A PC-BASED BIT ERROR RATE ANALYSER

FOR A 2 Mbps DATA LINK

by

GWAIN BAYLEY

B.Sc. (Electrical) Engineering (Cape Town)

Submitted to the University of Cape Town in

Fulfillment of the requirements for the degree of

Master of Science in Engineering

SEPTEMBER 1988

ABSTRACT

The CCITT recommended 2 Mbps data link standard is becoming increasingly more common especially in ISDN applications. This thesis covers the development of a personal computer based bit error rate tester designed specifically for a link at this data rate but easily alterable and flexible enough to operate from 64 kbps to 8 Mbps.

The theory of pseudo random bit sequence generation is covered and the characteristics of CCITT recommended sequences are compared to those of other sequences which are better suited to testing links which do not implement a signal encoding technique.

The test unit is developed as an extension card to a personal computer. High level menu windows are used to control the operation of the link with a full screen of statistical information regarding the bit errors counted on both sides of the link when testing in full duplex.

In addition a feature is implemented whereby an operator can change the operating parameters of the test unit on either side of the link from the same PC. This is achieved through an RS232 communications channel connecting the PCs at each side of the link.

ACKNOWLEDGEMENTS

I would like to thank the following people and organisations for their guidance and support during the duration of this thesis project:

Mr. M.J. Ventura of UCT for supervising this thesis.

Plessey SA for originally proposing the thesis topic and giving valuable support all the way through the project.

The CSIR for financial assistance during the project.

Mr. Graham Jack for giving general advice whenever needed.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

GLOSSARY OF TERMS

1. AMI :	Alternate Mark Inversion coding.
2. BAUD :	Signalling rate.
3. BER :	Bit Error Rate.
4. BIOS :	Basic Input/Output System (in ROM).
5. CCITT :	International Consultative Committee for Telegraphy and Telephony.
6. CEPT :	Conference of European Postal and Telegraph Administration.
7. CPU :	Central Processing Unit.
8. DOS :	Disk Operating System (by Microsoft).
9. ECL :	Emitter Coupled Logic.
10. EXNOR :	Exclusive NOR (gate or operator).
11. FSK :	Frequency Shift Keying.
12. FSR :	Feedback Shift Register.
13. HDB3 :	High Density Bipolar three code.
14. I/O :	Input / Output.
15. IRQ :	Interrupt Request.
16. ISDN :	Integrated Services Digital Network.
17. ISR :	Interrupt Service Routine.
18. kbps :	Kilobits per second.
19. Mbps :	Megabits per second.
20. NRZ :	Non Return to Zero code.
21. PABX :	Private Automatic Branch Exchange.
22. PC :	Personal Computer.

23. PCM : Pulse Code Modulation.
24. ppm : Parts per million.
25. PRBS : Pseudo Random Bit Stream.
26. PSK : Phase Shift Keying.
27. QAM : Quadrature Amplitude Modulation.
28. ROM : Read Only Memory.
29. TDM : Time Division Multiplex.
30. TTL : Transistor - Transistor Logic.
31. VLSI : Very Large Scale Integration.

CHAPTER 1

INTRODUCTION

In the field of modern electronic communication systems an increasingly common and widespread communications link is the European CEPT1 standard [1], at 2048 kbps, and the North American standard, the Bell systems' T1 digital carrier [2], at 1544 kbps. These links are often used to interconnect pairs of digital PABXs carrying 30 (in the case of CEPT1) and 24 (in the case of the T1 carrier) voice grade lines.

CEPT links can also be composed of multiplexed data sources and can also be linked to a PCM highway in a general Integrated Services Digital Network [3].

In the instance of a dedicated data link, such as a link between two digital PABXs, the cost of a commercially available bit error rate analyzer to test the integrity of the 2 Mbps link is of the order of R 50 000.00. Thus, the concept of a low cost bit error rate analyser, as an extension card to a personal computer for use on either or both ends of the link, is a very useful one. This is especially so because the personal computers involved can be utilised for other applications most of the time and only

when testing of the link is necessary do they need to become available.

The subject of this thesis is the development of a low cost bit error rate analyzer to be used primarily with a 2Mbps microwave data link and designed to reside in a personal computer and operate as part of a personal computer system.

The purpose of this thesis is to report on the various stages of the development, from theoretical background investigation to production and testing of a unit.

The cost of the finished unit, which includes two personal computers (one for either side of the link), is of the order of a tenth of the cost of a commercially available unit; excluding the PCs the cost would be more of the order of one 50th of the cost of a comparable commercial unit. The unit developed has only one frequency of operation compared to the commercially available unit's range of 10 often used frequencies, but the flexibility of both readout and general use is greater in the case of the developed unit.

A further advantage of a personal computer based system is in the instance of long term monitoring or logging (to disk or printer) of the link performance with respect to weather and seasonal variations in the operating environment.

The unit developed was intended to analyse the bit error rate of a link conforming to the 2048 kbps CEPT1 standard as defined by the International Consultative Committee for Telegraphy and Telephony (CCITT) recommendation G.703 [4]. The link standard does not impose a constraint on the technique for analysis of the link errors but in a seperate volume the CCITT recommend specific pseudo random bit sequences for testing links as listed in Feher [5].

The development of the test unit was initiated because of the concurrent development, by a local telecommunications company, of a microwave link according to the 2 Mbps CCITT recommendation, and intended for use by the department of Posts and Telecommunications in South Africa. There was thus a need and a potential market for the development of a device designed to test the above mentioned link.

The unit was, however, developed with flexibility in mind and conversion to the T1 standard is easy to achieve. Furthermore, it can also be modified to work at the CCITT 8 Mbps standard, or indeed any of the North American standards between 64 kbps and 6.312 Mbps.

To further increase the flexibility of the unit, a design with different pseudo random bit sequences to the CCITT recommended sequences was also implemented, for use on more

general links where a signal encoding scheme is not implemented and the bit stream requirements for testing could possibly be different.

The completed unit also includes an RS232 based communications link for conveying commands and information from one side of the link to the other, so that the performance of the link in both directions can be monitored on either side of the link. The working parameters of either side of the link can also be changed from either end, facilitating the use of only one operator.

This thesis report firstly compares, in chapter 2, the cost and capabilities of commercially available test equipment with the unit developed.

Secondly, in chapter 3, the theoretical modelling of pseudo random bit streams for testing purposes is covered. Pseudo random sequences are also compared to true random sequences in this section. The algebraic background to the modelling and the proof that the model generates pseudo random solutions are presented in Appendix A.

Chapter 4 compares the CCITT recommended sequences to other possible sequences in theoretical as well as practical and empirical terms, including the use of a pseudo random

sequence generator output simulation program (which is listed along with output material in Appendix B).

