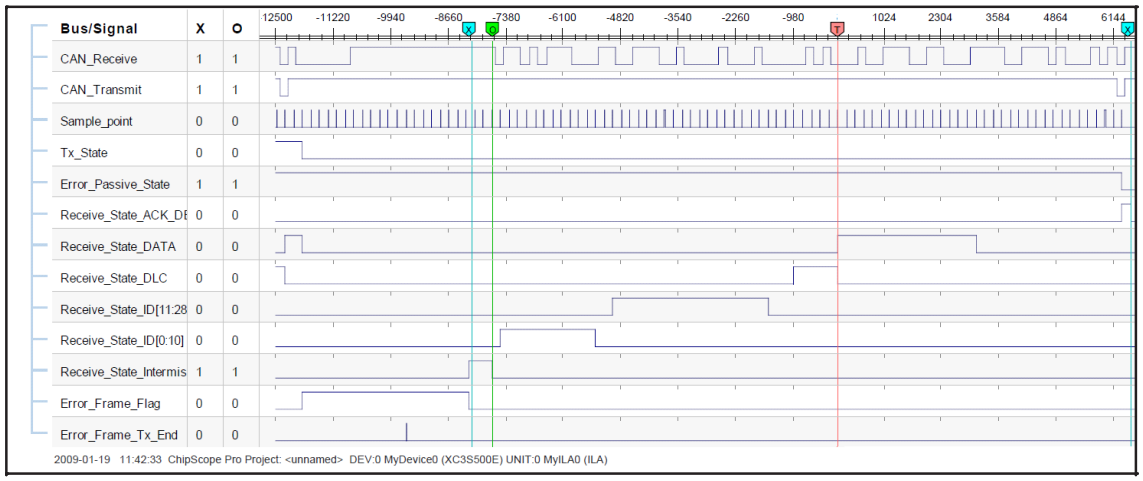


Figure 5.11: Chipscope snapshot, frame acceptance after passive error frame transmission.

is recessive signal on the first 6 bits showing by CAN_Transmit signal, this indicates IUT is in passive error state (Error_Passive_State). This recessive error flag is overwritten by the dominant bits indicated by the CAN_Receive signal. These dominant bits are between Error_Frame_Flag and Error_Frame_Tx_End signal. Error delimiter is a 7 bit recessive signal shown after the Error_Frame_Tx_End.

2. Between Marker *X* and *O* the Rx_State_Intermission signal is introduced for two sample points, after the completion of the Error_Frame_Flag (set to low indicating completion of error frame).

3. After Marker *O* the node starts to receive a new frame, which is indicated by a Tx_State recessive signal, and all the Receive_State signals indicates different states of Receive cycle. When the Receive_State_ACK_Delim is set high and CAN_Transmit goes low, this verifies a successful reception.

A successful reception of the message indicates the success of the message.

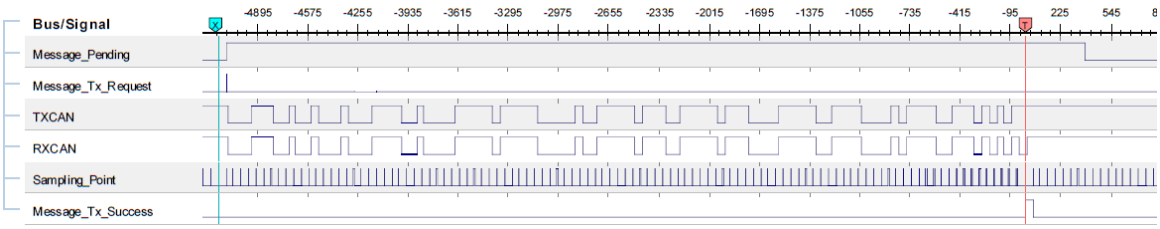


Figure 5.12: Chipscope snapshot for valid frame format test

5.4.7 Identifier and Number of Data Test in Standard Format (Both Transmission and Reception)

This test is part of the Valid Frame class, item number 7.1.1 and 8.1.1 [ISOa]. This test is significant for the experimental work done in this thesis, as this test verifies the conformance of the range of identifiers. With the successful completion of this test will provide sanctity to the experiments conducted by using these frame identifiers.

Purpose of Test: This test verifies the behaviour of the CAN Core while transmitting a frame with different identifiers and different number of data in a standard frame format.

Set up State: The IUT is in the default state.

Test State: IUT transmits and receives several frames with different identifiers and number of data bytes. The IUT acts as a transmitter for test cases involving LT as a receiver and vice versa.

Identifiers used in the Test: $\epsilon\,[000h, 7EFh] \cup [7F0h, 7FFh]$.

Number of Data bytes used in test: $\epsilon\,[0, 8]$

With the combinations given above for both transmission and reception, there are 32736 different frame messages in this particular test case. An example of the test case

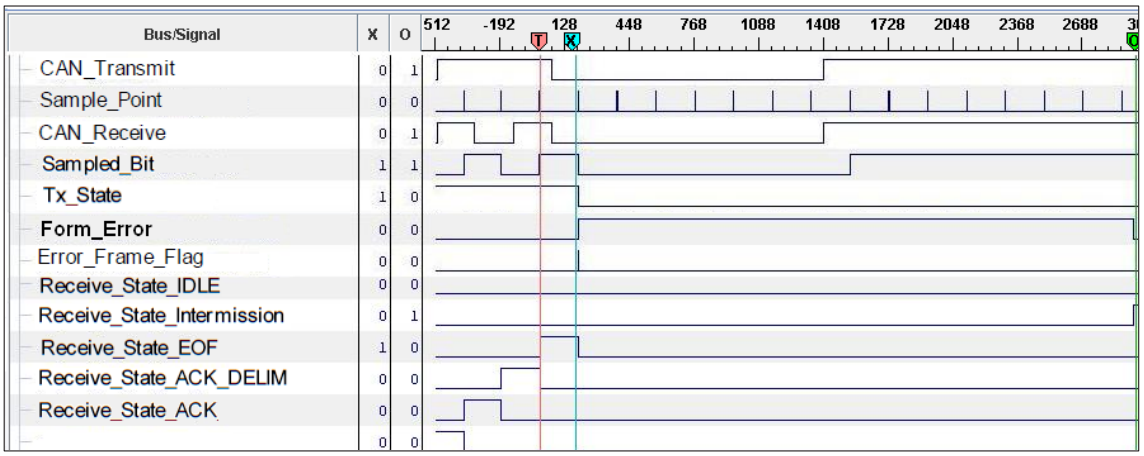


Figure 5.13: Chipscope snapshot for EOF form error

with a particular frame is given in Figure 5.12. Tests covering different combinations are conducted and is part of the document [SS09]. All the different combinations of identifiers with different data size has been tested using a "nested for loop".

Verification: In this particular test case, IUT is a transmitter. The Identifier of the message sent is 1C9 H and message size is 8 bytes. IUT transmits a message received successfully by the receiver(LT) as indicated by the Tx_Success signal.

5.4.8 Form Error

This test is part of the Error Detection class, item number 8.2.5 [ISOa]. This test covers form errors, if the bit/s is dominant on a fixed format CAN field. This includes CRC Delimiter, ACK Delimiter and the EOF. The test case to be presented here is of EOF field. Error in EOF frame field causes IMD (inconsistent message duplicates) which can cause unnecessary re-transmissions [RVA+98].

Set up State: The IUT is in the default state.

Test State: The IUT transmits a frame on the CAN bus, the fault injector introduce a dominant bit on the 1st bit of the EOF field (EOF field is a set of 6 recessive bits). **Verification State**: The IUT should send an error frame and re-transmit the message.

Figure 5.13 shows the snapshot for the form error.

1. The Marker T indicates the start of the EOF, which follows after an successful completion of a frame transmitted by IUT.

2. The fault injector node induces a dominant level on the second bit of the EOF. This is a form error, thus Frame_Error_Flag indicates the start of an error frame by the IUT.

3. The Form_Error signal indicates that this is a form error.

4. After the error frame finishes, the corrupted data frame is then retransmitted.

5.5 Comparative Study

This section presents a cost and flexibility comparison between conventional conformance test methods and the proposed facility, to make the effective use of hardware and software readily available in a lab environment in order to perform protocol testing. The first observation is that the next facility does not require expensive and specialised PC interface cards such as [NI 08, Sof07] which are normally required for CAN conformance testing [LK98]. These cards capture real-time bus data, which are then examined and logged. These cards require specialised hardware and software [LAB09] along with interface cables which adds to the cost and complexity of

the setup. In this work, the internal state of CAN IUT can be analysed. In addition, there are several key advantages of Chipscope over hardware logic analysers and pattern generators:

1. The standard bench analysers cannot show enough signals as required in case of CAN or any other relevant protocols conformance as illustrated in section 5.4. Although, there exists few logic analyser systems which can display a large number of signals simultaneously [TLA, Agi08], but there prices are 10 times more than integrated logic analyser. The standard bench analysers can display million of samples, while Chipscope limits to a sample size of 16K, this limitation can be extended by using Digital clock Manager [XAP03]. The DCM can divide or multiply the system clock by 'n' times, the maximum value of 'n' is 16 for the XC3s500e used for the test bench, thus a possibility to capture 16 times more samples than with a system clock.

2. In typical logic analysers, the number of probe pins and interfacing hardware increases with the the increase of analysable signals. Chipscope can handle magnitude of these signals using a single JTAG cable. There are a few solutions, like Agilents FPGA trace port [Tra03] available, which uses a simple interface to analyse multiple signals but it also requires a specialised equipment and ILA tool.

3. Not only all the I/O signals can be connected to the Chipscope core, but also internal wires can be traced [LFY+07] which are very helpful in conformance testing. This helps to setup additional triggering conditions, for example, in test case B, it is easy to setup a trigger condition to wait for an Overload Frame

signal, while this condition is hard to setup in external logic analysers.

4. Virtual I/O is a real-time tool for pattern generation, but does not require any physical interface or port therefore it not only saves resources but also does not have the physical impairments of an external signal. Also, the ability to insert Virtual I/O cores into a design allows users to verify the design much faster and easier, and the ability to define internal I/O can significantly reduce the time spent in verification.

	IUT State					
Class	Receiver		Transmitter		Bi-directional	
	Total Tests	Test performed	Total Tests	Test performed	Total Tests	Test performed
Valid Frame Format	13	6	7	6	0	0
Error Detection	9	4	5	4	0	0
Active Error Frame Management	4	4	4	4	0	0
Overload Frame Management	5	5	5	5	0	0
Passive Error Frame Management	6	4	14	6	0	0
Error Counters Management	19	6	19	7	2	2
Bit timing	10	4	8	4	0	0

Table 5.1: Statistics of CAN conformance tests

5.6 Test Coverage

The tests presented in this chapter are representative tests from each class of tests given in the ISO document [ISOa]. The number of total tests given in the ISO document is 130. There are few tests which are repeated for range of changed values; although the values are changed the procedures to run those tests has been the same. The statistics of the tests covered in the current work are given in Table 5.1. The total number of tests conducted in this work is around 76 and is documented in [SS09], these comprise about 65% of the total tests given in the ISO document. The tests covered in this work is not 100% of the listed tests, but based on the following findings

and observations these tests have been found enough for the functional verification of the CAN IP core and the goals of the research.

5.6.1 Selection of Tests

Since a selective number of tests were conducted to verify the functionality of the CAN IP core. The selection of test cases to perform was a very important step to justify the conformance. The basic selection criteria was based on two things: (i) Each and every identifier and frame should be tested which needs to be used for the experimental work and (ii) Each and every functional unit as shown in Figure 5.4 and Figure 4.1 of the CAN IP core is at least tested once and if possible more than one test cases should be selected.

More than 30% of the tests documented in the ISO document is almost similar with the only difference is that the default state of IUT in one case is a transmitter and in the second a receiver. This has been carefully determined that not all the similar tests conducted are repeated for both the default states hence making sure that each functional unit of the CAN IP core is tested. This is explained with an example:

In the Passive error state class, test number 7.5.3 is for the IUT as a receiver and test number 8.5.3 for IUT as a transmitter [ISOa]. The set-up, test and verification state of the IUT is similar for both test cases. During the test state in both cases the IUT transmits a passive error frame on detecting an error and in return the LT sends 7 dominant bits. The verification state is also similar, which is that the IUT accept this condition and does not generate further error frames. In this work only test number 8.5.3., is performed successfully, this test is for the IUT working as a

transmitter. Although the receiver test case is required to be done, but due to the similarity of test cases, our objective of testing the functionality of error confinement block in case of an error frame in error passive state is fulfilled.

5.6.2 Edge Test Cases

In the Valid frame and error management test classes, a few of the tests are repeated for a range of data sets and sequence of the particular field. For example test case $\epsilon\,[000h, 7EFh] \cup [7F0h, 7FFh]$. The test case has been completed, generating all the possible identifier using a 'for loop' in the testing code of the LT. The IUT is configured to receive these frames successfully and all of these tests were conducted successfully. The success of these cases was verified by not detecting any error condition while running of these tests. But special care had been taken to analyse the tests on the Chipscope for edge cases [1] such as for identifier '000', '7EF','7FF' e.t.c. This is to assure that all the possible range of test values has been generated and tested.

Some of the test cases related to error frame classes are repeated for different bit positions of a particular frame field, edge cases have been selected in this case for testing to assure the coverage of full range of values. For example in test case 8.5.2, the IUT has to signal a form error after detecting a 'dominant' bit on the EOF field. This test needs to be repeated for bit position (1, 4 and 7) of the EOF field. As this work performed selected tests, the tests were only conducted for bit 1 and 7 to ensure the IUT functionality is verified at the start and finish of the EOF field. These examples given above emphasize on the edge cases, considering the extreme are covered, this gives more confidence in the testing procedure and the conformity

[1] An edge case is a problem or situation that occurs only at an extreme (maximum or minimum) operating parameter [wik].

of the CAN IP core to the standards.

5.6.3 Use of Standard CAN Controllers

The test bench discussed in the last chapter is also adapted for CAN conformance testing with slight modifications. This test bench also contains standard off the shelf boards with CAN interfaces. These interfaces are fully CAN conformant, and only support communication as specified by the CAN standards. These boards generate errors when they detects any unspecified transmission on the CAN bus.

These boards have been used to conduct CAN experimentation during this research work. Extensive testing was done during the windowed messaging scheme for unbound retransmissions and single shot messaging. These tests were conducted successfully for durations of 24 hours or more and the standard CAN boards have been used to receive and logged the CAN messages. These experiments used random data contents, range of tested identifiers, and all possible data lengths. The messages were also transmitted using maximum and minimum stuffing. These messages generate a large number of possible values of CRC as well. From the final case study statistics given in Table 9.2, this can be seen that almost 100% of the messages are received successfully by these standard CAN interfaces. Although, there was no specific method employed in these experiments to check different error conditions or error counters, but this can be argued that over a large data set there were no errors or anomalous conditions, thus the CAN IP core generally satisfies the test conditions given in the ISO conformance document.

5.6.4 Extended Testing

This sections discuss tests which were conducted on the CAN IP core but are not part of the ISO document, these tests are also important for the correct working of the CAN IP core. The ISO document covers testing related to the functional blocks of a CAN controller, but this cannot be claimed that ISO document verifies all the functionalities related to a CAN controller. There are no tests designed to verify the functionality of the CAN registers which are used to store configuration values and stores the transmitted and received information. During the large set of experimentation in this work has ensured that the testing of these registers is carried in parallel. This is an important step towards the functional verification of the CAN IP core, as without the proper working of the interface and register units CAN IP core will not function properly.

Also another aspect of extended testing is the use of industry standard values to generate the bit timing for the CAN IP core. The kvaser website [kva] provides the bit timing register values for all the possible CAN data rates, these values are used by the industrial CAN network designers to configure standard CAN controllers such as [SJA00, Mic03]. This further supports the argument that the bit timing functionality of the CAN IP core fully conforms to the standard CAN controllers.

The final comment on the test coverage is that in this conformance testing work, more than 65% of the total tests listed in the ISO document have been conducted. These tests had been carefully selected to test as many aspects of the CAN standard as possible. Although, some of the tests defined in the standard were not carried out prior to the experimental work being undertaken, the large majority of these test comprised of repeated tests using a large range of message identifiers. To ensure

the integrity of the results presented in subsequent chapter of this thesis, only message identifiers for which testing had been fully carried out were therefore employed. The completion of conformance testing to cover these missing cases will be relatively straightforward future work. As such, the conformance testing exercises described in this chapter have formed a solid basis for the work described in the remaining chapters, as without them, it would be difficult to justify the practical significance of the research.

5.7 Conclusion

In conclusion this chapter discussed a test facility, which utilises Virtual I/O's and integrated logic analysers to perform CAN conformance testing in accordance with the ISO standards. The facility is capable of performing the full range of tests required for conformance to the relevant CAN standards. A number of tests cases have been demonstrated in this chapter, with details of the tests setup and verification stages.

In conclusion, this facility can be assembled and used for a fraction of the cost of a 'regular' test facility for CAN conformance. It can be seen that this technique is not restricted to the CAN protocol, and with suitable modifications can be used to test conformance of several alternate network protocols, for example, TTCAN [FH00], since TTCAN uses the same data link and physical layers as of the standard CAN.

Chapter 6

Overclocking in Controller Area Network: Higher Information Throughput

As discussed earlier in chapter 1, one of the major reasons for the speed limitation of the CAN protocol is due to the design of the physical layer. The wired-AND nature combined with the bit synchronisation methodology is a major hindrance towards developing increased speed CAN buses. This chapter discusses the possibility of how the IP CAN core may be used to implement an effective modification to increase the speed and overall information throughput of a CAN network.

For this purpose, a scheme to implement the overclocking of a CAN message is employed. This will not only improve the data rate and information throughput, but maintain the priority-driven arbitration mechanism that is a major beneficial feature of the protocol. As will be described, some simple modifications to the IP CAN are used to implement the overclocking. The effectiveness of this adapted controller is illustrated in a series of experiments, in which the data transmission rate is dynamically increased up to 10 Mbps during transmission of the data and CRC fields, and 15-bytes of data are sent; this allows an effective doubling of information throughput per message, coupled with a significantly reduced transmission time.

An insight into how overclocking can be used to effectively reduce the transmission jitter has also been carried out, alongside analysis and experimental work related to

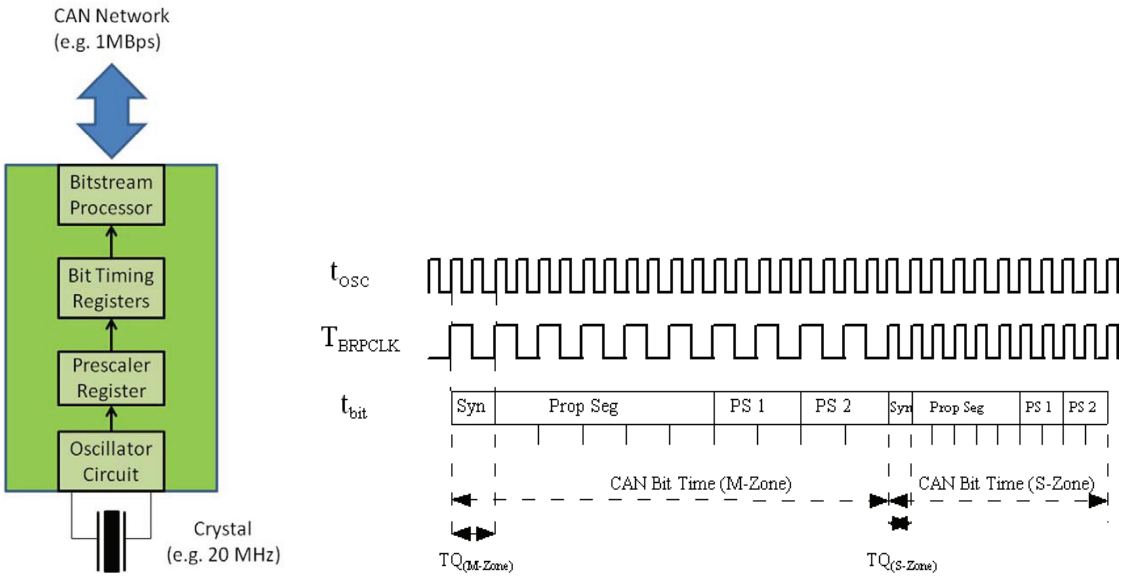


Figure 6.1: (a) set up of a CAN controller oscillator (b) bit time pre-scaling

the calculation of message transmission times and the effects of the technique on expected network bit error rates.

6.1 Problem Formulation

The CAN protocol employs a unique non-destructive priority-based arbitration scheme; when multiple nodes attempt to transmit messages simultaneously, the priority based non-destructive mechanism becomes active and ensures that the highest priority message gains bus access. The wired-AND nature of the physical layer requires that all nodes in the network achieve a logical consensus on the instantaneous bit-patterns appearing on the bus lines. Given the effects of signal propagation delays, it is this requirement of the protocol that acts to severely limit both the maximum transmission speed and bus length of a given CAN network.

Thus, the maximum transmission rate becomes inversely proportional to the CAN

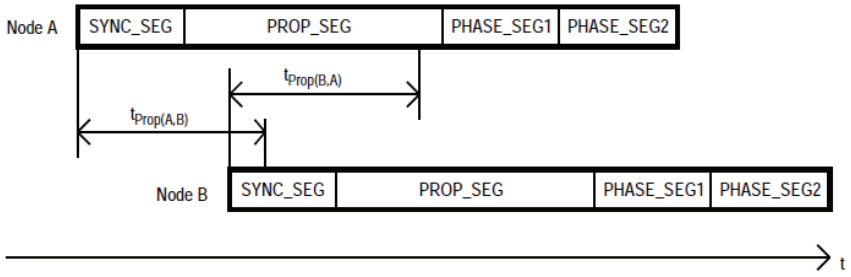


Figure 6.2: Propagation delay between two CAN nodes.

bit time. For higher transmission rates the CAN bit time needs are appropriately shortened. Shorter bit times then put limits on the length of the CAN bus, due to the smaller length of time that is available for signals to fully propagate to the extreme ends of the bus. As mentioned in chapter 3 and also shown in Figure 6.1(b), the CAN bit time is effectively divided into four segments; the second of these four segments is the propagation segment. The length of the propagation segment is configured to compensate for the (worst-case) physical delays due to signal propagation between CAN nodes. The propagation delay is twice the sum of the signals propagation time on the bus line, including the delays associated with the bus drivers. Figure 6.2 elaborates the significance of the propagation segment and its calculation.

