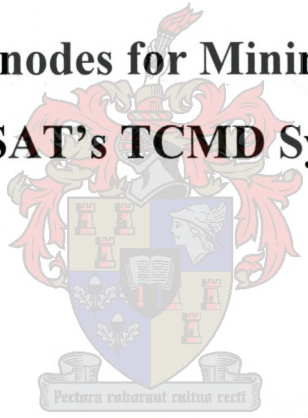
UNIVERSITEIT
STELLENBOSCH
UNIVERSITY

The Design of CAN nodes for Minimising Cables on the SUNSAT's TCMD System



R.D. Musetha

The Design of CAN nodes for Minimising Cables on the SUNSAT's TCMD System

By

Rendani D. Musetha

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Engineering Science at the University of Stellenbosch

Supervisor: Prof. P.J. Bakkes

Department of Electrical and Electronic Engineering

University of Stellenbosch

December 2003

Declaration

I, the undersigned hereby declare that the work contained in this thesis is my own original work and has not previously in its entirety or in part been submitted at any university for a degree.

Signature: Date:

Summary

The aim of this thesis is to investigate a design of a microcontroller based embedded system that will be used to minimise cable harness on the SUNSAT micro-satellite. The system is called CAN node.

The CAN node(s) implements CAN (Controller Area Network) serial bus architecture protocol. The protocol is implemented on the two nodes to transport data from the TCMD to the other trays of SUNSAT. CAN node(s) design project focuses on the TCMD tray, because it is the central point for data communication in SUNSAT and it acts as the eyes and hands of the satellite's operator. As a result most of the communication cables are located at this tray. The two nodes are called TX-node and RX-nodes. The TX-node is used to collect data from the TCMD tray and transmits them serially to RX-node. The RX-nodes receives the TCMD data from TX-node and transmits these data to their respective nodes. In application RX-nodes need to be ten, but only one is used for testing purpose.

The design had its shortcomings, of which they are discussed in this thesis. The recommendations of an ideal system are also given to elaborate how the system should behave in the real situation.

Despite its shortcomings, the CAN node(s) project has successfully proven that cable harness on the TCMD tray of SUNSAT can be minimised by using CAN technology.

Opsoming

Die doel van hierdie tesis is om die ontwerp van 'n mikro-beheerder gebaseerde stelsel wat die SUNSAT mikro-satelliet kabel harnas sal verklein, te ondersoek. Die stelsel word die CAN nodus genoem.

Die CAN nodus implementeer die CAN (Controller Area Network) bus argitektuur protokol. Die protokol is op twee nodusse geïmplementeer om data vanaf die TCMD na ander laaie van SUNSAT te voer. Die CAN nodus ontwerp fokus op die TCMD laai, want dit is die sentrale punt vir data kommunikasie in SUNSAT en dit tree soos die oog en hande van die satelliet operateur op. As 'n gevolg, is die meeste van die kommunikasie kabels in hierdie laai. Die twee nodusse is genoem TX-nodus en RX-nodus. TX-nodus word gebruik om die data van die TCMD af te kollekteer en dan versprei hulle tot hulle onderskeie nodusse. In die toepaslik moet daar tien RX-nodusse wees, maar net een is gebruik terwille van die toets. Die ontwerp het sy eie tekortkomings, wat in hierdie tesis bespreek word. Die rekommendasie van 'n ideale stelsel is ook gemaak om te bewys hoe die stelsel dit in 'n ware situasie moet gedra.

Ongeag die tekortkomings daarvan, het die CAN-nodus projek suksesvol bewys dat die kabel harnas in die TCMD laai van SUNSAT kan verminder word deur die gebruik van die CAN tegnologie.

Acknowledgements

First of all I would like to thank God for giving me the strength, throughout this project. I would like to express my deepest gratitude to all my family members, in-memory of my late younger brother Tshimangi. I would also like to thank all the people who supported me especially my friends who really stood by me in this thesis. Last but not least I want to thank the following people who were dedicated in making this thesis a success.

Prof. P.J Bakkes for being such a supportive supervisor.

Arno Barnard for being a supportive mentor.

X.C Farr for all the support he gave me.

Alec Rust my former mentor.

Contents

List of Figures

Acronyms and Abbreviations

A/D	Analog-to-Digital converter
ACK	AKCnowledgement
ADCS	Attitude Determination Control System
ALE	Address Latch Enable
AQ	AcQuisition
CAN	Controller Area Network
CE	Chip Enable
CPLD	Complex Programmable Logic Device
CRC	Cyclic Redundancy Check
CS	Chip Select
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DLC	Data Length Code
EOF	End Of Frame
EPROM	Erasable Programmable Read Only Memory
FPGA	Field Programmable Gate Array
IBUS	Instrumentation BUS
IEEE	Institute of Electrical and Electronic Engineers
JTAG	industry-standard Test Action Group
LE	Latch Enable
LED	Light Emitting Diode
LEO	Low Earth Orbit
LoNWorks	Local Operating NetWorks
LSB	Least Significant Bit
MAC	Medium Access Control
MO	Message Object
MSB	Most Significant Bit
NDA	Non-Destructive Arbitration
OBC1	On-board Computer 1
OBC2	On-board Computer 2
OE	Output Enable

PCB	Printed Circuit Board
PLD	Programmable Logic Device
PS	Passive Serial mode
PSEN	Program Store Enable
RF	Radio Frequency
RTR	Remote Transmission Request
RX	Receive
SBA	Serial Bus Architecture
SOF	Start Of Frame
SRAM	Static Random Access Memory
SSB	SUNSAT Serial Bus
SUNSAT	Stellenbosch UNiversity SATellite
TCMD	Telecommand
TLM	Telemetry
TCMD	Telemetry and TelecoMmand System
TX	Transmit
USB	Universal Serial Bus
UV	Ultra Violet
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

Chapter 1

1 Introduction

The main aim for this project is to design a Controller Area Network (CAN) technology based embedded digital system, in order to minimise cable harness on SUNSAT telecommand tray. Throughout this literature the system will be referred to as CAN node(s). This chapter gives the introduction of the design by (1) briefly explaining the background for the studies, (2) highlighting some few points that explain the need for cable harness minimisation and (3) concluding with an overview of all the chapters. The idea is to use a Serial Bus Architecture (SBA) protocol to reduce cable harness into few cables. SBA can be defined as a network protocol that is implemented over a serial communication wire. Although there are many SBA protocols, CAN have been chosen for this project. The reasons and advantages for using CAN technology will be discussed in Chapter 3 (Evaluation of various SBA technologies).

1.1 Background

SUNSAT (**S**tellenbosch **Un**iversity **Sat**ellite) is a LEO[1] amateur micro-satellite, which, was built by the University of Stellenbosch (in 1991). It was then successfully launched (in 1994) as the first satellite to be launched in the history of South Africa. One of its main objectives was to provide further researches and exploration for South Africa in the field of satellite engineering. Figure 1.1 gives a block structure of SUNSAT. The satellite consists of eleven (11) trays. Each of these trays houses one or more of the satellite' subsystems [1]. As it can be seen in figure 1.1, the signals between different layers are transported to and from different subsystems via dedicated wires.

The control of all subsystems on the satellite originates in the telecommand (TCMD) system, and as a result, 232 control lines are implemented, enabling the system to

[1] LEO-Low Earth Orbiting satellite.

manage hardware functions all over the satellite [2]. Therefore CAN node(s) design project focuses on this tray.

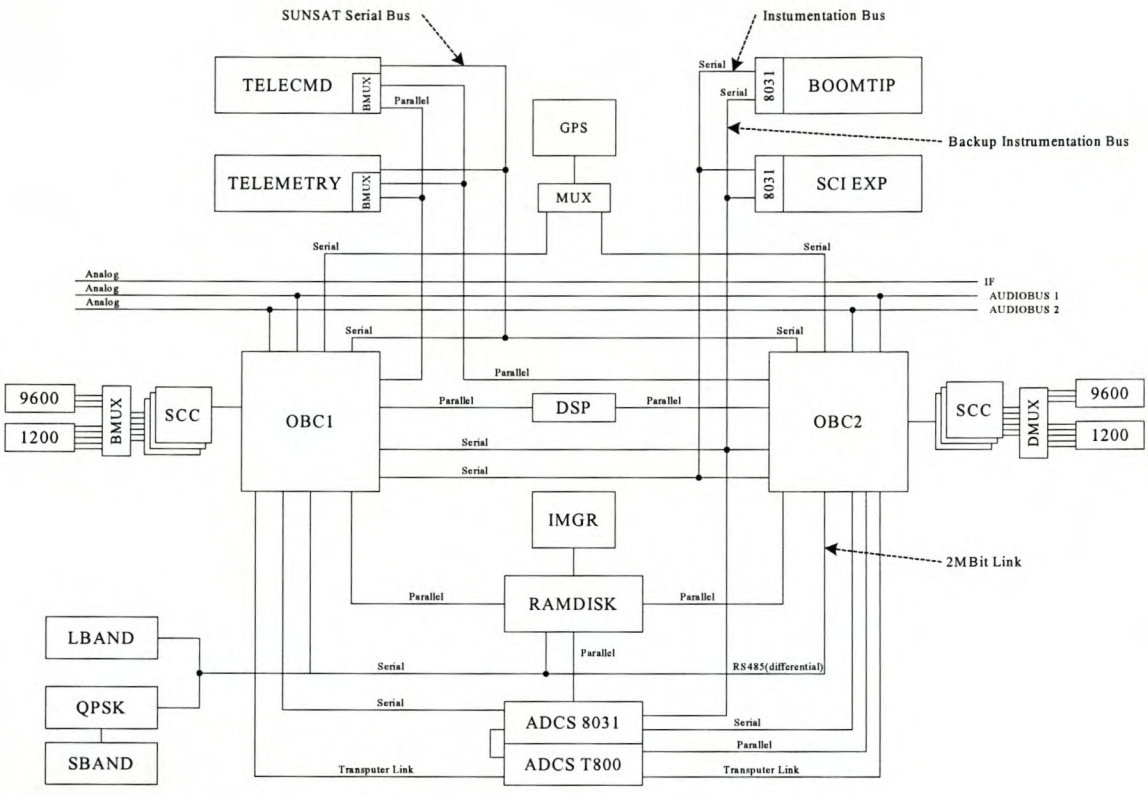


Figure 1.1: SUNSAT block diagram

Throughout this text, the words telecommand system and TCMD will be used interchangeably. The detailed description for this tray will be given in the next chapter (chapter 2).

1.2 The Need for Cable Harness Minimisation

CAN node(s) design project is one of the necessary marginal changes on the SUNSAT to address the need of minimising the complex harness of cables around the TCMD system. Several studies on SUNSAT1 have shown that it is crucial to minimise the cables around the system. The following points show that it is crucial to minimise cable harness on SUNSAT structure.

- The spaces for these cables are limited and the number of cabling must be minimised [1].
- The number of cables running through the satellite increases the chances of any single line failure, considering the effect of vibration during launch on the soldered connection [3].
- It is has been found difficult and time consuming to trace or diagnose a problem during system testing.
- These cables increase the weight of the satellite.
- Cables must be minimised to increase satellite's flexibility and reliability

1.3 Report Overview

The goal of this report is to describe the design and implementation of CAN node(s). Chapter 2 shows how CAN node(s) design fits in the whole picture of SUNSAT by describing the environment under which it (CAN node(s)) will be operating. The CAN node's objectives are explained by giving the systems specifications.

Chapter 3 gives the evaluation of several SBAs. CAN technology is discussed and compared to other serial bus architectures, to clarify the reasons of using it (CAN) as the preferred method of design.

Chapter 4 gives the design implementations of CAN node(s). This chapter explains how the system was built based on its schematic diagram. The functionality and operations of all the components used in the design are discussed in-detail.

Chapter 5 deals with the testing, debugging of the hardware and software. It highlights the problems encountered during the testing and then explains the solutions that were made to fix all the problems.

Chapter 6 explains the integration of the final software and the modified hardware, in order to test the whole principle of CAN node(s) design.

Chapter 7 gives the results of the project. This chapter explains the successes as well as the shortcomings of CAN node(s) design.

Chapter 8 discusses the conclusion and the recommendations for an ideal system. The recommendations are explained to show how the ideal CAN node(s) could be implemented.

Chapter 2

2 Design Considerations

The design of CAN node(s) is carried-out with the considerations of the environment of operation. In this case, the telecommand system together with all other systems to be interfaced with, act as the environment of consideration.

This chapter begins by explaining the objectives for the CAN node(s) by defining the system's specifications. It then focuses on the thorough description of the telecommand system, because control of all subsystems on the satellite originates in the TCMD system (as highlighted in chapter 1).

2.1 Objectives

To emphasis the objectives for this project the following system's specifications are considered:

- This system should be able to provide SUNSAT with the reduction of a number of complex wires into a serial bus connection.
- It should be able to provide flexible connectivity, thereby improving SUNSAT testability.
- The reliability of the system should include elimination of complex failure modes.

2.2 The TCMD System's Functional Definition

SUNSAT-1[2] structure comprises of 11 interlocking aluminium trays. See the SUNSAT block diagram in figure 2.1. As it can be seen here, the telecomand (TCMD) forms a physically separated unit on its own. Hardware interfacing between the different trays is implemented on two opposite sides of the structure using wiring harnesses [4].

[2] The words SUNSAT-1 or SUNSAT-2 are used to distinguish the existing SUNSAT micro-satellite from the future generations of SUNSAT micro-satellite respectively.

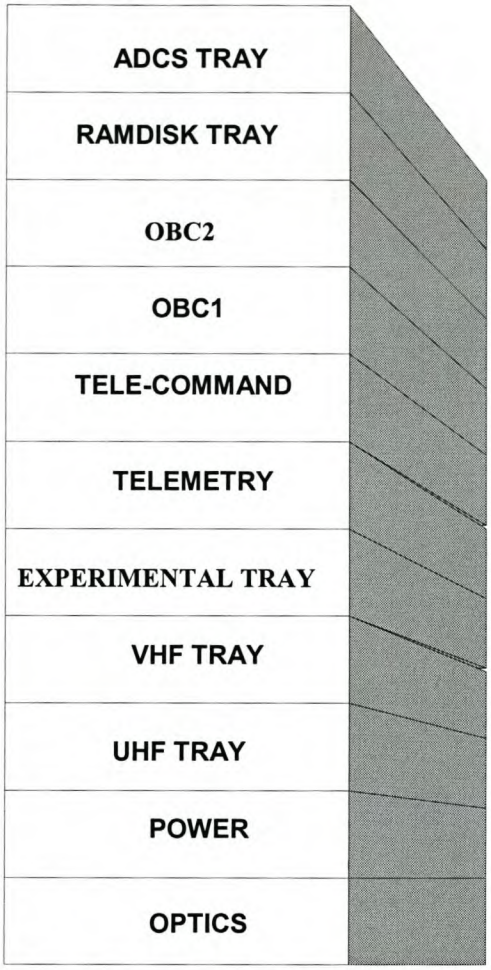


Figure 2.1: Block representation of SUNSAT layered structure

The telecommand module of SUNSAT is responsible for the configuration of subsystems, controlling power switches of modules, selecting different paths for data flow (to and from the satellite and other systems), or choosing operation modes, [2] (Only few responsibilities are mentioned here).

Typically, a command in the form of a binary data stream is formatted in the ground station and transmitted to a number of receivers and demodulators in the satellite via the RF communication link.

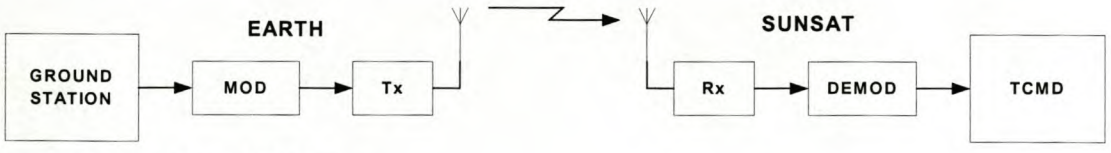


Figure 2.2: Function of TCMD

In the satellite, the command decoder examines incoming data to determine if commands are present. When a command is detected and verified, the TCMD responds by setting one or a number of command latches. The following diagram (figure 2.3) shows the TCMD system's components.

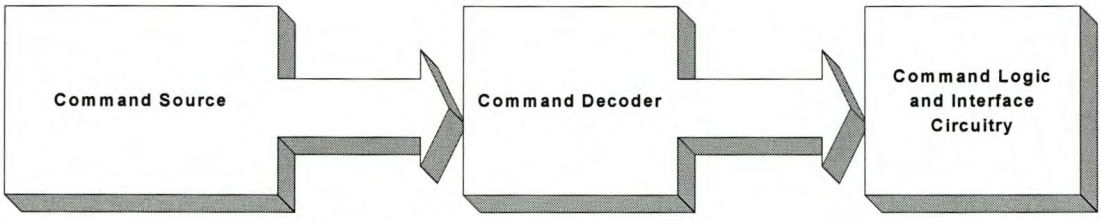


Figure 2.3: The three main TCMD functional components

These TCMD components are discussed in full in the following subsections.

2.2.1 Command Source

The RF up-link is not the only source of command in TCMD system; the commands can also originate from the on-board computers and the test port (which is used for test purposes). See figure 2.4 for these command sources.

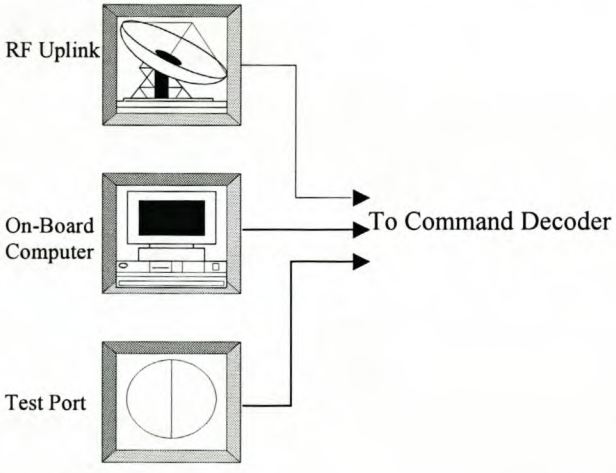


Figure 2.4: Command Sources

The RF link is the most important source of commands, since it acts as a gateway to the satellite [5]. These commands are received and demodulated by 1200-Baud and 9600-Baud modems implemented on the TCMD system (See figure 2.7 in page 10 for a complete TCMD system).

On SUNSAT the On-Board Computers (OBC1 and OBC2) control the telecommand system for most of the mission, as the satellite spends majority of its orbit being out of sight of the ground station. The software uploaded to the OBCs keeps track of all hardware functions required during flight. Each OBC can use either its own dedicated, eleven wires parallel data interface to the TCMD system, or use the SUNSAT Serial Bus (SSB) [4]. The SSB is common to both OBCs and the microprocessors of the TCMD and the power system [4].

The hardline test interface (Test Port) is one of the command sources, which is not used during the normal operation of a spacecraft. It serves as an easily accessible port to the TCMD system, which can be used to issue commands, monitor debug parameters, and verify system performance during the testing, integration and checkout phases of the project [5].

2.2.2 Command Decoder

Consider the following figure (figure 2.5), where the function of the command decoder is broken into three different tasks, namely the **commands source arbitration**, **command decoding** and **command message validation**.

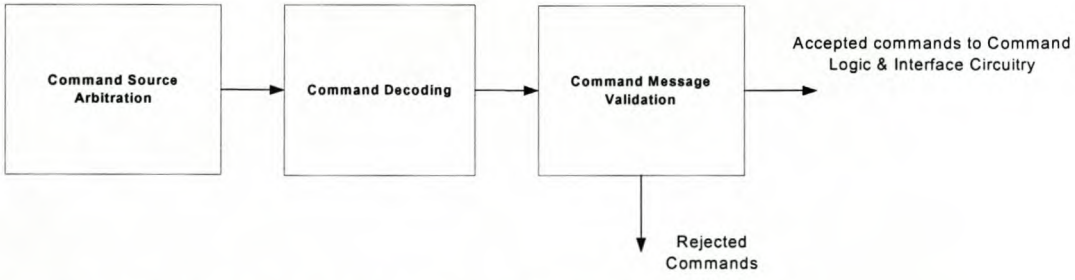


Figure 2.5: Three functional units of the Command decoder

The **command source arbitration** unit is used to make selection of the different sources from the command source component. The **command-decoding** unit decodes the received command to reproduce the original command message. The **command message validation** is made up of the reception of the synchronisation code, checking command message length; an exact match of the spacecraft address; an exact match of

any fixed bit patterns or unused message bits; and lack of error detecting using error check polynomial code [5]. The errors that could not be fixed by this unit are rejected.

2.2.3 Command Logic and Interface Circuitry

A typical TCMD system provides two types of command outputs: discrete and serial (See figure 2.6 in the next page). Discrete commands have a fixed amplitude and pulse duration and consist of two basic types:

- High-level discrete commands: a +28V, 10 to 100 ms pulse used to drive a latching relay coil or fire an ordnance device [5].
- High-level discrete commands: An open collector or 5V pulse typically interfacing with digital logic [5].