Chapter 5 covers the general design of the bit error rate analyser in terms of system hardware considerations and system software considerations.

The development of the practical circuit is then covered in several sections. Chapter 6 begins with interfacing the extension board to a personal computer bus. This is followed by the clock generation and pseudo random sequence generation circuitry. Reception of the incoming bit stream and synchronisation of the incoming sequence with the standard sequence, with suitable timing diagrams, is covered next. The circuitry for comparison of the incoming sequence to the standard sequence and error counting follows. Lastly, the circuit design for interfacing to the data link as a TTL signal or the CCITT recommended HDB3 encoded signal is described.

The software development is covered in chapter 7 and starts with a small section describing the compilers used, extra libraries referenced, and inter-language interfaces. Covered next is the development of the software modules (user interfacing routines, across-link communications routines,

controlling routines and hardware interrupt servicing routines).

A report on the practical application of the unit is included in chapter 8, followed by methods of testing the device in the laboratory, results obtained in practice and discussion thereof.

The conclusions reached from the thesis project follow in chapter 9.

The appendices contain all the algebraic background, circuit diagrams, complete listings of all the commented source code modules, program output files and logging results, simulation program source code and output and a short manual on the use of the unit.

CHAPTER 2

FUNCTIONAL SPECIFICATIONS

The optimum features required in an instrument intended for error analysis are set forth in Feher [6] in two sections, namely, pattern generator features and error detection features.

2.1. PATTERN GENERATOR FEATURES

a) At least two data bit rates corresponding to recommended TDM levels in European or North American digital Hierarchy.

b) Clock jitter capabilities to investigate jitter tolerance.

c) Pseudo Random bit sequences as recommended by CCITT rec O.151 [7] which are important for the simulation of live digital data on the link.

d) Other optional patterns for deeper investigation into error trends.

e) Data outputs in accordance with CCITT rec G.703 [8].

2.2. ERROR DETECTION FEATURES

a) Data inputs in accordance with CCITT rec G.703 [9].

b) Clock recovery at the two data rates which has maximum tolerance to input jitter.

c) Reference pattern generation with automatic and manual sequence synchronisation.

d) Error counter averaging periods of 10^6 to 10^{10} bits and 1 sec, 10 sec, 1 min or continuous time.

e) Minimal dead time on repetitive measurements and zero dead time on continuous gating to detect and count all the errors occurring in the system under test.

f) Various displays or means of analysing the detected errors in terms of total error count, bit error rate, number of seconds containing errors, error free seconds, percentage of error free seconds, and percentage of seconds for which the bit error rate (BER) is below a presettable threshold [6].

g) Optional error burst analysis capabilities for detailed investigation into error occurrence and link behaviour.

2.3. MARKET TRENDS

Currently available bit error rate testers were found to have the following common features:

a) Frequency range from 64 kbps up to a maximum of 144 Mbps switchable between preset values of recommended data rates.

b) Switchable length of pseudo random sequence generated generally between the two recommended CCITT values (summarised in table 5.1).

c) Transmit clock jitter capabilities.

d) Cost ranging from R40 000 upwards at the time of investigation.

2.4. DESIGN CONSIDERATIONS

The unit being developed is intended specifically for a 2 Mbps data link offering an option of 8 Mbps. There should be no reason why the unit should not be flexible enough to accommodate a change in data rate. The lower limit of 64 kbps poses no problem but using the fastest TTL technology available the upper limit on the frequency of operation will give a maximum data rate of 34 Mbps.

The CCITT recommended sequence configuration is the same for 2 and 8 Mbps and thus as a basic unit only one configuration is needed although a possibility of another configuration should be provided.

Clock jitter can be provided as an option in that the basic unit can be developed using a conventional oscillator for the clock. If clock jitter capabilities are found to be a necessity they can be included as a module later by replacing the oscillator section (with an external separate board if necessary).

The minimum requirements taken from Feher's list above and deemed to be useful to the specific problem at hand were decided to be as follows.

In terms of pattern generator features:

a) To generate data streams at the required rate of 2 Mbps, with an option of easily converting to 8 Mbps.

b) To generate the CCITT recommended PRBS pattern with additional optional patterns.

c) To provide data outputs in accordance with CCITT rec G.703 [10].

In terms of error detection features:

a) To provide data inputs in accordance with CCITT rec G.703 [11].

b) To provide clock recovery at the required bit rate.

c) To provide reference pattern generation with automatic sequence synchronisation recovery.

d) To provide averaging periods at least as wide as described by Feher above with a basic averaging period of 2.10^6 bits, being 1 sec of real time at 2 Mbps data rate.

e) To have minimal dead time on all measurements.

f) To provide at least all the formats and displays as described by Feher above and to display the relevant information for all formats at the same time (which very few error analysers available can do).

In addition to the features which are taken from the list set out by Feher above, it was decided to implement the following:

a) Logging of data to disk or printer for long term monitoring to facilitate evaluation of the link performance with respect to environmental changes.

b) The completed unit also includes an RS232 based [12] communications link for conveying commands and information from one side of the link to the other, so that the performance of the link in both directions can be monitored on either side of the link. The working parameters of either side of the link can also be changed from either end, facilitating the use of only one operator.

c) A design that would cost a tenth of the cost of a commercially available unit.

CHAPTER 3

PSEUDO RANDOM SEQUENCES

CCITT recommendation O.151 [13] specifies two pseudo random bit sequence patterns to test digital transmission systems. These are based on 15 stage and 23 stage shift registers and are summarised according to the intended bit rate of the system to be tested in figure 3.1.

COMS BIT RATE	SEQ LENGTH	SR POLYNOMIAL	FREQ SPACING
1.544 Mbps	2^{15}-1 bits	x^{15}-x^{14}-1	47.1 hertz
2.048 Mbps	2^{15}-1 bits	x^{15}-x^{14}-1	62.5 hertz
6.312 Mbps	2^{15}-1 bits	x^{15}-x^{14}-1	192.6 hertz
8.448 Mbps	2^{15}-1 bits	x^{15}-x^{14}-1	257.8 hertz
32.064 Mbps	2^{15}-1 bits	x^{15}-x^{14}-1	978.6 hertz
34.368 Mbps	2^{23}-1 bits	x^{23}-x^{18}-1	4.1 hertz
44.736 Mbps	2^{15}-1 bits	x^{15}-x^{14}-1	1365.3 hertz
139.264 Mbps	2^{23}-1 bits	x^{23}-x^{18}-1	16.6 hertz

figure 3.1. CCITT recommended sequence polynomials for testing digital transmission systems.

According to Feher [14] these patterns are selected according to the following properties:

1)The length of the sequence in bits compared to the number of bits transmitted in a second.

2) The shift register configuration which defines the binary run properties.

