

**FPGA IMPLEMENTATION
OF
REPROGRAMMABLE CONTROLLER FOR NCAP**

A DISSERTATION

*Submitted in partial fulfillment of the
requirements for the award of the degree*

of

MASTER OF TECHNOLOGY

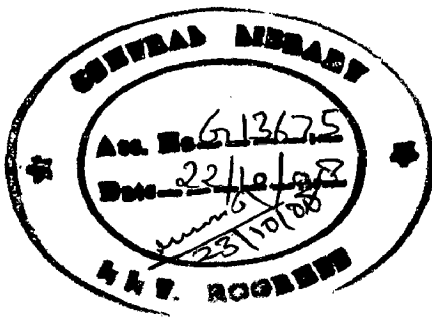
in

ELECTRICAL ENGINEERING

(With Specialization in Measurement and Instrumentation)

By

RATHNAKAR REDDY JUNUTHULA



**DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE - 247 667 (INDIA)
JUNE, 2008**

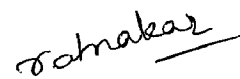
IR

I hereby declare that the work which is being presented in this dissertation entitled, "**FPGA IMPLEMENTATION OF REPROGRAMMABLE CONTROLLER FOR NCAP**" submitted in the partial fulfillment of the requirements for the award of the degree "**Master of Technology**" with specialization in **Measurement & Instrumentation**, to the **Department of Electrical Engineering, IIT Roorkee**, Roorkee is an authentic record of my own work carried out during the period from August 2007 to June 2008 under the supervision of **Dr. H.K.Verma**, Professor, Department of Electrical Engineering, IIT Roorkee, Roorkee.

The matters embodied in this report have not been submitted by me for the award of any other degree or diploma.

Date: ³⁰ June 2008


Place: Roorkee


(RATHNAKAR REDDY JUNUTHULA)

This is to certify that above statement made by candidates is correct to the best of my knowledge.

Date: 30.6.2008

Place: Roorkee



Dr H.K.Verma

Professor

Department of Electrical Engineering,
Indian Institute of Technology ,
Roorkee.

ACKNOWLEDGEMENT

I take this opportunity to express my sincere gratitude towards my supervisor, **Prof. H.K.Verma**, Professor, Electrical Engineering Department, IIT Roorkee for the guidance, advice and encouraging support which were invaluable for the completion of this dissertation work. I feel it is my fortune to have opportunity of working under his valuable guidance. This thesis work was enabled and sustained by his vision and ideas.

I am indebted to all the faculty of Measurement and Instrumentation group for their encouragement and suggestions, which helped me in finishing this work.

There are no words that adequately express my gratitude to my parents, for the wealth of love, support, guidance and practical assistance they have given me life-long.

Last, but not the least, I wish to thank all my friends and well wishers for their constant encouragement and support at various stages of this work.

Date: June 2008

Place: Roorkee.

Rathnakar
(RATHNAKAR REDDY JUNUTHULA)

Abstract

Sensor networking is a fast growing technology. Networked transducers offer many advantages to users. Today multiple control and sensor networking solutions are emerging, each requiring a separate and significant effort on the part of transducer manufacturers. It is too costly for transducer manufacturers to make unique smart transducers for each network on the market. Therefore a universally accepted transducer interface standard, the IEEE 1451 standard, has been evolved. The family of IEEE 1451 standards provides a common interface and enabling technology for the connectivity of transducers to microprocessors, control and field networks, and data acquisition and instrumentation systems.

IEEE 1451.1 defines Network Capable Application Processor for connecting communication networks on one side, and sensors (or actuators) on the other. An Interface controller is required while connecting the NCAP to the communication network. This report presents the successful development of a reprogrammable interface controller that is important part of network capable application processor (NCAP). This NCAP features low redundancy of hardware because interfaces are implemented by software. The RIC is implemented on FPGA board so that NCAP get the ability to reconfigure types and parameters of the communication Interfaces according to meet user requirements so required flexibility of our NCAP can be achieved. The interface controller has been designed in VHDL, and targeted to Spartan 3E FPGA kit. Important aspects of these tools are described in detail in various sections of the report. Design, simulation and FPGA implementation of a protocol controller for the Controller Area Network (CAN) 2.0A multi-master serial communication protocol are described. The design method of each functional block is also presented in the report as an experience of how to easily design digital systems to be reused in different applications, assuring its quality and reliability.

CONTENTS

Candidate's Declaration	i
Acknowledgements	ii
Abstract	iii
Table of Contents	iv
List of Figures	viii
List of Tables	ix
1 Introduction	1
1.1 The IEEE 1451 standard	1
1.1.1 IEEE 1451 goals	1
1.1.2 History	2
1.1.3 How can IEEE 1451 are applied	3
1.1.4 Benefits	4
1.2 The Family of standards	4
1.2.1 IEEE 1451.2	4
1.2.2 IEEE 1451.1	5
1.2.3 IEEE 1451.3	5
1.2.4 IEEE 1451.4	6
1.3 Objective and scope of work	8
1.4 Organization of report	8
2. NCAP Development	10
2.1 Network Capable Application Processor	10
2.1.1 Introduction	10

2.1.2 Physical view	11
2.1.3 Logical view	11
2.1.4 Information model	12
2.1.5 Data model	13
2.2 IEEE 1451.1 goals	13
2.3 IEEE 1451.1 users	14
2.4 IEEE 1451.1 benefits	15
2.5 Implementation of network smart sensors	15
2.5.1 Description of IEEE 1451.2 standard	15
2.5.2 NCAP	17
2.5.3 Hardware Implementation of network smart sensor	17
2.5.4 NCAP for CAN bus	18
3. Hardware software platforms	20
3.1 Software platforms	20
3.1.1 Hardware Description Language	20
3.1.2 Designing FPGA devices with Synthesis tools	20
3.2 Design flow	21
3.2.1 Steps in Implementation	22
3.3 Hardware platform	25
4. Controller Area Network (CAN) protocol	28
4.1 CAN	28
4.2 CAN network protocol	28
4.2.1 CAN properties	30
4.2.2 Layered structure of a CAN node	30
4.3 Messages	31

4.3.1 Data frame	31
4.3.2 Remote frame	34
4.3.3 Error frame	35
4.3.4 Overload frame	35
4.4 Bit stuffing	36
4.5 Error Detection	36
4.6 Error signaling and Recovery time	37
4.7 Fault Confinement	37
4.8 Connections	37
4.9 Single channel	37
4.10 Bus values	37
4.11 Acknowledgment	37
4.12 Error handling	37
4.13 Error signaling	38
4.14 Definition of Transmitter/Receiver	39
5. Design and Implementation of RIC	40
5.1 Reprogrammability requirement	40
5.2 Design of IEEE 1451 based smart module for network communications	41
5.3 Functional Description	42
5.3.1 Interface Management logic	43
5.3.2 Transmit Buffer	43
5.3.3 Receive Buffer	43
5.3.4 Acceptance checker	43
5.3.5 Bit Stream processor	43

5.3.6 Bit Timing Logic	44
5.3.7 Error Management logic	44
5.4 Implementation and state machine diagrams	44
5.4.1 Parameter registers	47
5.4.2 Transmitter buffer	48
5.4.3 Data/Remote frame generation	49
5.4.4 par-ser converter	50
5.4.5 CRC generator	51
5.4.6 Bit stuff unit	52
5.4.7 Overload/Error frame generation	53
5.4.8 Serialized frame transmitter	55
5.4.9 Message processor	55
5.4.10 Arbitration controller	56
5.4.11 Synchronizer	56
5.4.12 Bit destuff unit	57
5.4.13 Error checkers	58
5.4.14 Acceptance checker	60
5.4.15 Receive buffer	61
5.4.16 serial to parallel converter	61
5.4.17 Interface Management logic	62
6. Results	64
6.1 Simulation Results	64
6.2 Synthesis Results	67
7. Conclusion and Future scope	75
7.1 Conclusion	75

REFERENCES

APPENDIX

List of Figures:

1.1 How IEEE 1451 can be applied	3
1.2 IEEE 1451.2 STIM and NCAP Interface	5
1.3 IEEE 1451.4 mixed mode transducer	6
1.4 The family of IEEE 1451 standards	7
2.1 Network Capable Application Processor	11
2.2 Hardware model of Network Interface module	15
2.3 Conversion of 1451.2 models in different network	16
2.4 Whole structure of NCAP	18
2.5 NCAP structure for CAN	18
3.1 Design flow	21
3.2 create a new project	23
3.3 post implementation summary at a glance	23
3.4 assigning package pins	24
3.5 Impact dialogue box	24
3.6 Assigning configuration file	25
3.7 SPARTAN 3E FPGA kit	26
4.1 Layers of CAN	30
5.1 IEEE 1451 based smart module for a network	41
5.2 NCAP and CAN bus Interface	42

5.3 Blocks of Interface controller	42
5.4 Complete block diagram with internal blocks (modules)	45
5.5 State diagram for registers	48
5.6 Synthesized RTL of Tx_buffer and loadmsg signal	49
5.7 CAN data/remote frame generation	49
5.8 Dataflow diagram of parallel to serial converter	50
5.9 State diagram parallel to serial converter	51
5.10 CRC (14:0) out synthesized RTL	52
5.11 Overload frame generation	54
5.12 Arbitration mechanism	56
5.13 Dataflow diagram of clock divider	57
5.14 Bit destuffing	58
5.15 CRC checker	58
5.16 Dataflow diagram of CRC checker	59
5.17 CAN acknowledgement process	60
5.18 SM chart for serial to parallel converter	63

List of Tables

1.1 History of Standards	2
2.1 Transducer Independent Interface	17
4.1 Data bytes for corresponding DLC	33
5.1 Interface Management logic	62

CHAPTER 1

Introduction

1.1 The IEEE 1451 Standard:

The IEEE 1451 smart transducer interface standards provide the common interface and enabling technology for the connectivity of transducers to microprocessors, control and field networks, and data acquisition and instrumentation systems.

The standardized TEDS specified by IEEE 1451.2 allows the self-description of sensors and the interfaces provide a standardized mechanism to facilitate the ‘plug and play’ of sensors to networks. The network-independent smart transducer object model defined by IEEE 1451.1 allows sensor manufacturers to support multiple networks and protocols. Thus, transducer-to-network interoperability is on the horizon. The inclusion of 1451.3 and 1451.4 to the family of 1451 standards will meet the needs of the analog transducer users for high-speed applications. In the long run, transducer vendors and users, system integrators, and network providers can all benefit from the IEEE 1451 interface standards.

1.1.1 IEEE 1451 Overview/ Goals:

1. Provide standardized communication interfaces for smart transducers, both sensors and actuators. In the form of a standard hardware and software definition/specification.
2. Simplify the connectivity and maintenance of transducers to device networks through such mechanisms as common Transducer Electronic Data Sheet (TEDS) and Standardized Application Programming Interfaces (API).
3. Allow plug-and-play with 1451 compatible transducers among different devices using multiple control networks
4. Give sensor manufacturers, system integrators, and end -users the ability to support multiple networks and transducer families in a cost effective way

1.1.2 History:

In September 1993, the National Institute of Standards and Technology (NIST) and the Institute of Electrical and Electronics Engineers (IEEE)'s Technical Committee on Sensor Technology of the Instrumentation and Measurement Society co-sponsored a meeting to discuss smart sensor communication interfaces and the possibility of creating a standard interface. The response was to establish a common communication interface for smart transducers. Since then, a series of five workshops have been held and four technical working groups have been formed to address different aspects of the interface standard. The 1451.1 working group aims at defining a common object model for smart transducers along with interface specifications for the components of the model. The 1451.2 working group aims at defining a smart transducer interface module (STIM), a transducer electronic data sheet (TEDS), and a digital interface to access the data. The 1451.3 working group aims at defining a digital communication interface for distributed multi drop systems. The 1451.4 working group aims at defining a mixed-mode communication protocol for smart transducers. This family of IEEE 1451 standards is designed to work in concert with each other to ease the connectivity of sensors and actuators into a device or field network.

The working groups created the concept of smart sensors to control networks interoperability. So far, the project has produced a set of specifications which is approved and published by IEEE as IEEE Std 1451.2-1997, Standard for a Smart Transducer Interface for Sensors and Actuators - Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats. The table 1.1 describes the complete history of standards

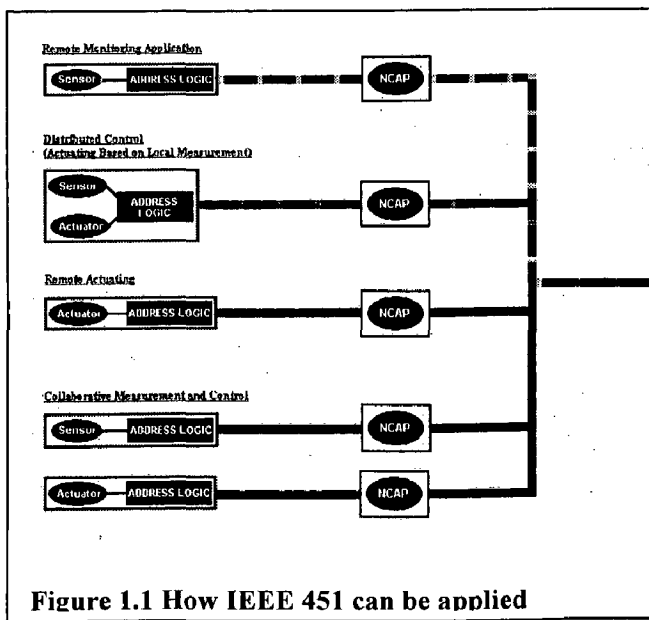
Table 1.1 History of standards

IEEE 1451.0	2007
IEEE 1451.1	1999
IEEE 1451.2	1997

IEEE 1451.3	1999
IEEE 1451.4	2004
IEEE 1451.5	2007
IEEE 1451.6	Proposed(PAR approved in 2004)
IEEE 1451.7	Proposed(PAR approved in 2007)

1.1.3 How can IEEE 1451 be applied?

- Remote monitoring.
- Distributed control.
- Remote actuating.
- Collaborative Measurement and Control.



These applications are shown in blocks in figure 1.1.

1.1.4 BENEFITS:

1. Enable any smart transducer to interface with any network protocol.
2. Enables any smart transducer with any network protocol.
3. Facilitate network interoperability
4. Concept of TEDS
5. Concept of self diagnosis
6. Eliminates wiring incase of wireless sensors

1.2 The Family of standards:

1.2.1 IEEE 1451.2 Transducer-to-Microprocessor Communication Interface:

The IEEE 1451.2 defines a Transducer Electronic Data Sheet (TEDS) and its data format, along with a 10-wire digital interface and communication protocol between transducers and a microprocessor. The framework of the IEEE 1451.2 interface [2] is shown in Figure 1.2. The TEDS, stored in a nonvolatile memory, contains fields that describe the type, attributes, operation, and calibration of the transducer. A transducer integrated with a TEDS provides a feature that makes the self-description of transducers to the network possible. Since the transducer manufacturer data in the TEDS always goes with the transducer and this information is electronically transferred to a Network Capable Application Process (NCAP) or host, human errors associated with entering sensor parameters manually is completely eliminated. Since the manufacturer data and calibration data (optional) are stored in the TEDS, losing transducer paper data sheet is no longer a concern. With the TEDS feature, upgrading transducers with higher accuracy and enhanced capability and replacing transducers for maintenance purposes are simply “plug and play”. The 1451.2 interface defines the Smart Transducer Interface Module (STIM). Up to 255 sensors or actuators of various digital and analog mixes can be connected to a STIM. The STIM is connected to a network node called NCAP

through the 10-wire Transducer Independent Interface using a modified Serial Peripheral Interface (SPI) for data transfer.

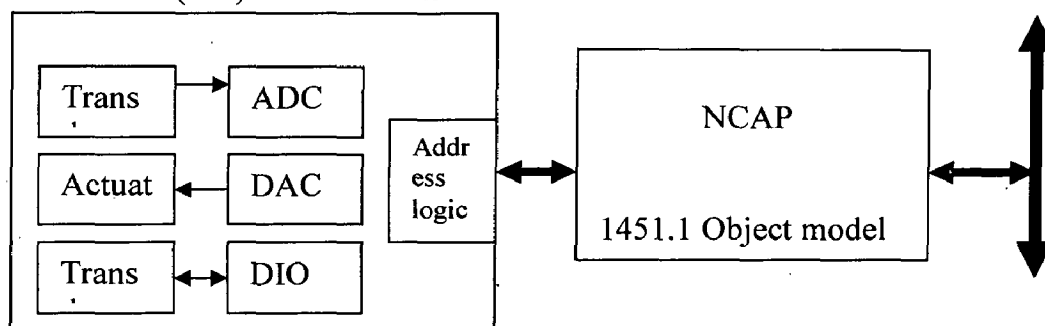


Figure 1.2 1451.2 STIM and NCAP Interface

1.2.2 IEEE 1451.1 - Networked Smart Transducer Model:

The IEEE 1451.1 project defines a common object model for a networked smart transducer and the software interface specifications for each class representing the model. Some of these classes form the blocks, components, and services of the conceptual transducer. The networked smart transducer object model encapsulates the details of the transducer hardware implementation within a simple programming model. This makes programming the sensor or actuator hardware interface less complex by using an input/output (I/O)-driver paradigm. The network services interfaces encapsulate the details of the different network protocol implementations behind a small set of communications methods [1].

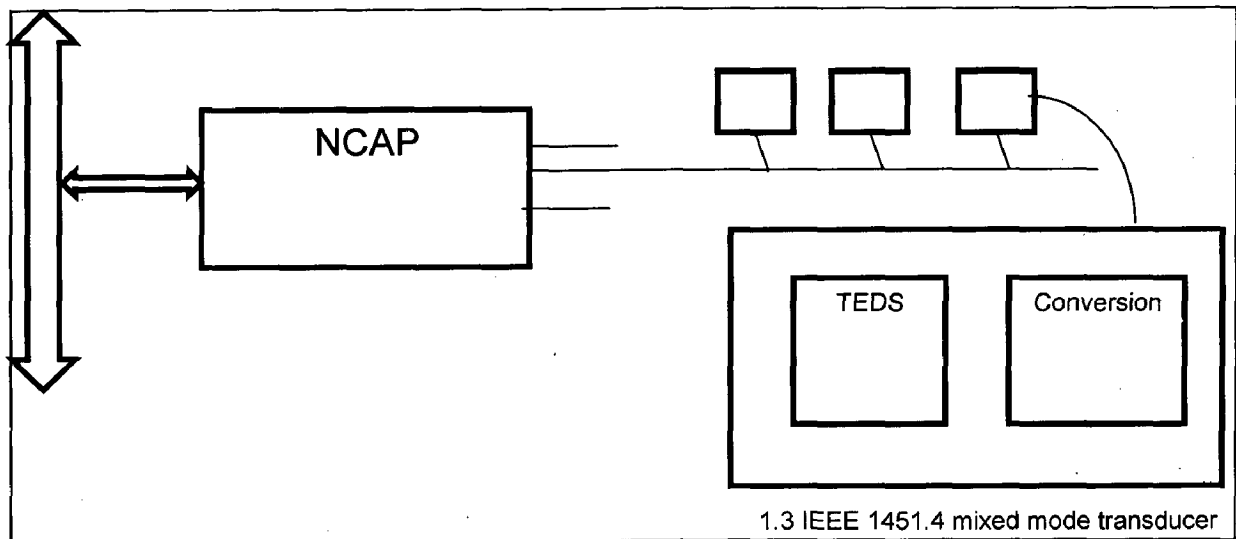
1.2.3 IEEE 1451.3 – Multi drop Distributed System:

Interfacing Smart Transducers During the course of the development of the IEEE 1451.1 and 1451.2 standards, some sensor manufacturers and users recognized the need for a standard interface for multi drop distributed smart sensor systems. In a distributed system a large array of sensors, in the order of hundreds, needs to be read in a synchronized manner. The bandwidth requirements of these sensors might be relatively high, in the order of several hundred kilohertz, with time correlation requirements in the range of nanoseconds. The IEEE 1451.3 was created to define the specification for such a standard. A single transmission line is proposed to be used to supply power to the transducers and to provide the communications between the bus controller and the

Transducer Bus Interface Modules (TBIM)[8]. A transducer bus is expected to have one bus controller and many TBIMs. A TBIM may contain one or more different transducers. The NCAP contains the controller for the bus and the interface to the network that may support many other buses.

