Mobile Radio data Network for Documentation Display

René du Toit

Thesis presented in partial fulfillment of the requirements for the degree ***Master of Science in Engineering*** *in the Department of Electrical and Electronic Engineering at the University of Stellenbosch.*

Supervisor:Prof. P.J. Bakkes

March 2000

Declaration

"I, the undersigned, hereby declare that the work contained in this thesis is my own original work, unless stated otherwise, and that I have not previously, in its entirety or in part, submitted it at any university for a degree."

R. du Toit
June 20, 2000

Abstract

This product is designed as an educational tool. It was developed to make a classroom more organised. Better and easier communication between the teacher and the students can be established, as well as improved communication between different classrooms. The stand alone unit can be used anywhere because there is no wiring between this and the computer. It can be a big addition to the educational system, especially in places where books are not readily available for the students.

The system consists of various components. These components are a computer, a stand alone unit and an interconnection for these two. The interconnection was incorporated with the use of radio links. At the computer side of the design, a windows interface was developed to enable the computer user to communicate with the stand alone unit user. The radio link was inserted to make the communication between the computer and the stand alone unit wireless. The stand alone unit consists of a keypad, liquid crystal display, serial port and a floppy drive. The serial port is used as connection between the stand alone unit and the radio links.

It is possible to expand the system. The keypad can be replaced by a keyboard to enable the stand alone user to type and save data on the stand alone unit itself, and a hard drive might be added to increase the storage space. The inclusion of the floppy drive makes the system more compact and creates the possibility of transferring data by use of the floppy, which in turn does not limit a person to working only in one place.

Opsomming

Hierdie produk is ontwerp as 'n hulpmiddel vir die onderwys stelsel. Dit is ontwerp om 'n klaskamer meer georganiseerd te maak. Beter en makliker kommunikasie tussen onderwysers en studente word op hierdie manier verseker, asook verbeterde kommunikasie tussen verskillende klaskamers. Die alleenstaande eenheid kan enige plek gebruik word, aangesien daar geen bedrading is tussen die eenheid en die rekenaar nie. Dit kan 'n groot bydrae wees tot die onderwysstelsel, veral in plekke waar boeke nie so alledaags beskikbaar is nie.

Die stelsel bestaan uit verskeie komponente. Hierdie komponente is 'n rekenaar, 'n alleenstaande eenheid en 'n verbinding tussen hierdie twee. Die verbinding tussen hierdie twee is met behulp van radio skakels gedoen. By die rekenaar kant van die ontwerp, is 'n rekenaar program geskryf wat die kommunikasie tussen die rekenaar en die alleenstaande eenheid beheer. Die radio skakel is ingesluit in die ontwerp om die kommunikasie tussen die rekenaar en die alleenstaande eenheid draadloos te hou. Die alleenstaande eenheid bestaan uit 'n miniatuur sleutelbord, vloeibare kristal vertoon, 'n serie poort en 'n slapskyf dryf. Die serie poort word gebruik as verbinding tussen die alleenstaande eenheid en die radio skakels.

Dit is moontlik om die stelsel te vergroot. Die miniatuur sleutelbord kan vervang word deur 'n standaard sleutelbord om die alleenstaande eenheid gebruiker in staat te stel om data op die eenheid self te tik en te stoor. 'n Hardeskyf kan ook ingevoeg word om die stoor spasie van die stelsel te vergroot. Die insluiting van die slapskyf veroorsaak dat die stelsel meer kompak is en ontwikkel die moontlikheid om die data oor te dra met behulp van die skyf, wat weer 'n persoon in staat stel om op meer as slegs een plek te werk.

Acknowledgements

I would like to give gratitude to the following:

Professor Bakkes for his leadership and help.
Christoffel and Estelle du Toit for their support and proof reading.
My friends and co-students for their support.
God.

Contents

List of Abbreviations **9**

List of Figures **11**

List of Tables **13**

1 Introduction **14**

2 Background Principles **17**

2.1 Floppy Drive 17

2.1.1 Types of Floppy Drives 17

2.1.2 Floppy Disk Formats 17

2.2 Communication Protocol 19

2.2.1 TDMA 21

2.2.2 FDMA 21

2.2.3 ALOHA 21

2.2.4 CSMA 22

2.2.5 CDMA 23

2.3 Conclusion 25

3 Design Considerations **26**

3.1 Microprocessor 26

3.2 Communication Protocol 27

3.3 Control of a Floppy Disk Drive 27

3.4 Other 28

3.4.1 LCD 28

3.4.2 Keypad 28

3.4.3 Interfaces 28

4 Radio link Communication **30**

4.1 Fundamentals of the RPC 30

4.1.1 Functional Description 30

4.1.2 Switches 31

4.1.3 User Configurable EEPROM 34
4.2 Interface to Host 34
4.2.1 Signals of the RPC 34
4.2.2 Data Transfer 36
4.3 Conclusion 40

5 Stand-alone Unit **41**
5.1 Developement Phase 41
5.1.1 Size 42
5.1.2 Operation 42
5.1.3 Compatibility 42
5.2 Conceptual Design Phase 42
5.2.1 Microcontroller 43
5.2.2 Screen 43
5.2.3 Floppy Drive 43
5.2.4 Computer 43
5.2.5 Keypad 44
5.3 Keypad Interface 44
5.4 LCD Interface 48
5.5 CPLD 49
5.6 RS232 50
5.7 Floppy Disk Controller 50
5.7.1 DMA and NON-DMA 50
5.7.2 Base, Special and PC-AT mode 51
5.7.3 Registers of the FDC 52
5.7.4 Command Parameters 54
5.7.5 Implementation 55
5.8 Problems encountered in the Design 57
5.9 Conclusion 60

6 Windows Interface **61**
6.1 Developement 61
6.1.1 Text Editor 61
6.1.2 Serial Communication 62
6.1.3 Database 63
6.1.4 About Box 63
6.2 Main Interface 63
6.2.1 Parent Form 64
6.2.2 Child Form 66
6.3 Serial Communication 67

6.3.1 Connect, Send and Disconnect 68
6.3.2 Unit Type 68
6.3.3 User Name 68
6.3.4 File Name 68
6.4 Database 69
6.4.1 TDBNavigator 69
6.4.2 TDBEdit Boxes 69
6.4.3 TDBGrid 70
6.4.4 TDataSource 70
6.4.5 TTable 70
6.5 Conclusion 70

7 Conclusion **71**

A Source Code for SAU **73**
A.1 Main Program 74
A.2 Keypad Program 75
A.3 LCD Program 80
A.4 FLOPPY Program 86
A.5 RS232 Program 97
A.6 Radio Link Program 100
A.7 VHDL Program 104

B Users Manual for SAU **106**
B.1 Installation 107
B.2 Setup 107
B.3 Use 107

C Source Code for Windows Interface **109**
C.1 Parent form 110
C.2 Child form 115
C.3 Serial communication 119
C.4 Data units 122
C.5 Serial interface 123

D Users Manual for Windows Interface **127**
D.1 Hardware Installation 128
D.2 Software Installation 128
D.3 Use 128

E WD37C65B Floppy Disk Subsystem Controller **130**

E.1 Status Registers 131

E.2 Commands 134

List of Abbreviations

A0	Address 0
ACK	Acknowledge
BCS	Binary Code Sequence
CDMA	Code Division Multiple Access
CPLD	Complex Programmable Logic Device
CR	Control Register
CSMA/CD	Carrier Sample Multiple Access with Collision Detection
CSMA	Carrier Sense Multiple Access
DAV	Data Available
DBT	Delay Before Transmit
DIO	Data Input
DMA	Direct Memory Access
DR	Data Register
DRV	Drive Type
DS-CDMA	Direct Sense Code Division Multiple Access
EEPROM	Electrically Eraseable Programmable Read Only Memory
ETX	End Of Transmission
FDC	Floppy Disk Controller
FDD	Floppy Disk Drive
FDMA	Frequency Division Multiple Access
FFH-CDMA	Fast Frequency Hopping Code Division Multiple Access
FH-CDMA	Frequency Hopping Code Division Multiple Access
GND	Ground
ID	Identification
IDE	Integrated Drive Electronics
I/O	Input/Output
IRQ	Interrupt
LBT	Listen Before Transmit
LCD	Liquid Crystal Display
LSB	Least Significant Bit
LSN	Least Significant Nibble

MAP	Multiple Access Protocol
MSB	Most Significant Bit
MSN	Most Significant Nibble
MSR	Master Status Register
OFT	Off Time
ONT	On Time
OR	Operations Register
p-Aloha	Pure Aloha
PC	Personal Computer
r-Aloha	Reservation Aloha
RPC	Radio Packet Controller
RQM	Request For Master
RX	RPC Transmission
RXA	RPC Transmission Accept
RXR	RPC Transmission Request
R/W	Read/Write
s-Aloha	Slotted Aloha
SAU	Stand Alone Unit
SFH-CDMA	Slow Frequency Hopping Code Division Multiple Access
SOH	Start Of Header
TDMA	Time Division Multiple Access
TC	Terminal Count
TH-CDMA	Time Hopping Code Division Multiple Access
TX	Host Transmission
TXA	Host Transmission Accept
TXR	Host Transmission Request
VHDL	Very high speed integrated circuit Hardware Description Language
WAP	Wireless Application Protocol

List of Figures

1.1 Integration Concepts (system setup) 14

2.1 Classification of Multiple Access Protocols 20
2.2 Frames and Time slot illustration for TDMA 21

3.1 Blockdiagram of the SAU 26

4.1 Switches found in the RPC 32
4.2 Interface between the RPC and the Host 35
4.3 Configuration of Control byte for Data Packet 36
4.4 Configuration of Control byte for Memory Access 37
4.5 TX download timing diagram 38
4.6 TX flowdiagram of the code implementation 38
4.7 RX upload time diagram 39
4.8 RX flowdiagram of the code implementation 39

5.1 SAU Appearance 42
5.2 SAU Conceptual Design 43
5.3 Original SAU Detail Schematic 45
5.4 Keypad Interface 46
5.5 Block diagram for Keypad Interface 47
5.6 LCD interface 48
5.7 Display position and DD RAM address for PC2004-A 49
5.8 RS232 interface 50
5.9 Flow Diagram illustrating the relationship between Base, Special and PC-AT mode 51
5.10 Floppy drive interface 55
5.11 Floppy drive pin connectors 55
5.12 Flow Diagram illustrating the source code implemented for the FDC 57
5.13 Revised SAU Detail Schematic 59

6.1 Text Editor appearance 62
6.2 Serial Communication interface 62

6.3 Data Units 63
6.4 Data Units 63
6.5 Parent form 64
6.6 Main menu items: (a) File, (b)Edit, (a) File, (b)Edit and (b)Edit. 65
6.7 Child form 67
6.8 Serconnect form at design time 67
6.9 Data Units 69
6.10 DBNavigator Bar 69

B.1 SAU Layout 107

D.1 Data Units 128
D.2 Send... Window 129

List of Tables

2.1 Various disk format Specifications 19

4.1 Codes generated by MM74C922 and the interpretation by the AT89C52 33
4.2 RPC Registers 34
4.3 Signals for Host interface 35
4.4 Packet type as depicted by PT (bit 7 of the control byte) 36

5.1 Codes generated by MM74C922 and the interpretation by the AT89C55 46
5.2 Instruction set for initialisation for PC2004-A 49
5.3 Address decoding for the CPLD 49
5.4 Master Status Register 52
5.5 Settings of DIO and RQM 53
5.6 Control Register Options 53
5.7 Data Register 53
5.8 The Commands of the Wd37C65B 54
5.9 Internal Diskdrive 56

E.1 Status Register 0 131
E.2 Status Register 1 132
E.3 Status Register 2 133
E.4 Status Register 3 133
E.5 Read Data 134
E.6 Write Data 134
E.7 Read ID 134
E.8 Format A Track 135
E.9 Recalibrate 135
E.10 Sense Interrupt Status 135
E.11 Specify 135
E.12 Sense Drive Status 135
E.13 Seek 135
E.14 Command Symbol Descriptions 136

Chapter 1

Introduction

The need for easy access to educational tools in school systems is very clear, especially in Southern Africa. This project was developed mainly to improve the organising in classrooms. With this product, there can be easy communications between the teacher and the students, as well as between different classrooms.

The entire concept of this product is very simple. The product consists of a Stand Alone Unit (SAU) and a Personal Computer (PC), with the inclusion of a modem or a server. Each student has a SAU of his or her own and the teacher has a PC and modem connection. The communication between the SAUs and the PC is via Radio links. Communication between two classrooms can be established either by using a server (if one is connected) or a modem at both PC-terminals.

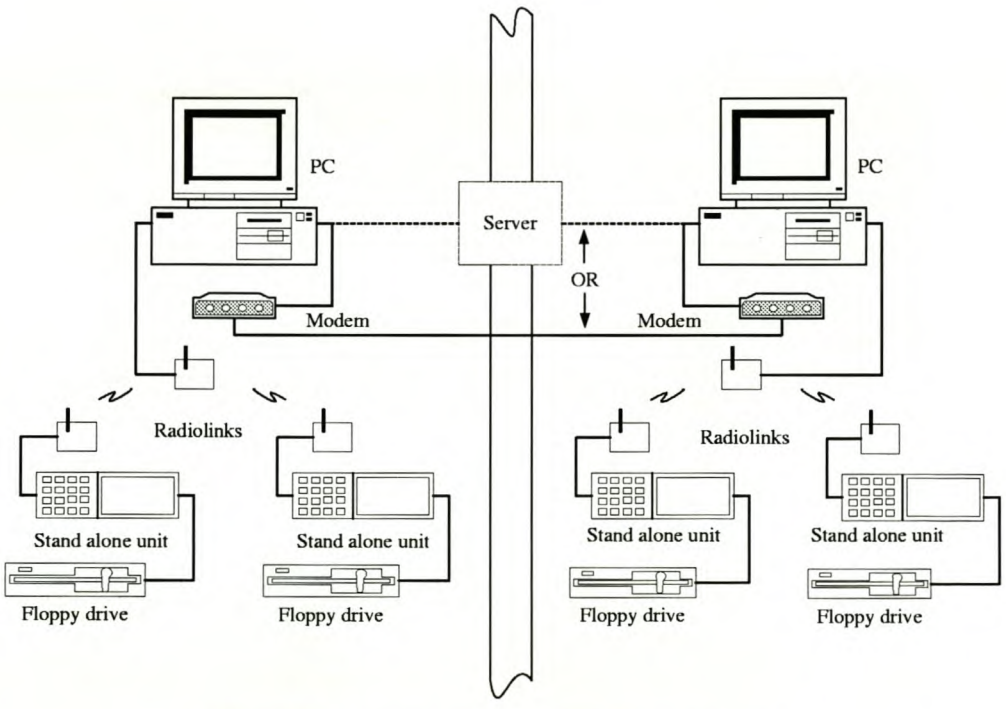

Figure 1.1: Integration Concepts (system setup)

The SAU is a fairly simple integration of components. It consists of a keypad, Liquid Crystal Display (LCD), Serial Port and a Floppy drive. The Serial Port is used as connection between the SAU and the Radio links. The software for communication between PC to PC and PC to SAU is in the PC Program. Figure 1.1 illustrates the integration concepts (system setup) of the systems.

The inclusion of radio links implies remote use of the system. This will save money, because there is no need for wiring between the SAUs and PC. It is easily upgraded by replacing the Floppy Drive with for instance a Hard Drive. It is also possible to replace the keypad with a keyboard which will enable the SAU user to type his/her own text files on the SAU.

An added advantage of using the radio links is, that if the SAU is used in conjuction with radio links that have higher transmission and reception distances, it can be used at remote classrooms.

Not only can this system be used for educational purposes, but also for many other types of communication between a PC and a SAU. It can also be used between different PCs in the same room. Another possible variation to the use of the system is where a Main Computer is used to communicate through a PC to the SAU. This operation can be done with or without an attendant at the PC.

The system was mainly designed for use in a classroom. It is therefore important that the system be as simple and efficient as possible. The SAU is very user friendly and easy to operate.

Although a SAU cannot replace a PC, it can serve the same purpose at much less cost. Adding or updating of a syllabus can easily be done on the PC and then transmitted to the SAU. Information sent by the PC to the SAU can be saved on a stiffy by the SAU user for later use.

In conclusion, a short introduction to each of the remaining chapters of this document.

Chapter 2 discusses the Background principles that was studied and investigated for this project. It consists of a detailed explanation of the available Floppy drives and their format, and also the different Communication Protocols that was considered.

In Chapter 3 the various decisions made during the developement of the project are explained and motivated. The decisions included in this chapter are:

- Which Microprocessor to use
- Which of the discussed Protocols to implement
- Which type of control to use for the Floppy Drive, as well as which type of Floppy Drive to use

The Radio links used, are discussed in Chapter 4. This chapter includes a discussion on the fundamental functioning of the radio links and then, of course, the interface to the host. The source code that was developed for the use of the radio links, are included in Appendix A.6.

The design and integration of the Stand Alone Unit is discussed in Chapter 5. All the various components added to the unit are discussed as well as their integration into the system. The source code written for the unit is included in Appendix A. A User Manual for the SAU is also included in Appendix B. More information on the Floppy Disk Controller can be found in Appendix E.

Chapter 6 introduces the Windows Interface of the project to the reader, how it was developed and constructed as well as some information on the various components used to build this application. The source code developed for the interface, can be found in Appendix C, and the User Manual on how to use the program can be found in Appendix D.

Finally Chapter 7 concludes the document with a brief discussion on all the information and detail included in this document. It also consists of a few recommendations on how to improve the design.

Chapter 2

Background Principles

Since the concepts required for the developement of this project were unknown, it was necessary to undertake intensive research. The largest part of this is the study done on the Floppy Drive and the Communication Protocol. In this chapter, the study conducted in these two general fields are presented. These studies were carried out to determine what equipment or components should be used in the design of the project.

2.1 Floppy Drive

The Floppy drive is a very valuable storage device. It can be used to store any type of data in a useful and compact manner. The floppy drive in itself could still be in use for at least another five to ten years from date. It is easily accessible and is a cheap storage medium. In this situation it was used to store and read text data.

2.1.1 Types of Floppy Drives

There are many different types of floppy drives available, but most of these are already outdated. The most commonly used is of course the $3\frac{1}{2}$" floppy drive. The only other drive of note is the $5\frac{1}{4}$" drive, although this drive's predecessor, the 8" drive, can still be found in some non-PC equipment. The $5\frac{1}{4}$" drive is rarely, if ever, used in computer applications these days. It is still available, but it is definitely not compatible with all computers, whereas the $3\frac{1}{2}$" floppy drive is certain to be found in almost all computers.

2.1.2 Floppy Disk Formats

In addition to the different types of Floppy Drives available, there are also different drive formats. These differ only in the size of the writable discs. The different formats available, are the following:

- 360 KB $5\frac{1}{4}$" Floppy
- 1.2 MB $5\frac{1}{4}$" Floppy
- 720 KB $3\frac{1}{2}$" Floppy

- 1.44 MB $3\frac{1}{2}$" Floppy
- 2.88 MB $3\frac{1}{2}$" Floppy

360 KB $5\frac{1}{4}$" Floppy

This drive format is one of the oldest formats used. It was developed as replacement for the 8" drive format, because it was just too large. The 360 KB $5\frac{1}{4}$" floppy is capable of storing 360 KB of data. Not very much, judged by today's standards.

1.2 MB $5\frac{1}{4}$" Floppy

This was the improved version of the 360 KB $5\frac{1}{4}$" floppy. It has a storage capacity of 1.2 MB as apposed to the 360 KB of its predecessor. It is exactly the same size as the 360 KB format, but it has a total of 80 tracks in comparison with the 40 tracks of the previous version. The sectors per track was also increased from 9 to 15, and the spindle speed of the 1.2 MB floppy is 60 RPM faster. The increase in speed is from 300 RPM to 360 RPM.

720 KB $3\frac{1}{2}$" Floppy

One of the biggest drawbacks of the $5\frac{1}{4}$" disk format, is its size. This also increased the probability of damage, as the protective cover of this oversized disk is rather soft. It was therefore necessary to create a smaller more effectively protected format. The first of these was the 720 KB $3\frac{1}{2}$" format. Even though the storage space of this format is less than that of the 1.2 MB $5\frac{1}{4}$" format, it is a better product due to its more improved durability.

1.44 MB $3\frac{1}{2}$" Floppy

The successor of the 720 KB $3\frac{1}{2}$" format is the 1.44 MB $3\frac{1}{2}$". This format is by far the best known and most used format of all. The storage capacity is 1.44 MB of data. This is acheived by the fact that there are 80 tracks in total with an amount of 18 sectors per track.

2.88 MB $3\frac{1}{2}$" Floppy

This format was created to replace the 1.44 MB $3\frac{1}{2}$" format. It is in every way exactly the same as the 1.44 MB $3\frac{1}{2}$", except for the amount of sectors. The Sectors per Track are double that of the 1.44 MB $3\frac{1}{2}$", which explains why it has twice the storage capacity of this floppy. Unfortunately it failed in its quest. The reason for this is totally unknown for it is actually a better format than that of the 1.44 MB $3\frac{1}{2}$".

The specifications for all these disk formats are listed in Table 2.1.

Specification	360KB $5\frac{1}{4}$"	1.2MB $5\frac{1}{4}$"	720KB $3\frac{1}{2}$"	1.44MB $3\frac{1}{2}$"	2.88MB $3\frac{1}{2}$"
Amount of Heads	2	2	2	2	2
Spindle Speed	300 RPM	360 RPM	300 RPM	300 RPM	300 RPM
Tracks (Cylinders)	40	80	80	80	80
Sectors per Track	9	15	9	18	36
Sectors per Disk	720	2400	1440	2880	5760

Table 2.1: Various disk format Specifications
Source: Extracted from [Kozierok 1999]

2.2 Communication Protocol

A Multiple Access Communication technique is needed in this project to ensure the correct communication between the different units. Prasad [1996], Pahlavan & Levesque [1995] and Rappaport [1996] identify three main classification groups of Multiple Access Protocols (MAPs). These groups are listed below.

- Contentionless (scheduling)
- Contention (random access)
- Code Division Multiple Access (CDMA)

Contentionless Protocols

Also known as Scheduling Protocols, this particular protocol relies on scheduling the transmissions of the users. In this way it ensures that the different users do not access the channel at the same time.

Contention Protocols

These protocols can also be considered as being Random Access Protocols. Because any user can transmit at any time, this causes collisions. These protocols should be able to resolve any conflicts during transmission.

Code Division Multiple Access

CDMA will be discussed in more detail in Section 2.2.5. This protocol can be considered as a Contentionless Protocol where a number of users can transmit simultaneously without any collisions. When, however the users become more than the threshold amount, contention will occur, and thus it will be a Contention Protocol. For this reason the CDMA protocol cannot be classified under either Contention or Contentionless protocols.

A few of the different Communication Protocols that were studied are listed below:

- Time Division Multiple Access (TDMA)
- Frequency Division Multiple Access (FDMA)
- ALOHA

- Carrier Sense Multiple Access (CSMA)
- CDMA

Figure 2.1 illustrates the different groups in which the Protocols to be discussed can be classified.

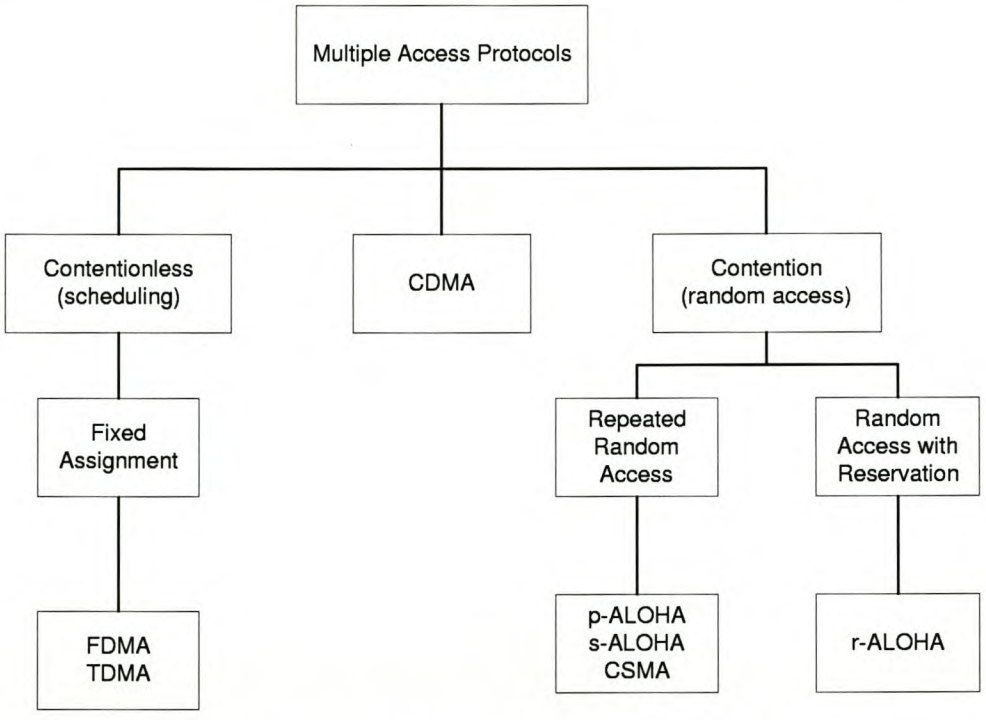

Figure 2.1: Classification of Multiple Access Protocols
Source: Extracted from [Prasad 1996, 19]

According to Figure 2.1 it is evident that, except for the three main groups, two of these can each be divided into secondary groups. These groups are:

- Fixed Assignment
- Repeated Random Access
- Random Access with Reservation

Fixed Assignment

The users are each given a fixed part of the channel capacity. This allocation is regardless whether the user has something to transmit or not. The division of channel capacity can be done in the time or frequency domain, hence the two resulting protocols: TDMA and FDMA.

Repeated Random Access

This type of protocol gives the users more freedom than Fixed Assignment. The users can access the channel randomly, whenever information is to be sent. However, this causes contention

between different users, as more than one user can try to access the channel at the same time. That is why these types of protocols are called Contention Based Protocols.

Random Access with Reservation

With this type of Protocol, the first packet is sent exactly in the same way as the Random Access Protocol. The difference is, that if no collisions occur after the first packet is sent, the user is allocated a fixed part of the channel. After the transmission of its final packet, the user returns the allocated capacity and then it can be used by a different user.

2.2.1 TDMA

In TDMA the radio transmission spectrum is divided into different time frames. Each of these frames is then divided into equally sized time slots. The time slots are then allocated to a specific user and only this user is allowed to receive or transmit during this time slot. The specific slot in every frame is the same for the user. Like FDMA, this protocol also belongs under the Fixed Assignment Contentionless Multiple Access Protocol. Figure 2.2 illustrates the concept of TDMA.

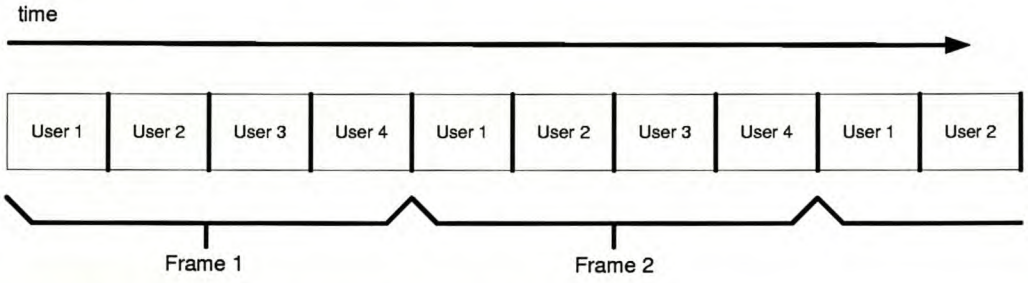

Figure 2.2: Frames and Time slot illustration for TDMA
Source: Extracted from [Prasad 1996, 21]

2.2.2 FDMA

FDMA is much the same as TDMA. In the case of FDMA it is not the time domain, as in TDMA, that is divided, but the frequency domain. The bandwidth channel is divided into a number of frequency bands. Between these frequency bands there are, what is called, a guard band, to ensure that the frequency separation of the adjacent slots are established. Each user thus has a frequency band of its own. This means that the user always has a transmission channel.

2.2.3 ALOHA

The ALOHA protocol was given this name, because it was first used at the University of Hawaii according to Halsall [1992]. It was used as an interconnection for a computer system between the different islands. This MAP was the first of its kind, and so ALOHA was born.

Pure ALOHA

As time progressed, variations on the ALOHA protocol were developed. The original ALOHA protocol was therefore renamed to the Pure ALOHA (p-ALOHA) protocol. The concept of p-ALOHA is very straightforward. A number of users can transmit to a base station whenever they have information. Because of this, it is inevitable that there will be collisions between different transmission packets. To compensate for this, each user waits for a specified amount of time for an Acknowledge (ACK) signal.

If the signal is not received the user will assume that there had been a collision. The packet will then be retransmitted after a random delay of time. This random delay prevents the system of going deadlock because of continuous collision. This can happen when users transmit at the same time, and they detect the collision at the same time. Therefore, if they both wait the same length of time, there will be another collision, etc.

p-ALOHA is classified under Repeated Random Access Protocols.

Slotted ALOHA

As p-ALOHA, Slotted ALOHA (s-ALOHA) is also classified under Repeated Random Access Protocols. It is much like p-ALOHA, with just a little improvement. The vulnerability time for collisions is decreased by a factor two in s-ALOHA. This is achieved by implementing time slots in which the users are allowed to transmit. The time slots are greater than the amount of time needed to transmit a packet of data.

All the different users are then synchronised to these time slots, so that each will only begin transmitting at the start of a time slot. If a collision occurs, the same tactic as in p-ALOHA is used. A random time delay is incorporated and then the user will retransmit the packet at the beginning of the time slot. As the users increase, the random delay time will also increase.

Reservation ALOHA

Reservation ALOHA (r-ALOHA) is classified under the Random Access with Reservation Contention Protocol. r-ALOHA use s-ALOHA to obtain the slots for reservation. The time slots are put in frames and each frame has the same amount of slots. The users can have more than one slot in a frame allocated to them. If a user does not have a time slot allocated to him/her, he/she can obtain one in the same way as in s-ALOHA. There are a number of ways to assign the reservation of the time slots.

2.2.4 CSMA

The biggest problem with ALOHA is the fact that none of the users are aware of what the others are doing. This, of course, causes collisions between the packets of the different users. This can be eliminated by implementing a protocol that can sense the channel before it transmits on

it. This means that the users know whether another user is transmitting or not. This type of protocol is known as CSMA.

Non-persistent CSMA

The user listens to the channel for any existing transmission. If it senses that the channel is idle, it will start to transmit, otherwise the user will wait for a random time and then try again. If no ACK is received by the user, it assumes a collision has occurred, and will then reschedule the packet for transmission.