3) The spectral line spacing of the sequence, which depends on the bit rate.

Since the device being designed is oriented primarily towards testing the link at data rates of 2 Mbps and secondarily at 8 Mbps, the polynomial recommended by the CCITT (suitable for both data rates) is

$$f(x) = x^{15} + x^{14} + 1 \ .$$

This polynomial sequence is particularly easy to implement as it only requires the use of 2 feedback taps and one feedback summing gate. As has already been stated, current thinking on the theory of PRBSs indicates that the use of many feedback taps (in general at least half as many as the number of stages) produces better sequences.

A second shift register configuration is thus chosen that should approximate more closely a true random sequence and this sequence will be compared to the CCITT recommended sequence in various ways.

The second configuration chosen should (according to current thinking) have at least n/2 feedback taps and thus a suitable polynomial (for the instance of n = 15) has the following characteristic equation

$$f(x) = x^{15} + x^{11} + x^{9} + x^{8} + x^{6} + x^{5} + x^{3} + x^{2} + 1 \ .$$

3.1. PROOF OF MAXIMUM LENGTH

The proof that the sequences used are created from irreducible polynomials and thus are of maximum length may be found in Appendix A.

3.2. SIMULATION

A simulation program was written to generate the sequences produced by user inputted shift register configurations and was useful in comparing the sequences produced by various configurations as well as providing information about the sequence around the area of the starting seed (which was later used as the synchronisation word as seen in section 5.2.2).

To compare the sequences it was decided to compare them individually against a true random sequence by looking at how a seed predicts a preceding and proceeding part of a

sequence and thus deduce intuitively how "random" the PRBS is.

This comparison is implemented by a PRBS simulation program which was developed to test the output sequences of various generator polynomials.

The simulation program is written in Turbo Pascal (with the source code listed as Appendix B) and allows the user to input length of shift register, configuration of feedback connections, and starting seed of the feedback shift register. The program will then cycle through either all the states necessary to reach the all ones vector or the maximum number (2 to the power of the length of the shift register) of states. The program will furthermore print out the vectors of the first twenty states following the seed and the last twenty vectors preceding termination.

The pages of the output in Appendix B list first the equation of the PRBS generator, then the seed and proceeding twenty vectors, a space, the twenty vectors leading to the end of the sequence and finally a listing of the number of steps counted in the sequence as well as the number of respective ones and zeroes.

3.3. COMPARISON OF SEQUENCES

With reference to the 3 properties, listed by Feher and quoted at the beginning of this chapter, specifying how the patterns are chosen, the following is noted.

1) The first property of sequence length is the same for both sequences and in fact is the same for all maximum length sequence configurations having k steps and is equal to $(2^k - 1)$.

2) The third property of spectral line spacing is also the same for both configurations and depends on the length of the sequence and the frequency of the clock.

3) The second characteristic of the binary run properties is defined by the shift register configuration polynomial which is of course different for the two examples.

In particular, the start and end of a binary sequence that has the all ones vector as a seed are interesting because this is where the least amount of bit transition occurs and the most notably non-random sections of the sequence are to be found. The number of bit transitions occurring can become important when the communications channel being tested does

not have encoding which limits the number of successive ones or zeroes.

In the case of a test unit output conforming to CCITT recommendation the output has HDB3 (high density bipolar three) encoding, which restricts the number of sequential bit values without transition to three. Long runs of no transition can thus be tolerated in the sequence because the subsequent HDB3 encoding will insert transitions to prevent loss of synchronisation. When testing links without HDB3 encoding, therefore, the bit transition patterns of the sequence and occurrences of long runs without transition in the sequence become important.

The CCITT do recommend an alternative specification [15] at 2048 kbps which features AMI (alternate mark inversion) bipolar signals and an accompanying timing signal having bit and byte synchronisation pulses. The specification includes a note to say that no more than 15 consecutive ones or zeroes should occur and the minimum mark to space ratio is recommended at 1 : 8. A pseudo random sequence with a large number of bit transitions (around the areas of long runs of either one or zero) is possibly a better test sequence on a link conforming to these specifications, as the localised mark to space ratio will always be higher.

When testing a link without HDB3 or other encoding or when testing on lower data rates such as 64 kbps, which do not feature encoding as part of their specification, it becomes advisable to ensure that as many bit transitions as is possible do occur after a long run at one constant level, to prevent loss of bit synchronisation. This is particularly important when frequency modulation systems such as FSK (frequency shift keying), PSK (phase shift keying) or QAM (quadrature amplitude modulation) are employed.

In the examples shown in Appendix B of output from the simulation program, using different characteristic formulae, all examples show the starting seed and the 20 following shift register vectors, a space and then the 20 vectors preceding the complete cycle of the sequence. To ease the resetting of shift registers the seed is almost always the all ones vector in practice therefore the examples all use this seed.

It is observed that in general k stage shift registers with 2 feedback taps (at the n'th and k'th stages) always have the seed of k ones preceded or proceeded by (k - n) zeroes and proceeded or preceded respectively by n zeroes.

Thus as an example, in the case of a 15 stage FSR with 2 feedback taps, the minimum run of zeroes adjacent to the

seed of 15 ones is seven zeroes (but this configuration is unfortunately not a maximum length one). In a maximum length configuration the minimum number of zeroes adjacent to the seed of 15 ones is 14, as can be noted in two of the examples in the appendix.

Feedback shift registers with multiple feedback taps (particularly in the order of k/2) are not as easy to characterise in general but it can be seen immediately from the examples that there is a much higher prevalence of bit transitions in the areas adjacent to the seed of all ones.

In the general characterisation of the CCITT recommended sequence it can be seen that a run of p ones is always followed by a run of (n - 1) zeroes, a result which is predictable and therefore detracting from the sequences' "randomness".

In the case of the FSR example having 8 feedback taps the number of bit transitions following a run of ones is always high with the maximum run of zeroes following a run of k ones being equal to the stage number of the first tap which is 2 in the case of the sequence I have chosen to use.

When a run of ones of length less than k (the length of the shift register) occurs, the preceding and proceeding

sections of the sequence become very unpredictable and therefore more comparable to a true random sequence.

Examples are also included of the output of PRBSs with 4 feedback taps and though they display better sequences than the 2 feedback tap versions they do not in general produce sequences with as many bit transitions as the 8 feedback tap versions do.

Thus one can see intuitively that the recommendations of both Vincent and Horowitz are correct when advocating feedback shift registers of at least k/2 feedback taps to produce the closest to true random sequences.

CHAPTER 4

SYSTEM DESIGN

The functional specifications of the error rate analyser have been set out in chapter 2 and this section will focus on the hardware and software design aspects.