1.2.4 IEEE 1451.4 - Mixed-mode Transducer and Interface:

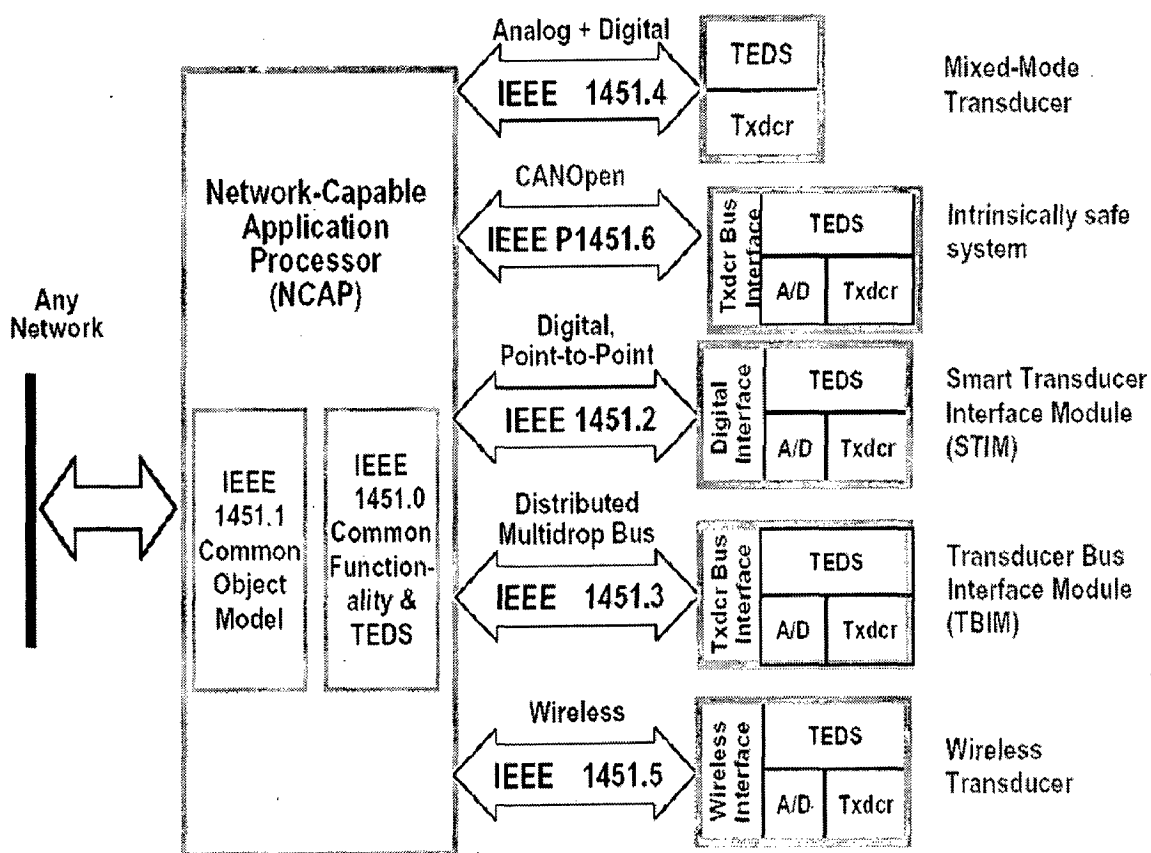
In order to reduce cabling and interfacing costs, a model using different wiring configurations is chosen as a transducer connection interface. If a single wire model is used, the analog transducer signal transmission and communication of the digital TEDS data to an instrument or a network are done on the same wire, but at separate times. If a multi-wire model is used, communication of digital data and analog signal can be accomplished simultaneously. The digital communication can be used to read the TEDS information and to configure an IEEE 1451.4 Transducer. The context of the mixed-mode transducer and its interface(s) are shown in Figure 1.3.



In the condition-based monitoring and maintenance industry, analog transducers such as piezoelectric, piezoresistive, and accelerometer-based transducers are used with electronics instruments to measure the conditional state of machinery. Transducer measurements are sent to an instrument or computer for analysis. The idea of having a small TEDS on an analog transducers and the ability to connect the transducers to a network have attracted transducer companies to work on the IEEE 1451.4 interface

standard. An IEEE 1451.4 Transducer, which could be a sensor or actuator with typically one addressable device, is referred to as a node containing a TEDS. The IEEE 1451.4 Transducer may be used to sense or control multiple physical phenomena. Each phenomenon sensed or controlled is associated with a node.

IEEE 1451 Family of Smart Transducer Interface Standards



Txdcr = Transducer (Sensor or Actuator)

Figure 1.4 The family of IEEE 1451 standards[8]

1.3 Objective and scope of work:

Broadly major objective is to present the development of a Network capable Application Processor (NCAP) compatible with IEEE 1451 standard.

In this work, the features of reprogrammable Interface controller (RIC) development are considered. RIC is the main part of Network Capable Application Processor.

This NCAP features low redundancy of hardware because interfaces are implemented by software. The RIC is implemented on FPGA board so that NCAP get the ability to reconfigure types and parameters of the communication Interfaces according to meet user requirements. So required flexibility of our NCAP can be achieved.

Considering the application areas of most of the sensors (like fire detector, glass break detector) CAN (Controller area Network) bus is selected as field network. CAN bus have some merits such as lowcost, high security and better anti jamming [7]. CAN (Controller Area Network) is a standard protocol for control networks . It was initially developed for control networks in automobiles, but now its is being used for other control applications as home systems, medical devices, industrial control, etc. The interest in CAN is increasing rapidly due to the different applications that are foreseen and the availability of devices integrating CAN in the market. Some of these applications will require higher levels of integration to reduce the size, the power consumption and the price of the final system.

In this work detailed description of design and implementation of RIC is presented.

1.4 Organization of report:

- Chapter 1 includes a brief introduction about family of IEEE 1451 standards. The various goals and benefits of the 1451 standard for smart transducer interface and how it can be applied is discussed along with different sub standards.
- In Chapter 2, concentrates on development of Network Capable Application Processor (NCAP) i.e. IEEE 1451.1 standard and explains the implementation of network smart sensor on a single chip.
- The Chapter 3 explains about the various softwares and hardware kits utilized for this dissertation. The softwares and the FPGA development kits are listed. Detailed explanation about each of them is included for a better understanding.

- The chapter 4 describes the CAN 2.0A network protocol for which Interface controller is designed in this work
- In Chapter 5, the implementation details of all the blocks of CAN controller is explained, results, schematics, synthesis details, simulations are all included wherever required.
- Chapter 6 summarizes the various results and discussions. The results included in this chapter are those which cannot be provided as a part of the other chapters but have overall significance.
- Chapter 7 gives the Conclusions and Scope for Future work. This includes the concluding remarks and the future developments expected in the field of sensor networks.

CHAPTER 2

NCAP Development

2.1 Network Capable Application Processor:

1451.1: Standard for Smart Transducer Interface for Sensors and Actuators-Network Capable Application Processor Information Model.

This standard defines an object model with a network-neutral interface for connecting processors to communication networks, sensors, and actuators. The object model containing blocks, services, and components specifies interactions with sensors and actuators and forms the basis for implementing application code executing in the processor.

2.1.1 Introduction:

The objective of the IEEE/NIST Working Group on transducer interface standards is to utilize existing control networking technology and develop standardized connection methods for Smart Transducers to control networks. Little or no changes would be required to use different methods of analog-to-digital (A/D) conversion, different microprocessors, or different network protocols and transceivers. This objective is achieved through the definition of a common object model for the components of a Networked Smart Transducer, together with interface specifications to these components. The Networked Smart Transducer model shows two key views of a smart transducer[1].

1. Physical view.
2. Logical view.

Figure 2.1 physical and logical views of IEEE 1451.1.

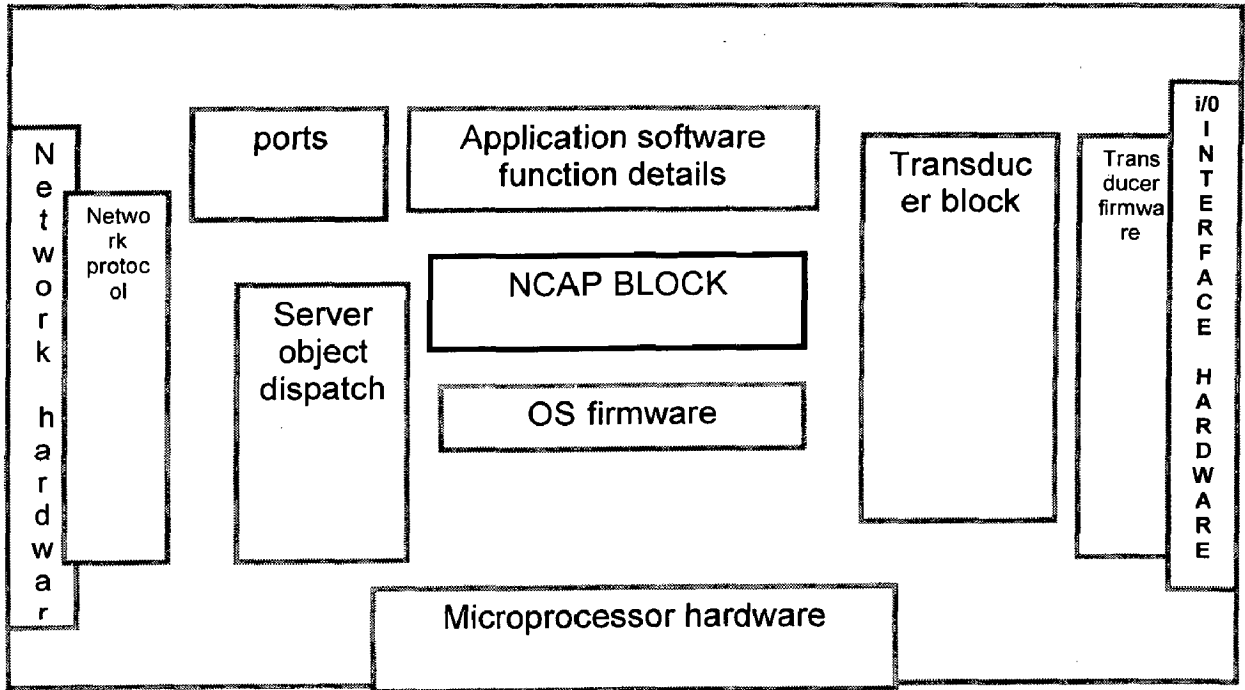


Figure 2.1 Network Capable Application Processor

2.1.2 Physical view:

The first view shows the physical components of the system. This view is indicated by components drawn in solid lines in Figure 2.1. Figure 2.1 shows a model composed of sensors and actuators connected to form a transducer. The transducer is connected over an interface to a microprocessor or controller that is in turn interfaced to the network. The Hardware Interface Specification between the sensor/actuator and the rest of the device hardware, known as the network capable application processor (NCAP), is indicated by the rightmost thick, dashed line in the figure. A typical specification is described in the companion standard [IEEE STD 1451.2-1997].

The NCAP hardware consists of the microprocessor and its supporting circuitry as well as hardware implementing the physical layer of the attached network and the input/output (I/O) interface to the transducer, as shown in Figure 2.1.

2.1.3 Logical view:

The second view is the logical view of the system and is indicated by components shown in dotted lines in Figure 2.1.

The logical components may be grouped into application and support components.

The support components are the

- **Operating system**

The operating system provides an interface to applications, indicated by the dashed line labeled “Logical Interface to NCAP support.”

- **The network protocol**

A second logical interface, labeled “Network Abstraction Logical Interface Specification,” consists of Port and Server Object Dispatch components defined in this standard. This interface provides an abstraction to hide communication details specific to a given network within a small set of communication methods. The details of this interface are defined by this standard.

- **Transducer firmware components**

The third logical interface, labeled “Transducer Abstraction Logical Interface specification” performs the same abstraction function for the specifics of the transducer I/O hardware and firmware. In effect, this interface makes all such transducer interfaces look like I/O drivers. The details of this interface are defined by this standard.

Applications are modeled as Function Blocks in combination with Components and Services. The NCAP block provides application organization and support for the other blocks. All of these Blocks, Components, and Services are defined by this standard.

These interfaces are optional in the sense that not all must be exposed in an implementation.

2.1.4 Information Model:

The IEEE 1451.1 standard specifies software architecture. This architecture is applicable to distributed systems consisting of one or more Network Capable Application Processors (NCAPs), communicating over a network. The NCAPs may interact with the physical world via attached transducers.

The standard provides

- A network abstraction layer
- A transducer abstraction layer

The standard specifies[1]

- The software interfaces between application functions on an NCAP and a communication network in a manner independent of any specific network.
- The software interfaces between application functions on an NCAP and transducers attached to that NCAP in a manner independent of any specific transducer driver interface.

Systems implemented according to IEEE 1451.1 standard will achieve a high degree of interoperability regardless of the underlying network or transducer technologies.

The IEEE 1451.1 software architecture is defined via three types of models[1]

- An object model (for the software components of IEEE 1451.1 systems)
- A data model (for the information communicated across the specified object interfaces)
- Two network communications models.

2.1.5 Data model:

The IEEE 1451.1 data model specifies the type and form of the information communicated across the IEEE 1451.1-specified Object interfaces in both local and remote communications. The model is realized in an implementation of IEEE 1451.1 as a collection of primitive data types and a collection of structure data types.

2.2 IEEE 1451.1 Goals:

“The specifications provide a comprehensive data model for the factory floor, and a simple application framework to build interoperable distributed applications.” Dr. Jay Warrior, Agilent Technologies, Chair IEEE 1451.1 WG.

In general, IEEE 1451.1 accomplishes this by providing:

- Transducer application portability (software reuse)
- Plug-and-play software capabilities (components)
- Network independence (network abstraction layer)

The standard specifies these capabilities by defining software interfaces for:

- Application functions in the NCAP that interact with the network that are Independent of any network

- Application functions in the NCAP that interact with the transducers that are Independent of any specific transducer driver interface

2.3 IEEE 1451.1 users:

There are three primary categories of users of the IEEE 1451.1 standard. These user categories are referred to throughout the remaining clauses of the standard.

The primary categories of users of the IEEE 1451.1 standard are

- System developers
- Component developers
- End users

2.3.1 System developers:

These are primarily manufacturers of

- NCAPs
- NCAP Block classes
- Transducer Block classes
- Other Object classes
- Network-specific infrastructure libraries

2.3.2 Component developers:

These are primarily manufacturers of reusable Function Block classes to be used as components in IEEE 1451.1 systems

2.3.3 End users:

These are primarily builders or installers of specific end-use Application Systems.

2.3.4 A minimal IEEE 1451.1 application consists [1]:

- An NCAP Block (consolidates system and communication housekeeping).
- A Transducer Block (provides the software connection to the transducer device).
- A Function Block (provides the transducer application algorithm (i.e., obtain and multicast temperature data every second)).
- Parameters (contains the network accessible variables that hold and update the data)

- Ports (network communication objects for publishing and subscribing to information or interacting with other NCAP s using client/server).

2.4 IEEE 1451.1 Benefits:

Using 1451.1 provides:

- An extensible object-oriented model for smart transducer application development and deployment
- Application portability achieved through agreed upon application programming interfaces (API)
- Network neutral interface allows the same application to be plug-and-play across multiple network technologies
- Leverages existing networking technology, does not re-implement any control network software or protocols
- A common software interface to transducer hardware i/o

2.5 Implementation of Network smart sensor:

2.5.1 Description of IEEE 1451.2 standard:

The dominant idea of IEEE1451.2 Standard is to turn intelligent transducer into separate unit, which makes it easier for transducers to install, uninstall and interchange. Plug and play transducers/sensors are achieved. In IEEE1451.2 Standard, a hardware model of network interface module is discussed, as fig 2.2

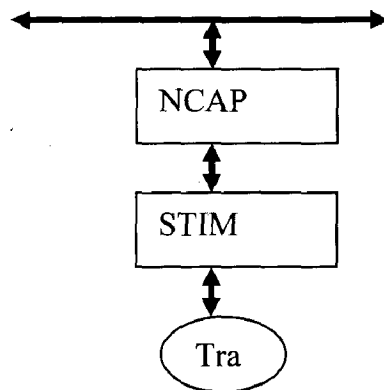


Fig 2.2 hardware model of network interface module

In the model, IEEE1451.2 Standard defines a Smart Transducer Interface Module, a ten-wire digital interface TII and NCAP. Information is collected by transducer, and then it is

Converted from analog to digital and becomes standard data format in STIM. A transducer electronic data sheet is also defined, which is called TEDS. The information, with the TEDS, is sent to the upper level through the ten-wire interface TII. The upper level is called NCAP in the standard.

This standard has two main characters:

a) **TEDS:** This is used for self-identification of transducers. Basic information about the transducer and the way to use them is stored in a non-volatile memory. This information includes the manufacturer, the number of channels, the magnitude measured by each of them, the data format, calibration or tolerance among others. All the information is read under demand from the upper level, and it is sent through the same TII interface [2]. Reading this information after start-up allows identifying new transducers without any other configuration or network redesign. This is necessary for true sensor-to-network interoperability.

Transducers based on IEEE1451.2 can convert different network through the standard interface TII, as Fig 2.3

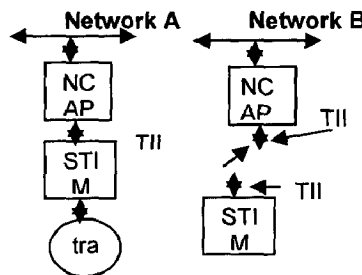


Fig 2.3 Conversion of 1451.2 models in different network.

b) **STIM:**

STIM module controls the data conversion, as ADCs and DACs. It also manages the TII interface, answering the NCAP requests. Besides, the STIM module is in charge of the TEDS, sending its data when demanded. The standard defines the different fields of data sheet and purposes. Some of them are mandatory, while others are optional. In addition, STIM module is described in state machine. There are three state-machines in STIM module: Main State-Machine, Data State-Machine and Trigger State-Machine. The Main State-Machine implements STIM behavior coordinating the other state machines, trigger

State-Machine operates on the physical channel while Data- State-Machine is dedicated to data transfer [9].

c) TII:

IEEE1451.2 proposes a ten-wire digital interface, which is called Transducer Independent Interface, and defines series of reading and writing functions. Functions of TII interface footprints are described in table 2.1.

Table 2.1 Transducer Independent Interface(TII).

DCLK	Driven by NCAP,data transfers are based on SPI
NTRIG	Driven by NCAP To initiate measurement
NTRACK	To acknowledge the requested function has been performed
NIO_INT	By STIM if any exception
Power supply	+5V for STIM and NCAP
DIN,DOUT	Data In, Data Out
Status Registers	To notify exceptions such as busy channels, calibration failure

2.5.2 NCAP

NCAP is a microcontroller module between STIM and network with part intelligent. STIM can connect with network through NCAP. NCAP also can calibrate the original data from STIM transducer. Its operation processes are not defined in the IEEE1451.2 standard. It just needs to comply with the TII electrical and timing requirements.

2.5.3 Hardware Implementation of Network Smart sensor:

A network sensor based on IEEE1451.2 is divided into four parts: field network, a network capable application processor, a Smart Transducer Interface Module and a ten wire digital interface connecting between STIM and NCAP. Network can select field Bus, Ethernet or Internet. In this work, CAN is chosen.

The whole structure is as in Fig. 2.4. XDCCR denotes transducer.

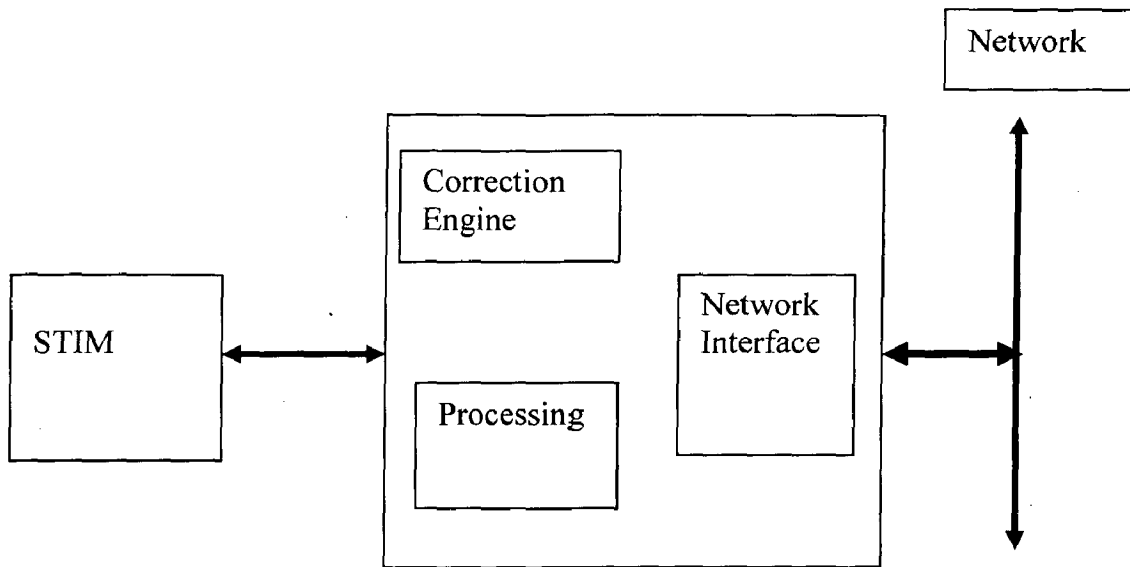


Fig 2.4 Whole structure of network sensor.

As stated in the previous section, a STIM module can connect up to 255 transducers (sensors or actuators).

2.5.4 NCAP for CAN BUS:

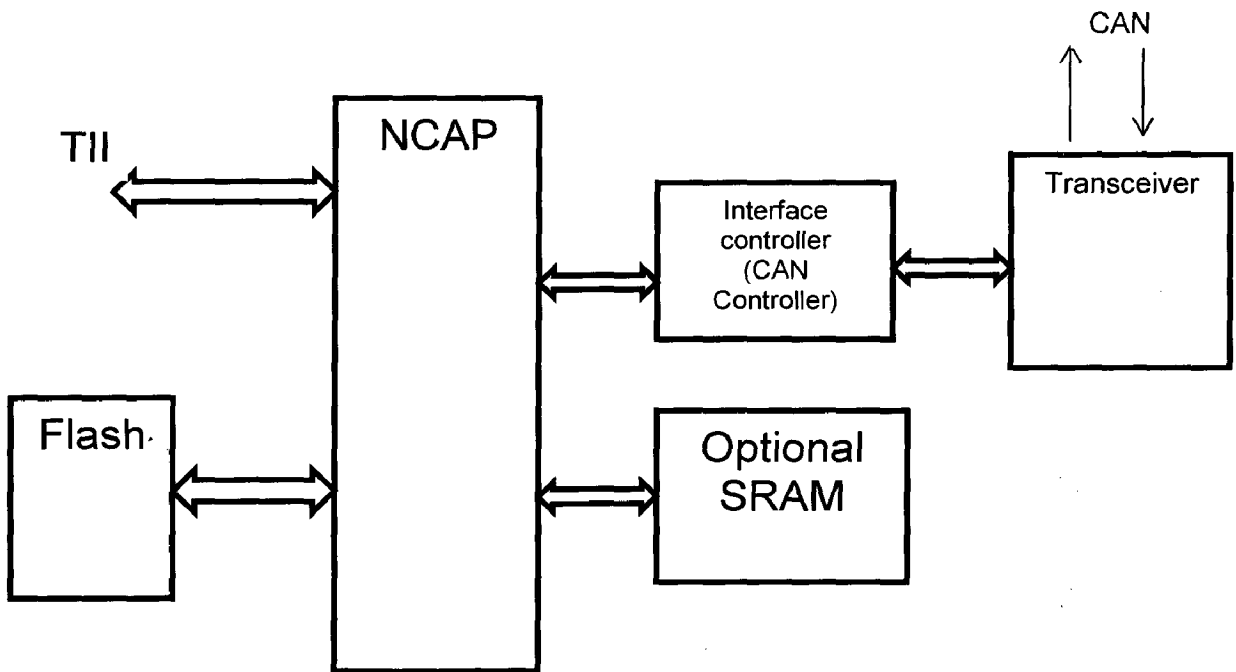


Fig 2.5 NCAP structure for CAN (source [5]).