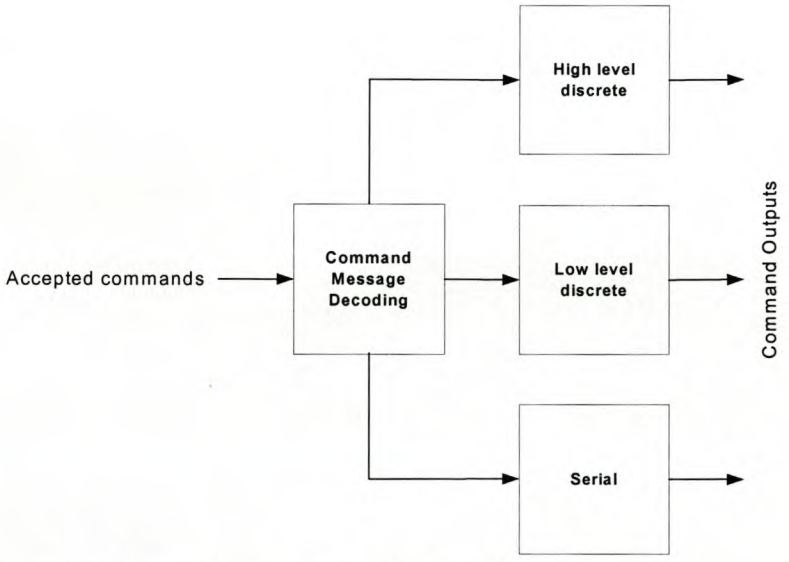


Figure 2.6: Command Logic and Interface Circuitry

A serial command is a three-signal interface consisting of a shift clock; serial command data; and a data enable, used to indicate that the interface is active. The interface circuitry has to adapt the logic signals of the command message decoder to match the requirements of the specific type of command [5].

2.2.4 Command Outputs

Consider figure 2.7 (shown below), which shows the complete block diagram of the TCMD system. The system implements two identical subsystems (subsystem 1 and subsystem 2) for redundancy purposes. Each subsystem has 232 switches (using latches and flip-flops) as command outputs [2]. Subsystem 1 can receive commands from two of the four telecommand modems (one 1200-Baud and one 9600-Baud) and both OBCs. Subsystem 2 can receive commands from the remaining two-telecommand modems, as well as from both OBCs.

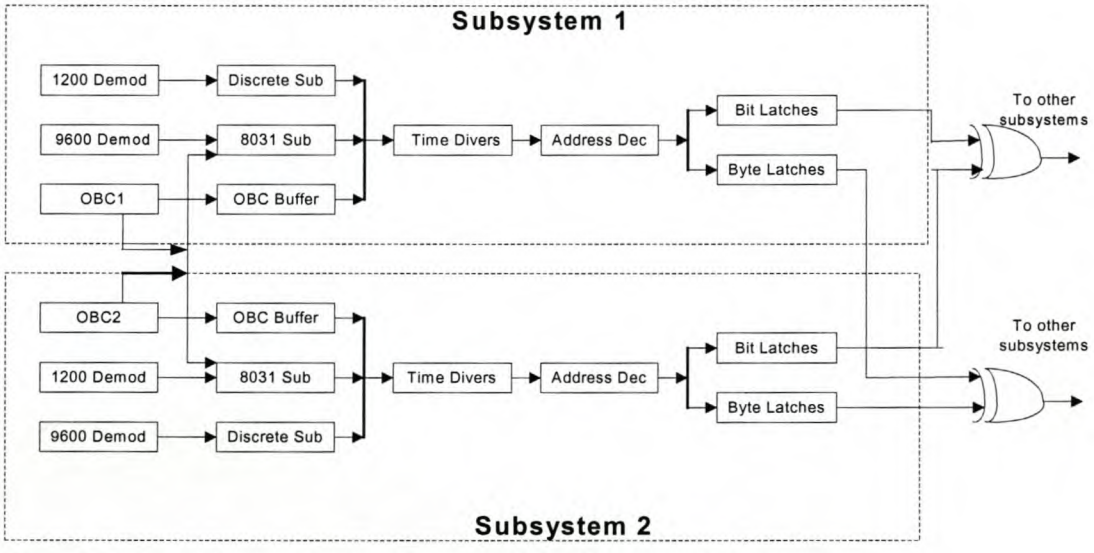


Figure 2.7: Block Diagram Layout of the Complete Tele-command System

The data flow entering the satellite via the modems is continuously monitored for validity by the discrete subsystem (indicated as **Discrete Sub** in figure 2.7). The functionality of the 8031-based subsystem (**8031 sub**) is equivalent to that of discrete subsystem and also connects the back-up serial connection via the SUNSAT serial bus (SSB) [4]. The **OBC Buffer** forms a parallel buffer to allow control from the OBCs. Time diversity (**Time Divers**) is used to force any command to be received twice before any action is taken [4]. The address decoding (**Address Dec**) is done to select a specific latch (see the next section for more details) to be switched. Each subsystem (subsystem 1&2) has 232 switches of which 128 are bit addressable (**referred to Bit Latches**) and 104 are byte addressable (**referred to Bit Latches**). The switches are combined using exclusive-or (XOR), this is done so that if an output of subsystem 1,

for example, gets stuck at one logic level, the output of the XOR-gate can still be changed by altering the other input of the gate using subsystem 2.

2.3 Attachment of CAN node(s) to the commands outputs

The TCMD system implements two types of latches (as mentioned in the previous section) namely: Bit latches and Byte Latches. Due to layout restrictions only 15x8 bit latches and 14x8 byte latches are implemented to provide the 232-telecommand output switches [4]. In the case of bit addressable latches, every single bit can be switched without affecting the other 7-bits i.e. each line can be set/reset individually. In the case of the byte addressable latches, all 8-bits of the byte can be controlled at once. In SUNSAT-1 the 232-telecommands output signals from the latches are transported to the other SUNSAT systems via dedicated cables (or wires).

As a result the CAN node(s) is going to be attached at commands outputs (see figure 2.8 below) to ensure that these cables are minimised for the reasons mentioned in Chapter 1. Figure 2.8 (in the next page) shows that CAN node(s) serve as the communication interface between TCMD and other SUNSAT subsystems. The TCMD data will be transmitted from the TCMD system to other SUNSAT systems using CAN protocol.

2.4 Conclusion

In this chapter the TCMD system was explored thoroughly in order to define the environment under which CAN node(s) is going to be operating. Consequently the chapter has clarified how the CAN node(s) design fits in the whole picture of SUNSAT. In Chapter 1&2 the problem statement and the environment in which the CAN node(s) will be operating have been clearly defined. The following four chapters (chapter 3,4,5,6) will be discussing the implementation of CAN node(s).

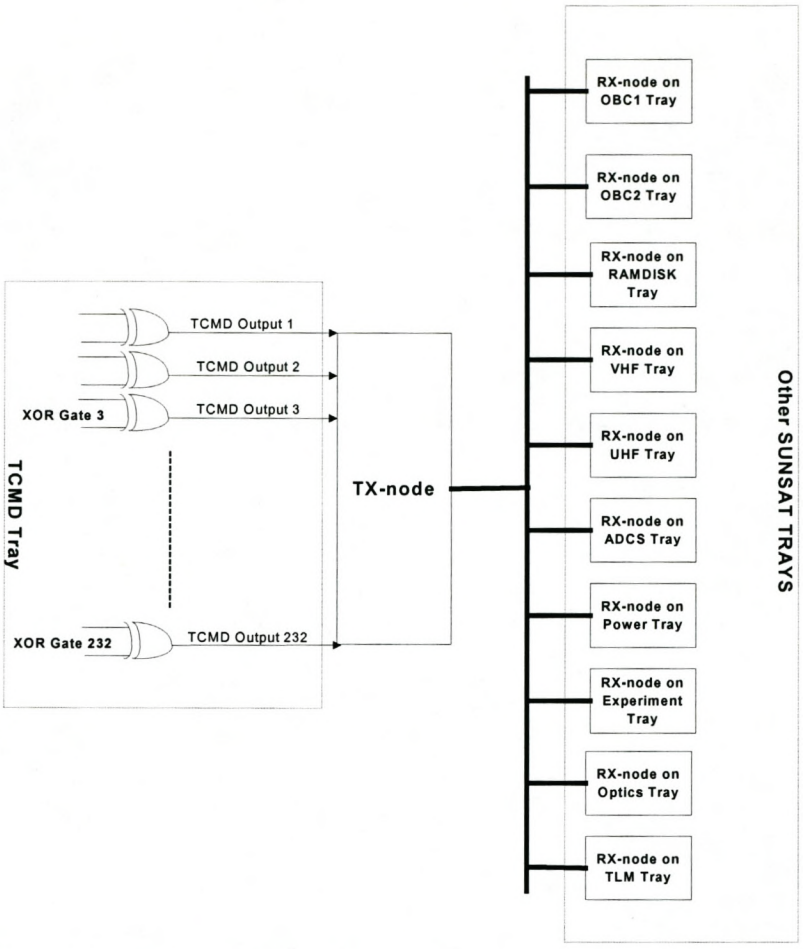


Figure 2.8: Attachment of CAN nodes to the commands outputs

Chapter 3

3 Evaluation of Various SBA technologies

This chapter gives the evaluation of several SBA technologies in order to distinguish and support the preferred CAN bus technology from other SBA technologies. Three major categories of serial bus technologies are considered in this evaluation, namely: **SUNSAT-1 buses**, **Consumer buses** and **Field buses,** depending on their relevance and link to CAN bus technology as well as to this study. CAN bus technology falls under the Field bus category, and as a result this category will be described in detail as compared to the other categories.

Most of this evaluation is done using the thesis of Jan-Albert Koekemoer, "*Investigation of a command and Data Handling Architecture for the SUNSAT-2 micro satellite*", (November 1999) as the source document.

3.1 SUNSAT-1 buses

Two of the prominent serial buses in use on SUNSAT-1 are discussed namely: SUNSAT Serial Bus (**SSB**) and SUNSAT Instrumentation Bus (**IBUS**).

3.1.1 SSB

The **SSB** was developed for low speed (9600-Baud) and is implemented to facilitate the transmission of dwell TLM packets from the 8031 TLM system processor to the OBCs. It also sets up and receives status information from the power system's 8031-processor and sets commands from the OBCs. The architecture is a half-duplex bus, and the access method is master-slave. This means that either of the OBCs, but not both simultaneously, is the master; and the TLM, power system, and TCMD processors are the slaves [5].

3.1.2 IBUS

The IBUS is a higher speed (19.2KBaud) serial bus and its purpose is to allow the OBCs to send commands and receive data from the tipmass[3] (not shown in SUNSAT block diagram figure 2.1) instruments and school experiments.

Half-duplex communication is used with multiple access and collision detection. To minimise multiple collisions, each **IBUS** node has a unique and fixed delay-time. It must wait for the specified period after the previous bus activity before beginning transmission. The proprietary protocol uses one header byte to indicate both the target node and the number of data bytes transmitted up to a maximum of 31 bytes [5].

3.2 Consumer buses

The two major consumer buses are discussed, namely: Universal Serial Bus (**USB**) and **IEEE-1394 or FireWire** bus.

3.2.1 USB

USB is a peripheral bus standard developed by PC and Telecom industry leaders that take plug and play of computer peripherals outside the PC enclosure. There are three components in any USB network: a host, functions or nodes, and hubs. There can be only one host in any USB network, and all wire segments are a point-to-point connection between the different components of the network.

The maximum bus speed on a USB version 1.0 network is 12Mbps. The physical medium is implemented with a differential driver supporting bi-directional half-duplex operation over a maximum cable length of five meters between any two components.

[3] Tipmass- indicated as the BOOMTIP in SUNSAT block diagram (figure 1.1)

3.2.2 FireWire

The 1394 standard of the **FireWire,** supports a peer-to-peer network with point-to-point signalling environment, therefore no host is required. Nodes on the bus may have several ports usually three, and each acts as a repeater, retransmitting any packets received by other ports within the node. The standard also defines two bus categories: backplane (12.5, 25 and 50Mbps) and cable (100, 200, and 400Mbps). A multi-master protocol allows multi-speed transactions on the same bus, due to the point-to-point signalling method.

3.3 Fieldbuses

The set (or category) of serial buses aimed at general industrial automation are called fieldbuses.

The sets of serial buses aimed at general industrial automation are called fieldbuses. Four types of fieldbuses are discussed in the following subsections, namely **Bitbus, Profibus, LonWorks**, and **CAN**.

3.3.1 Bitbus

Bitbus has been developed by Intel at the beginning of the 1980s as an open communication system and is optimised for the transmission of short real-time messages. Bitbus was officially standardised as IEEE-1118 in 1991.

It has a master-slave structure where up to 28 slaves can be addressed per segment. Each slave has its own network address, which makes it uniquely identifiable in the network. IEEE-1118 also makes provision for broadcasting and multitasking messages from the master, i.e. sending messages to all or only a selected group of slaves. The bit-rate of this bus is either 62.5Kbps or 375Kbps. Its total message length is 248 bytes.

3.3.2 Profibus

Profibus is an international, open fieldbus standard developed in the late 1980s in Europe. It has since been standardised as EN50170 and EN50254. The profibus family consists of three compatible versions: Profitbus-DP for high-speed applications, Profibus-PA for process automation – mainly chemical applications – and Profibus-FMS for general-purpose applications. Both the DP and PA versions implement the bottom two network layers, while the FMS version adds layer seven functionality as well. Profibus-FMS also permits data communication and power over the same bus according to international standard IEC1158-2. RS-485 is the transmission technology most frequently used by DP/FMS; and a unique transmission speed between 9.6Kbps and 12Mbps can be selected for all devices on the bus.

The bus access control used is a mixture of multi-master and master-slave: when the network is set up, certain nodes are designated masters and other slaves. A slave can only respond on the bus when asked to do so by a master. A master can send messages to slaves and other masters in a broadcast or multicast fashion only when it holds the bus access rights or token. The token is passed between the masters in a logical ring. A maximum length of 224 bytes per message and up to 32 stations in each network segment is allowed. The following figure illustrates this process.

3.3.3 LonWorks

The Echelon Corp. developed **LonWorks** as a complete, full-featured solution for generic control networks. LonWorks supports a hierarchy of buses, a variety of message transmission methods, encryption, and authentication services, which simplify network design and installation. Third-party vendors provide a variety of LonWorks-related products and consulting services.

LonWork's Medium Access Control (MAC) is form of carrier sense multiple access with collision avoidance (CSMA/CA). In this MAC, a node ready to transmit on a bus waits for it to become idle before sending a message. If multiple nodes begin transmitting nearly simultaneously, the messages will collide. If the bus is not idle or

a collision occurs, the transmitting node backs off and retries transmission after waiting for a random number of time slots. This process continues until the message is successfully transmitted. Collisions can be avoided by increasing the number of random slots, which reduces the probability of collisions.

3.3.4 CAN

The Controller Area Network (**CAN**) is defined as a serial communication protocol, which efficiently supports distributed real-time control with a very high level of security [5]. Forced by the increasing number of distributed control in cars and the increasing wiring costs of car body electronics, the availability of a powerful and reliable serial data communication system for the exchange of messages between the different control units was becoming urgent. This was the starting point for BOSCH, a main provider of electronic car equipment to develop the CAN protocol and standardised it as an international ISO standard.

CAN is a multi-master bus which support multicasting and broadcasting of massages from one node to a number nodes. Today, there are more available protocols that can be implemented in form of stand-alone controllers or integrated into a micro-controller. The following diagram shows a typical CAN bus connection.

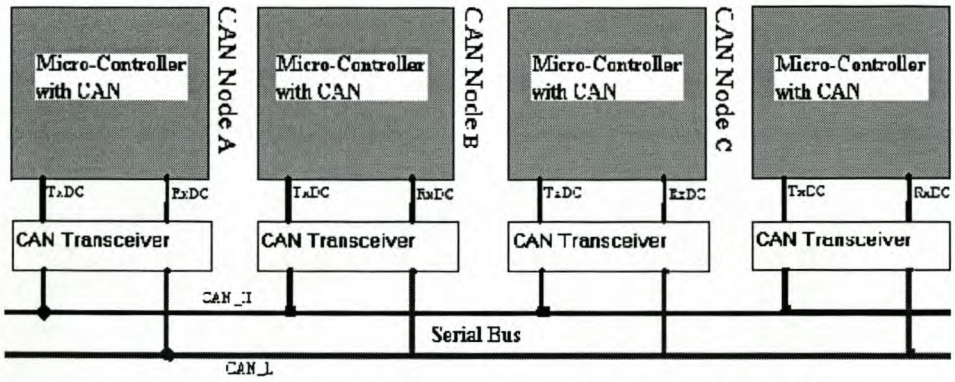


Figure 3.1: The micro-controller with CAN module

The micro-controller with CAN module[4] is used to implement CAN technology that will transport TCMD output data to other subsystems in the satellite. The serial bus is

[4] CAN module - The interfaced on-chip CAN peripheral.

used to link the CAN nodes to enhance communication protocol between them. The CAN Transceiver is used to drive the communication between the Serial Bus and the micro-controller.

In the CAN protocol, the bus nodes do not have a specific address, because the transmitted massage contains the identifiers that indicate the massage content and the priority of the massage. The original CAN specifications (Versions 1.0, 1.2 and 2.0A) defined the message identifier as having a length of 11 bits giving a possible 2048 message identifiers and have been referred to as "Standard CAN". The specification has since been updated (to version 2.0B) to remove possible limitations. CAN specification Version 2.0B (often referred to as "Extended CAN") allows message identifier lengths of 11 and/or 29 bits to be used (an identifier length of 29 bits allows over 536 Million message identifiers) [6].

3.4 General Comparison

Although the two proprietary SUNSAT buses are operational in space, neither of them implement sections of the ISO networks layers in hardware. The software protocols therefore incur additional overhead. The master-slave architecture of the SSB makes the bus access restrictive.

FireWire and USB are both elaborate protocols aimed at high-speed applications. Full utilisation of the capabilities of the USB standard requires a version of the Microsoft operating system that supports USB at the operating system level. This renders it unsuitable for a satellite application. The start-up cost for a FireWire network is higher than that for USB, and the official specification document for the former can only be obtained from the IEEE at extra cost.

Of the four fieldbuses presented, only CAN has a truly multi-master capability, allowing considerable freedom in network design and operation. All have sufficient speed capabilities, and allow enough nodes per segment for a microsatellite application without the need for repeaters. LonWorks has a huge start-up cost compared to the others, and does not provide a very wide range of application ICs. Of

all the buses presented, CAN has the lowest allowable data length per message – only 8 bytes.

In terms of hardware implementation of the fieldbus protocols, three out of four buses provide more than the bottom two layers of the ISO network structure. This negatively influences bus access times, since messages take longer to move up and down the protocol stack. All the commercial buses presented are aimed towards industrial or consumer applications. For space applications, a degree of freedom is therefore needed to add a customised application layer to the given protocol stack and this is offered only by Profibus and CAN. In summary, it can be seem that the general reason above indicate the superior suitability of CAN as the most adequate architecture.

3.5 Conclusion

Several SBAs where evaluated in this chapter and a general comparison was made based significantly in design goals, resulting in differences of speed, efficiency, development support, and features. In summary, it can be seem that the general reason above indicate the superior suitability of CAN as the most adequate architecture.

Chapter 4

4 The CAN node(s) Design

This chapter describes the design implementation of the CAN node(s) i.e. designing the schematic diagram (see SCHEMATIC 1 in Appendix A) and putting together components. This chapter explains how the system was built based on its schematic diagram. The functionality and operations of all the components used in the design are discussed in-detail.

4.1 Design implementation of CAN node(s)

In application, eleven linked CAN nodes are needed to transmit data from TCMD to the other subsystems i.e. each CAN node in each subsystem of the SUNSAT structure. See figure 4.1 in the next page for the illustration of CAN node implementation. CAN node(s) is divided into two components namely, TX-node and RX-node. The TX-node is attached to the TCMD tray to collect data. The data is then transmitted serially to the RX-node. The RX-node is a collective name used for all the nodes that are connected to the other ten trays in the SUNSAT structure. In the figure (figure 4.1) each valid command consist of 72 bits that are sent to the satellite twice before any execution. Therefore in order to change the state of a switch or a group of switches, a 144-bit data stream must be transmitted to the satellite by the ground station.

However, to demonstrate the design principle, the TX-node and one RX-node are used. These two nodes have the same hardware, but they implement different software. See the block diagram illustration in the next page (figure 4.2). This figure shows the data path between the major components of CAN node(s). The schematic diagram is supplied in appendix A and is designated as SCHEMATIC 1. The major components, together with the other components used in the design are thoroughly described in the following sections.