4.1. SYSTEM HARDWARE CONSIDERATIONS

The basic system configuration, shown in figure 4.1, consists of a transmitter (which is linked to a microwave data link by a CCITT recommended 2 Mbps interface) generating test data and transmitting it to a receiver across the link being tested. The test data are received from the link receiver circuitry, checked and the error statistics displayed and stored.

The statistics calculated at the receiver are also communicated back to the transmitter via an RS232 communications channel where it is displayed. Commands can also be issued from the receiver to alter the operation of the transmitter. In effect there is a transmitter and receiver present on both sides of the link for testing in

both directions simultaneously (with corresponding RS232 communications in both directions).

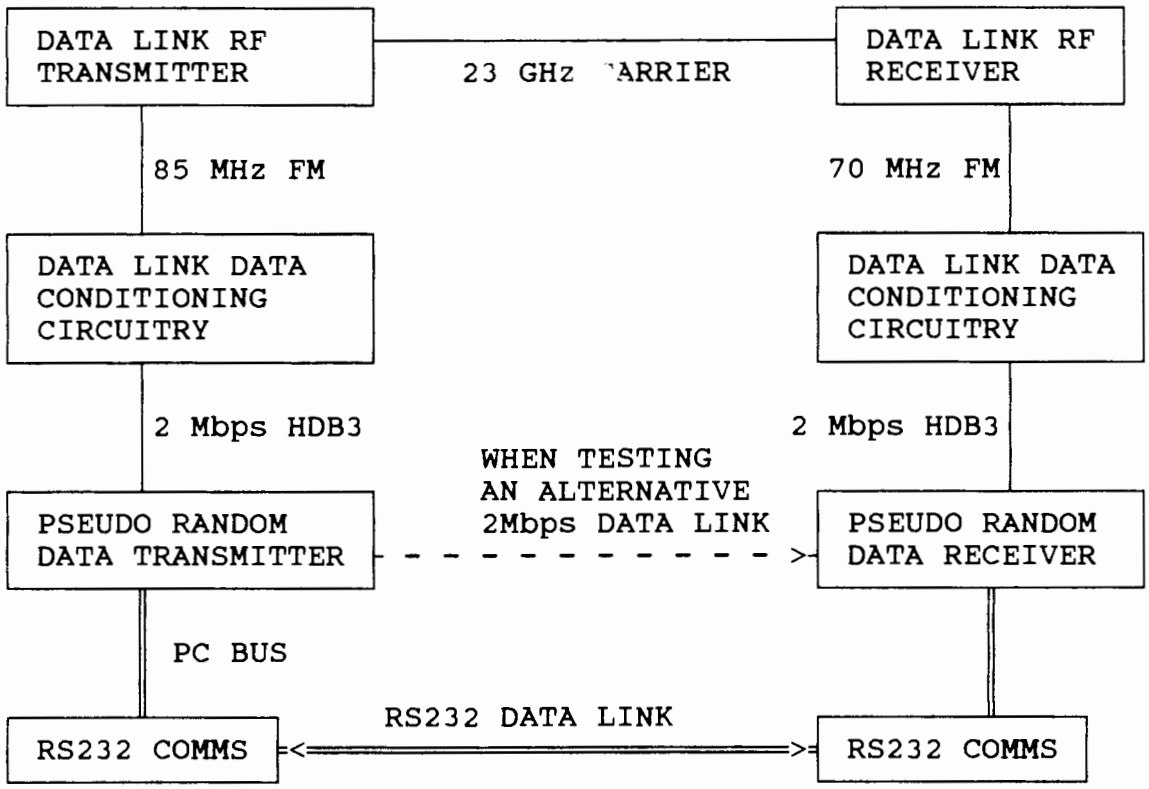


figure 4.1. Basic System Configuration

The first decision to make is whether the analyser should be a stand alone device or an optional addition to an existing system such as an IBM PC.

To support the functional requirements of the analyser (as specified in chapter 2) the following system configuration is needed:

1) An 80 by 25 CRT display

2) Long term information must be made available in printed form, requiring logging of results to printer or disk.

3) An RS232 communications link for transfer of data and commands across a link from one analyser to another.

4) A keyboard or keypad for user interaction with the analyser.

The full system requirement is thus shown in figure 4.2.

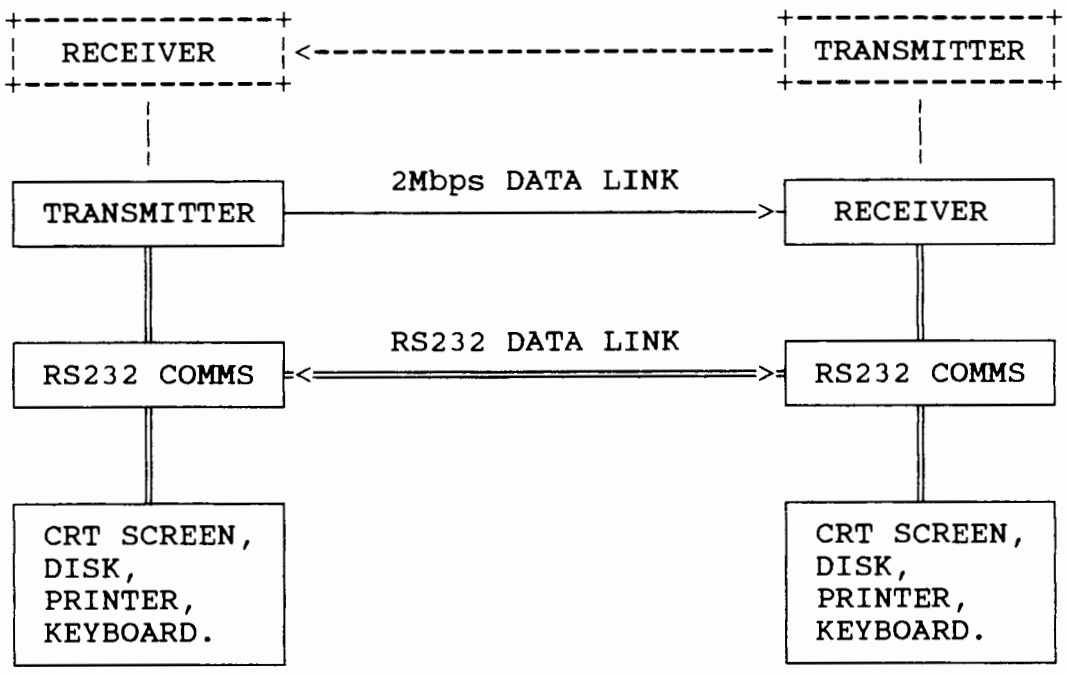


figure 4.2. Expanded System Configuration

4.2. SYSTEM SOFTWARE CONSIDERATIONS

Thus, if one considers a stand alone system there will be a considerable amount of hardware and software needed to drive the screen, printer, keyboard and RS232 link. Whereas, if one considers the analyser as an extension card to an IBM PC, these peripherals and their BIOS device drivers, together with the MSDOS operating system, are already available. The BIOS and DOS functions are easily utilised by application software to access any peripheral devices in the system.