1-persistent CSMA

1-persistent CSMA is a special case of p-persistent CSMA. It is the simplest form of CSMA. The terminal waits and listens to the channel. As soon as it detects that the channel is idle, the packet is transmitted. If the channel is busy, the packet is not rescheduled, the user just keeps on listening to the channel, until the channel is idle. Because of this, every user who wants to send a packet during a certain busy period, will have to wait until the channel is open and then try to send. This will cause a high probability for collision.

p-persistent CSMA

The collisions caused by the 1-persistent CSMA can be avoided by the use of the p-persistent CSMA protocol. In p-persistent CSMA the transmission of the packets are randomised by incorporating a probability of p. All the users who want to send a packet have to wait for the channel to become idle, and then transmit with a probability of p. With the probability of 1-p the user will defer the transmission for a specific time and then try transmission again with a probability of p.

CSMA/CD

With the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol, the user keeps on monitoring the channel. If any type of collision is detected, the transmission is aborted as soon as possible to save time.

2.2.5 CDMA

The multiple access property of CDMA is not realised by the division of transmission in the frequency or time domain, but by assigning a specific code to each user. These codes are used to transform the original signal into a wideband signal. Therefore, when a signal is received, the code assigned to the user is used to retransform this wideband signal into the original signal. The signal for other users remain in the wideband format, and appear as noise.

To keep the signal-to-noise ratio large enough to ensure extraction of the signal without errors, the total amount of users should be kept small. In this situation, CDMA could be regarded as a contentionless protocol.

As soon as the quantity of users increases above a certain number, the interferences become too large and the signal cannot be retrieved without errors. This basically causes the protocol to be contentionless, unless the users accessing the channel at the same time are too many.

For this reason, CDMA is placed between contention and contentionless protocols. CDMA can be divided into four groups, depending on the modulation method used to obtain the wideband signal. These groups are:

- Direct-sequence CDMA (DS-CDMA)
- Frequency hopping CDMA (FH-CDMA)
- Time hopping CDMA (TH-CDMA)
- Hybrid CDMA

Direct-sequence CDMA

The original signal is transformed to a wideband signal by modulation on a carrier and multiplication with a Binary Code Sequence (BCS). This BCS has a much larger bandwidth than the original signal and spreads the spectrum of the original signal.

This modulated signal is then transmitted. At the receiving end of the transmission, the modulated signal as well as interference (caused by the modulated signals of other users) are received.

This signal is now correlated with the BCS, and the desired signal is extracted. This is possible because the correlation between the desired signal and the BCS of the original signal will be substantial, whereas the correlation with the interfering signals will be quite small. The desired signal can then be demodulated to produce the original signal, without any errors.

Frequency hopping CDMA

In this protocol, the wideband signal is divided into frequency bands equal to the bandwidth of the original signal. During transmission, the carrier frequency is changed periodically. This change is governed by the unique code assigned to each user. At the receiving end, the demodulator only needs to follow these frequency changes for demodulation.

There can be distinguished between two types of FH-CDMA:

- Fast Frequency hopping CDMA (FFH-CDMA)
- Slow Frequency hopping CDMA (SFH-CDMA)

With FFH-CDMA the frequency change can be so rapid, that one bit is transmitted over several frequency bands. Whereas in SFH-CDMA, several bits can be transmitted in one frequency band.

In both of these cases, the probability of another user transmitting in the same frequency band, is relatively small. This is a direct result of the unique code assigned to each user. If, however, multiple transmissions happen in the same frequency band, the data can still be recovered by the use of interleaving and error correcting codes.

Time hopping CDMA

This is the same principle as in FH-CDMA. The transmission occurs in short bursts of data over the entire spectrum. Again the chances of multiple users transmitting at the same time is very small, because the start of each burst is decided by the unique code of each user. Interleaving and error correcting codes ensure easy recovery of the signal in the case of multiple users transmitting at the same time.

Hybrid CDMA

This protocol uses a combination of the modulation techniques of the previous three methods. Because of this combination, the hybrid communication uses the specific advantages that other CDMA methods offer.

2.3 Conclusion

The concepts discussed in this Chapter are the Floppy Drives available for use and their specific formats and types, as well as the Communication Protocols that was considered for the Radio Link Communication. The decision made in regards to the Communication Protocol is discussed in Chapter 3.

Chapter 3

Design Considerations

Various choices had to be made during the design and implementation of the unit. The most important of these are the choice of processor, the choice of protocol, and the reason for using a Floppy Disk Controller (FDC) instead of for instance an Integrated Drive Electronics (IDE) controller. This chapter is dedicated to the explanation of these choices and the reasons for them. Figure 3.1 is an illustration of the blockdiagram of the SAU. Also included is a short discussion on the other, smaller decisions that was made. These include the LCD, the Keypad and the various interfaces between all the different devices.

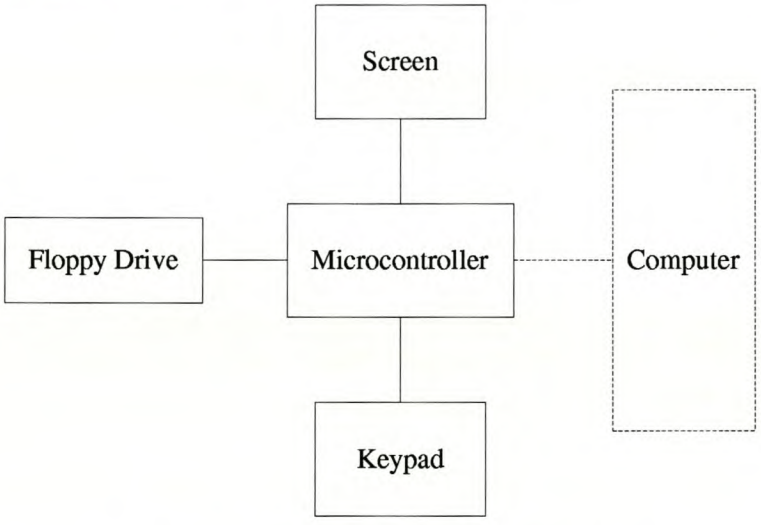

Figure 3.1: Blockdiagram of the SAU

3.1 Microprocessor

There was no need for an extremely powerful microprocessor (microcontroller), as the peripheral equipment added to the unit did not require it. The original choice for microprocessor was the AT89C52. It was, however, soon discovered that this processor did not have the required memory space for the software that had to be stored on it. The AT89C52 was also found to be too slow for the FDC. This could have been rectified by replacing the crystal with a faster one,

but, because the AT89C52 did not have enough memory, it was therefore decided to exchange this processor with a pin compatible version with more internal flash memory. The replacement that was found is the AT89C55. This microprocessor is also part of the 8031 architecture family.

The timing considerations of the AT89C55 was also taken into account. The AT89C55 is still too slow if the 11.0592 MHz crystal is used. This problem can be overcome by using a 20 MHz crystal. This, in return, gives rise to a new problem. Mismatch of the baud rate will occur between the PC and the SAU. An additional benefit of the radio links is that, when a 20 MHz crystal is incorporated into the design, the mismatch in baud rates can be ignored. At the SAU side, a 20 MHz crystal is used, and at the PC side, a 11.0592 MHz crystal. Only matching of the baud rates of the processors that communicate via the RS232 port is required.

3.2 Communication Protocol

In Chapter 2 the different Communication Protocols available for Multiple Access for a wireless system was discussed. All these protocols are very attractive, but are too complicated. The use of a Radio Packet Controller (RPC) was mentioned by a co-student, Hugo Steyn. Much thought however, was given to the use of the CDMA Protocol, but after careful consideration it was decided to rather use the RPC in the application.

After thorough investigation of the RPC it was found to be more suitable for the project. It implements its own protocol, of which the user can select a certain amount of functions. It is also compatible with the microprocessor when this is used in conjunction with a 20 MHz crystal, as was mentioned in Section 3.1.

The RPC consists of a BIM and a PIC. It was possible to use the BIM on its own, but that would mean additional attention to the shielding and all the other nitty-gritty of wireless communication. The RPC used is the Radiometrix Radio Packet Controllers and is discussed in Chapter 4 in detail.

3.3 Control of a Floppy Disk Drive

The Floppy Disk Drive (FDD) can be controlled by either an IDE controller or a FDC. In this application it was considered unnecessary to run a floppy drive from an IDE controller, as it had more functions than was needed. It was therefore decided to rather implement a FDC. The FDC is application specific for the floppy drive itself.

The next decision was which specific FDC to use. As the FDC is not freely available, the FDC used was therefore not chosen by the designer, but by default. The only FDC available, was the

Western Digital WD37C65B FDC.

The choice of FDD, is the very standard 1.44 MB $3\frac{1}{2}$" Floppy Drive, as it is freely available and compatible with most computers.

3.4 Other

As mentioned at the beginning of this chapter the other decisions made was the LCD, Keypad and the various interfaces between the devices.

3.4.1 LCD

The PC2004-A was chosen, because it could display the most amount of characters and lines possible in the price range. It has a total of four lines at 20 characters per line, which makes the display easy to read, but it was still affordable enough to keep the price of the system as low as possible. It should also be readily available in case of replacement.

3.4.2 Keypad

Here it was decided to use a 16 (4x4) Matrix Keypad. It was possible to use a smaller version of this, for instance the 12 (4x3) Matrix keypad. This was discarded, because with the 4x4 Keypad, it is possible to include more facilities for the display of the saved data. With the 4x4 Keypad it was possible to scroll through the lines or pages. This would not have been possible with the 4x3 Keypad, as it does not include the keys A, B, C and D. These are the keys of the 4x4 Keypad that were used to scroll through the data.

3.4.3 Interfaces

It was decided to include the following components in the design of the SAU as external memory to the microprocessor.

- FDC
- Keypad
- Memory

This made it easier to communicate with these devices. A CPLD was included to enable the necessary address decoding. The remaining devices that are not included in this method of interfacing, are:

- LCD
- RS232

It was unnecessary to include these two components in the interface method because, the RS232 uses the RXD and TXD ports of the AT89C55 for its serial communication. The data to the LCD has to be latched in, so it was easier to do that manually than to interface it through external memory.

Chapter 4

Radio link Communication

The Radio link communication was established by the use of the Radiometrix Radio Packet Controller. The RPC is a completely isolated unit and only needs to be connected to a Host interface. There should, however, be an antenna connected to the RPC as well as a 5 V supply. The interface to the host requires a byte wide bi-directional connection with the RPC. The transmission distance of the RPC is 30 meters in a building and 120 meters over open ground. Further discussions in this chapter will include the Fundamentals of the RPC (based on information acquired from Radiometrix [1998]) and the Interface with the host.

4.1 Fundamentals of the RPC

The RPC has a total of four different radio protocol implementations, which can be selected by the user. The choice is made by writing certain commands to the memory of the RPC. This memory contains in total, eight flags that will determine the operation of the RPC. The frequency of transmission and reception is 433MHz, which is the non-licensed commercial frequency of South Africa.

4.1.1 Functional Description

When a packet is received from the host, the RPC appends the following to the packet:

- Preamble
- Start Byte
- Error Check Code

When the RPC is not in Transmit mode it continuously listens for a valid Preamble. When detected, it will synchronise on the incoming transmission, decode it and validate the check sum. The host is then signalled that there is a valid packet waiting to be uploaded. The format of these packets are completely left to the user's choice. Except for the first byte, the Control Byte, which specifies whether the packet is data or a memory access by the host.

There are four different operating states:

- Idle/Sleep
- Host Transfer
- Transmit
- Receive

Idle/Sleep

This is the resting state of the RPC. The RPC continuously searches the radio noise for message preamble, if none is found, it goes back to "sleep". The On Time (ONT) is 5 ms and the Off Time (OFT) can be programmed in the Electrically Eraseable Programmable Read Only Memory (EEPROM) of the RPC. This can be any length of time between 22 ms and 2.9 s. The TX request line is continuously monitored during sleep mode and will be acted on immediately after detection.

Host Transfer

This is controlled by the TX and RX request lines. If the host wants to transfer data to the RPC, it sets the TX request line low. Similarly, if the RPC wants to transfer to the Host, it sets the RX request line low.

Transmit

The RPC receives a data packet from the host. Before this packet can be transmitted, the RPC appends the following to the packet:

- Preamble
- Frame Sync Byte
- Error Check Sum

After this, the packet is coded for mark space balance and then transmitted. There is also collision avoidance (Listen Before Transmit (LBT)) functions that can be enabled.

Receive

When preamble is detected from the radio receiver, the RPC will phase lock, decode and error check the incoming data and also whether it was a successful transmission. The received data is placed in a buffer and the RX Request line is activated (pulled low) to inform the Host of the available data.

4.1.2 Switches

The Switches of the RPC can be programmed at address 00H in the EEPROM. Figure 4.1 shows that there are 8 Switches. These Switches are used to control the operation of the RPC.

On power up or reset, the default values of the Switches are reloaded from address 08H in the EEPROM. The default value is 00H. The eight different Switches of the RPC will be discussed

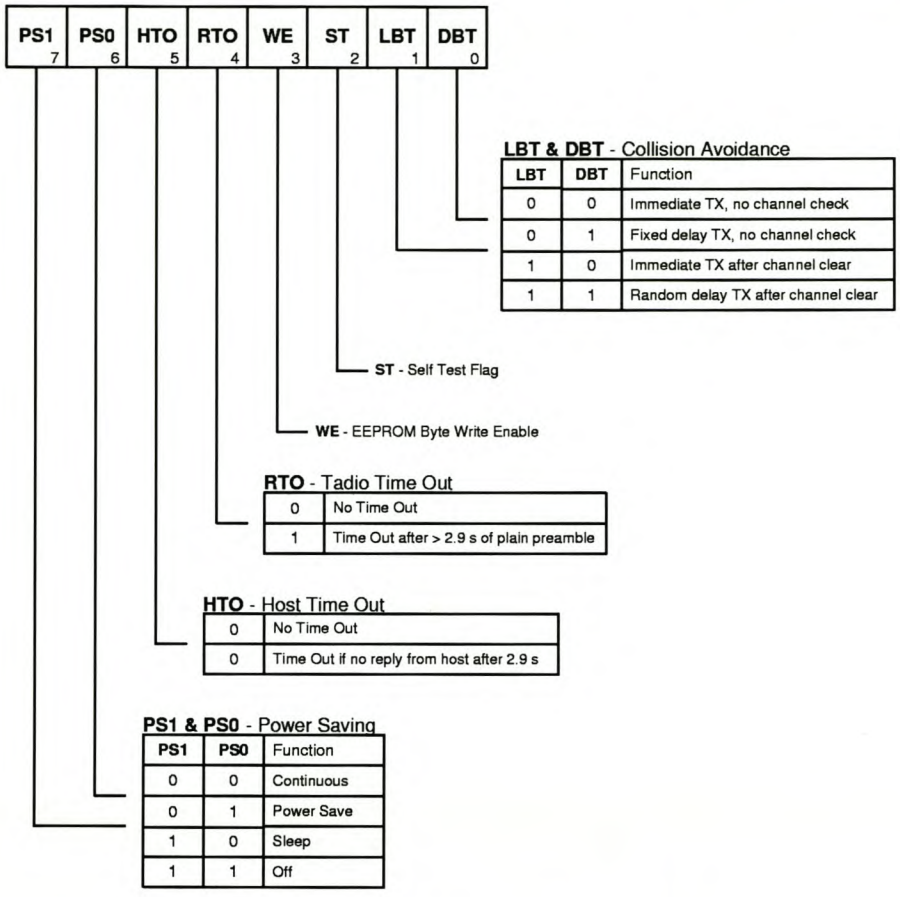

Figure 4.1: Switches found in the RPC
Source: Extracted from [Radiometrix 1998]

as follows:

- Power Saving
- Interface Time Out
- EEPROM Write Enable
- Self Test Flag
- Collision Avoidance

Power Saving

There are four different options of power saving to choose from. These different options are selected by PS0 and PS1. Table 4.1 is a representation of the choices that can be made.

Interface Time Out

The Host as well as the Radio can cause the RPC to hang until an external event is received.

PS1	PS0	Function	Description
0	0	Continuous	20mA (no power saving)
0	1	Power Save	programmable sleeptime in EEPROM address 03H
1	0	Sleep	$< 100muA$ (fixed off time of 2.9s)
1	1	Off	$< 50muA$ Transceiver is off (reset or TXR to wake-up)

Table 4.1: Codes generated by MM74C922 and the interpretation by the AT89C52
Source: Extracted from [Radiometrix 1998]

EEPROM Write Enable

This is the write protect bit of the EEPROM. No EEPROM writes are possible if this bit is not set. After every byte write, the RPC clears the bit.

Self Test Flag

If a 1 is written to this bit, the Self test is enabled. Data is fed to the TX and checked at the RX. If the test was a success, the ST bit is set.

Collision Avoidance

Listen Before Transmit (LBT) and Delay Before Transmit (DBT) determine the specific collision avoidance that the RPC will use. As was already mentioned, there are four different methods of collision avoidance. They are the following:

- Immediate TX, no channel check
- Fixed delay TX, no channel check (time slots)
- Immediate TX, if channel is clear
- Random delay TX, if channel is clear

Immediate TX, no channel check It is easy to deduce from the name that, in this case, the data will be transmitted as soon as the RPC has data to transmit. The channel will not be checked before the transmission.

Fixed delay TX, no channel check (time slots) In this case time slots are assigned to the units. The packet size, preamble length and change over delay must be the same for each unit.

Immediate TX, if channel is clear The channel is checked for any preamble or data. If nothing is found the RPC will immediately transmit its data.

Random delay TX, if channel is clear The channel is once again checked for any preamble or data. If there is nothing found, the RPC will transmit, but if the channel is busy, the RPC will have a random time delay before checking the channel again.

4.1.3 User Configurable EEPROM

The EEPROM consists of three different parts:

- RPC registers EEPROM (addresses 00H - 08H)
- Reserved EEPROM (addresses 09H - 0FH)
- User EEPROM (addresses 10H - 3FH)

Table 4.2 displays the different RPC registers in the EEPROM, the default value of the register as well as the EEPROM address where the Register is located.

EEPROM Address	RPC Registers	Default Value
00H	Control Switches	00H
01H	# Preamble Cycles	64H
02H	Wake Up Time	FFH
03H	Sleep Time	05H
04H	TX → RX Delay	1EH
05H	RX Power Up Time	1EH
06H	Tx back-off Time	03H
07H	Slot Number	01H
08H	Reset State Switches	00H

Table 4.2: RPC Registers
Source: Extracted from [Radiometrix 1998]

The only addresses that the user is allowed to use is 10H to 3FH. The other addresses are either for future use or used for the RPC Registers.

4.2 Interface to Host

The RPC was interfaced to the SAU with the use of a RS232 port. The design layout of the circuit used for the interconnection is shown in Figure 4.2.

The remainder of this section will be discussed under two subjects, namely:

4.2.1 Signals of the RPC

4.2.2 Data Transfer

4.2.1 Signals of the RPC

The reset line can be connected to the host or pulled up to Vcc. It is, however, recommended to rather connect the line to the host. From Figure 4.2 it can be seen that the reset line is connected to the host.

Except for reset, there are 8 other signals from the RPC that should be connected to the host. Table 4.3 displays the signals, their pin numbers, pin functions, I/O and their description.
It was very important to connect the TX and RX lines to the correct pins on the host. It was decided to connect the RX-request (RXR) line to one of the external interrupt lines of the

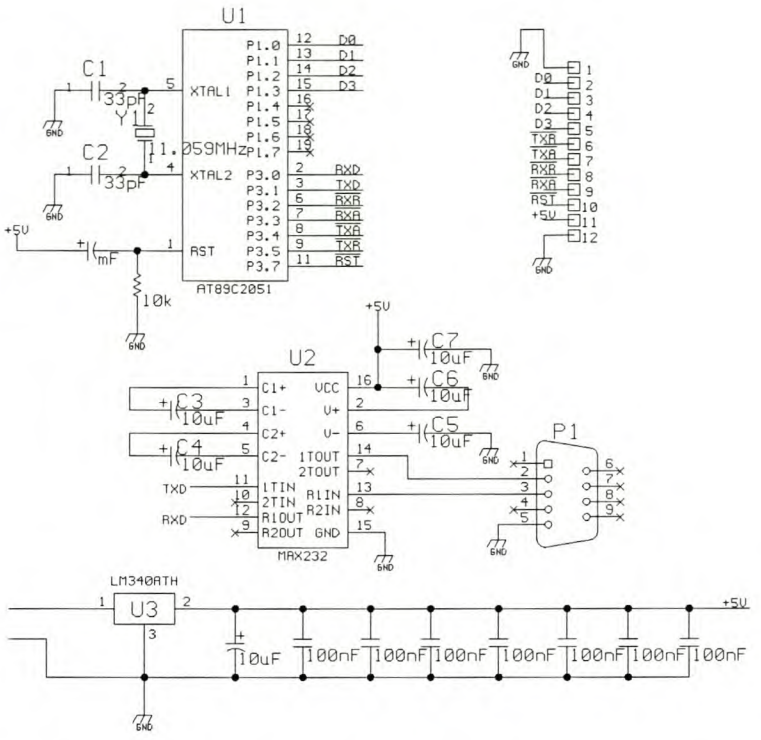

Figure 4.2: Interface between the RPC and the Host

Pin Name	Pin Number	Pin Function	I/O	Description
TXR	6	TX Request	Input	Data transfer request from Host to RPC
TXA	7	TX Accept	Output	Data accept handshake back to Host
RXR	8	RX Request	Output	Data transfer request from RPC to Host
RXA	9	RX Accept	Input	Data accept handshake back to RPC
D0	2	Data 0	Bi-directional	Data pin
D1	3	Data 1	Bi-directional	Data pin
D2	4	Data 2	Bi-directional	Data pin
D3	5	Data 3	Bi-directional	Data pin

Table 4.3: Signals for Host interface
Source: Extracted from [Radiometrix 1998]

AT89C2051. The reason for this is that whenever the RPC communicates with the host, an interrupt will occur and the host can service the RPC as fast as possible. The other possibility was to poll the lines of the RPC continuously, but that would cause the host to be unavailable for any other functions. RX-accept (RXA) was connected to P3.2 of the host.

The TX-accept (TXA) and TX-request (TXR) lines are connected to P3.4 and P3.5 respectively. There was no need to connect these to interrupt lines, as it is the lines that the host uses to send data to the RPC. After the host pulls TXR low, it waits for the RXA interrupt, and then the data can be sent. In the case of TXA, the host pulls TXA low only after the interrupt from RXR is received. This will be discussed in more detail in Subsection 4.2.2.

4.2.2 Data Transfer

Data is transferred in packets of 4 bits (nibbles) at a time. The Least Significant Nibble (LSN) is transferred first and after that the Most Significant Nibble (MSN) is transferred.

Control byte

One packet consists of 28 bytes, of which the first is the Control byte. The rest of the bytes in the packet can be anything the designer wants. Bit 7 of the control byte determines the type of packet to be transmitted. Table 4.4 shows the two possibilities. Figure 4.3 illustrates

PT	Packet Type
0	Data Packet
1	Memory Access

Table 4.4: Packet type as depicted by PT (bit 7 of the control byte)

the configuration of the Control byte for a data packet. The rest of the Control byte contains information about the remainder of the data packet. Bits 5 and 6 determines the type of preamble that will be used in the transmission of the byte, and bits 0 to 4 contains the length of the data packet, including the Control byte.

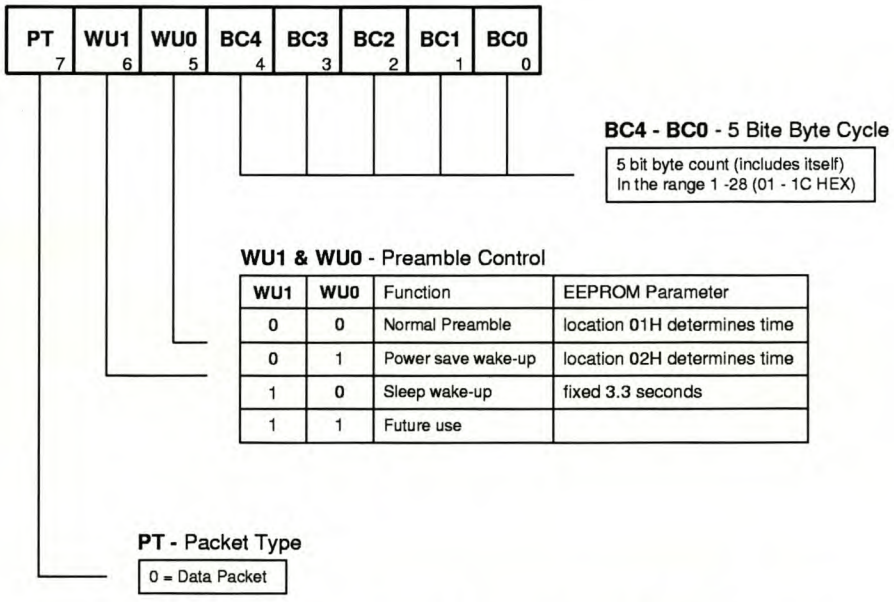

Figure 4.3: Configuration of Control byte for Data Packet
Source: Extracted from [Radiometrix 1998]

Figure 4.4 illustrates the configuration of the Control byte for memory access. In this case the rest of the Control byte carries the information needed to do a write or read to, or from the memory. Bit 6 states whether it is a read or a write, and bits 0 to 5 contains the memory address to be written to, or read from.

Memory Read With a memory read the host only transmits the control byte with the necessary settings for a memory read. The control byte contains the memory address to be read. The RPC responds with two bytes. The first is the control byte that was just sent by the host, and the next is the data contained in the specific memory address supplied.

Memory Write If a memory write is executed the host transmits two bytes. Again, first is the control byte with the necessary information for a memory write as well as the memory address. The second byte is the data that has to be written into the memory. The RPC gives no response to this command.

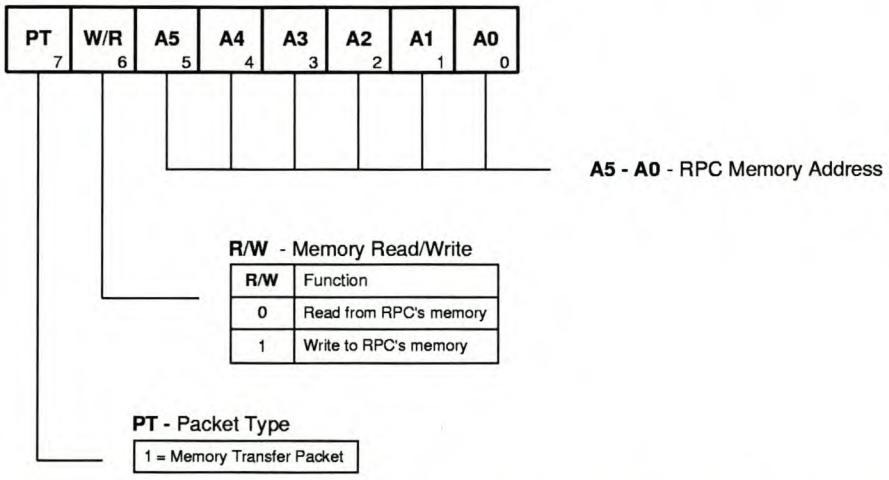

Figure 4.4: Configuration of Control byte for Memory Access
Source: Extracted from [Radiometrix 1998]

Protocol implemented

For this design it was decided to implement a fairly simple protocol for communication. It consists of a few basic signals that are sent and verified at the receiving end. The signals are as follows:

- Start of Header (SOH)
- TYPE
- Identification (ID)
- Data
- End of Transmission (ETX)

The signal has to be sent in a specific order, otherwise the unit will reject the data and it will have to be resent. The type of collision avoidance that was incorporated into the RPC communication is the random delay of TX if the channel is not clear. This is achieved by using the programmable switches in the RPC EEPROM. It will be remembered that bits 0 and 1 of address 00H in the EEPROM selects the type of collision avoidance that the RPC will use. These two bits are both set to 1 to select the specified property.

Transmit

Figure 4.5 illustrates the timing diagram of the TX download operation. From this illustration it is very evident how the RPC requires the data to be transferred. The host will start by asserting (put to 0) the TXR line. As soon as the RPC asserts TXA the host will make the data available on the data lines, and de-assert the TXR line. The LSN is transmitted first. As soon as the RPC has read all the data, it will de-assert the TXA line, indicating to the host that the data has been received. The host will then assert the TXR line again and the same process follows for the MSN.

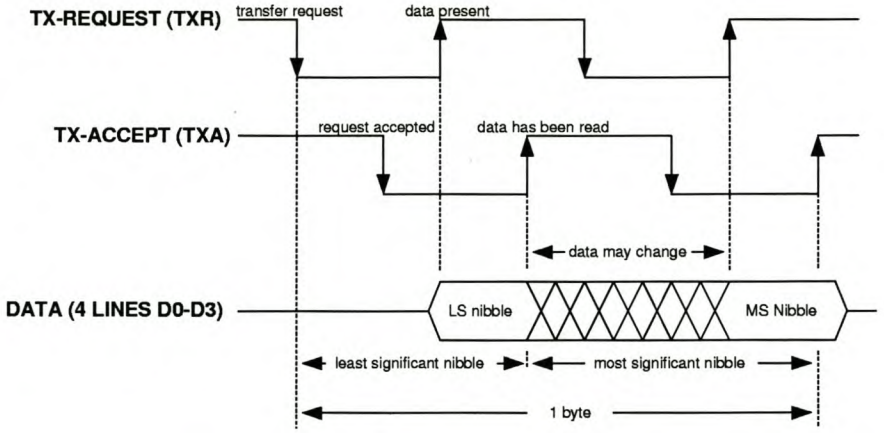

Figure 4.5: TX download timing diagram
Source: Extracted from [Radiometrix 1998]

Figure 4.6 depicts a flow diagram of the program written for the host.

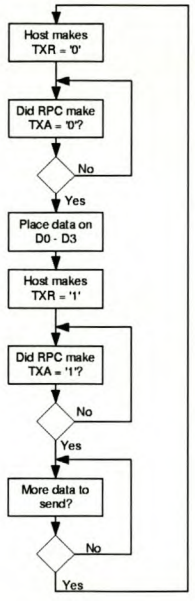

Figure 4.6: TX flowdiagram of the code implementation

Receive

Figure 4.7 shows the timing diagrams of the RX-upload operation. It is obvious from this timing diagram why the RXR line was connected to an external interrupt. The RPC asserts RXR and as soon as the host receives the interrupt, it asserts the RXA line. When the RXA line is low, the RPC places the data on the data lines, and de-asserts the RXR line. As soon as the host has read all the data it de-asserts the RXA line. This indicates to the RPC that the data was read and the process starts again.