As a part of development SoC for smart sensors, the CAN protocol controller is developed in HDL and complete design and implementation of interface controller is described in later chapters.

CHAPTER 3

Hardware and Software Platforms

In present work, soft wares were used for simulation and synthesis purposes. Hardware here is the FPGA spartan3E kit. Understanding FPGA architecture allows you to create HDL code that effectively uses FPGA system features.

3.1 Software platforms:

Software tools used are

1. Hardware Descriptive Language (HDL)
2. Xilinx ISE 9.2 I

This section discusses the above software details and provides a general overview of designing Field Programmable Gate Arrays (FPGA devices) with Hardware Description Languages (HDLs).

3.1.1 Hardware Description Language:

Designers use Hardware Description Languages (HDLs) to describe the behavior and structure of system and circuit designs.

Advantages of Using HDLs to Design FPGA Devices:

Using HDLs to design high-density FPGA devices has the following advantages:

- Top-Down Approach for Large Projects
- Functional Simulation Early in the Design Flow
- Synthesis of HDL Code to Gates
- Early Testing of Various Design Implementations
- Reuse of RTL Code

3.1.2 Designing FPGA Devices with Synthesis Tools:

Xilinx ISE 9.2i[14]:

Xilinx ISE stands for Xilinx Integrated System Environment (ISE). ISE controls all aspects of the design flow. Through the Project Navigator interface, one can access all of the design entry and design implementation tools. One can also access the files and documents associated with your project. Project Navigator maintains a flat directory structure; therefore, the project should be updated through the use of snapshots.

The Xilinx ISE] system is an integrated design environment that consists of a set of programs to create (capture), simulate and implement digital designs in a FPGA or CPLD target device. All the tools use a graphical user interface (GUI) that allows all programs to be executed from toolbars, menus or icons.

Most of the commonly-used FPGA synthesis tools have special optimization algorithms for Xilinx FPGA devices. Constraints and compiling options perform differently depending on the target device. Some commands and constraints in ASIC synthesis tools do not apply to FPGA devices. If you use them, they may adversely impact your results. You should understand how your synthesis tool processes designs before you create FPGA designs.

3.2 Design Flow [14]:

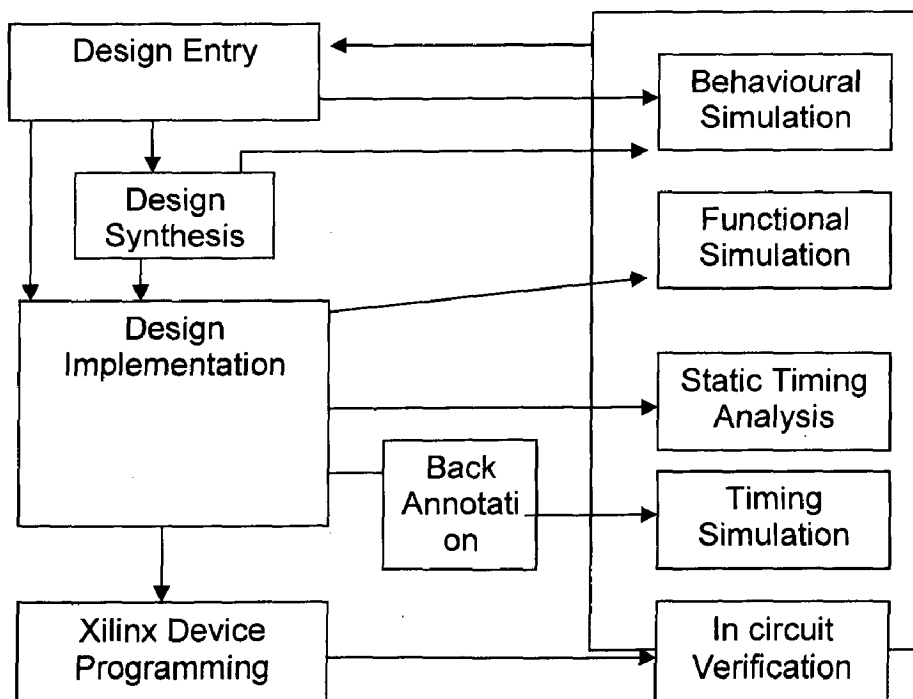


Figure 3.1 Design flow[14]

a) Functional Simulation

b) Synthesizing and Optimizing

It includes recommendations for compiling your designs to improve your results and decrease the run time.

c) Creating an Initialization File

Most synthesis tools provide a default initialization with default options. You may modify the initialization file or use the application to change compiler defaults, and to point to the applicable implementation libraries.

d) Placing and Routing

The overall goal when placing and routing your design is fast implementation and high-quality results. However, depending on the situation and your design, you may not always accomplish this goal, as described in the following examples.

- Earlier in the design cycle, run time is generally more important than the quality of results, and later in the design cycle, the converse is usually true.
- If the targeted device is highly utilized, the routing may become congested, and your design may be difficult to route. In this case, the placer and router may take longer to meet your timing requirements.
- If design constraints are rigorous, it may take longer to correctly place and route your design, and meet the specified timing.

3.2.1 Steps in Implementation[14]:

1. Create a New Project
2. Design Simulation
3. Create Timing Constraints
4. Implement Design and verify Constraints
5. Implementing the design
 - a. Assigning Pin location constraints
6. Download Design to Spartan 3E starter kit

These steps are clearly shown in the figures

Package pin locations:

Specify the pin locations for the ports of the design so that they are connected correctly on the Spartan-3E Startup Kit demo board.

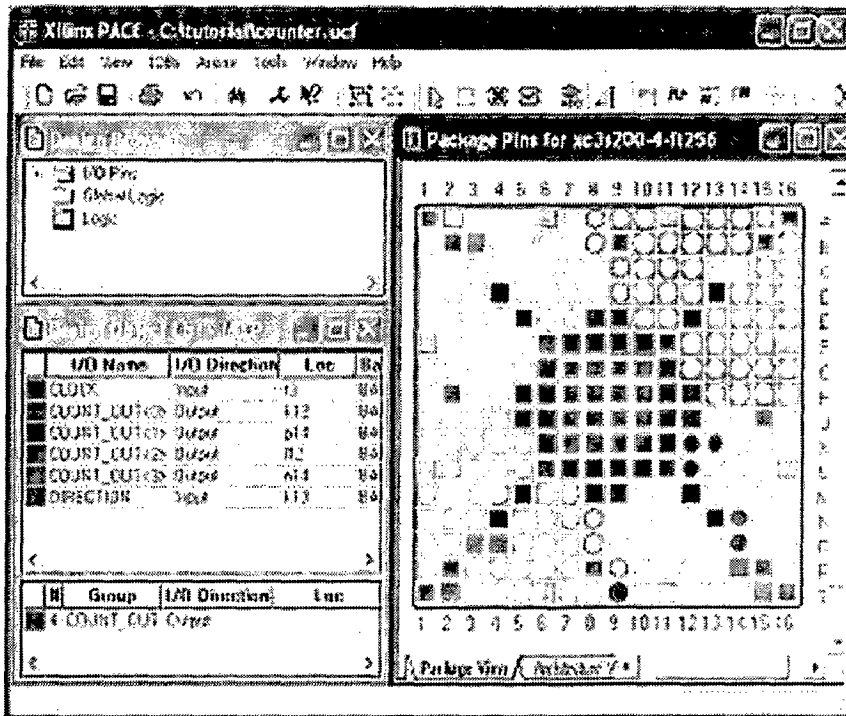


Fig 3.4 Assigning package pins[14].

Downloading design to the kit:

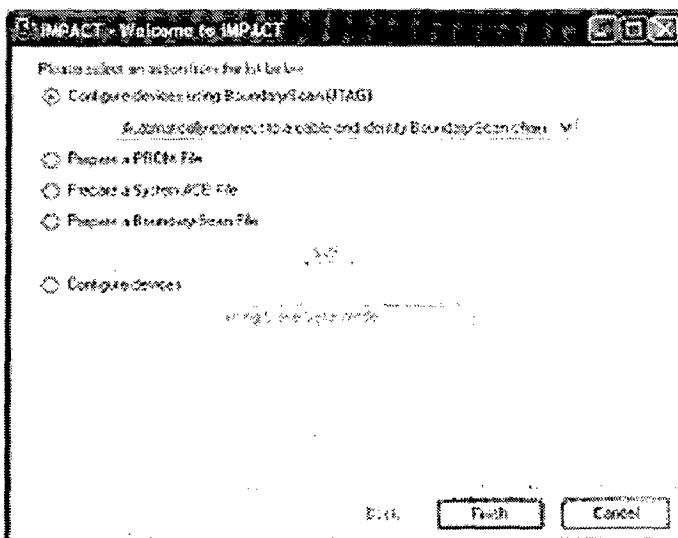


Fig 3.5 Impact welcome dialog box[14].

Spartan 3E FPGA starter kit:

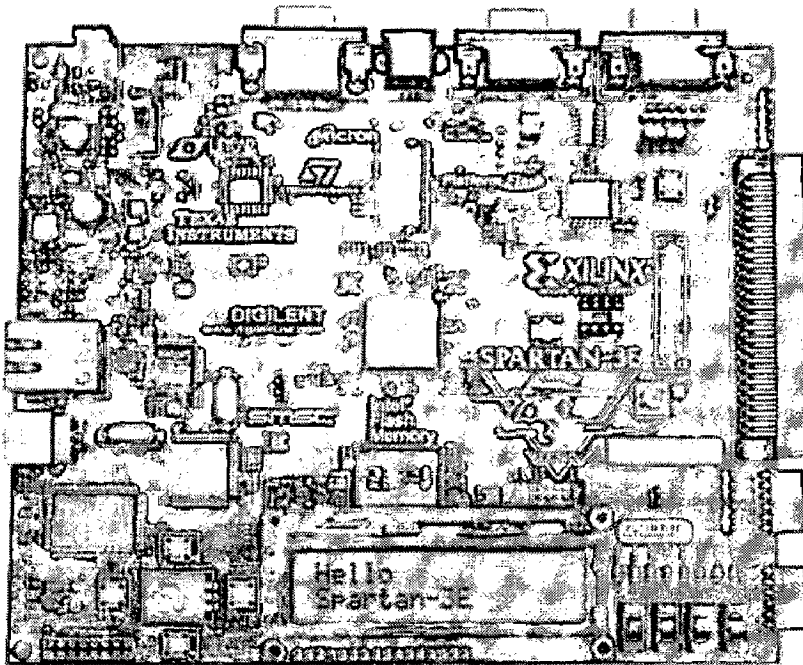


Fig 3.7 Spartan 3E FPGA kit[16]

Key Features[16]:

The key features of the Spartan-3E Starter Kit board are:

- Xilinx XC3S500E Spartan-3E FPGA
- Up to 232 user-I/O pins
- 320-pin FPGA package
- Over 10,000 logic cells
- Xilinx 4 Mbit Platform Flash configuration PROM
- Xilinx 64-macrocell XC2C64A CoolRunner CPLD
- 64 MByte (512 Mbit) of DDR SDRAM, x16 data interface, 100+ MHz
- 16 MByte (128 Mbit) of parallel NOR Flash (Intel StrataFlash)
- FPGA configuration storage
- MicroBlaze code storage/shadowing
- 16 Mbits of SPI serial Flash (STMicro)
- FPGA configuration storage
- MicroBlaze code shadowing

- 2-line, 16-character LCD screen
- PS/2 mouse or keyboard port
- VGA display port
- 10/100 Ethernet PHY (requires Ethernet MAC in FPGA)
- Two 9-pin RS-232 ports (DTE- and DCE-style)
- On-board USB-based FPGA/CPLD download/debug interface
- 50 MHz clock oscillator
- SHA-1 1-wire serial EEPROM for bitstream copy protection
- Hirose FX2 expansion connector
- Three Digilent 6-pin expansion connectors
- Four-output, SPI-based Digital-to-Analog Converter (DAC)
- Two-input, SPI-based Analog-to-Digital Converter (ADC) with programmable-gain
- pre-amplifier
- ChipScope™ SoftTouch debugging port
- Rotary-encoder with push-button shaft
- Eight discrete LEDs
- Four slide switches
- Four push-button switches
- SMA clock input
- 8-pin DIP socket for auxiliary clock oscillator

CHAPTER 4

Controller Area Network (CAN) protocol

4.1 CAN:

Controller Area Network (CAN) is a shared serial bus communication protocol, originally developed in 1986 by Robert Bosch GmbH. The increasing number of distributed control systems in cars and the increasing wiring costs of car body electronics led to the birth of the "Automotive Serial Controller Area Network" protocol[15]. Although initially developed for use in the automotive industry, its use quickly spread to a wide variety of embedded systems applications like industrial control where high-speed communication is required. With growing acceptance in various industries not necessarily related to the automotive industry, the protocol was renamed the Controller Area Network (CAN).

4.2 CAN network protocol:

The CAN communications protocol describes the method by which information is passed between devices.

It conforms to the Open Systems Interconnection model, which is defined in terms of layers. Each layer in a device apparently communicates with the same layer in another device. Actual communication is between adjacent layers in each device and the devices are only connected by the physical medium via the physical layer of the model[15].

In the basic CAN specification, it has a transmission rate of up to 250 KBaud while full CAN runs at 1 MBaud.

To achieve design transparency and implementation flexibility CAN has been subdivided into different layers.

- The (CAN) object layer
- The (CAN) transfer layer
- The physical layer

The object layer and the transfer layer comprise all services and functions of the data link layer defined by the ISO/OSI model. The scope of the object layer includes

- Finding which messages are to be transmitted
- Deciding which messages received by the transfer layer is actually to be used,
- Providing an interface to the application layer related hardware.

There is much freedom in defining object handling. The scope of the transfer layer mainly is the transfer protocol, i.e. controlling the framing, performing arbitration, error checking, and error signaling and fault confinement.

Within the transfer layer it is decided whether the bus is free for starting a new transmission or whether a reception is just starting.

Also some general features of the bit timing are regarded as part of the transfer layer. It is in the nature of the transfer layer that there is no freedom for modifications.

The scope of the physical layer is the actual transfer of the bits between the different Nodes with respect to all electrical properties. Within one network the physical layer, of Course, has to be the same for all nodes. There may be, however, much freedom in selecting a physical layer.

The scope of this specification is to define the transfer layer and the consequences of the CAN protocol on the surrounding layers.

Data Link Layer	LLC(Logic Link Layer) Acceptance filtering Overload Notification Recovery Management MAC(Medium Access Control) Data Encapsulation/Decapsulation Stuffing/Destuffing Bus Arbitration Error Detection Error Signaling Fault Confinement Acknowledgement Serialization/Deserialization
Physical Layer	PLS(Physical signaling) Bit Encoding/Decoding Bit Timing Synchronization

Figure 4.1 Layers of CAN

4.2.1 CAN properties

- Prioritization of messages
- Guarantee of latency times
- Configuration flexibility
- Multicast reception with time synchronization
- System wide data consistency
- Multimaster
- Error detection and signaling
- Automatic retransmission of corrupted messages as soon as the bus is idle again
- Distinction between temporary errors and permanent failures of nodes and Autonomous switching off of defect nodes

4.2.2 Layered Structure of a CAN Node

Object Layer

- Message Filtering
- Message and Status Handling

Transfer Layer

- Fault Confinement
- Error Detection and Signaling
- Message Validation
- Acknowledgment
- Arbitration
- Message Framing
- Transfer Rate and Timing

Physical Layer

- Signal Level and Bit Representation
- Transmission Medium

Application Layer

The Physical Layer defines how signals are actually transmitted. Within this specification the physical layer is not defined so as to allow transmission medium and signal level implementations to be optimized for their application.

- The Transfer Layer represents the kernel of the CAN protocol. It presents messages received to the object layer and accepts messages to be transmitted from the object layer. The transfer layer is responsible for bit timing and synchronization, message framing, arbitration, acknowledgment, error detection and signaling, and fault confinement.

The Object Layer is concerned with message filtering as well as status and Message handling. The scope of this specification is to define the transfer layer and the consequences of the CAN protocol on the surrounding layers.

4.3 Messages [15]:

Information on the bus is sent in fixed format messages of different but limited length. When the bus is free any connected unit may start to transmit a new message.

CAN have four frame types:

- **Data frame:** a frame containing node data for transmission
- **Remote frame:** a frame requesting the transmission of a specific identifier
- **Error frame:** a frame transmitted by any node detecting an error
- **Overload frame:** a frame to inject a delay between data and/or remote frames

4.3.1 Data frame

The data frame is the only frame for actual data transmission. There are two message formats:

- Base frame format: with 11 identifier bits
- Extended frame format: with 29 identifier bits

The CAN standard requires the implementation *must* accept the base frame format and *may* accept the extended frame format, but *must* tolerate the extended frame format.

Base frame format

The frame format is as follows:

Start-of-frame	1	Denotes the start of frame transmission
----------------	---	---

Identifier	11	A (unique) identifier for the data
Remote transmission request (RTR)	1	Must be dominant (0)Optional
Identifier extension bit (IDE)	1	Must be dominant (0)Optional
Reserved bit (r0)	1	Reserved bit (it must be set to dominant (0), but accepted as either dominant or recessive)
Data length code (DLC)	4	Number of bytes of data (0-8 bytes)
Data field	0-8 bytes	Data to be transmitted (length dictated by DLC field)
CRC	15	<u>Cyclic redundancy check</u>
CRC delimiter	1	Must be recessive (1)
ACK slot	1	Transmitter sends recessive (1) and any receiver can assert a dominant (0)
ACK delimiter	1	Must be recessive (1)
End-of-frame (EOF)	7	Must be recessive (1)

One restriction placed on the identifier is that the first 7 bits cannot be all recessive bits. (I.e., the 16 identifiers 1111111xxxx are invalid.)

a) Start of frame:

Marks the beginning of data frames and remote frames. It consists of a single 'dominant' bit. A station is only allowed to start transmission when the bus is idle. All stations have to synchronize to the leading edge caused by start of frame of the station starting transmission first.

b) Arbitration Field:

The arbitration field consists of the identifier and the RTR-bit identifier. The identifier's length is 11 bits. These bits are transmitted in the order from ID-10 to ID-0. The least significant bit is ID-0. The 7 most significant bits (ID-10 - ID-4) must not be all 'recessive'.

c) RTR bit

Remote Transmission Request BIT

In data frames the RTR BIT has to be 'dominant'. Within a remote frame the RTR BIT has to be 'recessive'.

d)Control Field:

The control field consists of six bits. It includes the data length code and two bits reserved for future expansion. The reserved bits have to be sent 'dominant'. Receivers accepts 'dominant' and 'recessive' bits in all combinations.

Data length code:

The number of bytes in the data field is indicated by the data length code.

This data length code is 4 bits wide and is transmitted within the control field.

e)Data frame:

Admissible numbers of data bytes:{0, 1... 7, 8}.

Other values may not be used.

Data field:

The data field consists of the data to be transferred within a data frame. It can

Table 4.1 data bytes for corresponding DLC

DLC	Number of bytes
1000	8
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

f) CRC Field:

CRC field Contains the CRC sequence followed by a CRC delimiter.

CRC sequence:

The frame check sequence is derived from a cyclic redundancy code best suited for frames with bit counts less than 127 bits (BCH Code).

In order to carry out the CRC calculation the polynomial to be divided is defined as the Polynomial, the coefficients of which are given by the de stuffed bit stream consisting of

start of frame, arbitration field, control field, data field(if present) and, for the 15 lowest coefficients, by 0. This polynomial is divided (the coefficients are calculated modulo-2) by the generator-polynomial:

$$X^{15} + X^{14} + X^{10} + X^8 + X^7 + X^4 + X^3 + 1.$$

The remainder of this polynomial division is the CRC sequence transmitted over the bus. In order to implement this function, a 15 bit shift register $CRC_rg(14:0)$ can be used.

After the transmission / reception of the last bit of the data field, CRC_rg contains the CRC sequence.

CRC delimiter: The CRC SEQUENCE is followed by the CRC DELIMITER which consists of a single 'recessive' bit.

g) ACK field:

The ack field is two bits long and contains the ack slot and the ack delimiter. In the ack field the transmitting station sends two 'recessive' bits. A receiver which has received a valid message correctly, reports this to the transmitter by sending a 'dominant' bit during the ack slot (it sends 'ACK').

ACK slot :

All stations having received the matching CRC sequence report this within the ACK slot by super scribing the 'recessive' bit of the transmitter by a 'dominant' bit.

ACK Delimiter:

The ACK Delimiter is the second bit of the ACK field and has to be a 'recessive' bit. As a consequence, the ACK slot is surrounded by two 'recessive' bits (CRC Delimiter, ACK Delimiter).

h) End of Frame:

Each data frame and remote frame is delimited by a flag sequence consisting of seven 'recessive' bits.

4.3.2 Remote frame

Generally data transmission is performed on an autonomous basis with the data source node (e.g. a sensor) sending out a Data Frame. It is also possible, however, for a

destination node to request the data from the source by sending a Remote Frame. •There are 2 differences between a Data Frame and a Remote Frame. Firstly the RTR-bit is transmitted as a dominant bit in the Data Frame and secondly in the Remote Frame there is no Data Field.

i.e.

RTR = 0; DOMINANT in data frame

RTR = 1; RECESSIVE in remote frame

In the very unlikely event of a Data Frame and a Remote Frame with the same identifier being transmitted at the same time, the Data Frame wins arbitration due to the dominant RTR bit following the identifier. In this way, the node that transmitted the Remote Frame receives the desired data immediately.

4.3.3 Error frame

Error frame consists of two different fields. The first field is given by the superposition of Error flags contributed from different stations. The following second field is the Error Delimiter

There are two types of error flags

Active Error Flag: Transmitted by a node detecting an error on the network that is in error state "error active".

Passive Error Flag: Transmitted by a node detecting an active error frame on the network that is in error state "error passive".