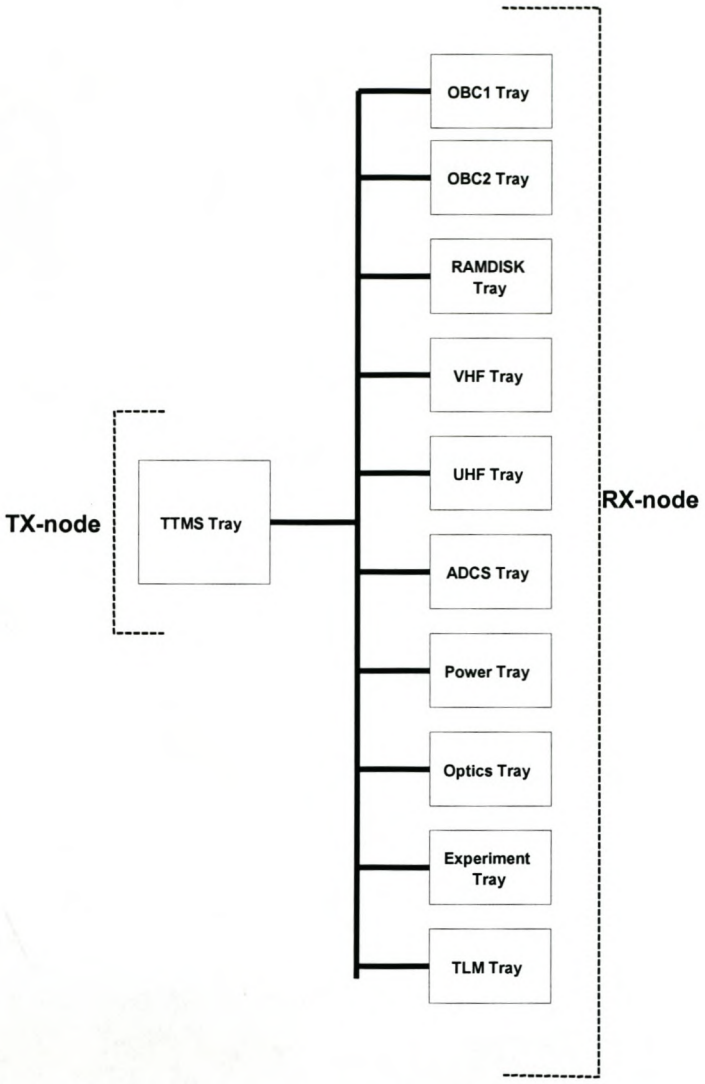


Figure 4.1: The architecture of a complete CAN node(s) implementation

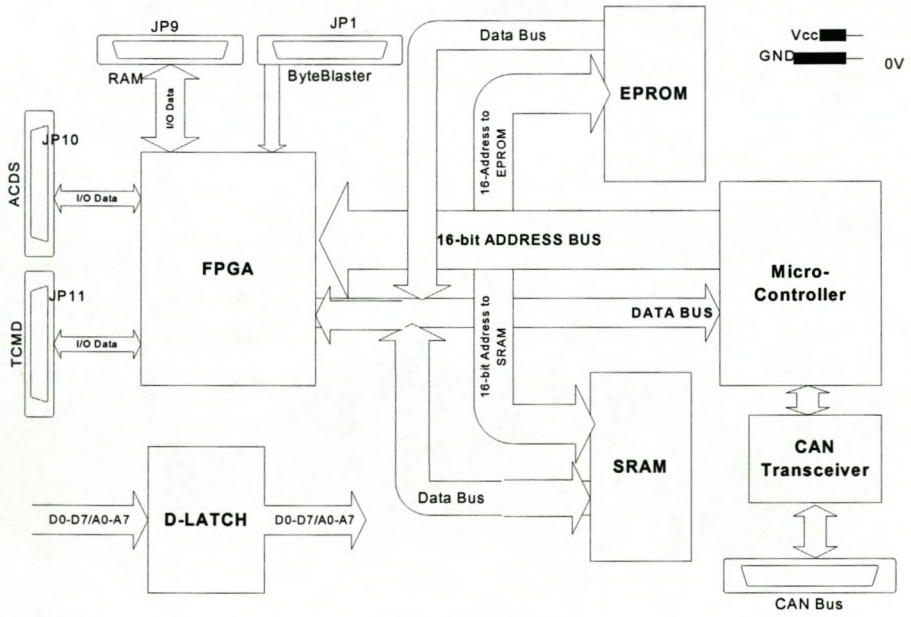


Figure 4.2: Block diagram for CAN node(s) circuit

4.1.1 The Micro-controller with CAN

4.1.1.1 Introduction

The SIEMENS C505C micro-controller with CAN module is used in this design. The C505C micro-controller is the member of the SIEMENS C500 family of 8-bit micro-controllers that are compatible to the standard 8051 micro-controllers. The word micro-controller will be used interchangeably with C505C. CAN module is one of the major on-chip peripherals which, is interfaced on the C505C. This interface is made of two major blocks namely the **CAN controller** and the **internal bus interface**.

The **CAN controller** is the functional heart which provides all resources that are required to run the standard CAN protocol (11-bit identifiers) as well as the extended CAN protocol (29-bit identifiers). Fifteen different **Message Objects (MO)** can be used independently by the C505C micro-controller, and each message has its own specific identifier. The CAN controller provides a sophisticated object layer to relieve the CPU of as much overhead as possible when controlling many different message objects (up to 15). For the C505C to implement the physical layer, external components have to be connected to the C505C e.g. CAN transceiver, serial bus and etc. (refer to SCHEMATIC 1 in appendix A). The **message object** is the primary means of communication between the micro-controller and the CAN module.

The internal bus interface connects the on-chip CAN controller to the internal bus of the micro-controller. The internal CAN module is located in the external memory address area at addresses $F700_H$ to $F7FF_H$ (for more information, see the memory organisation illustration diagram in appendix D1). The CAN module registers can therefore be accessed using MOVX instructions with addresses pointing to the respective address areas. As a result the registers and data locations of the CAN module are mapped to a specific 256-byte wide address.

The access to this special memory space requires the modification of register XPAGE to the value of $F7_H$. Furthermore, bits XMAP0 and XMAP1 in register SYSCON must be forced to "0". The register XPAGE provides the upper address byte for accesses to

CAN controller with MOVX @Ri instructions. If the address formed by XPAGE and Ri points outside the CAN controller address range, an external access is performed. For the C505 the content of XPAGE must be $F7_H$ or FF_H in order to use the CAN controller. This allows 8-bit addressing of CAN [7].

4.1.1.2 Application of the C505C

In the schematic diagram (see SCHEMATIC 1 in appendix A), the C505C is represented by U2 and in the block diagram (see figure 4.2) is labelled micro-controller. The C505C communicates to the SRAM and the FPGA using address-decoding technique, since both these devices are using the RD (for reading) and WR (writing) signal of the C505C. In this kind of connection the FPGA is treated as a second external data memory. See how C505C operates when using these two signals (RD and WR) to communicate with external data memory in the Siemens datasheet supplied in appendix D1.2. Through address decoding the C505C is able to specify which data memory it is communicating with between SRAM and FPGA. The application of this technique is explained in the respective sections for SRAM and FPGA. C505C also connects to the serial bus, via the CAN transceiver, through (RXDC) P4.1 (for receive) and (TXDC) P4.2 (for transmit). A clock source is a 12MHz crystal connected to XTAL1 and XTAL 2 pins of the C505C.

4.1.2 EPF6016 (FPGA)

Gate Arrays have historically been used for high volume designs [8]. ALTERA Programmable Logic Devices (PLD) is an ideal alternative for prototyping gate array designs and for high volume production. FPFA (Field Programmable Gate Array) is a type of PLD that offers a high capacity, high speed, and cost competitive solution for prototyping and production. ALTERA EPF6016 144-3 (FPGA) is used in this design. It is the member of the ALTERA FLEX 6000 device family with 144 pins. The EPF6016 implements the following design flows, namely the (Design Entry, Design Processing, Design Verification and Device Programming)[5]. VHDL[6] is used as the

[5] Design Entry- Software supports, such as ALTERA's MAX+PLUS II can be used to enter FPGA, using the schematic or HDL design entries. Design Processing- the design can be compiled using

design entry for the EPF6016 (see Chapter 5 for more details). EPF6016 is represented by U1 in the schematic (see SCHEMATIC 1 in appendix A) diagram. EPF6016 will be used interchangeably with FPGA to represent ALTERA EPF6016 144-3 (FPGA).

The TX-node uses the EPF6016 to collect TCMD output data and send them to the C505C, whereas the RX-node uses the EPF6016 to receive data from the C505C. There are approximately 104 pins that are used as IO (input/output) pins and 40 pins are dedicated for configuration purposes. The data is collected and transmitted using the IO pins that are connected to JP9, JP10 and JP11 headers (see SCHEMATIC 1 in appendix A).

The pins I/O33 – I/O57 and I/O58 – I/O82 are connected to RAM Disk (JP9 on the schematic) and ADCS (JP10 on the schematic) headers, respectively. These I/O pins are used to output data to their respective trays (the RAMDISK and ADCS trays are used as examples). A 32.768MHz crystal is used to provide a clock to the EPF6016. The address range of 8000H- FFFFH is used for the FPGA. The address decoding for the FPGA is specified by the VHDL code (see chapter 6 for more details).

4.1.3 The CAN Transceiver

The Philips PCA82C250 is used as the CAN transceiver. As mentioned in Chapter 3, the CAN Transceiver is used to drive the communication between the Serial Bus and the micro-controller. In the schematic (see SCHEMATIC 1 in appendix A) the PCA82C250 is represented by U4 and it connects to C505C through the RXDC (receive) and TXDC (transmit) pins. It therefore connects to the physical bus (serial bus), via the CANH and CANL through the CAN Bus_DB9 (RD9) connector.

MAX+PLUS II compiler. Design Verification- the simulation for design verification can be performed in the MAX+PLUS II simulator environment. Device Programming- MAX+PLUS II creates files for device programming and configuration.

[6] VHDL- VHSIC Hardware Description Language

4.1.4 Serial Bus

Serial bus is a data transportation medium that is able to transmit two possible bit states (dominant and recessive[7]). One of the most common and cheapest ways is to use a twisted wire pair, in which the bus lines are called "CAN_H" and "CAN_L" and may be connected directly to the nodes or via a connector. There's no standard defined by CAN regarding the connector to be used.

4.1.5 The Program memory

The AMD Am27C512 chip is used as the program memory since C505C does not have an internal program memory. The Am27C512 is a 512 Kilobit (64 K x 8-Bit) CMOS EPROM (Erasable Programmable Read Only Memory) that is ultraviolet erasable programmable read-only memory. In the schematic (see SCHEMATIC 1 in appendix A), it is designated by U6. The device offers separate Output Enable (OE#) and Chip Enable (CE#) controls. CE# is connected to ground (low) and thus making the Am27C512 always enabled. To allow accesses to external program memory the signal PSEN (program store enable) from the C505C is connected to OE# as a read strobe (see Am27C512 switching waveforms in the datasheet in appendix D3).

4.1.6 The Data Memory

The SUMSANG KM681000B SRAM (Static Random Access Memory) is used for data memory. The KM681000B is designated by U3A in the schematic (see SCHEMATIC 1 in appendix A). The C505C was interfaced to the data memory and the FPGA through address decoding technique. The timing diagrams, which show different modes of operation for the SRAM, are described in Appendix D4 (KM681000B Family). In this design the SRAM is connected as follows: CE1 (P22) connected to A15 (P31) and CE2 (P30) connected to Vcc (5V[8]). This mode of

[7] The bus logic corresponds to a "wired-AND" mechanism, "recessive" bits (mostly, but not necessarily equivalent to the logic level "1") are overwritten by "dominant" bits (logic level mostly "0"). As long as no bus node is sending a dominant bit, the bus line is in the recessive state, but a dominant bit from any bus node generates the dominant bus state.

[8] 5V voltage is represented interchangeably by 1 and high, whereas 0 or low represents 0V voltage.

connection allows the SRAM to be controlled by CE1 and the address. Consider the table in appendix D4 for SRAM operation. According to the table mentioned above it means that SRAM is selected at the address range of 0000H-7FFFH, because in this range A15 is low (0) and therefore CE1 is also low (0). The write and read operations depend on the software that is on the C505C.

4.1.7 Octal D-latch

The address and data lines of the program and data memory chips connect to the respective lines of the C505C. The least significant bits of the address lines share port 0 (P0) with the data bus lines whereas; the higher bits are connected on port 2 (P2). In the block diagram (figure 4.2), the list significant address bits are represented by A0-A7, whereas the data bits are represented by D0-D7. In order to implement this, the Philips 74HC373 Octal latch is used for providing the distinction between the address and the data lines of P0.

The ALE (Address latch Enable) line from the C505C is connected to the LE (Latch Enable) of the latch. OE (Output Enable) is connected to 0V (low). In this configuration the latch enters two modes (1) the transparent mode when ALE is high (2) "latch and read register" mode when ALE is low. See the function table in appendix D5 (74HC/HCT373 Octal D-type transparent latch datasheet) for the operation specifications of the D-latch.

4.1.8 Miscellaneous Components on CAN node(s)

The other components such as the LED, Switches, extra headers and TEXAS Instrument's Max232, are used for testing and debugging purpose. The functions for some (like switches, LED and etc) of these components are discussed in detail in the next chapter.

4.2 Conclusion

The TX-node and the RX-node were built using the bottom-up build method. The components were placed on the board one-by-one. Each component was tested individually for its basic operation. The following chapter (Chapter 5) discusses a higher level of hardware and software testing.

Chapter 5

5 Hardware and Software Test Performance

This chapter deals with the testing and debugging of the hardware and software to ensure the basic functionality of the system. The chapter starts by explaining the method being used for testing where the system had been broken down into phases. To perform hardware and software tests, the data bus system was broken into five phases (see figure 5.1).

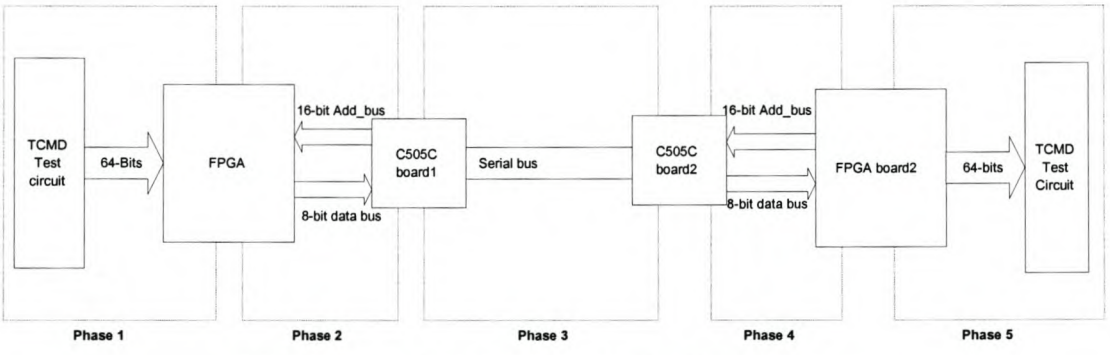


Figure 5.1: The CAN bus system broken down into phases

These phases are described in detail in section 5.2. During the tests it emanated that there had to be some number of changes that have to be made on the original circuit (refer to SCHEMATIC 1 in appendix A).

All the problems encountered during testing and necessary changes that have been applied on the hardware are discussed in detail in section 5.2.

5.1 Tests Performed

This section describes the tests that were performed, with the main focus placed on the hardware performance.

5.1.1 Phase1

This phase illustrates the link between the TX-node's FPGA and the data lines that are coming from the TCMD board into the data bus system. The figure below (figure 5.2) shows the schematic representation of the TCMD emulator circuit which, was designed for testing purposes.

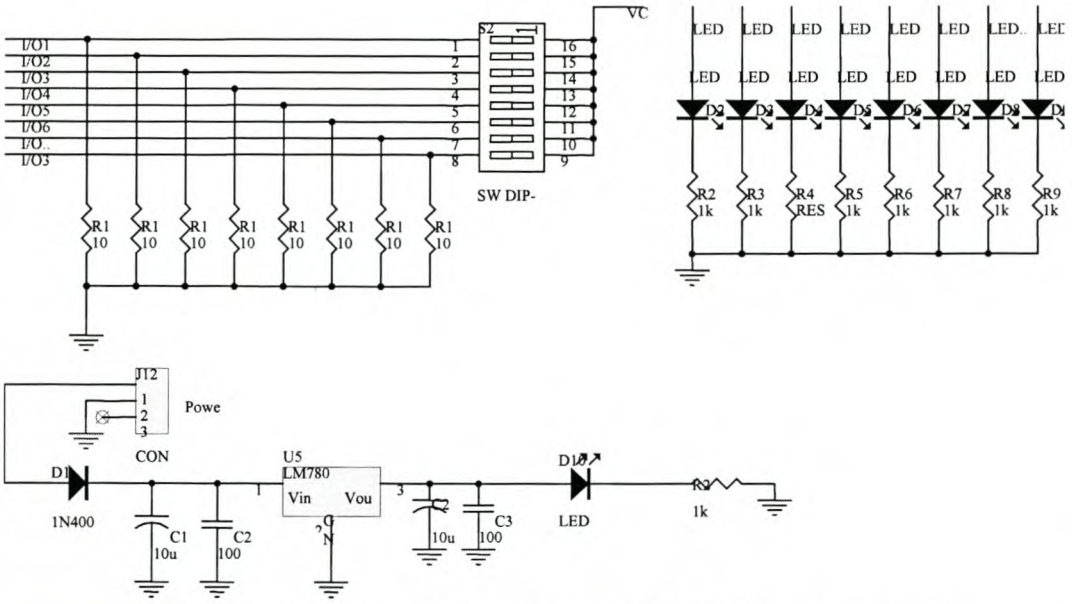


Figure 5.2: The schematic representation for the TCMD emulator circuit

The TCMD emulator circuit is composed of 64 switches, 64 LEDs (Light Emitting Diodes) and the power circuit. Switches represent the Tele-command data outputs and the LEDs provide the indication of which data has been received. For example, the 25 switches can be used to represent 25 TCMD output data going to the ADCS tray. In this case 64-bits are latched in the FPGA at a time. The FPGA package them into eight 8-bits bytes. These bytes are sent out of the FPGA on request. In this phase (Phase 1), three switches are used to request bytes by sending 3-bits address combinations to the FPGA. This is done by manually setting individual switches on or off. One switch is used as a chip-select signal which, is connected to pin 114 (labelled as CS1) of the EPF6016 (U1 in SCHEMATIC 1 in appendix A). See the VHDL code for the testing of this phase in Appendix C3.

In the example code 64-bits are latched to the FPGA. The above mentioned VHDL code is composed of two processes namely the Load_reg and the Write_out. The Load_reg is responsible for latching 64-bits which are assigned as FPGA_read. At each clock (clk) edge 8-bits are latched into eight 8-bits registers (reg0...7). During Write_out process reg0...7 are stored into temporary register called the FPGA_write which then writes the output to the data_bus port. The data_bus was connected to the LEDs of the TCMD emulator circuit via the eight free I/0 pins of the FPGA.

5.1.2 Phase 2

Phase 2 describes the communication between the TX-node's FPGA and C505C. The principle of phase 2 is similar to phase 1, except that in this case the micro-controller is used to request data from the FPGA, by sending a 16-bit address instead of the TCMD emulator switches. See the C code for this operation in Appendix B3. The VHDL code with these implementations is supplied in appendix (Appendix C2).

The C code mentioned above is also explained by the flowchart in the next page (figure 5.3). The address decoding for the FPGA is applied by connecting A15 to CS1. As mentioned in Chapter 4, the address range of 8000H-FFFFH is used to select the FPGA, since in this range A15 is high. The FPGA writes out data to the data bus when the C505C enters the read mode (refer to Chapter 4). The data from the FPGA is stored in the eight 8-bit buffers called DB0...DB7. Each of P3.1 to P3.5 was connected to a switch and the toggling of these pins helped to view the display of the corresponding data bytes on the LEDs.

5.1.3 Phase 3

This phase describes the communication between the C505C of TX-node and C505C of RX-node. Phase 3 has been identified as the most important phase in this project, since it describes the implementation of CAN technology. As a result its implementation will be described in Chapter 6 (CAN Software Implementation).

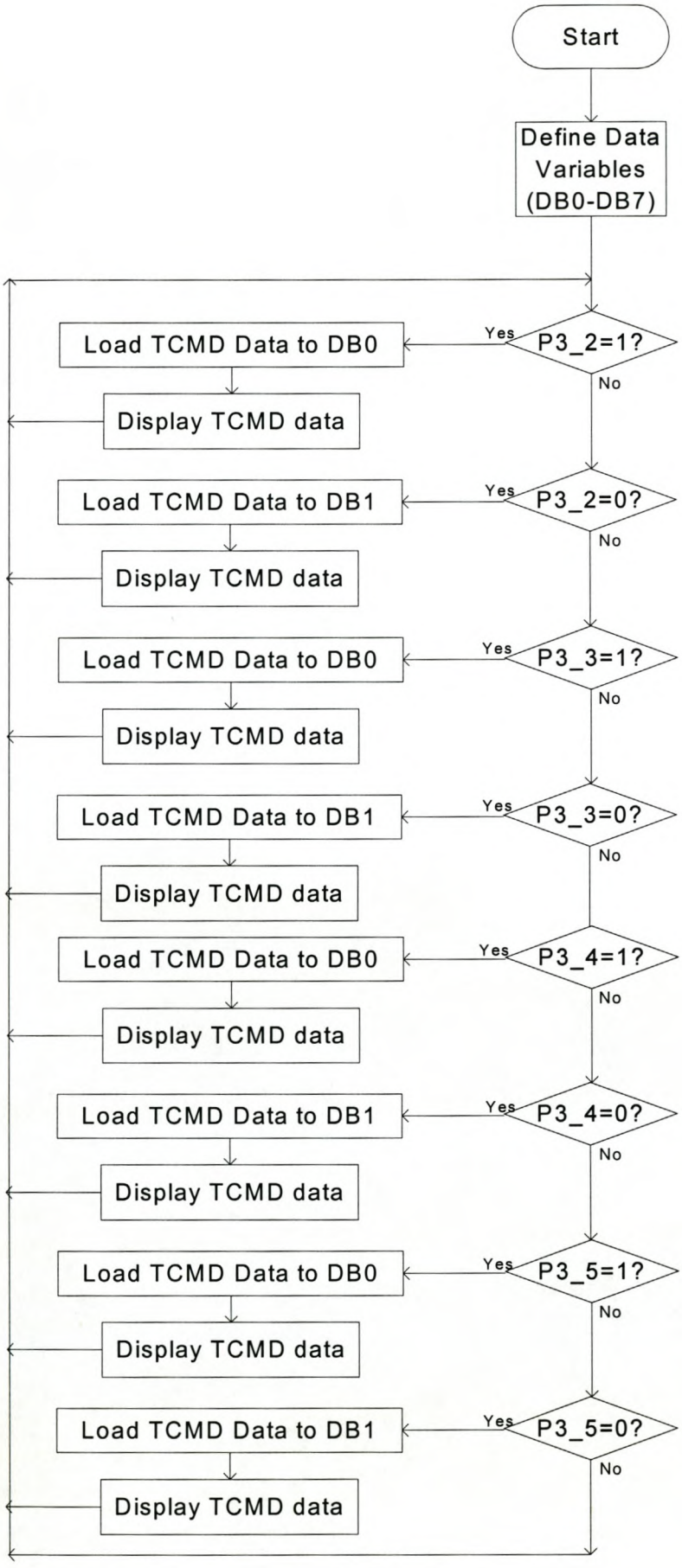


Figure 5.3: Flowchart representation for Phase 2

5.1.4 Phase 4

This phase describes the interface between the RX-node's FPGA and C505C. The software applications of phase 4 are basically similar to those of phase 2. The difference is that phase 2 is on the TX-node and phase 4 is on the RX-node.