6.1.1 Propagation Delay between Two CAN Nodes.

Node A and B are two participating nodes on a CAN bus, and are arbitrating to gain control of the CAN bus. The bit level transmitted by Node A is received by Node B after a time $t_{prop(A,B)}$. The bit level transmitted by Node B is received by Node A after a time $t_{Prop(B,A)}$); the bit level transmitted by Node B must be read by Node A before the end of Node As propagation segment. This will ensure that Node A will correctly sample the bit value. The condition to calculate the propagation time

segment [SK99]:

$$t_{propseg} > t_{prop(A,B)} + t_{prop(B,A)} \tag{6.1.1}$$

Where $t_{propseg}$ is the propagation time segment is explained in chapter 3. Another relation between propagation time segment and bus length is given by [Phi96a](typically for PCA82c250/251 CAN transceivers):

$$\sum_{i=1}^{n} Length_{bus}(i) < \frac{t_{propseg}}{10 * t_p} \tag{6.1.2}$$

t_p is the twisted pair cable's transmission delay typically taken as 5 ns/m [Law97].

Equation 6.1.2 introduces a limit to the bus length. For a transmission speed of 1 Mbps the Bit time is 1 μsec. Since the CAN bit time consists of 4 segments [Bos91] there is a limitation on the $t_{propseg}$which should be less than or equal to 50% of the total bit time [Phi96a]. In this case applying Equation 6.1.2 and taking $t_{propseg}$ to be 50% (maximum possible value) of the bit time, i.e. 500 ns, the bus length is calculated to be less than 10 m. Hence for transmission speeds higher than 1 Mbps the length of the CAN bus reduces drastically due to this need for bit-level consensus during multiple transmissions on the CAN bus. Clearly, since the principal use of CAN is in distributed embedded control systems, such restrictions on the bus length restrict the applicability of overlocked CAN networks significantly.

6.2 Solution Outline

A relatively simple technique to overcome this problem was first given in a short communication by Cena & Valenzano [CV99]. The basic idea they described which was also outlined for a simple control LAN several years earlier by the UITRON team [MMTS96] is described below: The technique is based upon the observation that a

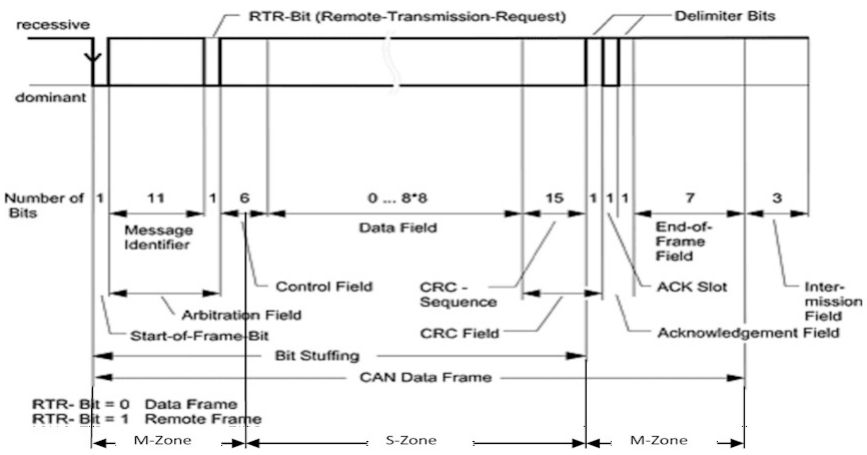


Figure 6.3: CAN frame format showing M and S zone.

CAN frame may be divided into several different zones, as depicted in Figure 6.3. The M-zones of the frame are the portions in which multiple writers are needed (i.e. during arbitration, acknowledgement, EOF sequence, and inter-message space). The S-zones of the frame are those portions in which only a single writer is needed and instead allows multiple writers to access the bus during an S-zone, which leads to bus errors. The S-zone principally consists of transmitting the DLC, payload and CRC.

During the S-zone, the single node which has already won the arbitration during the first M-zone becomes the only transmitter (except of course if an error occurs on the bus). This clearly reduces the restrictions on the length of tolerated propagation delay during the S-zone. Since only a single node is transmitting on the CAN bus, there is no need to compensate for the signals return path delay. The value of $t_{propseg}$ can easily be taken as half when only a single transmitter is active on the CAN bus. Considering Equation 6.1.1, the value of $t_{prop(B,A)}$ can be taken as zero. This allows a shorter bit time during the S-zone, maintaining the same network length.

For the example given in section 6.1.1 a speed of 2 Mbps can easily be achieved

on the CAN bus. The CAN bit time for a speed of 2 Mbps is 500 ns. For a bus length of 40 m one way propagation delay is 200 ns and keeping the propagation segment at 50%, $t_{propseg}$ is calculated to be 250 ns. This value of 250 ns is enough to keep bit-level consensus with a single transmitter on the CAN bus of length 10 m hence allowing us to reduce the overall CAN bit time to be 500 ns. This gives flexibility to operate with normal transmission rates during the M-zones. When the S-zone is entered, the CAN network can be overclocked to increase the transmission rate of the Data and CRC field.

6.3 Overclocking

Overclocking in CAN networks is a technique that may be used to artificially decrease the bit-time and hence the transmission time of a message. As illustrated in Figure 6.1(a), each CAN controller in a given network is driven by a local clock source (typically a crystal oscillator circuit). This clock signal is typically stepped down by an internal pre scaler register, before being used to drive - via the bit timing registers - the protocol controller and bit stream processor. As such, overclocking most often refers to the deliberate use of a faster oscillator - in combination with appropriate settings in the pre-scaler and bit timing registers - to increase the transmission rate. This is shown in Figure 6.1(b) with different T_{BRPCLK} (baud rate pre-scalar clock) for S-zone and M-zone. The baud rate pre-scalar clock is used to drive the CAN bit time and the transmission rates of the CAN bus.

In a time-triggered implementation of a CAN-based system, the overclocking problem does not pose many serious problems, save perhaps for certification issues. This is

because when an appropriate higher-level protocol is employed, it is possible to statically design a message schedule such that no run-time message collisions - and hence message arbitrations - actually take place. An example of such a higher-level protocol is the shared-clock family of algorithms, described by Ayavoo et al [APSP07]; with appropriate modifications, such algorithms are also suitable for use in redundant networks and may even be implemented in hardware [SP07]. Since no run-time collisions take place, the arbitration mechanism becomes redundant and there is no need for system-wide consensus as only a single node is in control of the bus. If an adequate technique is employed to deal with the ACK slot (e.g. by using message self-acknowledgement) the CAN controllers may be overclocked by a significant factor, resulting in a proportional increase in data throughput. For event-driven and hybrid systems, however, the arbitration mechanism must be heavily relied upon for effective use, and simply overclocking the CAN controllers is not guaranteed to work; the requirement for node-wide temporal consensus of bit values will be violated by the signal propagation delays as described previously in this section.

6.4 Modified CAN Controller Development

The CAN IP core which had been developed as discussed in previous chapters has been modified for the overclocking implementation described in section 2. A test bed has been created around this controller, the general structure of which is shown in Figure 4.12. For clarity the figure shows a system employing two CAN nodes, but the test bed may be extended as required to encompass an arbitrary amount of nodes. In the current study, the system was primarily based around four CAN nodes running the modified CAN protocol.

6.4.1 Issues Setting up the Test Bench

A number of features were required to be added to the controller during the setting up of the test bench to support the overclocking approach; these are now described in detail, along with the issues that were encountered.

1. In order to implement dual-rate CAN, the modified controller has an additional Dynamic Bit Timing Register (DBTR) added into its structure. This register contains the settings to be used when adapting to the higher data rate (for this study the rate employed was 5 Mbps although rates of up to 10 Mbps have been achieved during transmission of S-zones). An example code is illustrated below to show how dual rates are employed at different stages of the CAN field.

```
/* Different baud rate pre-scalar settings  for M and S zone */
assign clk_cnt_fix = (data_field|crc_field)?(baud_r_s_zone + 1'b1)<<1
 :(baud_r_m_zone + 1'b1)<<1;
always @ (posedge clock or posedge reset)
begin
  if (reset)
    brp_clk_cnt <= 7'h0;
  else if (brp_clk_cnt >= (clk_cnt_fix-1'b1))
    brp_clk_cnt <=#Tp 7'h0;
  else
    brp_clk_cnt <=#Tp brp_clk_cnt + 1'b1;
end

/* Bit clock enable to calculate the time quanta */
always @ (posedge clock or posedge reset)
begin
  if (reset)
    bit_clk_en  <= 1'b0;
  else if ({1'b0, brp_clk_cnt} == (clk_cnt_fix-1'b1))
    bit_clk_en  <=#Tp 1'b1;
  else
    bit_clk_en  <=#Tp 1'b0;
end
```

2. In order to enable effective switching, a state machine was used to generate a signal which is asserted only for the duration of the S-zone. During this time the baud_r_s_zone setting is employed instead of the standard baud_r_m_zone as shown in the above code listing.

3. In order to handle error signalling during the S-zone (eg. bit or stuff errors), the state machine was set to de-assert the switching signal in case of a detected error or error frame; the only acceptable behaviour in the network is for the controllers to dynamically resume to the lower data rate, as the entire message, including M-zones, is required to be retransmitted in this case.

4. The foremost issue that was encountered while developing this modified CAN controller was whether the existing CAN transceiver interface can reliably support data rates higher than 1 Mbps; experimentally, messages were successfully transmitted and received in the 4-node network with a S-zone configuration of 5 Mbps, using commercially-available transceivers.

5. The test bench connected CAN IP nodes on a network length of 10 meters, as this was an experimental set up, hence could not be evaluated for increased lengths due to wiring and noise issues.

6.4.2 Basic Transmission and Reception of Overclocked Messages

This section will briefly describe the behaviour of the transmitter and receiver nodes using Chipscope snapshots; as mentioned, for this experiment the M-zone data rate is 250 Kbps and the higher data rate S-zone is 5 Mbps. Figure 6.4 depicts the transmitter

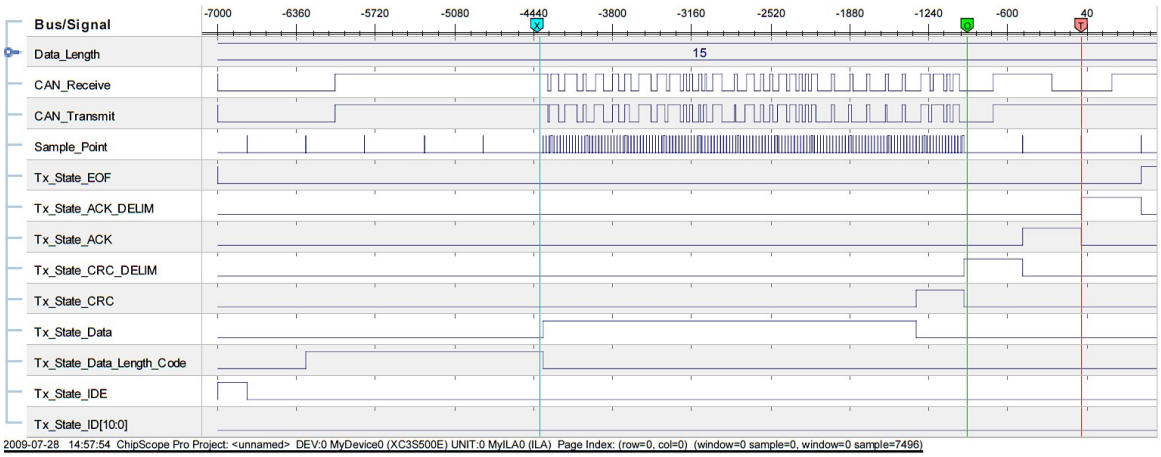


Figure 6.4: Overclocking implementation transmitter node

behaviour during frame transmission. From the figure, the message transmission on the CAN bus can be observed via the CAN Receive and CAN Transmit signals showing the bits being sent and received simultaneously. Additionally, for clarity the figure illustrates several important transmit states which correspond to fields of the CAN frame [11893]. The sample point signal is critical when switching between data rates; Figure 6.4 shows the assertion of the switching signal (Transmit_State_DATA) once the S-zone field has been entered, and the sample point can be seen occurring 20 times faster during the Transmission_State_DATA and Transmission_State_CRC. It can also be observed that the Data_ Length field shows 15 which represents the data payload size in bytes. A successful reception of the overclocked message can also be observed in Figure 6.5, which is a Chipscope snapshot taken on one of the participating receiver nodes. When the S-zone is exited, the switching signal is de-asserted and the sampling frequency reverts back to the lower rate. The successful transmission and reception of the message is demonstrated by the various transmit

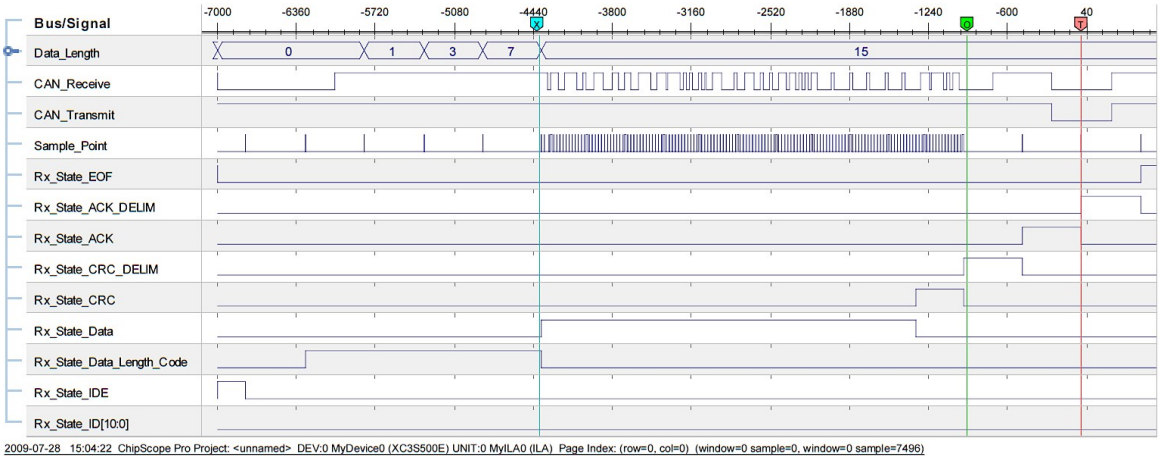


Figure 6.5: Overclocking implementation receiver node

and receive states illustrated in the figures; in Figure 6.5, an acknowledgement signal is sent by the receiver node as demonstrated by Receive_Ack_Delimiter field, and the CAN_Transmit signal becomes dominant.

To further verify the behaviour of this modified CAN controller, two test cases were specifically created to illustrate the operation of two of the main features of the protocol; the arbitration mechanism and error signalling. These tests are discussed and documented in the following sub-sections.

6.4.3 Arbitration

Figure 6.6 shows the arbitration phase on the CAN bus. The experiment was set up such that two nodes were simultaneously competing to gain access of the CAN bus. At this point, the arbitration scheme should ensure that the higher-priority message wins bus access, and be transmitted first. In this case the intent is to show that an overclock CAN node losing arbitration will change its state to a receiver node and will

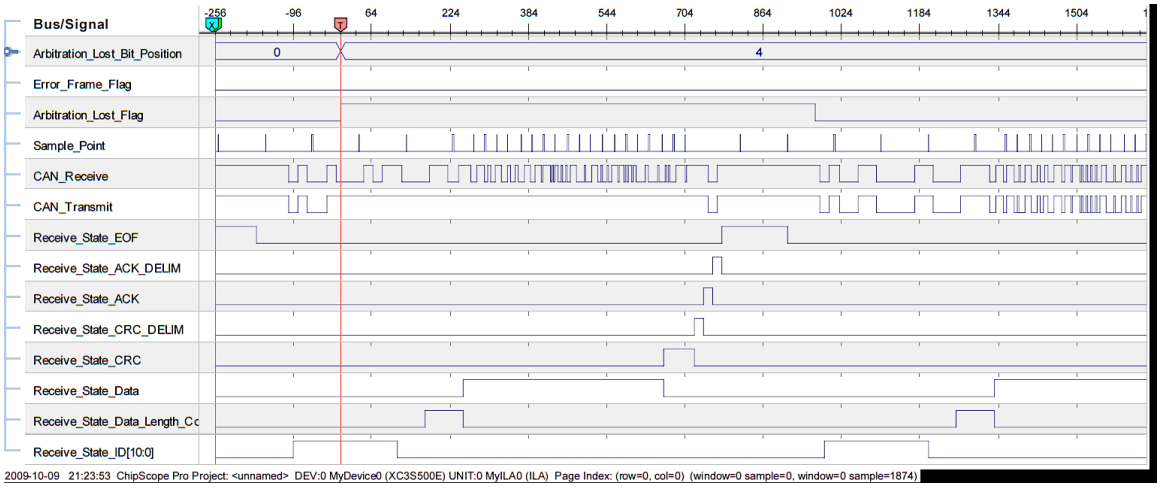


Figure 6.6: Arbitration in overclocking CAN

switch back and forth into higher and lower data rates successfully. Some significant observations had been taken during this process:

1. M-zone rate of communication for this test case is 1 Mbps and for S-zone is 5 Mbps.

2. Marker *O* shows the Start of frame point, while the bit values between marker *O* and *T* demonstrate the transmission of identifier field; also shown by high state of signal Receive_State_ID [10:0].

3. Marker *T* shows the CAN_Transmit signal changing to recessive state for the rest of the message fields showing that the node has lost arbitration and has turned into a receiver.

4. The value shown on the Arbitration_Lost_Bit signal is *4* which means that the arbitration round was lost on the 4th bit of the identifier.

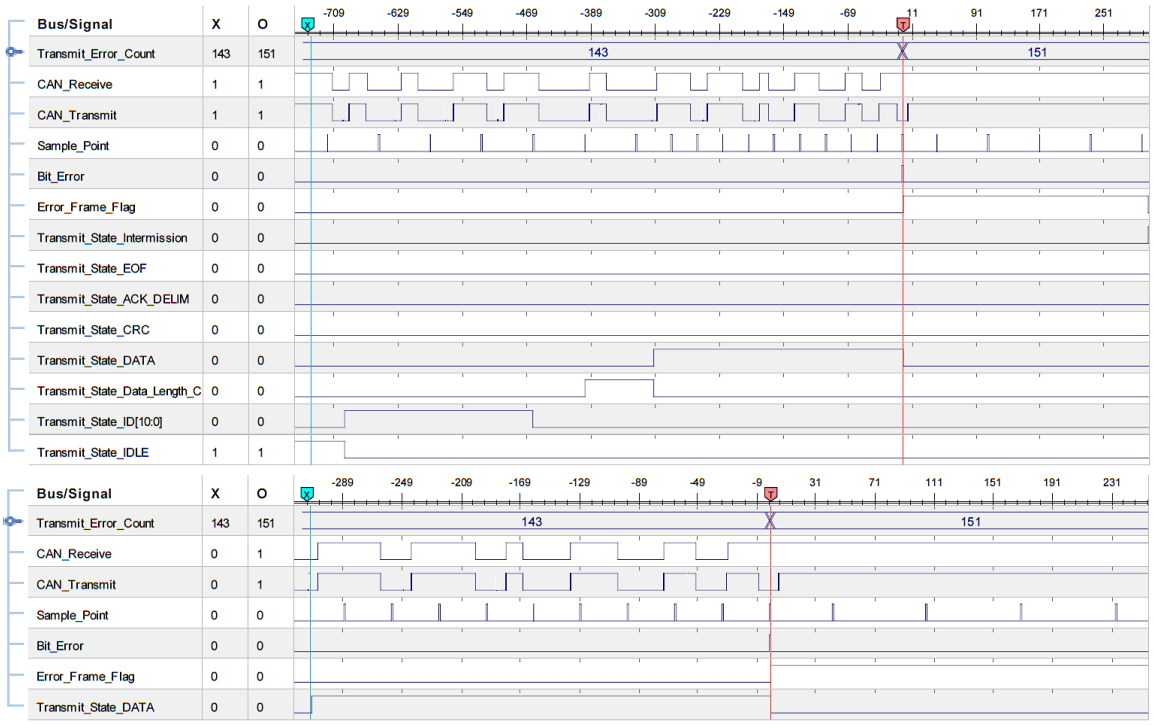


Figure 6.7: Error signalling and switch back in overclocked CAN

5. The switch over during S-zone can be seen, as with the successful reception of the message; CAN_ Transmit is pulled dominant at Receive_State_ACK_Delimiter field.

6. After Marker *X*, the pending message previously losing arbitration is now transmitted, as demonstrated by the CAN transmit/receive values and state registers.

6.4.4 Error Signalling

The second test case is related to error signalling, and demonstrates that in case of an error while data is being transmitted on the higher data speed, the controller

will dynamically switch back to its normal speed. In this test case a bit error was generated on the CAN bus by corrupting a dominant bit being transmitted while the sampled bit is recessive (the error was injected using an additional ARM 7 development board connected via transceivers with the CAN bus). Figure 6.7 shows a transmitter snapshot, and the following observations can be made:

1. Marker X demonstrates the start of a normal transmission, illustrating the different CAN fields during the initial phase of transmission.

2. At marker T the CAN_Transmit signal is recessive, whilst the CAN_Receive signal is made dominant by the injection of the bit error.

3. Before marker T on the Transmit_State_Data, the increased sample point frequency can be seen; as soon as the bit error occurs, and the start of the error signal begins, the sample rate switches to its normal rate.

4. The Error_Frame_Flag field (immediately after marker T) shows the transmission of the error frame at the lower M-zone data rate. Also observe that the transmit error counter has increased from **143** to **51** as per the protocol specification [11893].

These observations clearly states that the modified CAN IP core switched to normal data rate while signalling an error frame. An error frame is considered a multiple writer event on the CAN bus, hence a two way propagation delay is necessary to keep consensus between the participating nodes.

These two test cases demonstrate that the modified CAN controller, with enhanced functionality, can maintain conformance with the arbitration scheme and error handling requirements mandated by the CAN protocol standard.

6.5 Analysis

With respect to implementing a real-time system based on the modified CAN controller, there are several significant areas of analysis required. Firstly, any schedulability analysis of real-time network traffic needs accurate information of several parameters, including the worst-case transmission time of a given frame [KLSC00]. As such, some way to determine the worst-case length of a given message (and hence its transmission time) is needed. Secondly, for a dependable system, the expected probability of a transmission failure in a given set of environmental conditions should be investigated [GTA06]. This section will begin to explore these issues from both analytical and experimental perspectives, beginning with transmission time analysis.