4.3.4 Overload frame:

The overload frame contains the two bit fields Overload Flag and Overload Delimiter. There are two kinds of overload conditions that can lead to the transmission of an overload flag:

1. The internal conditions of a receiver, which requires a delay of the next data frame or remote frame.
2. Detection of a dominant bit during intermission.

The start of an overload frame due to case 1 is only allowed to be started at the first bit time of an expected intermission, whereas overload frames due to case 2 start one bit after detecting the dominant bit. Overload Flag consists of six dominant bits. The overall form corresponds to that of the active error flag. The overload flag's form destroys the fixed form of the intermission field. As a consequence, all other stations also detect an overload condition and on their part start transmission of an overload flag. Overload Delimiter consists of eight recessive bits. The overload delimiter is of the same form as the error delimiter.

4.4 Bit stuffing:

In CAN frames, a bit of opposite polarity is inserted after five consecutive bits of the same polarity. This practice is called bit stuffing, and is due to the "Non Return to Zero" (NRZ) coding adopted. The "stuffed" data frames are destuffed by the receiver. Since bit stuffing is used, six consecutive bits of the same type (111111 or 000000) are considered an error. Bit stuffing implies that sent data frames could be larger than one would expect by simply enumerating the bits shown in the tables above.

4.5 Error Detection:

For detecting errors the following measures have been taken:

- Monitoring (transmitters compare the bit levels to be transmitted with the bit Levels detected on the bus)
- Cyclic Redundancy Check
- Bit Stuffing
- Message Frame Check

Performance of Error Detection

The error detection mechanisms have the following properties:

- All global errors are detected.
- All local errors at transmitters are detected.
- Up to 5 randomly distributed errors in a message are detected.
- Burst errors of length less than 15 in a message are detected.
- Errors of any odd number in a message are detected.

4.6 Error Signaling and Recovery Time:

Corrupted messages are flagged by any node detecting an error. Such messages are aborted and will be retransmitted automatically. The recovery time from detecting an Error until the start of the next message is at most 29 bit times, if there is no further error.

4.7 Fault Confinement:

CAN nodes are able to distinguish short disturbances from permanent failures. Defective nodes are switched off.

4.8 Connections:

The CAN serial communication link is a bus to which a number of units may be connected. This number has no theoretical limit. Practically the total number of units will be limited by delay times and/or electrical loads on the bus line.

4.9 Single Channel:

The bus consists of a single channel that carries bits. From this data resynchronization information can be derived. The way in which this channel is implemented is not fixed in this specification. E.g. single wire (plus ground), two differential wires, optical fibres, etc.

4.10 Bus values:

The bus can have one of two complementary logical values: 'dominant' or 'recessive'. During simultaneous transmission of 'dominant' and 'recessive' bits, the resulting bus Value will be 'dominant'. For example, in case of a wired-AND implementation of the bus, the 'dominant' level would be represented by a logical '0' and the 'recessive' level by a logical '1'. Physical states (e.g. electrical voltage, light) that represent the logical Levels are not given in this specification.

4.11 Acknowledgment

All receivers check the consistency of the message being received and will acknowledge a consistent message and flag an inconsistent message.

4.12 Error Handling:

Error Detection

There are 5 different error types (which are not mutually exclusive):

- **Bit Error:**

A unit that is sending a bit on the bus also monitors the bus. A bit error has to be detected at that bit time, when the bit value that is monitored is different from the bit value that is sent. An exception is the sending of a 'recessive' bit during the stuffed bit stream of the arbitration field or during the ack slot. Then no bit error occurs when a 'dominant' bit is monitored. A transmitter sending passive error flag and detecting a 'dominant' bit does not interpret this as a bit error.

- **Stuff Error:**

A stuff error has to be detected at the bit time of the 6th consecutive equal bit level in a message field that should be coded by the method of bit stuffing.

- **CRC Error:**

The CRC sequence consists of the result of the CRC calculation by the transmitter. The receivers calculate the CRC in the same way as the transmitter. A CRC error has to be detected, if the calculated result is not the same as that received in the CRC sequence.

- **Form Error:**

A form error has to be detected when a fixed-form bit field contains one or more illegal bits.

- **Acknowledgment error:**

An acknowledgment error has to be detected by a transmitter whenever it does not monitor a 'dominant' bit during the ack slot.

4.13 Error Signaling:

A station detecting an error condition signals this by transmitting an error flag. For an 'error active' node it is an active error flag, for an 'error passive' node it is a passive error flag. Whenever a bit error, a stuff error, a form error or an acknowledgment error is detected by any station, transmission of an error flag is started at the respective station at the next bit.

Whenever a CRC error is detected, transmission of an error flag starts at the bit following the ack delimiter, unless an error flag for another condition has already been started.

4.14 Definition of transmitter/Receiver:

Transmitter:

A unit originating a message is called “transmitter” of that message. The unit stays transmitter until the bus is idle or the unit loses arbitration.

The message is valid for the transmitter, if there is no error until the end of end of frame. If a message is corrupted, retransmission will follow automatically and according to prioritization. In order to be able to compete for bus access with other messages, retransmission has to start as soon as the bus is idle [19].

Receiver:

A unit is called “receiver” of a message, if it is not transmitter of that message and the bus is not idle. The message is valid for the receivers, if there is no error until the last but one bit of End of frame[19].

CHAPTER 5

Design and Implementation of RIC

5.1 Reprogrammability Requirement:

Complex data processing algorithms in industrial application require substantial multiprocessor and hierarchical systems. Each processor in such a system executes the set of functions which are connected both with data processing and communications with other processors.

The IEEE-1451 set of standards which regulate the development of the smart systems are a very complete and comprehensive set of standards. An IEEE-1451-based smart system consists of:

- a) Lower level modules – STIM, TBIM, Mixed-Mode Transducer, Wireless Transducer, which are connected to sensors and actuators.
- b) Network Capable Application Processors (NCAPs) which form the intermediate (middle) level for current data processing and service of low level modules (sensor and actuator polling)
- c) The information customers who are connected to the upper network (some NCAPs can also be considered information customers).

The IEEE-1451 standards regulate the lower level interface of a NCAP, but do not regulate the higher level interface. As a consequence, the known NCAPs have varying upper network interfaces and this fact decreases their universality.

Thus, universal NCAP should support the following [4]:

1. A wide set of standard communication interfaces, including high performance ones.
2. The ability to reconfigure the types and parameters of the communication interfaces according to meet user requirements.
3. The ability to remotely change NCAP software in on-line mode.

It is clear that the required flexibility of our ideal NCAP can only be achieved through software. However, this software must have a very fast hardware base to be effective.

5.2 Design of IEEE 1451 based smart module for Network Communications:

The IEEE 1451 based smart module for a specific network communications is shown in figure 5.1.

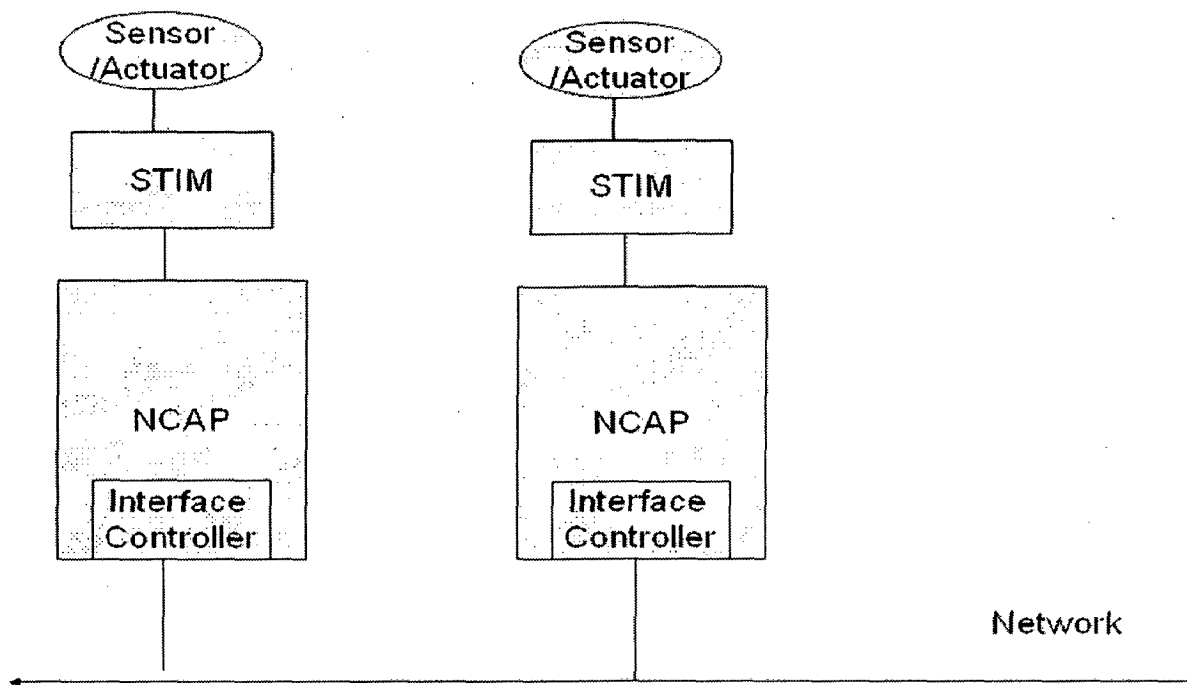


Fig 5.1 IEEE 1451 based smart module for a network

Considering the application areas of most of the sensors (like fire detector, glass break detector) CAN (Controller area Network) bus is selected as field network.

CAN bus have some merits such as low cost, high security and better anti jamming. It has been used on large scale in vehicle systems. CAN (Controller Area Network) is a standard protocol for sensor networks.

To implement an IEEE 1451 based smart module for CAN communications a suitable interface controller is designed and implemented on FPGA. These details are described in later sections.

The interfacing of NCAP and designed RIC is shown in figure 5.2.

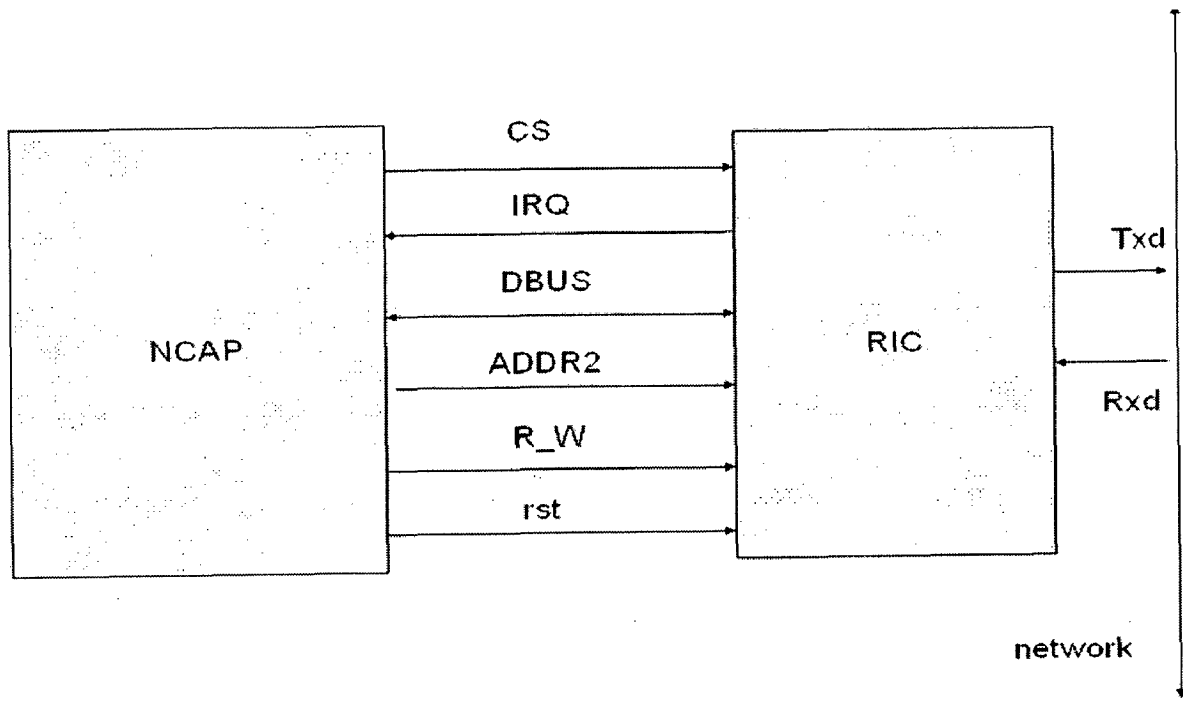


Fig 5.2 NCAP and CAN bus interface

5.3 Functional Description:

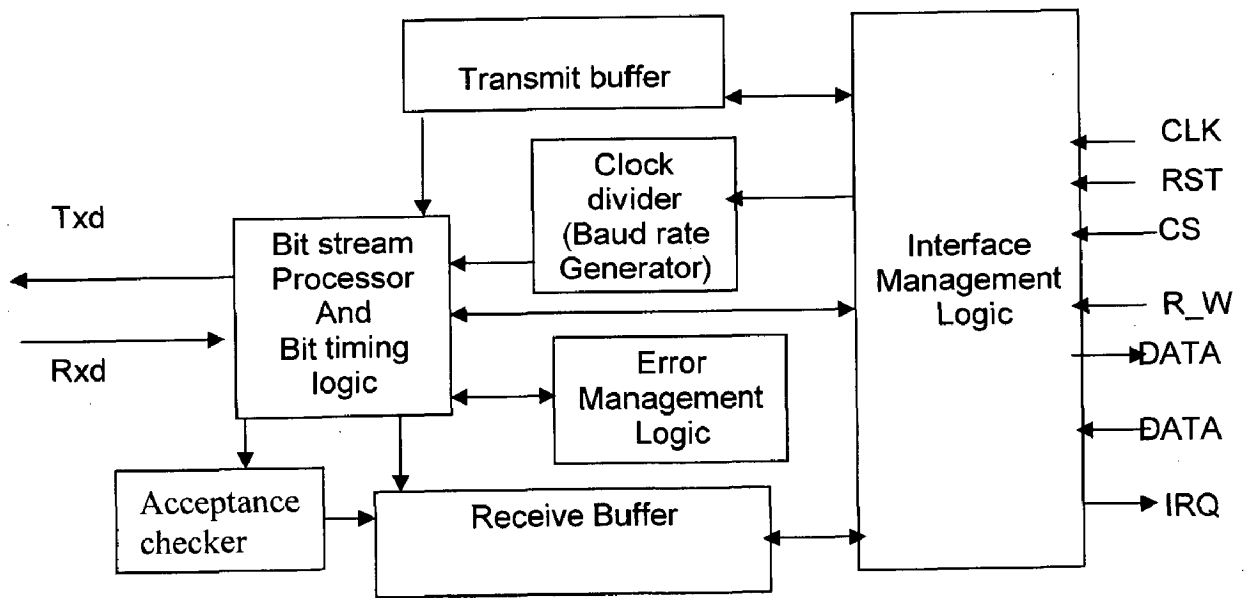


Fig 5.3 Blocks of Interface controller

Description of the controller blocks

5.3.1 Interface Management logic (IML):

The interface management logic interprets commands from the NCAP, controls addressing of the registers and provides interrupts and status information to the host processor.

5.3.2 Transmit Buffer (TXB):

The transmit buffer is an interface between the NCAP and the Bit Stream Processor (BSP) that is able to store a complete message for transmission over the CAN network. The buffer is 13 bytes long, written to by the host processor (NCAP) and read out by the bit stream processor.

5.3.3 Receive Buffer (RXB):

The receive buffer is an interface between the acceptance filter and the NCAP(CPU) that stores the received and accepted messages from the CAN-bus line. The Receive Buffer (RXB) represents a CPU-accessible 13-byte window of the Receive FIFO (RXFIFO). With the help of this FIFO the CPU is able to process one message while other messages are being received.

5.3.4 Acceptance Checker (AC):

The acceptance checker compares the received identifier with the acceptance checker register contents and decides whether this message should be accepted or not.

In the event of a positive acceptance test, the complete message is stored in the RX buffer.

5.3.5 Bit Stream Processor (BSP):

The bit stream processor is a sequencer which controls the data stream between the transmit buffer, RXFIFO and the CAN-bus. It also performs the error detection, arbitration, stuffing and error handling on the CAN-bus.

5.3.6 Bit Timing Logic (BTL):

The bit timing logic monitors the serial CAN-bus line and handles the bus line-related bit timing. It is synchronized to the bit stream on the CAN-bus on a 'recessive-to-dominant' bus line transition at the beginning of a message (hard synchronization) and re-synchronized on further transitions during the reception of a message (soft synchronization). The BTL also provides programmable time segments to compensate for the propagation delay times and phase shifts (e.g. due to oscillator drifts) and to define the sample point and the number of samples to be taken within a bit time.

5.3.7 Error Management Logic (EML):

The EML is responsible for the error confinement of the transfer-layer modules. It receives error announcements from the BSP and then informs the BSP and IML about error statistics.

5.4 Implementation and State Machine diagrams:

The CAN Protocol Controller receives unformatted message from the microprocessor, frames the messages as per the protocol specifications and also de-frames the received CAN message frames. The digital signals transmitted by the protocol controller are converted into electrical signals compatible with the CAN differential transmission medium by the CAN Transceiver which can also be designed as a separate entity.

The integration of these individual blocks would constitute the entire CAN interface Controller.

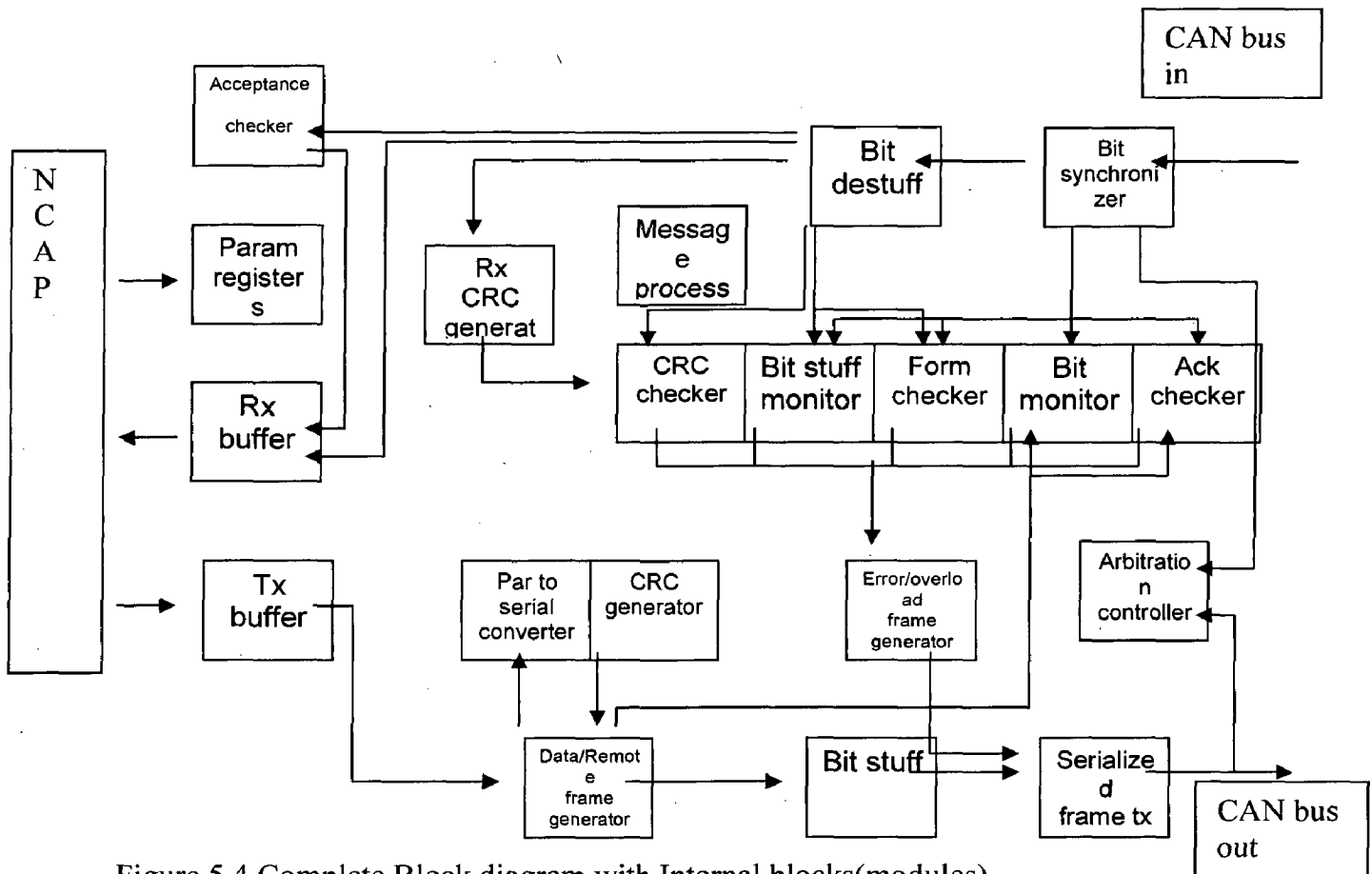


Figure 5.4 Complete Block diagram with Internal blocks(modules).

The various functional blocks in the diagram are described as follows:

- **NCAP:** This is the interfacing application which provides the CAN controller with the data to be transmitted across the CAN bus and also reads the received messages from the controller
- **Parameter Registers:** The control field register and identifier register specified for the CAN node.
- **TX Buffers:** There are thirteen transmit buffers. Each buffer can hold one byte of data. The controller receives the message to be transmitted from the host CPU and stores the message in the transmit buffer before further message processing takes place.
- **Data / Remote Frame Generator:** Data / Remote Frame Generator is responsible for generating the message frame as specified by the CAN protocol.
- **Par-Ser Converter:** This unit serializes the message to facilitate the CRC computation.