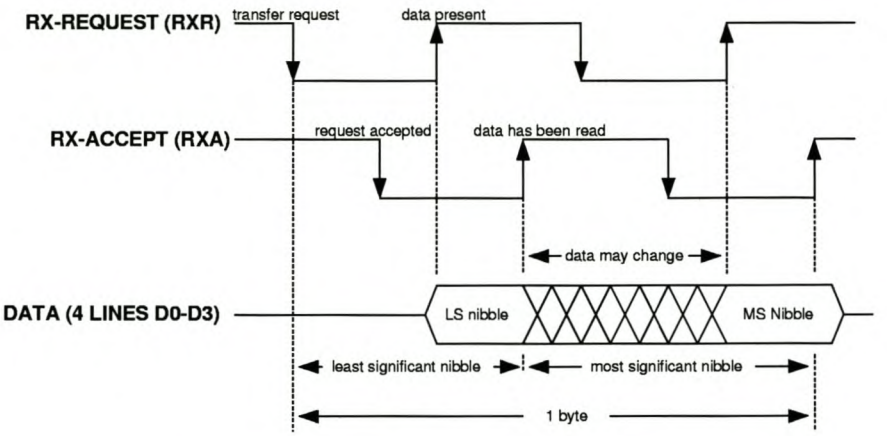

Figure 4.7: RX upload time diagram
Source: Extracted from [Radiometrix 1998]

Figure 4.8 depicts a flow diagram of the program written for the host.

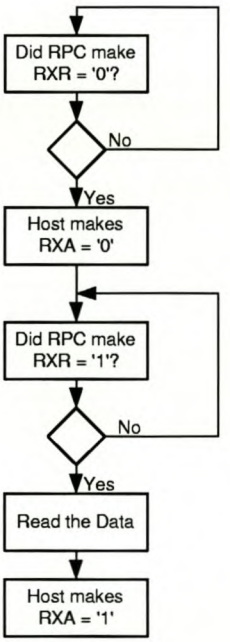

Figure 4.8: RX flowdiagram of the code implementation

4.3 Conclusion

It is evident that the interface of the RPC to the host is fairly simple. There was no need for extensive programming. The program that was written for the RPC communication can be found in Appendix A. This chapter introduced the operation of the RPC to the reader and also explained the use of the RPC in the design.

Chapter 5

Stand-alone Unit

The SAU was designed with inclusion of a number of components. Of these components, the most important are the following:

- Microprocessor
- Keypad
- LCD
- Floppy Drive
- Radio Link

The Floppy Drive is interfaced into the unit using a FDC. The unit can be updated in the future to include, instead of a FDC, an IDE controller. This would mean that the unit can be upgraded to have, for instance, a hard drive. This, of course will increase the amount of data that can be stored.

In this chapter the developement phase, the conceptual and detail design phases as well as the problems encountered during the design of the stand alone unit will be explained. The design phase of the unit will be discussed under the headings of each of the individual components.

The source code for the SAU can be found in Appendix A. There is also a Users Manual included in Appendix B, to present the reader with a logic understanding of how to operate the unit.

5.1 Developement Phase

In this section there are a few concepts in the developement of the project that will be discussed. These concepts include the size of the unit, the operation of the unit and the compatibility of the unit with the computer.

5.1.1 Size

It is optimal to keep the unit compact and small. Figure 5.1 is an illustration of what the unit looks like. This illustration is not a full sized representation of the unit. The actual size of the unit is about 20X10 cm, excluding the FDD and the RPC.

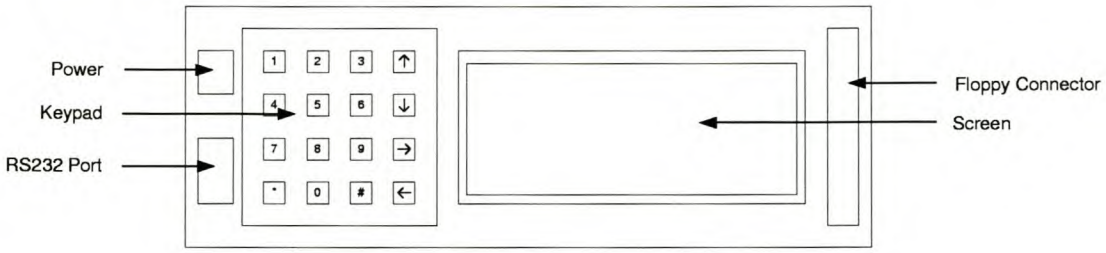

Figure 5.1: SAU Appearance

5.1.2 Operation

The operation of the unit has to be kept as simple as possible. It consists of an extremely easy to use keypad interface to the unit. From this keypad the user can choose between a couple of options to produce the desired results. The keypad will be discussed in more detail in Section 5.3. The LCD serves as the display unit for all the data and commands. This will also be discussed further in Section 5.4.

5.1.3 Compatibility

This was acquired by the use of Radio links to communicate between the PC and the SAU. The Radio links are connected via the RS232 port on the SAU. This port also enables the user to connect the SAU directly to the PC with a normal serial cable, should this be needed.

5.2 Conceptual Design Phase

A blockdiagram of the conceptual design can be seen in Figure 5.2. In this Section, the basic components of this unit, as found in Figure 5.2 will be discussed.

The short discussion on the conceptual design phase of this unit will be conducted as follows:

5.2.1 Microcontroller
5.2.2 Screen
5.2.3 Floppy Drive
5.2.4 Computer
5.2.5 Keypad

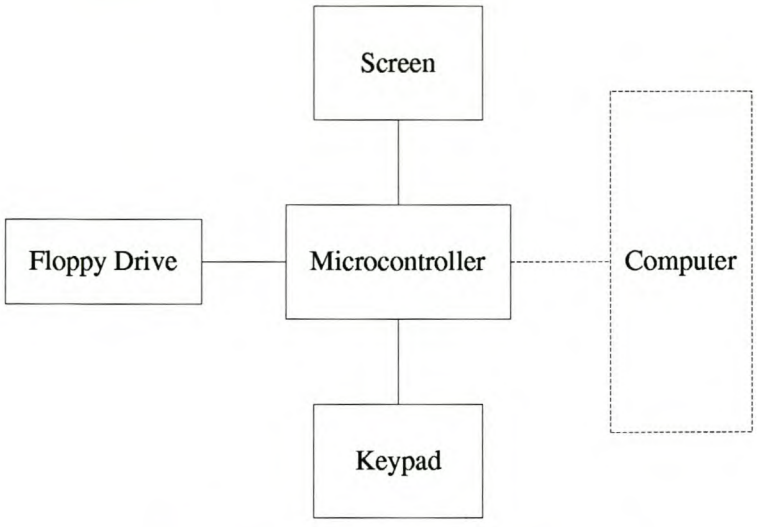

Figure 5.2: SAU Conceptual Design

5.2.1 Microcontroller

The microcontroller is responsible for all the data communication and program execution of the unit. The software that was generated for this unit will be discussed in all the relevant sections and the source code is included in Appendix A.

5.2.2 Screen

The screen will display the file read from, as well as written to the floppy. As soon as the unit is powered up the screen will display the necessary information to get it started.

5.2.3 Floppy Drive

The floppy drive is connected to the unit on a port by means of a ribbon cable. A power cable is also connected on the outside. The interface to the floppy drive is via the FDC.

5.2.4 Computer

When the user wants to write to the floppy drive, he/she has to use the interface program on the computer to type and save a file. When the correct settings are chosen, the data will be transmitted via radio link to the appropriate SAU or modem.

From this point onwards, the rest of the operation of the unit can be done without any communication with the computer.

5.2.5 Keypad

The whole concept of working is based on the use of certain codes. The status of the unit will be determined by the code entered on the keypad. When the unit is powered up, the user will first have to type in his/her personal code. After the code is verified by the SAU, the user will then have a menu of four options to choose from.

The choices are as follows:

1 Read File
2 Write File
3 Format Floppy
4 Exit

The unit has to be in the correct status. For example, when the SAU user wants to read from the floppy, the unit has to be in the Read File Status. This is the case for all the SAU's options.

Figure 5.3 is the complete interface of the SAU. This illustrates the initial layout of the design. To simplify this schematic for the reader, the component numbers (the number found at the top of each of these components in the schematic) will be listed with the component it refers to.

- U1 - AT89C52 (Microprocessor)
- U2 - HC373 (D-latch)
- U3 - 6225-8 (RAM)
- U4 - WD37C65 (Floppy Disk Subsystem Controller)
- U5 - MM74C922 (Keypad Encoder)
- U6 - EPM7032S (CPLD)
- U7 - MAX232 (RS232 interface)
- DISP1 - PC2004-A (LCD)
- P1 - Port 1 (Port for FDD)

5.3 Keypad Interface

A MM74C922 (Keypad encoder) is used to encode the signals that are received from the keypad. As soon as a button is pressed the MM74C922 encodes the signal and generates an interrupt through Data Available (DAV).

Because the AT89C55 receives an interrupt on the falling edge of the signal and DAV generates a rising edge interrupt, the DAV signal has to be inverted. This is implemented in the Complex Programmable Logic Device (CPLD), and will be discussed in Section 5.5.

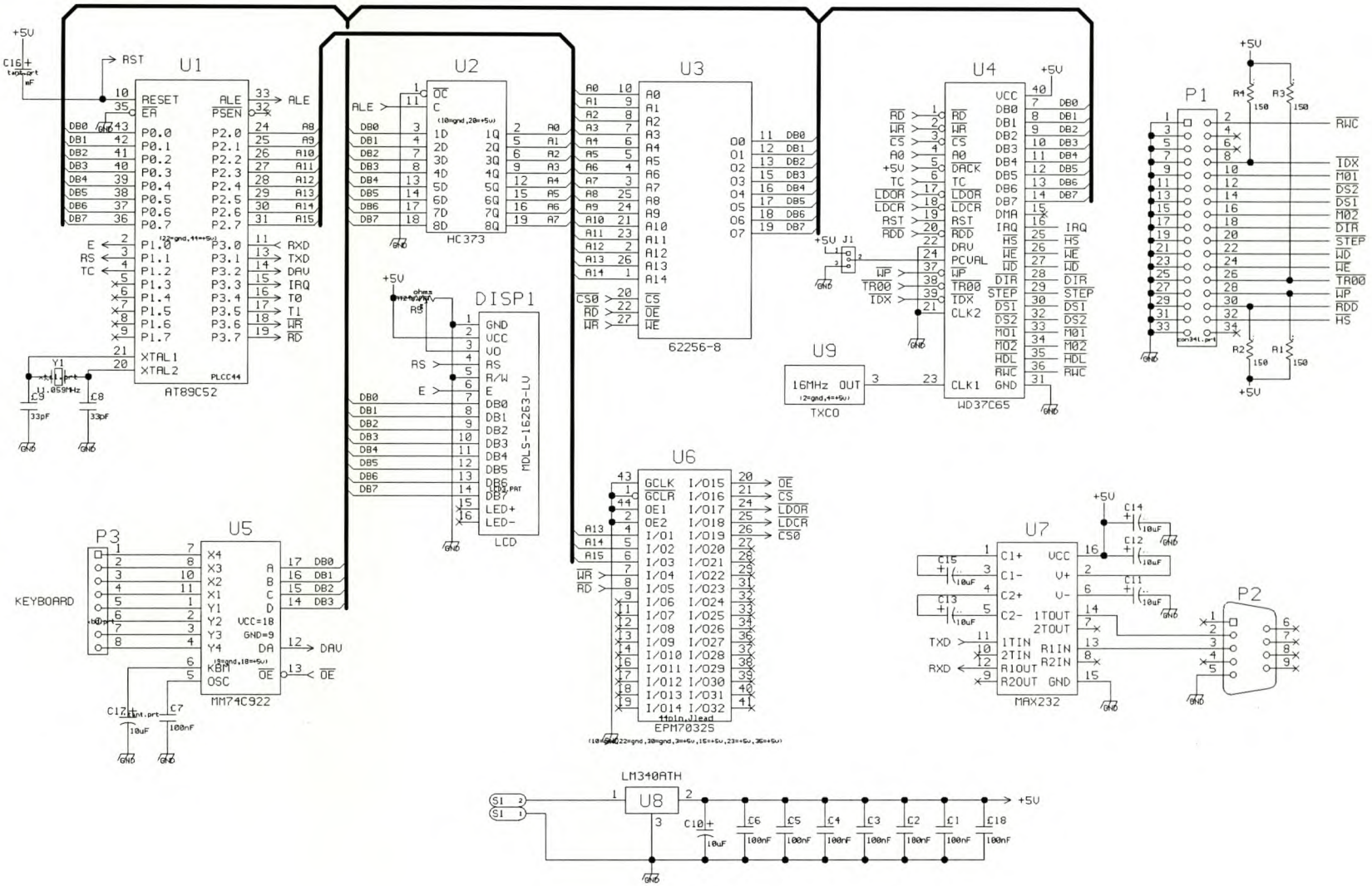

Figure 5.3: Original SAU Detail Schematic

Figure 5.4 illustrates the interface between the keypad and the MM74C922. Table 5.1 shows

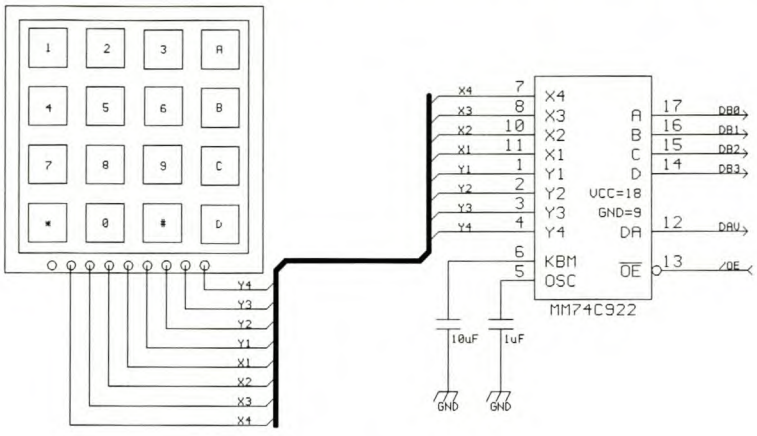

Figure 5.4: Keypad Interface

the codes generated by the MM74C922 when a button is pressed, and the decoding thereof by the AT89C55. The block diagram in Figure 5.5 is an illustration of the code that was generated

Key Pressed	D	C	B	A	Signals High at MM74C922	Output to AT89C55
1	0	0	0	0	X1 Y1	0
2	0	0	0	1	X2 Y1	1
3	0	0	1	0	X3 Y1	2
A	0	0	1	1	X4 Y1	3
4	0	1	0	0	X1 Y2	4
5	0	1	0	1	X2 Y2	5
6	0	1	1	0	X3 Y2	6
B	0	1	1	1	X4 Y2	7
7	1	0	0	0	X1 Y3	8
8	1	0	0	1	X2 Y3	9
9	1	0	1	0	X3 Y3	A
C	1	0	1	1	X4 Y3	B
*	1	1	0	0	X1 Y4	C
0	1	1	0	1	X2 Y4	D
#	1	1	1	0	X3 Y4	E
D	1	1	1	1	X4 Y4	F

Table 5.1: Codes generated by MM74C922 and the interpretation by the AT89C55

for the keypad interface.

The block diagram is implemented as a case statement in the program code for the keypad interface. The interpretation of the block diagram is discussed next. The code for the keypad interface is located in Appendix A.

When the unit is powered up, the screen is cleared and the unit goes into a waiting state. As soon as it receives a DAV interrupt, the key pressed is compared with the Begin Key, *.

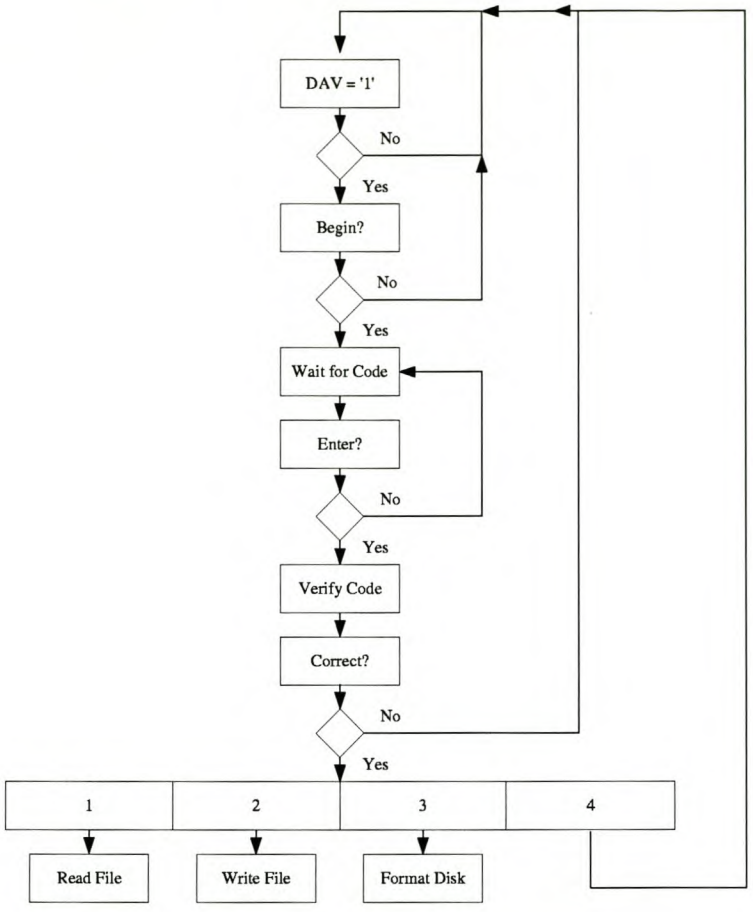

Figure 5.5: Block diagram for Keypad Interface

If the comparison between the Begin key and the key pressed is found to be true, the unit goes into the next state, the Waiting for Code state. In this state, every key that is pressed is compared with the User Identification Key. After the Enter key (#) is detected, the user is allowed login to the unit or not, depending on whether the correct code was entered.

If one or more digits of the code was entered incorrectly, the unit goes into the Logging off state and the screen is cleared again. At this point the user can try again by starting with the Begin Key.

If the code typed by the user, is verified, the unit enters the Option state. Here the user has one of four choices:

1 Read File
2 Write File
3 Format Floppy
4 Log off

Depending on the choice made by the user, the unit will either read the file from the disk, write

the file to the disk, format the disk, or log off from the system.

If the Read File choice is selected, the user can select line up/down or page left/right, or even type in a page number. At any stage during the operation of the unit, the user can Log off by pressing the Begin Key.

5.4 LCD Interface

The LCD used is the Powertip PC2004-A. This LCD has 4 lines and a total of 20 characters per line. Figure 5.6 is an illustration of how the LCD and the AT89C55 interface with each other.

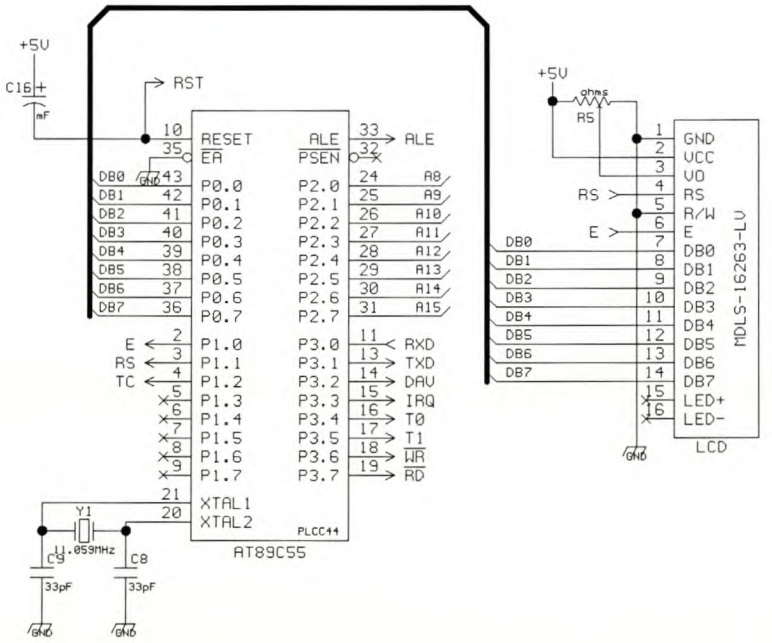

Figure 5.6: LCD interface

The Read/Write (R/W) line of the LCD is pulled to Ground (GND), because it was decided that it is unnecessary to read the status of the LCD. Instead the required Delay time was incorporated into the source code of the LCD interface. This Delay is sent before each command, to ensure correct communication with the LCD. The initialisation commands for the LCD are listed in Table 5.2.

It will be noticed that some of the commands in the table are repetitive. The reason for this is that these commands are listed in the precise order for correct initialisation of the LCD, and some of the commands have to be repeated at a later stage in the initialisation process.

The display positions of the LCD are not in a numeric sequence.

Instruction	RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Clear Display	0	0	0	0	0	0	0	0	1
Initialisation step 1	0	0	0	1	1	1	0	0	0
Initialisation step 2	0	0	0	1	1	1	0	0	0
Initialisation step 3	0	0	0	1	1	1	0	0	0
Function Set	0	0	0	0	0	0	1	1	0
Display off	0	0	0	0	0	1	0	0	0
Display on	0	0	0	0	0	0	0	0	1
Entry mode Set	0	0	0	0	0	0	1	1	0
Clear Display	0	0	0	0	0	0	0	0	1
Cursor Home	0	0	0	0	0	0	0	1	0
Cursor Off	0	0	0	0	0	1	1	0	0
Cursor Home	0	0	0	0	0	0	0	1	0

Table 5.2: Instruction set for initialisation for PC2004-A
Source: Extracted from [Powertip n.d.]

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	← Display Position
1-line	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	← DD RAM address
2-line	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53	
3-line	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	
4-line	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67	

Figure 5.7: Display position and DD RAM address for PC2004-A
Source: Extracted from [Powertip n.d.]

Figure 5.7 shows that the first line on the display is followed, numerically, by the third line. From this point some characters are skipped and are then followed by the second line. In return the second line is followed by the fourth. It was therefore necessary to equate the line position addresses very precisely.

5.5 CPLD

The CPLD is used for address decoding. Most of the devices in the SAU are interfaced as external memory devices. These are the FDC, Memory and Keypad. For this reason the three Most Significant Bits (MSBs) of the address line are used to decode between these different devices.

A15	A14	A13	Address	Device Signal
0	0	0	0x0XXX	Memory Select
0	0	1	0x2XXX	Keypad Select
1	0	0	0x8XXX	FDC Select
1	1	0	0xCXXX	FDC LDCR

Table 5.3: Address decoding for the CPLD

Because of this decoding it was possible to address the different devices as memory. The C-compiler that was used has two commands:

- peek - for external memory read
- poke - for external memory write

The VHDL-code that was generated can also be found in Appendix A.

5.6 RS232

The RS232 port is included for Serial Communication to and from the PC. It was only decided later to incorporate the Radio links, which are discussed in Chapter 4. The RS232 port is now used as the connection between the SAU and Radio link.

The port was interfaced to the system via a MAX232. Figure 5.8 illustrates the interconnection between the RS232 and the MAX232. It was necessary to connect only the RX, TX and GND lines of the RS232 port. The reason for this is that the communication is only for data and each data packet has its specific command parameters.

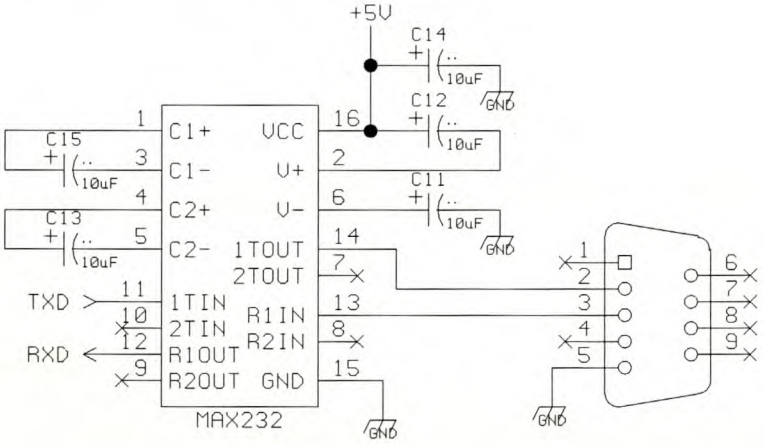

Figure 5.8: RS232 interface

5.7 Floppy Disk Controller

The FDC that is used is the WD37C65B. There are many different elements to consider when interfacing the FDC to a microprocessor. In this section the various tools used to accomplish this interface are discussed.

5.7.1 DMA and NON-DMA

With the WD37C65B it is possible to do both Direct Memory Access (DMA) as well as NON-DMA transfer. Because a AT89C55 was used, DMA transfer could not be incorporated into the design as this microcontroller does not support DMA transfers. Therefore the FDC had to be

used in NON-DMA transfer mode.

If the NON-DMA transfer mode is used, interrupts or polling can be used for data transfer. For the interrupt there is an external pin which can be connected to one of the interrupt pins of the microprocessor. The MSB of the Master Status Register (MSR), Request for Master(RQM), bit can be polled to determine whether the drive is ready or not.

5.7.2 Base, Special and PC-AT mode

There are three modes in which the FDC can be operated. Figure 5.9 is a flow diagram illustrating the relationship between the three modes.

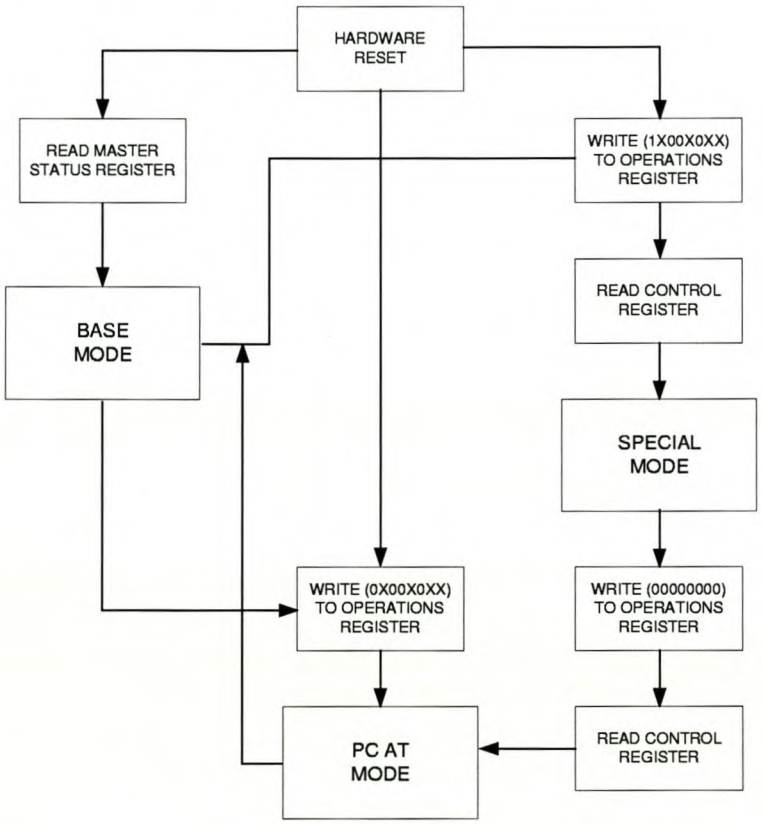

Figure 5.9: Flow Diagram illustrating the relationship between Base, Special and PC-AT mode
Source: Extracted from [Western Digital 1989]

Base Mode

It is accessed by any read or write to the FDC, but it is advised to do a read from the MSR first. In this mode the Operations Register (OR) cannot be used, but Interrupt (IRQ) and DMA are driven. For this application, only IRQ is needed. The outputs $\overline{DS1}$ to $\overline{DS4}$ are used to select between drives 1 to 4.

Special Mode

This mode allows the use of the OR. For this mode the OR is loaded with (1X00X0XX). After that a read to the Control Register (CR) is needed. In this mode $\overline{DS1}$ to $\overline{DS4}$ is again used to select the drive to use.

PC-AT Mode

For NON-DMA transfer mode this mode is not recommended, because after completion of any command, the FDC will issue an abnormal termination error status.

It is evident that the choice lies between the Base Mode and the Special Mode, because of the fact that the PC-AT Mode is not normal operation for the NON-DMA transfer mode. The mode that was chosen to work in was the Base Mode. The reason for this is simply that there was no response from the FDC when an attempt to use it in Special Mode was made.

5.7.3 Registers of the FDC

There are four primary registers and another four secondary registers.

Main Status Register

The MSR contains the status information of the FDC. This register can be read at any time during the operation of the FDC. Table 5.4 shows the information that can be read from the MSR.

BIT	NAME	SYMBOL	DESCRIPTION
DB0	FDD 0 Busy	D0B	FDD number 0 is busy
DB1	FDD 1 Busy	D1B	FDD number 1 is busy
DB2	FDD 2 Busy	D2B	FDD number 2 is busy
DB3	FDD 3 Busy	D3B	FDD number 3 is busy
DB4	FDC Busy	CB	Read or Write busy, accept no other command
DB5	Execution Mode	EXM	Set during Execution Phase
DB6	Data Input	DIO	Direction of Data Transfer
DB7	Request for Master	RQM	Data Register Ready

Table 5.4: Master Status Register
Source: Extracted from [Western Digital 1989]

The Data Input (DIO) and RQM bits can be used to poll the FDC. If these bits are in the correct state, as shown in Table 5.5, the data transfer to and from the FDC can be established.

The other four Status Registers are considered secondary registers, as they can only be read during the Result Phase of a command. The information Tables of these Status Registers can be found in Appendix E.

RQM	DIO	DESCRIPTION
0	0	No transfer
0	1	No transfer
1	0	Transfer from μP to FDC
1	1	Transfer from FDC to μP

Table 5.5: Settings of DIO and RQM
Source: Extracted from [Western Digital 1989]

Control Register

The two Least Significant Bits (LSBs) of the Data Register (DR) are latched in with $\overline{WR}$ and $\overline{LDCR}$. These two LSBs are known as CR0 and CR1. The CR is used to select the desired data rate. The data rate then determines the internal clock generation. Pin 22 of the FDC, Drive Type (DRV), also has an influence on the data rate.

Table 5.6 presents the CR options as well as the DRV influence. The data rate that is needed for the 1.44 MB $3\frac{1}{2}$" drive is 500 K.

CR1	CR0	DRV	DATA RATE
0	0	X	500 K
0	1	0	250 K
0	1	1	300 K
1	0	X	250 K
1	1	X	Not applicable

Table 5.6: Control Register Options
Source: Extracted from [Western Digital 1989]

Data Register

The DR can be selected by Address line 0 (A0). Table 5.7 demonstrates the relationship between A0, $\overline{RD}$, $\overline{WR}$, the DR and the MSR. The DR is used to load the data for the various

A0	$\overline{RD}$	$\overline{WR}$	FUNCTION
0	0	1	Read MSR
0	1	0	Illegal
0	0	0	Illegal
1	0	0	Illegal
1	0	1	Read for DR
1	1	0	Write to DR

Table 5.7: Data Register
Source: Extracted from [Western Digital 1989]

commands.