6.5.1 Experimental Evaluation

For evaluation of experimental results a test bench was set up. As before, the test bench consisted of 4 CAN nodes modified to support overclocking. One CAN node was configured as a transmitter while the other three nodes were configured as transmitter/receivers, set up to simulate real-world CAN traffic. The transmitter node was set up to periodically generate a normal message every millisecond; the message length and contents were generated pseudo-randomly in this case. In addition, in between every 50 normal messages, a best or worst-case length CAN message was also inserted, alternating between the two extremes. An independent ARM board with a high-precision timer was set up to receive two signals - one from a transmitter and one from a receiver node on the start and completion of a message simultaneously. These signals were time-stamped, this was used to capture the message transmission times. Tests were conducted over a range of 2 to 24 hours continuously with normal

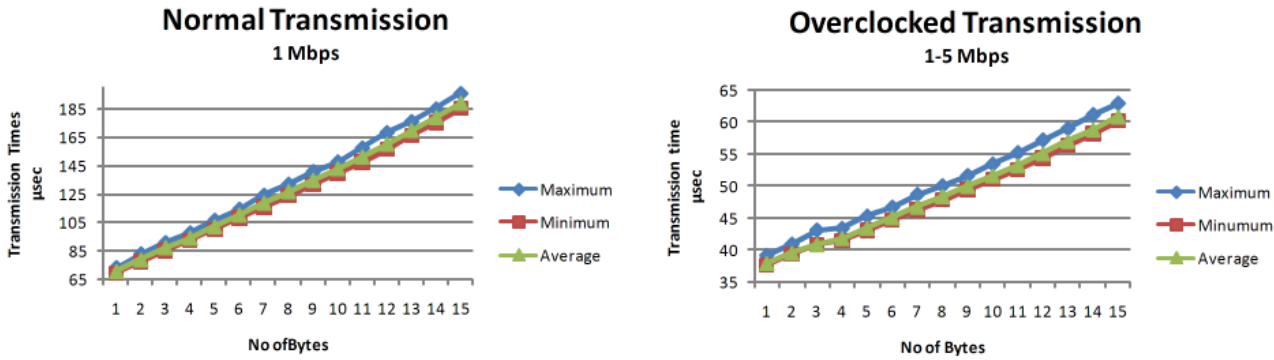


Figure 6.8: Normal and overclocked transmission times

transmission rates of 1 Mbps and a second test with overclocked data rate of 5 Mbps.

The results of this experiment are as shown in Figure 6.8. Several key observations may be made from these results; firstly, the immense reduction of transmission times due to overclocking; secondly, the average difference (jitter) between maximum and minimum transmission times is 2μsec for the overclocked system, as compared to 9 μsec for the regular system. This latter point indicates a decrease in transmission time variations (jitter), which is indicative of the potentially beneficial effects the overclocked technique may have on the predictability of CAN networks.

6.5.2 Analytical Analysis

According to CAN standard [Bos91] the total number of bits in a CAN message prior to bit stuffing is given by:

$$8d_i + g + 13 \tag{6.5.1}$$

A previous analysis of the bit stuffing mechanism by Nolte et al. showed that the worst-case number of bits to be transmitted in CAN frame i is equal to (assuming a

standard frame format) [NHNP01]:

$$8d_i + g + 13 + \lfloor \frac{8d_i + g - 1}{4} \rfloor \tag{6.5.2}$$

The parameter $d_i \epsilon [1, 8]$ is the number of bytes in the payload of frame i. Also the parameter $g \epsilon [34, 54]$ depends upon the CAN frame format. The value of g is 34 for a standard CAN frame and 54 for an extended frame format simultaneously. The term inside the floor function represents the worst possible number of stuff bits that can be added to the standard CAN frame. Since a CAN frame employed in the dual-rate controller has been sub-divided in two M-zones and a single S-zone; both the first M-zone and the S-zone are subject to bit stuffing. With respect to Figure 6.3, and labelling the number of bits transmitted in the two M-zones as m_1 and m_2 from left to right, the equations of Nolte et al. may be adapted to determine the worst-case number of bits transmitted in each zone:

$$\begin{aligned} m_1 &= 13 + \lfloor \frac{13}{4} \rfloor \\ s &= 21 + 8d_i + \lfloor \frac{21 + 8d_i - 1}{4} \rfloor \\ m_2 &= 13 \end{aligned} \tag{6.5.3}$$

Equations 6.5.3 is derived by analysing Figure 6.3. In this analysis CAN standard frame is under consideration hence the value of g is taken as 34. m_1 represents the first M-zone before the start of the Data field of the CAN frame. In the first M-zone the first 13 bits i.e. the SOF, 11 Bit identifier field and RTR bit is exposed to bit stuffing hence $m_1 = 13 + \lfloor \frac{13}{4} \rfloor$ where the terms inside the floor function represents the stuffed bits added to the fixed 13 bits.

The worst case length of the S-zone is represented by s in the Equation 6.5.3. The fixed value of 21 represents the 15 bits of CRC field, 4 bits of the data length code

field and the 2 bits represent the reserved bits r0 and r1. Since all the 21 bits and the 8 possible data bytes are exposed to bit stuffing hence the term given inside the floor function represents the number of stuff bits, the term $21 + 8d_i - 1$ is divided by 4 because the bit stuffing is applied after every consecutive 5th bit as shown in Figure 6.9.

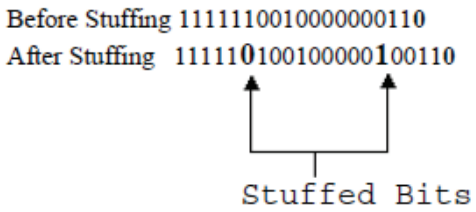


Figure 6.9: CAN bit stuffing

m_2 represents the last 13 bits of the CAN frame i.e. the CRC delimiter, 2 bits of acknowledgement field, 7 bits of EOF field and 3 bits from intermission field. All these CAN frame fields are of fixed format and are not exposed to bit stuffing, even any violation of these fields format is considered as a form error [Bos91] by the CAN nodes.

Note, however, that as demonstrated in this work the adoption of an IP core design allows a developer to change key protocol parameters with relative ease; suppose that D bits are required to encode the DLC, and C bits are chosen for transmission of the CRC. In this case, calculation of the number of bits transmitted in the S-zone in Equation 6.5.3 can be modified as follows:

$$s = 2 + D + C + 8d_i \lfloor \frac{2 + D + C + 8d_i}{4} \rfloor \tag{6.5.4}$$

Given knowledge of the CAN bit time τ, calculating the transmission time from

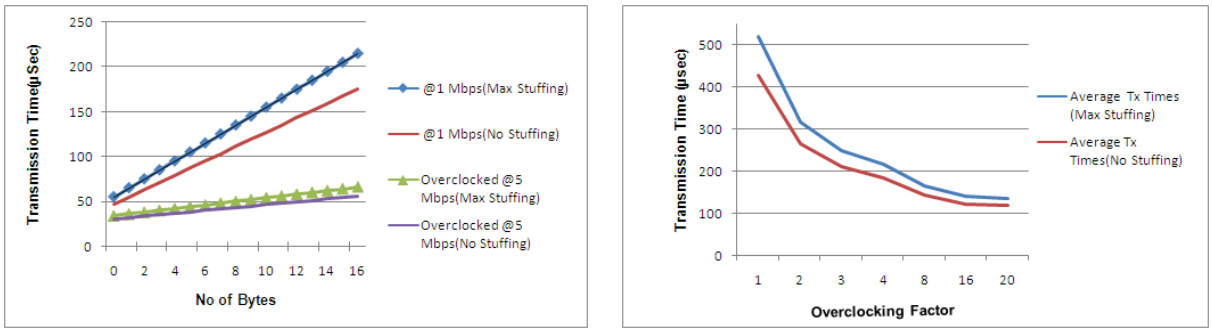


Figure 6.10: Transmission times variations with (a) bit stuffing (b) variable overclocking factors

this relationship is straightforward. In case of overclocking, two separate bit times for the M-zone and S-zone is given by τ_m and τ_s, Equation 6.5.5 may be used to calculate the message transmission time from these relationships given in Equation 6.5.3:

$$c_i = 29.\tau_m + \left[21 + 8d_i + \lfloor \frac{21 + 8d_i - 1}{4} \rfloor \right] .\tau_s \tag{6.5.5}$$

In Equation 6.5.5, the first term on the R.H.S of the equation i.e. $(29.\tau_m)$ represents the time taken to transmit the two M-zones, the algebraic value 29 is derived by adding m_1 and m_2 from Equation 6.5.3.

6.6 Case Studies

As was mentioned earlier in this chapter, as may be expected the use of overclocking will generally lead to a decrease not only in transmission times, but also transmission time variations (jitter). Two different test studies were conducted to further explore this relationship. In the first case, the transmission time variation of CAN Frames at 1 Mbps (with no overclocking) was compared with that of 5 Mbps overclocked messages, and the results are shown in Figure 6.10(a). Data was gathered for messages

employing both worst and best case bit stuffing. As can be seen in the Figure, the results are interesting in terms of transmission time variations; frames without overclocking have a comparatively larger spread in their transmission times, which increases as the number of data bytes sent is increased. The overclocked frames have significantly less spread between the best-case and worst-case stuffing. For example, with 16 byte messages sent at 1 Mbps with an S-zone overclocked at 5 Mbps, the difference of transmission time between best-case and worst-case stuffed messages is only 10 μs. For a regular 1 Mbps CAN message, this variation increases to 40 μs. Thus, the data indicates that a fourfold reduction in transmission time jitter may be achieved with a five-fold increase in clock speed during the S-zone.

The relationship between overlocking factor and transmission jitter was further investigated with case study two, during which the overclocking factor was varied between a factor of 1 and 20. The base (M-zone) transmission rate for all the cases was 250 Kbps. The S-Zone transmission rate was varied from 250 Kbps - i.e. an overclocking factor of 1 - to a transmission rate of 5 Mbps - i.e. an overclocking factor of 20. The results from this case study are as shown in Figure 6.10(b). The average transmission times for each case with best-case and worst-case stuffing was again observed.

The results illustrate that with a linear increase in the overclocking factor, the message transmission time variation is also reduced by a proportional factor. However, it is clear that since the M-zones cannot be overclocked, the message jitter cannot be removed in its entirety by simply overclocking and a lower limit will be reached; this effect can be observed for overclocking factors greater than 16. An intelligent

choice of message identifiers could further reduce the possibility of stuff bits in the M-zones [NHNP01], and hence reduce the overall jitter even further. However, a further modification to the protocol to weaken the lower limit which also allows for arbitrary identifiers will be described in a later chapter.

6.6.1 Transmission Failure Rates

In addition to the determination of message transmission times, another important measure of suitability of a real-time network in a given application is the expected level of transmission errors in the target environment. Since most real-time control systems are expected to achieve a certain target failure rate, this information can be of vital importance when planning a design. The expected target failure rate depends primarily on the number of messages transmitted in a given time frame and the expected probability that a message will fail during transmission. The message failure rate probability λ is largely determined by the bit error rate (BER) and the number of bits to be transmitted. For a standard CAN frame with b bits, assuming uncorrelated error arrivals the success or failure of the reception of a single-bit may be modelled as a Bernouilli process [SP07]. In this situation, λ may be approximated as follows [SP07]:

$$\lambda = 1 - (1 - BER)^b \tag{6.6.1}$$

For the case of correlated or partially correlated error arrivals, the underlying model may be changed appropriately (e.g. to a poisson process) as discussed by [BBRN04]. That is, the probability that b bits will not be transmitted error-free. In the case of the dual-rate controller, Equation 6.5.2 may be used to determine for a given frame how many bits will be transmitted (worst-case) in each zone. However, with

respect to the BER, this is known to be dependent on several factors, and for fixed environment conditions - is largely influenced by the transmission speed. Thus, to determine message failure rate, two BERs have to be considered for the two zones, namely BER_m and BER_s. If these two parameters are known, then the equation can be expressed using Equation 6.6.1:

$$\lambda = 1 - \left((1 - BER_m)^{m_1+m_2} \,.\, (1 - BER_s)^s\right) \tag{6.6.2}$$

Although the CAN protocol itself does not include low-level implementation details at the physical layer, to enable reliable device inter-operability such details are mandated in the relevant ISO documents. In any CAN network operating according to ISO-11898, bits are transmitted over a differential two-wire twisted pair; the actual voltages are required to adhere to certain characteristics, and the following nominal voltages should be observed:

1. During transmission of a recessive bit: CAN_HI = CAN_LOW = 2.5 V;

2. During transmission of a dominant bit: CAN_HI = 3.5 V, CAN_LOW = 1.5 V.

This ensures minimum power dissipation when the bus is idle. Since differential signalling is employed, this can be represented as a unipolar NRZ pulse train, with the characteristics shown in Figure 6.11: It is known that the BER of such a pulse train can be approximated as follows [Skl88]:

$$BER = Q\left(\sqrt{\frac{E_b}{N}}\right) E_b = \left(\frac{V^2 T}{2}\right) \tag{6.6.3}$$

Since differential signalling is employed, this can be represented as a unipolar NRZ pulse train, In which E_b is a measure of the average energy level per bit (which is dependent on the chosen bit-time T), N is a measure of the noise energy level, and

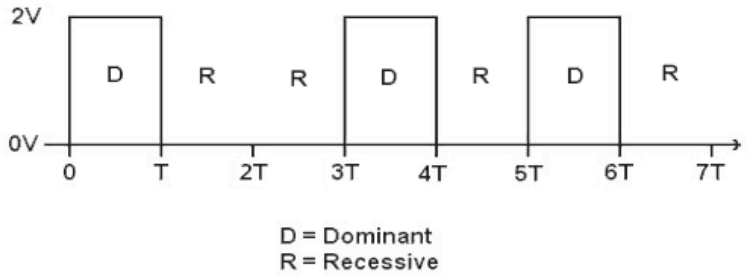


Figure 6.11: CAN NRZ unipolar pulse train

Environment	BER	Estimate of N
Benign	3.0×10^{-11}	9.055×10^{-8}
Normal	3.1×10^{-9}	1.139×10^{-7}
Aggressive	2.6×10^{-7}	1.508×10^{-7}

Table 6.1: Measured BER and calculated noise level for CAN @ 1 Mbps

Q is the complimentary error function. Clearly, E_b is calculable for a given CAN bit rate; additionally, previous experimental work has also attempted to categories the BER for CAN (at 1 Mbps) in various operating environments (benign, normal and aggressive) [FOFF04]. This information may be used to solve for estimates of N for these environments with respect to a CAN network and this information is displayed in Table 6.1 below: These calculations then allow estimates of the likely BER when employing CAN at higher bit-rates to be estimated. In this respect, a limitation of the physical interface seems to become apparent; even in benign environments, this data suggests that operating a CAN network at 5 Mbps is likely to have a dramatic effect on the BER, with an expected value of around 3.0×10^{-3}. Such a value would be wholly unacceptable in most situations; the only solution seems to be increasing

the differential voltage employed by the bus. In this case, increasing the differential voltage to 4 V when signalling dominant bits restores the expected bit error rate to 3.0×10^{-9}. Although further investigations are needed to confirm these findings experimentally, some additional evidence to support these predictions is given in a later chapter. As such, this apparent limitation of the physical layer - the need to scale voltage with bit rate, as with CPU overclocking - should be taken into account by developers before real industrial application of the technique.

6.7 Discussion and Conclusion

This chapter has considered an implementation of a dual-transmission rate CAN controller. The modified controller has been shown to be capable of transmitting CAN frames with an increased data payload size, and at a higher data rate than the protocol normally allows. Although the implementation described clearly does not exhibit backward compatibility with the original CAN protocol, the error and CRC mechanisms are well-defined and experimentally validated through the case study. As such, it can be expected that they are likely to provide error detection and signalling to the same levels as the original protocol.

Given the lack of backwards compatibility, it would seem that this scheme is perhaps best suited to new CAN implementations which require a high rate of information throughput. An alternate application for the technique may be found in situations in which the message set characteristics are found to be unschedulable when using a standard CAN networks (using analysis techniques described in [RVA+98],[BB01]for example). In these situations, overclocking may be applied to one or more of the messages to decrease transmission times (and hence also decrease

the non-preemptive blocking which is inherent in CAN), and improve worst-case response times to appropriate levels. The effect of decreasing jitter levels among messages with variable data may also increase the degree of fairness between CAN nodes with a different amount and priority of data to be transmitted on the bus.

Chapter 7

Jitterless Communication in CAN Networks

With the evolution of mechatronic technology, the increased complexity and distribution of modern control system requirements dictates a need for high-bandwidth networks, capable of delivering information throughput at a higher level with precise timing accuracy. This level is not directly achievable with CAN, or any software-based protocol employed with standard hardware. In the previous chapter, overclocking of the CAN network to increase the speeds up to 10 Mbps was discussed; but the question of variability in message transmission times of the CAN message transmission due to bit-stuffing is still unanswered.

This chapter will describe a mechanism to overcome the jitter introduced in CAN message transmission times due to bit stuffing, implemented by modifying the CAN IP core.

7.1 Bit Stuffing cause of Transmission Jitter

Previous research has focused on adapting CPU scheduling and feasibility analysis techniques to create systems which provably deliver all messages in a timely fashion in CAN networks. As with CPU scheduling, these topics can be broadly divided into two sub-categories; those based on static (table-driven) techniques, and those based on dynamic (non-idling) techniques.

In a statically-scheduled system, all message transmissions are created off-line and stored in some form of table (a message map) which each node makes use of to control the instants in time that messages are allowed to be transmitted. In order to be employed successfully, each node must have an accurate knowledge of the release time and the transmission times.

Although many enhancements to CAN have been proposed (discussed in Chapter 2), each is limited by the fact that they either require additional CPU overheads along with an increase in the complexity of software running in local CAN-enabled nodes, or they require modifications at the hardware level for implementation. Additionally, early experiments have shown that FPGA-based designs can provide the flexibility with which to implement sophisticated schemes, which cannot be implemented by the use of software-based protocols alone. These observations will be used as the basis for proposing further generic enhancements to CAN which are suitable for implementation on PLDs.

7.2 Fixed Length Messages to Reduce Jitter

The current enhancement is to eliminate the effect of jitter due to the presence of stuffed bits, which are unpredictable in their occurrence. A message of similar data length may have variable size and hence unpredictable transmission times. In this proposed technique, an extra field is added to the CAN standard frame which is known as Data Plus, the size of this field is variable and depends upon the worst-case stuffed frame lengths [NHNP01] given by Equation 7.3.1.

The length of data_plus is the difference of the transmitted frame length to the result obtained by Equation 7.3.1. This ensures that each message sent is of the same

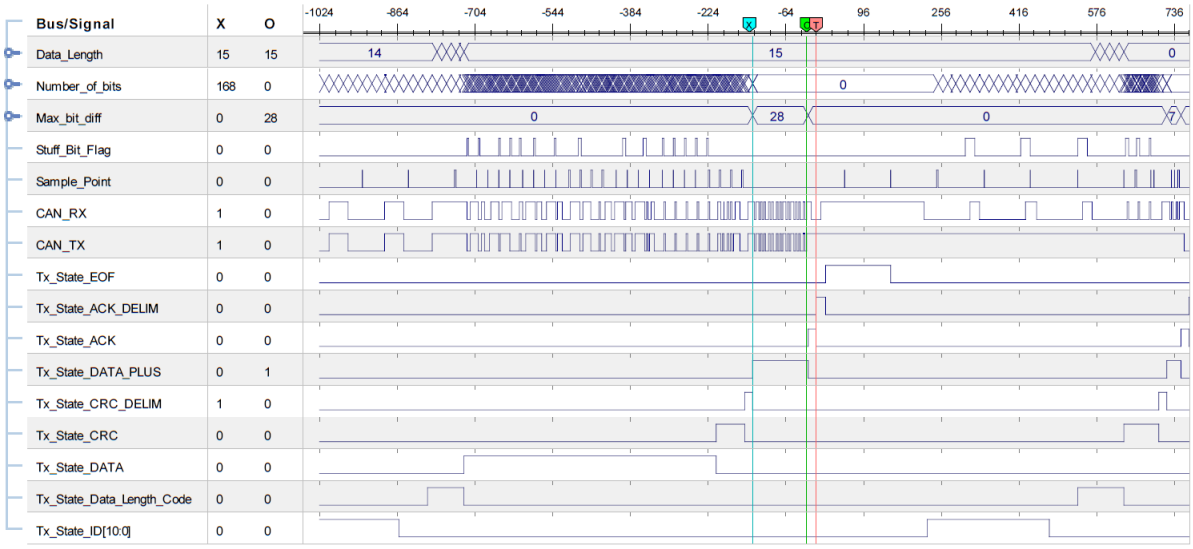


Figure 7.1: Chipscope snapshot of enhanced CAN transmitter

length as a worst-case stuffed frame depending upon the value of d_i which represents the number of data bytes sent in the CAN frame. The working of this enhanced CAN controller can be understood by the given Figure 7.1. Since enhanced CAN IP core can transmit more than 8 bytes upto 1024 bytes and also can overclock in S-zone, see Figure 7.1. A transmission state machine is shown with a 15 byte message overclocked from 1 Mbps to 5 Mbps transmitted successfully.

1. Tx_State_xxx indicates different states of transmission cycle, the states before Tx_State_Data are sending data at 1 Mbps, then the Data, CRC and Data Plus fields are overclocked to be transmitted at 5 Mbps.

2. Number_of_bits field indicates the total number of bits transmitted since the start of the message including any stuff bits. (At Marker X the value indicates 168, i.e. before data_plus field.)

3. The field Max_bit_diff indicates the difference of bits transmitted in a worst case

stuffed frame and the current frame. The value of Max_bit_diff is calculated by subtracting Equation 7.3.1 from Number_of_bits, the valid value is obtained before the start of the Data_Plus field.

4. The Data_plus field will last for 28 bit time and will send alternate 0 and 1 to compensate for the worst case message, (the choice of alternate bits is the minimize the chances of stuffing even in case of erroneous transmission/reception of a single or number of bits). This will make the total bits sent as 196 added the fixed 9 bits of Acknowledgement and End of frame fields.

The enhancement has been designed such that if there is erroneous bit transmission in data_plus field no error frame would be generated. The length of the data_plus field is calculated on the local node depending upon the Data_Length_Code field, therefore there is no possibility of data_plus length error. In case if, there is an error detected in Data_Length_Code field then an error frame is generated and no further transmission take place.

7.3 Case Study

Before the results of the case study are discussed, formation of the test bench is explained below: The test bench consisted of 4 CAN nodes, and each node consisted of a development board with integrated FPGA and a host ARM controller. One of the CAN nodes was working as a transmitter node while the other three nodes primarily worked as receivers but were set up to generate messages randomly to create a real world CAN network. The transmitter node was setup to generate a message at an interval of 1 millisecond, where a message of random length and contents was

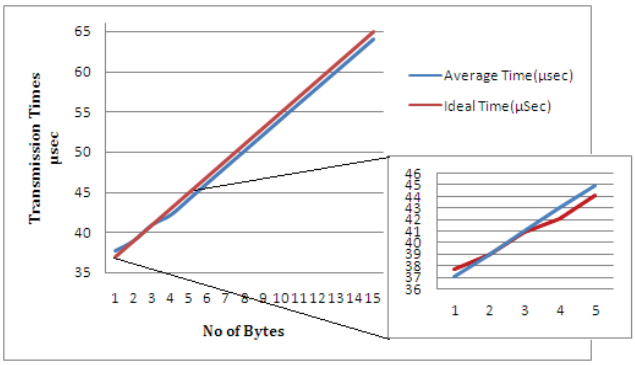


Figure 7.2: Average vs ideal transmission times

generated each time and after every 50 message transmissions a best and worst case stuffed message was inserted. An independent ARM board was set up to receive two signals, one each from transmitter and one of the slave node, this was used to capture the transmission time between the start of a message on transmitter and successful reception on a receiver node. These tests were conducted over different periods ranging from 2 to 24 hours continuously.