5.1.5 Phase 5

Phase 5 describes the communication between the RX-node's FPGA and the LEDs of the TCMD test circuit (or a specific tray). Phase 5 software applications are almost similar to those of phase 1. The difference is that phase 5 is on the RX-node and phase 1 is on the TX-node.

5.2 Changes made on the original circuit

Two changes were made on the ALTERA EPF6016 FPGA, namely the **Configuration** and **conventional** changes. These changes are discussed below.

5.2.1 Configuration

Configuration changes can be defined as the changes that are performed on the configuration pins of the FPGA. A PC using the ByteBlaster parallel port download cable configures the EPF6016. The ByteBlaster cable has a 25-pin male header that connects to the PC parallel port. The cable provides two download modes namely, Passive Serial (PS) mode and Industry-standard Test Action Group (JTAG) mode. A 10-pin female plug connects to the ByteBlaster header (JP1 in the SCHEMATIC 1 in appendix A) using one of these modes. For further information on the ByteBlaster download cable (see Appendix D2.1). In this design the JTAG mode was used (see SCHEMATIC 1 in appendix A), but unfortunately the Flex6000 (in this case EPF6016TC) devices do not support this mode. Alternatively, PS mode is used to configure these devices. See appendix D2.1 for JTAG and PS connections.

5.2.2 Conventional

Conventional changes can be defined as the changes that are made to assemble the pins according to the physical component (FPGA). The design's schematic diagram (SCHEMATIC 1 in appendix A) was implemented using Protel software package. The Protel's components library had EPF6016TC144-3 FPGA with the pin arrangements that were different from the real component. Please refer to appendix D2.2 for the attached document, which shows the correct pin arrangement and numbering from the ALTERA datasheet. The following figure (figure 5.3) provides a summarised illustration of all the FPGA changes discussed above.

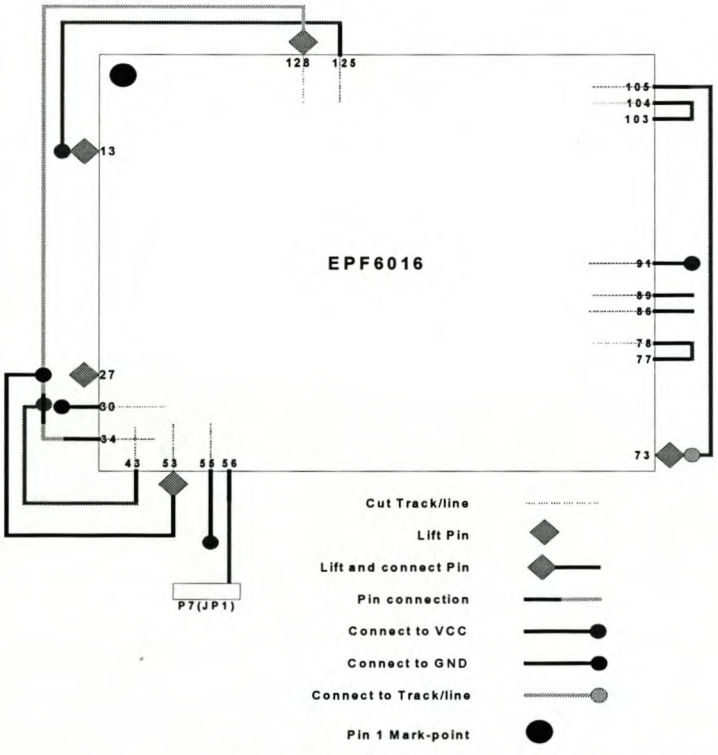


Figure 5.4: Illustration of Operational changes on the FPGA

Varieties of changes have been implemented on different pins due to the specific functions of each pin and accessibility to individual pins on the PCB. For example, pin 13 (which is TDI and was initially connected to the configuration header) is just lifted and not connected to anything, because it is basically not useful in this design (refer to section 5.3.1 above)

5.3 Conclusion

The tests that were performed helped in the debugging of the hardware because they gave important information about hardware functionality. The necessary changes were made and the hardware was basically functioning. As mentioned earlier in this chapter, phase 3 is the most important phase in this project, because it explores the functionality of CAN technology. The following chapter discusses the implementation of phase3.

Chapter 6

6 CAN Software Implementation

In the previous chapter (chapter 5) the hardware was tested according to phase 1,2,4 and 5 of figure 5.1 using small codes of software. This chapter describes the implementation of CAN protocol which is illustrated as test phase3 in figure 5.1. Considering figure 5.1 it can be seen that phase 1,2 and part of phase3 constitute TX-node whereas RX-node is constituted by part of phase3 and phases 4 and 5. Phase3 can be divided into two sections, the first section being the implementation of transmission software on the TX-node and the second section being the implementation of the receive software on the RX-node.

6.1 Implementation of a transmission software on TX-node

Phase 1 receives TCMD data and sends this data to Phase 2. The C505C on the TX-node receives data from the FPGA as described in phase 2. The data is stored in the data bytes of the message object to be transmitted over the serial bus to the RX-node's C505C. Consider the following flowchart (figure 6.1 in the next page) for the C software code (supplied in appendix B1) that has been programmed on the C505C to implement CAN software on the TX-node. The functions of the flow diagram and the C codes described above are detailed in the following subsections.

6.1.1 Initialise CAN

Consider the can_int procedure in the code (appendix B1). The names of the registers and specific bits used in the code are defined in the header files from Siemens documentation (all header files are also attached in appendix B4 [11]). The procedure starts by initialising the C505C to be able to communicate with the CAN module. As mentioned in Chapter 4 the initialisation for CAN protocol programming requires the modification of XPAGE and SYSCON registers.

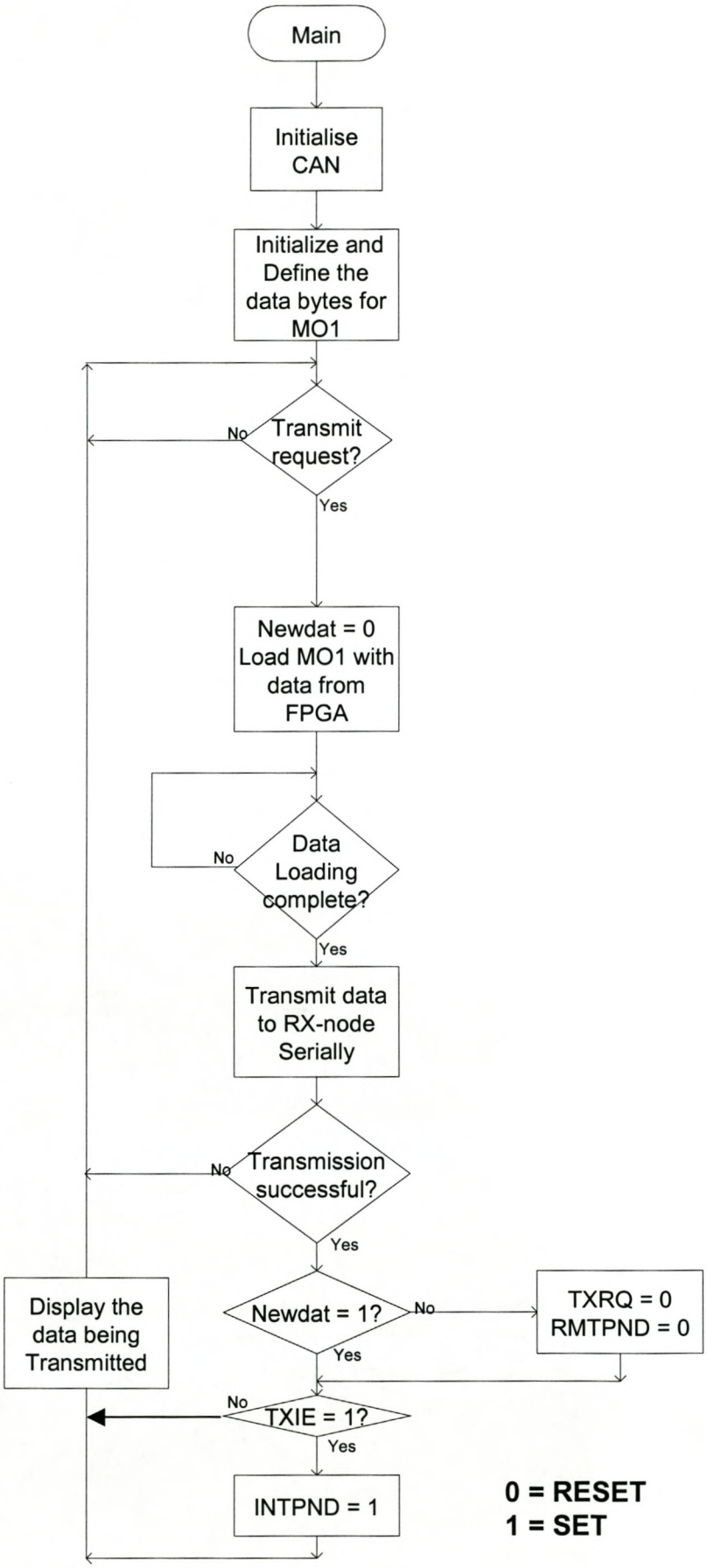


Figure 6.1: Flowchart for the implementation of TX-node software

SYSCON is programmed as follows (1) set bit EALE for ALE generation, since the C505C is using external code memory (2) clear bit CMOD for CAN prescaler selection since the interfaced oscillator frequency is 12MHz (is over 10 MHz)[7]. The data transmission rate was set to 125 Kbaud .The Siemens CAN-bit calculator tool was used to determine the values of the timing registers BTR0 and BTR1 to be 43_H and $7E_H$ respectively. The Global Mask Short Registers are programmed to filter incoming messages with standard identifier. All fifteen MO are initialised to 55_H to declare them invalid by clearing bit MSGVAL in register MCR0_M1. CAN interrupts are disabled, because the C505C is used entirely for CAN and all data come on request. At the end of initialisation bit INIT of register CR is cleared. It was set at the beginning of this procedure to enable initialisation process.

6.1.2 Definition of MO (called define_m1 in the code)

This procedure defines the message object. Only one MO (m1) is used in this project since 32-bits are being used for testing purpose. The message object is programmed according to the following steps: (1) defining the MO as a standard transmission data frame using MCFG_M1 register (2) Loading of eight data bytes with initial values (3) validation and access control for the MO using registers MCR0_M1 and MCR0_M1 respectively.

6.1.3 Transmission of MO (send_m1)

The message is transmitted according to this procedure. The MO is revalidated first and then sent by setting bit TXRQ (Transmission request bit). This bit indicates that the transmission of this message object is requested by the CPU or via a remote frame and is not yet done.

6.1.4 Loading of MO from FPGA (load_m1)

All the data bytes of m1 are updated by the data that is coming from the FPGA. This procedure starts by testing if bit TXRQ is set, so that the data is not updated while the message is still being transmitted. Note that the addresses used to request data from

the FPGA start from 8000_H as discussed in Chapter 5 (phase 2 description). The data is being displayed by sending it to port1 (P1), which is connected to LEDs. This is done so that the TX-node data can be compared to that of RX-node.

6.2 Implementation of a transmission software on RX-node

The RX-node's C505C sends data from the TX-node to the FPGA according to phase 4. TCMD data is sent to phase 5, which then concludes the reception of TCMD data, by a specific tray. Consider the flowchart (in the next page) for the C software code (supplied in appendix B2) that has been programmed on the C505C to implement CAN software on the RX-node:

6.2.1 Initialise CAN

The initialise procedure for the RX-node is the same as that of TX-node

6.2.2 Store M1 (store_m1)

This procedure defines M1 by firstly declaring it valid and secondly allowing access to the message by using registers MCR0_M1 and MCR0_M1 respectively. The received data bytes from the TX-node are stored in the corresponding data bytes of the RX-node. To check if the new data is stored in the data bytes or lost bit MSGLST of register MCR1_M1 is tested. The data is displayed to see if it corresponds to that of TX-node.

6.2.3 Check M1 (check_m1)

The message object is checked and validated before it can be stored in the buffer. This is done by loading the identifiers that are similar to the expected frame from the TX-node into the message object of the RX-node's frame. The received frame's identifiers (from TX-node) are compared to those of the RX-node, to see if they are

identical. Thereafter the received frame will be stored, provided the identifiers are identical.

6.3 Conclusion

The code has been prepared and tested on the CAN node(s) hardware and the results are discussed in Chapter 7 (Results).

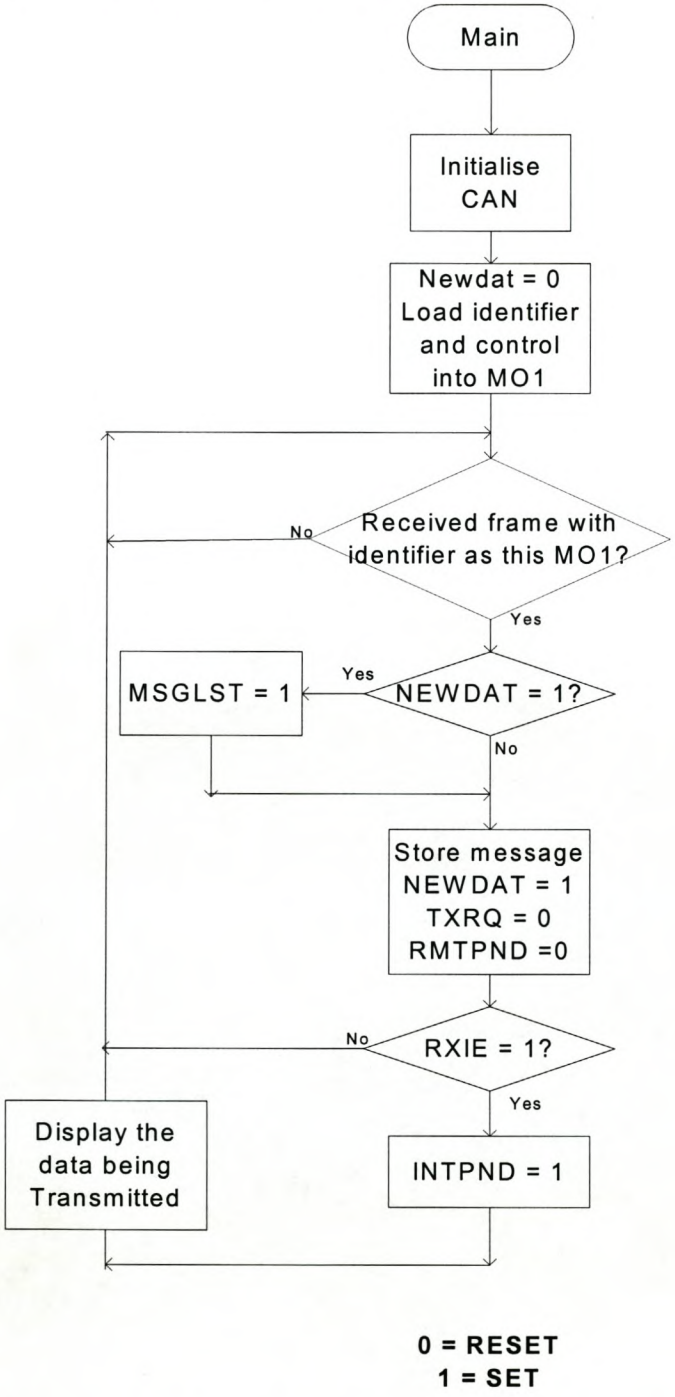


Figure 6.2: Flowchart for the implementation of RX-node software

Chapter 7

7 Results

This chapter discusses the results for CAN node(s). The chapter begins by describing the successes for CAN node(s) to show how the design met the specifications. The problems encountered during testing along the solutions to the problems are highlighted later in this chapter.

7.1 Successes of CAN node's

7.1.1 Reduction of TCMD Wires to Serial Bus Connection

CAN node(s) is able to reduce 232 TCMD wires to serial bus connection, because it implements CAN SBA network protocol. Considering figure 2.8 it can be seen that only two wires are used to implement the serial bus which links 232 commands outputs to the other SUNSAT trays. The trays connect to the bus using two short wires each i.e. 22 (2 wires X 11 trays) short wires are utilised. As a result the TCMD cable harness is reduced by approximately 90%.

7.1.2 Provision of flexible connectivity

CAN node(s) is able to provide flexible connectivity, thereby improving SUNSAT testability because few standard connectors can be used for serial bus and TCMD data connectivity. Consider figure 5.1. In phase 1, 32 I/O pins of the TX-node's FPGA are used to collect 32 TCMD data outputs into the system (see the TX-node's VHDL simulation below, figure 7.1). In figure 7.1 32-bits (FPGA_read) data is latched in the FPGA and these bits are packaged into four 8-bits registers (reg0 to reg3). The data is then placed to the data bus (data_bus) for transmission via eight I/O pins to the micro-controller.

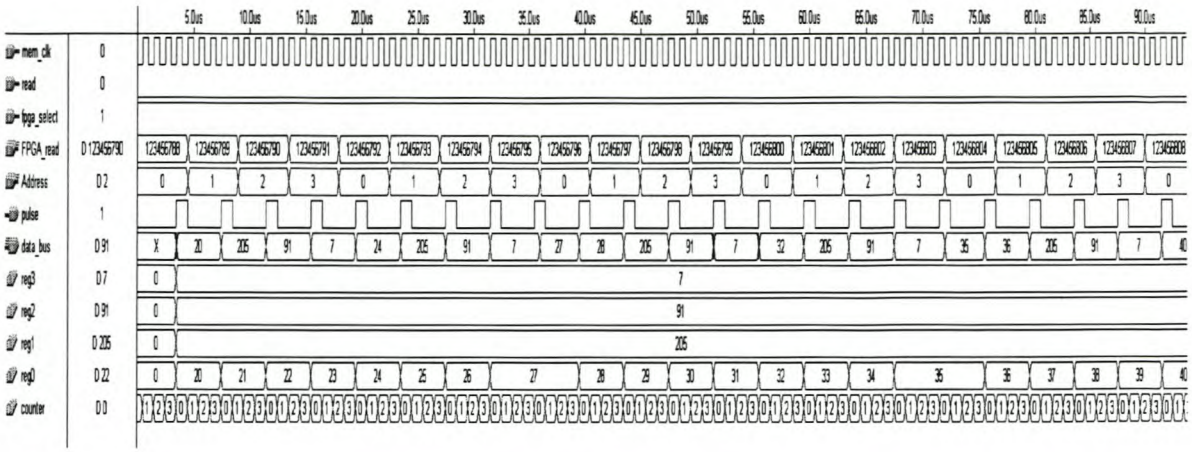


Figure 7.1: TX-node simulation diagram

Two standard serial connectors are used in phase 3 to provide an interface between TX-node and the RX-node(s). In the TX-node the TCMD data from the data bus is transmitted through one standard serial connector. The second standard serial connector is used on the RX-node.