Operations Register

The OR is used to control the FDD spindle motor and to select the desired disk drive. The data of this register is latched in with $\overline{WR}$ and $\overline{LDOR}$. When the FDC is used in the Basic Mode, the OR is inaccessible, therefore this application does not use the OR.

5.7.4 Command Parameters

There are a total of fifteen commands that the WD37C65B can execute. Although all these commands are not used in the design they are all listed in Table 5.8. Each of these commands

Read Data
Read Deleted Data
Write Data
Write Deleted Data
Read a Track
Read ID
Format a Track
Scan Low or Equal
Scan High or Equal
Recalibrate
Sense Interrupt Status
Specify
Sense Drive Status
Seek

Table 5.8: The Commands of the Wd37C65B
Source: Extracted from [Western Digital 1989]

consists of three different phases:

- Command Phase
- Execution Phase
- Result Phase

Command Phase

The FDC receives all the information needed to execute a certain command from the microprocessor.

Execution Phase

The FDC executes the Command given to it during the Command Phase.

Result Phase

The FDC makes different information available to the processor.

Each of the commands used in the design is illustrated in table form in Appendix E. A short discussion on each of these commands can also be found in this Appendix.

5.7.5 Implementation

Figure 5.10 shows the interface of the FDC with the microcontroller and the floppy drive.

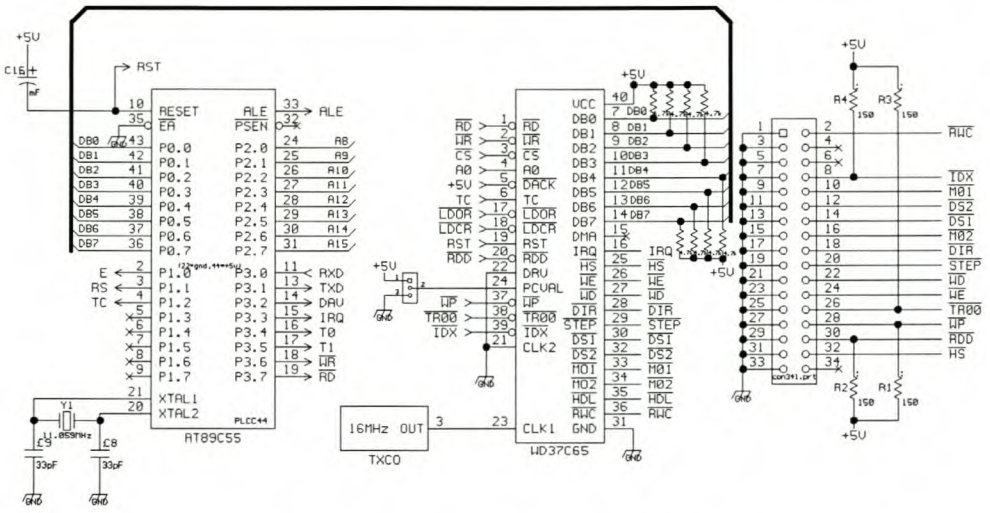

Figure 5.10: Floppy drive interface

Floppy Drive Interface

The interface between the FDC and the FDD is standard. Figure 5.11 illustrates the pin connection of the floppy drive and Table 5.9 displays all the pin names. If these pins are compared to that of the FDC, it is apparent that the interconnection between the FDC and the FDD is standard. With the exception of four pull-up resistors (at $\overline{IDX}$, $\overline{TR00}$, $\overline{WP}$ and $\overline{RDD}$), the pins are connected on the corresponding connectors of the floppy drive.

Microcontroller Interface

For the interconnection between the FDC and the Microcontroller, a little more attention was needed. The Data lines of the FDC were connected to the Data lines of the microcontroller. However, there was found to be ringing on the lines, and therefore it was needed to insert pull-up resistors. The reason for this is that TTL cannot drive CMOS signals. This is not true for the

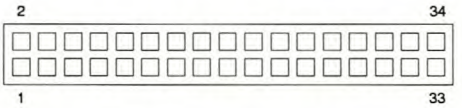

Figure 5.11: Floppy drive pin connectors
Source: Extracted from [Kozierok 1999]

PIN	NAME	DESCRIPTION
2	$\overline{REDWC}$	Density Select
4	n/c	no connection
6	n/c	no connection
8	$\overline{INDEX}$	Index
10	$\overline{MOTEA}$	Motor Enable A
12	$\overline{DRVSB}$	Drive select B
14	$\overline{DRVSA}$	Drive select A
16	$\overline{MOTEB}$	Motor Enable B
18	$\overline{DIR}$	Direction
20	$\overline{STEP}$	Step
22	$\overline{WDATE}$	Write Data
24	$\overline{WGATE}$	Floppy Write Enable
26	$\overline{TRK00}$	Track 0
28	$\overline{WPT}$	Write Protect
30	$\overline{RDATA}$	Read Data
32	$\overline{SIDE1}$	Head Select
34	n/c	no connection

Table 5.9: Internal Diskdrive
Source: Extracted from [Kozierok 1999]

opposite, though. CMOS has no problems driving TTL signals. This problem is discussed in full in Section 5.8.

There are a number of signals that are connected directly from the Microcontroller to the FDC. These signals are:

- $\overline{RD}$
- $\overline{WR}$
- A0
- TC
- RST

$\overline{RD}$ and $\overline{WR}$ are connected directly to the RD and WR lines of the microcontroller, and RST is connected to the RESET line of the microcontroller. In effect A0 is not directly connected to the Microcontroller, because of the latch that is between them. Terminal Count (TC) is connected to P1.2, because this signal can be toggled manually.

In Section 5.5 the signals that were connected via the CPLD was already mentioned. There are three of them and they are the $\overline{LDOR}$, $\overline{LDCR}$ and $\overline{CS}$ signals. The reason for this connection is that the FDC can now be addressed as external memory, as was stated in Section 5.5.

The source code written for the interface between the FDC and the microcontroller can be found in Appendix A.4. Figure 5.12 illustrates the flow diagram that was implemented as the source code for the FDC.

From Figure 5.12 it can be seen that Sense Interrupt Status is included twice. The reason

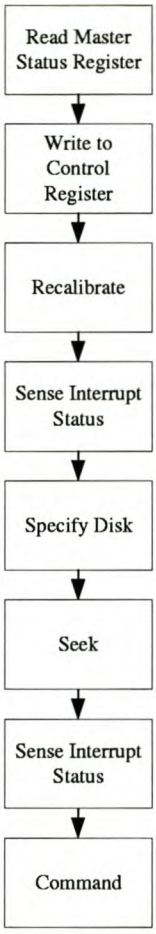

Figure 5.12: Flow Diagram illustrating the source code implemented for the FDC

for this is that the Sense Interrupt Status is used in conjunction with the Seek and Recalibrate commands, because these two commands do not have a Result phase.

The Command in Figure 5.12 can be defined as one of four possibilities:

- Format A Track
- Read Data
- Write Data
- Read ID

As was already stated all these commands are discussed in detail in Appendix E.

5.8 Problems encountered in the Design

Most of the problems that were encountered, was in the floppy drive design. Initially it was decided to communicate with the FDC in the Special Mode (see Section 5.7.2). The reason for this was the fact that there are more options available to make the operation of the FDC easier. This mode is very precise in the various settings of the FDC. The first problem thus encountered was in this mode. No response was found from the FDC. Various different settings were tried, but still no response from the FDC. This occurrence led to the dismissal of this approach.

The new approach was to try and induce a reaction by using the FDC in Base Mode. With this attempt, a response was received, but it was not the anticipated response. The data that was returned to the microprocessor, was viewed on the Logic Analyzer. Here it was seen that the MSR returned a value that was unwanted by the designer.

The next step was to study the signals with a Logic Analyser. This indicated that there was ringing on the data lines of the FDC. This was eliminated through the use of pull-up resistors. The pull-up resistors had a value of 4,7 kΩ each. The revised detail schematic of the SAU, can be found in Figure 5.13. The cause of the ringing was already described in Section 5.7.5, namely TTL signals cannot drive CMOS signals.

Why do TTL signals need pull-up resistors when driving CMOS signals? This question is easily answered when looking at the input and output characteristics of both CMOS and TTL. This information was found in Horowitz & Hill [1995]. TTL requires a +5V supply voltage, but only pulls high to about +3,5V. It has, however, good sinking, almost to ground. When driving CMOS signals, full swing from rail to rail is needed. This means, the signal input has to be +5V for a +5V supply CMOS. Therefore, it is important that the TTL be able to supply an output swing of +5V. As TTL cannot supply this, it is therefore necessary to include a pull-up resistor to +5V between TTL and CMOS.

The choice of size of this resistor is a compromise. A smaller value will ensure faster reaction, but this uses more power. The typical value of a pull-up resistor is advised by Horowitz & Hill [1995] as 4,7kΩ. This pull-up resistor will bring the TTL output swing all the way up to +5V. CMOS swings from rail to rail in it's output, so it has no problems driving TTL.

After this addition to the design it was found that the FDC selects the correct drive as stipulated by the programmer. A Recalibrate command was then sent to the FDC. This simply moves the head of the floppy drive to the start position. After this command it was needed to do a Sense interrupt Status command. The Recalibrate command instigates a interrupt after execution so the reason for the interrupt had to be investigated before continuation was possible. A Seek command was then executed to move the head of the floppy drive to a specific position on the floppy. These commands were executed successfully.

The next logical step was to try Formatting the disk. It was then discovered that the AT89C52 was too slow to give all the needed commands on time. As stated in Section 3.1, another problem was that the AT89C52 did not have enough internal flash memory for the program source code. The AT89C52 was replaced by the pin compatible but faster AT89C55. A faster crystal was then inserted into the design, but still the command was not executed successfully.

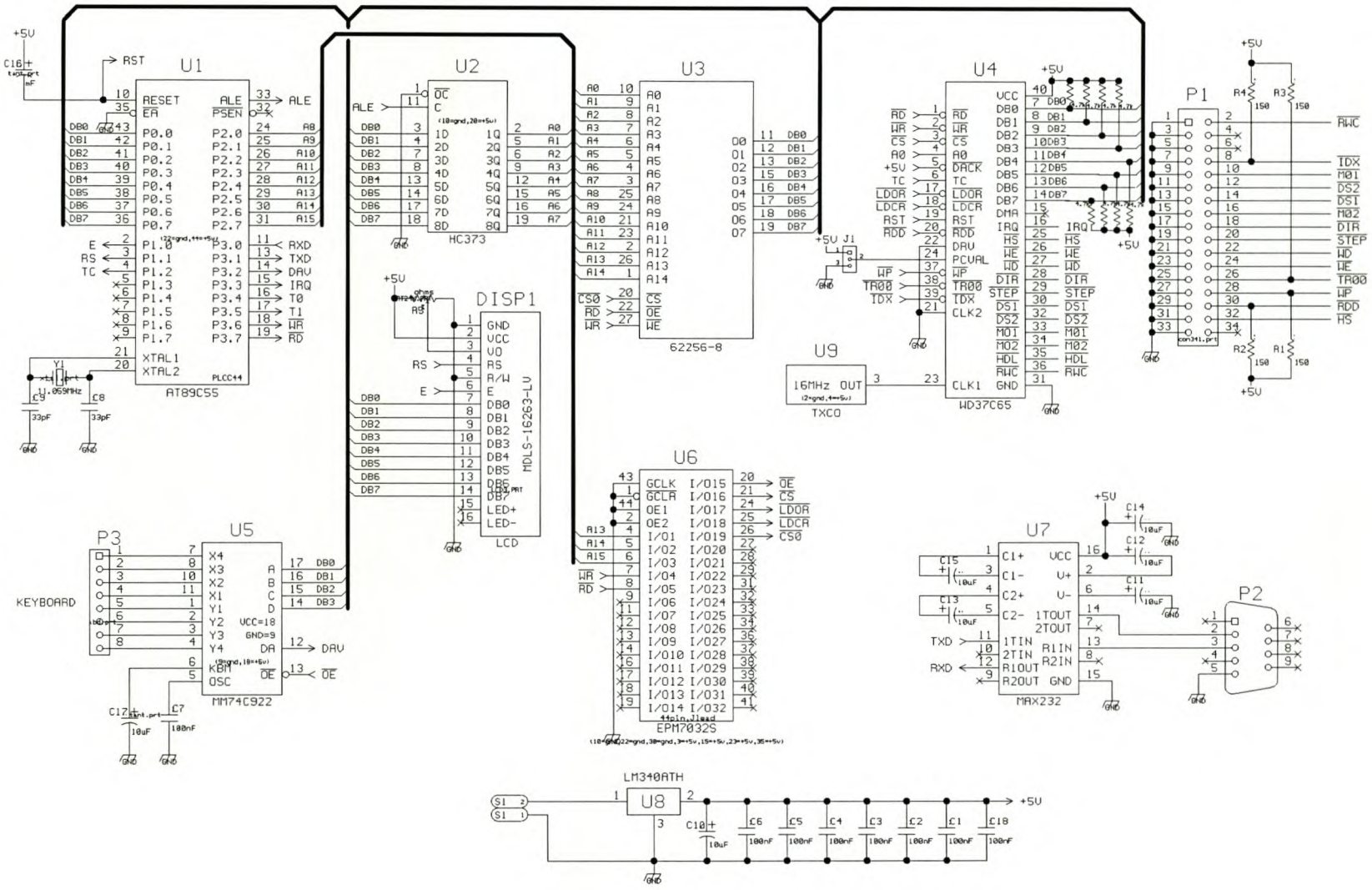

Figure 5.13: Revised SAU Detail Schematic

At this time it was decided rather to continue with the rest of the project. The floppy drive project was then set aside.

5.9 Conclusion

The floppy drive is still not in a state of full operation. Too many problems were encountered. As too much time was wasted on the floppy drive project, it was decided to abandon it, as this was not such an important part of the design and it can be completed at a later stage. There was also not much information and support on the working of the FDC as these components are out of date and information on them is not freely available. The FDC used in the design was found in an old computer.

It is advisable not to include a FDC in the system, but instead a device that is more freely available and has more support on its operation. If a FDD is a must to be included, rather use an IDE controller instead of the FDC. An example of a different choice of device to add would be a Hard Disk Drive. For this device it would be necessary to include an IDE controller. This implies that it can be possible to still include the FDD and also add a CD ROM if it was required.

Chapter 6

Windows Interface

The Windows interface was developed in Delphi 4. The basic construction is an editor, with the addition of the tools needed for transmission of data from the PC to the SAU, as well as transmission via the modem to other PC's.

The interface was designed to be user friendly. This chapter will include discussions on the developement of the interface in Delphi, as well as detailed explanations of how the Serial and Modem Communications were established. The radio links are discussed in Chapter 4. A User Manual for the windows interface is included in Appendix D. The source code for all the applications written in Delphi can be found in Appendix C.

6.1 Developement

A number of components in the program had to be designed for the implementation of this interface. It was decided to design a text editor with all the basic components, to enable the user to type a text file and save it. Except for the text editor an interface to the RS232 port of the computer also had to be designed. This interface will be discussed in Section 6.3. It was also necessary to include a database that could be edited by the users for the different user numbers. A short discussion on each of these components will be continued as follows:

6.1.1 Text Editor
6.1.2 Serial Communication
6.1.3 Database
6.1.4 About Box

6.1.1 Text Editor

The text editor was designed to be in the same category as the windows editor, Wordpad. Figure 6.1 is an illustration of the text editor at runtime. From this illustration all the Main Menu Items can be seen. These items as well as their subsections will be discussed in more detail in Section 6.2

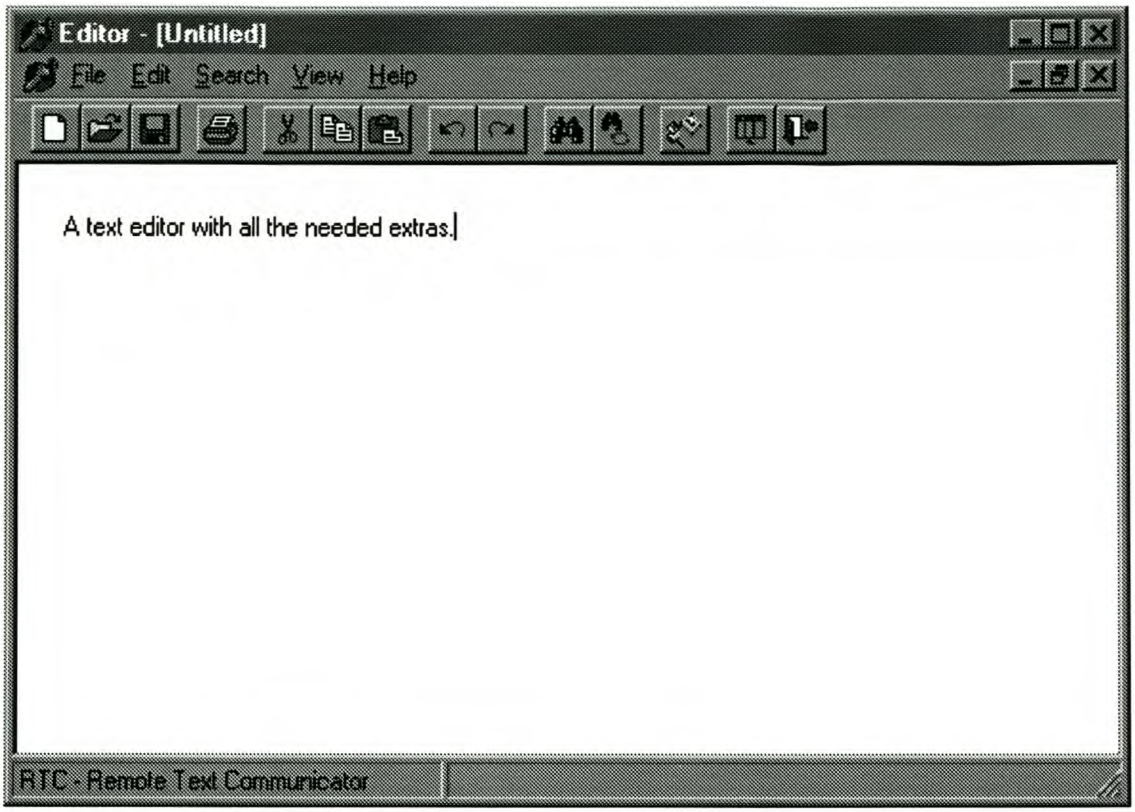

Figure 6.1: Text Editor appearance

6.1.2 Serial Communication

This is one of the most important components of the program. This is the interface between the windows application and the communication ports of the computer. According to the unit the specific communication port is chosen. This in return enables either one of the SAU's or the modem. The user chosen, indicates a specific unit, and the file is only the text file that is going to be sent. Figure 6.2 illustrates the Serial Communication form at runtime.

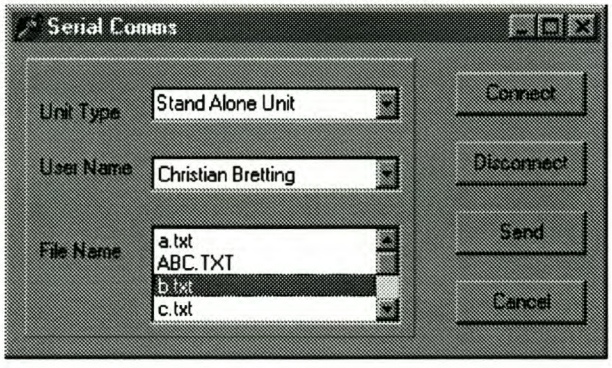

Figure 6.2: Serial Communication interface

6.1.3 Database

Figure 6.3 illustrates the interface for entering the names of the users and the unit numbers at runtime. These unit numbers are now used to communicate with a specific unit to the matching unit number. The names entered against the unitnumbers are, for example, the name of the person using that specific unit.

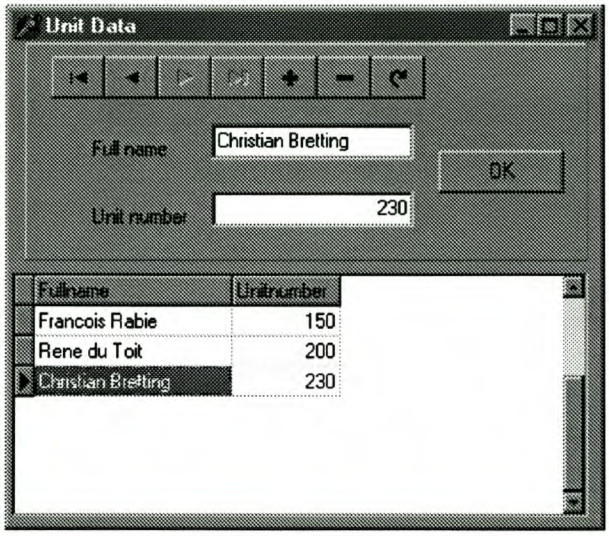

Figure 6.3: Data Units

6.1.4 About Box

Under the Help menu item there is a sub item, About, that displays some information on the program. Figure 6.4 is an illustration of the pop up box that appears when this item is selected. To exit this form, the user only has to select the OK button.

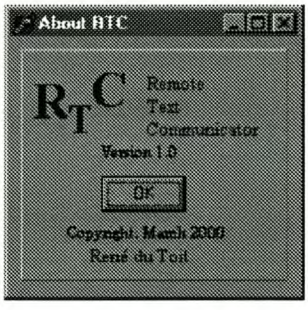

Figure 6.4: Data Units

6.2 Main Interface

The main interface is the text editor. As was already stated in Section 6.1.1, this interface was designed to correspond with the Windows text editor, Wordpad. It can be asked, why create a

text editor as well, and why not just create an application from where the file can be sent. The reason for this is simply: to make it more user friendly. It is much easier for the user to only use one program that can do everything needed for this application, instead of a different one for each specific application.

The application was designed by using Parent and Child forms. The main form is the parent, whereas the new document is the child form. Figures 6.5 and 6.7 show the two different forms at design time. The rest of this section will be discussed under the following two headings:

6.2.1 Parent Form

6.2.2 Child Form

6.2.1 Parent Form

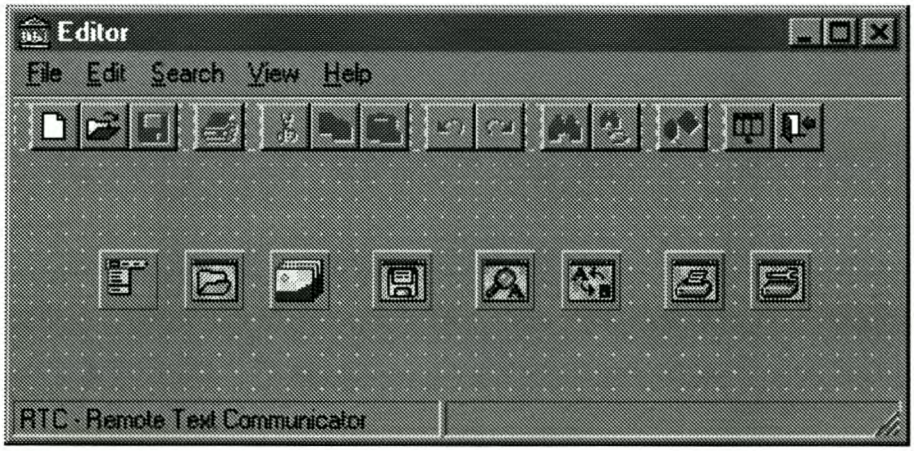

Figure 6.5: Parent form

From Figure 6.5 it can be seen that there are eight dialog boxes on this form. These dialog boxes are used for the different commands inside the program. The dialog boxes are from left to right:

- TMainMenu
- TOpenDialog
- TImageList
- TSaveDialog
- TFindDialog
- TReplaceDialog
- TPrintDialog
- TPrinterSetupDialog

TMainMenu This dialog is used to create the main menu of the form. In Figure 6.5 the Main menu of the application can be seen just above the shortcut buttons. Each of these headings in the Main menu has its own subdivisions, as seen in Figure 6.6 (a) to (e). It is noticable that

some of the selections are not visible. This is because their visible attribute was set to false. These selections will become visible as soon as the child form is opened.

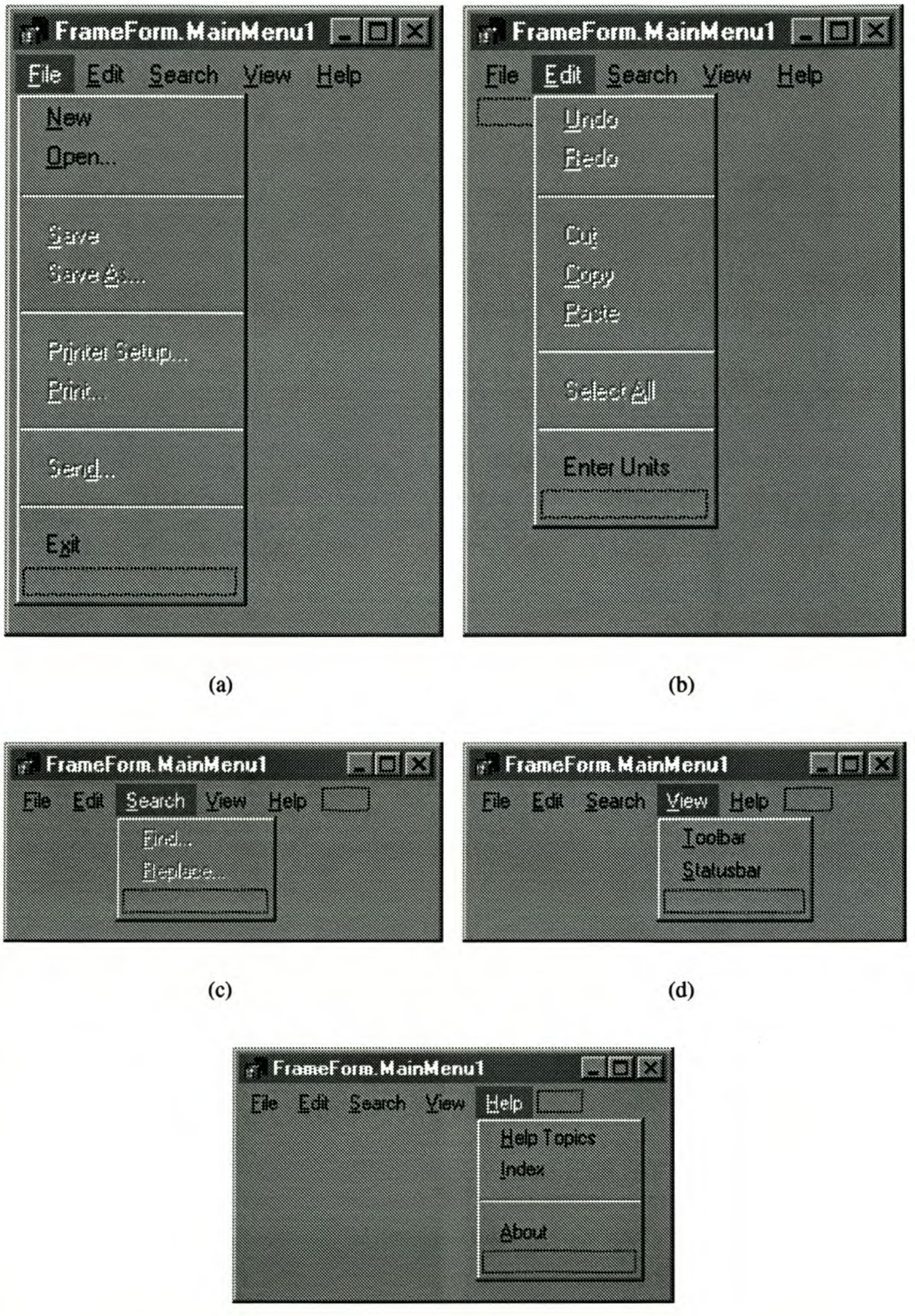

Figure 6.6: Main menu items: (a) File, (b)Edit, (a) File, (b)Edit and (b)Edit.

TOpenDialog This dialog is used to display a windows box to select and open a specific file. This dialog only appears when the Open selection on the main menu is made. As soon as the user selects the open command the dialog is closed again and the file chosen, is opened.

TImageList The TImageList Dialog box is used to handle all the icons that are used in the program. Each icon added to the Image List has a unique indexnumber. To use the specific icon, just choose the Image List containing the icon and insert the index number of the icon in the ImageIndex attribute of the specific item, for instance a button or a menu item, etc.

TSaveDialog This dialog is used to display a windows box for saving files. The dialog only appears the moment when the Save or Save as selection on the main menu is made. As soon as the user selects the save command the dialog is closed again and the file is saved.

TFindDialog This creates the Find box where the user can type in the string that he/she wants to find. This box does not appear at runtime, only when the user selects the Find option.

TReplaceDialog Very much the same as the TFindDialog, this creates the Replace box where the user can type in the string that he/she wants to replace. This box also does not appear at runtime, only when the user selects the Replace option.

TPrintDialog This enables the Print dialog as soon as the user selects the Print option. The user can select the printer with this option as well as the properties of the chosen printer. When the Print button is selected, the current document is printed and the Print box closes.

TPrinterSetupDialog This Dialog enables the setup of the various printers. It appears when the user selects the Printer Setup option and closes after the user has made all the settings and selects the OK option.

6.2.2 Child Form

From Figure 6.7 it can be seen that there is only one dialog box for this form. The dialog box is the TMainMenu dialog. The child form Main Menu merges with the parent form Main Menu as soon as the child form is opened. In this case it can be done by opening a new or existing file.

The Main Menu of the child form is identical to that of the parent form, with the exception that the items that are inactive on the parent form are active on the child form.

Figure 6.7: Child form

6.3 Serial Communication

The program utilised to create the Serial Communication is Seriecoms [van der Merwe 1998]. Serconnect is the program that was written by the author as the Serial Communication program. Seriecoms as well as Serconnect can both be found in Appendix C.

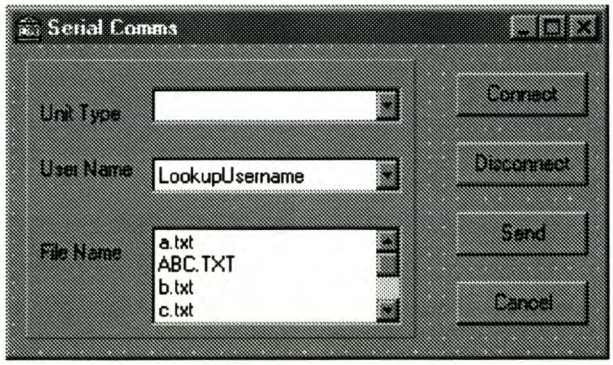

Figure 6.8: Serconnect form at design time

Figure 6.8 displays the form of Serconnect at designtime. In this illustration it can be seen that there are no Dialog boxes on this form.