The results of the study are given in Figure 7.2 and Table-7.1, the ideal transmission times for bit-stuffed frames has been calculated using Equation 7.3.2 modified from Equation 7.3.1 given in [NHN02].

$$8d_i + 34 + 13 + \lfloor \frac{8d_i + 34 - 1}{4} \rfloor \tag{7.3.1}$$

$$c_i = 31.\tau_m + \left[16 + 8d_i + \lfloor \frac{15 + 8d_i - 1}{4} \rfloor \right] .\tau_s \tag{7.3.2}$$

In which the parameter d_i are the number of bytes in the payload of frame i. τ_m and τ_s are the M-zone and S-zone network bit times. The results in Figure 7.2 indicate the average transmission times of worst case stuffed messages for number of bytes ranging from 1 to 15 compared with the ideal worst case transmission times. It is clearly evident that the average values are very near to the calculated ideal values.

Bytes	Ideal time(μsec)	Average time	Std Dev	Max-Min time
1	37	37.6907	0.1783	3.1734
2	39	38.9368	0.1786	2.9766
3	41	40.9210	0.1674	3.2226
4	43	42.1386	0.1668	3.075
5	45	44.1220	0.1891	3.075
6	47	46.1165	0.1626	2.8782
7	49	48.1082	0.1774	3.2226
8	51	50.0901	0.1807	3.799
9	53	52.0856	0.1574	3.0012
10	55	54.0808	0.1684	3.2964
11	57	56.0700	0.1691	3.1242
12	59	58.0500	0.1707	3.198
13	61	60.0303	0.1757	3.198
14	63	62.0106	0.1672	3.0504
15	65	63.9915	0.1617	3.1242

Table 7.1: Statistics from the case study

Messages with higher data lengths have transmission times of averages less than the ideal worst case values, which is evident in the enlarged view of Figure 7.2, hence strengthening the argument that the addition of padded bits with overclocking mechanism has little effect on transmission times. The statistics presented in Table 7.1 show a comparison between calculated worst case transmission times and observed behaviour of the proposed implementations, such as the standard deviation from the mean values varying in range of 0.1-0.2 μsec. This is a considerable achievement, when compared to the previous techniques, keeping in view the transmission jitter of existing CAN 2.0B messages which are in range of 1 to 34 μsec for message lengths of upto 8 bytes when clocked at 1 Mbps. The Figure 7.3 shows a comparison between the worst and best case transmission times of the proposed scheme and the CAN

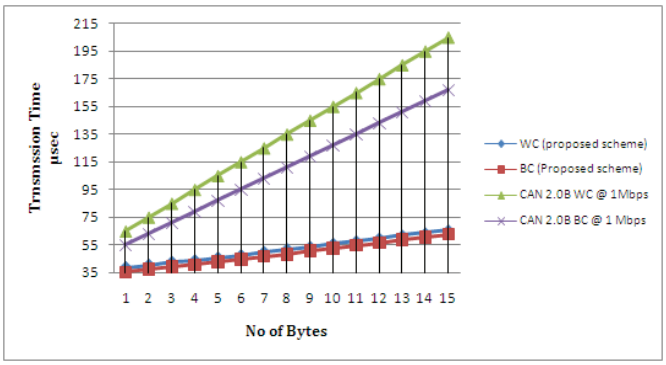


Figure 7.3: Worst case vs best case transmission times

2.0B messages. An evident gap exists between the worst case and best case transmission times. While the statistics of Table 7.1 also shows that the difference between peak maximum and minimum transmission times for various data length messages recorded using the proposed scheme is on average around 3.1 μsec, it is a substantial improvement of 10 times over conventional CAN worst case frame transmission times.

These data would suggest a significant improvement over previous methods, both in terms of transmission jitter and information throughput. For example, Nahas et al [NPS09] quote difference and average jitter figures of 4.1 μs and 0.7 μs respectively when using the SBS technique on almost identical hardware, and sending 5 bytes of random data in an 8-byte CAN frame @ 1 Mbps (note that the remaining 3 bytes are reserved for use by the SBS protocol employed by Nahas et al). As shown in Table 7.1, the data suggests a further improvement over SBS with max-min and average jitter figures of 3.07 μs and 0.19 μs respectively, when sending 5 bytes of data. Also, note that in this case (unlike when using SBS) only 5 actual data bytes need be sent; maximum and average message transmission times are therefore reduced by a significant factor (>50%), thus making potentially significant improvement in the total information throughput.

7.4 Conclusion

This chapter deals in detail with the problems of bit stuffing jitter which can affect the predictability of message transmission times and has proposed and implemented a technique which does eliminate jitter to a minimum level with the expense of some added information. The addition of a data plus field has the ability to virtually eliminate transmission jitter and precisely control the amount of retransmission that is allowed for a given CAN message. The modified controller has been shown to be capable of transmitting CAN frames with an increased data payload size, and the presented techniques are also applicable and effective in these cases. Although initial experiments with the technique indicate its effectiveness, it does have several drawbacks; it seems unlikely that such a scheme can be made backwardly-compatible with existing CAN implementations. When employed in a mixed system with one or more standard CAN controllers, numerous errors (most notably bit stuffing and CRC errors) are observed and signalled. A potential area of future work would be examining any possible impact of the techniques on bit-error susceptibility and message transmission failure-rates in noisy environments.

Chapter 8

Predictable Windowed Transmission

A real-time communication system lays the foundation of a distributed control application [LKJ99], and can be divided into soft or hard real-time, based on the schedules or tasks running in the control application. A hard real-time communication is one where a message delivery under its deadline is a must or otherwise it is taken as a lost or useless message. Meeting time constraints especially in hard real-time messages is a critical requirement and hence appropriate scheduling for transmitting messages is required. In shared embedded communication networks, the scheme to access the bus and taking control of it is of utmost importance for timely transmission of messages. In these networks any node can compete to take control of the medium and hence remaining nodes have to back off, at least till the transmission has finished and the communication medium is free. This can cause uncertainty in designing a time schedule for hard real-time communication. One viable solution is to assign a time slot to each of the participating nodes in a time period, hence no other node would try to access the medium. This type of transmission is called time-triggered communication. A time trigger is a control signal that is generated at a particular point in time of a synchronised global time base.

Many embedded multiple access networked systems employ TDMA-based or "time-triggered" communications, which have several key benefits. Various studies, both

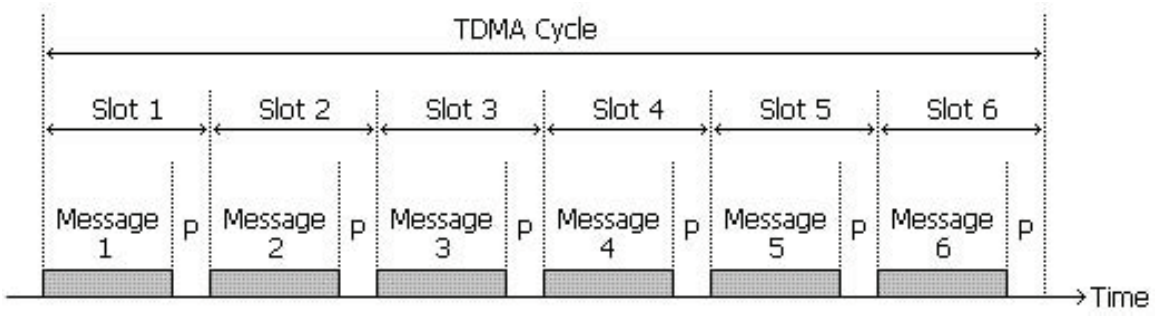


Figure 8.1: TDMA structure with inter-slot spacing

empirical and simulation based, show that it increases the predictability and overall reliability of the network, along with several other benefits (for example, [Kop00, SP07, SPF08]) -have shown that these time line benefits are most often at the expense of message transmission reliability, as the schemes crucially depend upon the enforcement of "single-shot" message transmissions; even a single bit-error during a TDMA slot leads to a message being omitted or discarded.) For critical message streams requiring a specified level of reliability of delivery, duplicated message instances, in the form of redundant slots in the TDMA cycle, will often be required.

8.1 Problem Formulation

CAN supports guaranteed delivery with no upper bound of re-transmission attempts in case of failures. Hence this makes CAN highly supportive for event-triggered or soft-real time communications between unsynchronised nodes [Bos91, Kop00]. A potential drawback with most existing time-triggered CAN implementations lies in the enforcement of single-shot message transmissions; although this effectively bounds worst-case message transmission times, single bit-errors may directly lead to critical message omissions [SP07, BB01].

8.2 Related Work

This section analyses the basics of time-triggered communication, as already discussed with the protocol level enhancements of CAN and other protocols in chapter 2 related to time-triggered communication.

8.2.1 Time-Triggered CAN Communications

A number of hardware and software-based protocol extensions and modifications have been proposed to enable time-triggered communications on CAN; comprehensive reviews are provided by Short & Pont [SP07] and Rodriguez-Navas et al. [RNRP08]. The described techniques tend to rely on the use of a global clock which, in turn, supports a Time Division Multiple Access (TDMA) message schedule. Key to achieving clock synchronisation is the reliable broadcast of time reference messages from a "time master" node. These reference messages are then generally used with a hardware or software-based distributed clock synchronisation algorithm.

The general goal of all these protocols, whether hardware or software based, is the creation of a collision free (and hence arbitration free) bus access schedule, such as that depicted in Figure 8.1 [SP07]. Due to the finite clock error e which always exists between any two clocks in the distributed system, a small inter-slot spacing $P = 2e$ must be employed. In general, designing a message schedule to meet a given set of period requirements is strongly NP-Hard, but in practice many fast algorithms (both optimal and heuristic) are known to generate feasible TDMA schedules (see e.g. [SN06][HBS09]). It is not uncommon to achieve bus utilisations in excess of 90% using such techniques [SP07].

8.2.2 Fault-Tolerant CAN communications

As has been argued by many previous authors, the notion of global time - and the requirement for clock synchronisation has become of great relevance when CAN is employed in dependable applications (e.g. [SP07, RNRP08, FH00]). When messages are required to be sent over multiple redundant CAN channels to improve reliability, then replica determinism becomes a requirement. Replica determinism can be enforced - in part - by the use of single-shot transmissions or upper bounding the latest time that a message may commence transmission. If replica determinism can be enforced, then multiple redundant and fault-tolerant CAN networks may be operated in parallel to increase the reliability of the physical layer.

When messages are subject to interference such as electromagnetic disturbances, this tends to manifest itself as random bit errors on the network. In response to any detected errors, under the CAN protocol an error frame is generated, which may have a length of up to 31 bits [DBBL07]. Due to the automatic retransmission mechanism of CAN, in a real-time system this can be very problematic due to deadlines being missed in a "domino-style" effect [BB01]. Experimental studies would seem to place the Bit Error Rate (BER) for CAN in the region of 10^{-10} in 'benign' environments, increasing to 10^{-6} in 'hostile' environments [FOFF04]. In some extreme cases, BERs as high as 10^{-3} have been reported for vehicles operating in close proximity to sources of large electromagnetic disturbances, e.g. radio transmission stations [GN04].

With this in mind, in a real-time application some form of time out or upper bound is required to limit the worst-case transmission time of a given frame; in a hard real-time system, it is better not to receive a message (if this improves subsequent time lines) than to receive the message late. Unfortunately, such time outs are not provided

as a standard CAN feature; at the expense of local CPU overheads, several techniques have been described to enforce this behaviour. Previous works such as [RNRP08] and [FH00] have suggested that only single-shot transmissions be attempted in the TDMA round. For critical message streams, duplication of messages can be used to achieve the desired reliability. For a message with c bits to be transmitted in an environment with a BER of β, if r message duplicates are sent then the probability of failure is given as follows:

$$\lambda = (1 - (1 - \beta)^c)^r \tag{8.2.1}$$

However, in many cases sending full message duplicates, provides a bandwidth-inefficient solution to this problem. Since bandwidth is relatively scarce in CAN networks, this can be a major issue. Another interesting scheme similar to the current work - is the "Timely CAN" scheme proposed in [BB01]. The author proposed a reliable technique to upper-bound the latest time that a particular message can be scheduled for re-transmission in native CAN networks under an optimal priority assignment, in order to prevent domino-effect deadline misses. First, an estimate of message worst-case response times is obtained (It should be noted that the original analysis provided in [BB01] contains an error due to "push-through" blocking; this was corrected by Davies et.al to add the worst-case error frame overhead in worst-case response time calculations [DBBL07]). From these estimates, the worst-case message transmission times are subtracted to obtain the required "time out" parameters. In the worst-case, a single bit error will still lead to an omission; however if "slack" is placed in the time out, then a limited amount of re-transmission can be achieved. The amount of slack employed can be determined by adding an error model, where error arrivals are treated as sporadic events [DBBL07, PHN00]. In the next section,

an efficient and generic solution (which builds from the work in [SP07] and [BB01]) which is applicable to TDMA-operated CAN networks will be proposed.

8.3 Solution Outline

A notion of the λ-firm real-time constraint is introduced. Such a constraint intends to provide a multi-criteria real-time/reliability guarantee, which can act as an extension to existing time-triggered transmission schemes. A bounded amount of re-transmission is allowed for each message within its pre-defined TDMA slot, providing increased reliability in the presence of errors. Message instances will either be delivered on-time or omitted, with the probability of omission lower than a pre-specified failure rate λ.

An efficient algorithm for calculating optimal message window sizes is presented in this section, to ensure that statistical guarantees of timely delivery in the presence of errors are provided. This is a simple technique to act as an extension to existing time-triggered transmission schemes, in which an effective "window" is defined for each message. A bounded amount of re-transmission is allowed for each message within its defined window; the window size (as opposed to basic frame size) is then used to pack the frame schedule. This effectively provides a bandwidth-efficient compromise between the two extremes.

8.4 Proposed Windowed Transmission Scheme

The proposed windowed transmission scheme is relatively simple. An assumption is made that a CAN message set is made up of n message streams, each stream f being

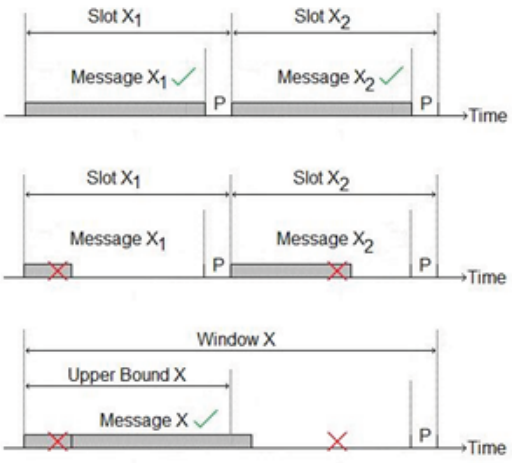


Figure 8.2: Basic concept of a windowed transmission

described by the following four parameters:

$$f_i = (p_i, c_i, d_i, \lambda_i) \tag{8.4.1}$$

Where p_i is the message period, c_i is the message (worst-case) transmission time, d_i is the message relative deadline and λ_i is the target failure rate of the message stream, specified as a probability of unsuccessful transmission per unit time. Note that λ_i may be derived from a higher-level safety analysis, e.g. a Safety Integrity Level (SIL).

8.4.1 Basic Concept

The proposed technique intends to provide the following multi-criteria real-time/reliability guarantee: if a message is delivered, it is delivered on time; if a message is not delivered (omitted), the failure rate for omission is $\leq \lambda_i$. The scheme is best illustrated by way of example. Suppose there is a a critical message "X" that requires two duplicate copies to be sent every TDMA cycle. Figure 8.2 (top) shows such a situation, where

for clarity the duplicated copy is assigned a slot immediately following the original. Suppose that a bit-error occurs in both of these slots, as shown in Figure 8.2 (middle) this will lead to both messages effectively being dropped, which can potentially result in wasted bandwidth, as only single-shot transmission is allowed. However, now consider the situation depicted in Fig 8.2 (bottom), both slots are merged in a single window of length m, where m is specified in network bit times; (re)transmission of message X is only allowed from the start of the slot, up to the point labelled "Upper Bound X", i.e. m-c units after transmission has first been requested. As can be seen, this has a positive effect on the reliability of the message delivery; even when numerous bit-errors occur, the probability of successful message delivery can be maximised.

As shown in Figure 8.2, the slots have effectively been merged and only a single inter-slot spacing is required. In fact, in most cases the required window size can be reduced by a significant amount over sending duplicated single-shot copies; much better use can be made of the available bandwidth. For each of the messages, once the slot sizes m_i have been determined, the TDMA schedule may be created using appropriate techniques. However, this raises an important question; namely, for a given message length and bit error probability, how large does the transmission window have to be to ensure the message will be delivered with the required probability? This question will be addressed in the following two sections, beginning with computation of optimal window sizes.

8.4.2 Computing the Optimal Window Sizes

This Section will first consider how the probability of a successful transmission can be computed for increasing values of m. As with previous works, consider the transmission of successive bits on a CAN network in a noisy environment to be a Bernoulli process, with the probability of success given by $(1-\beta)$, and the probability of failure given by β, where β is the BER. The following theorem establishes that the probability of successfully transmitting a message in a window of size j can be formulated as a one-dimensional Bernoulli sequence problem.

Theorem 8.4.1. *If the probability of an individual bit failure is β, then the probability p_j that at least b bits have been consecutively transmitted without error in a sequence of j bits (with $j > b$) can be calculated recursively as follows:*

$$p_j = p_{j-1} + (1 - p_{j-b-1})\,\beta\,(1-\beta)^b \tag{8.4.2}$$

Proof. by induction on j

Base case: for $j \leq b$

When the number of bits transmitted is less than b, achieving a consecutive stream of b bits (with or without error) is impossible, thus $p_0 = p_1 = p_2 = p_{j-1} = 0$.

case $j = b$:
it is a well known statistical property that
$p_b = (1-\beta)^b$.

Inductive step case: for all $j > b$

p_j must be equal to the summed probabilities of a successful outcome in any one of the previous $j-1$ trials (Σ term), plus the additional probability that a successful sequence has just been received with the jth bit completing the run of b successes (Δ term). Given the recursive solution, taking the summation of p_{j-1} at every step gives the required Σ term, since at step jp_j

$(\Sigma j + \Delta j) = (\Sigma j - 1 + \Delta j - 1 + \Delta j) = (\Sigma j - 2 + \Delta j - 2 + \Delta j - 1 + \Delta j) = (\Sigma i < jp_i + \Delta j)$.

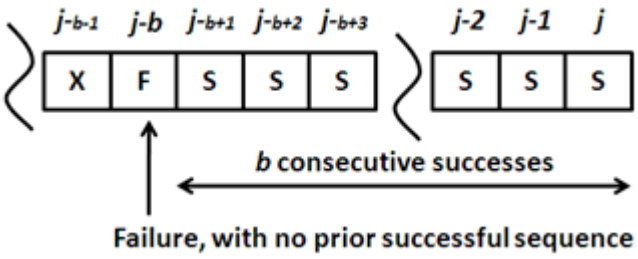


Figure 8.3: Conditions required for a successful outcome at the j^{th} step

Considering the Δ term, as shown in Figure 8.3 the following series of events must occur:

1. At step j, the last b bits have been received without error; i.e. bits $j-b+1$ though j inclusively have been successes, with combined probability $(1-\beta)^b$.
2. The bit received at step $j-b$ must have been received in error to start the sequence (else there was b+1 or more consecutive successes), with probability β.
3. Prior to this - at step $j-b-1$ - no successful run of b consecutive bits had been received, which by definition of p_j is given by the probability $(1-p_{j-b-1})$.

Thus the Δ term consists of the combined probability of these events occurring, i.e:

$\Delta j = (1-p_{j-b-1}).\beta.(1-\beta)^b$.

Thus stating that

$p_j = (\Sigma j + \Delta j) = p_j - 1 + (1-p_{j-b-1}).\beta.(1-\beta)^b$.

Hence the theorem is proved. □

From this result, a simple algorithm may be derived to determine the optimal window size for a particular message. This is the subject of the next section.

8.4.3 Optimal Window Sizing Algorithm

A relatively simple algorithm is derived in the last section, to determine the smallest value of m (window size) such that a message requiring b bits can be transmitted with a failure probability $\leq \lambda$, in an environment having a BER of β. It is clear that

the loop between lines (1) and (2) of the algorithm 8.4.1 will not terminate until the first value of m such that $(1.0 - p_m) \leq \lambda$ has been found, or m exceeds the message relative deadline d which is supplied as an input parameter and expressed in network bits.

Algorithm 8.4.1: WINDOWED TRANSMISSION($lower, upper$)

procedure OPTIMAL_WINDOW(b, λ, β, d)
for $i \leftarrow 0$ **to** $b - 1$
do $\{ p_i := 0;$
$p_b := (1 - \beta)^b;$
$m := b;$
while $((1.0 - p_m) > \lambda)$ **and** $(m < d)$
do $\begin{cases} m := m + 1; & (1) \\ p_m = p_{m-1} + ((1 - p_{m-b-1}).\beta.(1 - \beta)^b); & (2) \end{cases}$
return (m)

8.4.4 Algorithm Analysis

Given that p_j is a monotone increasing function of j (due to the Σ term combined with the Δ term, which is always > 0 for input values $\lambda \in (0, 1)$ and $\beta \in (0, 1)$), the algorithm is guaranteed to converge. However, for small λ and large β, convergence may take some time; given the real-valued representation of these parameters, it is very difficult to provide a (worst-case) run-time bound on the time complexity of the algorithm, when the input is expressed in bits. In a real-time application, however, it can be observed that the message can never meet its deadline (with probability less than λ) if its window size exceeds the message deadline. As such, the additional

termination condition applied in the 'while loop'($m < d$) bounds m such that the algorithm terminates when $m = d$, where d is the relative deadline of the message, expressed in bit times. Note that an appropriate error message can be generated at this point. As with response time analysis, this effectively bounds the time complexity to be pseudo-polynomial in the task parameters. For n message streams, optimal window sizes may therefore be computed with complexity $O(nd_{max})$.

In terms of the required memory storage space, it can be noted that only the last $(b + 1)$ values are required to be stored for the recursion. Therefore an array of size $b + 1$ can be used to limit the required memory storage of the algorithm; this array may be indexed $m|b + 1|$ to effectively overwrite old calculations. Since b is upper-bounded by the very nature of the CAN protocol a message can only carry up to 8 bytes - the memory storage requirements have complexity $O(1)$.

8.4.5 Illustrative Example

Example 1: Suppose a message with $b = 5$ bits required to be transmitted, with a failure probability $\lambda = 0.1$, in an environment having a BER of $\beta = 0.2$. Running the algorithm described above on this example yields the values as tabulated for p_m in Table 8.1, terminating when the target failure rate is achieved with m = 19 and the probability of success $p_{19} = 0.91$. By contrast, only duplicate copies of the 5-bit message is allocated, then according to Equation 8.2.1, 6 copies (= 30 bits) would be required; the windowed approach leads to a saving of 36.7% in terms of the number of bits requiring transmission.