In phase 5, each tray receives its TCMD data via the RX-node's FPGA I/O pins (see the TX-node's VHDL simulation below, figure 7.2)

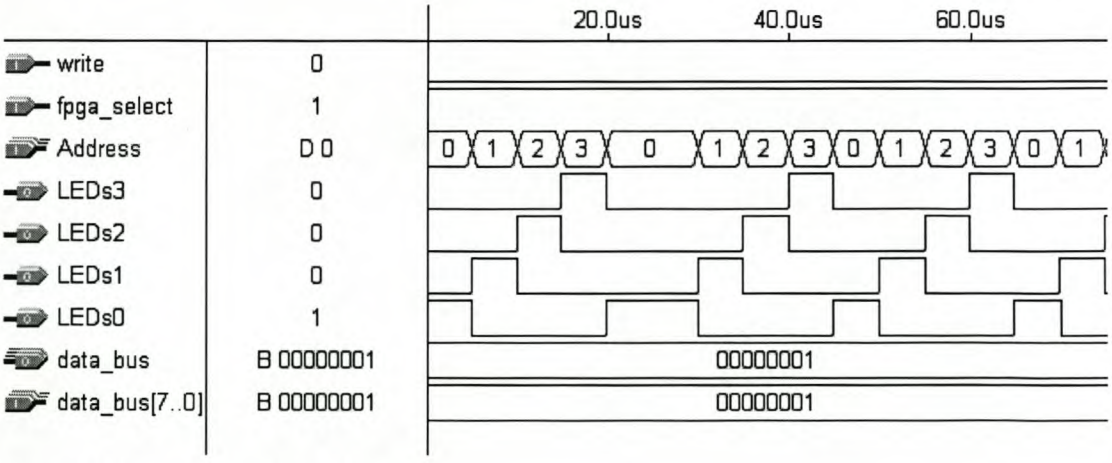


Figure 7.1: RX-node simulation diagram

In this diagram (figure 7.2) the last bit in the data bus (data_bus) is being displayed on the four LEDs (LEDs0 to LEDs3 that are connected to the four I/O pins of the RX-node's FPGA) depending on the specific address.

7.1.3 Elimination of Complex failures

The use of robustness of CAN technology eliminates complex failures on the CAN node(s). CAN technology implements a robust protocol that uses Carrier Sense Multiple Access/Collision Detection with Non Destructive Arbitration (CSMA/CD with NDA) for bus arbitration. CAN node(s) implements Carrier Sense (CS) to check if the data bus is free of any message that was transmitted by TX-node. This is done by checking the transmission request bit (refer to TX-node C code in appendix B1). Since only one node is transmitting (i.e. no Multiple Access (MA)), collision could rarely occur in CAN node(s).

7.1.4 Sufficient throughput for TCMD data transmission

The throughput for CAN was sufficient for TCMD data transmission. As it can be seen in the software for CAN nodes, the transmission and reception baud rate is set to 125Kbaud. This means that at this baud rate the CAN node(s) can be able to transmit 232 TCMD output data at 1.856ms over the serial bus, from TX-nodes to all other RX-nodes. Considering the fact that one valid command contains a total of 144 bits (see Chapter 4 Section 4.1) then at 125Kbaud CAN nodes is capable of transmitting one valid command at 1.152ms. The baud rate can be increased to up-to 1Mbits/sec since this is the maximum baud rate for CAN protocol over short distance. Taking figure 2.6 into consideration it is clear that the sources of the TCMD output data are 1200 and 9600-baud modems. This means that TCMD signals are slow compared to the transmission speed of CAN-node(s).

7.2 Shortcomings of CAN node(s)

The problems encountered during the testing of the system are highlighted along with the solutions to those problems.

7.2.1 Problem occurred during testing of CAN nodes.

The TX-node and the RX-node were built using the bottom-up build method, where each component was tested separately using phases as described in Chapter 5. Although each component worked according to its specifications, unfortunately the most difficult stage in the development cycle of a micro-controller-based circuit occurs during the integration of hardware and software. Some very subtle bugs and errors that eluded simulation emerge during real-time execution [9]. This fact was prove to be true in this design, because during the testing of the whole set-up or principle (i.e. the integration of complete software and the hardware) the design of CAN node(s) failed to yield the expected results.

7.2.2 Solution Applied to the Problem

Several attempts were made to diagnose the problem, but most of them did not give good result, until it was decided to test the software on a working hardware that contains C505C and FPGA (or CPLD)[10]. As a result the software worked perfectly in this hardware, but few adjustments and changes had to be made.

These changes were made on CAN node(s) software in order to test the principle of transmitting and receiving from TCMD tray to other trays. The following two facts were considered:

1. The new hardware is using a small type of PLD (EPM7128C100-15, it will be referred to as EPM in this text) as compared to EPF6016.
2. Most of EPM pins as well as those of the C505C (for the new hardware) were dedicated for specific use on the hardware.

Consequently these two facts limited the number of bits to be transmitted to only four-bits. Therefore instead of eight data bytes in the MO, four data bytes were used, each with one bit from FPGA. The transmission of these four bits was tested and confirmed on the LEDs.

Chapter 8

8 Conclusion and Recommendations

This chapter gives the conclusion of the whole design as well as the recommendations for an ideal system design.

8.1 Conclusion

As mentioned earlier in Chapter 7, several diagnostic attempts were made on the CAN node(s). The option of using another hardware was chosen, because further attempts to make CAN node(s) hardware to work, were time consuming. Although the hardware failed to yield the results, this project has successfully proven that cable harness on the TCMD tray of SUNSAT can be minimised by using CAN technology. Consider the following illustrative figure (figure 8.1).

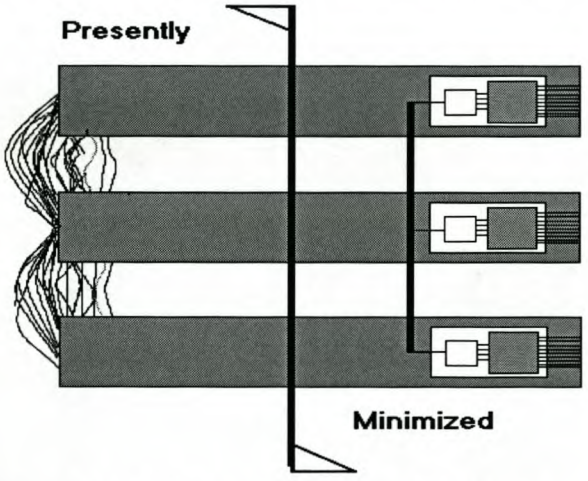


Figure 8.1: Block diagram illustrating the cable harness reduction on SUNSAT

On the left of the above figure (Figure 8.1), is an illustrative representation of how the cables are connected between the present TCMD tray and the other trays on the SUNSAT. Hence, the right side of the above figure shows the minimised structure after the implementation of CAN node(s).

8.2 Recommendations for an Ideal System

This section gives the recommendations for an ideal system that can be used in the real situation. The recommendations are as follows: the

- The ideal system's software should use CAN capabilities of error handling to implement error detection and fault recovery capabilities. For example the TX-node has to wait for ACK-bit before transmitting. If there is any error the TX-node should be able to re-send a data to recover the communication.
- The crystal oscillator on the TX-node's FPGA should be relatively faster than that of the C505C of both TX-node and RX-node to avoid bottlenecking the system. The oscillator on the RX-node's FPGA can be equal to that of the C505C. In this regard if there could be any problems, the external RAM can be used to relieve either the C505C or the FPGA. Furthermore, the software can also help by implementing interrupts procedures.
- The software of the ideal system should implement CAN capabilities of multi-master network, so that the RX-node(s) can also send remote frames to request data from TX-node. The sending of remote frames can help to avoid bottleneck problems.
- The RX-node's PLD should be relatively small, with the capabilities of transmitting the data bytes that are assigned to it.

References

[1] de Swardt I., "A Telemetry System for SUNSAT", December 1994

[2] le Roux A.G., "An FPGA-based Approach to the Compacting of the SUNSAT-1 TTM System", October 1998

[3] Chetty C., "Evaluation of the Effectiveness, Performance and Integrity of SUNSAT Telemetry System", December 1999

[4] Botha T.F., "Design of a Reliable Telecommand System for the SUNSAT Microsatellite", January 1994

[5] Koekemoer J., "Investigation of a Command and Data Handling architecture for the SUNSAT-2 micro satellite", November 1999

[6] Siemens, "C505C 8-bit CMOS Microcontroller User's Manual ", August 1997

[7] Siemens "Application Note AP082001 on The CAN Controller in the C515C", December 1996.

[8] Chan Pak K. /Samiha Mourad, "Digital Design using Programmable Gate Arrays", 1994.

[9] MacKenzie I. Scott, "The 8051 Microcontroller ", 1999.

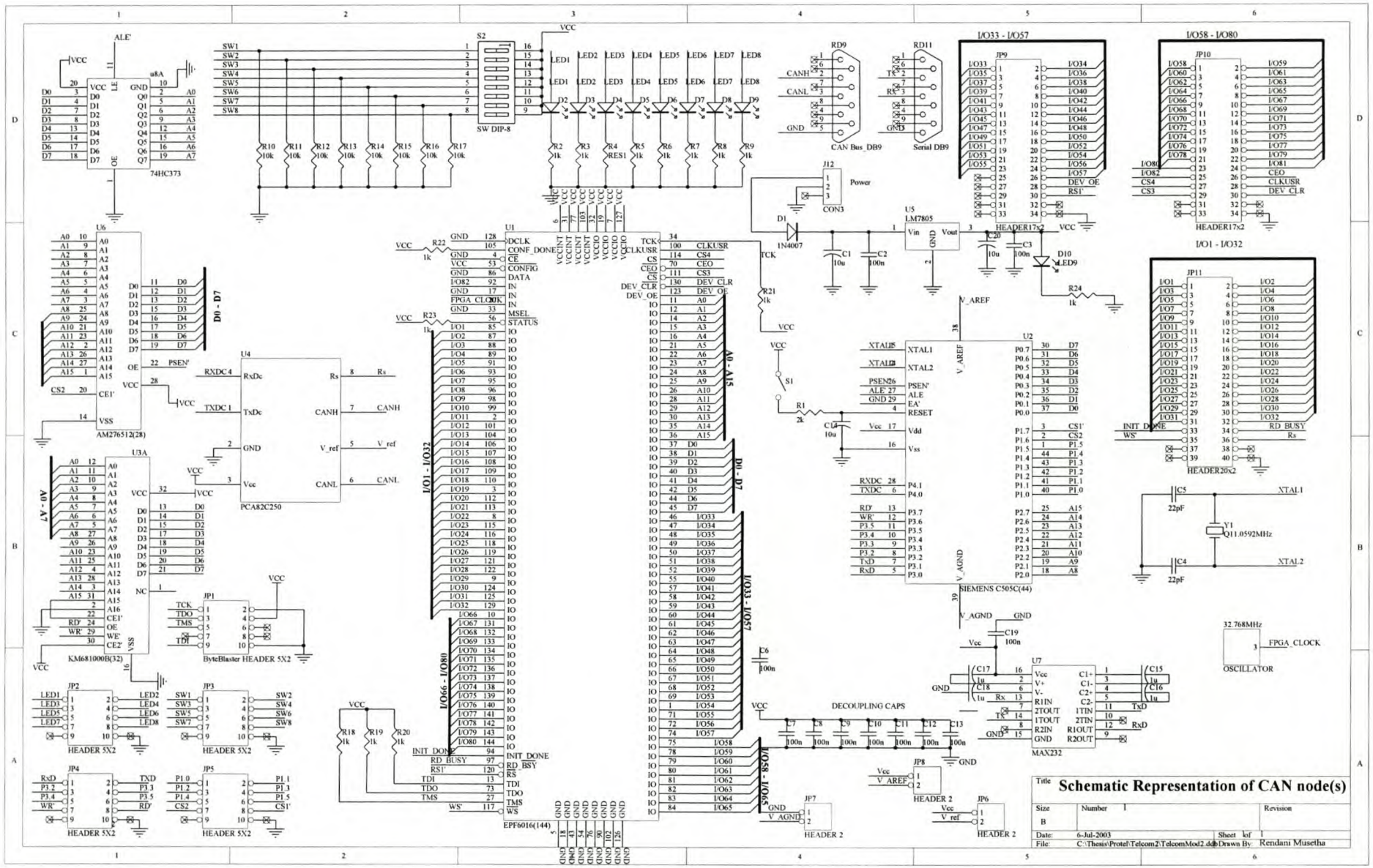
[10] Farr Xandri C., "Development of a Fault-Tolerant Bus System suitable for a High-Performance, Embedded,Real-Time Application on SUNSAT 's ADCS", November 2000

[11] Dr. Jens Barrenscheen, HL MC PD, Axel Wolf, HL MC AT, "Header files" June 1997

Appendix A

Schematic

The schematic in the following page is a representation of CAN node(s) hardware design. It is designated as **SCHEMATIC 1**.



B1 TX-node C code

```
/*****************************************************************************
 * Program name:        "TX-node.C"
 * Compiler used:       Franklin Proview32
 * Task:                To transmit TCMD data to other SUNSAT trays
 * Authors:             Rendani Musetha
*****************************************************************************
/
#pragma SMALL
#pragma DEBUG OBJECTEXTEND CODE          // Command lines directives.

#include "regc505c.h"                    /* special function register declarations for
C505C */
#include "intc505c.h"                    /* interrupt definition file */
#include <absacc.h>
#include "canreg.h"                      /* can definition file */

#define true 1
#define false 0
//############################## BAUDRATE SELECTION ##############################/
#define BTR0_VAL 0x43
#define BTR1_VAL 0x7E
/* This results in 125kBaud @ 8MHz (CAN prescaler disabled) */
/* This results in 125kBaud @ 16MHz (CAN prescaler enabled) */

//##########################GLOBAL DECLARATION OF VARIABLES ##################/

unsigned char pdata canreg[256];          //declaration of can register array

unsigned char data i,j=0;
sbit P3_2 = 0xB2;                        //declaration of test pins P3_2 - P3_5
sbit P3_3 = 0xB3;
sbit P3_4 = 0xB4;
sbit P3_5 = 0xB5;

sbit CAN_STDBY = P1 ^ 5;                 //Application specific to the working board

//############################ INITITIALIZATION PROCEDURE ####################/

//                                       CAN initialisation
```

```
void can_init (void)
{
 SYSCON &= 0xF0;                    //set EALE and clear CMOD,XMAP1,XMAP0
 XPAGE = 0xF7;                      // 8 bit addressing of CAN

 CR = 0x41;                         // set init and CCE (enable access baudrate)
 BTR0 = BTR0_VAL;
 BTR1 = BTR1_VAL;                   // 125 kBaud
 CR = 0x01;                         // reset CCE (disable access baudrate)

 GMS0 = 0xFF;  GMS1 = 0xFF;         // global mask short
 UGML0 = 0xFF; UGML1 = 0xFF;        // global mask long
 LGML0 = 0xFF; LGML1 = 0xFF;
 UMLM0 = 0xFF; UMLM1 = 0xFF;        // global mask short
 LMLM0 = 0xFF; LMLM1 = 0xFF;

//********************************** All MO declared invalid *************************//

 MCR0_M1 = 0x55;                    // message 1 not valid
 MCR0_M2 = 0x55;
 MCR0_M3 = 0x55;
 MCR0_M4 = 0x55;
 MCR0_M5 = 0x55;
 MCR0_M6 = 0x55;
 MCR0_M7 = 0x55;
 MCR0_M8 = 0x55;
 MCR0_M9 = 0x55;
 MCR0_M10 = 0x55;
 MCR0_M11 = 0x55;
 MCR0_M12 = 0x55;
 MCR0_M13 = 0x55;
 MCR0_M14 = 0x55;
 MCR0_M15 = 0x55;                   // message 15 not valid

 SR = 0xE7;                         // clear TXOK and RXOK
 IEN1 = 0;                          //All CAN interrupt disabled
 CR = 0x00;                         // reset INIT, no interrupts
}
#endif
```

```
//############################ Definition of M0 ######################################//
//                          standard transmission frame

void define_m1 (void)
{
 MCR1_M1 = 0xDB;                        // CPUUPD=1, Newdata=0;
 UAR0_M1 = 0x40;  UAR1_M1 = 0x23;       // identifier  ID 01000000 00100011 for standard frames
 MCFG_M1 = 0x48;                        // transmit object, data length code = 4
//*****************************All 8 data bytes reset to
0x00*****************************//
 DB0_M1          = 0x00;
 DB1_M1          = 0x00;
 DB2_M1          = 0x00;
 DB3_M1          = 0x00;

 MCR1_M1 = 0x56;                        // RMTPND=0, TXRQ=0, CPUUPD=0, NEWDAT=0
 MCR0_M1 = 0x95;                        // MSGVAL=1, TXIE=0, RXIE=0, INTPND=0
}

//################################# Transmission of MO ###################################//
// send message 1 CPUUPD=0, TXRQ=1, MSGVAL=1

void send_m1 (void)
{
 MCR0_M1 = 0x95;                  // MSGVAL=1, TXIE=0, RXIE=0, INTPND=0 (validation of
message)
 MCR1_M1 = 0xE7;                        // TXRQ = 1 (tranmission)
}

//################################# Loading MO ###########################################//

void Load_m1(void)
{
  MCR1_M1 = 0xFA;                       // set CPUUPD and NEWDAT

  DB0_M1        = XBYTE[0x08000];       //Data_bytes from FPGA
  DB1_M1        = XBYTE[0x08001];      //from address locations 8000H to 8003H
  DB2_M1        = XBYTE[0x08002];
  DB3_M1        = XBYTE[0x08003];
```

```
  MCR1_M1 = 0x56;                    // RMTPND=0, TXRQ=0, CPUUPD=0, NEWDAT=0;
}
#endif

//################################## Main Procedure ###################################/
void main (void)
{
 can_init();                         // CAN initialisation
 CAN_STDBY = false;                  // Application specfic to working hardware
 define_m1();                        // Definition of MO
 while(1)
 {
 if ((MCR1_M1 & 0x03 )== 0x01)       //test if newdata is written to the data bytes,NEWDAT=0;
 {
  send_m1();
 }
 if((MCR1_M1 & 0x30) == 0x10)        // if reset TXRQ
 {
  Load_m1();
 }
//**********************Display of data bits that are being transmitted**********************/
 if(DB0_M1 && 0x01)
                    {
                     P3_2 = 0;
                    }
 else
                     P3_2 = 1;
 if(DB1_M1 && 0x01)
                    {
                     P3_3 = 0;
                    }
            else
                     P3_3 = 1;
 if(DB2_M1 && 0x01)
                    {
                     P3_4 = 0;
                    }
            else
                     P3_4 = 1;
          if(DB3_M1 && 0x01)
```

```
            {
             P3_5 = 0;
            }
        else
             P3_5 = 1;
}
}
```

B2 RX-node C code

```
/*****************************************************************************
 * Program name:        "RX-node.C"
 * Compiler used:       Franklin Proview32
 * Task:                To receive TCMD data from TX-node
 * Author:              Rendani Musetha
*****************************************************************************/
#pragma SMALL
#pragma DEBUG OBJECTEXTEND CODE          // Command lines directives.

#include "regc505c.h"                    /*special function register declarations for
C505C */
#include "intc505c.h"                    /* interrupt definition file */
#include <absacc.h>
#include "canreg.h"                      /* can definition file */

#define true 1
#define false 0
//################################BAUDRATE SELECTION#########################
#define BTR0_VAL 0x43
#define BTR1_VAL 0x7E
/* This results in 125kBaud @ 8MHz (CAN prescaler disabled) */
/* This results in 125kBaud @ 16MHz (CAN prescaler enabled) */

//######################### GLOBAL DECLARATION OF VARIABLES ##################

unsigned char pdata canreg[256];          //declaration of can register array

unsigned char data i,j=0;
sbit P3_2 = 0xB2;                         //declaration of test pins P3_2 - P3_5
sbit P3_3 = 0xB3;
sbit P3_4 = 0xB4;
sbit P3_5 = 0xB5;

sbit CAN_STDBY = P1 ^ 5;                  //Application specific to the working board

//##########################INITITIALIZATION PROCEDURE ####################/##

//                                        CAN initialisation
```

```
void can_init (void)
{
 SYSCON &= 0xF0;                    //set EALE and clear CMOD,XMAP1,XMAP0
 XPAGE = 0xF7;                      // 8 bit addressing of CAN

 CR = 0x41;                         // set init and CCE (enable access baudrate)
 BTR0 = BTR0_VAL;  BTR1 = BTR1_VAL;          // 125 kBaud
 CR = 0x01;                                  // reset CCE (disable access baudrate)

 GMS0 = 0xFF;  GMS1 = 0xFF;         // global mask short
 UGML0 = 0xFF;  UGML1 = 0xFF;       // global mask long
 LGML0 = 0xFF;  LGML1 = 0xFF;
 UMLM0 = 0xFF;  UMLM1 = 0xFF;       // global mask short
 LMLM0 = 0xFF;  LMLM1 = 0xFF;


//****************************** All MO declared invalid *******************************//

 MCR0_M1 = 0x55;                    // message 1 not valid
 MCR0_M2 = 0x55;
 MCR0_M3 = 0x55;
 MCR0_M4 = 0x55;
 MCR0_M5 = 0x55;
 MCR0_M6 = 0x55;
 MCR0_M7 = 0x55;
 MCR0_M8 = 0x55;
 MCR0_M9 = 0x55;
 MCR0_M10 = 0x55;
 MCR0_M11 = 0x55;
 MCR0_M12 = 0x55;
 MCR0_M13 = 0x55;
 MCR0_M14 = 0x55;
 MCR0_M15 = 0x55;                   // message 15 not valid

 SR = 0xE7;                         // clear TXOK and RXOK
 IEN1 = 0;                          //All CAN interrupt disabled
 CR = 0x00;                         // reset INIT, no interrupts
}
#endif
```

```
//################################### Store M1 ###################################//

#if 1
void store_m1 (void)
{
 MCR1_M1 = 0xDB;                                                    //        CPUUPD=1,
Newdata=0;
 UAR0_M1 = 0x40;  UAR1_M1 = 0x23;     // identifier ID 01000000 00100011 for standard frames
 MCFG_M1 = 0x40;                                                    // receive object, data
length code = 4

//************************** All 8 data bytes sent to FPGA **************************//

 XBYTE[0x08000] = DB0_M1;              //Data bytes sent to Address location 8000H - 8003H
 XBYTE[0x08001] = DB1_M1;
 XBYTE[0x08002] = DB2_M1;
 XBYTE[0x08003] = DB3_M1;

 MCR1_M1 = 0x56;                       // RMTPND=0, TXRQ=0, CPUUPD=0, NEWDAT=0
 MCR0_M1 = 0x95;                       // MSGVAL=1, TXIE=0, RXIE=0, INTPND=0
}

//############################## Check for receiced MO ##############################/

void check_m1 (void)
{
 if((MCR1_M1 & 0x03) == 0x02)          // if NEWDATA = 1
  {
        MCR1_M1 = 0xFB;                // the message is lost MSGLST = 1
  }
  else
  MCR1_M1 = 0xF7;                      // the message is received MSGLST = 0
}

//############################### Main Procedure ###################################/
void main (void)
{
 can_init();                          // CAN initialisation
 CAN_STDBY = false;                    // Application specfic to working hardware
```

```
while(1)
{
store_m1();                          // MO stored in Data bytes
if (MCR1_M1 & 0x0C) == 0x04; // if MSGLST = 0
{
//************************** Display of data bits that are being transmitted *****************/
 if(DB0_M1 && 0x01)
                    {
                     P3_2 = 0;
                    }
 else
                     P3_2 = 1;
 if(DB1_M1 && 0x01)
                    {
                     P3_3 = 0;
                    }
        else
                     P3_3 = 1;
 if(DB2_M1 && 0x01)
                    {
                     P3_4 = 0;
                    }
        else
                     P3_4 = 1;
        if(DB3_M1 && 0x01)
                    {
                     P3_5 = 0;
                    }
        else
                     P3_5 = 1;
 }
 }
}
```