The rest of the discussion in this section will be conducted as follows:

- 6.3.1 Connect, Send and Disconnect
- 6.3.2 Unit Type
- 6.3.3 User Name
- 6.3.4 File Name

6.3.1 Connect, Send and Disconnect

As was already mentioned, the Serial communication was enabled using the program Seriecoms [van der Merwe 1998]. As soon as the Connect button is selected, the program jumps to the SerConnectClick procedure. In this procedure the necessary serial communication variables are setup. There are two different setup properties. Firstly if a SAU is selected and secondly if a modem is selected as communication device. The only difference between these two devices is the communication port that is used and the baud rate. As soon as all the settings are correct, the procedure OpenS_Coms() is called and the port is opened at the correct baud rate and parity settings etc.

After the port is opened, the data can be sent. When the Send button is selected, the necessary protocol extras are appended and the file chosen is sent. The data is written to the port using the WriteS_port() procedure.

When all the data has been successfully transferred, the port is closed again on the selection of the Disconnect button. The CloseS_Coms() procedure is called when the Disconnect button is pressed.

6.3.2 Unit Type

The choice between the Modem and the SAU can be made in the first lookup box. As soon as one of the two is selected the program jumps to the Unit_typeChange. According to the unit type, a specific TYPE number is assigned. This number is then used to select the communications port that has to be opened.

6.3.3 User Name

The User Name lookup box is called a TDBLookupComboBox. This lookup box is connected to the Database created by the Data units option. It is linked to the database via a datasource. By inserting the list source option of the lookupbox as Fullname, the complete name of the different entries of the database is displayed in the box. There is also a keyfield option that has to be stipulated, which is selected as Unitnumber. This ensures that as soon as a selection of the name has been made, the Usernumber of that name is extracted from the database and allocated to the ID property. This property ensures that the data will be sent to the correct unit.

6.3.4 File Name

For this selection the lookup box is called a TFileListBox. The files included in the box are selected by the Mask option. The pathname and extension can be typed in and then only those

specific files will be included. In this application the only files that are included is text (*.txt) files.

6.4 Database

Figure 6.9 is an illustration of the Unit data form at design time. From this figure it can be seen that there are a number of unknown items thus far. These items will all be discussed in this section.

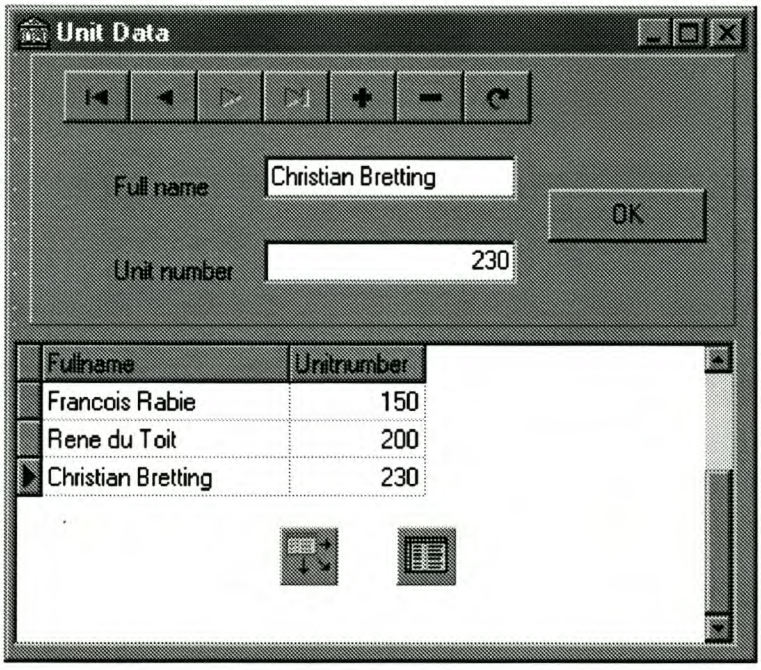

Figure 6.9: Data Units

6.4.1 TDBNavigator

In Figure 6.10 the TDBNavigator Bar can be seen. This bar is used to navigate through the data in a dataset. It is used to insert and delete entries into or out of the dataset. Again it is necessary to assign the DataSource property to a valid datasource.

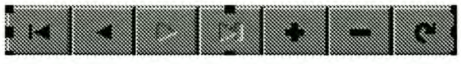

Figure 6.10: DBNavigator Bar

6.4.2 TDBEdit Boxes

These are just data base edit boxes. Each of these boxes are assigned to the DataSource, as well as the specific Datafield that it points to. These datafields determine in which column of the Table the entry will be added.

6.4.3 TDBGrid

This is used to display and manipulate the data in the dataset which makes it possible to display the data in a tabular grid. It has to be linked to a specific DataSource. As soon as the correct DataSource is selected and the Table is created, the data will be displayed in the Grid.

6.4.4 TDataSource

TDataSource provides the interface between the dataset and all the necessary components. These components include the Edit boxes, the Grid, the Navigator bar, the LookupCombo box of the Serconnect program, and the Table. The DataSource connects all these components to the dataset that is used for this application. The DataSource has to be connected to the Dataset that contains the data needed. This can be done in the DataSet field of the DataSource.

6.4.5 TTable

The TTable is used to create the Dataset. The FieldDefinitions are edited and the specific fields are added to the Table. After the settings are made, right click on the TTable dialog and select create table. This will create the dataset and everything is enabled.

6.5 Conclusion

The windows interface is the connection between the SAU and the PC as well as the PC and modem. Successful communication between the PC and the SAU was established with the use of a RS232 cable. The only problem at this stage is that the PC does not respond to data being sent from the SAU to the PC. There is not an interrupt service incorporated into the design of the Windows interface, and this is the reason why the PC will not respond to any data from the SAU.

Chapter 7

Conclusion

The system consists of various components. A PC, a SAU and an interconnection for these two. The interconnection was done with the use of a RPC. At the PC side of the design, a Windows interface was developed to enable the PC user to communicate with the SAU user.

The RPC was inserted to make the communication between the PC and the SAU wireless. This led to a reduced cost, because wiring was unnecessary. The lack of wiring also led to a neater environment. The SAU has a FDD which enables the SAU user to save the data received from the PC on a Floppy Disk.

This product is basically an educational tool. As a remote system, the SAU can be used anywhere, as no wiring is necessary. It will never replace books, but it can be a big addition to the world of literature. Especially in places with underprivileged children, where books are sometimes not readily available for the hungry minds of these young scholars.

The SAU is very user friendly and easy to use. This implies that it can also be used by fairly young children. This will introduce the children to technology at an early age. It can also be expanded to become a more advanced learning tool.

A better choice of interface used with the storage device, would have been an IDE controller. It is essential that the device that is to be used should be freely available, because the availability of sufficient data on this component is of utmost importance. If the choice had to be made again, an IDE controller would have been used instead of a FDC.

It would be wise not to spend too much time on a specific problem. Too much time was spent on the Floppy Drive, which led to less attention been paid to other details that might have been more important.

The fact that the PC would not respond to any attempt for communication from the SAU, can be

considered a shortcoming in the design. This problem occurred due to the fact that the Delphi Program could not respond to an interrupt received on the communication ports. It can be fixed by incorporating a service routine, that runs in the background of the application, constantly checking the communication ports for any sign of information.

All these problems encountered, led to a broadening of experience. Sometimes it is better to abandon a specific problem, than to waste too much time on it. Rather spend more time on matters that are more critical.

A big variety of components had to be designed for this project and it proved to be quite challenging to the designer. New information was acquired, new approaches to problems was necessary, and the project in itself was a lot of fun.

It is necessary to communicate with co-students and lecturers. More knowledge is gained from your own and other people's experience than anywhere else.

Expansion of the system is possible by making it more user friendly and more advanced. The keypad can be replaced by a keyboard to enable the SAU user to type and save data on the SAU itself. If the FDC is replaced by an IDE controller, a Hard Drive can be added to the system. This will mean more storage space for data. The SAU can be made Dos compatable, this way the SAU user can also work at home, or any other place where access to a PC is possible. The data can then just be transferred from the Floppy to the SAU.

The SAUs can also be enabled to communicate with each other, by making use of a more complex Communication Protocol which would have to be incorporated in the design.

In conclusion, the concept is very useful as an educational tool. The inclusion of the Floppy Drive makes the system more compact and creates the possibility of transferring data by use of the Floppy, which in turn does not limit a person to working only in one place.

Appendix A

Source Code for SAU

A.1 Main Program

```
/****************** Include Files ******************/
#include <8051io.h>
#include <8051reg.h>
#include <8051int.h>
#include <8051bit.h>
#include "d:\tesis\progra~1\lcd.c"
#include "d:\tesis\progra~1\floppy.c"
#include "d:\tesis\progra~1\serie.c"
#include "d:\tesis\progra~1\keypad.c"

main(void)
{
  Key_code[0] = Key_0;
  Key_code[1] = Key_0;
  Key_code[2] = Key_0;
  Key_code[3] = Key_0;
  Eerste_keer = 1;
  Key_code_buf_pos = 0;
  New_serie_data_flag = 0;
  Flopflag = 0;
  Floppy_interrupt_flag = 0;
  New_button_flag = 0;
  Ser_een = 1;
  Data_there = 0;
  Data_tel = 1;
  ID_correct = 0;
  TYPE_correct = 0;
  SOH_received = 0;
  NCN = 0;
  PCN = 0;
  request = 0;
  R = 0;
  C = 0;
  H = 0;
  R = 0;
  N = 0;
  inttel = 0;

/*  InitTimer0();*/
  Serial_Init();
  InitDisplay();
  write_data(0x35);
  InitFDC();

  while(1)
  {
    if (New_serie_data_flag == 1)
    {
      Receive_seriedata();
      New_serie_data_flag = 0;
    }

    if (New_button_flag == 1)
    {
      New_button_flag = 0;
      Keypad_state();
    }
  }
}
```

A.2 Keypad Program

```
/*
 * Main program for the keypad
 */

/* DAV - P3.2
 * A   - P1.0
 * B   - P1.1
 * C   - P1.2
 * D   - P1.3
 */

/****************** Include Files ******************/
#include "d:\tesis\progra~1\keypad.h"

/****************** Global Variables ***************/
int Key_code_buf_pos, Key_page_num_pos;

byte Key_code[4];
byte Key_code_buf[10];
byte Key_page_num[5];

byte DAV;
byte Previous_button;
byte Button;
byte New_button;
byte New_button_flag;
byte Current_keypad_state;
byte Eerste_keer;

byte Line_memory_buffer[20];


INTERRUPT (_IE0_) Button_is_pressed()
{
  New_button = peek(0x2000);
  New_button_flag = 1;
}

Keypad_state(void)
{
  Previous_button = Button;
  Button = New_button | 0xF0;
  if (Eerste_keer)
  {
    if (Button == Key_begin)
    {
      Eerste_keer = 0;
    }
    else
      Current_keypad_state = Keypad_idle;
  }
  if (Button == Key_begin)
  {
    Current_keypad_state = Keypad_wait_for_code;
    Disp_clear();
    blahfl = wait_code_phrase;
    First_line();
  }

  switch(Current_keypad_state)
  {
    case Keypad_idle:
      break;

    case Keypad_wait_for_code:
      if (Button != Key_begin)
      {
        if (Button == Key_enter)
        {
        /*  Disp_clear();
          blahfl = code_finish;
```

```
        First_line ();*/
        for ( Key_code_buf_pos = 0; Key_code_buf_pos < 4 ; Key_code_buf_pos++)
        {
        /*   Disp_clear ();
          blahfl = code_compare;
          First_line ();*/
          if ( Key_code_buf[ Key_code_buf_pos ] != Key_code[ Key_code_buf_pos ])
          {
            Disp_clear ();
            blahfl = code_incorrect;
            First_line ();
          /*   write_command( sl );
            write_data ( Key_code_buf[ i ]);
            write_data ( Key_code[ j ]);
            write_data ( i + 0x30);
            write_data ( j + 0x30);
            write_command( tl );
            write_data ( Key_code_buf[0] - 0xCE);
            write_data ( Key_code_buf[1] - 0xCE);
            write_data ( Key_code_buf[2] - 0xCE);
            write_data ( Key_code_buf[3] - 0xCE);
            write_command( fol );
            write_data ( Key_code[0] - 0xCE);
            write_data ( Key_code[1] - 0xCE);
            write_data ( Key_code[2] - 0xCE);
            write_data ( Key_code[3] - 0xCE);*/
            Current_keypad_state = Keypad_exit;
            New_button_flag = 1;
            Key_code_buf_pos = 0;
            break;
          }
          else
          {
            Disp_clear ();
            blahfl = code_correct;
            First_line ();
            Disp_clear ();
            blahfl = View_file_choice;
            blahsl = Write_file_choice;
            blahtl = Format_disk_choice;
            blahol = Exit_choice;
            First_line ();
            Second_line ();
            Third_line ();
            Fourth_line ();
            Current_keypad_state = Keypad_main_menu;
            New_button_flag = 1;
            break;
          }
        }
      }
      else
      {
        if ( Previous_button == Key_begin)
        {
          Key_code_buf_pos = 1;
          Disp_clear ();
          Key_code_buf[0] = Button;
          write_data (0x2A);
        }
        else
        {
          /* write_command(0x14 );*/
          Key_code_buf[ Key_code_buf_pos ] = Button;
          /* write_data ( Key_code_buf_pos + 0x30);
          write_data ( Key_code_buf[ Key_code_buf_pos ] - 0xCE);*/
          write_data (0x2A);
          Key_code_buf_pos++;
        }
      }
    }
    break;
```

```
case Keypad_main_menu:
  switch (Button)
  {
    /* case Key_1:
      Current_keypad_state = Keypad_new_code;
      Disp_clear();
      blahfl = code_new;
      First_line();
      Key_code_buf_pos = 0;
      break;*/
    case Key_1:
      Current_keypad_state = Keypad_view_file;
      New_button_flag = 1;
      break;
    case Key_2:
      Current_keypad_state = Keypad_write_file;
      New_button_flag = 1;
      break;
    case Key_3:
      Current_keypad_state = Keypad_format_disk;
      New_button_flag = 1;
      break;
    case Key_4:
      Current_keypad_state = Keypad_exit;
      New_button_flag = 1;
      break;
    default:
      break;
  };
  break;

/* case Keypad_new_code:
  if (Button == Key_enter)
  {
    for (Key_code_buf_pos = 0; Key_code_buf_pos < 4; Key_code_buf_pos++)
      Key_code[Key_code_buf_pos] = Key_code_buf[Key_code_buf_pos];
    Current_keypad_state = Keypad_main_menu;
  }
  else
  {
    Key_code_buf[Key_code_buf_pos] = Button;
    write_data(0x2A);
    Key_code_buf_pos++;
  }
  break;*/

case Keypad_view_file:
  /* switch(Button)
  {
    case Key_line_up:
      Scroll_line_up();
      break;
    case Key_line_down:
      Scroll_line_down();
      break;
    case Key_page_up:
      Scroll_page_up();
      break;
    case Key_page_down:
      Scroll_page_down();
      break;
    case 'Key_1, Key_2, Key_3, Key_4, Key_5, Key_6, Key_7, Key_8, Key_9, Key_0, Key_enter':
      if (Button == Key_enter)
        Goto_page();
      else
        Key_page_num[Key_page_num_pos++] = Button;
      break;
    default:
      break;
  }*/
  Read_file();
  break;
```

```
    case Keypad_write_file :
      Current_keypad_state = Keypad_main_menu;
    /*  if (Data_there == 1)
      {*/
        Data_there = 0;
        Data_tel = 0;
        Disp_clear ();
        Write_file ();
      /*  blahfl = Disk_data;
        First_line ();
        write_command(fl);
        for (Data_tel=0; Data_tel<20; Data_tel++)
        {
          setbit(P1.1);
          P0 = Disk_data[Data_tel];
          setbit(P1.0);
          clrbit(P1.0);
        }
      }
      else
      {
        Disp_clear ();
        blahfl = Nothing;
        First_line ();
      }
      Data_tel = 0;*/
      break;

    case Keypad_format_disk :
      Disp_clear ();
      blahfl = disk_format;
      First_line ();
      /*for (NCN = 0; NCN <= 80; NCN++)
      {*/
        inttel = 0;
        Recalibrate_disk ();
        Sense_interrupt_status ();
        Format_disk ();
/*      }*/
      Current_keypad_state = Keypad_main_menu;
      break;

    case Keypad_exit :
      Disp_clear ();
      Sense_drive ();

    /*  Disp_clear ();
      blahfl = exit_menu;
      First_line ();
      Current_keypad_state = Keypad_idle;*/
      New_button_flag = 1;
      break;

    default :
      Current_keypad_state = Keypad_exit;
      break;
  }
}

Scroll_line_up (void)
{
  int i;
  for (i = 0; i < 21; i++)
  {
    Line_memory_buffer[i] = Line1_LCD_buffer[i];
    Line1_LCD_buffer[i] = Line2_LCD_buffer[i];
    Line2_LCD_buffer[i] = Line3_LCD_buffer[i];
    Line3_LCD_buffer[i] = Line4_LCD_buffer[i];
/*    Line4_LCD_buffer[i] = Disk_data[i];*/
  };
}
```

```
Scroll_line_down (void)
{
  int i;
  for (i = 0; i < 21; i++)
  {
    Line2_LCD_buffer[i] = Line1_LCD_buffer[i];
    Line3_LCD_buffer[i] = Line2_LCD_buffer[i];
    Line4_LCD_buffer[i] = Line3_LCD_buffer[i];
    Line1_LCD_buffer[i] = Line_memory_buffer[i];
  };
}

Scroll_page_up (void)
{
  int i;
  for (i = 0; i < 21; i++)
  {
/*    Line1_LCD_buffer[i] = Disk_data[i];
    Line2_LCD_buffer[i] = Disk_data[i + 21];
    Line3_LCD_buffer[i] = Disk_data[i + 42];
    Line4_LCD_buffer[i] = Disk_data[i + 63];
*/  };
}

Scroll_page_down (void)
{
  int i;
  for (i = 0; i < 21; i++)
  {
/*    Line1_LCD_buffer[i] = Disk_data[i - 63];
    Line2_LCD_buffer[i] = Disk_data[i - 42];
    Line3_LCD_buffer[i] = Disk_data[i - 21];
    Line4_LCD_buffer[i] = Disk_data[i];
*/  };
}

Goto_page (void)
{
}

/* Header program for Keypad interface.
 */
#define byte     unsigned char

#define Keypad_idle       0
#define Keypad_begin      1
#define Keypad_wait_for_code   2
#define Keypad_main_menu      3
/*#define Keypad_new_code     4*/
#define Keypad_view_file      4
#define Keypad_write_file     5
#define Keypad_format_disk    6
#define Keypad_exit       7
#define Keypad_page_number    8

#define Key_1       0xF3
#define Key_2       0xF2
#define Key_3       0xF1
#define Key_4       0xF7
#define Key_5       0xF6
#define Key_6       0xF5
#define Key_7       0xFB
#define Key_8       0xFA
#define Key_9       0xF9
#define Key_0       0xFE
#define Key_begin     0xFF
#define Key_enter     0xFD
#define Key_line_up     0xF0
#define Key_line_down   0xF4
#define Key_page_up     0xF8
#define Key_page_down   0xFC
```

A.3 LCD Program

```
/*
 * Main program for the LCD module for integration on the floppy disk drive interface.
 */

/* DB0 - P0.0
 * DB1 - P0.1
 * DB2 - P0.2
 * DB3 - P0.3
 * DB4 - P0.4
 * DB5 - P0.5
 * DB6 - P0.6
 * DB7 - P0.7
 * E   - IO15 (CPLD)
 * R/W - IO32 (CPLD)
 * RS  - IO31 (CPLD)
 */


/****************** Include Files ******************/
#include "d:\tesis\progra~1\lcd.h"

#define s1   0x38  /*0x38*/
#define s2   0x38  /*0x38*/
#define s3   0x38  /*0x38*/
#define fs   0x06  /*0x3C*/
#define dof  0x08  /*0x08*/
#define don  0x01  /*0x0F*/
#define ems  0x06  /*0x06*/
#define dc   0x01  /*0x01*/
#define ch   0x02  /*0x02*/
#define doc  0x0F
#define cds  0x14
#define co   0x0C  /*0x0C*/
#define fl   0x80  /*0x80*/
#define sl   0xC0  /*0xC0*/
#define tl   0x94  /*0x94*/
#define fol  0xD4  /*0xD4*/

/****************** Global Variables ****************/
byte Line1_LCD_buffer[20];
byte Line2_LCD_buffer[20];
byte Line3_LCD_buffer[20];
byte Line4_LCD_buffer[20];
byte *blahfl;
byte *blahsl;
byte *blahtl;
byte *blahol;
byte *codech;
byte *viewfl;
byte *writef;
byte *diskfo;


byte  View_file_choice[]   = "1 View File";
byte  Write_file_choice[]  = "2 Write File";
byte  Format_disk_choice[] = "3 Format Disk";
byte  Exit_choice[] = "4 Exit";

byte wait_code_phrase[] = "Waiting for code...";
byte code_finish[] = "Code entered";
byte code_correct[] = "Code correct";
byte code_incorrect[] = "Code incorrect";
byte code_compare[] = "Compare code";
byte exit_menu[] = "Exitting...";
byte file_write[] = "Writing file...";
byte disk_format[] = "Formatting Disk...";
/* byte code_new[] = "Type new code: ";*/
byte disk_init[] = "Init floppy";
byte LCD_init[] = "Init LCD";

byte Line_change[] = "Ready line changed";
```

```
byte Normal_term [] = "Normal_termination";
byte Abnormal_term [] = "Abnormal_termination";
byte Default_term [] = "Default_case";
byte Nothing [] = "Nothing";
byte Data_blah [] = "data_daar";

/* byte Line1_LCD_buffer [] = "Hello Rene";
byte Line2_LCD_buffer [] = "Hello Ian";
byte Line3_LCD_buffer [] = "Hello Andreas";
byte Line4_LCD_buffer [] = "Hello Xandri";
*/


byte crap;


/****************** Initialise Display *************/
InitDisplay (void)
 {
  Disp_clear ();
  Stap_1 ();
  Stap_2 ();
  Stap_3 ();
  Function_set ();
  Disp_off ();
  Disp_on ();
  Entry_mode_set ();
  Disp_clear ();
  Cursor_home ();
  Cursor_off ();
  Cursor_home ();
}

InitTimer0 (void)
{
/*  TMOD = 0x01;
  IE = 0x82;
  PCON = 0x00;
  IE = 0x93;
  IP = 0x01;
  TMOD = 0x21;
  TH1 = -3;
  SCON = 0x50;
  setbit (TCON.6);
  setbit (TCON.0);
*/
}

Delay (void)
{
  TL0 = 0x00;
  TH0 = 0x255;
  setbit (TCON.4);

  while (!(TCON & 0x20));

  clrbit (TCON.4);
  clrbit (TCON.5);
}

write_command (char D)
{
  clrbit (P1.1);
  P0 = D;
  setbit (P1.0);
  clrbit (P1.0);
}

write_data (byte D)
{
/*  Delay ();*/
  setbit (P1.1);
  P0 = D;
```

```
    setbit (P1.0);
    clrbit (P1.0);
}

/****************** Step 1 in Initialisation *******/
Stap_1 (void)
{
    Delay ();
    write_command (s1);
}

/****************** Step 2 in Initialisation *******/
Stap_2 (void)
{
    Delay ();
    write_command (s2);
}

/****************** Step 3 in Initialisation *******/
Stap_3 (void)
{
    Delay ();
    write_command (s3);
}

/****************** Function set ******************/
Function_set (void)
{
    Delay ();
    write_command (fs);
}

/****************** Display off *******************/
Disp_off (void)
{
    Delay ();
    write_command (dof);
}

/****************** Display on ********************/
Disp_on (void)
{
    Delay ();
    write_command (don);
}

/****************** Entry Mode Set ****************/
Entry_mode_set (void)
{
    Delay ();
    write_command (ems);
}

/****************** Display Clear *****************/
Disp_clear (void)
{
    Delay ();
    write_command (dc);
}

/****************** Cursor Home *******************/
Cursor_home (void)
{
    Delay ();
    write_command (ch);
}

/****************** Cursor off ********************/
Cursor_off (void)
{
    Delay ();
    write_command (co);
}
```

```
/****************** Display on off control *********/
Disp_control (void)
{
  Delay ();
  write_command (doc);
}

/****************** Cursor and display shift *******/
Cursor_shift (void)
{
  Delay ();
  write_command (cds);
}
/****************** First line of Display **********/
First_line (void)
{
  write_command (fl);
/*   blahfl = Line1_LCD_buffer;*/

  while (*blahfl)
  {
    Delay ();
    write_data (*blahfl++);
  }
}

/****************** Second line of Display *********/
Second_line (void)
{
  write_command (sl);
/*   blahsl = Line2_LCD_buffer;*/

  while (*blahsl)
  {
    Delay ();
    write_data (*blahsl++);
  }
}

/****************** Third line of Display **********/
Third_line (void)
{
  write_command (tl);
/*   blahtl = Line3_LCD_buffer;*/

  while (*blahtl)
  {
    Delay ();
    write_data (*blahtl++);
  }
}

/****************** Fourth line of Display *********/
Fourth_line (void)
{
  write_command (fol);
/*   blahol = Line4_LCD_buffer;*/

  while (*blahol)
  {
    Delay ();
    write_data (*blahol++);
  }
/*   write_command(0x7E);
  write_data(0xC6);*/
}

/****************** Write Character ***************/
Write_char (char Letter)
{
  Delay ();
  write_data (Letter);
```

```
  /* write_command(0x14);*/

}

/****************** Write Data ********************/
Write_data_blah (void)
{
  int i;
  i = 0;
  for (i = 0; i < 21; i++)
  {
  /*   write_data (blah);*/
  };
}

/****************** Delay ************************/
Delaynot(int counter)
{
  word i;
  i = 0;
  while (i < counter)
  {
    i++;
  }
}

/* byte testmsg[] = "Hello Ren";

main(void)
{
  int i;
  i = 0;

  InitTimer0();
  InitDisplay();

  codech = Change_code_choice;
  viewfl = View_file_choice;
  writef = Write_file_choice;
  diskfo = Format_disk_choice;

  blahfl = codech;
  blahsl = viewfl;
  blahtl = writef;
  blahol = diskfo;

  First_line();
  Second_line();
  Third_line();
  Fourth_line();
  while(1)
  {}

  for (i=0; testmsg[i]; i++)
    Line1_LCD_buffer[i] = testmsg[i];
  Line1_LCD_buffer[] = {'H', 'e', 'l', 'l', 'o'};
  First_line();
  while(1)
  {

  }
}*/

/*
 * Header program for the LCD module for integration on the floppy disk drive interface.
 */

#ifndef __LCD_H
#define __LCD_H

/****************** Global Defines ****************/
#define byte     unsigned char
#define word     unsigned int
```

```
extern byte *blahfl;
extern byte *blahsl;
extern byte *blahtl;
extern byte *blahol;
extern byte *codech;
extern byte *viewfl;
extern byte *writef;
extern byte *diskfo;

/* extern byte Change_code_choice [];*/
extern byte View_file_choice [];
extern byte Write_file_choice [];
extern byte Format_disk_choice [];
extern byte Exit_choice [];

extern byte wait_code_phrase [];
extern byte code_finish [];
extern byte code_correct [];
extern byte code_incorrect [];
extern byte code_compare [];
extern byte exit_menu [];
extern byte file_write [];
extern byte disk_format [];
/* extern byte code_new [];*/
extern byte disk_init [];
extern byte Nothing [];
extern byte Data_blah [];


extern byte Line_change [];
extern byte Normal_term [];
extern byte Abnormal_term [];
extern byte Default_term [];

/* extern byte Line1_LCD_buffer [] = "Hello Rene";
extern byte Line2_LCD_buffer [] = "Hello Ian";
extern byte Line3_LCD_buffer [] = "Hello Andreas";
extern byte Line4_LCD_buffer [] = "Hello Xandri";
*/

extern InitDisplay (void);

#endif /* __LCD_H */
```