- **TX CRC Generator:** Before transmission, this unit computes the CRC for the message to be transmitted. The generated CRC frame is appended to the message being transmitted before bit-stuffing is performed.
- **Bit Stuff Unit:** This unit performs bit-stuffing as specified by the CAN protocol, making the message suitable for transmission across the CAN network.
- **Overload / Error Frame Generator:** Generates Error or Overload frame whenever error or overload condition occurs.
- **Serialized Frame Transmitter:** This unit transmits the data/ remote frame or the error / overload frame or a dominant bit during the acknowledgment slot based on the prevalent conditions.
- **Message Processor:** This is the central unit which provides all the control and the status signals to the various other blocks in the controller. This unit routes the different signals generated in various blocks to the necessary target blocks.
- **Arbitration Controller:** The arbitration controller is responsible for indicating the arbitration status of the node.
- **Synchronizer:** This unit performs the bit timing logic necessary for synchronizing the CAN controller to the bit stream on the CAN bus. The recessive to dominant transition edges present on the received bit stream are used for synchronization and re-synchronization.
- **Bit De-stuff Unit:** This unit performs the de-stuffing of the messages received from the CAN network. This unit also extracts the relevant information from the received message.
- **RX CRC Generator:** After reception, this unit computes the CRC for the message received.
- **Cyclic Redundancy Checker:** This unit compares the generated CRC for the received message with the CRC frame received by the node. An error is generated if the two CRC values do not match.
- **Bit Stuff Monitor:** This unit signals a stuff error when six consecutive bits of equal polarity is detected in the received message.

- **Form Checker:** A form error is generated if any of the fixed-form fields in a received CAN message is violated. The fixed form fields include the CRC delimiter, ACK delimiter and the EOF field.
- **Bit Monitor:** A CAN node acting as the transmitter of a message, samples back the bit from the CAN bus after putting out its own bit. If the bit transmitted and the bit sampled by the transmitter are not the same, a bit error is generated.
- **Acknowledgment Checker:** During the transmission of the acknowledgement slot a transmitter transmits a recessive bit and expects to receive a dominant bit. If the node receives a recessive bit in the acknowledgement slot an ACK error is signaled.
- **Acceptance Checker:** This unit checks the incoming message ID and determines if the received frame is valid.
- **Receive Buffer:** There are two 13 byte buffers that are used alternatively to store the messages received from the CAN bus. This enables the host CPU to process a message while another message is being received by the controller.

Building Blocks of the CAN Controller:

Each block of the CAN controller performs a specific operation. The functionality of the basic building blocks of the CAN Controller along with its operation is described below.

5.4.1 Parameter Registers:

The controller receives the control field register, the identifier register and specified for the CAN node from the host CPU and stores them in the parameter registers. The content of the control field , identifier registers are used to determine the acceptance of the message. The Synchronous Jump Width register content is used for bit synchronization. The state diagram for the Parameter Registers is shown in Fig 5.5

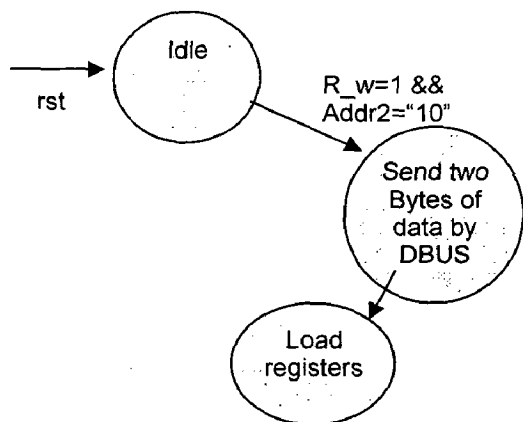


Figure 5.5 State diagram for registers.

Initially the identifier and control field need to be loaded into the CAN controller. The $R_w='1' ADDR2= "10"$ is asserted by the CPU when it needs to load new values in registers. This indicates to the CAN controller that the CPU wishes to load the parameters into the corresponding registers.

Following the high assertion of the $R_w='1'$ signal the CPU proceeds to send the data on the 8 bit *DBUS* in bytes. The 8 least significant bits of the identifier register are transferred first. This is followed by the transfer of the 8 bit data formed by the concatenation of the 5 least significant bits of the control field register and the 3 most significant bits of the identifier register.

Once the transmission is complete the registers retain the values stored in them. A new value is loaded only when the CPU initializes another parameter load by asserting the $R_w='1' ADDR2= "10"$ signals. A global reset to the system removes the parameters stored in these registers.

5.4.2 Transmitter Buffer:

There are thirteen transmit buffers. Each buffer can hold one byte of data. The controller receives the message to be transmitted from the host CPU and stores the message in the transmit buffer before further message processing takes place.

The CPU sends the message in the order of the message identifier first, followed by the control bits, and then the data bytes with the most significant byte of the data being sent first. The signal *loadmsg* goes high when the controller completes loading the transmit buffer and stays high till the message has been transmitted successfully. On

successful transmission the buffers are reset, the *loadmsg* signal goes low and the controller is ready to receive the next message.

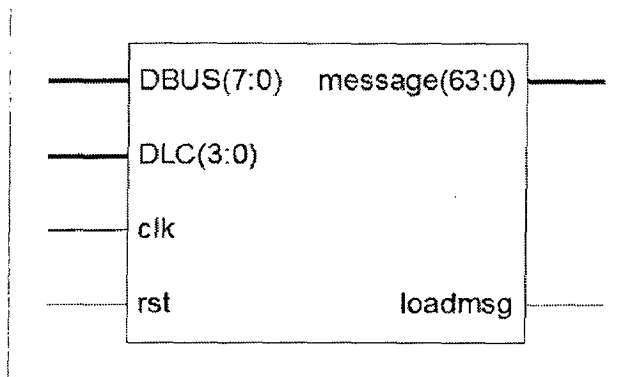


Figure 5.6 Synthesized RTL diagram of Tx buffer and *loadmsg* signal are shown

5.4.3 Data / Remote Frame Generation:

Data / Remote Frame Generator is responsible for generating the message frame as specified by the CAN protocol. The state diagram for the Data / Remote Frame Generation is shown in Fig. 5.7

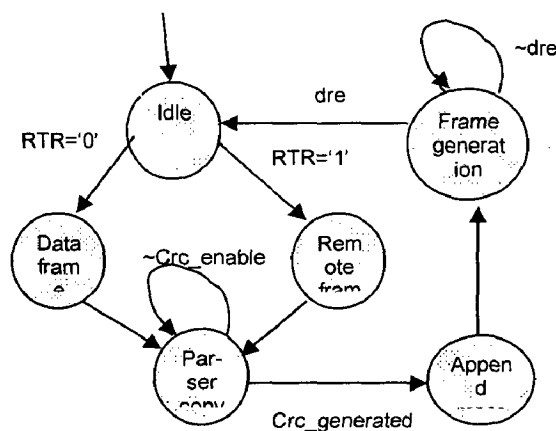


Fig. 5.7 CAN Data / Remote Frame Generation

On loading the final transmit buffer the data / remote frame generation is initialized.. Based on the Data Length Code (DLC) and the Remote Transfer Request (RTR) bit the *par_ser_data* frame is generated. If the RTR bit is recessive, the message to be transmitted is a Remote. In this case the *par_ser_data* frame does not have any Data Field and will be formed by the concatenation of the dominant Start Bit, the Message Identifier and the Control Field.

In case the RTR bit is dominant, the message to be transmitted is a Data Frame and hence will contain a Data Field. In this case the `par_ser_data` frame is formed by the concatenation of the dominant Start Bit, the Message Identifier, the Control Field and the Data Field.

The Data Field is of variable length given by the DLC. The Data Field can contain zero to eight bytes of data. The data is transmitted with the MSB first.

In case of a Remote Frame or a Data Frame, with DLC less than 8 the frame is appended with dominant bits to counter for the trailing bits which are not defined by the message. The `par_ser_data` frame is serialized using a Parallel to Series converter and fed as input to a CRC generator. The high assertion of the `crc_generated` signal indicates the completion of the CRC calculation.

The generated CRC frame is then appended to the end of the Data Field in a Data Frame or to the end of the Control Field in case of a Remote Frame. The message is appended with recessive bits to counter for the trailing bits which are not defined by the message. The message frame generated after appending the CRC frame is the *transmessage*. The transmessage is then bit stuffed before transmission.

5.4.4 Par-Ser Converter:

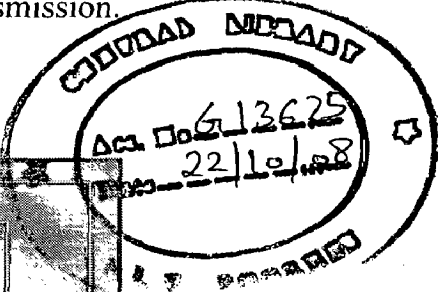
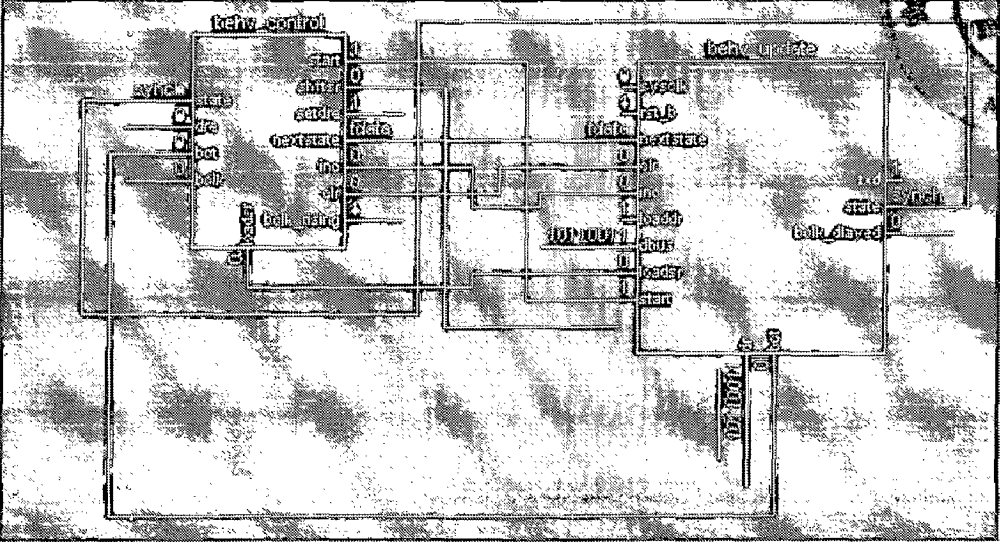


Figure 5.8 Dataflow diagram of parallel to serial converter.

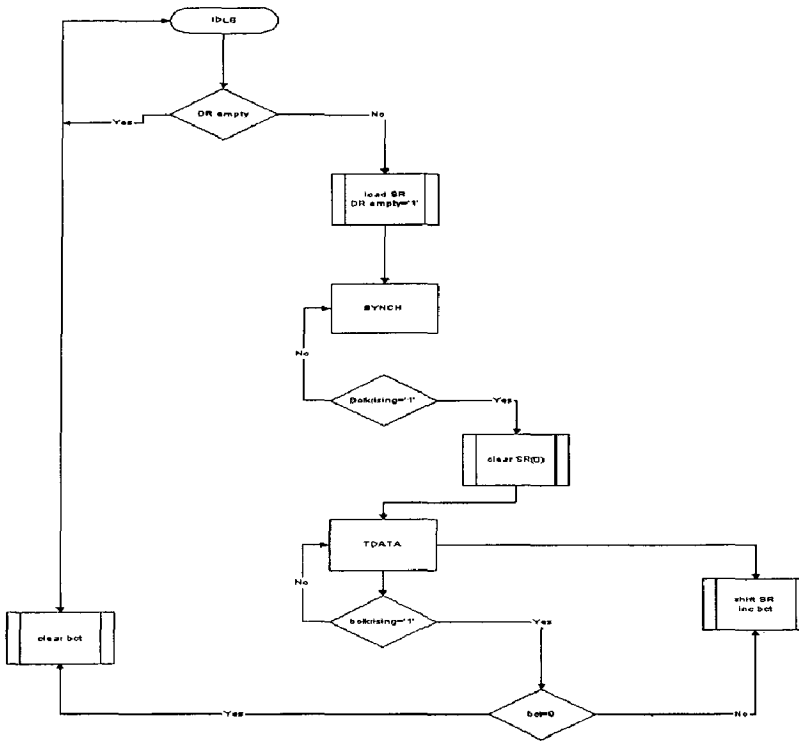


Fig. 5.9 State diagram Parallel – Serial Conversion

Consists of a data register DR and Shift Register SR and the transmit control.

It interfaces with DRE data register empty and the data bus (DBUS). The first process represents the combinational network, which generates the next state control signals. The second process updates the registers on the rising edge of the clock.

In the Idle state the state machine(SM) waits until DR has been loaded and DRE is cleared. In the SYNCH state, the SM waits for the rising edge of clock and then clears the lower order bit of SR to transmit '0' for one bit time.

In the TDATA state, each time rising edge of clock is detected SR shifted right to transmit the next data bit and the bit counter is incremented This unit serializes the message to facilitate the CRC computation. The state diagram for the Par-Ser Converter is shown in Fig. 5.9

5.4.5 CRC Generator:

For the input serial data the 15 bit CRC is calculated and appended to the data message while transmitting. While receiving acts like a crc checker The CRC frame calculation commences with the high assertion of the CRC *enable* signal. The CRC frame is

initialized to fifteen zeros with the *CRC initialize* signal. In order to carry out the CRC calculation the polynomial to be divided is defined as the Polynomial, the coefficients of which are given by the de stuffed bit stream consisting of start of frame, arbitration field, control field, data field(if present) and, for the 15 lowest coefficients, by 0. This polynomial is divided (the coefficients are calculated modulo-2) by the generator-polynomial:

$$X^{15} + X^{14} + X^{10} + X^8 + X^7 + X^4 + X^3 + 1.$$

The remainder of this polynomial division is the *CRC* sequence transmitted over the bus. In order to implement this function, a 15 bit shift register *CRC (14:0)* can be used.

After the transmission / reception of the last bit of the data field, *CRC (14:0)* contains the *CRC* sequence.

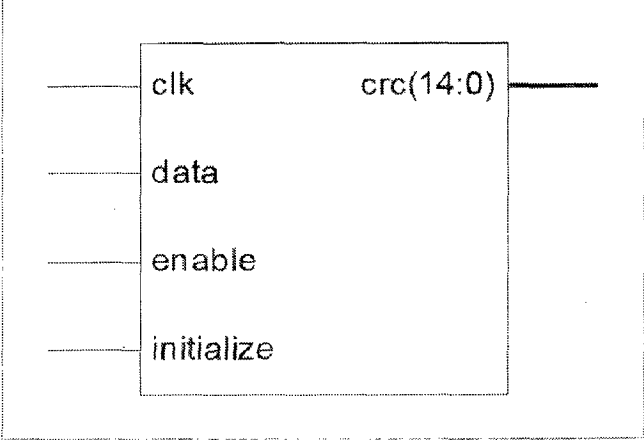


Figure 5.10 CRC(14:0) out synthesized RTL.

5.4.6 Bit Stuff Unit:

This unit performs the bit-stuffing mechanism as specified by the CAN protocol, making the message suitable for transmission across the CAN network. As per specifications, bit stuffing is done only on Data or Remote Frames. The input to the stuffing unit is the 98 bit *transmessage*. The concatenated 98 bit register, *transmessage* contains the input to the stuffing input and the bits are in the order [97:0] == [Start bit, messageidentifier [10:0], RTR bit, Control Field [5:0], Data Field [MSB:LSB], CRC Frame[14:0]].

The register may not always be full due to the variation in the data length, to ensure that the remaining bits do not contain junk data they are padded with recessive bits, 1's in this case.

The bit stuffing is initialized by the *bit_stuf* signal that goes high on appending the CRC frame to the message and stays high for one clock cycle. To bit stuff the message stream, the message stream is serialized and then checked for the bit stuffing condition. The stuffed bit stream being put out on the CAN bus has the Start Bit being transmitted first, followed by the message identifier's most significant bit and so on .

On receiving the *bit_stuf* the message stream to be stuffed, *transmessage* is stored in a temporary register *transmsg*. Then the 98th bit of *transmsg* is checked during each clock cycle. If it is a 1, a counter variable for one, one's count is incremented or if it is a 0, a counter variable for zero_count is incremented. The states machine stays in state outputs logic 1, if the sequence of ones is less than five. If the sequence of ones is equal to five then it enters next state and outputs a logic 1, the next state would be the zero stuff state as the protocol requires a stuff bit of opposite polarity to be transmitted after every sequence of five similar bits. In the zero stuff state the output is logic 0. Similarly when a sequence of five zero's is detected the state machine enters the one stuff state and outputs a logic 1.

As long as the sequence of ones or zeros is less than five the contents of *transmsg* are shifted by 1 to the left, thus discarding the bit already transmitted. The bit count is also incremented. However if there is a sequence of five consecutive ones or zeros then the *transmsg* register remains the same, without being shifted. The bit count is also not incremented and remains the same. The stuffing process is stopped when the bit count equals the length of the frame to be transmitted given by DLC.

The CRC Delimiter, ACK slot, ACK delimiter and EOF bits are appended to the bit stuffed message to form the Data / Remote Frame; these bits are transmitted as recessive bits.

5.4.7 Overload / Error Frame Generation:

On detecting an overload or error condition, an Overload Frame or an Error Frame is transmitted. A message received when both the buffers are full cannot be stored

in the receive buffers and will be lost. To avoid this situation an Overload Frame is generated by the receiver to indicate an overload condition to the other participating nodes. The transmission of the next message on the Bus is delayed by the transmission of the Overload Frame.

Similarly when an error is detected the node sends out an Error Frame. The transmission of the Error Frame produces an error condition in the other participating nodes and causes the message to be retransmitted.

The Error Confinement process is also taken care of by the Overload / Error Frame Generation unit. It is designed as per the specifications in the Data Link Layer of the CAN protocol.

5.4.7.1 Overload Frame:

The overload signaling and overload frame generation is demonstrated in Fig. 5.11. As discussed earlier in the Receive Buffer Storage section the rx_buff_0_wrtn and rx_buff_1_wrtn signals indicate that the buffers have been written into when they are asserted high. When both the buffers are loaded the node asserts the over_ld flag high to indicate an overload condition. The high assertion of the over_ld signal initiates the transmission of an Overload Frame. This is indicated by the ovid_flg_tx flag going high. The other nodes in the network detect a stuff error in the arbitration field and send out their respective Error Flag. When a stuff error is detected in the arbitration field the error counters are not incremented. Thus the overload flag delays the transmission of the next frame giving it sufficient time for the host controller to read the data from the receive buffers

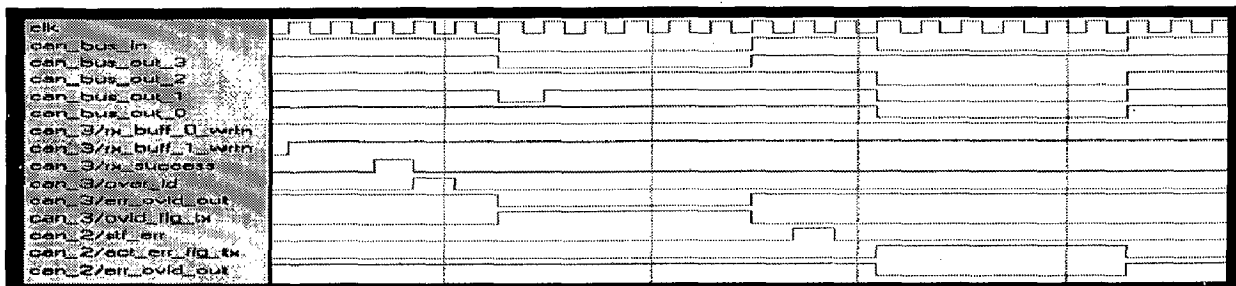


Figure 5.11 Overload frame generation.

5.4.8 Serialized Frame Transmitter:

This unit transmits the Data / Remote Frame or the Error / Overload Frame or a dominant bit during the acknowledgment slot based on the prevalent conditions. The transmitting node continues to transmit the message until the last bit, provided there is no error condition encountered during the transmission. In the event of an error the node starts transmitting the Error / Overload Frame. The node does not transmit an Error Frame when the node is in Bus Off state. Once the transmission is complete the node returns to the idle state.

The node transmits a dominant bit during the acknowledge slot, when functioning as a receiver. The node transmits recessive bits during the idle state.

5.4.9 Message Processor:

The message processor is the central unit which provides all the control and the status signals to the various other blocks in the controller. This unit routes the different signals generated in various blocks to the necessary target blocks.

The success of a transmission or reception is indicated by this block. A successful transmission is indicated by the high assertion of the *tx_success* signal similarly the successful reception is signaled by the high assertion of the *rx_success* signals. These two signals facilitate the registers to be reset.

During arbitration messages if a node loses arbitration it has to contend for bus access only after the completion of the current transmission. The high assertion of the *re_tran* signal initializes the retransmission of the message that lost arbitration.

The overload condition is also indicated by the message processor. If both the receive buffers are full and the *rd_en* signal of the node is not low the node signals an overload condition by asserting the *over_ld* signal high.

The message processor also provides information to other modules if an error occurred during the current transmission or reception. In case of an error it ensures that the error is recorded for further use. This module also acknowledges the successful reception of a message till the end of the CRC Field by asserting the

send_ack signal high. This ensures that a dominant bit is transmitted during the acknowledgement slot.

5.4.10 Arbitration Controller:

The arbitration controller is responsible for indicating the arbitration status of the node. If the output of arbitration controller of the node is logic 1 then the node is a transmitter if it is logic 0 then the node has lost arbitration and functions as a receiver of the ongoing message. The node which loses arbitration asserts a signal to indicate that a message is due for transmission.

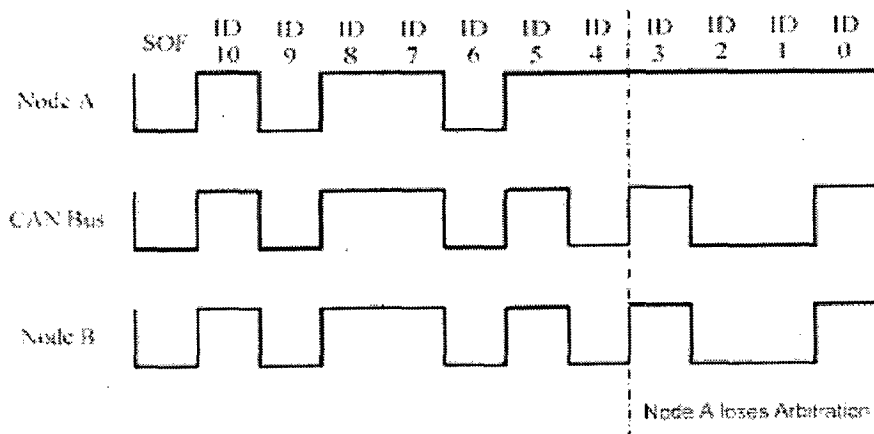


Figure 5.12 Arbitration mechanism (source: www.can-cia.de)

5.4.11 Synchronizer:

This unit configures the timing parameters of the bit time for the CAN node. Each CAN node is configured individually to create a common bit rate for all the nodes on the network even though the CAN nodes oscillator periods may be different.