Example 2: Suppose there is a critical CAN message using 11-bit identifiers, requiring 8 bytes of data thus $b = 166$ bits are required to be transmitted (135 bits in

m	p_m	m	p_m	m	p_m	m	p_m
0	0.00	5	0.33	10	0.66	15	0.83
1	0.00	6	0.39	11	0.70	16	0.86
2	0.00	7	0.46	12	0.74	17	0.88
3	0.00	8	0.52	13	0.78	18	0.89
4	0.00	9	0.59	14	0.81	19	0.91

Table 8.1: Computing the optimal window size for example 1

the main message plus 31 bits for a worst-case error frame [DBBL07]), with a period $p = 100ms$. The safety integrity level of the system specifies a target failure rate of $\lambda = 1.0 \times 10^{-9}$, in an environment with a BER of $\beta = 2.6 \times 10^{-7}$. Since 36000 messages are to be sent every hour, the probability that a single instance of this critical message is not delivered should be $\leq 2.78 \times 10^{-14}$. Running the algorithm described above on this example reveals that the target failure rate will be achieved with $m = 489$, and the probability of failure with this size window is $p_{489} = 1.34 \times 10^{-14}$. By contrast, if we must allocate only duplicate copies of the 166-bit message, then according to Equation 8.2.1, 4 copies (= 664 bits) would require transmission. In this case the windowed approach leads to a saving of 26.4% in terms of the number of bits requiring transmission.

8.4.6 Bursty Links

The analysis given in the sections above considers only the case of uncorrelated error behaviours. In order to begin to consider the effects of burst errors, a basic approach would be to consider β to be the burst error rate, with each burst having a duration of exactly u bits. In this case, bursts can be considered by adapting the delta term as follows:

$$\Delta j = (1 - p_{j-b-u}).\beta.(1 - \beta)b; \tag{8.4.3}$$

In other words, the error in j-bth bit is in fact the last in a sequence of u consecutive errors, and taking the convention that $p_j = 0$ for $j \leq 0$. With $D = 1$, the bit and burst error behaviours become identical. However, this may result in somewhat conservative values of m. Employment of more detailed error models (e.g. based on Markov chains [Gil60]) which are known to accurately model burst behaviours, should improve the situation. The next section describes some practical issues related to the implementation of the proposed transmission scheme.

8.5 Implementation Issues

In order to successfully implement a windowed transmission scheme, as prospective designers may choose either software-based or hardware-based solutions, or a combination of the two, some of the merits of both approaches are given:

8.5.1 Software-Based Solution

Some modern CAN controllers now have direct hardware support for single-shot messages transmissions. Much previous work has concentrated on the creation of software-based protocols to enforce this behaviour, as native CAN does not directly support time-triggered communications. In order to implement message time-outs within such a frame work, then one possible solution would be as follows. In time-triggered systems (under fault-free conditions), when a message is scheduled for transmission by the host CPU the bus should already be in the idle state. To enforce a time-out on message i, the host can simply set a (high-priority) timer interrupt to occur m_i time units into the future just prior to setting the transmit request (TXRQ) flag in the CAN controller. When this interrupt occurs, the host immediately resets the TXRQ

flag in the controller. The main advantage of this solution is that it is relatively simple. However it has several drawbacks; any jitter and latency affecting the servicing of the ISR coupled with clock drift between the host timer and the CAN bus oscillator - may skew the actual (real) time the transmission is cancelled. It also requires the use of a timer with at least as good a precision, as a CAN network bit time may also place a relatively large load on the CPU if there are numerous messages to transmit.

8.5.2 Hardware-Based Solution

In the context of the current work, a powerful modification to CAN in hardware is proposed and implemented. In addition to allowing, each CAN object in the controller to be operated in "standard" or single-shot mode, a third mode of operation - windowed mode - was introduced. In this mode, a 32-bit hardware counter C (when active, increments by 1 with every bit time on the bus) and two additional 32-bit match registers (CLB and CUB) is attached to each CAN object. In addition, the host CPU sees an effective TXRQ bit, but this is in fact a dummy, used only for interface purposes; a hidden register TXRQ#, controls the actual transmission logic. When a host initiates a message transmission (setting TXRQ), the counter C is resetting to 0; setting of the TXRQ# bit of the CAN object is delayed until C = CLB. Also, when C = CUB, both the TXRQ and TXRQ# bits are automatically reset in hardware. Since C increments at the same rate as the bit-time, this provides for a programmable allowed transmission window for a given CAN object which does not require the need for further CPU intervention. Additionally, the impact of jitter and latency on the host CPU is minimized, as the timer is referenced to the local oscillator. The only potential drawback of this solution is that it requires a very small increase in hardware complexity. However, since this modification not only allows

Figure 8.4: Successful windowed transmission with a fault

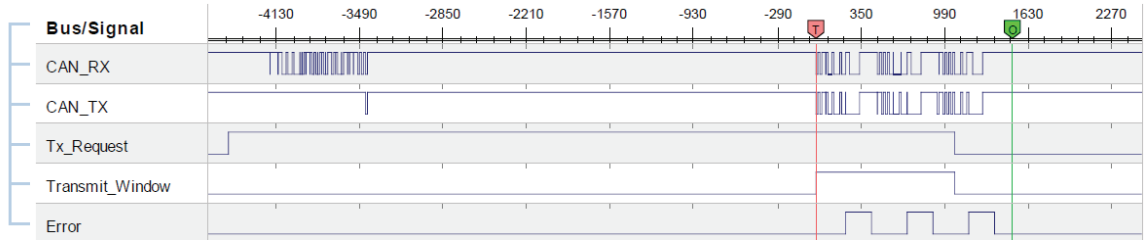


Figure 8.5: Unsuccessful windowed transmission with faults

the effective implementation of the alternate (similar) protocols such as the Timely CAN [BB01] and shared-clock [APSP07] protocols, this very simple hardware change could easily be incorporated into future generations of CAN controllers.

Here, the working of this modification is described with the help of real-time snapshot from the Chipscope as shown in Figures 8.4 and 8.5.

Figure 8.4 shows a scenario where a transmission is successful after an error occurred during the first transmission attempt. As the error occurred inside the transmission window, the concerned node was allowed to retransmit the unsuccessfully transmitted message. Another important signal in the snapshot is the Tx_Request which indicates the difference between the standard CAN protocol and the windowed transmission. In standard CAN a node would start to transmit as soon as transmit request is received and the CAN bus is free. In the windowed transmission the node will wait until the start of the transmission window although the transmit request

has been received and the CAN bus is also free for transmission.

Figure 8.5 shows a scenario of burst errors. In this snapshot three errors occurred; two inside the transmission window and the third one outside the transmission window. As soon as the third error occurred, the message was dropped, demonstrating successfully the working of the window transmission scheme, which is quite a good solution for TDMA or time slot based transmission.

8.6 Simulation Study

In order to begin to investigate the proposed technique, a small simulation study was carried out. This study was carried out to assess the potential bandwidth savings that may be achieved by employing the windowed technique as the message parameters are varied. Three experiments were carried out. In each experiment, 100,000 message streams were generated with parameters randomly selected (using a uniform distribution) from the following intervals: $p_i \in [1, 1000]ms$, $DLC_i[0, 8]$ bytes, $\lambda_i[10^{-9}, 10^{-5}]$ failures/hour. Standard and extended identifiers were randomly employed; a 31-bit worst-case error frame was also added to the length of each message. The generated target failure rates cover all four SIL's as specified in IEC 61508, and individual message failure rates were derived from the target λs based on their periods. Three different classes of BER were employed; Benign/Normal with $\beta \in [10^{-9}, 10^{-7}]$, Normal/Aggressive with $\beta \in [10^{-7}, 10^{-5}]$ and Aggressive/Hostile with $\beta \in [10^{-5}, 10^{-3}]$. In each case the percentage reduction in the number of bits requiring transmission was calculated when employing windowed transmission and message duplicate transmission. The average and maximum recorded values for each environment are as shown in Table 8.2. Also shown is the average and maximum recorded window sizes

	Percent reduction		Window Size	
Environment	Average	Maximum	Average	Maximum
Benign/Normal	2.7	33.3	280	571
Normal/Aggressive	12.0	33.2	410	797
Aggressive/Hostile	29.1	39.1	810	1883

Table 8.2: Reductions in bits required for transmission

m, which also gives an indication of the quick convergence of the slot sizing algorithm, even for low failure rates and high BERs.

From this table, it can be seen that the average effectiveness of the proposed technique depends upon the target environment; the worse the expected BER, the higher the average reduction in bandwidth required. For hostile environments, an average 29% reduction can be achieved. Given the scarcity of available bandwidth with CAN, this is a potentially large saving. In normal and benign environments, the average reductions drop to 12% and 2.7% respectively; however, the best case reductions remain at around 3%. In each case, the worst-case reductions were 0%, i.e. the technique performed is no worse than sending full duplicates.

8.7 Experimental Study

8.7.1 Experimental Configuration

The test bench employed 4 nodes communicating over a CAN bus, using the modified FPGA CAN controllers as described above. Figure 8.6 shows a schematic of the test bench configuration, the CAN network configured to run at 1 Mbps. Each node in the system sends a message, with a fixed schedule given in Table 8.3. In each case, the data size is 8 bytes, corresponding to a worst-case message length of 165 bits. Medium access control was implemented using a static TDMA schedule, repeating

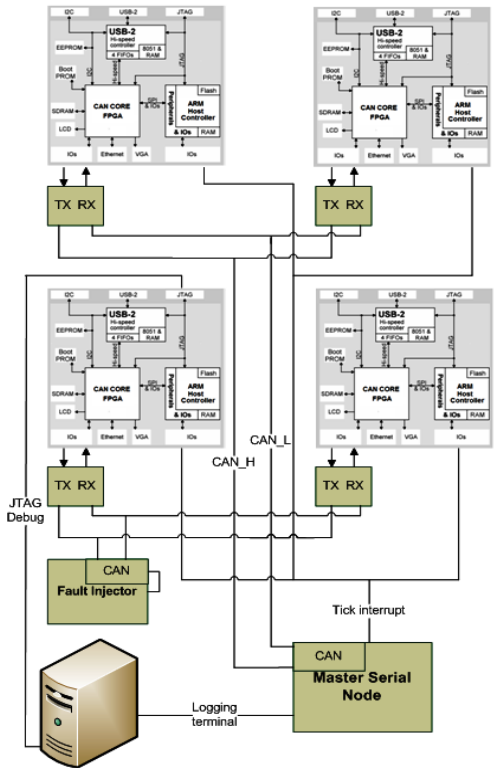


Figure 8.6: Test bench employed in the experiments.

every 8 ms. A bandwidth allocation of 300 bits per slot was allotted. In order to prevent any clock drift issues potentially impacting on the results, the node clocks were explicitly synchronised using a separate strobe line from a designated master node to the 4 slaves. A tick is generated every ms using a timer overflow interrupt.

The data content of each message sent by each CAN node was generated randomly, but the identifiers were kept constant for identification of the CAN node. As shown in the Figure 8.6, an additional node was used to inject faults onto the CAN

Message	Period (msec)	CLB(μsec)	CUB1/CUB2(μsec)
A	1	0	1/165
B	2	500	501/665
C	4	1500	1501/1665
D	8	3500	3501/3665

Table 8.3: Static schedule running on the CAN nodes

bus. A pseudo-random fault generation algorithm was employed on the fault injector node and was configured to disturb the bus and inject bit errors with a uniform BER of approximately 0.5×10^{-4}, corresponding to an aggressive environment. Two experiments were conducted, each for a duration of 24 hours. In the first experiment, standard single-shot transmission was used in each TDMA slot. In the second, the window transmission technique was employed. The corresponding CUB and CLB values to achieve the desired behaviour for experiments 1 and 2 are shown in Table 8.3. For both experiments, in error-free conditions 1875 messages should be sent every second. During the course of the two experiments, the master node was employed to receive all messages successfully sent by each CAN node, and this count was logged in memory. The logged data was then transmitted at regular intervals through a serial interface to a data logging PC. The results obtained are described in the next section.

8.7.2 Results

Experiment 1

Message transmissions were only single-shot during the TDMA slot. Only a single message copy could be sent in the allotted slot size of 300 bits. The statistics found were as follows:

The average number of messages successfully received every second over the test

run was 1871.58, therefore an average drop of 1875-1871.58 = 3.42 messages per second. The probability of successful delivery is therefore, approximately 0.9925. The standard deviation from the mean was 2.31 messages. The maximum and minimum values recorded over the duration of the test run were 1875 and 1871 respectively.

Experiment 2

Message transmission attempts are allowed up to 165 bits following the start of the TDMA slot. The statistics found were as follows: The average number of messages successfully received every second over the test run was 1873.80, an average drop of 1875-1873.80 = 1.2 messages per second. The probability of delivery is therefore, approximately 0.99936. The standard deviation from the mean was 0.61 messages. The maximum and minimum values recorded over the duration of the test run were 1875 and 1873 respectively.

8.8 Discussion and Conclusion

As can be seen from the results, the window technique significantly improved the reliability of message delivery over the course of the experiments. For a bit error rate of 0.5×10^{-4} and 165 bit messages, the expected probability of message delivery is 0.99936, which is close to the value obtained in simulation. Using the techniques described in section 4, for m = 300 the expected probability of delivery is 0.992, again close to the value obtained. Although the fault injection conditions were somewhat artificial and ideally realised in the experiments, the results obtained provide some justification for the proposed techniques. However, further experimental work (employing more advanced error models, and including comparisons against duplicated message streams) should provide a better picture of the expected behaviour of the

technique in realistic situations.

This chapter has considered the potential timeliness benefits and reliability drawbacks that may occur when implementing TDMA based networked sensing systems. The notion of an l-firm real-time constraint intends to provide a multi-criteria real-time/reliability guarantee that is well suited to the probabilistic nature of real-time sensor networks. An algorithm to compute the smallest TDMA slot sizes so that messages can meet a $\lambda - firm$ irm real-time constraint has been described. In comparison to sending full message duplicates, simulation and empirical studies both indicate that the technique can - in some cases - significantly reduce the required bandwidth whilst maintaining both reliability and time line. Equally, for a given bandwidth allocation, the technique can significantly improve reliability whilst maintaining time lines.

Chapter 9
Case Study

The basis of this chapter is a comprehensive case study covering all the modifications discussed in the previous chapters. A number of experimental case studies will be presented, based on the Windowed transmission scheme. Two of the related experimental case studies on "windowed transmission" and "single-shot transmission" has already been presented in the last chapter. This chapter will present a further combination of modifications on the standard CAN protocol presented in chapter 6 and 7. The experiments run on the same static schedule given in Table 8.3.

Based on their results, these detailed case studies are evaluated to find out whether the modifications have increased reliability, predictability and the rate of transmission to the standard Controller Area Network.

9.1 Modifications to the Test Bench

To conduct these studies basic setup of the test bench is slightly different as described in the last chapter. The experiments with the overclocked and fixed length messages have a modified CAN frame structure, therefore, normal off the shelf CAN controllers will not be able to identify these messages successfully. The modified controllers, if connected to these off the shelf CAN controllers, will first flood the network with error frames and will then goes in to bus off state after unsuccessful receipt of messages. These experiments ran for an average length of 24 hours, with each scheme running

Maximum(μsec)	Minimum (μsec)	Average(μsec)
73	1	23.2

Table 9.1: Statistics of the Fault duration

with and without fault injection. The fault injection is of random duration and interval, the mean of distribution of the faults is 2 per second. Another aspect which has to, be considered before analysing the results based on different schemes is that the test bench designed here is not at the industry level, so any random errors can occur during transmission due to impedance mismatch and reflections on the CAN bus, although standard wiring and termination was used. Table 9.1 are showing statistics of fault injections.

A total of 12 experiments with 6 different configurations on this facility were conducted. These experiments examine different aspects of the controller, with and without fault injection. Two of these configurations windowed, and single-shot experiments have already been discussed in the last chapter. Each experiment ran for a period of 24 hours. The statistics for the fault injection is shown in the Table 9.1. Different experimental configurations are presented in this section.

9.1.1 Running a Static Schedule on Standard CAN Network

Experiments were conducted, involving transmission of the standard CAN message following a static schedule as shown in Table 8.3. This schedule was partially followed as there was no upper or lower transmission bounds. Transmissions followed the standard CAN protocol, with guaranteed delivery and with infinitesimal number of retransmissions. This schedule ran successfully on a standard CAN.

The static schedule based on a TDMA cycle, and the messages should not encounter any contention. Therefore, the messages should be transmitted without a delay or failure. With the injection of faults, specially with burst errors, the corrupted messages have to wait before being retransmitted. This will cause pending messages to be delayed further as some of them have to face arbitration on the bus. Hence this message schedule is delayed further without bounds, therefore, it can be said that a fixed message schedule is unlikely to be followed when running a standard CAN within a noisy environment.

9.1.2 Windowed Transmission with Overclocking

Overclocking is a scheme where certain fields of a message are sent at a higher rate than the other fields. This scheme is successfully implemented for Controller Area Network with transmission speeds of 10 Mbps. In this experiment, the S-zone of the CAN messages were overclocked from 1 to 2 Mbps. The reason not to include higher rates of overclocking is due to the high number of errors which are encountered due to transceiver internal delay.

9.1.3 Windowed Transmission with Fixed Length Messages

Fixed length messages were proposed to increase the predictability of message transmission times and reduce the jitter in transmission times, as the message lengths are fixed. The results should not be too different from the CAN standard protocol running the window scheme, as the only difference is that the susceptibility of error is increased due to longer duration of messages.

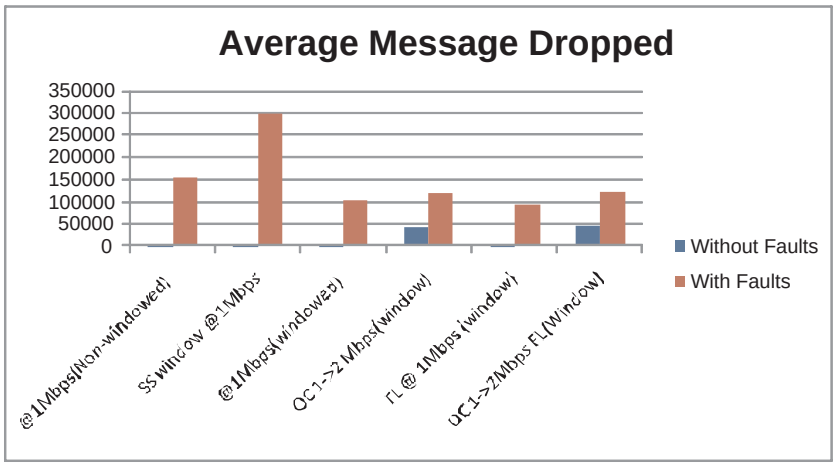


Figure 9.1: Average message drops per second

9.1.4 Windowed Transmission with Fixed Length and Over-clocked Messages

This case study comprises of fixed length, over clocked messages, therefore the improvement in information throughput reduces the effect of extra bits in a fixed length message. The TDMA slot has been set to be 165 bit times.

9.2 Discussion and Analysis

In this section, results and analysis of the 12 different experiments are presented, based on 6 different configurations with and without fault injection on the modified CAN IP controller.

Figure 9.1 shows the number of dropped messages for the complete run of 24 hours on an average basis in each experiment. For reference, the ideal number of successfully transmitted messages should be 162×10^6. Figure 9.2 shows the standard deviation on the average messages dropped per second thus showing the variation span of the

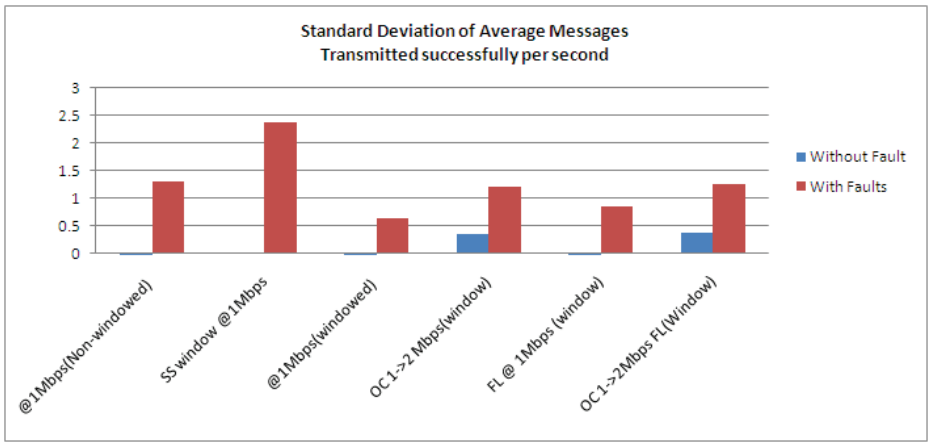


Figure 9.2: Standard deviation of message drops from the mean

number of messages dropped. Table 9.2 gives the statistics of these experiments. The transmission rate is 1 Mbps for all other schemes when overclocking the rate varies from 1 to 2 Mbps.

First the results for experiments without external faults are considered. The results should and are pretty straight forward. With standard/modified CAN protocol following the TDMA schedule, there should be no message drops considering that this experiments were conducted in a benign environment [FOFF04]. The messages should not experience any contention for the bus access and therefore the messages should be transmitted without any delay. There is an exception in the results, when messages are overclocked. Even with no external errors injected, 0.5 to 0.54 messages per second are dropped. However with external errors the CAN transmission at an over clocking from 1 to 2 Mbps, and BER of 2.4×10^{-6}, the failure rate gives a value of 0.68 errors/second. The reasons have been analysed in chapter 6, and have suggested a remedy to increase the performance of overclocked networks in higher BER's. The transmission behaviour tends to vary in case of faults induced with different experimental configurations. In the first case, of non-windowed transmission,

Mode	Without Fault				With Faults			
	Maximum	Minimum	Average	Std Dev	Maximum	Minimum	Average	Std Dev
CAN Standard	1875	1874	≃1875	0.001	1875	1872	1873.21	1.30
Single Shot Window	1875	1874	≃1875	0.031	1875	1861	1871.58	2.39
Windowed CAN	1875	1874	≃1875	0.0001	1875	1873	1873.80	0.61
Overclocking	1875	1873	1874.5	0.35	1875	1870	1873.66	1.21
Fixed Length	1875	1874	≃ 1875	0.011	1875	1871	1873.92	0.48
Fixed & Overclocked	1875	1873	1874.46	0.39	1875	1869	1873.62	1.27

Table 9.2: Statistics for message transmission

without any faults, there are no message delays due to non-contention on bus access. With induced faults, messages which meet errors, are re-transmitted, as there is no upper bound on re-transmission attempts and the next scheduled messages are further delayed, which can create a domino effect. These messages will now be queued till they are successfully transmitted, so nodes with pending messages have to arbitrate for the CAN bus, before transmitting these messages. Single Shot transmission has encountered the highest number of message drops per second, this is due to a single transmission attempt being allowed in the scheduled cycle. A single-bit error can delay the transmission and the complete message has to be re-transmitted. As can be seen in Figure 9.1, that when compared with the SS mode of transmission almost every windowed scheme showed a 200% increase in transmission reliability. Single shot transmission is used in few of the time-triggered protocols such as [FH00] and does not provide any mechanism to improvise in case of failures or errors. The messages are dropped immediately and remain pending till the next TDMA slot.