Due to the open architecture of the PC, specific devices on the analyser card can easily be mapped into the I/O map of the PC for ease of access by application programs.

In conclusion, therefore, the development time, effort and cost of a stand alone unit will be much greater than the PC based alternative, when a very limited number of units is being produced.

4.3. ANALYSER HARDWARE

The PC extension card supports two major functions:

1) To generate data, receive data and count errors.

2) To interface the analyser hardware to the PC system using the system's data bus, control signals and timing signals.

There are two possible methods of generating, receiving and checking test data.

Firstly, the PRBS generating polynomial discussed in the appendix A can be implemented with feedback shift registers clocked at the required frequency, thus transmitting a pseudo random data stream to simulate the data stream in a live traffic environment.

The receiver would then receive this pseudo random data stream by clocking it into a register. The sequence received is then synchronised to the output of a reference PRBS generator (identical to the transmitter PRBS generator) and the two streams are compared bit for bit. The errors could then be registered in a hardware counter, read and displayed at fixed intervals by software.

Secondly, one could use a simulation program such as the one discussed earlier in chapter 4 to generate a complete pseudo-random sequence in software and store it in memory. This would require 8 Kbytes of memory per sequence stored. This data could then be accessed via a DMA channel by a serial communications chip (or block of circuitry) and

transmitted as a pseudo-random bit stream. The reception would be the above process in reverse, and comparison of the standard sequence to the received sequence could be done by software at intervals of one complete sequence with two receive buffers being used alternately.

This second system would work at lower data rates but at 2 Mbps the system bus bandwidth would not support much more than transmission and reception. The maximum rate of DMA data transfer on a standard PC bus is of the order of 1 Mbyte per sec. However, this rate is only achievable in bursts which cannot be longer than 72 clocks as a cell of the system dynamic RAM must be refreshed within this period. Both transmission and reception would require continuous rates of 2 Mbit per sec, which thus uses more than half the bandwidth available.

The system must also carry out a real time comparison of two million bits every second in software (corresponding to comparing two blocks of 256 kBytes of data per sec). If one takes 20 clocks for comparison and counting of bit errors per byte of data, the comparison procedure will require 5 million clocks every second, which accounts for 100% of the CPU time.

The system must further generate a screen display, dump data to disk or printer and transfer data across an RS232 link. Consequently the second choice may be ruled out.

Therefore, using the first option, the analyser card hardware can be partitioned into the functional blocks shown in figure 4.3.

The extra features of optional error injection (in the transmitter) and optional internal loopback (in the receiver circuitry) are included to provide a means of connecting the receiver section to the transmitter section on the same card to inject a fixed amount of errors into the data stream at fixed time intervals and verify the operation of the analyser card.

One of the requirements shown in the block diagram is for three counters/dividers: one is needed to count the bit errors, one to control the error injection process and one to generate a timing interrupt for the system.

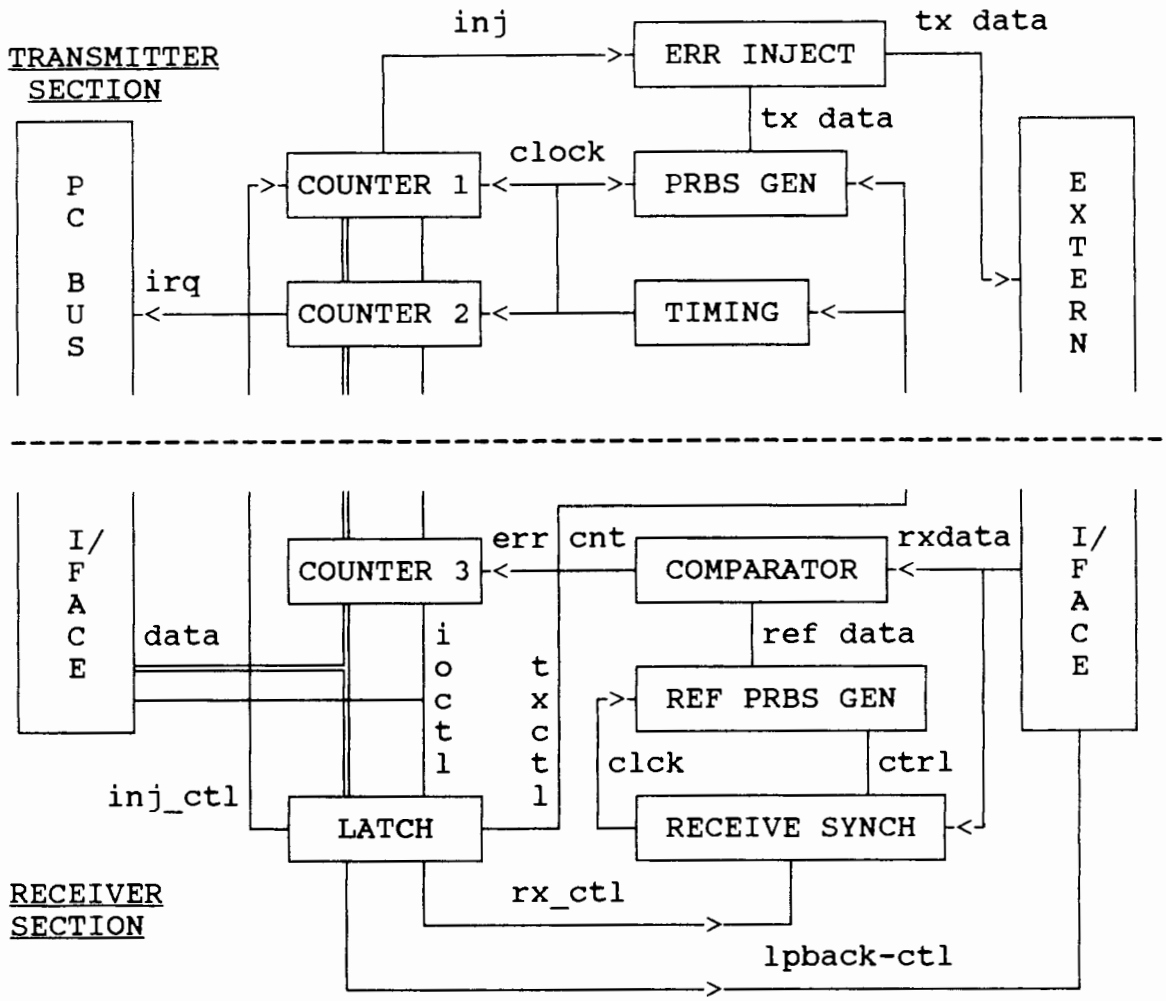


figure 4.3. Block Diagram of Analyser Card.