A.4 FLOPPY Program

```
/*
 * Main program for the floppy drive
 */

/****************** Include Files ******************/
#include "d:\tesis\progra~1\floppy.h"

byte FIRQ;
byte master;
byte blie;
byte blip;
byte blap;
register byte Flopflag;
byte Floppy_interrupt_flag;
byte Sense_int_ST0;
byte Sense_blah
byte Sense_int_PCN;
byte Format_ST0;
byte Format_ST1;
byte Format_ST2;
byte Format_C;
byte Format_H;
byte Format_R;
byte Format_N;
byte Read_ST0;
byte Read_ST1;
byte Read_ST2;
byte Read_return_C;
byte Read_return_H;
byte Read_return_R;
byte Read_return_N;
byte Write_ST0;
byte Write_ST1;
byte Write_ST2;
byte Write_return_C;
byte Write_return_H;
byte Write_return_R;
byte Write_return_N;
byte Read_Id_ST0;
byte Read_Id_ST1;
byte Read_Id_ST2;
byte Read_Id_C;
byte Read_Id_H;
byte Read_Id_R;
byte Read_Id_N;
byte Sense_ST3;
byte kyk;
byte kykweer;
byte kyknog;
byte control;
byte state;
byte NCN;
byte PCN;
register byte request;
byte C;
register byte H;
register byte R;
register byte N;
byte verkrydata;
register byte inttel;

/****************** Interrupt ***********************/
INTERRUPT (_IE1_) FDC_interrupt()
{
  FIRQ = P3;
  FIRQ = FIRQ & 0x08;
  if (FIRQ == 0x08)
  {
    Flopflag = 1;
  };
}
```

```
/* FDC_interrupt ()
{
  asm
  {
Tag3     org  $19
     push PSW
     push A
     ljmp Tag3
     org  Tag3
     mov  Flopflag ,# $01
     pop  A
     pop  PSW
     reti
  }
}
*/


/****************** Delay *************************/
Delay12u ( void )
{
  word i ;
  i = 0;
  while ( i < 150)
  {
    i ++;
  }
}

Write_operation ( byte D)
{
  poke (0 xA001 , D);
}

Write_control ( byte D)
{
  poke (0 xC001 , D);
}

byte Read_control ( void )
{
  control = peek (0 xC001 );
  return control ;
}

byte Read_operation ( void )
{
  byte D;
  D = peek (0 xA001 );
  return D;
}

Poll_master ( void )
{
  kyk = peek (0 x8000 );
  while (( kyk | 0 x91 ) != 0 x91 )
  {
    kyk = peek (0 x8000 );
    clrbit (P1 .2);
  }
}

Poll_command ( void )
{
  kykweer = peek (0 x8000 );
  while (( kykweer | 0 x91 ) != 0 x91 )
  {
    kykweer = peek (0 x8000 );
    clrbit (P1 .2);
  }
}
```

```
Poll_execution (void)
{
  kyknog = peek(0x8000);
  while ((kyknog | 0xD1) != 0xD1)
  {
    kyknog = peek(0x8000);
    clrbit(P1.2);
  }
}

Poll_executionid (void)
{
  kyknog = peek(0x8000);
  while ((kyknog & 0x80) != 0x80)
  {
    kyknog = peek(0x8000);
    clrbit(P1.2);
  }
}

Poll_executionformat (void)
{
  asm
  {
    mov DPTR,#$8000
lus1    movx A,[DPTR]
    jnb A.7, lus1
    mov A,#$00
    mov DPTR,#$8001
    movx [DPTR], A
    mov DPTR,#$8000
lus2   movx A,[DPTR]
    jnb A.7, lus2
    mov A,#$00
    mov DPTR,#$8001
    movx [DPTR], A
    mov DPTR,#$8000
lus3   movx A,[DPTR]
    jnb A.7, lus3
    mov A,#$00
    mov DPTR,#$8001
    movx [DPTR], A
    mov DPTR,#$8000
lus4   movx A,[DPTR]
    jnb A.7, lus4
    mov A,#$02
    mov DPTR,#$8001
    movx [DPTR], A
  }
}


Poll_executionformat (void)
{
  while (inttel < 72)
  {
    if (Floppy_interrupt_flag)
    {
      Floppy_interrupt_flag = 0;
      inttel++;
      request++;
      switch (request)
      {
        case 1:
          C = 0x00;
          Send_data(0x8001,C);
          write_data(C);
          break;
        case 2:
          H = 0x00;
          Send_data(0x8001,H);
          write_data(H);
```

```
            break;
          case 3:
            Send_data(0x8001,R);
            write_data(R);
            break;
          case 4:
            N = 0x02;
            Send_data(0x8001,N);
            write_data(N);
            request = 0;
            break;
          default:
            break;
        }
        R++;
      }
    }
}

byte Read_master(void)
{
  byte D
  D = peek(0x8000);
  return D;
/*  Delay12u();*/
}

/****************** Send Data **********************/
Send_data(word Adr, byte D)
{
  poke(Adr,D);
}

byte Read_status(void)
{
  clrbit(P1.2);
  state = peek(0x8001);
  return state;
}

Poll_executionread(void)
{
  kyknog = peek(0x8000);
  if (kyknog == 0x81)
  {
    Delay12u();
    verkrydata = Read_status();
    write_data(verkrydata);
    Delay12u();
    verkrydata = Read_status();
    write_data(verkrydata);
    Delay12u();
    verkrydata = Read_status();
    write_data(verkrydata);
    Delay12u();
    verkrydata = Read_status();
    write_data(verkrydata);
    Delay12u();
    verkrydata = Read_status();
    write_data(verkrydata);
    Delay12u();
    verkrydata = Read_status();
    write_data(verkrydata);
    Delay12u();
    verkrydata = Read_status();
    write_data(verkrydata);
  }
  clrbit(P1.2);
}

Poll_executionwrite(void)
{
```

```
  kyknog = peek(0x8000);
  if (kyknog   == 0x81)
  {
    Delay12u ();
    Send_data (0x31);
    write_data (0x31);
    Delay12u ();
    Send_data (0x32);
    write_data (0x32);
    Delay12u ();
    Send_data (0x33);
    write_data (0x33);
    Delay12u ();
    Send_data (0x34);
    write_data (0x34);
    Delay12u ();
    Send_data (0x35);
    write_data (0x35);
    Delay12u ();
    Send_data (0x36);
    write_data (0x36); }
  clrbit (P1.2);
}

/****************** Initialise ********************/
InitFDC (void)
{
  blap = Read_master ();
       Write_control (0x00);

  Disp_clear ();
  write_data (0x30);
  Recalibrate_disk ();
  write_data (0x31);
  Sense_interrupt_status ();
  write_data (0x32);
  Specify_disk ();
  write_data (0x33);


/*   blahfl = disk_init;
  First_line ();
  clrbit (P1.2);
  Seek_disk ();
  Sense_interrupt_status ();
  Recalibrate_disk ();
  Sense_interrupt_status ();*/
}


/****************** Format a Track ****************/
Format_disk (void)
{
  asm
  {
    mov DPTR,#$8000
luus1    movx A,[DPTR]
    jnb A.7, luus1
    mov A,#$4D
    mov DPTR,#$8001
    movx [DPTR], A

    mov DPTR,#$8000
luus2    movx A,[DPTR]
    jnb A.7, luus2
    mov A,#$00
    mov DPTR,#$8001
    movx [DPTR], A

    mov DPTR,#$8000
luus3    movx A,[DPTR]
    jnb A.7, luus3
```

```
    mov A,#$02
    mov DPTR,#$8001
    movx [DPTR], A

    mov DPTR,#$8000
luus4   movx A,[DPTR]
    jnb A.7, luus4
    mov A,#$09
    mov DPTR,#$8001
    movx [DPTR], A

  }
}
/****************** Recalibrate ******************/
Recalibrate_disk (void)
{
  Poll_master ();
  Send_data (0x8001, Recalibrate_1 );
/*  Delay12u ();*/
  Poll_command ();
  Send_data (0x8001, Recalibrate_2 );
/*  Delay12u ();*/
  Poll_command ();
}

/****************** Read ID ***********************/
Read_id (void)
{
  Delay12u ();
  Poll_master ();
  Send_data (0x8001, Read_Id_1 );
  Delay12u ();
  Poll_command ();
  Send_data (0x8001, Read_Id_1 );
  Delay12u ();
  Poll_executionid ();
  Read_Id_ST0 = Read_status ();
  write_data (Read_Id_ST0 + 0x30);
  Delay12u ();
  Read_Id_ST1 = Read_status ();
  write_data (Read_Id_ST1 + 0x30);
  Delay12u ();
  Read_Id_ST2 = Read_status ();
  write_data (Read_Id_ST2 + 0x30);
  Delay12u ();
  Read_Id_C = Read_status ();
  write_data (Read_Id_C + 0x30);
  Delay12u ();
  Read_Id_H = Read_status ();
  write_data (Read_Id_H + 0x30);
  Delay12u ();
  Read_Id_R = Read_status ();
  write_data (Read_Id_R + 0x30);
  Delay12u ();
  Read_Id_N = Read_status ();
  write_data (Read_Id_N + 0x30);
  Delay12u ();
}

/****************** Read Data *********************/
Read_file (void)
{
  Delay12u ();
  Poll_master ();
  Send_data (0x8001, Read_1 );
  Delay12u ();
  Poll_command ();
  Send_data (0x8001, Read_2 );
  Delay12u ();
  Poll_command ();
  Send_data (0x8001, Read_C );
  Delay12u ();
  Poll_command ();
```

```
  Send_data (0x8001, Read_H);
  Delay12u ();
  Poll_command ();
  Send_data (0x8001, Read_R);
  Delay12u ();
  Poll_command ();
  Send_data (0x8001, Read_N);
  Delay12u ();
  Poll_command ();
  Send_data (0x8001, Read_EOT);
  Delay12u ();
  Poll_command ();
  Send_data (0x8001, Read_GPL);
  Delay12u ();
  Poll_command ();
  Send_data (0x8001, Read_DTL);
  Delay12u ();
  Poll_executionread ();
  Read_ST0 = Read_status ();
  write_data (Read_ST0 + 0x30);
  Delay12u ();
  Read_ST1 = Read_status ();
  write_data (Read_ST1 + 0x30);
  Delay12u ();
  Read_ST2 = Read_status ();
  write_data (Read_ST2 + 0x30);
  Delay12u ();
  Read_return_C = Read_status ();
  write_data (Read_return_C);
  Delay12u ();
  Read_return_H = Read_status ();
  write_data (Read_return_H);
  Delay12u ();
  Read_return_R = Read_status ();
  write_data (Read_return_R);
  Delay12u ();
  Read_return_N = Read_status ();
  write_data (Read_return_N);
  Delay12u ();
}

/****************** Write Data ********************/
Write_file (void)
{
  Delay12u ();
  Poll_master ();
  Send_data (0x8001, Write_1);
  Delay12u ();
  Poll_command ();
  Send_data (0x8001, Write_2);
  Delay12u ();
  Poll_command ();
  Send_data (0x8001, Write_C);
  Delay12u ();
  Poll_command ();
  Send_data (0x8001, Write_H);
  Delay12u ();
  Poll_command ();
  Send_data (0x8001, Write_R);
  Delay12u ();
  Poll_command ();
  Send_data (0x8001, Write_N);
  Delay12u ();
  Poll_command ();
  Send_data (0x8001, Write_EOT);
  Delay12u ();
  Poll_command ();
  Send_data (0x8001, Write_GPL);
  Delay12u ();
  Poll_command ();
  Send_data (0x8001, Write_DTL);
  Delay12u ();
  Poll_executionwrite ();
```

```
  Write_ST0 = Read_status ();
  write_data ( Write_ST0 + 0x30 );
  Delay12u ();
  Write_ST1 = Read_status ();
  write_data ( Write_ST1 + 0x30 );
  Delay12u ();
  Write_ST2 = Read_status ();
  write_data ( Write_ST2 + 0x30 );
  Delay12u ();
  Write_return_C = Read_status ();
  write_data ( Write_return_C );
  Delay12u ();
  Write_return_H = Read_status ();
  write_data ( Write_return_H );
  Delay12u ();
  Write_return_R = Read_status ();
  write_data ( Write_return_R );
  Delay12u ();
  Write_return_N = Read_status ();
  write_data ( Write_return_N );
  Delay12u ();
}

/****************** Seek ************************/
Seek_disk ( void )
{
 /* Delay12u ();*/
  Poll_master ();
  Send_data (0x8001 , Seek_1 );
/*  Delay12u ();*/
  Poll_command ();
  Send_data (0x8001 , Seek_2_head_1 );
/*  Delay12u ();*/
  Poll_command ();
  Send_data (0x8001 , Seek_4 );
/*  Delay12u ();*/
  Poll_command ();
}


/****************** Specify ************************/
Specify_disk ( void )
{
/*  Delay12u ();*/
  Poll_master ();
  Send_data (0x8001 , Specify_1 );
/*  Delay12u ();*/
  Poll_command ();
  Send_data (0x8001 , Specify_2 );
/*  Delay12u ();*/
  Poll_command ();
  Send_data (0x8001 , Specify_3 );
/*  Delay12u ();*/
  Poll_command ();
}


Sense_drive ( void )
{
/*  Delay12u ();*/
  Poll_master ();
  Send_data (0x8001 , Sense_1 );
/*  Delay12u ();*/
  Poll_command ();
  Send_data (0x8001 , Sense_2 );
  Poll_execution ();
/*  Delay12u ();*/
  Sense_ST3 = Read_status ();
  write_data ( Sense_ST3 + 0x30 );
/*  Delay12u ();*/
}

/****************** Sense Interrupt Status *********/
```

```
Sense_interrupt_status (void)
{
/*   Delay12u ();*/
  Poll_command ();
  Send_data (0x8001, Intstate_1 );
  Poll_execution ();
  Sense_int_ST0 = Read_status ();
/*   Delay12u ();*/
  Sense_int_PCN = Read_status ();
/*   Delay12u ();*/
  Sense_blah = Sense_int_ST0 & 0xE0;
/*   Delay12u ();*/
/*   switch ( Sense_blah )
  {
    case Ready_line_change :
      Disp_clear ();
      blahfl = Line_change;
      First_line ();
      break;

    case Normal_termination :
      Disp_clear ();
      blahfl = Normal_term;
      First_line ();
      break;

    case Abnormal_termination :
      Disp_clear ();
      blahfl = Abnormal_term;
      First_line ();
      break;

    default :
      Disp_clear ();
      blahfl = Default_term;
      First_line ();
      break;
  }
*/}

/* Header pgrogram for floppy
 */

#define byte   unsigned char
#define word   unsigned int

#define Specify_1  0x03
#define Specify_2  0xDF
#define Specify_3  0x05

#define Ready_line_change    0xC0
#define Normal_termination   0x20
#define Abnormal_termination   0x60

#define Recalibrate_1  0x07
#define Recalibrate_2  0x00

#define Sense_1    0x04
#define Sense_2    0x00

#define Seek_1     0x0F
#define Seek_2_head_1  0x00
#define Seek_4     0x32
#define Seek_3_head_2  0x04

#define Intstate_1     0x08

#define Format_1     0x4D
#define Format_2_head_1  0x00
#define Format_3_head_2  0x04
#define Bytes_sector   0x02
#define Sector_track   0x09
#define Gap3       0x54
#define Filler_byte    0xFF
```

```
#define  Read_1      0xE6
#define  Read_2      0x00
#define  Read_C      0x00
#define  Read_H      0x00
#define  Read_R      0x00
#define  Read_N      0x02
#define  Read_EOT      0x12
#define  Read_GPL      0x1B
#define  Read_DTL      0xFF

#define  Write_1     0xC5
#define  Write_2     0x00
#define  Write_C     0x00
#define  Write_H     0x00
#define  Write_R     0x00
#define  Write_N     0x02
#define  Write_EOT     0x12
#define  Write_GPL     0x1B
#define  Write_DTL     0xFF

#define  Read_Id_1     0x4A
#define  Read_Id_2     0x00

extern  byte  FIRQ;
extern  byte  master;
extern  byte  blie;
extern  byte  blip;
extern  byte  blap;
extern  byte  NCN;
extern  byte  PCN;
extern  byte  Floppy_interrupt_flag;
extern  byte  Sense_int_ST0;
extern  byte  Sense_blah
extern  byte  Sense_int_PCN;
extern  byte  Format_ST0;
extern  byte  Format_ST1;
extern  byte  Format_ST2;
extern  byte  Format_C;
extern  byte  Format_H;
extern  byte  Format_R;
extern  byte  Format_N;
extern  byte  Read_ST0;
extern  byte  Read_ST1;
extern  byte  Read_ST2;
extern  byte  Read_return_C;
extern  byte  Read_return_H;
extern  byte  Read_return_R;
extern  byte  Read_return_N;
extern  byte  Write_ST0;
extern  byte  Write_ST1;
extern  byte  Write_ST2;
extern  byte  Write_return_C;
extern  byte  Write_return_H;
extern  byte  Write_return_R;
extern  byte  Write_return_N;
extern  byte  Read_Id_ST0;
extern  byte  Read_Id_ST1;
extern  byte  Read_Id_ST2;
extern  byte  Read_Id_C;
extern  byte  Read_Id_H;
extern  byte  Read_Id_R;
extern  byte  Read_Id_N;
extern  byte  Sense_ST3;
extern  byte  kyk;
extern  byte  kykweer;
extern  byte  kyknog;
extern  byte  control;
extern  byte  state;
/* extern  byte  request;*/
extern  byte  C;
/* extern  byte  H;
extern  byte  R;
```

A.5 RS232 Program

```
/*
 * Serial communication with the RS232 port of the computer
 */

/****************** Include Files ******************/
#include "d:\tesis\progra~1\serie.h"

/****************** Variables ********************/
byte Disk_data[20];
byte Previous_disk_data[20];
byte Previous_serie_data;
byte Serie_data;
byte New_serie_data_flag;
byte Ser_een;
byte Data_there;
int Data_tel;
byte * Disk_data_ptr;
byte ID_correct;
byte TYPE_correct;
byte SOH_received;
byte sum;

/****************** Serial initialise **************/

Serial_Init(void)
{
  PCON = 0x00;
  IE = 0x93;
  IP = 0x01;
  TMOD = 0x21;
  TH1 = -24;
  SCON = 0x50;
  setbit(TCON.6);
  setbit(TCON.0);
  clrbit(TCON.5);
}


INTERRUPT (_SER_) Serial_comms()
{
  Previous_serie_data = Serie_data;
  Serie_data = SBUF;
  New_serie_data_flag = 1;
  clrbit(SCON.0);
}

Send_seriedata(byte D)
{
  SBUF = D;
  while ((SCON & 0x02) != 0x02)
  {
    /* Do nothing */
  }
  clrbit(SCON.1);
}

Receive_seriedata(void)
{
  switch(Serie_data)
  {
    case SOH:
      SOH_received = 1;
      break;

    case TYPE:
      TYPE_correct = 1;
      break;

    case ID:
      ID_correct = 1;
      break;
```

```
    case ETX:
      ID_correct = 0;
      TYPE_correct = 0;
      SOH_received = 0;
      break;

    default:
      Display_data();
      break;
  }
}

Display_data(void)
{
  if (SOH_received && TYPE_correct && ID_correct)
  {
    if ((Serie_data != ID) && (Serie_data != TYPE))
    {
      switch(Data_tel)
      {
        case 1:
          write_command(f1);
          break;

        case 21:
          write_command(s1);
          break;

        case 41:
          write_command(t1);
          break;

        case 61:
          write_command(fo1);
          break;

        case 81:
          Data_tel = 0;
          break;

        default:
          break;
      }
      write_data(Serie_data);
      sum = Serie_data + sum;
      Data_tel++;
    }
  }
}
/*
 * Header file for Serie.c
 */

#define byte     unsigned char

#define SOH    0x01
#define ETX    0x03
#define ACK    0x06
#define SYN    0x16
#define ETB    0x17
#define    TYPE     0x80
#define    ID     0x90

extern byte Disk_data[];
extern byte Previous_disk_data[];
extern byte New_serie_data_flag;
extern byte Ser_een;
extern byte Data_there;
extern byte Previous_serie_data;
extern byte Serie_data;
extern byte ID_correct;
extern byte TYPE_correct;
```

A.6 Radio Link Program

```
/****************** Include Files ******************/
#include <8051io.h>
#include <8051reg.h>
#include <8051int.h>
#include <8051bit.h>
#include "d:\tesis\progra~1\radio.h"

/****************** Variables **********************/
byte Previous_receive_data;
byte Receive_data;
byte New_receive_data_flag;
byte Send_data;
byte Data_lenght;
byte Data_lees;
byte ID_correct;
byte TYPE_correct;
byte SOH_received;
byte Data_stuur;
byte temp;

/****************** Serial initialise **************/

Serial_Init(void)
{
  PCON = 0x00;
  IE = 0x93;
  IP = 0x01;
  TMOD = 0x21;
  TH1 = -24;
  SCON = 0x50;
  setbit(TCON.6);
  setbit(TCON.0);
  clrbit(TCON.5);
}

Send_seriedata(byte Data)
{
  SBUF = Data;
  while ((SCON & 0x02) != 0x02)
  {
    /* Do nothing */
  }
  clrbit(SCON.1);
}

byte RPC_to_Host(void)
{
  asm
  {
    clr P3.3

lusa    mov A,P3
    jnb A.2,lusa

    mov A,P1
    anl A,#$0F
    mov R1,A
    setb P3.3

lusb    mov A,P3
    jb A.2,lusb

    clr P3.3

lusc    mov A,P3
    jnb A.2,lusc

    mov A,P1
    swap A
    anl A,#$F0
    orl A,R1
```

```
    mov Data_lees,A
    setb P3.3
  }
  return Data_lees;
}

Host_to_RPC(void)
{
  asm
  {
    clr P3.5

lus   mov A,P3
    jb A.4,lus
    mov A,Data_stuur

    mov P1,A
    setb P3.5

    swap A
    mov temp,A

lus1    mov A,P3
    jnb A.4,lus1
    clr P3.5

lus2    mov A,P3
    jb A.4,lus2

    mov A,temp
    mov P1,A
    setb P3.5

lus3    mov A,P3
    jnb A.4,lus3
  }
}


INTERRUPT (_SER_) Serial_comms()
{
  Previous_receive_data = Receive_data;
  Receive_data = SBUF;
  New_receive_data_flag = 1;
  clrbit(SCON.0);
}

INTERRUPT (_IE0_) Radio_comms()
{
  Send_data = RPC_to_Host();
  Send_seriedata(Send_data);
}

Receive_seriedata(void)
{
  switch(Receive_data)
  {
    case SOH:
      SOH_received = 1;
      break;

    case TYPE:
      TYPE_correct = 1;
      break;

    case ID:
      ID_correct = 1;
      break;

    case ETX:
      ID_correct = 0;
      TYPE_correct = 0;
      SOH_received = 0;
```

```
        break;

      default:
        Display_data();
        break;
    }
}

Display_data(void)
{
  if (SOH_received && TYPE_correct && ID_correct)
  {
    if (Previous_receive_data == ID)
    {
      Data_lenght = Receive_data;
      Data_stuur = (Data_lenght & 0x1F);
      Host_to_RPC();
      Data_stuur = SOH;
      Host_to_RPC();
      Data_stuur = TYPE;
      Host_to_RPC();
      Data_stuur = ID;
      Host_to_RPC();
    }
    if ((Receive_data != ID) && (Receive_data != TYPE))
    {
      Data_stuur = Receive_data;
      Host_to_RPC();
    }
  }
}


Init_RPC(void)
{
  Data_stuur = 0xC0;
  Host_to_RPC();
  Data_stuur = 0x33;
  Host_to_RPC();
  Data_stuur = 0xC1;
  Host_to_RPC();
  Data_stuur = 0x01;
  Host_to_RPC();
}

main(void)
{
  New_receive_data_flag = 0;

  while(1)
  {
    if (New_receive_data_flag == 1)
    {
      Receive_seriedata();
      New_receive_data_flag = 0;
    }

  }
}
/*
 * Header program for the Radio link.
 */

#define byte     unsigned char

#define SOH    0x01
#define ETX    0x03
#define ACK    0x06
#define SYN    0x16
#define ETB    0x17
#define    TYPE     0x80
#define    ID     0x90
```

```
/* extern  byte  Previous_receive_data ;
extern  byte  Receive_data ;
extern  byte  New_receive_data_flag ;
extern  byte  Send_data ;
extern  byte  Data_lenght ;
extern  byte  Data_lees ;
extern  byte  ID_correct ;
extern  byte  TYPE_correct ;
extern  byte  SOH_received ;*/
```

A.7 VHDL Program

```
Library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_arith.all;
   use IEEE.std_logic_unsigned.all;

ENTITY CPLD IS
  PORT(
       A13    : IN  STD_LOGIC;
       A14    : IN  STD_LOGIC;
       A15    : IN  STD_LOGIC;
       WR     : IN  STD_LOGIC;
       RD     : IN  STD_LOGIC;
       DAV    : IN  STD_LOGIC;
       OE     : OUT STD_LOGIC;
       CS     : OUT STD_LOGIC;
       LDOR   : OUT STD_LOGIC;
       LDCR   : OUT STD_LOGIC;
       CS0    : OUT STD_LOGIC;
       INT0   : OUT STD_LOGIC
    );
END CPLD;


ARCHITECTURE MODEL OF CPLD IS
BEGIN


MEMORY:
PROCESS ( A15, A14, A13)
  VARIABLE DEC : STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN

DEC := A15 & A14 & A13;
IF DEC = "000" THEN
  CS0    <= '0';
ELSE
  CS0    <= '1';
END IF;

END PROCESS MEMORY;


KEYPAD_SELECT:
PROCESS ( A15, A14, A13)
  VARIABLE DEC : STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN

DEC := A15 & A14 & A13;
IF DEC = "001" THEN
  OE     <= '0';
ELSE
  OE     <= '1';
END IF;

END PROCESS KEYPAD_SELECT;


KEYPAD_INTERRUPT:
PROCESS ( DAV)
BEGIN

INT0   <= NOT DAV;

END PROCESS KEYPAD_INTERRUPT;


FDC_CS:
PROCESS ( A15, A14, A13)
  VARIABLE DEC : STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN
```

```
DEC := A15 & A14 & A13;
IF DEC = "100" THEN
  CS     <= '0';
ELSE
  CS     <= '1';
END IF;

END PROCESS FDC_CS;


FDC_LDOR:
PROCESS ( A15, A14, A13)
  VARIABLE DEC : STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN

DEC := A15 & A14 & A13;
IF DEC = "101" THEN
  LDOR   <= '0';
ELSE
  LDOR   <= '1';
END IF;

END PROCESS FDC_LDOR;


FDC_LDCR:
PROCESS ( A15, A14, A13)
  VARIABLE DEC : STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN

DEC := A15 & A14 & A13;
IF DEC = "110" THEN
  LDCR   <= '0';
ELSE
  LDCR   <= '1';
END IF;

END PROCESS FDC_LDCR;


END MODEL;
```

Appendix B

Users Manual for SAU

B.1 Installation

The Stand Alone Unit (SAU) comes with a RS232 connection to the Radio Link, as well as the ribbon cable for the floppy drive. There is also a power cable that has to be connected to the outside of the SAU. Figure B.1 shows the layout of the SAU. From this figure it is very easy to see where to connect the various cables. After the cables have been connected to the correct slots, the SAU is ready to be used. It's that Simple!

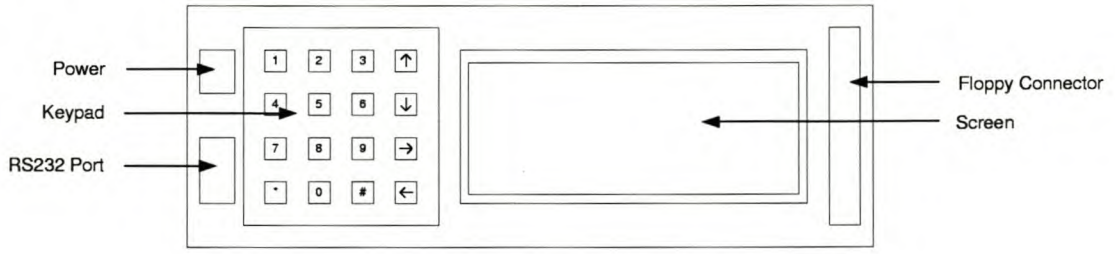

Figure B.1: SAU Layout

B.2 Setup

The SAU is very user friendly and there is no need for any setup from the user. Each SAU comes with its own specific Unitnumber, which can be located on the back of the SAU. This Unitnumber is important, because this will be how the PC will be able to recognise the specific SAU. Remember this number.

Each SAU also comes with a specific User code. Your User code is: 0000. Remember this code and do not give it to anyone else. As soon as the SAU is switched on, the Unit will wait for this code to be entered. If the code is not entered correctly, you will be thrown off the system. This code cannot be changed.

B.3 Use

As soon as the SAU is switched on, the screen goes completely blank. Follow the next steps to get your SAU into full operation:

STEP 1: Press the [*] button. This will tell your unit that you are ready to proceed and immediately the following line appears on the screen: Waiting for code...

STEP 2: Now you can start typing in your code. With each button pressed * will be displayed on the screen. When you have finished typing in your code, the screen will display the following: ****

STEP 3: Now you can press the Enter button [#]. This will let your SAU know that you have finished typing in the code. If the code was typed in correctly, the SAU will immediately tell you that it was correct and then display your four options:

1. Read File
2. Write File

3. Format Disk
4. Exit

STEP 4: Press one of the four buttons [1] to [4] depending on what you want to do. If you want to read a file from the disk press [1], if you want to write a file to the disk press [2], etc.

STEP 5: If you pressed [1] read Step 6 , if you pressed [2] read Step 7 , if you pressed [3] read Step 8 and If you pressed [4] read Step 9.

STEP 6: You are now in the Read File state. Now you can use the up and down arrows on the keypad to scroll through the lines one by one or the left and right arrows to scroll through the pages.

STEP 7: You are now in the Write File state. It is now possible for the person attending to the PC to send the data to you and it will be written to your Floppy Drive.

STEP 8: You are now formatting your Disk. Really hope that's what you wanted to do!

STEP 9: You are now exiting the system. If you want to return, you have to start at the beginning again. Press [*] and then continue from Step 2.