B3 Phase 2 C code

```
void main (void)
{
  uchar DB0,DB1,DB2,DB3,DB4,DB5,DB6,DB7,DB8;
        FOREVER
        {
         if (P3_2 == true)
         {
          DB0 = XBYTE[0x8000];
          P1 = DB0;
         }
         else
         if (P3_2 == false)
         {
          DB1 = XBYTE[0x8001];
          P1 = DB1;
         }
         else
         if (P3_3 == true)
         {
          DB2 = XBYTE[0x8002];
          P1 = DB2;
         }
         else
         if (P3_3 == false)
         {
          DB3 = XBYTE[0x8003];
          P1 = DB3;
         }
         else
         if (P3_4 == true)
         {
          DB4 = XBYTE[0x8004];
          P1 = DB4;
         }
         else
         if (P3_4 == false)
         {
          DB5 = XBYTE[0x8005];
```

```
            P1 = DB5;
           }
           else
           if (P3_5 == true)
           {
            DB6 = XBYTE[0x8006];
            P1 = DB6;
           }
           else
           if (P3_5 == false)
           {
            DB7 = XBYTE[0x8007];
            P1 = DB7;
           }
          }
}
```

B4 Siemens Header Files

Appendix C

VHDL Codes

C1 TX-node VHDL code

```
--C1     TX-node VHDL code
-- Program name:        "TX-node.vhd"
-- Developed using:     Max-II-plus
--Task:                 To collect TCMD data and send to C505C
--Author:               Rendani Musetha

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity finalTx is
        port(mem_clk, read      : in    std_logic;                  -- Clock source
                fpga_select     : in    std_logic;          -- The chip select connected to A15
                Address         : in    std_logic_vector(14 downto 0); --Address A0 - A14
                FPGA_read       : in    std_logic_vector(31 downto 0); -- TCMD to be collected
                test            : out   std_logic_vector(7 downto 0); -- test signal
                pulse           : out   std_logic; --test for sampling for simulation purpose
                data_bus        : inout std_logic_vector(7 downto 0) --the data bus
                );
end finalTx;

architecture behavioral of finalTx is
        signal FPGA_write : std_logic_vector(7 downto 0); --temp signals to write-out TCMD data
        signal reg0,reg1,reg2,reg3 : std_logic_vector(7 downto 0); --temp signals to store TCMD
data
        signal clk_pulse        : std_logic;            --sampling pulse
begin

--provides sampling to avoid data loss
Sampling:
process(mem_clk)is
        variable counter        : integer range 0 to 3;
        begin
                if (mem_clk'event and mem_clk = '1') then --A 32MHz clock is divided by 4
                        if (counter = 3)then
                                counter := 0;
                                clk_pulse <= '1';
```

```
                else
                        clk_pulse <= '0';
                        counter := counter + 1;
                end if;
        end if;
    pulse <= clk_pulse;
    end process Sampling;

--TCMD data is stored in temp registers at each clock pulse
Load_regs:
process(clk_pulse) is
    begin
        if (clk_pulse = '1') then
            reg3 <= FPGA_read(31 downto 24);
            reg2 <= FPGA_read(23 downto 16);
            reg1 <= FPGA_read(15 downto 8);
            reg0 <= FPGA_read(7 downto 0);
        end if;
end process Load_regs;

--Stored data is written to the write_out registers FPGA_write
Write_out: process(mem_clk,fpga_select,Address) is
begin
    if (clk_pulse = '1') then     -- check clock pulse
        if (fpga_select = '1') then          --check for Chip select (CS1)
            if (read = '0' and Address = "000000000000000") then
            --Check for read signal --and validation of Addres
                FPGA_write <= reg0;
                Test <= "00000001";
            elsif (read = '0' and Address = "000000000000001") then
                FPGA_write <= reg1;
                Test <= "00000010";
            elsif (read = '0' and Address = "000000000000010") then
                FPGA_write <= reg2;
                Test <= "00000100";
            elsif (read = '0' and Address = "000000000000011") then
                FPGA_write <= reg3;
                Test <= "00001000";
            end if;
        end if;
```

```
        end if;
end process Write_out;

data_bus <= FPGA_write when read = '0' else "ZZZZZZZZ"; -- write data to the data bus

end behavioral;
```

C2 RX-node VHDL code

```
-- Program name:        RX-node.vhd
-- Developed using:     Max-II-plus
--Task:                 To collect TCMD data and send to C505C
--Author:               Rendani Musetha

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity Rx_node is
        port    (write    : in    std_logic;              -- Clock source
                 fpga_select    : in    std_logic;      -- The chip select connected to A15
                 Address        : in    std_logic_vector(14 downto 0); --Address A0 - A14
                 LEDs           : out   std_logic_vector(3 downto 0); -- TCMD to be displayed
                 data_bus       : inout std_logic_vector(7 downto 0) --the data bus
                 );
end Rx_node;

architecture behavioral of Rx_node is
begin
--Display data
Display: process(fpga_select,Address,write) is
begin
        if (fpga_select = '1') then             --check for Chip select (CS1)
                if (write = '0') then  --Check for write
                        if (Address = "000000000000000") then
                        -- signal and validation of Address
                                LEDs(0) <= data_bus(0);
                                data_bus(7 downto 1) <= "0000000";
                                LEDs(3 downto 1) <= "000";
                        elsif(Address = "000000000000001") then
                                LEDs(1) <= data_bus(0);
                                data_bus(7 downto 1)<= "0000000";
                                LEDs(3 downto 2) <= "00";
                                LEDs(0) <= '0';
                        elsif (Address = "000000000000010") then
                                LEDs(2) <= data_bus(0);
```

```
                    data_bus(7 downto 1)<= "0000000";
                    LEDs(1 downto 0) <= "00";
                    LEDs(3) <= '0';
                elsif (Address = "000000000000011") then
                    LEDs(3) <= data_bus(0);
                    data_bus(7 downto 1)<= "0000000";
                    LEDs(2 downto 0) <= "000";
                else
                    data_bus <= "ZZZZZZZZ";
                end if;
            else
                data_bus <= "ZZZZZZZZ";
            end if;
        end if;
end process Display;

end behavioral;
```

C3 Phase 1 VHDL code

```
entity phs1_A is
        port(clk,cs1            : in    std_logic;
                Address         : in    std_logic_vector(2 downto 0);
                FPGA_read       : in    std_logic_vector(63 downto 0);
                data_bus: inout std_logic_vector(7 downto 0)
                );
end phs1_A;

architecture behavioral of phs1_A is
        signal FPGA_write               : std_logic_vector(7 downto 0);
        signal reg0,reg1,reg2,reg3      : std_logic_vector(7 downto 0);
        signal reg4,reg5,reg6,reg7      : std_logic_vector(7 downto 0);
begin
Load_regs: process(clk) is
        begin
                if (clk'event and clk = '1') then
                        reg7 <= FPGA_read(63 downto 56);
                        reg6 <= FPGA_read(55 downto 48);
                        reg5 <= FPGA_read(47 downto 40);
                        reg4 <= FPGA_read(39 downto 32);
                        reg3 <= FPGA_read(31 downto 24);
                        reg2 <= FPGA_read(23 downto 16);
                        reg1 <= FPGA_read(15 downto 8);
                        reg0 <= FPGA_read(7 downto 0);
                end if;
end process Load_regs;

Write_out: process(clk,cs1,Address) is
begin
        if (clk'event and clk = '1') then
                if (CS1 = '1') then
                        if Address = "000" then
                                FPGA_write <= reg0;
                        elsif Address = "001" then
                                FPGA_write <= reg1;
                        elsif Address = "010" then
                                FPGA_write <= reg2;
                        elsif Address = "011" then
```

```
                    FPGA_write <= reg3;
                elsif Address = "100" then
                    FPGA_write <= reg4;
                elsif Address = "101" then
                    FPGA_write <= reg5;
                elsif Address = "110" then
                    FPGA_write <= reg6;
                elsif Address = "111" then
                    FPGA_write <= reg7;
                end if;
            end if;
        end if;
end process Write_out;
        data_bus <= FPGA_write when cs1 = '1' else "ZZZZZZZZ";
end behavioral;
```

Appendix D

Datasheets

Appendix D1.1

Siemens C505C Memory Organisation

3 Memory Organization

The C505 CPU manipulates operands in the following four address spaces:

- up to 64 Kbytes of program memory (16K on-chip program memory for C505-2R)
- up to 64 Kbytes of external data memory
- 256 bytes of internal data memory
- 256 bytes of internal XRAM data memory
- 256 bytes CAN controller registers / data memory (C505C only)
- a 128 byte special function register area

Figure 3-1 illustrates the memory address spaces of the C505.

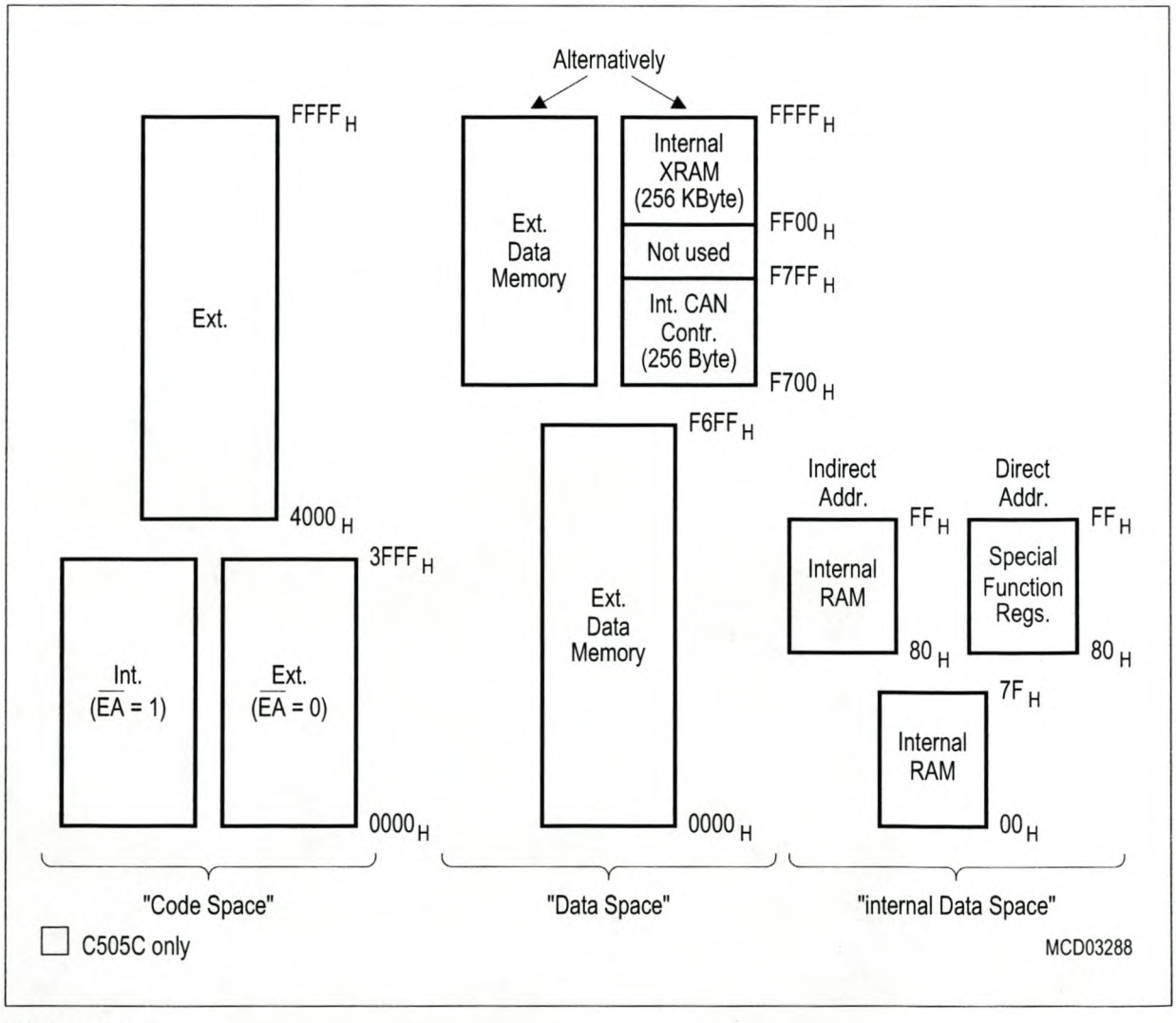


Figure 3-1
C505 Memory Map

3.4 XRAM Operation

The XRAM in the C505 is a memory area that is logically located at the upper end of the external data memory space, but is integrated on the chip. Because the XRAM is used in the same way as external data memory the same instruction types (MOVX) must be used for accessing the XRAM.

3.4.1 XRAM/CAN Controller Access Control

Two bits in SFR SYSCON, XMAP0 and XMAP1, control the accesses to XRAM and the CAN controller. XMAP0 is a general access enable/disable control bit and XMAP1 controls the external signal generation during XRAM/CAN controller accesses. CAN controller accesses are applicable only in the case of the C505C versions.

Special Function Register SYSCON (Address $B1_H$) **Reset Value : $XX100X01_B$**

Bit No.	MSB 7	6	5	4	3	2	1	LSB 0	
$B1_H$	–	–	EALE	RMAP	CMOD	–	XMAP1	XMAP0	SYSCON

The functions of the shaded bits are not described here.

Bit	Function
XMAP1	XRAM/CAN controller visible access control Control bit for $\overline{RD}$/$\overline{WR}$ signals during XRAM/CAN Controller accesses. If addresses are outside the XRAM/CAN controller address range or if XRAM is disabled, this bit has no effect. XMAP1 = 0 : The signals $\overline{RD}$ and $\overline{WR}$ are not activated during accesses to the XRAM/CAN Controller XMAP1 = 1 : Ports 0, 2 and the signals $\overline{RD}$ and $\overline{WR}$ are activated during accesses to XRAM/CAN Controller. In this mode, address and data information during XRAM/CAN Controller accesses are visible externally.
XMAP0	Global XRAM/CAN controller access enable/disable control XMAP0 = 0 : The access to XRAM and CAN controller is enabled. XMAP0 = 1 : The access to XRAM and CAN controller is disabled (default after reset!). All MOVX accesses are performed via the external bus. Further, this bit is hardware protected.
–	Reserved bits for future use. Read by CPU returns undefined values.

When bit XMAP1 in SFR SYSCON is set, during all accesses to XRAM and CAN Controller $\overline{RD}$ and $\overline{WR}$ become active and port 0 and 2 drive the actual address/data information which is read/written from/to XRAM or CAN controller. This feature allows to check the internal data transfers to XRAM and CAN controller. When port 0 and 2 are used for I/O purposes, the XMAP1 bit should not be set. Otherwise the I/O function of the port 0 and port 2 lines is interrupted.

Appendix D1.2

Siemens C505C External Bus Interface

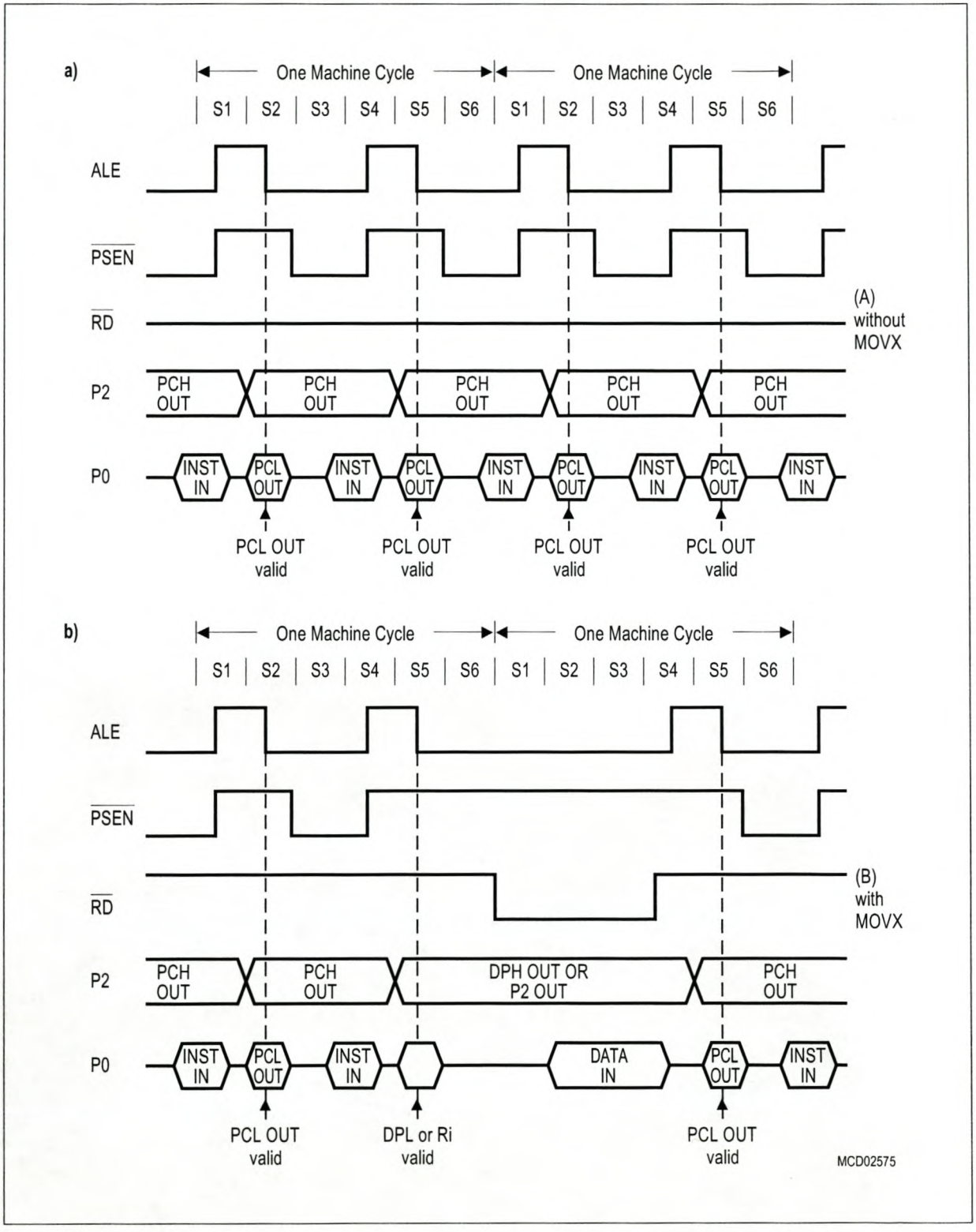


Figure 4-1
External Program Memory Execution

Special Function Register SYSCON (Address $B1_H$) **Reset Value : $XX100X01_B$**

Bit No.	MSB 7	6	5	4	3	2	1	LSB 0	
$B1_H$	–	–	EALE	RMAP	CMOD	–	XMAP1	XMAP0	SYSCON

The functions of the shaded bits are not described here.

Bit	Function
CMOD	Prescaler selection for CAN controller Control bit for CAN controller input clock selection. The time quantum (t_q) of the CAN controller timing is affected by this (and hecne the baudrate). CMOD = 0 : The ÷ 2 prescaler is enabled (reset value). CMOD = 1 : The ÷ 2 prescaler is disabled. This bit must be cleared when f_{osc} is over 10 MHz.