A general purpose latch is also included for controlling the separate functions of the card such as clocking, transmitting, receiving, injecting errors, reference sequence generating and internal loopback enabling.

4.4. INTERFACE HARDWARE

The interface hardware provides two basic functions:

1) To buffer the data and control lines from the PC I/O bus so that no more than one TTL load is presented to any line. Refer to the IBM Technical Reference [16].

2) To decode the address lines of the PC I/O bus to give the analyser board (and the specific devices thereon) an address space within the general PC I/O map.

The I/O address map of the analyser ports, including the RS232 ports (which is allocated the primary serial communications address space) is shown in figure 4.4.

I/O ADDRESS	FUNCTION OR DEVICE
000 - 0FF	RESERVED FOR PC SYSTEM BOARD
020 - 03F	INTERRUPT CONTROLLER 8259
31B	ANALYSER CARD GENERAL LATCH
31C - 31F	ANALYSER CARD COUNTERS 8254
378 - 37F	PARALLEL PRINTER PORT 1
3B0 - 3BF	MONOCHROME DISPLAY CARD
3D0 - 3DF	COLOUR GRAPHICS CARD
3F8 - 3FF	SERIAL COMMUNICATIONS PORT 1

figure 4.4. I/O Address Map

There are two interrupt vectors which will be changed to point to application routines:

1) The IRQ3 interrupt, specified in the IBM Technical Reference for secondary serial communications, is used by

the analyser card to signal that its counters need to be read.

2) The IRQ4 interrupt, specified for primary serial communications, is used by the RS232 card at the transmitter to signal that a communications byte has been received from the opposite end of the link (where the PRBS stream is received).

The IRQ4 interrupt service routine need only respond before the duration of the reception of a further byte. The IRQ3 interrupt is timed to read the error counter on the analyser card at 10 mSec intervals (100 readings thus making up one second of error readings). The IRQ3 interrupt is thus given a higher priority than the IRQ4 on the 8259 system interrupt controller.

The interrupt vectors used and their addresses in low memory are listed in figure 4.5.

VECTOR NO.	ADDRESS	FUNCTION
0B hex	2C - 2F	SECONDARY COMMUNICATIONS
0C hex	30 - 33	PRIMARY COMMS (ANALYSER CARD)

figure 4.5. Hardware Interrupt Vector Listing

CHAPTER 5

CIRCUIT DEVELOPMENT

The overall design of the error rate tester is partitioned into four functional blocks:

1) The test data generation and transmission block consists of a pseudo random sequence generator (using feedback shift registers) and a clock generator. A section of circuitry injects periodic errors into the transmitted data stream. This addition is used to verify the operation of the bit error rate analyser during initial loopback testing and also during normal operation.

2) The receiving block receives and clocks the incoming PRBS data, checks for a synchronisation word to synchronise and immediately compare the sequence received against a reference sequence.

3) The system interface block provides buffering and decoding from a PC system I/O channel in order that the analyser board may be accessed by application software running on the system.

4) The external interface block provides appropriate signal conditioning and connections according to the specifications of the link as specified in reference 4.

5.1. SEQUENCE GENERATOR

The generation of the pseudo random bit sequence is broken up into 3 sections: the clock generation, the feedback shift register and the error injection circuitry.

5.1.1. Clock

The CCITT clock specification [17] for the link is 2048 kbps ± 50 parts per million.The clock generator used in this project (as shown in figure 1 of Appendix I) is an adaptation of a crystal oscillator circuit design used by Matthys [18] and it is basically a standard TTL inverter feedback oscillator with trimming capacitors.

Matthys [19] specifies a frequency stability for the TTL inverter oscillator of 1 ppm per supply volt swing using 1 % tolerance resistors and a crystal with a 0.5 ppm change in frequency over the operating temperature range. The operating temperature range of the PC is defined by the IBM Technical Reference [20] as 15 to 32 degrees centigrade.

This crystal oscillator circuit has a stability of the order of 8 ppm under normal operating conditions which is well within the CCITT recommended limits quoted above.

To fulfill all the criteria of a full error rate tester the clock should be of high stability, temperature compensated crystal oscillator with the ability to introduce calibrated amounts of clock jitter. This capability has not however been included as the development of a precision jitter signal generator was envisioned as a seperate project which could interface to this one.

The CCITT specified lower limit input jitter tolerance [21] is 1.5 UI (unit interval, see Appendix L for definition) at jitter frequencies from 20 Hz to 2400 Hz and 0.2 UI at frequencies above 18 kHz.

CCITT studies have shown that jitter measurements can be influenced by the fact that the data transmitted is pseudo random as opposed to live traffic [22]. The exact correlation between these two conditions has yet to be studied in depth but the present recommendation for pseudo random patterns is a maximum peak to peak output jitter of 1 UI [23] which the unit again easily achieves.

The waveform at the crystal is squared off and buffered by a third inverter and then buffered and divided by a 74ls73 providing the clock and inverted clock necessary for transmission and error injection on the board. A further divide by 2 gives the clock necessary for the timing interrupt circuitry.

5.1.2. Feedback Shift Register

In the general theory and modelling of feedback shift registers (FSR's) in chapter 3 the shift register has been modelled as having a starting seed of all ones and a linear summing feedback circuit consisting of exclusive OR gates.

In practice, however, the easiest reset condition of a shift register is the all zeroes vector. In conventional FSR's this seed gives a sequence of one step only. However, if one uses exclusive NOR gates in the feedback stages and inverts the final output the sequence obtained is the same as the one modelled with the advantage thar the reset circuitry is greatly simplified.

Two 8 bit parallel out shift registers, 74LS164's, are cascaded and the last output left unused to give the necessary 15 stages as shown in the fixed polynomial generator in the circuit diagram figure I.1.

5.1.3. Error Injection

A facility is provided whereby an exact amount of errors can be periodically injected into the transmitted stream so that self-verification of the bit error rate analyser unit can be performed.

The injection circuitry is shown in figures 1 and 3 of Appendix I and is achieved by using one of the counters on the 8254 timer device. The clock for the counter is the inverted transmit clock and the counter is programmed to be in the interrupt mode where on count termination it produces a pulse on its output pin that is one clock period long. The counter then reloads the initial count value and repeats the process.

The pulse produced by the counter is fed to one input of a 2 input EXNOR gate where it is used to selectively invert the transmitted stream of data produced by the PRBS generator thus injecting a very defined number of errors per second (set by the start value of the counter which is programmable).