Appendix C

Source Code for Windows Interface

C.1 Parent form

```
unit Edit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, StdCtrls, ComCtrls, ActnList, ImgList, ToolWin, Seriecoms, UnitData,
  About;

type
  TFrameForm = class(TForm)
    OpenDialog1: TOpenDialog;
    ToolbarImageP: TImageList;
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    New1: TMenuItem;
    Open1: TMenuItem;
    N1: TMenuItem;
    Save1: TMenuItem;
    SaveAs1: TMenuItem;
    N2: TMenuItem;
    PrinterSetup1: TMenuItem;
    Print1: TMenuItem;
    N4: TMenuItem;
    Send1: TMenuItem;
    N5: TMenuItem;
    Exit1: TMenuItem;
    Edit1: TMenuItem;
    Undo1: TMenuItem;
    N6: TMenuItem;
    Cut1: TMenuItem;
    Copy1: TMenuItem;
    Paste1: TMenuItem;
    N8: TMenuItem;
    SelectAll1: TMenuItem;
    Search1: TMenuItem;
    Find1: TMenuItem;
    Replace1: TMenuItem;
    View1: TMenuItem;
    Statusbar1: TMenuItem;
    Help1: TMenuItem;
    HelpTopics1: TMenuItem;
    Index1: TMenuItem;
    N9: TMenuItem;
    About1: TMenuItem;
    ToolBar1: TToolBar;
    NewButton: TToolButton;
    OpenButton: TToolButton;
    SaveButton: TToolButton;
    Space1: TToolButton;
    Space2: TToolButton;
    PrintButton: TToolButton;
    Space3: TToolButton;
    CutButton: TToolButton;
    CopyButton: TToolButton;
    PasteButton: TToolButton;
    Space4: TToolButton;
    UndoButton: TToolButton;
    Space5: TToolButton;
    FindButton: TToolButton;
    ReplaceButton: TToolButton;
    Space6: TToolButton;
    ConnectButton: TToolButton;
    Space7: TToolButton;
    ExitButton: TToolButton;
    SaveDialog1: TSaveDialog;
    FindDialog1: TFindDialog;
    ReplaceDialog1: TReplaceDialog;
    PrintDialog1: TPrintDialog;
    PrinterSetupDialog1: TPrinterSetupDialog;
    EnterUnitsButton: TToolButton;
```

```
    N10: TMenuItem;
    EnterUnits1 : TMenuItem;
    RedoButton : TToolButton ;
    Redo1 : TMenuItem;
    Toolbar : TMenuItem;
    StatusBar2 : TStatusBar ;
    procedure Open1Click ( Sender : TObject );
    procedure Exit1Click ( Sender : TObject );
    procedure New1Click ( Sender : TObject );
    procedure FindDialog1Find ( Sender : TObject );
    procedure ReplaceDialog1Replace ( Sender : TObject );
    procedure SaveButtonClick ( Sender : TObject );
    procedure FindButtonClick ( Sender : TObject );
    procedure ReplaceButtonClick ( Sender : TObject );
    procedure ConnectButtonClick ( Sender : TObject );
    procedure UndoButtonClick ( Sender : TObject );
    procedure CutButtonClick ( Sender : TObject );
    procedure CopyButtonClick ( Sender : TObject );
    procedure PasteButtonClick ( Sender : TObject );
    procedure PrintButtonClick ( Sender : TObject );
    procedure EnterUnits1Click ( Sender : TObject );
    procedure EnterUnitsButtonClick ( Sender : TObject );
    procedure About1Click ( Sender : TObject );
    procedure ToolbarClick ( Sender : TObject );
    procedure RedoButtonClick ( Sender : TObject );
    procedure Statusbar1Click ( Sender : TObject );
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FrameForm : TFrameForm;
  PathName : string ;
implementation

  uses NewEdit, Serconnect;

{$R *.DFM}

procedure TFrameForm.New1Click ( Sender : TObject );
begin
{Create a new form named NewEditor}
  TNewEditor.Create ( Self );
{Enable all the buttons after new file is opened}
  FrameForm.Save1.Enabled := True;
  FrameForm.SaveButton.Enabled := True;
  FrameForm.PrintButton.Enabled := True;
  FrameForm.CutButton.Enabled := True;
  FrameForm.CopyButton.Enabled := True;
  FrameForm.PasteButton.Enabled := True;
  FrameForm.UndoButton.Enabled := True;
  FrameForm.RedoButton.Enabled := True;
  FrameForm.FindButton.Enabled := True;
  FrameForm.ReplaceButton.Enabled := True;
  FrameForm.ConnectButton.Enabled := True;
end;

procedure TFrameForm.Exit1Click ( Sender : TObject );
begin
{Close form}
  Close;
end;

procedure TFrameForm.Open1Click ( Sender : TObject );
begin
{Open excisting file using the Open dialog box}
  if OpenDialog1.Execute then
  with TNewEditor.Create ( Self ) do
    Open( OpenDialog1.FileName );

{Enable all the buttons after an excisting file is opened}
```

```
  FrameForm.SaveButton.Enabled := True;
  FrameForm.PrintButton.Enabled := True;
  FrameForm.CutButton.Enabled := True;
  FrameForm.CopyButton.Enabled := True;
  FrameForm.PasteButton.Enabled := True;
  FrameForm.UndoButton.Enabled := True;
  FrameForm.RedoButton.Enabled := True;
  FrameForm.FindButton.Enabled := True;
  FrameForm.ReplaceButton.Enabled := True;
  FrameForm.ConnectButton.Enabled := True;
end;

procedure TFrameForm.SaveButtonClick(Sender: TObject);
begin
{This procedure uses the Save dialog box to save the file being edited, it also
 checks whether the file has been saved under a different name or not}
  if PathName = DefaultFileName then
  begin
    SaveDialog1.FileName := PathName;
    if SaveDialog1.Execute then
    begin
      PathName := SaveDialog1.FileName;
      NewEditor.Caption := ExtractFileName(PathName);
    end;
  end
  else
    begin
      NewEditor.NewRichEdit.Lines.SaveToFile(PathName);
      NewEditor.NewRichEdit.Modified := False;
    end;

end;

procedure TFrameForm.FindButtonClick(Sender: TObject);
begin
{This procedure executes the Find dialog box}
  FindDialog1.Execute;
end;

procedure TFrameForm.FindDialog1Find(Sender: TObject);
var
  FoundAt: LongInt;
  StartPos, ToEnd: integer;
begin
{This is the Find dialog box and all the settings in it}
  with NewEditor.NewRichEdit do
  begin
    if SelLength <> 0 then
      StartPos := SelStart + SelLength
    else
      StartPos := 0;
    ToEnd := Length(Text) - StartPos;
    FoundAt := FindText(FindDialog1.FindText, StartPos, ToEnd, [stMatchCase]);
    if FoundAt <> -1 then
    begin
      SetFocus;
      SelStart := FoundAt;
      SelLength := Length(FindDialog1.FindText);
    end;
  end;
end;

procedure TFrameForm.ReplaceButtonClick(Sender: TObject);
begin
{This procedure executes the Replace dialog box}
    ReplaceDialog1.Execute;
end;

procedure TFrameForm.ReplaceDialog1Replace(Sender: TObject);

var
  SelPos: Integer;
begin
```

```
{This is the Replace dialog box and all the settings in it}
  with TReplaceDialog(Sender) do
  begin
    SelPos := Pos(FindText, NewEditor.NewRichEdit.Lines.Text);
    if SelPos > 0 then
    begin
      NewEditor.NewRichEdit.SelStart := SelPos - 1;
      NewEditor.NewRichEdit.SelLength := Length(FindText);
      NewEditor.NewRichEdit.SelText := ReplaceText;
    end
    else MessageDlg(Concat('Could not find "', FindText, '" in NewEditor.'),
     mtError, [mbOk], 0);
  end;
end;

procedure TFrameForm.ConnectButtonClick(Sender: TObject);
begin
{Shows the Serial connect form as soon as the connect button is selected. The
 form is shown as a Modal form, which means that the application cannot continue
 before this form is not closed}
   SerialConnect.ShowModal;
end;

procedure TFrameForm.UndoButtonClick(Sender: TObject);
begin
{If this button is selected, the Undo process in form NewEditor is executed}
   NewEditor.Undo1Click(Sender);
end;

procedure TFrameForm.CutButtonClick(Sender: TObject);
begin
{If this button is selected, the Cut process in form NewEditor is executed}
    NewEditor.Cut1Click(Sender);
end;

procedure TFrameForm.CopyButtonClick(Sender: TObject);
begin
{If this button is selected, the Copy process in form NewEditor is executed}
    NewEditor.Copy1Click(Sender);
end;

procedure TFrameForm.PasteButtonClick(Sender: TObject);
begin
{If this button is selected, the Paste process in form NewEditor is executed}
    NewEditor.Paste1Click(Sender);
end;

procedure TFrameForm.PrintButtonClick(Sender: TObject);
begin
{Executes the Print dialog and also states which file to print}
  if PrintDialog1.Execute then
    NewEditor.NewRichEdit.Print(PathName);
end;

procedure TFrameForm.EnterUnits1Click(Sender: TObject);
begin
{Displays the Unit data form when selected under the Edit menu item}
    UnitData.Data.Visible := True;
end;

procedure TFrameForm.EnterUnitsButtonClick(Sender: TObject);
begin
{Jumps to the process where the Unit data form is made visible}
   FrameForm.EnterUnits1Click(Sender);
end;

procedure TFrameForm.About1Click(Sender: TObject);
begin
{Displays the About form when selected under the Help menu item}
   AboutForm.Visible := True;
end;

procedure TFrameForm.ToolbarClick(Sender: TObject);
```

```
begin
{Displays or removes the Toolbar}
   if ToolBar1.Visible then
     begin
      Toolbar.ImageIndex := -1;
      ToolBar1.Visible := False
     end
   else
     begin
      ToolBar1.Visible := True;
      Toolbar.ImageIndex := 13;
     end;
end;

procedure TFrameForm.RedoButtonClick(Sender: TObject);
begin
{If this button is selected, the Redo process in form NewEditor is executed}
   NewEditor.Redo1Click(Sender);
end;

procedure TFrameForm.Statusbar1Click(Sender: TObject);
begin
{Displays or removes the Statusbar}
   if StatusBar2.Visible then
     begin
      Statusbar1.ImageIndex := -1;
      StatusBar2.Visible := False
     end
   else
     begin
      StatusBar2.Visible := True;
      Statusbar1.ImageIndex := 13;
     end;
end;

end.
```

C.2 Child form

```
unit NewEdit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, StdCtrls, ComCtrls, SckComp, ToolWin, ImgList, ActnList, Serconnect;

type
  TNewEditor = class(TForm)
    NewRichEdit: TRichEdit;
    MainMenu1: TMainMenu;
    PopupMenu1: TPopupMenu;
    File1: TMenuItem;
    New1: TMenuItem;
    Open1: TMenuItem;
    N1: TMenuItem;
    Save1: TMenuItem;
    SaveAs1: TMenuItem;
    N2: TMenuItem;
    PrinterSetup1: TMenuItem;
    Print1: TMenuItem;
    N4: TMenuItem;
    Send1: TMenuItem;
    N5: TMenuItem;
    Exit1: TMenuItem;
    Edit1: TMenuItem;
    N6: TMenuItem;
    Cut1: TMenuItem;
    Copy1: TMenuItem;
    Paste1: TMenuItem;
    SelectAll1: TMenuItem;
    Search1: TMenuItem;
    Find1: TMenuItem;
    Replace1: TMenuItem;
    View1: TMenuItem;
    Statusbar1: TMenuItem;
    Help1: TMenuItem;
    HelpTopics1: TMenuItem;
    Index1: TMenuItem;
    N8: TMenuItem;
    About1: TMenuItem;
    Undo2: TMenuItem;
    N9: TMenuItem;
    Cut2: TMenuItem;
    Copy2: TMenuItem;
    Paste2: TMenuItem;
    N10: TMenuItem;
    Delete2: TMenuItem;
    N11: TMenuItem;
    SelectAll2: TMenuItem;
    ActionList1: TActionList;
    N12: TMenuItem;
    EnterUnits1: TMenuItem;
    Toolbar: TMenuItem;
    N3: TMenuItem;
    Redo1: TMenuItem;
    Undo1: TMenuItem;
    procedure FormCreate(Sender: TObject);
    procedure Open1Click(Sender: TObject);
    procedure New1Click(Sender: TObject);
    procedure Send1Click(Sender: TObject);
    procedure Save1Click(Sender: TObject);
    procedure Saveas1Click(Sender: TObject);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Print1Click(Sender: TObject);
    procedure PrintSetup1Click(Sender: TObject);
    procedure Copy1Click(Sender: TObject);
    procedure Cut1Click(Sender: TObject);
    procedure Paste1Click(Sender: TObject);
```

```
    procedure Find1Click(Sender: TObject);
    procedure Replace1Click(Sender: TObject);
    procedure Edit1Click(Sender: TObject);
    procedure Exit1Click(Sender: TObject);
    procedure Undo1Click(Sender: TObject);
    procedure SelectAll1Click(Sender: TObject);
    procedure EnterUnits1Click(Sender: TObject);
    procedure ToolbarClick(Sender: TObject);
    procedure PrinterSetup1Click(Sender: TObject);
    procedure Redo1Click(Sender: TObject);

  private
    { Private declarations }
  public
    { Public declarations }

    procedure Open(AFileName: string);
  end;

var
  NewEditor: TNewEditor;

const
  DefaultFileName = 'Untitled';

implementation

uses Clipbrd, Printers, Edit, Seriecoms;

{$R *.DFM}

procedure TNewEditor.FormCreate(Sender: TObject);
begin
  PathName := DefaultFileName;
end;

procedure TNewEditor.Open(AFileName: string);
begin
  PathName := AFileName;
  Caption := ExtractFileName(AFileName);
  NewRichEdit.Lines.Add(AFileName);
  with NewRichEdit do
   begin
    Lines.LoadFromFile(AFileName);
    SelStart := 0;
    Modified := False;
   end;
end;

procedure TNewEditor.Open1Click(Sender: TObject);
begin
  FrameForm.Open1Click(Sender);
end;

procedure TNewEditor.New1Click(Sender: TObject);
begin
  FrameForm.New1Click(Sender);
end;

procedure TNewEditor.Send1Click(Sender: TObject);
begin
  FrameForm.Exit1Click(Self);
end;

procedure TNewEditor.Save1Click(Sender: TObject);
begin
  if PathName = DefaultFileName then
    Saveas1Click(Self)
  else
    begin
      NewRichEdit.Lines.SaveToFile(PathName);
      NewRichEdit.Modified := False;
    end;
```

```
end;

procedure TNewEditor.Saveas1Click(Sender: TObject);
begin
  FrameForm.SaveDialog1.FileName := PathName;
  if FrameForm.SaveDialog1.Execute then
  begin
    PathName := FrameForm.SaveDialog1.FileName;
    Caption := ExtractFileName(PathName);
    Save1Click(Sender);
  end;
end;

procedure TNewEditor.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
 const
   SWarningText = 'Save changes to %s?';
begin
  if NewRichEdit.Modified then
  begin
    case MessageDlg(Format(SWarningText, [PathName]), mtConfirmation,
      [mbYes, mbNo, mbCancel], 0) of
      idYes: Save1Click(Self);
      idCancel: CanClose := False;
    end;
  end;
  FrameForm.SaveButton.Enabled := False;
  FrameForm.PrintButton.Enabled := False;
  FrameForm.CutButton.Enabled := False;
  FrameForm.CopyButton.Enabled := False;
  FrameForm.PasteButton.Enabled := False;
  FrameForm.UndoButton.Enabled := False;
  FrameForm.RedoButton.Enabled := False;
  FrameForm.FindButton.Enabled := False;
  FrameForm.ReplaceButton.Enabled := False;
  FrameForm.ConnectButton.Enabled := False;
end;

procedure TNewEditor.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Action := caFree;
end;

procedure TNewEditor.Print1Click(Sender: TObject);
begin
  if FrameForm.PrintDialog1.Execute then
    NewRichEdit.Print(PathName);
end;

procedure TNewEditor.PrintSetup1Click(Sender: TObject);
begin
  FrameForm.PrinterSetupDialog1.Execute;
end;

procedure TNewEditor.Copy1Click(Sender: TObject);
begin
  NewRichEdit.CopyToClipboard;
end;

procedure TNewEditor.Cut1Click(Sender: TObject);
begin
  NewRichEdit.CutToClipboard;
end;

procedure TNewEditor.Paste1Click(Sender: TObject);
begin
   NewRichEdit.PasteFromClipboard;
end;

procedure TNewEditor.Find1Click(Sender: TObject);
begin
   FrameForm.FindDialog1.Execute;
end;
```

```
procedure TNewEditor.Replace1Click(Sender: TObject);
begin
  FrameForm.ReplaceDialog1.Execute;
end;

procedure TNewEditor.Edit1Click(Sender: TObject);
  var
   HasSelection: Boolean;
begin
  Paste1.Enabled := Clipboard.HasFormat(CF_TEXT);
  Paste2.Enabled := Paste1.Enabled;
  HasSelection := NewRichEdit.SelLength > 0;
  Cut1.Enabled := HasSelection;
  Cut2.Enabled := HasSelection;
  Copy1.Enabled := HasSelection;
  Copy2.Enabled := HasSelection;
end;

procedure TNewEditor.Exit1Click(Sender: TObject);
begin
  FrameForm.Exit1Click(Sender);
end;

procedure TNewEditor.Undo1Click(Sender: TObject);
begin
  NewRichEdit.Undo;
end;

procedure TNewEditor.SelectAll1Click(Sender: TObject);
begin
  NewRichEdit.SelectAll;
end;

procedure TNewEditor.EnterUnits1Click(Sender: TObject);
begin
  FrameForm.EnterUnits1Click(Sender);
end;

procedure TNewEditor.ToolbarClick(Sender: TObject);
begin
  FrameForm.ToolbarClick(Sender);
end;

procedure TNewEditor.PrinterSetup1Click(Sender: TObject);
begin
  FrameForm.PrinterSetupDialog1.Execute;
end;

procedure TNewEditor.Redo1Click(Sender: TObject);
begin
  NewRichEdit.Undo;
end;

end.
```

C.3 Serial communication

```
unit Serconnect;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls, Seriecoms, FileCtrl, CheckLst, Unitdata, Db, DBTables,
  DBCtrls;

type
  TSerialConnect = class(TForm)
    Panel1: TPanel;
    Unit_type: TComboBox;
    SerConnect: TButton;
    SerDisconnect: TButton;
    SerSend: TButton;
    SerCancel: TButton;
    Label1: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    FiletoUnit: TFileListBox;
    SerialdataSource: TDataSource;
    LookupUsername: TDBLookupComboBox;
    procedure SerConnectClick(Sender: TObject);
    procedure SerDisconnectClick(Sender: TObject);
    procedure SerSendClick(Sender: TObject);
    procedure Unit_typeChange(Sender: TObject);
    procedure DBComboBox1Change(Sender: TObject);
    procedure LookupUsernameClick(Sender: TObject);
{   procedure ReceiveClick(Sender: TObject);}
  private
    { Private declarations }
  public
    { Public declarations }
      DriverRunning: Boolean;
      DriverClosed: Boolean;
      ID: byte;
      TYP: byte;
  end;

var
  SerialConnect: TSerialConnect;

implementation

uses NewEdit;

{$R *.DFM}

procedure TSerialConnect.SerConnectClick(Sender: TObject);
var
   SeriePort     : TPort;
   buffersize    : Integer;
   Baudrate      : Tbaud;
   ParityEnable  : boolean;
   ByteSize      : Integer;
   Parity        : TParity;
   StopBits      : TStopB;
   Test          : Boolean;
begin
  if (TYP = $80) then
    begin
        SeriePort     :=com2;     {Setup Serie port}
        buffersize    :=20;
        baudrate      :=B38400;
        ParityEnable  :=false;
        ByteSize      :=8;
        Parity        :=0;
        StopBits      :=B1;
        Test:=OpenS_Coms(SeriePort, buffersize, Baudrate, ParityEnable, ByteSize, Parity, StopBits);
        Application.MessageBox('Com_port_opened_and_connected', 'Serial_Communication',
```

```
            MB_OK);
      end

    else if (TYP = $81) then
      begin
           SeriePort     :=com3;     {Setup Serie port}
           buffersize    :=20;
           Baudrate      :=B56000;
           ParityEnable  :=false;
           ByteSize      :=8;
           Parity        :=0;
           StopBits      :=B1;
           Test:=OpenS_Coms(SeriePort, buffersize, Baudrate, ParityEnable, ByteSize, Parity, StopBits);
           Application.MessageBox('Com_port_opened_and_connected', 'Serial_Communication',
             MB_OK);
      end

end;

procedure TSerialConnect.SerDisconnectClick(Sender: TObject);
begin
    CloseS_Coms;
    DriverClosed:=true;
    Application.MessageBox('Com_port_disconnected', 'Serial_Communication',
         MB_OK);
end;

procedure TSerialConnect.SerSendClick(Sender: TObject);
const
  SOH = $01;
  ETX = $03;
  ACK = $06;
  SYN = $16;
  ETB = $17;

var
  Fee: file of byte;
  Fed: TextFile;
  Spe: byte;
  Spd: byte;
  eax: Integer;
  size: Longint;
  LENGTH: Longint;

begin
     AssignFile(Fee, SerialConnect.FiletoUnit.FileName);
     Reset(Fee);
     size := FileSize(Fee);
     LENGTH := size + ID + TYP;
     WriteS_port(SOH);
     {WriteS_port(LENGTH);}
     WriteS_port(TYP);
     WriteS_port(ID);

     while not Eof(Fee) do
     begin
       Read(Fee, Spe);
       WriteS_port(Spe);
     end;
     WriteS_port(ETX);

  {  ReadS_port(Spd);
     Write(Fed, Chr(Spd));
     ReadS_port(Spd);
     Write(Fed, Chr(Spd));
     ReadS_port(Spd);
     Write(Fed, Chr(Spd));
     ReadS_port(Spd);
     Write(Fed, Chr(Spd));}
      CloseFile(Fee);
{     CloseFile(Fed);          }
end;
```

```
{procedure TSerialConnect.ReceiveClick(Sender: TObject);
var
  Fee: TextFile;
  Spe: byte;
  eax: Integer;
begin
   eax := FileCreate('NEWFILE.TXT');
   AssignFile(Fee, 'NEWFILE.TXT');
   ReadS_port(Spe);
   Write(Fee, Chr(Spe));
   while not Eof(Fee) do
   begin
     ReadS_port(Spe);
     Write(Fee, Chr(Spe));
   end;
   CloseFile(Fee);
end; }

{procedure TSerialConnect.Unit_numberChange(Sender: TObject);
begin
    if (SerialConnect.Unit_number.Text = 'Sannie') then
    begin
        ID := $90;
    end
    else if (SerialConnect.Unit_number.Text = 'Jannie') then
    begin
        ID := $91;
    end
    else if (SerialConnect.Unit_number.Text = 'All') then
    begin
        ID := $84;
    end;

end;}

procedure TSerialConnect.Unit_typeChange(Sender: TObject);
begin
    if (SerialConnect.Unit_type.Text = 'Stand_Alone_Unit') then
    begin
        TYP := $80;
    end
    else if (SerialConnect.Unit_type.Text = 'Modem') then
    begin
        TYP := $81;
    end
    else if  (SerialConnect.Unit_type.Text = 'Both') then
    begin
        TYP := $82;
    end;
end;

procedure TSerialConnect.DBComboBox1Change(Sender: TObject);
begin
    Data.UnitdataTable.FieldDefs[1];
end;

procedure TSerialConnect.LookupUsernameClick(Sender: TObject);
begin
   ID := LookupUsername.KeyValue;
end;

end.
```

C.4 Data units

```
unit Unitdata;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, DBCtrls, ExtCtrls, Mask, Db, DBTables, Grids, DBGrids;

type
  TData = class(TForm)
    Panel1: TPanel;
    Name: TLabel;
    Unitnumber: TLabel;
    Fullname: TDBEdit;
    Number: TDBEdit;
    UnitdataSource: TDataSource;
    UnitdataTable: TTable;
    DBNavigator1: TDBNavigator;
    DBGrid1: TDBGrid;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Data: TData;

implementation

{$R *.DFM}

end.
```

C.5 Serial interface

```
unit Seriecoms;
interface
uses
  Windows, Messages, SysUtils, Classes, Dialogs,
  StdCtrls, ExtCtrls;

type
  ExCom = class(exception);
  TPort = (com1, com2, com3);
  Tbaud = (B110, B300, B600, B1200, B2400, B4800, B9600,
           B14400, B19200, B38400, B56000, B57600, B115200,
           B128000, B256000);
  TParity = 0..4;
  TStopB =(B1, B1_5, B2);
var
  Port                : String;
  is_open             : boolean = false;
  com                 : Thandle;
  DCB, DCBBack        : TDCB;
  timeouts            : TCommTimeouts;
  Critical1           : TRTLCriticalSection;

function OpenS_Coms(SeriePort     : TPort;
                    buffersize    : Integer;
                    Baudrate      : Tbaud;
                    ParityEnable  : boolean;
                    ByteSize      : Integer;
                    Parity        : TParity;
                    StopBits      : TStopB ): boolean;

function CloseS_Coms : integer;
function ReadS_port(var Read_val : byte) : boolean;
procedure WriteS_port(Write_val : byte);

implementation

{*************************************************************}
{* This procedure closes communication port                  *}
{ It returns a 0 for a unsucsessfull closing attempt
{ or a 1 for a sucsessful attempt
{*************************************************************}

function CloseS_Coms : integer;
 begin
  if is_open then
  CloseS_Coms:=1;
  begin
    if not CloseHandle(com) then
       begin
            raise Excom.Create('Error closing com port');
            CloseS_Coms:=0;
       end;
    is_open := false;
  end;
 DeleteCriticalSection(Critical1);
end;
{*****************************************************************************}
{** Error message handling                                                   **}
{*****************************************************************************}

procedure error(s : string);

begin
 if not SetCommState(com, DCBBack) then ShowMessage('Error restoring settings on com port');
 CloseS_Coms;
 raise Excom.Create(s);
end;

{*****************************************************************************}

procedure Open_Com(Port : string);
```

```
Var
    ComPortSet : Boolean;
    ComNumber : Integer;
begin
  ComPortSet := false;
  ComNumber:=2;
While (Not ComPortSet) do
begin
  if is_open then closeHandle(com);
  com := CreateFile(PChar(Port), GENERIC_READ or GENERIC_WRITE,
           0, nil, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
  if com>0 then
    begin
       is_open := true;
       ComPortSet := true;
    end
    else
    begin
       if ComNumber<3 then inc(ComNumber)
       else
       begin
            ComPortSet := true;
            raise Excom.Create('Com_port_in_use_by_other_device');
       end;
       port := 'com'+IntToStr(ComNumber);
    end;
end;
end;
{*************************************************************************}

procedure set_buf(len_in, len_out: longint);

begin
  if not SetupComm(com, len_in, len_out) then
    error('Error_setting_com_port_buffer_size');
end;

{*************************************************************************}

procedure set_com(Rbaud : Dword; RParityE : Boolean;
                  RParity, RStopB : Byte;
                  RByteSize : integer);

begin
  if not GetCommState(com, DCB) then
    error('Error_getting_com_port_settings');
       DCBBack := DCB;
       DCB.DCBlength    := SizeOf(Tdcb);
       DCB.BaudRate     := RBaud;
       DCB.ByteSize     := RByteSize;
       DCB.Parity       := RParity;
       DCB.StopBits     := RStopB;
       DCB.wReserved    := 0;
   if not SetCommState(com, DCB) then
     error('_Error_setting_com_port');
end;

{*************************************************************************}

procedure set_com_timing;

{*************************************************************************}
{*If an application sets the ReadIntervalTimeout and readTotalTimeoutMultiplier
members to MAXDWORD and the ReadTotalTimeoutConstant member to a value greater
than zero and less than MAXDWORD, one of the following occurs when the readFile
function is called:Selecting a button typically causes the following events:

*  If there are any characters in the input buffer, ReadFile returns immediately
   with the characters in the buffer.
*  If there are no characters in the input buffer, ReadFile waits until a
   character arrives and then returns immediately.
*  If no character arrives within the time specified by the
```

```
   ReadTotalTimeoutConstant member, ReadFile times out.                    *}
{************************************************************************}

begin
  timeouts.ReadIntervalTimeout          := MAXDWORD;
  timeouts.ReadTotalTimeoutMultiplier   := MAXDWORD;
  timeouts.ReadTotalTimeoutConstant     := 50; {Prev =500}
  timeouts.WriteTotalTimeoutMultiplier  := 0;
  timeouts.WriteTotalTimeoutConstant    := 0;

  if not SetCommTimeouts(com, timeouts) then
    error('Error setting com port timing');
end;
{************************************************************************}
{ This function writes to the serie port
{ The Write_val value is writen to the port
{ The procedure returns a true for correct read
{ Procedure returns a false if the read time out / fails
{************************************************************************}

procedure WriteS_port(Write_val:byte);

var
  res: Dword;

begin
  EnterCriticalSection(Critical1);
  WriteFile(com, Write_val, 1, res, nil);
  LeaveCriticalSection(Critical1);
end;

{************************************************************************}
{ This function reads the serie port
{ The value is returned to the calling procedure in variable
{ Read_val
{ The procedure returns a true for correct read
{ Procedure returns a false if the read time out / fails
{************************************************************************}
function ReadS_port(var Read_val :byte):boolean;
var
  res: Dword;
begin
  ReadFile(com, Read_val, 1, res, nil);
  ReadS_port:=(res=1);
end;
{*********************************************************}
Procedure ClearReadBuf ;
var RB: byte;
begin
    while ReadS_port (RB) do;
end;
{*********************************************************}
{ This function opens the Serie port for communications
{*********************************************************}
{ The function call requires
{   Name          Type            Usefull values
{ 1. SeriePort    TPort           com1
{                                 com2
{                                 com3
{ 2. Buffersize   Integer         The size of the input and output buffers
{ 3. Baudrate     Tbaud           B110, B300, B600, B1200, B2400,
{                                 B4800, B9600, B14400, B19200, B38400,
{                                 B56000, B57600, B115200, B128000, B256000
{ 4. ParityEnable Boolean
{ 5. ByteSize     Integer
{ 6. Parity       Tparity         0 no Parity
{                                 1 odd Parity
{                                 2 even Parity
{ 7. Stopbits     TStop           0   1   stop bits
{                                 1   1.5 stop bits
{                                 2   2   stop bits
}
function OpenS_Coms(SeriePort     : TPort;
```

```
                    buffersize    : Integer;
                    Baudrate      : Tbaud;
                    ParityEnable  : boolean;
                    ByteSize      : Integer;
                    Parity        : TParity;
                    StopBits      : TStopb ): boolean;
var
   PParityE  : Boolean;
   PBaud  : Dword;
   PParity, PStopB  : Byte;
begin
InitializeCriticalSection(Critical1);
OpenS_coms:=true;

port:='com2';
Open_com(port);
set_buf(buffersize, buffersize);
case Baudrate of
     B110  : PBaud:=CBR_110;
     B300  : Pbaud:=CBR_300;
     B600  : Pbaud:=CBR_600;
     B1200 : Pbaud:=CBR_1200;
     B2400 : Pbaud:=CBR_2400;
     B4800 : Pbaud:=CBR_4800;
     B9600 : Pbaud:=CBR_9600;
     B38400  : Pbaud:=CBR_38400;
     else  Pbaud:=CBR_14400;
     end;
PParityE := ParityEnable;
Case Parity of
     1 : PParity:=1; {Odd Parity}
     2 : PParity:=2; {Even Parity}
     3 : PParity:=3; {Mark Parity}
     else  PParity:=0;{No Parity}
     end;

Case  StopBits of
      B1  : PStopB:=0;
      B1_5  : PStopB:=1;
      else  PStopB:=2;
      end;

set_com(Pbaud, PParityE,  PParity, PStopB, ByteSize);
set_com_timing;
{ClearReadBuf}
end;
{*********************************************************}
end.
```

Appendix D

Users Manual for Windows Interface

D.1 Hardware Installation

The Windows interface comes with a Radio link, including a Serial port between the Radio Link and PC, and a Modem.

Connect the Serial Port to Communication Port 1 of your PC and the Modem to Communication Port 2. This is possible, only if the PC consists of a PS2 Mouse Port. If this is not the case, contact your distributor.

Connect the other end of the Serial Port to your Radio Link. You are now ready to start.

D.2 Software Installation

STEP 1: To install the Software on the system, insert the Disk into Drive A.

STEP 2: Go to the Start Button and select the Run option.

STEP 3: Type A:\install.bat and press the Enter button.

STEP 4: The program will now be installed under the Folder RTC on the C Drive. Just run the Txtedit.exe file and the program will be started.

D.3 Use

STEP 1: Go to C:\RTC and double click on Txtedit.exe. This will start the Remote Text Communicator program.

STEP 2: Go to Edit in the Main Menu and select the Enter Units option. A window, that looks like Figure D.1 will now appear. This is where all the various SAU Users and their corresponding SAU User number can be entered.

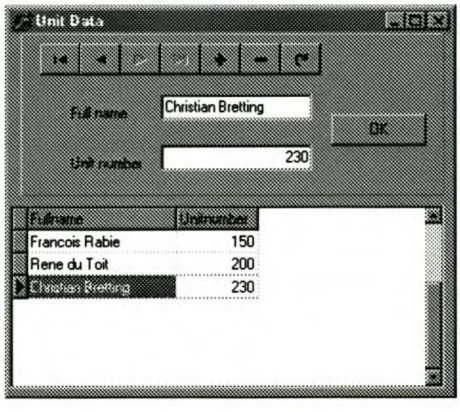

Figure D.1: Data Units

STEP 3: The Bar at the top of this new window, is called the Navigation Bar. The Buttons on the Navigation bar can be used to Enter, Edit, etc the data of the Table. By pointing the cursor to

each of these buttons individually and keeping it there for a couple of seconds, a description tag of the button will pop up. This will enable you to use the Navigation Bar without any problems.