The software Initialization is enabled by setting bit INIT in the control register. This can be done by the microcontroller via software, or automatically by the CAN controller on a hardware reset, or if the EML switches to busoff state.

While INIT is set

- all message transfer from and to the CAN bus is stopped
- the CAN bus output TXDC is “1” (recessive)
- the control bits NEWDAT and RMTPND of the last message object are reset
- the counters of the EML are left unchanged.

Setting bit CCE in addition, allows to change the configuration in the bit timing register.

For initialization of the CAN Controller, the following actions are required:

- configure the bit timing register (CCE required)
- set the Global Mask Registers
- initialize each message object.

If a message object is not needed, it is sufficient to clear its message valid bit (MSGVAL), ie. to define it as not valid. Otherwise, the whole message object has to be initialized.

After the initialization sequence has been completed, the microcontroller clears the INIT bit.

Now the BSP synchronizes itself to the data transfer on the CAN bus by waiting for the occurrence of a sequence of 11 consecutive recessive bits (ie. bus idle) before it can take part in bus activities and start message transfers.

Appendix D2.1

Altera ByteBlasterMV Connections

Download Modes

The ByteBlaster cable provides two download modes:

- Passive serial (PS) mode—Used for configuring FLEX 10K, FLEX 8000, and FLEX 6000 devices
- JTAG mode—Industry-standard Joint Test Action Group (JTAG) boundary-scan test (BST) circuitry (compliant with IEEE Std. 1149.1-1990) implemented for configuring FLEX 10K or programming MAX 9000, MAX 7000S, and MAX 7000A devices

ByteBlaster Connections

The ByteBlaster cable has a 25-pin male header that connects to the PC parallel port, and a 10-pin female plug that connects to the circuit board. Data is downloaded from the PC's parallel port through the ByteBlaster download cable to the circuit board via the connections discussed in this section.

To configure/program 3.3-V devices (e.g., FLEX 10KA and MAX 7000A devices) using the ByteBlaster cable, connect the cable's VCC pin to a 5.0-V power supply and the device to a 3.3-V power supply. FLEX 10KA and MAX 7000A devices have 5.0-V tolerant inputs, so the ByteBlaster cable's 5.0-V output will not harm these 3.0-V devices. The pull-up resistors should be connected to the 5.0-V power supply.

ByteBlaster Header & Plug Connections

The 25-pin male header connects to a parallel port with a standard parallel cable. Table 1 identifies the pins and the download modes.

Table 1. ByteBlaster 25-Pin Header Pin-Outs

Pin	JTAG Mode Signal Name	PS Mode Signal Name
2	TCK	DCLK
3	TMS	nCONFIG
8	TDI	DATA0
11	TDO	CONF_DONE
13	NC	nSTATUS
15	GND	GND
18 to 25	GND	GND

Figure 3. ByteBlaster 10-Pin Female Plug Dimensions

Dimensions are shown in inches. The spacing between pin centers is 0.1 inch.

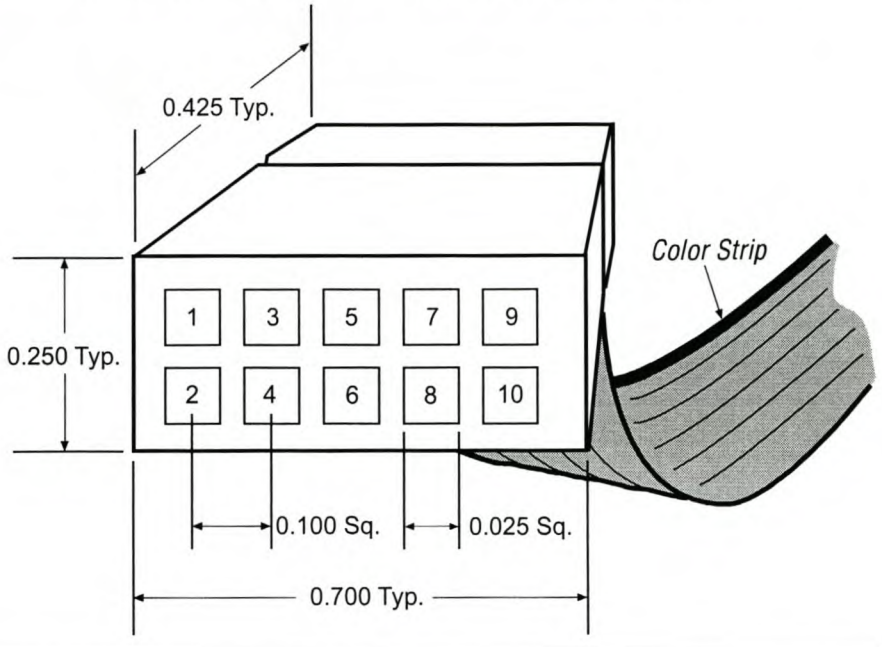


Table 2 identifies the 10-pin female plug's pin names for the corresponding download mode.

Table 2. ByteBlaster Female Plug's Pin Names & Download Modes

Pin	JTAG Mode		PS Mode	
	Signal Name	Description	Signal Name	Description
1	TCK	Clock signal	DCLK	Clock signal
2	GND	Signal ground	GND	Signal ground
3	TDO	Data from device	CONFIG_DONE	Configuration control
4	VCC	Power supply	VCC	Power supply
5	TMS	JTAG state machine control	nCONFIG	Configuration control
6	–	No connect	–	No connect
7	–	No connect	nSTATUS	Configuration status
8	–	No connect	–	No connect
9	TDI	Data to device	DATA0	Data to device
10	GND	Signal ground	GND	Signal ground

☞ The circuit board must supply V_{CC} and ground to the ByteBlaster cable.

Appendix D2.2

Altera Flex 6000 Device Pin-Outs

Table 40 shows the data sources for each configuration scheme.

Table 40. Configuration Schemes

Configuration Scheme	Data Source
Configuration device	EPC2, EPC1, or EPC1441 configuration device
Passive serial (PS)	BitBlaster™, ByteBlaster™, ByteBlasterMV™, or MasterBlaster™ download cables, or serial data source *(1)*
Passive serial asynchronous (PSA)	BitBlaster, ByteBlaster, ByteBlasterMV™, or MasterBlaster™ download cables, or serial data source *(1)*

Note:
(1) The ByteBlaster cable is obsolete and is replaced by the ByteBlasterMV cable, which can program or configure 2.5-V, 3.3-V, and 5.0-V devices.

Device Pin-Outs

Tables 41 and 42 show the pin names and numbers for FLEX 6000 device packages.

Table 41. FLEX 6000 Device Pin-Outs (Part 1 of 2) *Notes (1), (2)*

Pin Name	100-Pin TQFP EPF6010A	100-Pin TQFP EPF6016A	100-Pin FineLine BGA EPF6010A EPF6016A	144-Pin TQFP EPF6010A	144-Pin TQFP EPF6016 EPF6016A EPF6024A
MSEL *(3)*	22	22	H2	33	33
nSTATUS *(3)*	39	39	G5	56	56
nCONFIG *(3)*	36	36	K5	53	53
DCLK *(3)*	89	89	D6	128	128
CONF_DONE *(3)*	72	72	C9	105	105
INIT_DONE *(4)*	64	64	E10	94	94
nCE *(3)*	4	4	C2	4	4
nCEO *(5)*	49	49	K9	70	70
nWS *(5)*	81	81	C7	117	117
nRS *(5)*	83	83	A7	120	120
nCS *(5)*	77	77	A9	111	111
CS *(5)*	78	78	C8	114	114
RDYnBUSY *(5)*	67	67	D10	97	97
CLKUSR	69 *(3)*	69 *(5)*	C10	100 *(3)*	100 *(5)*
DATA *(3), (6)*	86	86	A6	125	125
TDI *(7)*	10	10	D2	13	13

Table 41. FLEX 6000 Device Pin-Outs (Part 2 of 2) *Notes (1), (2)*

Pin Name	100-Pin TQFP EPF6010A	100-Pin TQFP EPF6016A	100-Pin FineLine BGA EPF6010A EPF6016A	144-Pin TQFP EPF6010A	144-Pin TQFP EPF6016 EPF6016A EPF6024A
TDO *(7)*	51	51	K10	73	73
TCK	23 *(3)*	23 *(7)*, *(8)*	G3	34 *(3)*	34 *(7)*, *(8)*
TMS	18 *(3)*	18 *(7)*	G2	27 *(3)*	27 *(7)*
Dedicated Inputs	12, 13, 62, 63	12, 13, 62, 63	E1, E2, F9, F10	17, 20, 89, 92	17, 20, 89, 92
DEV_CLRn *(4)*	91	91	B5	130	130
DEV_OE *(4)*	85	85	B6	123	123
VCCINT	6, 21, 38, 54, 71, 88	6, 21, 38, 54, 71, 88	D7, E4, E5, F6, F7, G4	6, 31, 77, 103	6, 31, 77, 103
VCCIO	–	–	–	7, 19, 32, 55, 78, 91, 104, 127	7, 19, 32, 55, 78, 91, 104, 127
GND	5, 20, 37, 53, 70, 87	5, 20, 37, 53, 70, 87	D4, E6, E7, F4, F5, G7	5, 18, 30, 54, 76, 90, 102, 126	5, 18, 30, 54, 76, 90, 102, 126
No connect (N.C.)	3, 7, 19, 52, 55, 56, 68 *(9)*	–	–	3, 8, 9, 28, 29, 74, 75, 79, 80, 98, 99, 101 *(10)*	–
Total user I/O pins *(11)*	71	81	81	102	117

Table 42. FLEX 6000 Device Pin-Outs (Part 1 of 3) *Notes (1), (2)*

Pin Name	208-Pin PQFP EPF6016 EPF6016A EPF6024A	240-Pin PQFP EPF6016 EPF6024A	256-Pin BGA EPF6016	256-Pin BGA EPF6024A	256-Pin FineLine BGA EPF6016A	256-Pin FineLine BGA EPF6024A
MSEL *(3)*	46	52	T3	T3	L5	L5
nSTATUS *(3)*	80	92	W11	W11	K8	K8
nCONFIG *(3)*	77	89	Y11	Y11	N8	N8
DCLK *(3)*	184	212	C10	C10	G9	G9
CONF_DONE *(3)*	150	172	E18	E18	F12	F12
INIT_DONE *(4)*	135	155	J19	J19	H13	H13
nCE *(3)*	6	9	E1	E1	F5	F5
nCEO *(5)*	102	117	V18	V18	N12	N12
nWS *(5)*	169	195	B15	B15	F10	F10
nRS *(5)*	174	200	C13	C13	D10	D10
nCS *(5)*	159	184	B17	B17	D12	D12

Appendix D3

AMD EPROM Characteristics Tables and Waveform

CE# should be decoded and used as the primary device-selecting function, while OE#/V_{PP} be made a common connection to all devices in the array and connected to the READ line from the system control bus. This assures that all deselected memory devices are in their low-power standby mode and that the output pins are only active when data is desired from a particular memory device.

System Applications

During the switch between active and standby conditions, transient current peaks are produced on the rising and falling edges of Chip Enable. The magnitude of these transient current peaks is dependent on the output capacitance loading of the device. At a minimum, a 0.1 µF ceramic capacitor (high frequency, low inherent inductance) should be used on each device between V_{CC} and V_{SS} to minimize transient effects. In addition, to overcome the voltage drop caused by the inductive effects of the printed circuit board traces on EPROM arrays, a 4.7 µF bulk electrolytic capacitor should be used between V_{CC} and V_{SS} for each eight devices. The location of the capacitor should be close to where the power supply is connected to the array.

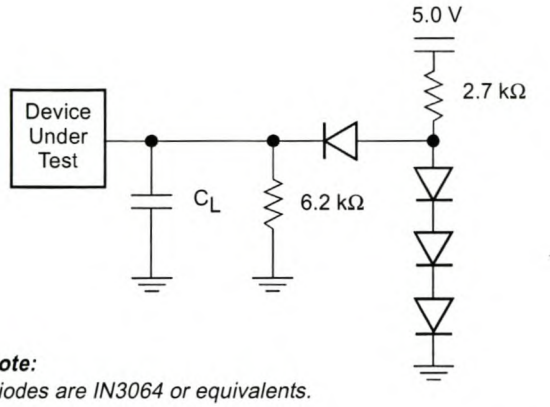
MODE SELECT TABLE

Mode		CE#	OE#/V_{PP}	A0	A9	Outputs
Read		V_{IL}	V_{IL}	X	X	D_{OUT}
Output Disable		X	V_{IH}	X	X	High Z
Standby (TTL)		V_{IH}	X	X	X	High Z
Standby (CMOS)		$V_{CC} \pm 0.3$ V	X	X	X	High Z
Program		V_{IL}	V_{PP}	X	X	D_{IN}
Program Verify		V_{IL}	V_{IL}	X	X	D_{OUT}
Program Inhibit		V_{IH}	V_{PP}	X	X	High Z
Autoselect (Note 3)	Manufacturer Code	V_{IL}	V_{IL}	V_{IL}	V_H	01h
	Device Code	V_{IL}	V_{IL}	V_{IH}	V_H	91h

Notes:

1. *V_H = 12.0 V ± 0.5 V.*
2. *X = Either V_{IH} or V_{IL}.*
3. *A1–A8 and A10–15 = V_{IL}*
4. *See DC Programming Characteristics for V_{PP} voltage during programming.*

TEST CONDITIONS



Note:
Diodes are IN3064 or equivalents.

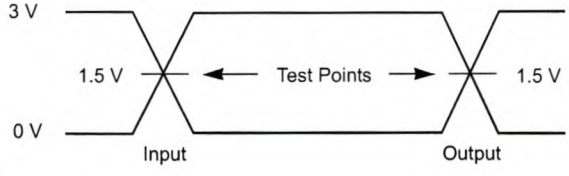
08140J-7

Figure 3. Test Setup

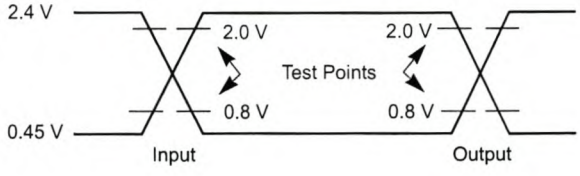
Table 1. Test Specifications

Test Condition	-55	All others	Unit
Output Load	1 TTL gate		
Output Load Capacitance, C_L (including jig capacitance)	30	100	pF
Input Rise and Fall Times	≤ 20		ns
Input Pulse Levels	0.0–3.0	0.45–2.4	V
Input timing measurement reference levels	1.5	0.8, 2.0	V
Output timing measurement reference levels	1.5	0.8, 2.0	V

SWITCHING TEST WAVEFORM



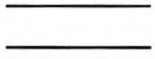
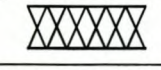
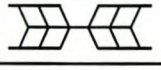
Note: *For C_L = 30 pF.*



Note: *For C_L = 100 pF.*

08140J-8

KEY TO SWITCHING WAVEFORMS

WAVEFORM	INPUTS	OUTPUTS
	Steady	
	Changing from H to L	
	Changing from L to H	
	Don't Care, Any Change Permitted	Changing, State Unknown
	Does Not Apply	Center Line is High Impedance State (High Z)

KS000010-PAL

AC CHARACTERISTICS

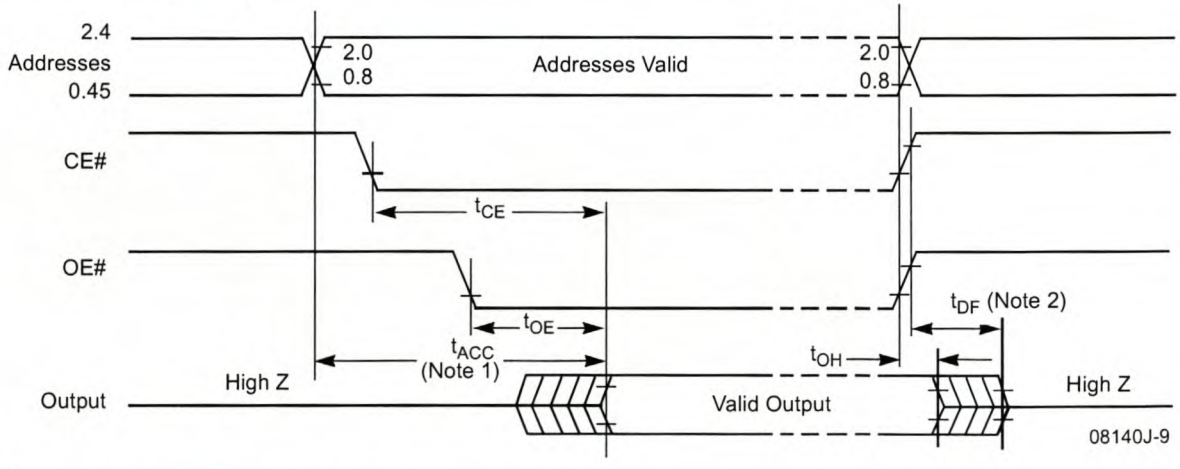
Parameter Symbols		Description	Test Setup		Am27C512							Unit
JEDEC	Standard				-55	-70	-90	-120	-150	-200	-255	
t_{AVQV}	t_{ACC}	Address to Output Delay	CE#, OE# = V_{IL}	Max	55	70	90	120	150	200	250	ns
t_{ELQV}	t_{CE}	Chip Enable to Output Delay	OE# = V_{IL}	Max	55	70	90	120	150	200	250	ns
t_{GLQV}	t_{OE}	Output Enable to Output Delay	CE# = V_{IL}	Max	35	40	40	50	50	75	75	ns
t_{EHQZ} t_{GHQZ}	t_{DF} (Note 2)	Chip Enable High or Output Enable High to Output High Z, Whichever Occurs First		Max	25	25	25	30	30	30	30	ns
t_{AXQX}	t_{OH}	Output Hold Time from Addresses, CE# or OE#, Whichever Occurs First		Min	0	0	0	0	0	0	0	ns

Caution: *Do not remove the device from (or insert it into) a socket or board that has V_{PP} or V_{CC} applied.*

Notes:

1. *V_{CC} must be applied simultaneously or before V_{PP}, and removed simultaneously or after V_{PP}.*
2. *This parameter is sampled and not 100% tested.*
3. *Switching characteristics are over operating range, unless otherwise specified.*
4. *See Figure 3 and Table 1 for test specifications.*

SWITCHING WAVEFORMS



Notes:

1. *OE# may be delayed up to $t_{ACC} - t_{OE}$ after the falling edge of the addresses without impact on t_{ACC}.*
2. *t_{DF} is specified from OE# or CE#, whichever occurs first.*

PACKAGE CAPACITANCE

Parameter Symbol	Parameter Description	Test Conditions	CDV028		PL 032		PD 028		Unit
			Typ	Max	Typ	Max	Typ	Max	
C_{IN}	Input Capacitance	V_{IN} = 0	10	12	9	12	6	10	pF
C_{OUT}	Output Capacitance	V_{OUT} = 0	10	13	9	12	6	10	pF

Notes:

1. *This parameter is only sampled and not 100% tested.*
2. *T_A = +25°C, f = 1 MHz.*

Appendix D4

Samsung SRAM Functional Tables and Waveform

PRODUCT LIST

Commercial Temperature Products (0~70°C)		Extended Temperature Products (-25~85°C)		Industrial Temperature Products (-40~85°C)	
Part Name	**Function**	**Part Name**	**Function**	**Part Name**	**Function**
KM681000BLP-5	32-DIP,55ns,L-pwr	KM681000BLGE-7	32-SOP,70ns,L-pwr	KM681000BLGI-7L	32-SOP,70ns,LL-pwr
KM681000BLP-5L	32-DIP,55ns,LL-pwr	KM681000BLGE-7L	32-SOP,70ns,LL-pwr	KM681000BLTI-7L	32-TSOP F,70ns,LL-pwr
KM681000BLP-7	32-DIP,70ns,L-pwr	KM681000BLTE-7L	32-TSOP F,70ns,LL-pwr	KM681000BLRI-7L	32-TSOP R,70ns,LL-pwr
KM681000BLP-7L	32-DIP,70ns,LL-pwr	KM681000BLRE-7L	32-TSOP R,70ns,LL-pwr		
KM681000BLG-5	32-SOP,55ns,L-pwr				
KM681000BLG-5L	32-SOP,55ns,LL-pwr				
KM681000BLG-7	32-SOP,70ns,L-pwr				
KM681000BLG-7L	32-SOP,70ns,LL-pwr				
KM681000BLT-5L	32-TSOP F,55ns,LL-pwr				
KM681000BLT-7L	32-TSOP F,70ns,LL-pwr				
KM681000BLR-5L	32-TSOP R,55ns,LL-pwr				
KM681000BLR-7L	32-TSOP R,70ns,LL-pwr				

Note : LL means Low Low standby current.