The waveforms produced can be seen in figure 5.3. with INJ being the injection pulse produced by the 8254 and DIN and DOUT being the data stream before and after the error

injection respectively as shown on the circuit diagram in figure 1 of Appendix I.

5.2. RECEPTION

The receiver section is divided into four sections namely: data stream reception, data stream synchronisation, reference generation and data stream comparison with error counting.

5.2.1. Incoming Stream

The PRBS data are in HDB3 encoded form according to CCITT specifications [24] for a 2 Mbps data link. Since each analyser card has a transmitter and a receiver, the decoding and encoding are implemented by a special codec chip, the MV1441, from Plessey Semiconductors . The clock recovery is achieved internally with the two TTL constituent parts of the received HDB3 signal (A and B signals shown in appendix K) OR'ed together to give a TTL signal with at most two bit periods with no transition. This signal is synchronised to a 16.384 MHz crystal clock in a divide by 8 phase lock loop. The resultant clock is in phase with the received data and has a high jitter tolerance (theoretically 1/16 of the clock frequency).

The HDB3 codec circuitry is shown in figure 4 and the receiver circuitry in figure 2 of Appendix I.

TTL level signals are also made available at the external interface (for unit self-testing purposes or for links which do not utilise HDB3 encoding) but no clock recovery circuitry is provided for the straight TTL interface on the board so clock and data signals are necessary to use this interface.

The incoming stream is clocked into a 16 bit shift register (2 cascaded 74ls164's) provided the master receive enable line (RX-SR-EN in figure 2 of Appendix I) has been enabled.

5.2.2. Synchronisation

Synchronisation is defined here as the synchronisation of the incoming data stream to a reference stream generated by the receiver.

Conceptually synchronisation can be done once, when receiving a recognised pattern or "synch word" for the first time and the two streams should stay in synchronisation from then on as they are clocked by the same clock. However, when operating in an environment with a large amount of electromagnetic interference a receiver clock glitch could

be induced which could throw the reference one state ahead of the incoming stream. It is thus seen as advisable to resynchronise the two streams once every sequence period when the synch word is repeated. This occurs once every 32767 bits or 64 times a second at a data rate of 2 Mbps.

The seed chosen for the sequence is the all ones state but if this is used as a synch word, one finds that there are many possible instances on a link where all ones could occur. For instance many links implement an all ones state when the link is idle and thus if the transmitter on the opposite end of the link is not enabled a continuous stream of ones may be received.

A better approach is thus to look for a longer synch word by checking before and after the all ones vector for the zero bit that must be present on either side of it. The very first synch word received after transmission starts can still be identified correctly (even though it may not have a preceding zero) and resynchronisation will be surer.

To implement this synchronisation format (as shown in figure 2 of Appendix I) one has to monitor 16 bits instead of 15 and to check for two successive patterns being firstly a zero followed by 15 ones, and secondly 15 ones followed by a zero. This two phase synchronisation is done to prevent

glitch errors in comparitor from being counted when the reference generator is reset with its synch pulse. The reference PRBS generator is thus reset by recognition of the first pattern and enabled by recognition of the second.

The enable line to the error counter is also disabled and then enabled during resynchronisation to prevent spurious errors being counted during the resynchronisation process. This enabling and disabling is controlled by an RS flip flop with the R and S inputs being gated by the positive and negative receive clock edges respectively to prevent glitches. This flip flop is designed so that the reset pulse will disable the receive clock feeding the receiver reference generator resetting it on the last bit of the previous sequence. The flip flop will then enable the reference generator and the clock feeding it as well as the error counter gating input. These three actions are all carried out at the same time (on the first bit of the next received sequence) so that the first bits of the two sequences are compared synchronously (the reference is clocked one stage as soon as the clock is enabled because it is leading edge sensitive).

From the above explanation one can deduce the need for two signals to be derived from the receiver shift register. One must consist of 15 ones followed by a zero and the other

must consist of a zero followed by 15 ones. Both are gated by the receive clock to ensure that they are synchronous signals.

The equation for the reset line in figure 2 of Appendix I is thus

$$\overline{RESET} = \overline{Q14.Q13.Q12.Q11.Q10.Q9.Q8.Q7.Q6.Q5.Q4.Q3.Q2.Q1.Q0} + Q15 + \overline{CLOCK}$$

and the equation for the set pulse is

$$\overline{SET} = \overline{Q15.Q14.Q13.Q12.Q11.Q10.Q9.Q8.Q7.Q6.Q5.Q4.Q3.Q2.Q1} + Q0 + CLOCK.$$

But since one wants to generate both pulses with common circuitry one extracts a common term (output U18 in figure I.2).

$$\overline{COM} = \overline{Q14.Q13.Q12.Q11.Q10.Q9.Q8.Q7.Q6.Q5.Q4.Q3.Q2.Q1}$$

and thus U28 output

$$\overline{RESET} = \overline{COM} + \overline{Q0} + Q15 + \overline{CLOCK}$$

and the set pulse becomes

$$\overline{SET} = \overline{COM} + \overline{Q15} + Q0 + CLOCK$$

A timing diagram of the relationship between the pulses RESET and FFSET (called SET in the circuit diagram) and the received data (DREC) and reference data (DRGEN) is shown in figure 5.1 (these signals are also shown in the circuit diagram figure 2 of Appendix A). It can be seen that this is the first synch word to be received, since the reference generator is reset in the first half of the diagram and the comparator line CLK1 has half the bits received in error (although the gate for the error counter is low, disabling it).

In the same diagram it can be seen that the reference sequence (DRGEN) is bit synchronised to the incoming sequence (DREC) and that in terms of the error detector synchronisation takes place before the all ones vector is compared between the two sequences.