Synchronizations and resynchronizations are performed on the recessive to dominant transition edges. The purpose is to control the distance between edges and Sample Points.

The specifications used for the design have been obtained from are given below:

- Bit Rate for CAN transmission: 1Megabits/second
- CAN bus length: 20 meters
- Main oscillator frequency: 8 MHz

The design for the bit timing

and synchronization unit involves the calculation of the time quanta required for the Bit timing Parameters. With a baud rate prescaler (BRP) value of 1, the CAN system clock frequency is 8MHz.

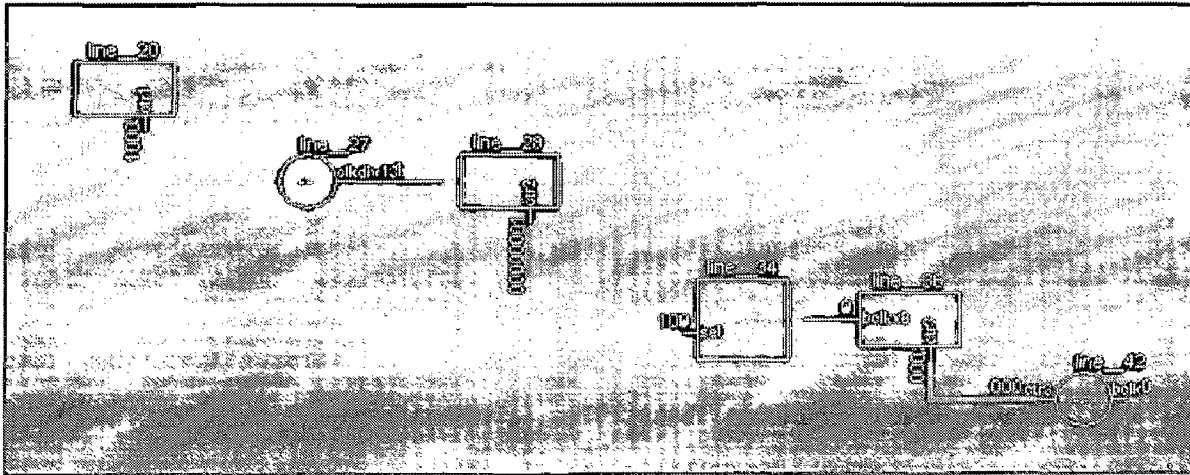


Figure 5.13 Dataflow diagram of clock divider(pre scaler)

5.4.12 Bit De-stuff Unit:

This unit has similar operation of Bitstuff unit. It performs the de-stuffing of the messages received from the CAN network. This unit also extracts the relevant information from the received message.

The CAN bus bit stream is sampled by the Synchronizer of the CAN controller.

This sampled bit stream is then de-stuffed before the relevant information is extracted from the received message. Due to the bit stuffing process of the CAN protocol a stuff bit of opposite polarity follows a sequence of 5 consecutive bits of the same polarity. The function of the de-stuffing unit is to remove the stuffed bits from the received message.

The de-stuffing process is initialized by the high assertion of the `bit_destuff_intl` signal. As soon as the de-stuffing process is initialized the CRC calculation of the received bit stream is enabled by asserting the `enable` signal high. A 64 bit temporary register `message` stores the received and de-stuffed bits. The temporary register is shifted to the left by one bit position for every de-stuffed bit and the incoming de-stuffed bit is moved into the 0th bit position of the temporary register. Figure demonstrates the method of De-stuffing.

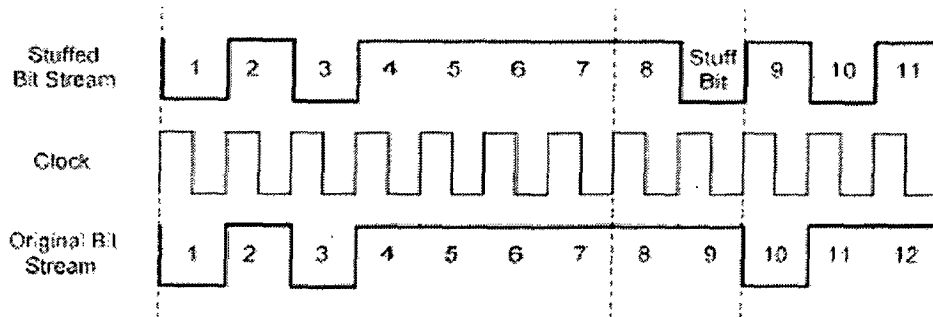


Figure 5.14 Bit destuffing(source www.can-cia.de)

5.4.13 Error checkers:

These consist of checking blocks for CRC, Stuff, Form, Bit and Acknowledgement errors according to the CAN specification.

Cyclic Redundancy Checker :

The CAN controller on receiving the *rcvd_crc_flg* performs the CRC sequence comparison. The *rcvd_crc_flg* register holds the received CRC Frame and the *rx_crc_frm* register holds the generated CRC for the received application data. A CRC Error is flagged by asserting the *crc_err* signal high if the received CRC frame and the generated CRC do not match.

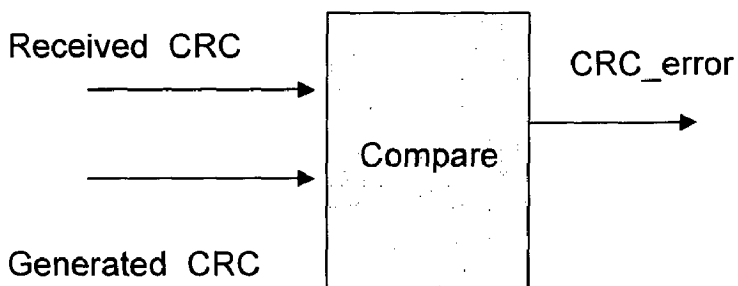


Figure 5.15 CRC checker

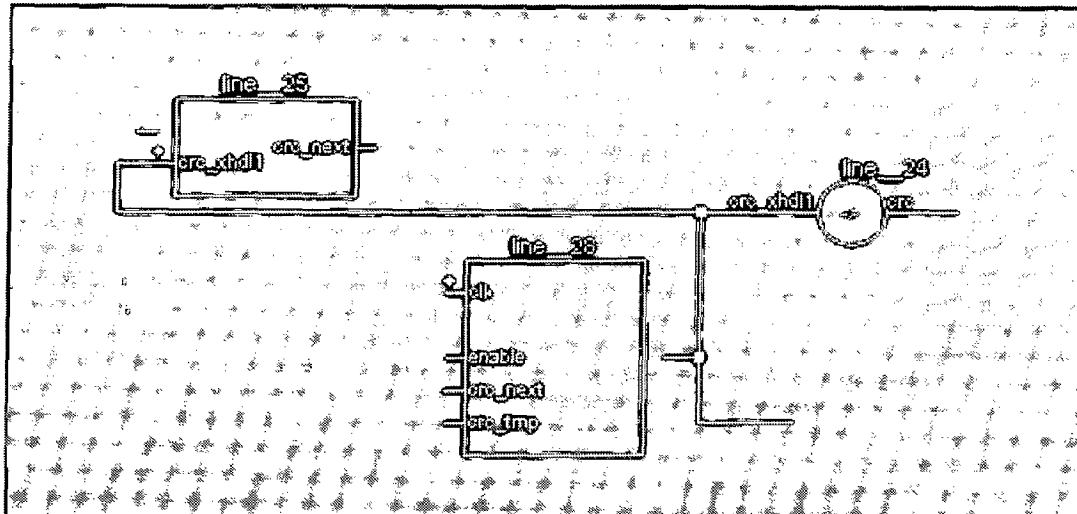


Figure 5.16 Dataflow diagram of CRC generator

Bit Stuff Monitor

This unit signals a stuff error when six consecutive bits of equal polarity are detected in between Start of Frame and the CRC Delimiter of the received message. The one's count and zero's count are fed as input to the bit stuff monitor module. A stuff error is flagged if the one's count is equal to five and the serial input is equal to logic 1 or if the zero's count is equal to five and the serial input is equal to logic 0. The occurrence of a stuff error is signaled by asserting the `stf_err` signal high.

Form Checker:

This unit checks for the serial input at the fixed from fields which are the

- CRC Delimiter bit
- Acknowledge Delimiter bit
- End of Frame Space bits.

If the receiver detects a dominant bit in any of these fields a Form Error is signaled by asserting the `frm_err` signal high.

Bit Monitor:

A CAN node acting as the transmitter of a message, samples back the bit from the CAN bus after putting out its own bit. A Bit Error occurs if a transmitter sends a dominant bit but detects a recessive bit on the bus line or, sends a recessive bit but detects a dominant bit on the bus line. A Bit Error is signaled by asserting the `bt_err` signal high.

Acknowledgment Checker

During the transmission of the acknowledgement slot a transmitter transmits a recessive bit and expects to receive a dominant bit. If the transmitting node receives a recessive bit in the acknowledgement slot it is understood that none of the nodes in the network received the message correctly and an ACK error is signaled. If the node receives a dominant bit in the acknowledgement slot then it is understood that at least one other node, has received the frame correctly. The presence of an acknowledgement error is signaled by asserting the `ack_err` signal high. Figure 5.17 demonstrates the CAN acknowledgement process.

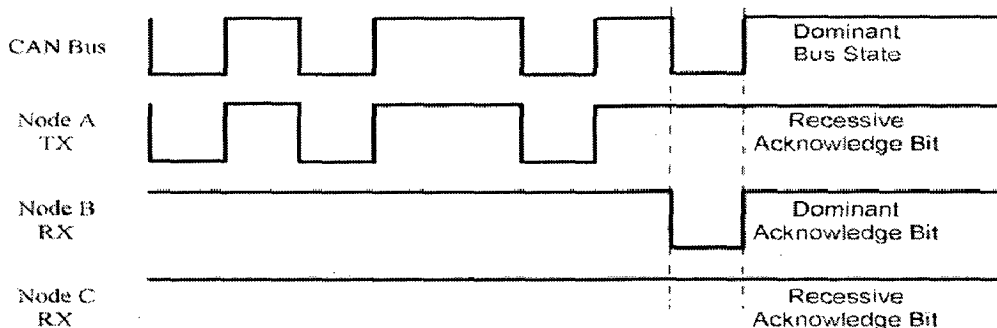


Figure 5.17 CAN acknowledgement process

5.4.14 Acceptance Checker :

This unit checks the incoming message ID and determines valid frame. The Acceptance Filter, based on the identifier, control field register of the host application, filters the received messages and stores only those required by the host application. Thus the filtering mechanism ensures only relevant messages are processed and the rest are ignored. All messages that are let through the filter must be read and checked by the CPU. This means that the final filtering is done in software.

The design of the Acceptance Filter is defined by two parameters, identifier register and control field register. These specify which particular bits to compare in the acceptance parameter with the identifier of the message.

5.4.15 Receive Buffer:

There are two 10 byte buffers, rx_buff0 and rx_buff1 that are used alternatively to store the messages received from the CAN bus. This enables the host CPU to process a message while another message is being received by the controller.

If the rx_buff0 is not written into the rx_buff_0_wr_stat signal is asserted high. If rx_buff0 is written into rx_buff_0_wrtn is asserted high. Similarly the status of rx_buff1 is also indicated. The received message is stored into the first buffer that is free on passing the acceptance filtering. The CAN controller checks the status of rx_buff0, if rx_buff0 is written then rx_buff1 is checked and written into. As soon as the data is written into the buffers the corresponding buffer written signal is asserted high.

The data is written into the buffer in the following order the received message Id given by rcvd_msg_id, rcvd_rtr, the rcvd_dlc and the rcvd_data_frm with the MSB first. The data is read from the receive buffers by the host application by asserting the rd_en signal low. This initializes the read operation.

Two signals rx_buff_0_active and rx_buff_1_active ensure that the data is read in the correct order. These signals go high as soon as the read operation for a particular buffer is initialized. At end of the read operation of a particular buffer the controller checks if the other buffer has been written into. If the other buffer is written into the controller asserts the active signal of the other buffer high, if not the active signal of the other buffer is asserted low. This ensures that the controller reads the data in the order in which the data arrives.

Once a buffer is read the rx_buff_0_read or the rx_buff_1_read signal is asserted high for the corresponding buffer. With the high assertion of the buffer read signal the contents of the corresponding buffer is reset and the buffer is ready to take in a new message. The data is sent out on the data_out bus one byte at a time.

5.4.16 Serial to parallel converter:

In Receiver logic serial to parallel converter is used and is based on the SM chart given. Receiver contains DR and SR registers and the receive control.

The control interfaces with status register and DR can drive data onto the data bus. The first process represents the combinational network, which generates the next state and control signals. The second process updates the registers on the rising edge of the clock. The SM chart is shown in fig 5.18 on next page

5.4.17 Interface Management Logic:

1. Interprets commands from NCAP.
2. Provides Interrupt and status information to the NCAP.

Used two eight bit registers as status and control registers for the interface controller.

Control register has the information about interrupt enabling signals for transmitter and receiver while the status register consists of the information about errors and status information of bus. These bits work like inputs and outputs for bit stream processor.

Table 5.1 Interface Management Logic

ADDR2	R_W	action
00	0	DBUS←DR
00	1	DBUS→DR
01	0	DBUS←Status Register
01	1	--
1X	0	DBUS←control register
1X	1	DBUS→control register

The blocks are Implemented in VHDL and synthesized with xilinx ISE 9.2i.

The simulation and synthesized RTLs are discussed in next chapter.

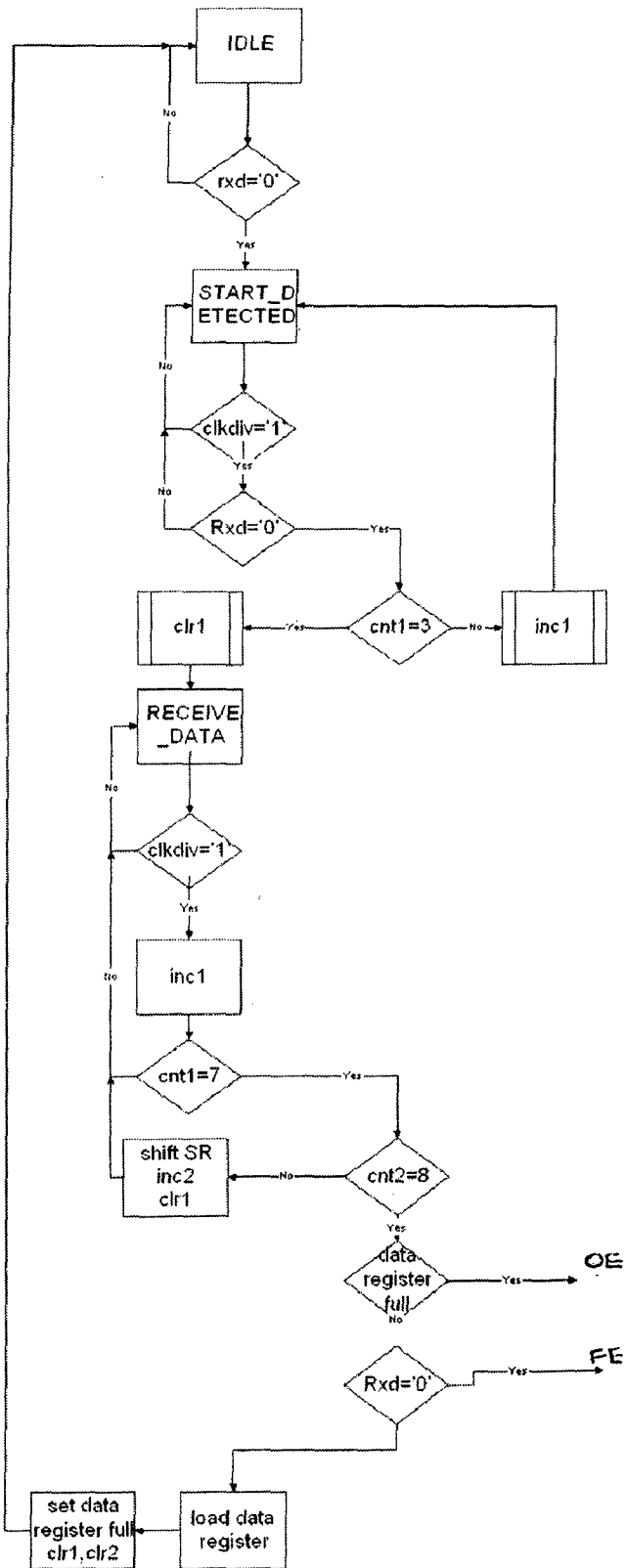


Figure 5.18 SM chart for serial to parallel converter:

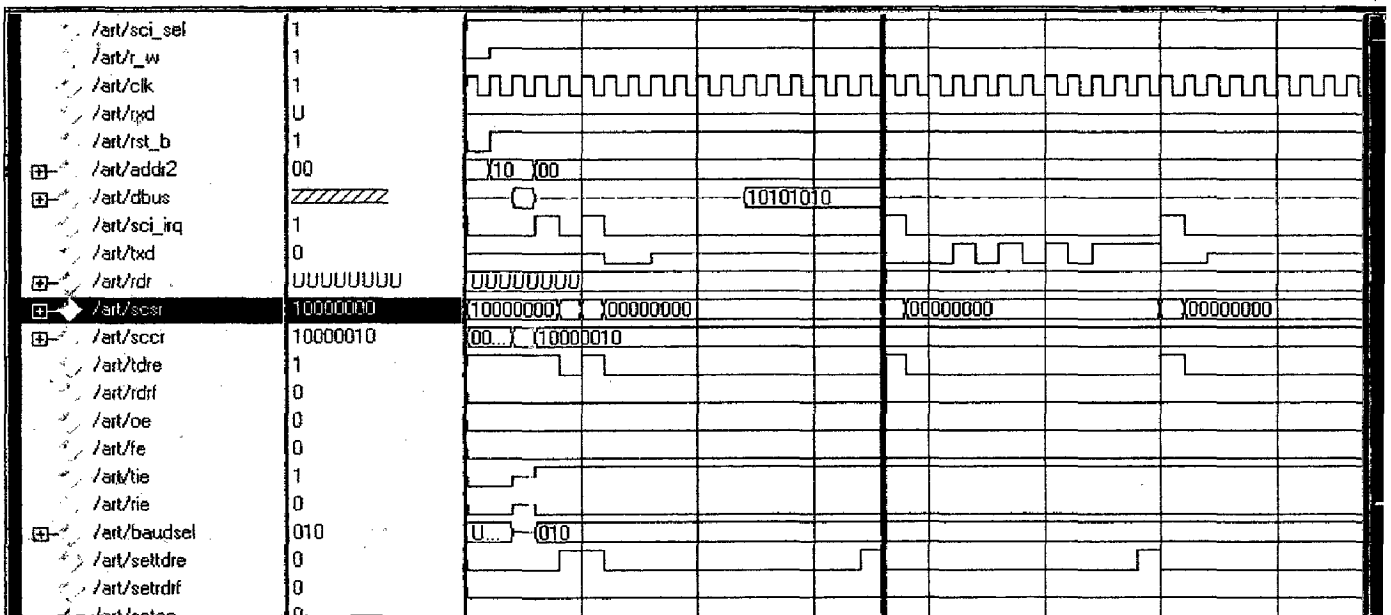
CHAPTER 6 RESULTS

6.1 Simulation results

a) Interface Management logic:

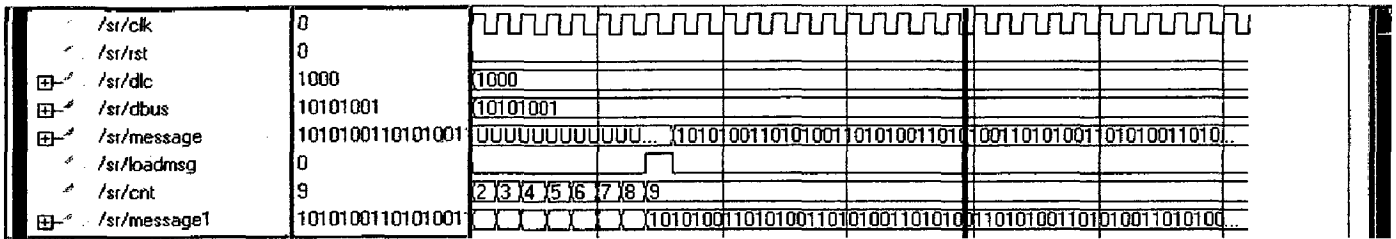
When the `addr2="10"` and `R_w='1'` the host processor writes the message to the parameter registers of the node.

The control register enable the `sci_irq`(Interrupt request to processor) when the node needs attention. After each transmission of 8 bits from DBUS the `irq` has become high is shown in simulation diagram.