FUNCTIONAL DESCRIPTION

$\overline{CS_1}$	CS_2	$\overline{OE}$	$\overline{WE}$	I/O Pin	Mode	Power
H	X[1)]	X[1)]	X[1)]	High-Z	Deselected	Standby
X[1)]	L	X[1)]	X[1)]	High-Z	Deselected	Standby
L	H	H	H	High-Z	Output Disabled	Active
L	H	L	H	Dout	Read	Active
L	H	X[1)]	L	Din	Write	Active

1. X means don't care.(Must be low or high state.)

ABSOLUTE MAXIMUM RATINGS[1)]

Item	Symbol	Ratings	Unit	Remark
Voltage on any pin relative to Vss	V_{IN},V_{OUT}	-0.5 to 7.0	V	-
Voltage on Vcc supply relative to Vss	V_{CC}	-0.5 to 7.0	V	-
Power Dissipation	P_D	1.0	W	-
Storage temperature	T_{STG}	-65 to 150	°C	-
Operating Temperature	T_A	0 to 70	°C	KM681000BL
		-25 to 85	°C	KM681000BLE
		-40 to 85	°C	KM681000BLI
Soldering temperature and time	T_{SOLDER}	260°C, 10sec (Lead Only)	-	-

1. Stresses greater than those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. Functional operation should be restricted to recommended operating condition. Exposure to absolute maximum rating conditions for extended periods may affect reliability.

RECOMMENDED DC OPERATING CONDITIONS[1)]

Item	Symbol	Min	Typ	Max	Unit
Supply voltage	V_{CC}	4.5	5.0	5.5	V
Ground	V_{SS}	0	0	0	V
Input high voltage	V_{IH}	2.2	-	V_{CC}+0.5[2)]	V
Input low voltage	V_{IL}	-0.5[3)]	-	0.8	V

Note
1. Commercial Product : T_A=0 to 70°C, unless otherwise specified
   Extended Product : T_A=-25 to 85°C, unless otherwise specified
   Industrial Product : T_A=-40 to 85°C, unless otherwise specified
2. Overshoot : Vcc+3.0V in case of pulse width≤30ns
3. Undershoot : -3.0V in case of pulse width≤30ns
4. Overshoot and undershoot are sampled, not 100% tested

CAPACITANCE[1)] (f=1MHz, T_A=25°C)

Item	Symbol	Test Condition	Min	Max	Unit
Input capacitance	C_{IN}	V_{IN}=0V	-	6	pF
Input/Output capacitance	C_{IO}	V_{IO}=0V	-	8	pF

1. Capacitance is sampled, not 100% tested

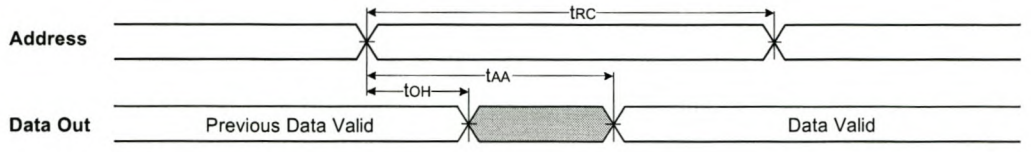
DC AND OPERATING CHARACTERISTICS

Item	Symbol	Test Conditions		Min	Typ	Max	Unit
Input leakage current	I_{LI}	V_{IN}=Vss to Vcc		-1	-	1	μA
Output leakage current	I_{LO}	$\overline{CS_1}$=V_{IH} or CS_2=V_{IL} or $\overline{OE}$=V_{IH} or $\overline{WE}$=V_{IL}, V_{IO}=Vss to Vcc		-1	-	1	μA
Operating power supply	I_{CC}	$\overline{CS_1}$=V_{IL}, CS_2=V_{IH}, I_{IO}=0mA, V_{IN}= V_{IL} or V_{IH}		-	7	15[1)]	mA
Average operating current	I_{CC1}	Cycle time=1μs, 100% duty, I_{IO}=0mA, $\overline{CS_1}$≤0.2V, CS_2≥Vcc-0.2V, V_{IN}≤0.2V or V_{IN}≥Vcc-0.2V		-	-	10[2)]	mA
	I_{CC2}	Cycle time=Min, 100% duty, I_{IO}=0mA, $\overline{CS_1}$=V_{IL}, CS_2=V_{IH}, V_{IN}=V_{IL} or V_{IH}		-	-	70	mA
Output low voltage	V_{OL}	I_{OL}=2.1mA		-	-	0.4	V
Output high voltage	V_{OH}	I_{OH}=-1.0mA		2.4	-	-	V
Standby Current(TTL)	I_{SB}	$\overline{CS_1}$=V_{IH}, CS_2=V_{IL}, Other input=V_{IL} or V_{IH}		-	-	3	mA
Standby Current(CMOS)	I_{SB1}	$\overline{CS_1}$≥Vcc-0.2V, CS_2≥Vcc-0.2V or CS_2≤0.2V Other input=0~Vcc	KM681000BL KM681000BL-L	- -	- -	100 20	μA
			KM681000BLE KM681000BLE-L	- -	- -	100 50	μA
			KM681000BLI KM681000BLI-L	- -	- -	100 50	μA

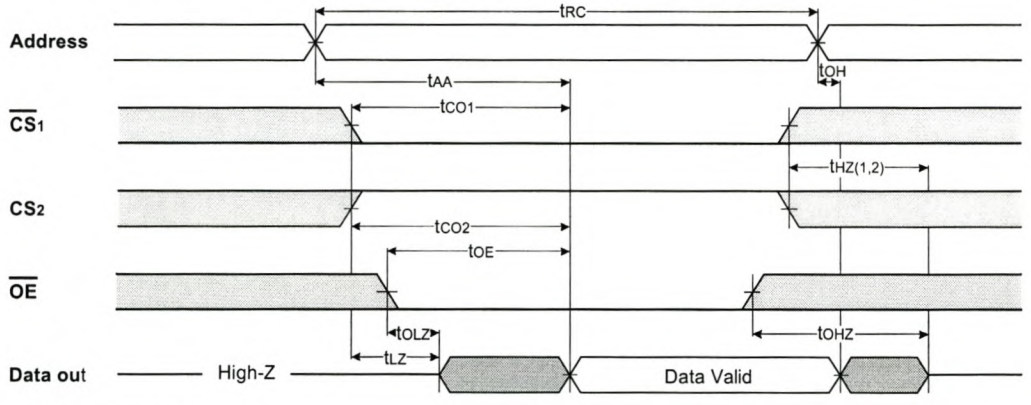
1. 20mA for Exteneded and Industrial Products
2. 15mA for Extended and Industrial Products

TIMMING DIAGRAMS

TIMING WAVEFORM OF READ CYCLE(1) (Address Controlled, $\overline{CS}1=\overline{OE}=V_{IL}$, $\overline{WE}=V_{IH}$)



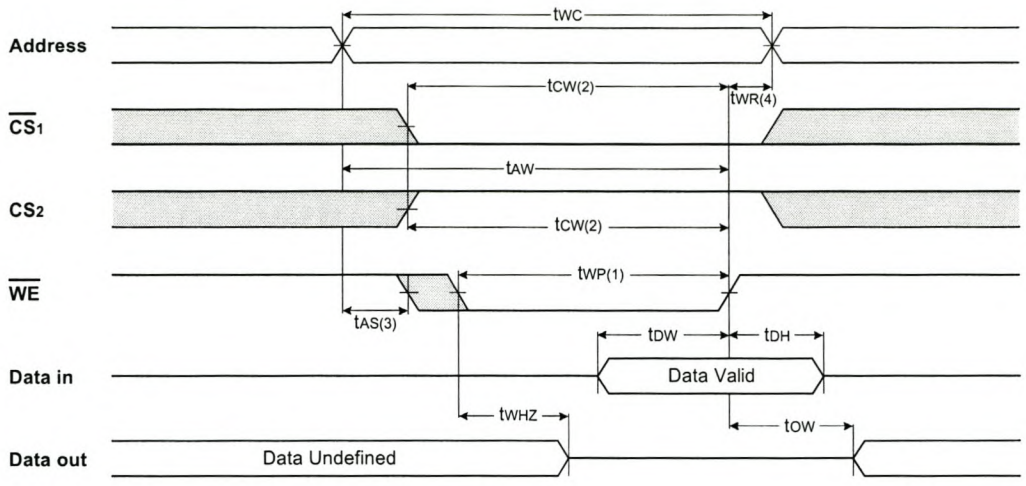
TIMING WAVEFORM OF READ CYCLE(2) ($\overline{WE}=V_{IH}$)



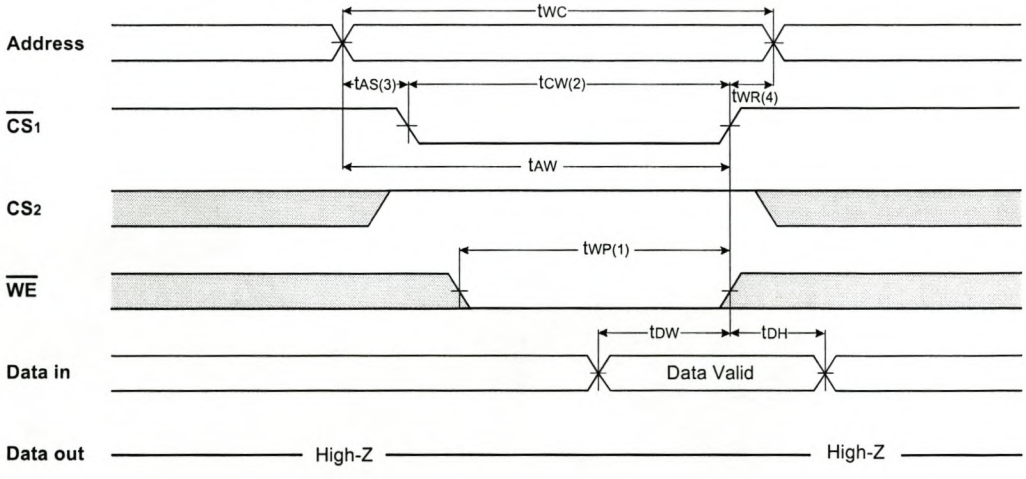
NOTES (READ CYCLE)

1. t_{HZ} and t_{OHZ} are defined as the time at which the outputs achieve the open circuit conditions and are not referenced to output voltage levels.
2. At any given temperature and voltage condition, t_{HZ}(Max.) is less than t_{LZ}(Min.) both for a given device and from device to device interconnection.

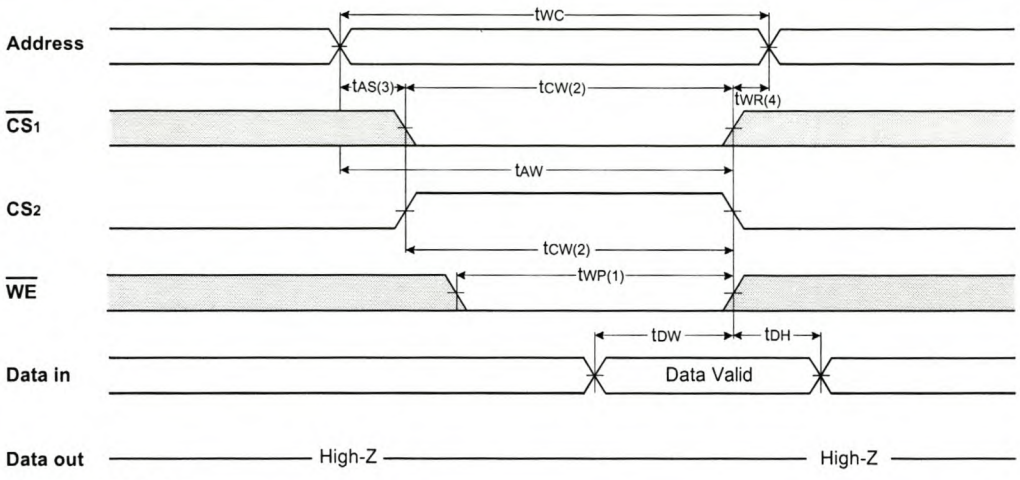
TIMING WAVEFORM OF WRITE CYCLE(1) ($\overline{WE}$ Controlled)



TIMING WAVEFORM OF WRITE CYCLE(2) ($\overline{CS_1}$ Controlled)



TIMING WAVEFORM OF WRITE CYCLE(3) (CS2 Controlled)

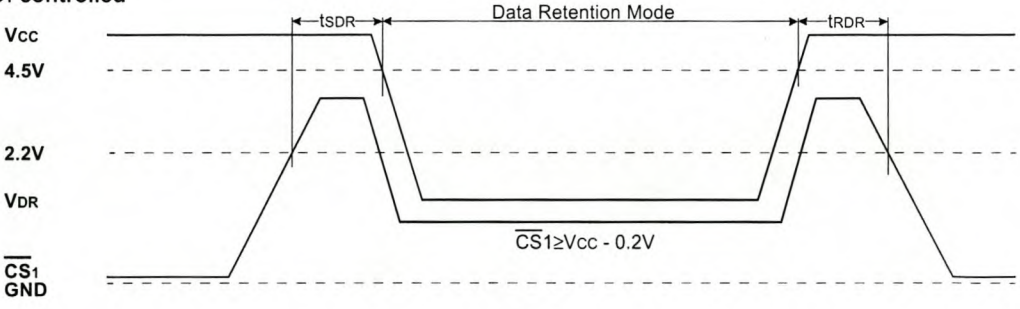


NOTES (WRITE CYCLE)

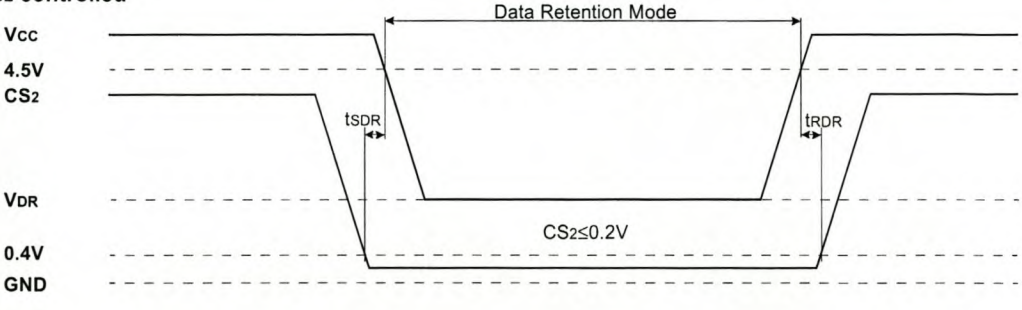
1. A write occurs during the overlap of a low $\overline{CS_1}$, a high CS_2 and a low $\overline{WE}$. A write begins at the latest transition among $\overline{CS_1}$ goes low, CS_2 going high and $\overline{WE}$ going low : A write end at the earliest transition among $\overline{CS_1}$ going high, CS_2 going low and $\overline{WE}$ going high, t_{WP} is measured from the beginning of write to the end of write.
2. t_{CW} is measured from the $\overline{CS_1}$ going low or CS_2 going high to the end of write.
3. t_{AS} is measured from the address valid to the beginning of write.
4. t_{WR} is measured from the end of write to the address change. $t_{WR(1)}$ applied in case a write ends as $\overline{CS_1}$ or $\overline{WE}$ going high $t_{WR(2)}$ applied in case a write ends as CS_2 going to low.

DATA RETENTION WAVE FORM

$\overline{CS_1}$ controlled



CS_2 controlled



Appendix D5

Philips Octal D-Latch Functional Tables

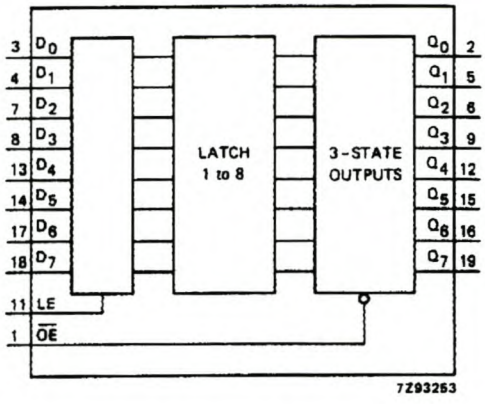


Fig.4 Functional diagram.

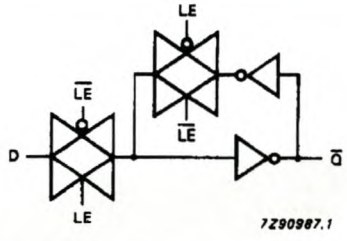


Fig.5 Logic diagram (one latch).

FUNCTION TABLE

OPERATING MODES	INPUTS			INTERNAL LATCHES	OUTPUTS
	$\overline{OE}$	LE	D_n		Q_0 to Q_7
enable and read register (transparent mode)	L	H	L	L	L
	L	H	H	H	H
latch and read register	L	L	l	L	L
	L	L	h	H	H
latch register and disable outputs	H	X	X	X	Z
	H	X	X	X	Z

Notes

1. H = HIGH voltage level
   h = HIGH voltage level one set-up time prior to the HIGH-to-LOW LE transition
   L = LOW voltage level
   l = LOW voltage level one set-up time prior to the HIGH-to-LOW LE transition
   X = don't care
   Z = high impedance OFF-state

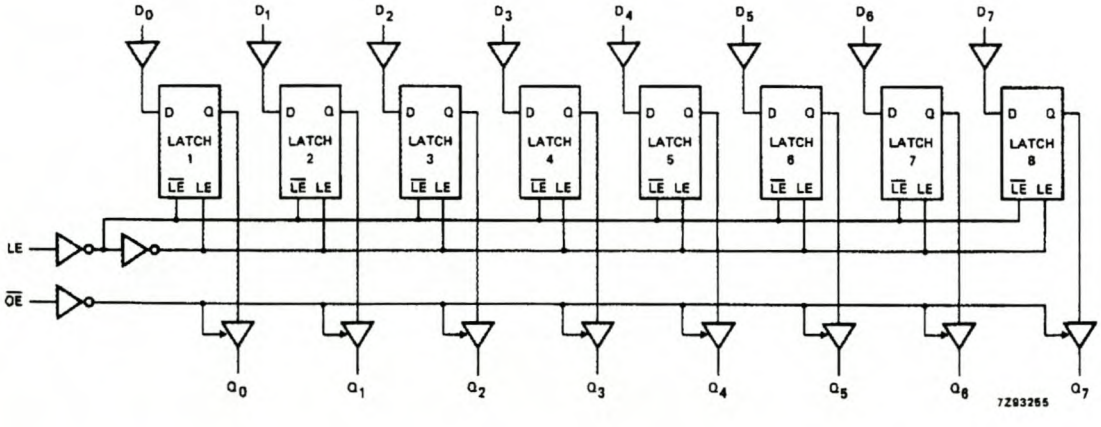


Fig.6 Logic diagram.

Appendix D6

Philips CAN Transceiver Functional Tables

FUNCTIONAL DESCRIPTION

The PCA82C250 is the interface between the CAN protocol controller and the physical bus. It is primarily intended for high-speed applications (up to 1 Mbaud) in cars. The device provides differential transmit capability to the bus and differential receive capability to the CAN controller. It is fully compatible with the *"ISO/DIS 11898"* standard.

A current limiting circuit protects the transmitter output stage against short-circuit to positive and negative battery voltage. Although the power dissipation is increased during this fault condition, this feature will prevent destruction of the transmitter output stage.

If the junction temperature exceeds a value of approximately 160 °C, the limiting current of both transmitter outputs is decreased. Because the transmitter is responsible for the major part of the power dissipation, this will result in a reduced power dissipation and hence a lower chip temperature. All other parts of the IC will remain in operation. The thermal protection is particularly needed when a bus line is short-circuited.

The CANH and CANL lines are also protected against electrical transients which may occur in an automotive environment. Pin 8 (Rs) allows three different modes of operation to be selected: high-speed, slope control or standby.

For high-speed operation, the transmitter output transistors are simply switched on and off as fast as possible. In this mode, no measures are taken to limit the rise and fall slope. Use of a shielded cable is recommended to avoid RFI problems. The high-speed mode is selected by connecting pin 8 to ground.

For lower speeds or shorter bus length, an unshielded twisted pair or a parallel pair of wires can be used for the bus. To reduce RFI, the rise and fall slope should be limited. The rise and fall slope can be programmed with a resistor connected from pin 8 to ground. The slope is proportional to the current output at pin 8.

If a HIGH level is applied to pin 8, the circuit enters a low current standby mode. In this mode, the transmitter is switched off and the receiver is switched to a low current. If dominant bits are detected (differential bus voltage >0.9 V), RxD will be switched to a LOW level. The microcontroller should react to this condition by switching the transceiver back to normal operation (via pin 8). Because the receiver is slow in standby mode, the first message will be lost.

Table 1 Truth table of CAN transceiver

SUPPLY	TxD	CANH	CANL	BUS STATE	RxD
4.5 to 5.5 V	0	HIGH	LOW	dominant	0
4.5 to 5.5 V	1 (or floating)	floating	floating	recessive	1
<2 V (not powered)	X	floating	floating	recessive	X
2 V < V_{CC} < 4.5 V	>0.75V_{CC}	floating	floating	recessive	X
2 V < V_{CC} < 4.5 V	X	floating if V_{Rs} > 0.75V_{CC}	floating if V_{Rs} > 0.75V_{CC}	recessive	X

Table 2 Rs (pin 8) summary

CONDITION FORCED AT Rs	MODE	RESULTING VOLTAGE OR CURRENT AT Rs
V_{Rs} > 0.75V_{CC}	standby	I_{Rs} < |10 μA|
−10 μA < I_{Rs} < −200 μA	slope control	0.4V_{CC} < V_{Rs} < 0.6V_{CC}
V_{Rs} < 0.3V_{CC}	high-speed	I_{Rs} < −500 μA