Looking at the different configurations employing windowed transmission, improved performances in comparison to the non-windowed schemes can be observed. The improvements are significant except in case of overclocked messaging, where there is a slight degradation of performance, this is due increased BER at high speeds and

needs to be compensated by utilizing higher bus voltages. Fixed-length messages have not made a significant impact on the BER and message delivery success rate, although the standard deviation of message drop is the lowest in this case. This presents more accurate and predictable performance with regards to error analysis and is easier to schedule messages for time critical transmissions. Thus, it can be summarized, that the modified CAN controller work demonstrated in previous chapters, has helped to increase the predictability, and meet the time lines of the given schedules. These modifications have also shown to be effective by reducing the burden on the application layer and to neutralize the effects of re-transmissions and bit stuffing introduced by the standard CAN protocol.

Chapter 10
Discussion & Conclusion

10.1 Reasons and Motivation of the Thesis work

The work described in this thesis relates to the enhancement of the Controller Area Network protocol to improve the support for predictable, fast and guaranteed message transmissions.

This research work is based on an engineering design and practical implementation of the CAN IP core. The key contributions and the extent to which the goals, stated below, are achieved are discussed here.

Goals of the Research

The goals set for the research was:

1. Improve standard CAN to support time-triggered applications, with an increase in transmission reliability and speeds.

2. Propose enhancements to CAN at the physical and data link layer, taking a different research direction then conventional approach of adding enhancements at the application layers.

3. Implementation of CAN [11893] data link and part of the physical layer on a flexible plate form, this should be a practical solution.

4. The CAN standard implementation should conform to the specifications of the protocol.

5. To envisage and demonstrate the limits to which changes can be done at the protocol level and the strength of these changes.

6. To propose, implement and run extensive experiments on enhancements done to the CAN standard protocol.

7. To demonstrate with the help of results obtained from the experiments that the goal set in point 1, CAN is now closer to support time-triggered applications, higher speeds and increased transmission predictability, as it was before the start of this research.

10.2 Review of the Contributions

10.2.1 Findings of the Literature Review

The findings from the literature review indicate that although software based enhancements to the CAN protocol can be somewhat effective, they do not address specific protocol level problems such as bit stuffing, speed limitations and arbitrary re-transmission attempts in their entirety.

As the nature of the protocol places physical limitations on the CAN data rate, software changes alone cannot speed up the transmission rate but changes are also required at the physical layer if this is to be achieved. The other drawbacks of the CAN protocol such as bit stuffing and TDMA based communication are primarily concerned with features not likely to be rectified at the application or software layer,

or where they are implemented at this level, this could lead to clumsy solutions with unwanted CPU overheads.

Considering these points, work was started on the improvement of the CAN protocol at the hardware level. It was decided that in order to achieve the flexibility required for changes at this level, the protocol should be implemented at the RTL level, using soft-core technology. This should result in the maximum possible freedom to implement modifications and also conduct verification experiments to allow comparisons to the previous work done in this field.

10.2.2 CAN Protocol in Silicon IP Core

Since this work emphasises the changes at the protocol level, the implementation of the CAN IP core forms the basis of the work done in this research. A complete design flow for the implementation of CAN IP core is discussed in Chapter 4. The basic functional blocks of the CAN protocol with details of implementation in Verilog HDL is given, followed by a presentation of complete state machine implementation of the transmit and receive cycles of a CAN message. This follows with the synthesis and mapping of the design on to the physical hardware i.e. FPGA. The implementation is shown to be feasible for the chosen hardware, in terms of timing closure and resource utilisation.

10.2.3 CAN Conformance Testing

Conformance testing of the CAN protocol controller is a complex and costly process. The ISO has drafted a CAN conformance testing document [ISOa], which enumerates a total of 130 different tests which are required to be conducted to claim CAN

conformance.

A relatively straightforward and cost-effective method of CAN conformance testing for soft-core implementation of the protocol is introduced in this research. The method makes use of integrated logic analysers, along with additional HDL test cores to perform the conformance tests. The methodology introduced for conformance testing was purposely designed for IP core testing in a research environment, using standard lab equipment and software. Integrated logic analysers are an acceptable compromise between simulation and hardware bench logic analysers. Also, the use of virtual I/Os and soft-core test pattern generators are a valuable replacement to the costly and often complex pattern generators needed for conformance testing.

In chapter 5, one test case each from the 7 test classes defined by the ISO document [ISOa] is presented. In this work more than 65% of the total listed test had been conducted, these tests had been carefully selected to justify the conformance of the IP core. The number of tests conducted, covered all the classes and types of conformance tests. The number of tests conducted were thought of as enough with the reasons that functionality of each CAN block was tested and also the IP core worked fine with standard CAN controllers, without any noticeable anomalies.

10.2.4 Overclocking: A solution to Increase Transmission Speeds

As defined one of the major goals of this research was to improve the transmission speed of the CAN. Although like other early day embedded protocols, CAN supports communication for small payload and slow communication speeds. For high speed embedded applications, CAN is not a choice, and the designers prefer contemporary

protocols such as FlexRay [Fle04], to support high speed applications.

In Chapter 6 a scheme of overclocking was presented, which aims to partially lift the bps restrictions due to propagation delays on the CAN bus. The overclocking makes use of dual-rate transmission for the data and CRC fields of the CAN frame, which are sent at a higher rate than rest of the fields. Although this scheme has previously been discussed in the literature, to date it has not been experimentally validated as it requires protocol-level modifications. Chapter 6 discusses the first implementation and testing of the overclocking scheme, achieved with modifications to the CAN IP core. It was demonstrated that with this implementation of overclocking, a CAN frame can easily be transmitted at a dual rate of 1/10 Mbps.

In addition to demonstrating the validity of the scheme, experimentation at higher data rates also revealed the fact that overclocking has a beneficial effect on transmission jitter. It was identified that the impact of bit-stuffing is minimized due to the shorter time taken to transmit the data and CRC fields, as these fields are more susceptible to bit-stuffing.

Although initial experiments with the technique indicate its effectiveness, it does have several drawbacks, for example the potentially drastic effect on BER. When overclocking is employed in a mixed system with one or more standard CAN controllers, numerous errors (most notably bit stuffing and CRC errors) are observed and signalled. It was also noted, through observation and examined analytically that the overclocked fields are prone to higher BER and failure rates due to rapid changes in voltage levels on the bus. A further work was also proposed as a possible solution to higher BER by increasing bus voltages. In conclusion, with appropriate care in the implementation stages, this modifications to the CAN protocol seems to indeed

be feasible, and can be used to increase the transmission rates to 10 Mbps.

10.2.5 Jitterless Communication

Although with the use of overclocking scheme it was demonstrated that CAN attain higher speeds than existing, but still the transmission jitter is a major bottleneck when designing TDMA based transmission scheme. Chapter 7 discusses a modification that would allow fixed length messages to be transmitted over the network, regardless of the message data contents. As such, this presents a novel idea to help overcome transmission jitter caused by the bit-stuffing.

The technique which can be enabled or disabled dynamically as required by the host CPU, forces every message to be a worst case bit-stuffed message, by adding an additional dummy field to the CAN frame. In situations in which the worst-case stuffing does not occur during transmission of the regular message fields, the difference is made up by stuffing extra bits into this additional dummy field, hence, the messages are always of a fixed length. This can be considered as a potential loss of bandwidth, adding up to an additional 24 μsec delay per message for a bit rate of 1 Mbps.

However, removing any ambiguity on the transmission times - with no added processing needed to be done at the software layer has obvious advantages in some situations. Hence there is a trade-off of predictability vs bandwidth for this mechanism, as often occurs in real-time systems which require analysis to be carried out under worst-case conditions. In case of the modified controller a fixed length message when combined with overclocking the message transmission times are further reduced; for example an 8 byte message can have maximum of 24 extra bits transmit, but with

overclocked at 5 Mbps will only take an extra 4.8 μsec.

10.2.6 Optimal Size Windows for Time-Triggered Communication

With the previous work where the CAN has been enhanced to work at 10 Mbps and transmission times more predictable with the addition of jitterless scheme; there was still some work left to support time-triggered communication. Specially in statically-scheduled system, all message transmissions are created off-line and stored in some form of table (a 'message map') that each node makes use of to control the instants in time that messages are allowed to be transmitted. With CAN's arbitration scheme it is impossible to predict which node will be able to access the bus at a given instant of time. Also in high EMI conditions, bus errors may lead to continuous retransmissions which can cause missing of several deadlines. This problem is addressed using a 'windowed transmission' scheme as discussed in Chapter 8.

The 'windowed transmission' provides time-triggered communications of comparatively increased reliability for a given bandwidth allocation. Windowed transmission uses fix time windows allocated to each node; the size of these time windows has been optimized using an efficient algorithm. The implementation has shown that the number of missed deadlines is reduced. The other advantage of this scheme is that it is very simple to implement and even can be added as a separate module with some modifications to a CAN IP. This final modification is fully compatible to the standard CAN protocol.

In order to employ windowed transmission successfully, each node must have an accurate knowledge of the network global time. This can be achieved by using a

simple global time scheme which synchronizes the nodes on the CAN network. One such scheme is the shared clock protocol [Pon01, APSP07], which is a master slave model, the master sends periodic messages to synchronize the running of the slave tasks with the network time.

10.2.7 A Comprehensive Case Study

After the completion of all the modifications (aimed for this research) to the CAN IP core, a comprehensive case study is presented in chapter 9. Although in the relevant chapters impact of each modifications to the standard CAN protocol was analysed individually, this case study was conducted to evaluate experimentally the effect of combinations of these modifications.

Results are presented from experiments conducted with and without induced faults. The results provide evidence that each of the proposed modification is interoperable, and it is possible to achieve increased throughput, predictable communications of increased reliability using the modified controller. Previous studies have shown that [FOFF04], in aggressive environments the BER on the CAN bus is drastically increased, with correspondingly high message failure rates observed. The case study has strengthened the argument that with the help of windowed transmission, timeliness guarantees are increased in comparison to single-shot or infinite retransmission schemes. Finally, in chapter 6, an observation was made about potentially increased BER when a CAN bus is overclocked. As has been discussed, the results in the case study provide some partial results to support this prediction.

10.3 Practical Adoption of the Modified Controller: A Realistic Proposal?

Since this research work is based on an engineering implementation and presents several modifications to a widely used embedded protocol, it is necessary to analyse the possibility of the CAN IP core and modifications to be adopted for real world designs.

The CAN IP core which is implemented in this project and have passed through the CAN conformance, can easily be adopted as a real-world design. In fact this work has been adopted for a small-scale research and industrial setup with suitable modifications [TTE]. An IP core implementation is initially more time-consuming than a heavily software-based solution, and less suitable for changes implemented at the application layer. In this case the implementation time had already been spent during the research work. The IP core can easily be converted in to a black box implementation using the EDA (Electronic Design Automation) tools; this black box implementation provide a standard interface to become part of of a SoC (System on chip).

Looking at the modifications done to the CAN protocol, experimental results have shown that these modifications resulted in improvements in speed, predictability and reliability. But are these modifications practical and whether they can be adapted for real-design implementations? This is a big question. In the opinion of the author of this work, these modifications at the current state focus more on the research side of the world i.e. at the moment the modifications make this protocol CAN-like but not CAN. This can put off some with the argument that this is another protocol in the

market, and what is the point of deploying a new protocol when there exists protocols which can support time-triggered and faster communication?

However, this does not completely put the practical usability of these modifications out of question. The lower physical layer (CAN Bus and Transceiver's) used with these modifications is still the same as the standard CAN network. This makes this work stand out as a good starting point for industries to utilise the existing infrastructure to build a better and faster embedded communication network. With further discussions given in the future work, it has been shown that with suitable changes these modified CAN controllers can work with the existing CAN nodes. As an industrial designer (with CAN experience) to deploy a completely new network protocol with altogether different infrastructure is a more subtle and expensive solution than using a CAN-like protocol which can be made backward compatible.

Another interesting argument given is: what if every designer starts modifying the protocol, and what about the standardisation of these modifications? A lot of major communication protocols have been prone to changes over time; industrial and educational research has enhanced their working and suitability for the changing needs. The work in this thesis is in the same direction and this is not the first effort done on the CAN to be modified for different practical environments. These changes have not only been adapted by the industry but have been standardised (TTCAN, CAN open e.t.c.). The question about standardisation of the modifications done in this work is only possible once a formal specification of this work is done with the extension of conformance tests to verify the behaviour of the modifications.

10.4 Critical Analysis of the Modifications

In this section, some of the difficulties and disadvantages which can possibly be caused by these modifications will be examined.

As (Chapter 6) demonstrated with analysis and experimental results in the case studies, overclocking technique allows for much higher information throughput, a major drawback seems to be in the simultaneous increase of BER. This phenomenon needs to be examined further. Since an increase in bus differential voltages will not only restore the SNR but will directly increase the power consumption, the use of overclocked controllers may be limited in power constrained environments. This may be improved somewhat by employing DVS [BB95], with voltages dynamically increased only as and when needed according to the overclocking factor.

For the jitterless scheme transmission times for messages with zero and maximum stuffing is the same, this can be argued as a waste of precious network time. In actual jitterless communication there is a compromise between extra information and the increased predictability of the transmission times. A better analogy to understand this compromise is the use of CRC field in the CAN frame, although without the CRC field 16 bits of added information can be removed to save precious network time, but this saving will be at the expense of error-detection capability.

Another major question which arises is related to the backward compatibility of the modified CAN controller. As mentioned previously, the use of windowed transmissions is clearly backwardly compatible with existing networks. Given the clear lack of backwards compatibility for overclocking and jitterless communications, each node in a network will require modified controllers as all existing CAN nodes are seemingly incompatible. However, because these features can be dynamically enabled or

disabled, the option to employ these features lies entirely with the network designer. As such, the options would seem to be of most practical use for new network designs. The possibility to solve this will be explored in future work.

10.5 Modified CAN vs other embedded Protocols

Table 10.1 and 10.2 shows a comparison between the modified CAN with major embedded protocols currently existent in the market. The modified CAN supports synchronous transmission at a higher data rate of 10 Mbps up and increased payload of 15 bytes. This improvement to the basic CAN is a considerable achievement of this work.

The modified CAN can now be seen as comparable to the other fast and synchronous major embedded protocols; on the other hand the standard CAN lacks support for time-triggered and high data rate transmission. The notable point in the modified CAN, is that apart from its improvement, it has not lost the stand-out features of standard CAN. Modified CAN still supports error confinement, priority based bus access and acknowledgement based delivery guarantees.

Features	CAN	TTP	FlexRay
Message transmission	async	synch	sync and asyncs
Message identification	message identifier	time slot	message identifier
Data rate	1 Mbps	2 Mbps	10 Mbps
Bit encoding	NRZ with bit stuffing	MFM	NRZ with start/stop bits
Latency Jitter	bus load dependent	constant	constant for high priority messages according t_cyc
Extensibility	excellent in non-time critical applications	if planned in original design	possible for high priority messages
Flexibility	flexible bandwidth for each node	one message/node and TDMA cycle	flexible bandwidth for each node
Data Length	8 bytes	Variable	256 bytes

Table 10.1: Comparison of modified CAN with other protocols-I

Features	Byte flight	TTCAN	Modified CAN
Message transmission	sync and async	sync	sync(for Windowed transmission) and async
Message identification	time slot	time slot	message identifier
Data rate	10 Mbps	1 Mbps	10 Mbps
Bit encoding	NRZ with start/stop bits	NRZ with bit stuffing	NRZ with bit stuffing
Latency Jitter	constant for all messages	bus load dependent	Constant and Minimum in Fixed Length messages
Extensibility	separation of functional and structural domain	only if extension is planned	Extensible in time critical applications in Windowed transmission
Flexibility	multiple slots per node, dynamic	Single shot message allocation	Flexible with limited retransmissions in windowed slot
Data Length	8 bytes	8 bytes	15 bytes extendible to 1024

Table 10.2: Comparison of modified CAN with other protocols-II

10.6 Future Research

The design of the modified CAN controller with features such as overclocking and fixed length messages opens up some new research questions. The major points left unanswered by this thesis and areas of future work can be summarized as follows:

Achieving Error Free Overclocked CAN Networks

Overclocking techniques found to be very useful to attain higher speed, but an observation is on the higher BER's, an area of future research. There is two prong strategy to address this problem.

1. Design of a new transceiver to support the use of fast driver transceivers [NXP02] can help to achieve higher data rates with less susceptibility to errors. As discussed in chapter 6, the major limiting factor to reduce the CAN bit time is the propagation segment which compensates for the physical line delays. The physical line delays also include the driver and receiver delay of the transceivers. The typical value of this delay is around 150 ns [Phi96a].

   Transceivers with lower driver delays of around 20 to 25 ns can help reduce the physical line delays to a value of 100 ns. This will then support communication

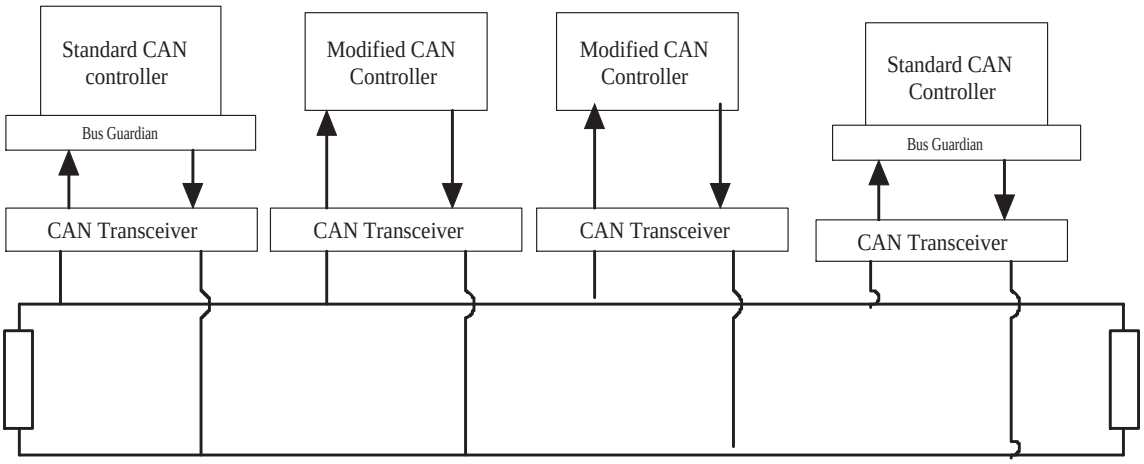
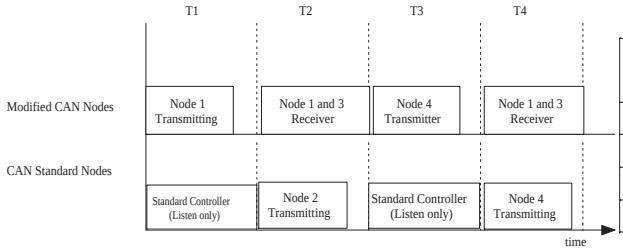


Figure 10.1: A mixed network with standard and modified CAN Controllers.

rate of 5 M bps over a length of 20 meters without any considerable change in the BER rates. Differential line bus transceivers [Tex00] used for RS-485 communication can support a data rate of upto 30 Mbps and NRZ-L encoding. The design can be adapted for overclocked CAN bus communication.

2. . The second strategy to reduce BERs on overclocked networks is to increase the differential voltages on the CAN bus. A higher differential voltage will increase the noise margin on the bus and susceptibility of bit corruption is reduced. The increase in differential voltages will result in higher power consumption - a dynamic voltage scaling scheme solves this problem. Voltage can be dynamically increased to higher voltages for higher data rates and scaled down on lower voltages, an example of such an implementation is done for radio communication transceivers [GDZG07].



Time Slot	Standard CAN (Node 2 and 4)	Modified CAN (Node 1 and 3)
T 1	Listen Only	Node 1 Transmitting
T 2	Node 2 Transmitting	Normal Receiver
T 3	Listen Only	Node 2 Transmitting
T 4	Node 4 Transmitting	Normal Receiver

Figure 10.2: TDMA schedule for a mixed network, using "listen only mode".

Backward Compatibility

The modifications presented in this work such as overclocking and fixed-length messages, change the standard CAN frame format. These modifications are not backward compatible to the CAN standard, an area of future research. There are few suggestions to implement a mixed network of modified CAN controllers and the standard CAN.

1. In passive mode, the CAN controller will send and receive 11-bit identifiers and silently drop 29-bit identifiers without generating an error. If every standard CAN controller in a network is operating in CAN 2.0B passive mode, then the modified controllers may operate in CAN 2.0B active mode. They can use the reserved bits to signal to each other if the frame is overclocked and/or jitterless. The CAN 2.0B passive nodes will then ignore the message regardless.

2. When a mixed network contains CAN 2.0A controllers, this will not work for the solution presented in last point. The CAN2.0A has certain reserved bits which are kept for future modifications. These bits can be use to form a mixed network of both standard and modified CAN controllers. Normally these reserved bits

r0 and r1 are recessive and can be used to distinguish between a normal and modified frame with the help of of bus guardians [BB03]. Figure 10.1 shows such a mix network scenario. In this, the standard CAN nodes are attached to bus guardians which will stop any further communication in case of a dominant r0 or r1 bit. The bus guardian will disconnect the CAN controller from the bus, for rest of this frame transmission. This bus guardian is quite analysable and can easily be implemented in a soft-core.

3. In a TDMA-based network, additional information can be stored in the message table related to whether the frame is overclocked or jitter-free. Normal CAN controllers could be switched to the listen only mode by the host CPU, for the duration of an overclocked or jitterless message timeslot. This will stop the standard CAN nodes to generate any error frames on the network, and communication between modified controllers can continue without interruption. An Example message table of a mix network is given in Figure 10.2

Automating CAN Conformance Testing

The CAN conformance testing procedure, presented in this research work has helped to perform all the representative and edge test cases given in the ISO standard [ISOa]. Since the method used in this work is quite manual, it was not possible to run all the documented tests. Further work could be done to perform all these tests by automating the procedure to perform these tests.

Automation of these tests is possible using the Bosch CAN VHDL reference model [Bos99], although this model is principally used for simulation, proper scripting and the use of additional soft-cores to generate and log tests is possible. It is also possible

to run fully automated tests with the enhancement in features of the integrated logic analysers to support capturing and logging of signals over larger time-scales.