STEP 4: After you have finished inserting all the data into the Units table, you can click on the OK button. This will close the Unit number table.

STEP 5: Now a new file can be created. Go to File in the Main Menu and select the New option. This will create a new file where you can type the information that you want to save.

STEP 6: After you have finished typing the file, go to File on the Main Menu and select the Save As... option. Type in a name for your file, remember to add the extension .txt to your file, and select the Save option.

STEP 7: Now you are ready to send the file you have just created. Go to File in the Main Menu and select the Send... option. A window that looks like Figure D.2, will now appear.

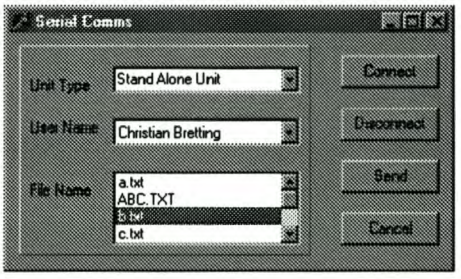

Figure D.2: Send... Window

STEP 8: In this window there are three boxes. These boxes are Unit Type, User Name and File Name. Click on the down arrow next to each of these boxes and make the required selection.

STEP 9: After you have made the necessary selections, you can click on the Connect button to connect to the communication port. As soon as you click this button the following message will appear:
Com port opened and connected.

STEP 10: Click OK and then select the Send button. Your file is now sent to the SAU User that you selected.

STEP 11: Click the Disconnect button. Now the following message will appear:
Com port disconnected.

STEP 12: Click OK. Now you can select the Cancel button to return to the program. A new file can now be created, or the existing file can be sent to a new user.

STEP 13: To exit the program, go to File in the Main menu and select the Exit option.

STEP 14: All these selections has a shortcut button just underneath the Main menu. If you point the cursor to one of these buttons and keep it there for a few seconds an information tag will appear. This will enable you to use the Shortcut buttons.

Appendix E

WD37C65B Floppy Disk Subsystem Controller

This Appendix was added to include the most important details of the WD37C65B for the reader. It was very difficult to acquire this document and therefore a simplified version is attached to make it easier for the reader.

E.1 Status Registers

Tables E.1 to E.4 are extracted directly out of Western Digital [1989].

BIT	NAME	SYMBOL	DESCRIPTION
D7 D6	Interrupt Code	IC	D7=0 and D6=0. Normal termination of command was completed and properly executed. D7=0 and D6=1. Abnormal termination of command, (AT). Execution of command was started but not successfully completed. D7=1 and D6=0. Invalid command issue. Command issued was never started.
D5	Seek End	SE	When the FDC completes the Seek command, this flag is set to 1(high).
D4	Equipment Check	EC	If the Track 0 signal fails to occur after 255 step pulses (Recalibrate Command), then this flag is set.
D3	Not Ready	NR	Since drive Ready is always presumed true, this will always be a logic 0.
D2	Head Select	HS	This flag is used to indicate the state of the head at the interrupt.
D1	Unit Select 1	US1	This flag is used to indicate a Drive Unit Number at interrupt.
D0	Unit Select 0	US0	This flag is used to indicate a Drive Unit Number at interrupt.

Table E.1: Status Register 0

*CRC - Cyclic Redundancy Check
**IDR - Internal Data Register
***C - Cylinder

BIT	NAME	SYMBOL	DESCRIPTION
D7	End of Cylinder	EN	When the FDC tries to access a sector beyond the final sector of a cylinder, this flag is set.
D6			Not used. This bit is always 0 (low).
D5	Data Error	DE	When the FDC detects a *CRC error in either the ID field or the data field, this flag is set.
D4	Overrun	OR	If the FDC is not serviced by the host system during data transfers within a certain time interval, this flag is set.
D3			Not used. This bit is always 0 (low).
D2	No Data	ND	During execution of READ DATA, WRITE DELETED DATA, or SCAN command, if the FDC cannot find the sector specified in the * *IDR Register, this flag is set. During execution of the READ ID command, if the FDC cannot read the ID field without an error, then this flag is set. During execution of the READ A TRACK command, if the starting sector cannot be found, then this flag is set.
D1	Not Writeable	NW	During execution of WRITE DATA, WRITE DELETED DATA or FORMAT A TRACK commands, if the FDC detects a $\overline{WP}$ signal from the FDD, then this flag is set.
D0	Missing Address Mark	MA	If the FDC cannot detect the ID Address Mark after encountering the index hole twice, then this flag is set. If the FDC cannot detect the Data Address Mark or Deleted Data Address Mark, this flag is set. At the same time the MD (Missing Address MArk in data field) of Status Register 2 is set.

Table E.2: Status Register 1

BIT	NAME	SYMBOL	DESCRIPTION
D7			Not used. This bit is always 0 (low).
D6	Control Mark	CM	During execution of the READ DATA or SCAN Command, if the FDC encounters a sector which contains a Deleted Data Address Mark, this flag is set.
D5	Data Error	DD	If the FDC detects a CRC error in the data field, then this flag is set.
D4	Wrong Cylinder	WC	This bit is related to the ND bit, and when the contents of ***C on the medium is different from that stored in the IDR, this flag is set.
D3	Scan Equal	SH	During execution of the SCAN dommand, if the condition of "equal" is satisfied, this flag is set.
D2	Scan Not	SN	During execution of the SCAN command, if the FDC cannot find a sector on the cylinder which meets the condition, then this flag is set.
D1	Bad Cylinder	BC	This bit is related to the ND bit, and when the contents of C on the medium is different from that stored in the IDR and the contents of C is FF, then this flag is set.
D0	Missing Address Mark in Data Field	MD	When data is read from the medium, if the FDC cannot find a Data Address Mark or Deleted Data Address Mark, then this flag is set.

Table E.3: Status Register 2

BIT	NAME	SYMBOL	DESCRIPTION
D7			Not used. This bit is always 0 (low).
D6	$\overline{WriteProtected}$	$\overline{WP}$	This bit is used to indicate the status of the $\overline{WriteProtected}$ signal from the FDD.
D5	Ready	RY	This bit will always be a logic 1. Drive is presumed to be ready.
D4	Track 0	T0	This bit is used to indicate the status of the Track 0 signal from the FDD.
D3	$\overline{WriteProtected}$	$\overline{WP}$	This bit is used to indicate the status of the $\overline{WriteProtected}$ signal from the FDD.
D2	Head Select	HS	This bit is used to indicate the status of the Side Select signal to the FDD.
D1	Unit Select 1	US1	This bit is used to indicate the status of the Unit Select 1 signal to the FDD.
D0	Unit Select 0	US0	This bit is used to indicate the status of the Unit Select 0 signal to the FDD.

Table E.4: Status Register 3

E.2 Commands

Tables E.5 to E.14 are all extracted from Western Digital [1989]. The descriptions of the commands are also extracted from Western Digital [1989]

PHASE	R/W	D7	D6	D5	D4	D3	D2	D1	D0	REMARKS
COMMAND	W	MT	MF	SK	0	0	1	1	0	Command Codes
	W	X	X	X	X	X	HS	US1	US0	
	W	←			C				→	Sector ID information prior to command execution. The four bytes are compared against header on floppy disk.
	W	←			H				→	
	W	←			R				→	
	W	←			N				→	
	W	←			EOT				→	
	W	←			GPL				→	
	W	←			DTL				→	
EXECUTION										Data transfer between FDD and main system.
RESULTS	R	←			ST0				→	Status information after command execution.
	R	←			ST1				→	
	R	←			ST2				→	
	R	←			C				→	Sector ID information after command execution.
	R	←			H				→	
	R	←			R				→	
	R	←			N				→	

Table E.5: Read Data

PHASE	R/W	D7	D6	D5	D4	D3	D2	D1	D0	REMARKS
COMMAND	W	MT	MF	0	0	0	1	0	1	Command Codes
	W	X	X	X	X	X	HS	US1	US0	
	W	←			C				→	Sector ID information prior to command execution. The four bytes are compared against header on floppy disk.
	W	←			H				→	
	W	←			R				→	
	W	←			N				→	
	W	←			EOT				→	
	W	←			GPL				→	
	W	←			DTL				→	
EXECUTION										Data transfer between FDD and main system.
RESULTS	R	←			ST0				→	Status information after command execution.
	R	←			ST1				→	
	R	←			ST2				→	
	R	←			C				→	Sector ID information after command execution.
	R	←			H				→	
	R	←			R				→	
	R	←			N				→	

Table E.6: Write Data

PHASE	R/W	D7	D6	D5	D4	D3	D2	D1	D0	REMARKS
COMMAND	W	0	MF	0	0	1	0	1	0	Command Codes
	W	X	X	X	X	X	HS	US1	US0	
EXECUTION										The first correct ID information on the cylinder is stored in Data Register.
RESULTS	R	←			ST0				→	Status information after command execution.
	R	←			ST1				→	
	R	←			ST2				→	
	R	←			C				→	Sector ID information read during Execution Phase from floppy disk.
	R	←			H				→	
	R	←			R				→	
	R	←			N				→	

Table E.7: Read ID

PHASE	R/W	D7	D6	D5	D4	D3	D2	D1	D0	REMARKS
COMMAND	W	0	MF	0	0	1	1	0	1	Command Codes
	W	X	X	X	X	X	HS	US1	US0	
	W	←			N				→	Bytes/Sector
	W	←			SC				→	Sectors/Track
	W	←			GPL				→	Gap 3
	W	←			D				→	Filler Byte
EXECUTION										Floppy Disk Controller (FDC) formats an entire track.
RESULTS	R	←			ST0				→	Status information after command execution.
	R	←			ST1				→	
	R	←			ST2				→	
	R	←			C				→	In this case, the ID information has no meaning.
	R	←			H				→	
	R	←			R				→	
	R	←			N				→	

Table E.8: Format A Track

PHASE	R/W	D7	D6	D5	D4	D3	D2	D1	D0	REMARKS
COMMAND	W	0	0	0	0	0	1	1	1	Command Codes
	W	X	X	X	X	X	0	US1	US0	
EXECUTION										Head retracted to Track zero.

Table E.9: Recalibrate

PHASE	R/W	D7	D6	D5	D4	D3	D2	D1	D0	REMARKS
COMMAND	W	0	0	0	0	1	0	0	0	Command Codes
RESULTS	R	←			ST0				→	Status information about the FDC at the
	R	←			PCN				→	end of seek operation.

Table E.10: Sense Interrupt Status

PHASE	R/W	D7	D6	D5	D4	D3	D2	D1	D0	REMARKS
COMMAND	W	0	0	0	0	0	0	1	1	Command Codes
	W	←	SRT	→		←		HUT	→	
	W	←		HLT					→	ND

Table E.11: Specify

PHASE	R/W	D7	D6	D5	D4	D3	D2	D1	D0	REMARKS
COMMAND	W	0	0	0	0	0	1	0	0	Command Codes
	W	X	X	X	X	X	HS	US1	US0	
RESULTS	R	←			ST3				→	Status information about the FDC

Table E.12: Sense Drive Status

PHASE	R/W	D7	D6	D5	D4	D3	D2	D1	D0	REMARKS
COMMAND	W	0	0	0	0	1	1	1	1	Command Codes
	W	X	X	X	X	X	HS	US1	US0	
	W	←			NCN				→	
EXECUTION										Head is positioned over proper cylinder on the diskette.

Table E.13: Seek

SYMBOL	NAME	DESCRIPTION
A0	ADDRESS LINE 0	A0 controls selection of Main Status Register (A0=0) or Data Register (A0=1).
C	CYLINDER NUMBER	C stands for the current/selected cylinder (track) numbers 0 through 255 of the medium.
D	DATA	D stands for the data pattern which is going to be written into a sector.
D7 - D0	DATA BUS	8-bit DATA BUS, where D7 stands for a most significant bit, and D0 stands for a least significant bit.
DTL	DATA LENGTH	When N is defined as 00, DTL stands for the DATA LENGTH which users are going to read out or write into the sector.
EOT	END OF TRACK	EOT stands for the final sector number on a cylinder. During Read or Write operations, FDC will stop data transfer after a sector number equal to EOT.
GPL	GAP LENGTH	GPL stands for the length of Gap 3. During the FORMAT Command, it determines the size of Gap 3.
H	HEAD ADDRESS	H stands for head number 0 or 1, as specified in the ID field.
HLT	HEAD LOAD TIME	HLT stands for the HEAD LOAD TIME in FDD (2 to 254ms in 2ms increments).
HS	HEAD SELECT	HS stands for a selected head number 0 or 1 and controls the polarity of pin 25 (in 40 pin DIP) or pin 28 (in 44 pin PLCC).
HUT	HEAD UNLOAD TIME	HUT stands for the HEAD UNLOAD TIME after a Read or Write operation has occurred (16 to 240ms in 16ms increments).
MF	FM or MFM	If MF is low, FM mode is selected. If it is high, MFM mode is selected.
MT	MULTITRACK	If MT is high, a MULTITRACK operation is performed. If MT=1 after finishing Read/Write operation on side 0, FDC will automatically start searching for sector 1 on side 1.
N	NUMBER	N stands for the NUMBER of data bytes written in a sector.
NCN	NEW CYLINDER NUMBER	NCN stands for a NEW CYLINDER NUMBER which is going to be reached as a result of the Seek operation. Desired position of head.
ND	NON-DMA MODE	ND stands for operation in the NON-DMA MODE.
PCN	PRESENT CYLINDER	PCN stands for the cylinder number at the completion of the SENSE INTERRUPT STATUS Command. Position of head at present time.
R	RECORD	R stands for the sector number which will be read or written.
R/W	READ/WRITE	R/W stands for either READ or WRITE signal.
SC	SECTOR	SC indicates the number of sectors per cylinder.
SK	SKIP	SK stands for SKIP Deleted Data Address mark.
SRT	STEP RATE TIME	SRT stands for the Stepping Rate for the FDD (1 to 16ms in 1ms increments). Stepping Rate applies to all drives. In 2's complement format, F(Hex)=1ms, E(Hex)=2ms, etc.
ST0 ST1 ST2 ST3	STATUS 0 STATUS 1 STATUS 2 STATUS 3	ST0 – 3 stands for one of four registers which store the STATUS information after a command has been executed. This information is available during the result phase after command execution. These registers should not be confused with the Main Status Register (selected by A0=0). ST0 – 3 may be read only after a command has been executed and contains information relevant to that particular command.
STP		During a SCAN operation, if STP=1, the data in contiguous sectors is compared byte by byte with data sent from the processor (or DMA); if STP=2, then alternate sectors are read and compared.
US0,US1	UNIT SELECT	US stands for a selected drive; binary encoded, 1 of 4.

Table E.14: Command Symbol Descriptions

Read Data

A set of nine byte words are required to place the FDC into the Read Data Mode. After the Read Data command has been issued, the FDC loads the head (if it is in the unloaded state), waits the specified head settling time (defined in the Specify Command), and begins reading ID Address Marks and ID fields. When the current sector number ("R") stored in the ID Register (IDR) compares with the sector number read off the diskette, then the FDC outputs data (from the data field) byte-to-byte to the main system via the data bus.

After completion of the read operation from the current sector, the Sector Number is incremented by one, and the data from the next sector is read and output on the data bus. This continuous read function is called a "Multi-sector Read Operation." The Read Data Command may be terminated by the receipt of a Terminal Count signal. TC should be issued at the same time that the DACK for the last byte of data is sent. Upon receipt of this signal, the FDC stops outputting data to the processor, but will continue to read data from the current sector, check CRC (Cyclic Redundancy Count) bytes, and then at the end of the sector terminate the Read Data command. The amount of data which can be handled with a single command to the FDC depends upon MT (multitrack), MF (MFM/FM), and N (number of bytes/sector). Table 31 lists the Transfer Capacity.

TABLE 31. TRANSFER CAPACITY

Multi-Track MT	MFM/FM MF	Bytes/Sector N	Maximum Transfer Capacity (Bytes/Sector) (Number of Sectors)	Final Sector Read from Diskettes
0	0	00	(128) (26) = 3,328	26 at Side 0
0	1	01	(256) (26) = 6,656	or 26 at Side 1
1	0	00	(128) (52) = 6,656	26 at Side 1
1	1	01	(256) (52) = 13,312	
0	0	01	(256) (15) = 3,840	15 at Side 0
0	1	02	(512) (15) = 7,680	or 15 at Side 1
1	0	01	(256) (30) = 7,680	15 at Side 1
1	1	02	(512) (30) = 15,360	
0	0	02	(512) (8) = 4,096	8 at Side 0
0	1	03	(1024) (8) = 8,192	or 8 at Side 1
1	0	02	(512) (16) = 8,192	8 at Side 1
1	1	03	(1024) (16) = 16,384	

The "multi-track" function (MT) allows the FDC to read data from both sides of the diskette. For a particular cylinder, data will be transferred starting at Sector 1, Side 0 and completing at Sector L, Side 1 (Sector L = last sector on the side). Note, this function pertains to only one cylinder (the same track) on each side of the diskette.

When N = 0, then DTL defines the data length which the FDC must treat as a sector. If DTL is smaller than the actual data length in a sector, the data beyond DTL in the sector is not sent to the Data Bus. The FDC reads (internally) the complete sector performing the CRC check, and depending upon the manner of command termination, may perform a Multi-Sector Read operation. When N is non-zero, then DTL has no meaning and should be set to FF hexidecimal.

At the completion of the Read Data command, the head is not unloaded until after Head Unload Time Interval (specified in the Specify command) has elapsed. If the processor issues another command before the head unloads, then the head settling time may be saved between subsequent reads. This time out is particularly valuable when a diskette is copied from one drive to another.

If the FDC detects the Index Hole twice without finding the right sector, (indicated in 'R'), then the FDC sets the ND (No Data) flag in Status Register 1 to a 1 (high), and terminates the Read Data command. (Status Register 0 also has bits 7 and 6 set to 0 and 1 respectively.)

After reading the ID and Data Fields in each sector, the FDC checks the CRC bytes. If a read error is detected (incorrect CRC in ID field), the FDC sets the DE (Data Error) flag in Status Register 1 to 1 (high). If a CRC error occurs in the Data Field, the FDC also sets the DD (Data Error in Data Field) flag in Status Register 2 to a 1 (high), and terminates the Read Data command. (Status Register 0 also has bits 7 and 6 set to 0 and 1 respectively.)

If the FDC reads a Deleted Data Address Mark off the diskette, and the SK bit (bit D5 in the first Command Word) is not set (SK = 0), then the FDC sets the CM (Control Mark) flag in Status Register 2 to a 1 (high), and terminates the Read Data command, after reading all the data in the sector. If SK = 1, the FDC skips the sector with the Deleted Data Address Mark and reads the next sector. The CRC bits in the deleted data field are not checked when SK = 1.

During disk data transfers between the FDC and the processor, via the data bus, the FDC must be serviced by the processor every 27 μs in the FM mode, and every 13 μs in the MFM mode, or the FDC sets the OR (Overrun) flag in Status Register 1 to a 1 (high), and terminates the Read Data command.

If the processor terminates a read (or write) operation in the FDC, then the ID information in the Result phase is dependent upon the state of the MT bit and EOT byte. Table 32 shows the values for C, H, R, and N, when the processor terminates the command.

TABLE 32. C, H, R, AND N VALUES

MT	HD	Final Sector Transferred to Processor	ID Information at Result Phase C	H	R	N
0	0	Less than EOT	NC	NC	R+1	NC
	0	Equal to EOT	C+1	NC	R=01	NC
	1	Less than EOT	NC	NC	R+1	NC
	1	Equal to EOT	C+1	NC	R=01	NC
1	0	Less than EOT	NC	NC	R+1	NC
	0	Equal to EOT	NC	LSB	R=01	NC
	1	Less than EOT	NC	NC	R+1	NC
	1	Equal to EOT	C+1	LSB	R=01	NC

Notes: NC (No Change): The same value as the one at the beginning of command execution. LSB (Least Significant Bit): The least significant bit of H is complemented.

Write Data

A set of nine bytes is required to set the FDC into the Write Data mode. After the Write Data command has been issued the FDC loads the head (if it is in the unloaded state), waits the specified head settling time (defined in the Specify command), and begins reading ID fields. When all four bytes loaded during the command (C, H, R, N) match the four bytes of the ID field from the diskette, the FDC takes data from the processor byte-by-byte via the data bus and outputs it to the FDD.

After writing data into the current sector, the sector number stored in 'R' is incremented by one, and the next data field is written into. The FDC continues this 'Multisector Write Operation' until the issuance of a Terminal Count signal. If a Terminal Count signal is sent to the FDC it continues writing into the current sector to complete the data field. If the Terminal Count signal is received while a data field is being written, then the remainder of the data field is filled with zeros.

The FDC reads the ID field of each sector and checks the CRC bytes. If the FDC detects a read error (CRC error) in one of the ID fields, it sets the DE (Data Error) flag of Status Register 1 to a 1 (high) and terminates the Write Data command. (Status Register 0 also has bits 7 and 6 set to 0 and 1 respectively.)

The Write command operates in much the same manner as the Read command. The following items are the same, and one should refer to the Read Data command for details:

- Transfer capacity
- EN (End of Cylinder) flag
- ND (No Data) flag
- Head Unload Time interval
- ID Information when the processor terminates command
- Definition of DTL when N = 0 and when N ≠ 0

In the Write Data mode, data transfers between the processor and FDC via the data bus, must occur every 27 μs in the FM mode and every 13 μs in the MFM mode. If the time interval between data transfers is longer than this, then the FDC sets the OR (Overrun) flag in Status Register 1 to a 1 (high) and terminates the Write Data command. (Status Register 0 also has bits 7 and 6 set to 0 and 1 respectively.)

Format A Track

The Format command allows an entire track to be formatted. After the index hole is detected, data is written on the diskette; Gaps, Address marks, ID fields and data fields, all per the IBM System 34 (double density) or System 3740 (single density) format are recorded. The particular format which will be written is controlled by the values programmed into N (number of bytes/sector), SC (sectors/cylinder), GPL (gap length), and D (data pattern) which are supplied by the processor during the Command phase. The data field is filled with the byte of data stored in D. The ID field for each sector is supplied by the processor; that is, four data requests per sector are made by the FDC for C (cylinder number), H (head number), R (sector number) and N (number of bytes/sector). This allows the diskette to be formatted with nonsequential sector numbers, if desired.

The processor must send new values for C, H, R, and N to the WD37C65C for each sector on the track. If FDC is set for the DMA mode, it will issue four DMA requests per sector. If it is set for the Interrupt mode, it will issue four interrupts per sector and the processor must supply C, H, R, and N loads for each sector. The contents of the R register are incremented by 1 after each sector is formatted; thus, the R register contains a value of R when it is read during the Result phase. This incrementing and formatting continues for the whole track until the FDC detects the index hole for the second time, whereupon it terminates the command.

Table 33 shows the relationship between N, SC, and GPL for various sector sizes.

TABLE 33. N, SC AND GPL RELATIONSHIP

Format	Sector Size	N	SC	GPL[1]	GPL[2] [3]
	8" Standard Floppy				
FM Mode	128 bytes/sector	00	1A	07	1B
	256	01	0F	0E	2A
	512	02	08	1B	3A
	1024	03	04	47	8A
	2048	04	02	C8	FF
	4096	05	01	C8	FF
MFM Mode[4]	256	01	1A	0E	36
	512	02	0F	1B	54
	1024	03	08	35	74
	2048	04	04	99	FF
	4096	05	02	C8	FF
	8192	06	01	C8	FF
	5¼" Minifloppy				
FM Mode	128 bytes/sector	00	12	07	09
	128	00	10	10	19
	256	01	08	18	30
	512	02	04	46	87
	1024	03	02	C8	FF
	2048	04	01	C8	FF
MFM Mode[4]	256	01	12	0A	0C
	256	01	10	20	32
	512	02	08	2A	50
	1024	03	04	80	F0
	2048	04	02	C8	FF
	4096	05	01	C8	FF
	3½" Sony Microfloppy				
FM Mode	128 bytes/sector	0	0F	07	1B
	256	1	09	0E	2A
	512	2	05	1B	3A
MFM Mode[4]	256	1	0F	0E	36
	512	2	09	1B	54
	1024	3	05	35	74

Notes: [1] Suggested values of GPL in Read or Write commands to avoid splice point between data field and ID field of contiguous sections.
[2] Suggested values of GPL in format command.
[3] All values except sector size are hexidecimal.
[4] In MFM mode FDC cannot perform a Read/Write/format operation with 128 bytes/sector. (N – 00)

Seek

The Read/Write head within the FDD is moved from cylinder to cylinder under control of the Seek command. FDC has four independent Present Cylinder Registers for each drive. They are cleared only after the Recalibrate command. The FDC compares the PCN (Present Cylinder Number) which is the current head position with the NCN (New Cylinder Number), and if there is a difference, performs the following operations:

PCN < NCN: Direction signal to FDD set to a 1 (high), and step pulses are issued. (Step In)

PCN > NCN: Direction signal to FDD set to a 0 (low), and step pulses are issued. (Step Out)

The rate at which step pulses are issued is controlled by SRT (Stepping Rate Time) in the Specify command. After each step pulse is issued NCN is compared against PCN, and when NCN = PCN, the SE (Seek End) flag is set in Status Register 0 to a 1 (high), and the command is terminated. At this point FDC interrupt goes high. Bits D_0B–D_3B in the Main Status Register are set during the Seek operation and are cleared by the Sense Interrupt Status command.

During the command phase of the Seek operation the FDC is in the FDC Busy state; but during the Execution phase, it is in the non-busy state. While the FDC is in the non-busy state, another Seek command may be issued, and in this manner parallel Seek operations may be done on up to four drives at once. No other command can be issued for as long as the FDC is in the process of sending step pulses to any drive.

If the time to write three bytes of Seek command exceeds 150μs, the timing between the first two step pulses may be shorter than set in the Specify command by as much as 1ms.

Recalibrate

The function of this command is to retract the Read/Write head within the FDD to the Track 0 position. The FDC clears the contents of the PCN counter and checks the status of the Track 0 signal from the FDD. As long as the Track 0 signal is low, the Direction signal remains 0 (low) and step pulses are issued. When the Track 0 signal goes high, the SE (Seek End) flag in Status Register 0 is set to a 1 (high) and the command is terminated. If the Track 0 signal is still low after 255 step pulses have been issued, (for the WD37C65 and the WD37C65A) or 77 step pulses (WD37C65B), the FDC sets the SE (Seek End) and EC (Equipment Check) flags of Status Register 0 to both 1s (highs), and terminates the command after bits 7 and 6 of Status Register 0 are set to 0 and 1 respectively.

The ability to do overlap Recalibrate commands to multiple FDDs and the loss of the Ready signal, as described in the Seek command, also applies to the Recalibrate command.

Sense Interrupt Status

An interrupt signal is generated by the FDC for one of the following reasons:

1. Upon entering the Result phase of:
   a. Read Data command
   b. Read A Track command
   c. Read ID command
   d. Read Deleted Data command
   e. Write Data command
   f. Format A Cylinder command
   g. Write Deleted Data command
   h. Scan commands
2. Ready Line of FDD changes state
3. End of Seek or Recalibrate command
4. During Execution phase in the non-DMA mode

Interrupts caused by reasons 1 and 4 above occur during normal command operations and are easily discernible by the processor. During an Execution phase in non-DMA mode, DB5 in the Main Status Register is high. Upon entering the Result phase, this bit gets cleared. Reasons 1 and 4 do not require Sense Interrupt Status commands. The interrupt is cleared by Reading/Writing data to the FDC. Interrupts caused by reasons 2 and 3 above may be uniquely identified with the aid of the Sense Interrupt Status command. This command, when issued, resets the Interrupt signal and via bits 5, 6, and 7 of Status Register 0 identifies the cause of the interrupt.

TABLE 31. INTERRUPT CAUSE

Seek End	Interrupt Code		
Bit 5	Bit 6	Bit 7	Cause
0	1	1	Ready Line changed state, either polarity
1	0	0	Normal Termination of Seek or Recalibrate command
1	1	0	Abnormal Termination of Seek or Recalibrate command

The Sense Interrupt Status command is used in conjunction with the Seek and Recalibrate commands which have no Result phase. When the disk drive has reached the desired head position, the WD37C65/A/B will set the interrupt line true. The host CPU must then issue a Sense Interrupt Status command to determine the actual cause of the interrupt, which could be Seek End or a change in ready status from one of the drives. See Figure 7.

The Specify command sets the initial values for each of the three internal timers. The HUT (Head Unload Time) defines the time from the end of the Execution phase of one of the Read/Write commands to the head unload state. This timer is programmable from 16 to 240ms in increments of 16ms (01 = 16ms, 02 = 32 ms . . . $0F_{16}$ = 240ms). The SRT (Step Rate Time) defines the time interval between adjacent step pulses. This timer is programmable from 1 to 16 ms in increments of 1 ms (F = 1ms, E = 2ms, D = 3ms, etc.). The HLT (Head Load Time) defines the time between when the Head Load signal goes high and the Read/Write operation starts. This timer is programmable from 2 to 254 ms in increments of 2 ms (01 = 2ms, 02 = 4ms, 03 = 6ms . . . 7F = 254ms).

The time intervals mentioned above are a direct function of the clock (CLK on pin 23). Times indicated above are for a 16MHz clock; if the clock was reduced to 8MHz, then all time intervals are increased by a factor of 2. If the clock was increased to 32 MHz, then all time intervals are decreased by half.

The choice of DMA or non-DMA operation is made by the ND (Non-DMA) bit. When this bit is high (ND = 1), the Non-DMA mode is selected; and when ND = 0, the DMA mode is selected.

Sense Drive Status

This command may be used by the processor to obtain the status of the FDDs. Status Register 3 contains the Drive Status information stored internally in FDC registers.

Bibliography

Halsall, F. [1992]. *Data Communications, Computer Networks and Open Systems*, third edn, Addison-Wesley Publishing Company.

Horowitz, P. & Hill, W. [1995]. *The art of electronics*, second (low price) edn, Cambridge University Press.

Kozierok, C. M. [1999]. The PC Guide: Floppy Disk Drives.
URL: `http://www.pcguide.com/ref/fdd/index.htm`

Pahlavan, K. & Levesque, A. H. [1995]. *Wireless Information Networks*, Wiley-Interscience.

Powertip [n.d.]. *Powertip Character LCDs*, Powertip Technology Corporation, Costa Mesa, California.

Prasad, R. [1996]. *CDMA for Wireless Personal Communications*, Artech House.

Radiometrix [1998]. *Radio Packet Controller*, Radiometrix Ltd, Hertfordshire, England.

Rappaport, T. S. [1996]. *Wireless Communications: Principles and Practices*, Prentice Hall PTR.

van der Merwe, D. [1998]. Delphi software program: Seriecoms.

Western Digital [1989]. *WD37C65/A/B Floppy Disk Subsystem Controller*, Western Digital Corporation, Irvine, California.