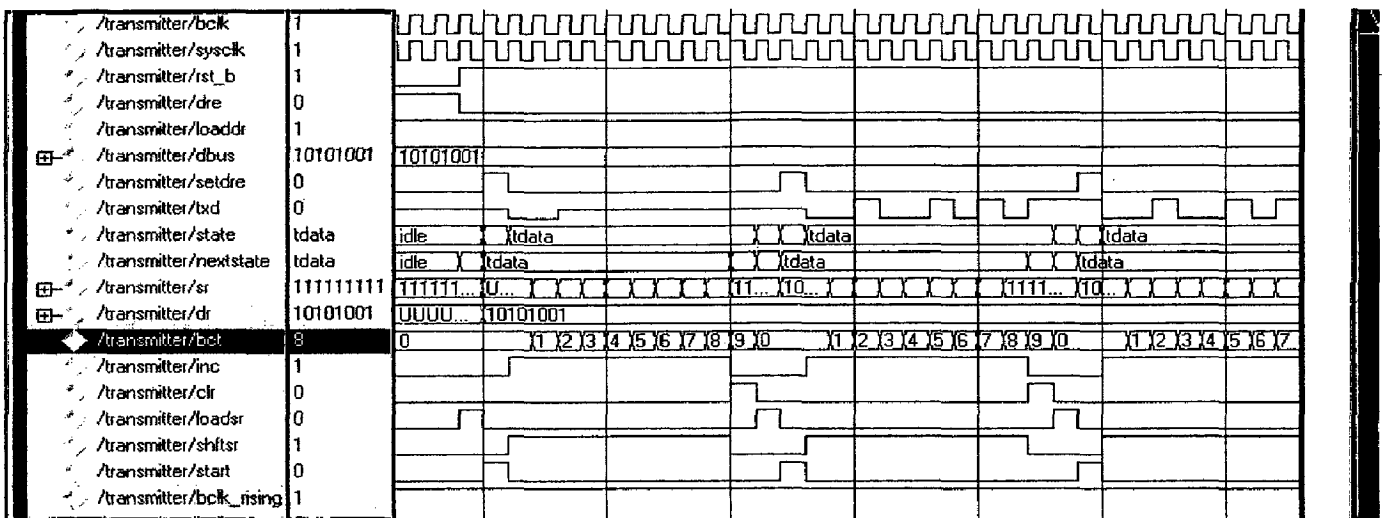


b) Tx Buffer:

In the simulation diagram shown below with the given DLC of "1000" after `cnt` reaches the eight the `loadmsg` signal becomes high and this initiates the data/remote frame generator. Now the message is tx buffer and this message is converted to serial by parallel to serial converter.

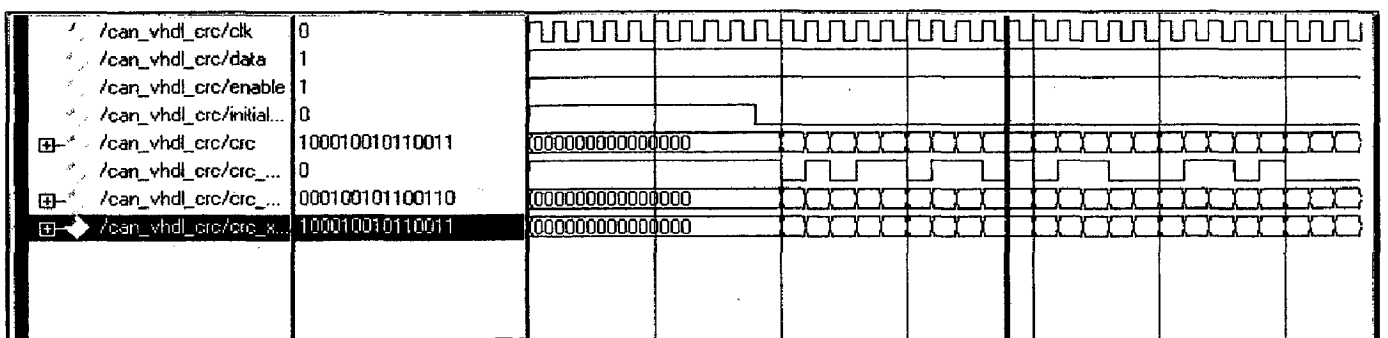


c) Parallel to serial converter:



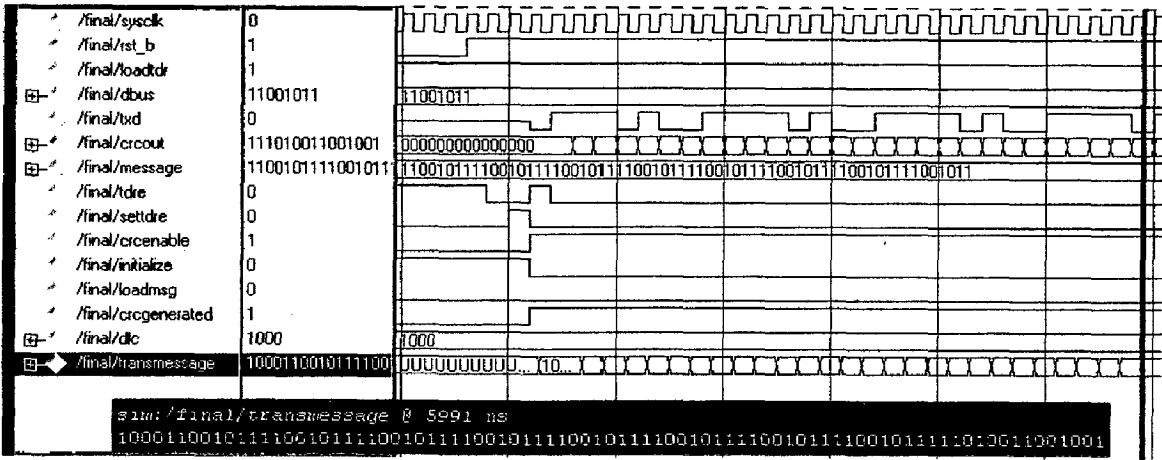
The simulation result shows the parallel to serial converter. The *message* is written to this component byte by byte on eight bit DBUS. When *dre* (*data register empty*) signal goes '0' the state machine goes to *tdata* state and transmits data serially on the line *txd*. This *txd* input is given to the *data* input of CRC generator/checker..

d) TX CRC generator:



The output of parallel to serial converter is given to the data input pin and crc calculation is initiated by data frame generator when *settdre* signal goes high in the previous

simulation diagram. And initialize becomes '0'. The calculated CRC is stored in CRC(14:0) register.



The above result shows the complete CAN message before bit stuffing.

e) Bit stuffing:

The bit stuffing mechanism is demonstrated in Fig. With the initialization of bit stuffing the contents of the 98 bit register dt_rm_frm are transferred to a temporary register msg. The contents of the msg register are serialized and then checked for the bit stuffing condition. At every positive edge of the clock the contents of the msg register are shifted left by 1.

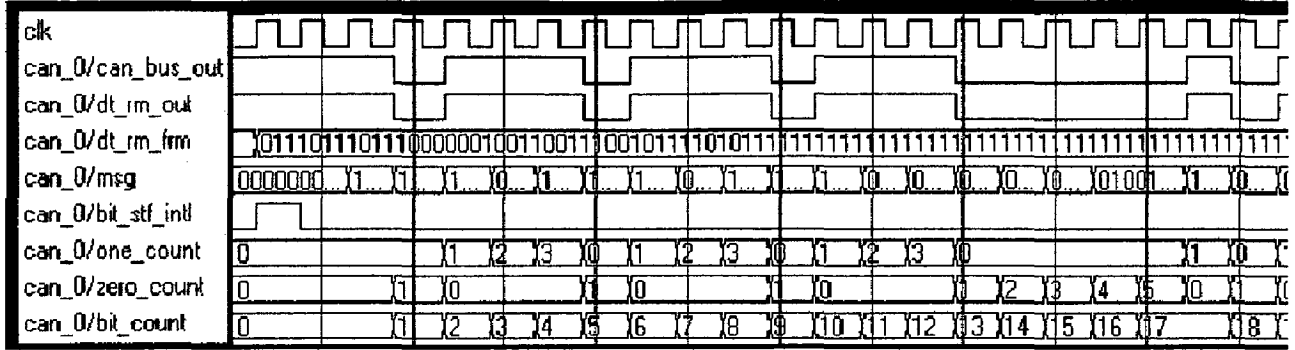
The left shifted bit is placed on the dt_rm_out bus which is connected to the CAN bus. A count of the bits in the original un-stuffed message stream is kept in the register bit_count. The stuffed bit stream being put out on the CAN bus has the Start Bit being transmitted first, followed by the message identifier's most significant bit and so on.

Two counters one_count and zero_count keep tab of the number of consecutive ones and zeros encountered respectively in the message stream

DLC. On detecting five zeros at bit_count 17 a recessive bit is stuffed into the bit stream.

The bit_count is not incremented and the msg register is not shifted left as explained

above. The bit stuffing mechanism is demonstrated in Figure.



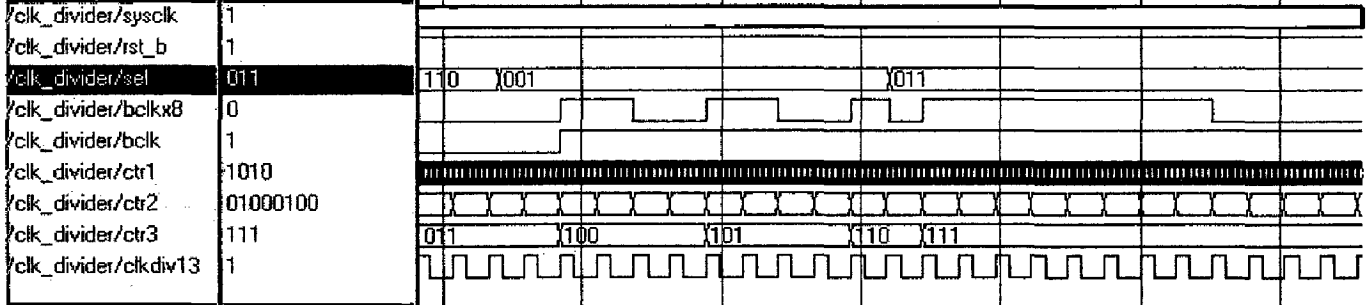
Bit destuffing in receiving is similar to bit stuffing in transmission and CRC generator can be used as CRC checker.

RX buffer is implemented same as TX Buffer.

Bus arbitration block is implemented by using XNOR gate with the two inputs. One is Rxd from other CAN node and second one is txd.

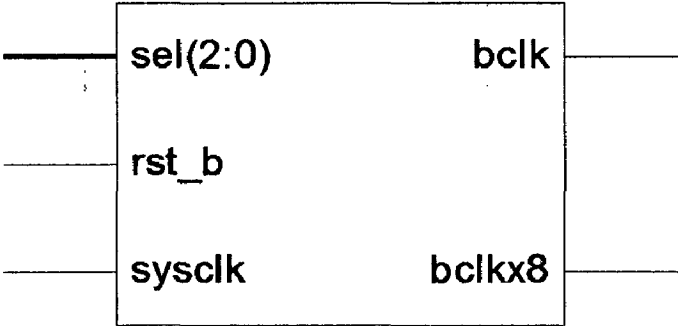
The clock prescaler (baud rate generator) provides the required clock of 8MHz.

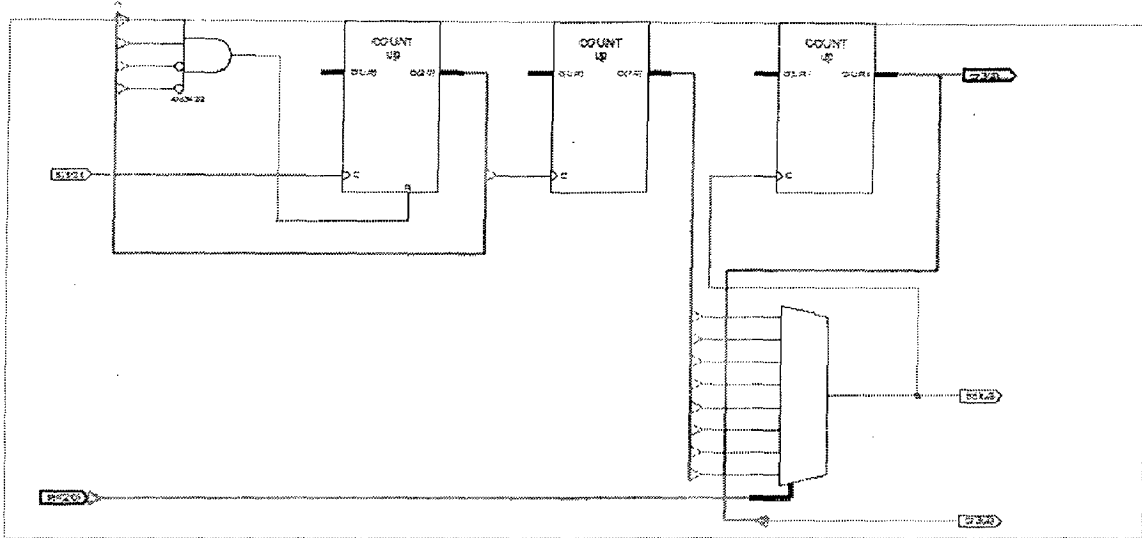
.Three bits in control register can be used to select any one of eight baud rates.



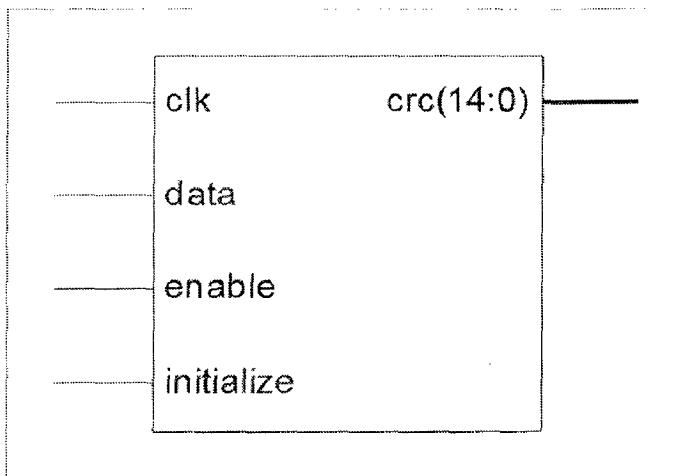
6.2 Synthesis Results:

a) Clock divider synthesized RTL:

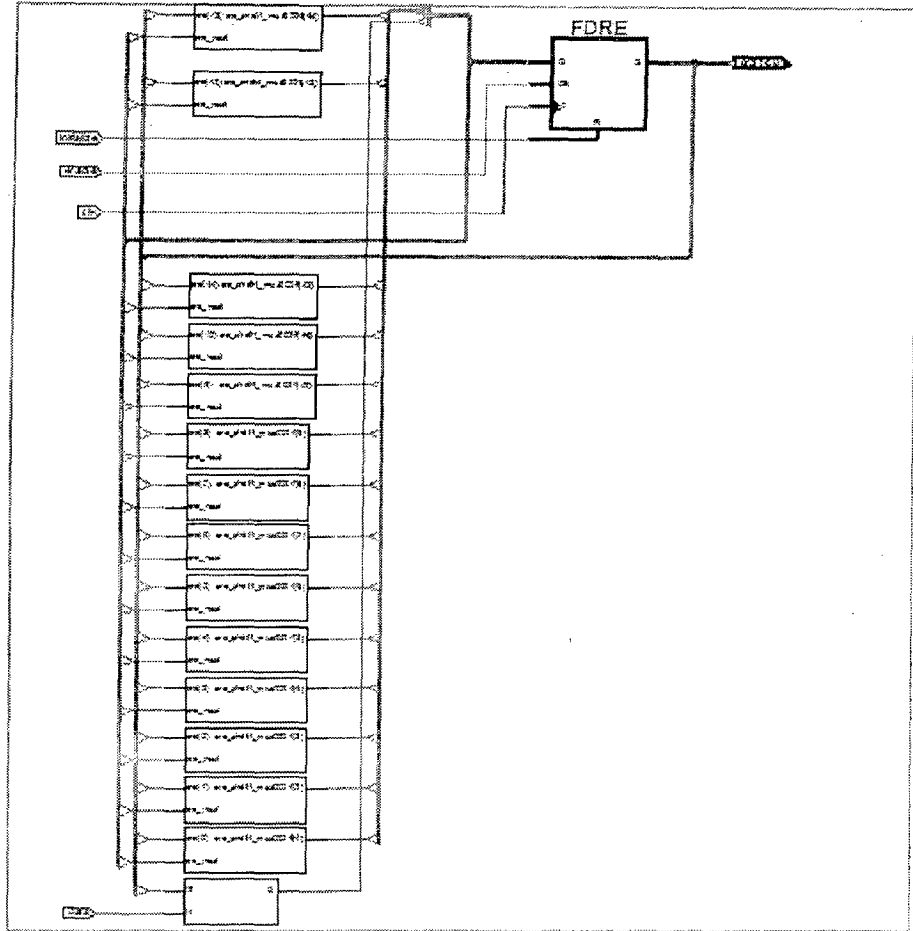




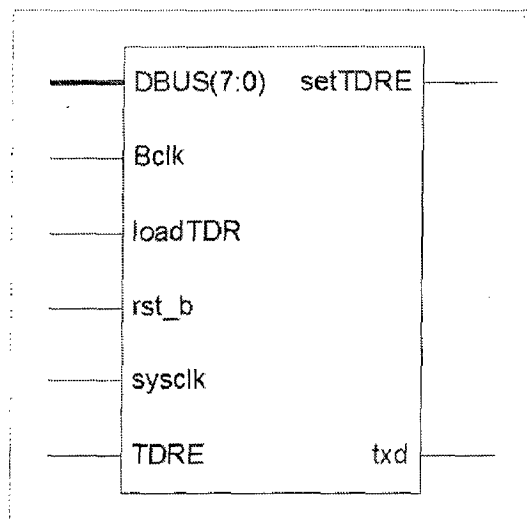
b) CRC generator/checker:



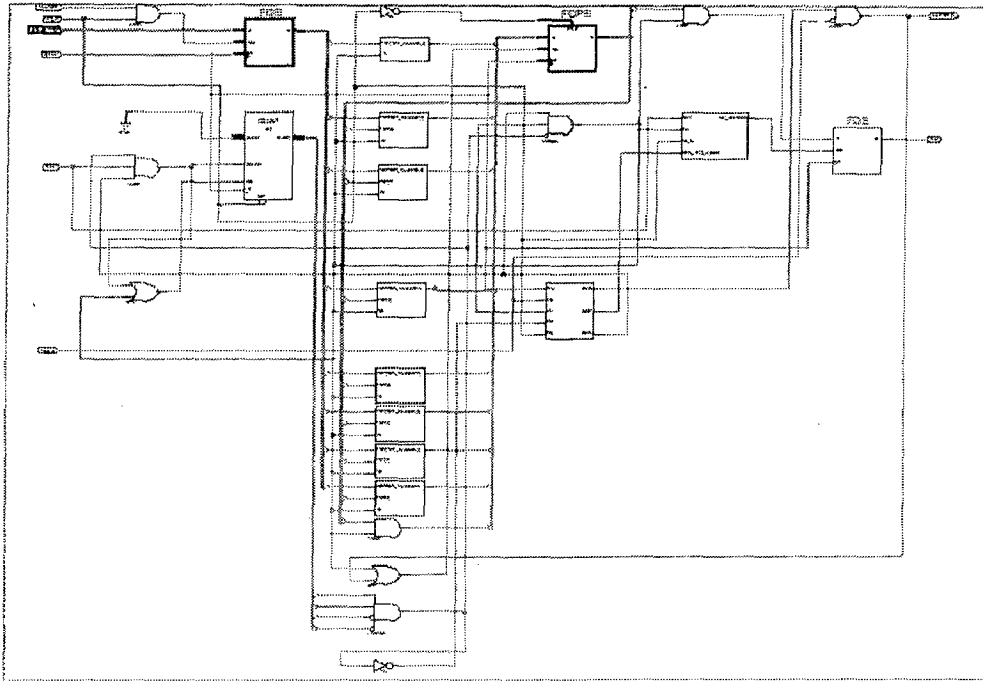
Synthesized RTL:



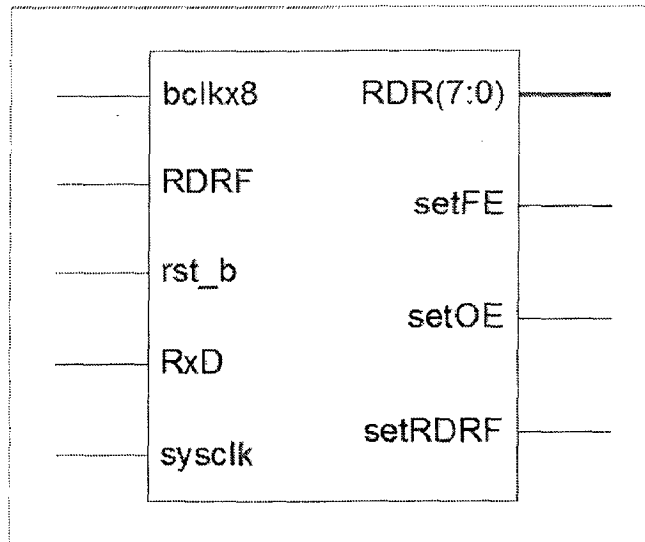
c) Parallel to serial Converter:



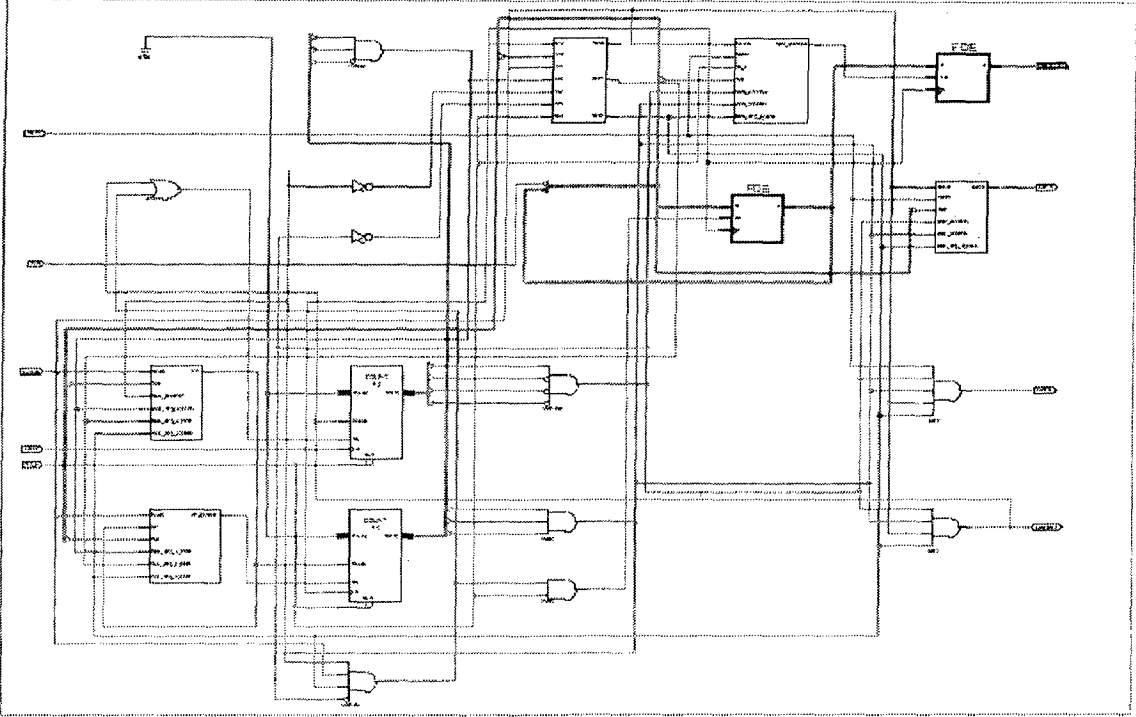
Synthesized RTL



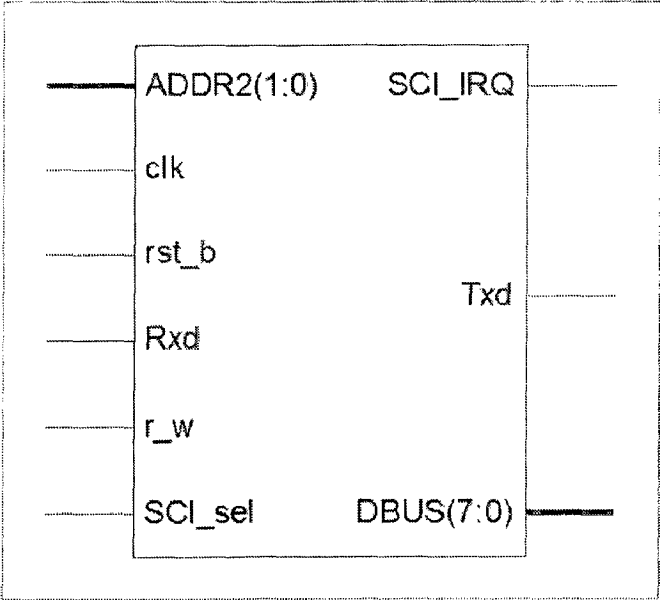
d) Receiving component:



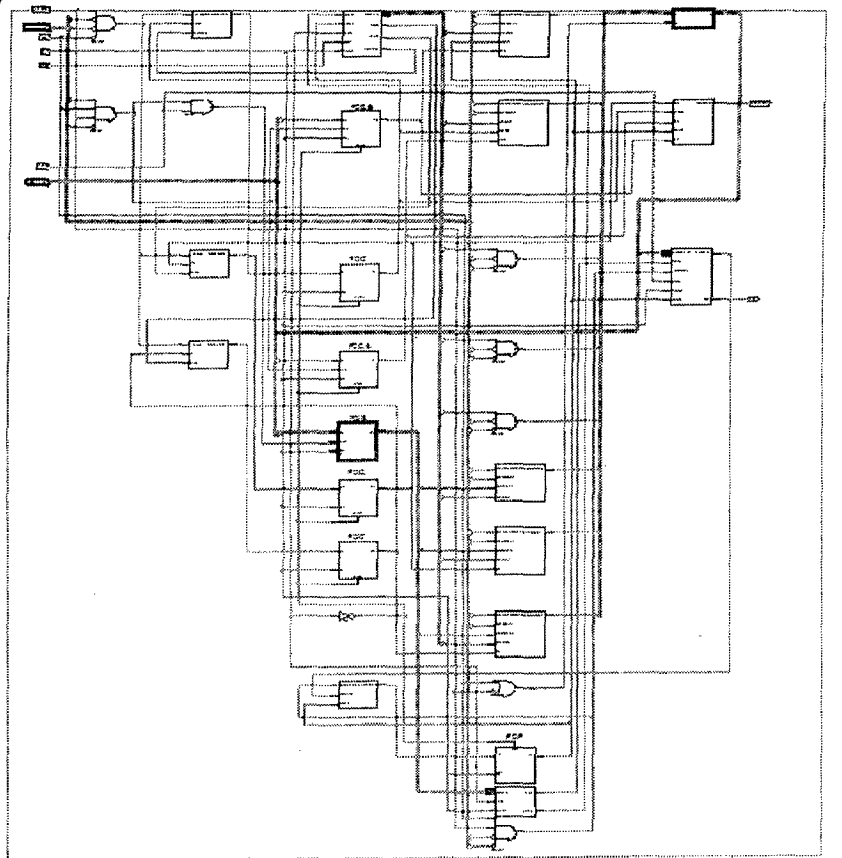
Receiver synthesized RTL



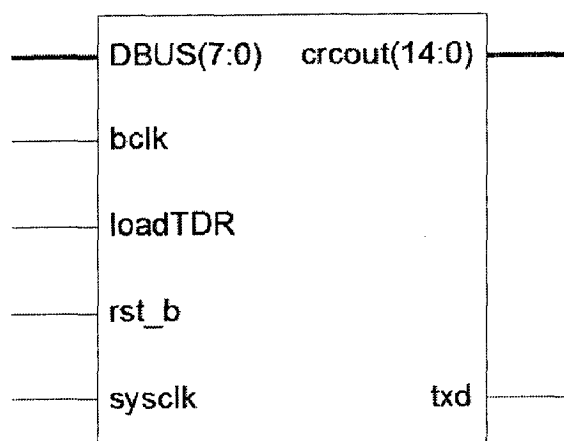
e) Interface management logic:



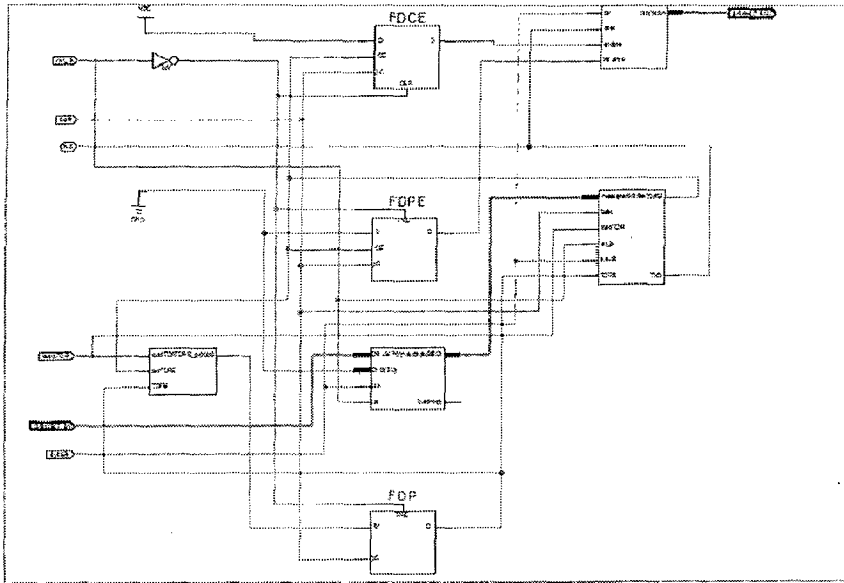
Synthesized RTL



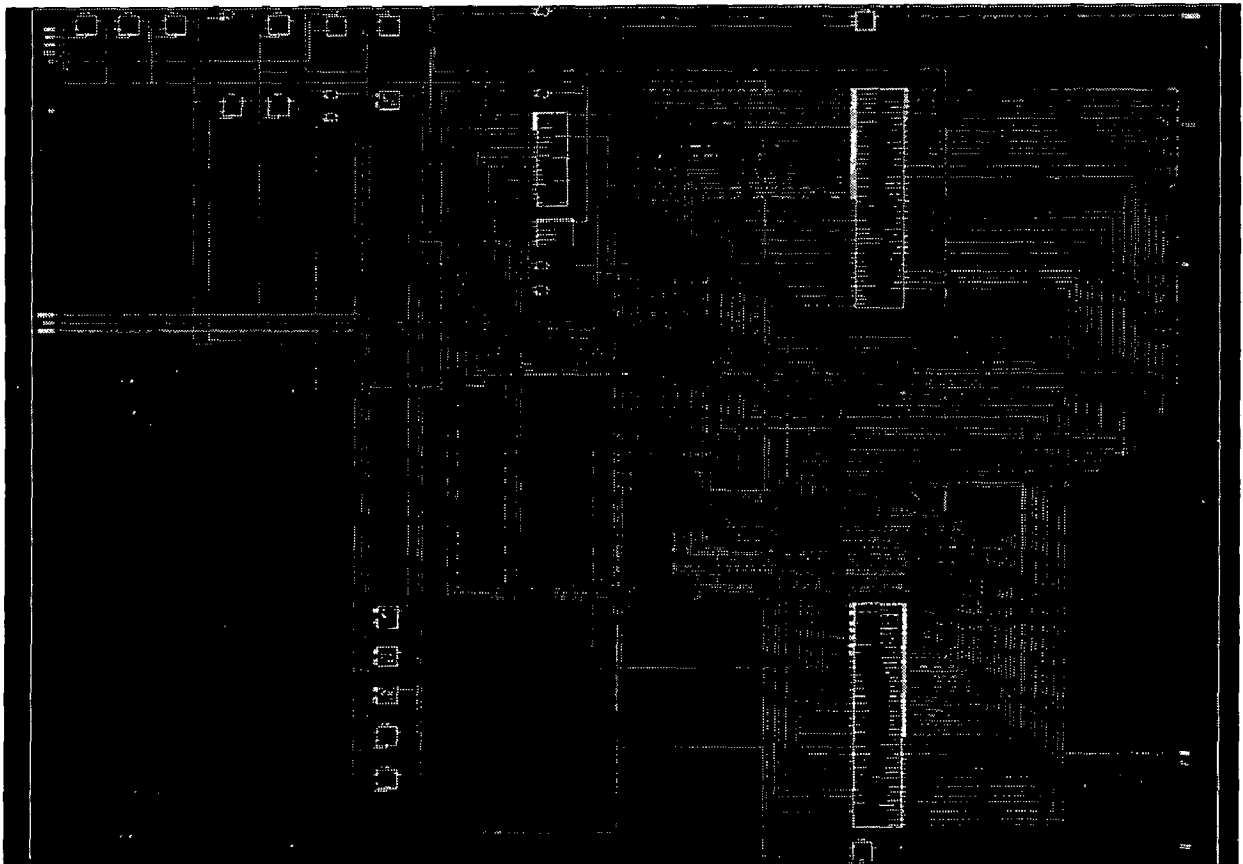
f) Before bit stuffing :



Synthesized RTL:



Final Synthesized RTL schematic:



Device Utilization Summary:

Sources:

Sources for: Synthesis/Implementation

- CAN
- xc3s500e-4fg320
- can_vhdl_top - RTL (/can_part)

Sources | Snapshots | Libraries

Processes:

Processes for: can_vhdl_top - RTL

- Add Existing Source
- Create New Source
- View Design Summary
- Design Utilities
- User Constraints
- Synthesize - XST
 - View Synthesis Report
 - View RTL Schematic
 - View Technology Schematic
 - Check Syntax
- Generate Post-Synthesis Sim
- Implement Design

FPGA Design Summary

- Design Overview
 - Summary
 - IOB Properties
 - Timing Constraints
 - Pinout Report
 - Clock Report
- Errors and Warnings
 - Synthesis Messages
 - Translation Messages
 - Map Messages
 - Place and Route Messages
 - Timing Messages
 - Bitgen Messages
 - All Current Messages
- Detailed Reports
 - Synthesis Report
 - Translation Report
 - Map Report
 - Place and Route Report

Project Properties

- Enable Enhanced Design Summary
- Enable Message Filtering
- Display Incremental Messages

Enhanced Design Summary Contents

- Show Partition Data
- Show Errors
- Show Warnings
- Show Failing Constraints
- Show Clock Report

CAN Project Status			
Project File:	CAN.isc	Current State:	Synthesized
Module Name:	can_vhdl_top	Errors:	No Errors
Target Device:	xc3s500e-4fg320	Warnings:	91 Warnings
Product Version:	ISE 9.2i	Updated:	Sun Aug 24 19:56:45 2008

CAN Partition Summary	
No partition information was found.	

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	704	4656	15%
Number of Slice Flip Flops	549	9312	5%
Number of 4 input LUTs	1322	9312	14%
Number of bonded IOBs	36	232	15%
Number of GCLKs	2	24	8%

Detailed Reports					
Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report	Current	Sun Aug 24 19:56:44 2008	0	91 Warnings	4 Infos
Translation Report					
Map Report					
Place and Route Report					
Static Timing Report					
Bitgen Report					

CHAPTER 7

Conclusion and Future scope

7.1 Conclusion:

The design of the RIC (In present case CAN controller) for this project has been described in detail in Chapter 4 and the results obtained from the simulation of the post layout model, have been elucidated in Chapter 6. Thus the FPGA implementation of a basic protocol controller for CAN 2.0A protocol has been accomplished.

The software Implementation of communication interfaces by RIC allows having minimal redundancy for supporting additional interfaces and giving opportunity to change set of supported interfaces by development of new software for RIC.

The interface controllers for I2C and SPI are simplified versions of the structure of the RIC designed for CAN bus.

The reconfiguration of FPGA by the NCAP (micro controller) provides an implementation of the set of interfaces. The requirements of FPGA size are determined by the complexity of the interface controller.

.Network sensors based on IEEE 1451 will share more quotients in the sensors market as a dominant product. We can see that network sensors have more extensive application in the future.

7.2 Scope of Future work:

Future work for the presented work consists of the following:

- Complete an Application Specific Integrated Circuit (ASIC) for this IEEE 1451 single chip solution discussed in chapter 2.
- Design a completely re-configurable TIM block. This proposed block would have a database of TEDS associated with it that contains information about a variety of transducers. Then, using this database we can select the transducer(s) that we would like to implement, and the complete transducer channel TIM block would be instantiated. In order to accomplish this, we would need to design a Graphical User Interface (GUI) that has the capability of dynamically setting the I/O ports of the control unit as well as the individual transducer channels.

REFERENCES

- [1] IEEE standard for a Smart transducer Interface for sensors and actuators-Network Capable Application Processor (NCAP) Information Model, IEEE standard 1451.1-1999, 1999.
- [2] IEEE Standard for a Smart Transducer Interface for sensors and actuators- Transducer to Microprocessor Communication Protocols and transducer Electronic Data Sheet (TEDS) formats, IEEE Standard 1451.2-1997, 1997
- [3] J.Camara, J.Samitier, and O.Ruiz, "Complete IEEE 1451 Node, TIM and NCAP, Implemented for a CAN network," in Proc.IEEE Instrum.Meas.Tech.Conf.IMTC2000, Baltimore, MD, May2000
- [4] Raman Kochan,Volodymyr Kochan,Antoly Sachenko,Ihor Maykiv,Iryna Turchenko "Network Capable Application Processor Based on a FPGA"IMTC2005,Ottawa,Canada,17-19 May 2005
- [5] Jingjun Cui,Dagui Huang,Zhonglai Wang,Dongxing Qin "Implementation of Network Smart Sensors for Ultraviolet Fire Detector" international .conf.on mechatronics and automation, June 2006,china
- [6] D. Wobshall. "An Implementation of IEEE 1451 NCAP for Internet Access of Serial Port-Based Sensor", Proceedings of second Sensor for Industry Conference Slcon/02, 19-21 November 2002. Houston, Texas, ISBN 1-55617-X344, pp.157-160.
- [7] Kyung Chang Lee, Man Ho Kim, Suk Lee and Hong Hee Lee, "IEEE 1451-Based Smart module for In vehicle Networking Systems of Intelligent Vehicles", IEEE transactions on Industrial Electronic,vol 51,no. 6,December 2004.
- [8] Kang Lee, "IEEE 1451: A standard in Support of Smart transducer Networking", IEEE Instrumentation and Measurement Technical conference, Baltimore,MD USA May,2000.
- [9] P.Ferrari,A.Flemmini,D.Marioli,A.Taroni," VHDL Implementation of IEEE 1451.2 Smart Sensor", IEEE Instrumentation and Measurement Technology Conference proceedings, vol 1, pages 716-720, 2003

- [10] Angel de Castro, Esteban, Teresa, "A System on Chip for Smart sensors" IEEE Instrumentation and Measurement Technology Conference proceedings, 2002.
- [11] J Bhaskar , " A VHDL Primer" third edition, Pearson Prentice Hall.
- [12] R. Kochan', K. Lee', V. Kochan', A. Sachenko' "Development of a Dynamically Reprogrammable NCAP", IMTC 2004 Instrumentation and Measurement Technology Conference Coma Italy. May 18-20, 2004.
- [13] R. Kochan, V. Kochan, A. Sachenko, I. Maykiv, A. Stepanenko "Interface and Reprogramming Controller for Dynamically reprogrammable Network Capable Application Processor (NCAP)"IEEE Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications 5-7 September 2005, Sofia, Bulgaria
- [14] Design flow at www.xilinx.com/support/software_manuals
- [15] CAN Specification Version 2.0, Robert Bosch GmbH, Stuttgart, Germany, 1991.
- [16] Spartan 3E FPGA starter kit manuals.

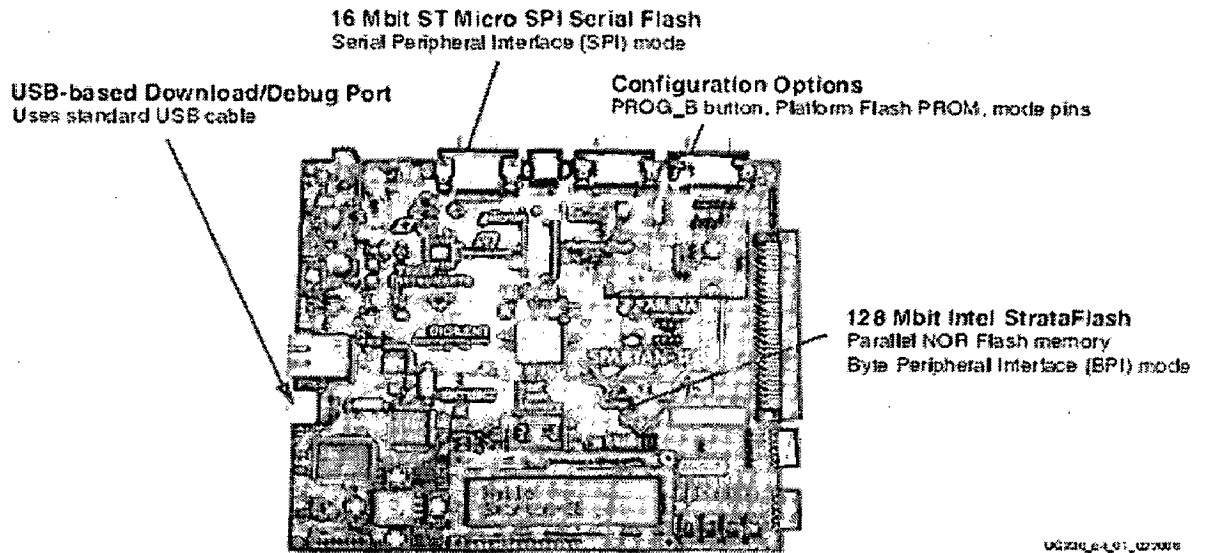
- [17] Randy Frank, "Understanding Smart Sensors", Second Ed., Artech House, April 2000
- [18] NIST website describes IEEE 1451 standard at <http://ieee1451.nist.gov/>
- [19] Blagomir Donchev, Marin Hristov, Implementation of CAN Controller With FPGA Structures, 7th International Conference, CADSM, 2003.

APPENDIX

FPGA configuration options

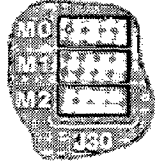
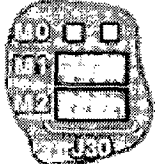
A few system-level design trade-offs were required in order to provide the Spartan-3E Starter Kit board with the most functionality.

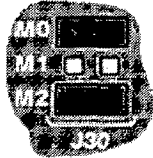
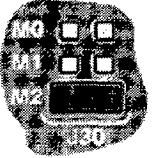
- Download FPGA designs directly to Spartan-3E FPGA via JTAG (**Joint Test Action Group (JTAG)** is the usual name used for the IEEE 1149.1 standard), using the onboard USB interface. The on-board USB-JTAG logic also provides in-system programming for the on-board platform Flash PROM and the Xilinx XC2C64A CPLD.
- Program the on-board 4Mbit Xilinx XCF04S serial Platform Flash PROM, then configure the FPGA from the image stored in the Platform Flash PROM using Master Serial mode.
- Program the on-board 16 Mbit ST Microelectronics SPI serial Flash PROM, then configure the FPGA from the image stored in the SPI serial Flash PROM using SPI mode.
- Program the on-board 128Mbit Intel StrataFlash parallel NOR Flash PROM, then configures the FPGA from the image stored in the Flash PROM using BPI Up or BPI Down configuration modes. Further, an FPGA application can dynamically load two different FPGA configurations using the Spartan-3E FPGA's MultiBoot mode.



Configuration Mode Jumpers:

As shown in the Table, the J30 jumper block settings control the FPGA's configuration mode. Inserting a jumper grounds the associated mode pin. Insert or remove individual jumpers to select the FPGA's configuration mode and associated configuration memory source.

Configuration Mode	Mode Pins M2:M1:M0	FPGA Configuration Image Source	Jumper Settings
Master Serial	0:0:0	Platform Flash PROM	
SPI	1:1:0	SPI Serial Flash PROM starting at address 0	

BPI UP	0:1:0	StrataFlash parallel Flash PROM, starting at address 0 and incrementing through address space. The CPLD controls address lines A[24:20] during BPI configuration.	
BPI Down	0:1:1	StrataFlash parallel Flash PROM, starting at address 0x1FF_FFFF and decrementing through address space. The CPLD controls address lines A [24:20] during BPI configuration.	
JTAG	0:1:0	Downloaded from host via USB-JTAG port	