10.7 Summary of Results

The modified CAN controller developed in this thesis has the following enhancements with respect to a standard CAN controller:

1. The modified protocol controller can be operated, if desired, as a fully-conformant, standard CAN controller;

2. One or more messages can be sent with a fixed (and hence predictable) transmission time regardless of the payload contents;

3. One or more messages can carry a payload of up to 1024 bytes (The current implementation can support 15 byte messages);

4. A full time-triggered schedule can be implemented and messages can be transmitted in fixed allotted time windows.

This concludes the work done in this thesis, with final remarks that the goal set to achieve better time-triggered operations on CAN are achieved, with some room to improve results in the future work. This work has successfully given a different dimension of using an IP core implementation to modify the protocol to a larger extent, then the traditional approaches. The reader of this work will surely benefit from the engineering aspect of this work, and this work will also provide a good insight into what standard CAN is and how modifications can be applied to enhance this widely popular protocol.

Appendix A

Formation of Test bench and Design Analysis

A.1 Interface Code

This section provides the simple interface functions used to read and write data between the host processor and the CAN IP core.

Interface header file

```
/*------------------------------------------------------------
  This header File contains the various declarations
  LPC interface to the FPGA CAN implementation
  File: CAN_INT.h
  Project: CAN Interface to FPGA
  Author: Sheikh Imran
  Started on: 21/06/2007
  Last updated:  13/12/2010
------------------------------------------------------------*/
 #ifndef _CAN_INT_H
 #define _CAN_INT_H
 #include <LPC21xx.H>


            void delay(unsigned long int);
    void CAN_Init();
void CAN_Self_Test();
void CAN_Test_Register();
void CAN_Basic_Frame_Tx(unsigned long int);
void CAN_Basic_Frame_Rx(unsigned long int);
void CAN_Extended_Frame_Tx(unsigned long int);
void CAN_Extended_Frame_Rx(unsigned long int);
void WriteData(unsigned char, char);
            char ReadData(unsigned char);
/****************** Signal definitions**************/

  #define     ALE 0x00000200 //P0.9
  #define      RD  0x00000400 //P0.10
  #define      WR  0x00001000 //P0.12
  #define      CS  0x00000100 //P0.8
          #define      RST 0x80000000 //P0.31

  #define      P0_data  0x000F00F0 // P0.3-P0.7(lower nibble) and P0.16-P0.19(higher nibble)
  #define      P1_control  0x80001F00 // Port 1 for control signals
```

```
  #define      LED 0x00FF0000
  #define      Rs  0x00400000 // slope control for transceiver
/*************** CAN configuration values************/

//Clock Divider register
  #define CDR 0x84 //PelicanCAN mode,clock off,comparator bypassed
  //Acceptance Code register
  #define ACR 0xFF //Allow all identifiers
  //Acceptance Mask register
  #define AM 0xFF //Allow all identifiers
  //Bus timing values for 48 MHz clock, 1Mb/s bit rate
  //Bus timing register 0
   #define BTR_0 0x00
  //Bus timing register 1
  #define BTR_1 0x21   // TSEG1=(2+1)x TQ, TSEG2= (1+1)x TQ

/*************** CAN Configuration Registers and Bits ********************/
/* Address and bit definitions for Mode Register */
#define ModeReg 0x00
#define RM_RR_Bit 0x01 /* reset mode (request) bit */
#define RIE_B_Bit 0x02 /* Receive Interrupt enable bit */
#define TIE_B_Bit 0x04 /* Transmit Interrupt enable bit */
#define EIE_B_Bit 0x08 /* Error Interrupt enable bit */
#define DOIE_B_Bit 0x10 /* Overrun Interrupt enable bit */

/* Address and bit definitions for command Register */
#define CommandReg 0x01
#define TR_Bit 0x01 /* transmission request bit */
#define AT_Bit 0x02 /* abort transmission bit */
#define RRB_Bit 0x04 /* release receive buffer bit */
#define CDO_Bit 0x08 /* clear data overrun bit */
#define SRR_Bit 0x10 /* self reception request bit (Extended mode)*/
#define OFR_Bit 0x20 /* Goto Sleep Mode

/* Address and bit definitions for Status Register */
#define StatusReg 0x02
#define RBS_Bit 0x01 /* receive buffer status bit */
#define DOS_Bit 0x02 /* data overrun status bit */
#define TBS_Bit 0x04 /* transmit buffer status bit */
#define TCS_Bit 0x08 /* transmission complete status bit */
#define RS_Bit 0x10 /* receive status bit */
#define TS_Bit 0x20 /* transmit status bit */
#define ES_Bit 0x40 /* error status bit */
#define MODE_Bit 0x80 /* bus status bit */

/* Address and bit definitions for Interrupt Register */
#define InterruptReg 0x03
#define RI_Bit 0x01 /* receive interrupt bit */
#define TI_Bit 0x02 /* transmit interrupt bit */
#define EI_Bit 0x04 /* error warning interrupt bit */
#define DOI_Bit 0x08 /* data overrun interrupt bit */
#define WUI_Bit 0x10 /* wake-up interrupt bit */

#define EPI_Bit 0x20 /* error passive interrupt bit */
#define ALI_Bit 0x40 /* arbitration lost interrupt bit */
#define BEI_Bit 0x80 /* bus error interrupt bit */

#define BusTimingReg_0 0x06;
#define BusTimingReg_1 0x07;
#endif
```

Write Data to a CAN Register

```
void WriteData(unsigned char Address, char Data)
{

1. IODIR0 |= P0_data; // set the direction of the pins as output
2. IOCLR0 |= P0_data; // clear the data pins
3. IOSET0 |= ((Address&0x0F)<<4)|((Address&0xF0)<<12);// write address to AD_0_7(AD) bus
4. IOSET0 |= CS;  // insert the chip select
5. IOSET0 |= ALE; // assert the Address Latch to write
6. IOCLR0 |=  ALE; //de-assert the Address Latch
7. IOCLR0 |=  P0_data; // Clear the AD bus
8. IOSET0 |= ((Data&0x0F)<<4)|((Data&0xF0)<<12); // Write the data on the AD bus
9. IOSET0 |= WR; //assert the write signal
------------------------------------------------------------
10. IOCLR0 |=  P1_control; // clear all the control signals (read, write, cs)
11. IOCLR0 |= P0_data; (clear the data pins)
}
```

Read Data from a CAN Register

```
char ReadData(unsigned char Read_Address)
{
1. unsigned int message1,message_lower,message_upper;
2. char message;
3. IODIR0 |= P0_data; // Set the AD bus as output
4. IOCLR0 |= P0_data; // Clear the AD bus
5. IOSET0 |= ((Read_Address&0x0F)<<4)|((Read_Address&0xF0)<<12); // Write the Address on the AD bus
6. IOSET0 |=CS; // Select the chip
7. IOSET0 |= ALE; // Latch the address to the CAN IP address register
8. IOCLR0 |=  ALE; // Free the AD bus
9. IOCLR0 |=  P0_data; // Clear the contents from the AD bus
10.    IODIR0 |=  0x00000000; // Set the direction of AD bus as input
11.    IOSET0 |= RD; // assert the read signal
12. message1 = IOPIN0; // Read the value on the PORT 0
13. IODIR0 |= P0_data;
14. IOCLR0 |= P0_data;
15. IOCLR0 |= P1_control;
16. message_lower=((message1& 0x000000F0)>>4); // shift and adjust the 32 bit read value
17. message_upper=((message1& 0x000F0000)>>12);
18. message = message_upper|message_lower; // Adjusted Read value
19. return message;
}
```

CAN Initialisation Routine

```
void CAN_Init()
{
   WriteData(ModeControlReg,0x01); // Setting up configuration mode

   WriteData(Clock_Div_Reg,0x80); // Setting the extended mode

   WriteData(ACR0,0x00);  // acceptance code 0
   WriteData(ACR1,0x00); // acceptance code 1
   WriteData(ACR2,0x00); // acceptance code 2
   WriteData(ACR3,0x00); // acceptance code 3
   WriteData(AMR0,0xFF); // acceptance mask 0, Don't care
   WriteData(AMR1,0xFF); // acceptance mask 1, Don't care
   WriteData(AMR2,0xFF); // acceptance mask 2, Don't care
```

```
   WriteData(AMR3,0xFF); // acceptance mask 3, Don't care

   WriteData(BTR_0,0x00); //Bus timing register 0, for 1 Mbps
   WriteData(BTR_1,0x21); //Bus timing register 1
   WriteData(Int_en,0x80);  // Transmit Interrupt enable
   WriteData(ModeControlReg,0x00); // Normal operating mode
}
```

Basic Frame Transmission

```
void CAN_Basic_Frame_Tx(unsigned long int Time_out);
{

/* wait until the Transmit Buffer is released */
/* Transmit Buffer is released, a message may be written into the buffer */
/* in this example a Standard Frame message shall be transmitted */
WriteData(0x10,0x08); /* SFF (data), DLC=4 */
WriteData(0x11,0x39); /* ID1 = A5, (1010 0101) */
WriteData(0x12,0x21); /* ID2 = 20, (0010 0000) */
WriteData(0x13,0x32); /* data1 = 51 */
WriteData(0x14,0x1F);
WriteData(0x15,0xF8);
WriteData(0x16,0x1F);
WriteData(0x17,0xE7);
WriteData(0x18,0xF8);
WriteData(0x19,0x7D);
WriteData(0x1A,0xE1);
WriteData(CommandReg,TR_Bit) ; /* Set Transmission Request bit */

do
{
status=ReadData(InterruptReg);
if(status & BEI_Bit ==0x80)
       {
       IOSET0 |=0x40000000|RST;
       delay(1000);
       IOCLR0 |=0x40000000|RST;
       }
status=ReadData(StatusReg);

}
while((status & TCS_Bit) != 0x08) | Time_out); /* Wait till transmit complete */
}
```

Basic Frame Reception

```
void CAN_Basic_Frame_Rx(unsigned long int Time_out)
{
char status;
char message[10];
int count=0;
IOCLR0 |=0xC0000000;

\* Wait for the receive interrupt */
do
{
status=ReadData(InterruptReg);

}
```

```
while((status & RIE_Bit) != 0x01)| Time_out);
if (status == 0x01)
{
message[0] = ReadData(0x10); // read data length
message[1] = ReadData(0x11); // read identifier byte 1
message[2] = ReadData(0x12); // read identifier byte 2

    data_len  = (message[0]&0xF); // mask the data length bits
for(i = 0;i<data_len;i++) // read the data message
{
message[i+3] = ReadData(k);
k++;
        }

WriteData(CommandReg,RRB_Bit); // release the receive buffer of CAN IP
}
}
```

A.2 Issues related to the Test Bench

Problem No 1

The function WriteData is part of the interface software written to communicate between the host processor and the CAN IP core registers. **Statement:** The lines 10 and 11 in the code were not included in the initial version of the function WriteData. It was assumed that once the write function is over the temporary Address/Data registers of the CAN IP core will not retain the previous values. Since all the control signals data type are declared as 'registers', the previous write status were latched and random values were written to the CAN IP registers.

Solution: Added line 10 and 11 to ensure that all the control signals are set to low and no arbitrary values are been written on to the registers.

Remarks: The problem was solved.

Problem No 2

Statement: Before the start of a transmission, the CAN IP core shows an high error status.

Observation: The CAN specifications [11893] states that the error status should only be high if transmit or receive error count is greater than or equal to 96. The value of 96 is written to the Error warning limit(EWL) register, during the configuration routine.

Reason: Before the value '96' is written to the EWL, the default value is taken as '0' and as soon as the CAN IP core is powered-up it change the error status to high.

Solution: The reset value of Error warning limit register was changed from '0' to '96' in the HDL code.

```
//previous code
```

```
always @ (posedge clock or posedge reset)
begin
  if (reset)
    EWL <= 8'd0;
  else if (write_ewl)
    EWL <=#1 input_data;
end
\\ Changed code
always @ (posedge clock or posedge reset)
begin
  if (reset)
    EWL <= 8'd96;
  else if (write_ewl)
    EWL <=#1 input_data;
end
```

Remarks: This solved the problem, and then there was no need to write the EWL by the host processor.

Problem No 3

Statement: During transmission, there were continuous errors generated.

Observation: Checked the Error capture (EC) register. The EC status = 10011011. EC interpretation is

- Error type= Form error
- Direction= Transmission
- Position = CRC Delimiter

The shows an error in CRC delimiter which is a form error.

Reason: The CRC calculation was done without the bit-stuffing consideration, hence on the receiver side when there was continuous stream of five similar bits, the next bit of the CRC bit is discarded hence generating a different CRC value.

Solution: Bit-stuffing is also applied on the CRC field.

Remarks: This solved the problem.

Problem No 4

Statement: The CAN IP core remains in reset mode (active on high), although the reset pin is set to low by the host processor.

Observation: In the pin assignment tool (PACE) of the FPGA, the reset pin is set as 'keeper'.

Solution: In PACE the pin assigned to the signal reset was set to a default Pull down value.

Remarks: It solved the problem, on the default the reset pin is pull down and does not keep the previous logic level(1 or 0).

Problem No 5

Statement: There is no communication on the CAN bus, although all the internal CAN signals demonstrate normal activity.

Observation: The SOF signal was inserted, immediately the CAN bus returns back to recessive state, indicating no communication.

Solution: Since the CAN bus speed was 1 Mbps, at high data speeds the PCA82c250 transceiver's pin 8 [Phi96a] needs to be connected to the ground via (1 to 10 $K\Omega$).

Remarks: Normal communication was observed, the pin 8 (Rs) of the transceiver is for slope control and this provide an effective device delay of 145 ns which is 200 ns in default condition.

Problem No 6

Statement: The communication between the CAN nodes was successfully working but with lot of intermittent errors.

Observation: The cable used is unshielded UTP and there was no common ground between the transceiver's on the CAN bus.

Solution: A new shielded UTP is used and a common ground is provided between the transceiver's.

Remarks: Improved and working.

A.3 Device Utilization

DEVICE UTILIZATION SUMMARY(Balanced)			
LOGIC UTILIZATION	USED	UNUSED	% USED
Number of slice Flip flops	687	9312	7
Number of 4 input LUTs	1693	9312	18
LOGIC DISTRIBUTION			
Number of occupied Slices	1105	4656	23
Number of slices with related Logic	1105	1105	100
Number of slices with unrelated logic	0	1105	0
TOTAL NO OF 4 INPUT LUTs	1715	9312	18
Number used as a logic	1693		
Number used as a route-thru	22		
Number of bounded IOBs	28	158	17
IOB Flip Flops	21		
Number of RAM16s	3	20	15
Number of BUFGMUXs	2	24	8
Number of DCMs	1	4	25
Average Fanout of Non-Clock Nets	3.66		
TOTAL EQ GATE COUNT	218,756		
Addition JTAG Gate count	1,392		

Table A.1: CAN IP device statistics (Balanced)

DEVICE UTILIZATION SUMMARY(Timing optimized)			
LOGIC UTILIZATION	USED	UNUSED	% USED
Number of slice Flip flops	673	9312	7
Number of 4 input LUTs	1598	9312	17
LOGIC DISTRIBUTION			
Number of occupied Slices	1062	4656	22
Number of slices with related Logic	1062	1062	100
Number of slices with unrelated logic	0	1062	0
TOTAL NO OF 4 INPUT LUTs	1715	9312	18
Number used as a logic	1598		
Number used as a route-thru	16		
Number of bounded IOBs	26	158	16
IOB Flip Flops	21		
Number of RAM16s	3	20	15
Number of BUFGMUXs	1	24	4
Number of DCMs	1	4	25
Average Fanout of Non-Clock Nets	3.66		
TOTAL EQ GATE COUNT	220,652		
Addition JTAG Gate count	1,392		

Table A.2: CAN IP device statistics (Timing Optimization)

DEVICE UTILISATION SUMMARY(Floor Plan)			
LOGIC UTILISATION	USED	UNUSED	% USED
Number of slice Flip flops	673	9312	7
Number of 4 input LUTs	1754	9312	18
LOGIC DISTRIBUTION			
Number of occupied Slices	901	4656	19
Number of slices with related Logic	901	1062	100
Number of slices with unrelated logic	0	1062	0
TOTAL NO OF 4 INPUT LUTs	1768	9312	18
Number used as a logic	1650		
Number used as a route-thru	14		
Number used for Dual port RAMs	104		
Number of bounded IOBs	26	158	16
IOB Flip Flops	21		
Number of RAM16s	3	20	15
Number of BUFGMUXs	1	24	4
Number of DCMs	1	4	25
Average Fanout of Non-Clock Nets	3.71		

Table A.3: CAN IP device statistics (After Floor planning)

A.4 Static Timing Analysis

```
Derived Constraint Report
Derived Constraints for TS_1
+-------------------------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+
|                               |   Period    |       Actual Period       |      Timing Errors        |      Paths Analyzed       |
|           Constraint          | Requirement |-------------+-------------|-------------+-------------|-------------+-------------|
|                               |             |   Direct    | Derivative  |   Direct    | Derivative  |   Direct    | Derivative  |
+-------------------------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+
|TS_1                           |     20.833ns|     10.000ns|     17.263ns|            0|            0|            0|        66887|
| TS_instance_External_Clock_CLK|     20.833ns|     17.263ns|          N/A|            0|            0|        66887|            0|
| 0_BUF                         |             |             |             |             |             |             |             |
+-------------------------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+

All constraints were met.


Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)

Setup/Hold to clock Source_Clock
------------+------------+------------+------------------+--------+
            |Max Setup to|Max Hold to |                  | Clock  |
Source      | clk (edge) | clk (edge) |Internal Clock(s) | Phase  |
------------+------------+------------+------------------+--------+
AD_0_7<0>   |    7.712(R)|   -1.593(R)|CLK0_OUT          |   0.000|
AD_0_7<1>   |    7.535(R)|   -1.462(R)|CLK0_OUT          |   0.000|
AD_0_7<2>   |    7.916(R)|   -1.654(R)|CLK0_OUT          |   0.000|
AD_0_7<3>   |    8.100(R)|   -1.520(R)|CLK0_OUT          |   0.000|
AD_0_7<4>   |    9.131(R)|   -1.464(R)|CLK0_OUT          |   0.000|
AD_0_7<5>   |    9.948(R)|   -1.506(R)|CLK0_OUT          |   0.000|
AD_0_7<6>   |    8.124(R)|   -1.575(R)|CLK0_OUT          |   0.000|
AD_0_7<7>   |    8.809(R)|   -1.670(R)|CLK0_OUT          |   0.000|
------------+------------+------------+------------------+--------+

Clock to Setup on destination clock Source_Clock
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
Source_Clock   |   17.263|         |         |         |
---------------+---------+---------+---------+---------+

TIMEGRP "AD_Group" OFFSET = IN 20.833 ns VALID 19.5 ns BEFORE COMP         "Source_Clock" "RISING";
Worst Case Data Window 8.486; Ideal Clock Offset To Actual Clock -5.378;
-----------------+------------+------------+---------+---------+-------------+
                 |            |            |  Setup  |  Hold   |Source Offset|
Source           |   Setup    |    Hold    |  Slack  |  Slack  |  To Center  |
```

```
------------------+------------+------------+---------+---------+-------------+
AD_0_7<0>         |    7.712(R)|   -1.593(R)|   13.121|    0.260|        6.431|
AD_0_7<1>         |    7.535(R)|   -1.462(R)|   13.298|    0.129|        6.585|
AD_0_7<2>         |    7.916(R)|   -1.654(R)|   12.917|    0.321|        6.298|
AD_0_7<3>         |    8.100(R)|   -1.520(R)|   12.733|    0.187|        6.273|
AD_0_7<4>         |    9.131(R)|   -1.464(R)|   11.702|    0.131|        5.786|
AD_0_7<5>         |    9.948(R)|   -1.506(R)|   10.885|    0.173|        5.356|
AD_0_7<6>         |    8.124(R)|   -1.575(R)|   12.709|    0.242|        6.233|
AD_0_7<7>         |    8.809(R)|   -1.670(R)|   12.024|    0.337|        5.844|
------------------+------------+------------+---------+---------+-------------+
Worst Case Summary|       9.948|      -1.462|   10.885|    0.129|             |
------------------+------------+------------+---------+---------+-------------+


Timing summary:
---------------

Timing errors: 0  Score: 0  (Setup/Max: 0, Hold: 0)

Constraints cover 67180 paths, 0 nets, and 7689 connections

Design statistics:
   Minimum period:  17.263ns{1}   (Maximum frequency:  57.927MHz)
   Minimum input required time before clock:   9.948ns


------------------------------------Footnotes-----------------------------------
1)  The minimum period statistic assumes all single cycle delays.

Analysis completed
--------------------------------------------------------------------------------
```

A.5 User Constraints File

```
#Created by Constraints Editor (xc3s500e-pq208-4)
#PACE: Start of Constraints generated by PACE
#PACE: Start of PACE I/O Pin Assignments
NET "AD_0_7<0>"  LOC = "p19" |IOSTANDARD = LVCMOS33 ;
NET "AD_0_7<1>"  LOC = "p18" |IOSTANDARD = LVCMOS33 ;
NET "AD_0_7<2>"  LOC = "p16" |IOSTANDARD = LVCMOS33 ;
NET "AD_0_7<3>"  LOC = "p15" |IOSTANDARD = LVCMOS33 ;
NET "AD_0_7<4>"  LOC = "p3" |IOSTANDARD = LVCMOS33 ;
NET "AD_0_7<5>"  LOC = "p2" |IOSTANDARD = LVCMOS33 ;
NET "AD_0_7<6>"  LOC = "p199" |IOSTANDARD = LVCMOS33 ;
NET "AD_0_7<7>"  LOC = "p200" |IOSTANDARD = LVCMOS33 ;
NET "ale_i"  LOC = "p12" |IOSTANDARD = LVCMOS33 ;
NET "bus_status" IOSTANDARD = LVCMOS33 ;
NET "chip_select"  LOC = "p14" |IOSTANDARD = LVCMOS33  |PULLDOWN ;
NET "interrupt" IOSTANDARD = LVCMOS33 ;
NET "led<0>"  LOC = "p146" |IOSTANDARD = LVCMOS33 ;
NET "led<1>"  LOC = "p147" |IOSTANDARD = LVCMOS33 ;
NET "led<2>" IOSTANDARD = LVCMOS33 ;
NET "led<3>" IOSTANDARD = LVCMOS33 ;
NET "led<4>" IOSTANDARD = LVCMOS33 ;
NET "led<5>" IOSTANDARD = LVCMOS33 ;
NET "led<6>" IOSTANDARD = LVCMOS33 ;
NET "led<7>" IOSTANDARD = LVCMOS33 ;
NET "read"  LOC = "p11" |IOSTANDARD = LVCMOS33 ;
NET "reset"  LOC = "P51" |IOSTANDARD = LVCMOS33  |PULLDOWN ;
NET "RXCAN"  LOC = "P54"  ;
NET "Source_Clock"  LOC = "p181" |IOSTANDARD = LVCMOS33 ;
NET "TXCAN"  LOC = "p56"  ;
NET "write_enable"  LOC = "p8" |IOSTANDARD = LVCMOS33 ;

NET "Source_Clock" TNM_NET = Source_Clock;
TIMESPEC TS_1 = PERIOD "Source_Clock" 20.833 ns HIGH 50%;
#Created by Constraints Editor (xc3s500e-pq208-4) - 2011/04/05
OFFSET = IN 20.833 ns VALID 16 ns BEFORE "Source_Clock" RISING;
INST "AD_0_7<0>" TNM = AD_Group;
INST "AD_0_7<1>" TNM = AD_Group;
INST "AD_0_7<2>" TNM = AD_Group;
INST "AD_0_7<3>" TNM = AD_Group;
INST "AD_0_7<4>" TNM = AD_Group;
INST "AD_0_7<5>" TNM = AD_Group;
INST "AD_0_7<6>" TNM = AD_Group;
INST "AD_0_7<7>" TNM = AD_Group;
TIMEGRP "AD_Group" OFFSET = IN 16 ns VALID 20.833 ns BEFORE "Source_Clock" RISING;
INST "ale_i" TNM = Cont_Group;
INST "read" TNM = Cont_Group;
INST "write_enable" TNM = Cont_Group;
TIMEGRP "Cont_Group" OFFSET = IN 16 ns VALID 20.833 ns BEFORE "Source_Clock" RISING;
```

Bibliography

[10B01] IEEE conformance test methodology for IEEE standards for local and metropolitan area networks - specific requirement 3: Carrier sense multiple access with collision detection (CSMA/CD)access method and physical layer specifications. *IEEE Std 1802.32001*, pages 1–85, 2001.

[11893] ISO 11898:1993(E). Road vehicles-interchange of digital information-Controller Area Network (CAN) for high speed communication. Technical report, ISO, November 1993.

[AG] A. Albert and W. Gerth. Evaluation and comparison of real-time performance of CAN and TTCAN. In *proceedings of ICC 2003 - 9th International CAN Conference*, Munich, Germany.

[Agi08] 16900 series logic analysis system mainframes. http://cp.literature.agilent.com/litweb/pdf/59890421EN.pdf., 2008.

[Alt] Altera. SignalTap II embedded logic analyser.

[AM05] A. Arora and S. Mahmud. Performance analysis of fault tolerant TTCAN system. In *proceeding of the SAE 2005 World Congress*, Detroit, Michigan, USA, April 2005.

[APF02] L. Almeida, P. Pedreiras, and J. Fonseca. The FTT-CAN protocol: Why and how. *IEEE Transaction on Industrial Electronics*, 49(6):1189–1201, Dec 2002.

[APSP07] D Ayavoo, M.J. Pont, M. Short, and S. Parker. Two novel shared-clock scheduling algorithms for use with 'Controller Area Network' and related protocols. *Microprocess. and Microsyst.*, 31(5):326–334, 2007.

[ARM05] ARM Ltd. *ARM Architecture Reference Manual*, 2005.

[BACM03] Di Blasi, F. A. Colucci, and R. Mariani. Y-CAN platform: A re-usable platform for design, verification and validation of CAN-based systems on a chip. In *ETS- 2003 Symposium*, May 2003.

[BB95] T.D. Burd and R.W. Brodersen. Energy efficient cmos microprocessor design. In *System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on*, volume 1, pages 288 –297, January 1995.

[BB01] I. Broster and A. Burns. Timely use of the CAN protocol in critical hard real-time systems with faults. In *Real-Time Systems, 13th Euromicro Conference on, 2001.*, pages 95–102, 2001.

[BB03] I. Broster and I. Burns. An analyzable bus-guardian for event triggered communication. In *24th IEEE Real-Time Systems Symposium*, pages 410–419, December 2003.

[BBRN04] I. Broster, A. Burns, and G. Rodriguez-Navas. Comparing real-time communication under electromagnetic interference. In *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*, pages 45 – 52, 2004.

[Bos91] R Bosch. CAN specification 2.0. Technical report, Robert Bosch Gmbh, 1991.

[Bos99] Bosch. *VHDL Reference CAN: User's Manual*, 1999.

[Bot98] B. Bottoms. The third millenniums test dilemma. *IEEE Design & Test of Computers*, 15:7–11, 10 1998.

[Cad04] Cadence Design Systems. *Incisive Unified Simulator*, 2004.

[CIA] CAN physical layer. http://www.can-cia.org/index.php?id=88.

[CiA06] Automotive electronics take off. In *CAN Newsletter*, volume Automotive. CiA, CAN in Automation, 2006.

[Cor02] S. Corrigan. *Introduction to the Controller Area Network (CAN)*. Texas Instruments, August 2002.

[CV99] G. Cena and A. Valenzano. Overclocking of Controller Area Networks. *Electronics Letters*, 35(22):1923–1925, October 1999.

[CV00] G. Cena and A. Valenzano. FastCAN: A high-performance enhanced CAN-like network. *Industrial Electronics, IEEE Transactions on*, 47(4):951–963, August 2000.

[CV06] G. Cena and A. Valenzano. On the properties of the flexible time division multiple access technique. *Industrial Informatics, IEEE Transactions on*, 2(2):86 – 94, May 2006.

[DBBL07] I. Davies, A. Burns, R. Brill, and J. Lukkien. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

[Dig05] Overview: Digilent jtag-usb cable and the Digilent and jtag-usb cable, 2005.

[DS95] M. DiNatale and J.A. Stankovic. Applicability of simulated annealing methods to real-time scheduling and jitter control. In *Real-Time Systems Symposium, 1995. Proceedings., 16th IEEE*, pages 190 –199, December 1995.

[EE96] H. Eisele and Jhnk. E. Can transceiver, 1996.

[eVC05] Can 2.0 evc. 2005.

[FA09] R. Froschauer and F. Auinger. A survey on the integration of the FlexRay bus in distributed automation and control systems. In *Logistics and Industrial Informatics, 2009. LINDI 2009. 2nd International*, pages 1 –6, sept 2009.

[FH00] B. Dieterle W. Fuhrer, T. Mller and F. Hartwich. Time-triggered communication on CAN (Time-Triggered can TTCAN). In *Proceedings of iCC 2000*, Amsterdam, The Netherlands, 2000.

[Fle04] FlexRay Consortium. *FlexRay Communications System Protocol Specification*, 2004.

[FOFF04] J. Ferreira, A. Oliveira, P. Fonseca, and J.A. Fonseca. An experiment to assess bit error rate in CAN. In *3rd international workshop on real-time networks RTN 2004, Proceedings*, June 2004.

[fpg09] fpga4fun.com. Knjn LLC, FX2 FPGA development boards, 2009.

[FRB99] M. Farsi, K. Ratcliff, and M. Barbosa. An overview of Controller Area Network. *Computing and Control Journal of Engineering*, 10(3):113–120, 1999.

[GBP00] R. Griessbach, J. Berwanger, and M. Peller. Byteflight neues hochleistungsdatenbussystem fr sicherheitsrelevante anwendungen. Technical report, Friedrich Vieweg & Sohn Verlagsgesellschaft mbH, January 2000.

[GDZG07] R. Garg, Chunjie Duan, Jinyun Zhang, and S. Gezici. Low power UWB transceiver design using dynamic voltage scaling. In *Wireless Communications and Networking Conference, 2007.WCNC 2007. IEEE*, pages 1757 –1762, march 2007.

[Gil60] E.N. Gilbert. Capacity of a burst-noise channel. *Bell Systems Technical Journal,*, 39:1253–1261, 1960.

[Gmb07] R.B. Gmbh. *E-Ray Flex Ray IP Module*, revision 1.2.6 edition, 2007. User manual.

[GN04] B. Gaujal and N. Navet. Fault confinement mechanisms on CAN : Analysis and improvements. *IEEE Transactions On Vehicular Technology*, 2004.

[GTA06] R. Ghostine, J. Thiriet, and J. Aubry. Dependability evaluation of networked control systems under transmission faults. In *IFAC Symposium Safe process 2006*, page 11291134, Beijing, PRC, 2006.

[HBS09] Z. Hanzalek, P. Burget, and P. Sucha. Profinet IO IRT message scheduling. *Real-Time Systems, Euromicro Conference on*, pages 57–65, 2009.

[IEE] IEEE 802 standard for local and metropolitan area networks: Overview and architecture.

[IEE94] Type 10baseT MAU conformance test methodology (Section 6). *IEEE Std 1802.3d-1993*, May 1994.

[IS] ISA-SP50-1987. Field busDraft standard.

[ISOa] ISO16845. Road Vehicles- Controller Area Network (CAN) - Conformance test plan.

[ISOb] ISO9646-1. Information technology- **ISO** - Conformance testing methodology and frame work- Part 1: General concepts.

[Jam04] P. James. Mechatronics and automotive systems design. *International Journal of Electrical Engineering Education*, 41:307–312, 2004.

[Jer77] A.J. Jerri. The Shannon sampling theorem: Its various extensions and applications: A tutorial review. *Proceedings of the IEEE*, 65(11):1565 – 1596, 1977.

[Kar02] A. Karlsson. X-by-wire systems and time-triggered protocols. Master's thesis, Uppsala University, Box 256 751 05 Uppsala, Sweden, November 2002.

[KB03] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112 – 126, 2003.

[KEI06] KEIL. *MCB2100: User's Guide*, July 2006.

[KEI08] uVision IDE tool, 2008.

[KG93] H. Kopetz and G. Grunsteidl. TTP - A time-triggered protocol for fault- tolerant real-time systems. In *23rd International Symposium on Fault-Tolerant Computing*, page 524533, 1993.

[KLSC00] T. Kim, J. Lee, H. Shin, and N. Chang. Best case response time analysis for improved schedulability analysis of distributed real-time tasks. In *In Proceedings ICDCS Workshops on Distributed Real-Time Systems*, pages 14–20, April 2000.

[Kop00] H. Kopetz. A comparison of CAN and ttp. *Annual Reviews in Control*, 24:177–188, 2000.

[KRWG96] A. Kirschbaum, F.M. Renner, A. Wilmes, and M. Glesner. Rapid-prototyping of a CAN-bus controller: A case study. In *In proceedings of Rapid System Prototyping, Seventh IEEE International Workshop on*, pages 146–151, 6 1996.

[kva] http://www.kvaser.com/en/support/bit-timing-calculator.html.

[LAB09] Lab VIEW. http://www.ni.com/labview86, 2009.

[Lai02] R. Lai. A survey of communication protocol testing. *Journal of Systems and Software*, 62:21–46, 2002.

[Law97] W. Lawrenz. *CAN system engineering: from theory to practical applications*, volume 1. Springer, 1997.

[LCZS09] F. Luo, J. Chen, G. Zhuang, and Z. Sun. Research on CAN controller conformance test system. In *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*, pages 582 –585, aug. 2009.

[LFY+07] T. Lee, Y. Fan, S. Yen, C. Tsai, and R. Hsiao. An integrated functional verification tool for FPGA systems. In *Second International Conference on Innovative Computing, Information and Control, ICICIC '07*, page 203, 9 2007.

[LH02] G. Leen and G. Heffernan. TTCAN: a new time-triggered Controller Area Network. *Microprocessors and Microsystems*, 26(2):77–94, 2002.

[LK98] P. Lawrenz, W. Kinowski and G. Kircher. CAN conformance testing - state of the art and test experience. In *In Proceedings of 5th International CAN Conference iCC98*, San Jose, California, Nov 1998.

[LKJ99] M.A. Livani, J. Kaiser, and W. Jia. Scheduling hard and soft real-time communication in a controller area network. *Control Engineering Practice*, 7(12):1515 – 1523, 1999.

[LKK98] W. Lawrenz, P. Kinowski, and G. Kircher. CAN conformance testing-The developing ISO standard and necessary extensions. In *In Proceedings of International Truck and Bus Meeting and Exposition*, Indianapolis, Indiana, November 1998.

[MAF05] E. Martins, L. Almeida, and A. Fonseca. An FPGA-based coprocessor for real-time field bus traffic scheduling–architecture and implementation. *Journal of Systems Architecture*, 51(1):29–44, 2005.

[Men07] Mentor Graphics. *Modelsim*, 2007.

[MH92] F. Miesterfeld and R. Halter. Survey of vehicle multiplexing encoding techniques. In *Automotive Technology International 92*, pages 253–265. Sterling Publications International, 1992.

[Mic03] Microchip. MCP2515 stand-alone CAN controller with SPI interface. http://www.avrcard.com/Documents/datasheets/mcp2515.pdf, 2003.

[MMTS96] H. Mori, Y. Mano, H. Takada, and K. Sakamura. μITRON bus: a real-time control LAN for open network environment. In *Real-Time Computing Systems and Applications, 1996. Proceedings., Third International Workshop on*, pages 227–234, October 1996.

[MP10] A. Muhammad and M.J. Pont. A time-triggered communication protocol for CAN-based networks with a fault-tolerant star topology. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 2347 –2354, July 2010.

[NDK+05] K. Nimsub, K. Dawi, C. Kyuhyung, K. Jinsang, and C. Wonkyung. Design and verification of a CAN controller for custom ASIC. In *CAN in Automation Proceedings of 10th iCC*, 2005.

[NHN02] T. Nolte, H. Hansson, and C. Norstrm. Minimizing CAN response-time jitter by message manipulation. In *8th Real-time and Embedded Technology and Applications Symposium, Proceedings*, pages 197–206, 2002.

[NHNP01] T. Nolte, H. Hansson, C. Norstrm, and S. Punnekkat. Using bit-stuffing distributions in CAN analysis. In *IEEE Real-Time Embedded Systems Workshop, Proceeding*, December 2001.

[NI 08] National Instruments. *1 Port, High Speed CAN, USB Interface*, 2008.

[Nov09] J. Novak. Flexible approach to the Controller Area Networks test and evaluation. In *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2009. IDAACS 2009. IEEE International Workshop on*, pages 44 –48, September 2009.

[NP03] R.G. Navas and J. Proenza. Analyzing atomic broadcast in TTCAN networks. In *5th IFAC International Conference on Fieldbus Systems and their Applications (FET 2003)*, pages 153–156, Aveiro, Portugal, 2003.

[NP05] M. Nahas and M. Pont. Using XOR operations to reduce variations in the transmission time of CAN messages: a pilot study. In A. Koelmans, Bystrov, A. Pont, R. M Ong, and A. Brown, editors, *Proceedings of the Second UK Embedded Forum 2005*, pages 4–17, Birmingham, UK, October 2005.

[NPS09] M. Nahas, M. Pont, and M. Short. Reducing message-length variations in resource-constrained embedded systems implemented using the Controller Area Network (CAN) protocol. *Journal of Systems Architecture*, 55:344–354, 2009.

[NSP05] M. Nahas, M. Short, and M. Pont. The impact of bit stuffing on the real-time performance of a distributed control system. In *10th International CAN conference,Proceeding*, pages 10–1 to 10–7, Rome, Italy, March 2005.

[NXP02] NXP. *TJA1050: High speed CAN transceiver*, 2002.

[OAF05] A.R. Oliveira, N.L. Arqueiro, and P.N. Fonseca. CLAN A technology independent synthesizable CAN controller. In *proceedings of ICC 2005 - 10th International CAN Conference*. CiA - CAN in Automation, 2005.

[OMS05] O. Oltu, P. Milea, and A. Simion. Testing of digital circuitry using Xilinx Chipscope logic analyzer. In *Proceedings International Semiconductor Conference, CAS 2005*, volume 2, pages 471–474, October 2005.

[Par07] D. Paret. *Multiplexed Networks for Embedded System*. John Wiley & Sons Ltd, Chister, England, 2007.

[PF04] J.R. Pimentel and J.A. Fonseca. FlexCAN: A flexible architecture for highly dependable embedded applications. In *3rd Int. Workshop on Real-Time Networks, Proceedings*, Catania, Italy, July 2004.

[Phi96a] Philips. *Application Note: PCA82C250/251 CAN Transceiver*, 1996.

[Phi96b] Philips Semiconductors. *Application Note: PCA82C250/251 CAN Transceiver*, 1996.

[Phi04] Philips. *LPC2119/2129/2194/2292/2294 User Manual*. Philips Semiconductors, 2004.

[PHN00] S. Punnekkat, H. Hansson, and C. Norstrm. Response time analysis under errors for CAN. In *In Proceedings of the 6th Real-Time Technology and Applications Symposium (RTAS)*, pages 258–265. IEEE Computer Society, 2000.

[PKS07] K. Park, M. Kang, and D. Shin. Mechanism for minimizing stuffing-bit in CAN messages. In *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE*, pages 735 –737, Nov 2007.

[Pon01] M.J. Pont. *Patterns for Time-Triggered Embedded Systems*, volume 1. ADDISON-WESLEY, 2001.

[PP07] J.R. Pimentel and J. Paskvan. Experimental jitter analysis in a flex-CAN based drive-by-wire automotive application. In *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, pages 290 –293, June 2007.

[PRA06] R. Pinto, J. Rufno, and C. Almeida. Specification and engineering of the CANELy prototype board. Technical Report DARIO Technical Report RT-06-06, Instituto Superior Tecnico, Lisbon, Portugal, October 2006.

[QPFM05] C. Quigley, B. Pope, J. Finney, and R. McLaughlin. An automotive specification of a time-triggered CAN implementation: Doubling CAN's usable data. *SAE Transactions*, 114(7):509–518, 2005.

[Raj06] N. Raja. FPGA implementation of a SIP message processor. Master's thesis, Computer Engineering Department, North Carolina University, 2006.

[RG88] T.V. Ramabadran and S.S. Gaitonde. A tutorial on crc computations. *Micro, IEEE*, 8(4):62 –75, 8 1988.

[RNRP+04] G. Rodriguez-Navas, J. Rigo, J. Proenza, J. Ferreira, L. Almeida, and J.A. Fonseca. Design and modeling of a protocol to enforce consistency among replicated masters in FTT-CAN. In *Factory Communication Systems, 2004. Proceedings. 2004 IEEE International Workshop on*, pages 229 – 238, September 2004.

[RNRP08] G. Rodrguez-Navas, S. Roca, and J. Proenza. Orthogonal, fault-tolerant and high-precision clock synchronization for the Controller

Area Network. *IEEE Transactions on Industrial Informatics*, 4(2):92–101, May 2008.

[RTE07] Industrial communication networks - profiles - Part 2: Additional field bus profiles for real-time networks based on iso/iec 8802-3. 2007.

[Ruf02] J. Rufino. *Computational System for Real-Time Distributed Control*. PhD thesis, Departamento de Engenharia Electrotecnica e de Computadores,Instituto Superior Tecnico, Lisbon, Portugal, 2002.

[RVA+98] J. Rufino, P. Verissimo, G. Arroz, C. Almeida, and L. Rodrigues. Fault-tolerant broadcasts in CAN. In *Fault-Tolerant Computing, 1998. Digest of Papers. Twenty-Eighth Annual International Symposium on*, pages 150–159, Jun 1998.

[RVA03] J. Rufino, P. Verssimo, and G. Arroz. Node failure detection and membership in CANELy. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks*, California, USA, June 2003.

[SG00] I Schieferdecker and J. Grabowski. Conformance testing with TTCN. *Languages for Telecommunications Applications*, 96(4):85–95, 2000.

[SJA00] SJA1000 stand-alone CAN controller, Jan 2000. Data Sheet.

[SK99] R. Stuart and E. Kilbride. *Application Notes: CAN Bit Timing Requirements*. FreeScale semiconductors, 1999.

[Skl88] B. Sklar. *Digital communications: fundamentals and applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[Smi96] D. Smith. VHDL and Verilog compared and contrasted plus modelled example written in VHDL, Verilog and C. In *ACM/IEEE Design Automation Conference*, pages 771–776, June 1996.

[SN06] R. Saket and N. Navet. Frame packing algorithms for automotive applications. *Embedded Computing*, 2(1):93–102, 2006.

[Sof07] Softing AG. *CAN/CANopen/DeviceNet Interface boards*, 2007.

[SP07] M. Short and M.J. Pont. Fault-tolerant time-triggered communication using CAN. *Industrial Informatics, IEEE Transactions on*, 3(2):131–142, May 2007.

[SPF08] M. Short, M. Pont, and J. Fang. Assessment of performance and dependability in embedded control systems: Methodology and case study. *Control Engineering Practice*, 16(11):1293 – 1307, 2008.

[SS09] I. Sheikh and M. Short. CAN conformance testing-A new approach. Technical Report Tech-Report ESL-09-01, ESL, Engineering Department, University of Leicester, Feb 2009.

[Sta07] W. Stallings. *Data and Computer Communications*. Pearson Prentice Hall, 8th edition, 2007.

[TBW95] K. Tindell, A. Burns, and A. Wellings. Calculating Controller Area Network (CAN) message response times. *Control Engineering Practice*, 3(8):1163–1169, 1995.

[TDT+06] H. Tan, R.F. DeMara, A.J. Thakkar, A. Ejnioui, and A.D. Sattler. Complexity and performance evaluation of two partial reconfiguration interfaces on FPGAs: A case study. In *In Proceedings of the ERSA*, 2006.

[Tex00] Texas Instruments. *SN65LBC176A: Differential Bus Transceivers*, 2000.

[TLA] TLA 5000b logic analyzers.

[TMS97] *Texas Instruments Reference set volume 1 & 2: TMS320C243 DSP controllers*, 1997.

[Tr98] M. Trngren. Fundamentals of implementing real-time control applications in distributed computer systems. *Real-Time Systems*, 14:219–250, 1998.

[Tra03] Deep storage with Xilinx Chipscope pro and Agilent technologies FPGA trace port analyzer, 2003.

[TTE] http://www.tte-systems.com/.

[UK93] B. Upender and P. J. Koopman. Embedded communication protocol options. In *Proceedings of the Embedded Systems Conference*, San Jose, CA, October 1993.

[Ver06] IEEE std 1364 -2005 IEEE standard for Verilog hardware description language. *IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001)*, pages 1–560, 2006.

[War09] D. Waraus. Steer-by-wire system based on FlexRay protocol. In *Applied Electronics, 2009. AE 2009*, pages 269 –272, sept 2009.

[WBTMG96] A. Winter, D. Bittruf, Y. Tanurhan, and K.D. Muller-Glaser. Rapid prototyping of a communication controller for the CAN bus. In *Rapid System Prototyping, 1996. Proceedings., Seventh IEEE International Workshop on*, pages 152–157, Jun 1996.

[wik] http://en.wikipedia.org/wiki/edge_case.

[XAP03] Xilinx Inc. *Using Digital Clock Managers (DCMs) in Spartan-3 FPGAs*, 2003. Application Note.

[xil] http://www.xilinx.com/itp/xilinx5/help/xpower/html/d_definitions /d_clock_to_setup_path.htm.

[Xil07] Xilinx. *Chipscope Pro Software and Cores*, January 2007.

[Xil08] ISE foundation, 2008.

[ZCD+06] G. Zarri, F. Colucci, F. Dupuis, R. Mariani, M. Pasquariello, G. Risaliti, and C. Tibaldi. On the verification of automotive protocols. In *In Proceedings of Design, Automation and Test in Europe*, volume Mar, pages 6–10, 3 2006.

[ZT09] S. Ziermann, T. Wildermann and J. Teich. CAN+: A new backward-compatible Controller Area Network (CAN) protocol with up to 16x higher data rates. In *2009 Design, Automation and Test in Europe*, pages 1088–1093, April 2009.