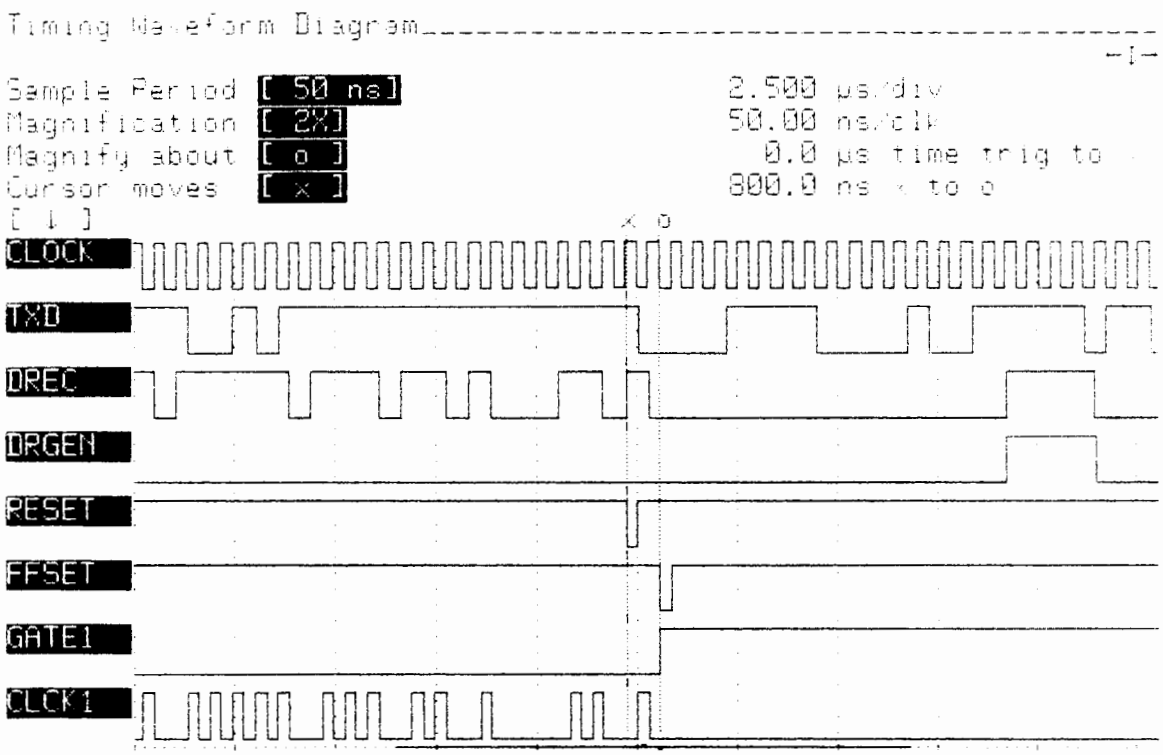


figure 5.1. Receiver flip flop timing diagram

A clearer picture of resynchronisation (when the synch word is not the first to be received) is shown in figure 5.2; it can be noted that the error generated (on CLK1) by the reset pulse is gated and therefore not counted.

If there are errors in the block of 15 ones in the synch detect word then there will be no reset or start operation caused by the synch word at the end of the incoming

sequence. However, the receiver PRBS generator will automatically cycle and restart the sequence which should be in phase with the incoming sequence as long as no receive clock glitches have caused the reference sequence to become one step ahead of the incoming sequence.

In the case of a clock glitch occurring during a sequence there will be a very high error reading for the duration of the present sequence until the next synch word resynchronises the generator (as the resynch occurs roughly 64 times every second at 2 Mbps data rate this should not cause too marked an effect in long term monitoring).

If the error in the synch word occurs in the leading zero then the generator will not reset but will automatically restart its sequence and count the single bit error.

If the error occurs in the lagging zero of the synch word then the generator will reset but not start up again and there should be 16383 bit mismatches in the duration of the next sequence. None of these should, however, be counted as errors because the gate of the error counter will have been reset and will not be set again until the next resynch.

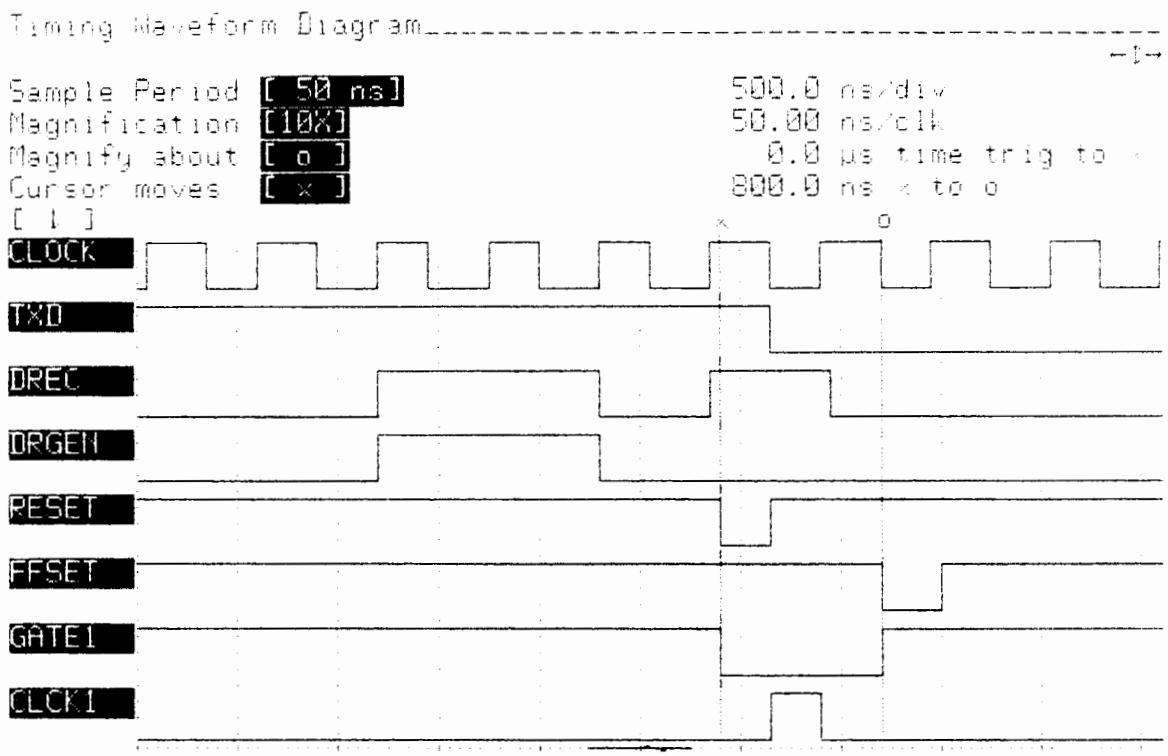


figure 5.2. Resynchronisation timing diagram

The only other possible synchronisation error which can occur is when a false synchronisation word sequence is created by bit errors in the normal sequence in which case the generator will be out of synch with the incoming data stream and a high error count will occur until the next resynch.

If the error rate on the link is very high then the probability of this synch error occurring will be higher and the error readings will tend to wander up and down. In measurement observation this false synchronisation has only been seen to occurr when the amount of errors injected into the stream is of the order of 1000 ppm. Accurate measurement of error statistics this high are not of practical importance as the link would be too degraded to support any practical communications.

5.2.3. Reference Generation

The reference pseudo random sequence generator (see figure 2 of Appendix I for the circuit diagram) in the receiver is basically the same circuit as its transmitting counterpart. The only difference is that the reset and clock inputs are taken from the synchronisation circuitry and it is generally only reset by this circuitry although the whole synchronisation and reference generation section may be reset by software at shutdown.

It is not necessary for the output of the reference generator to be inverted because it is compared to an inverted version of the received PRBS data as shown in the circuit diagram.

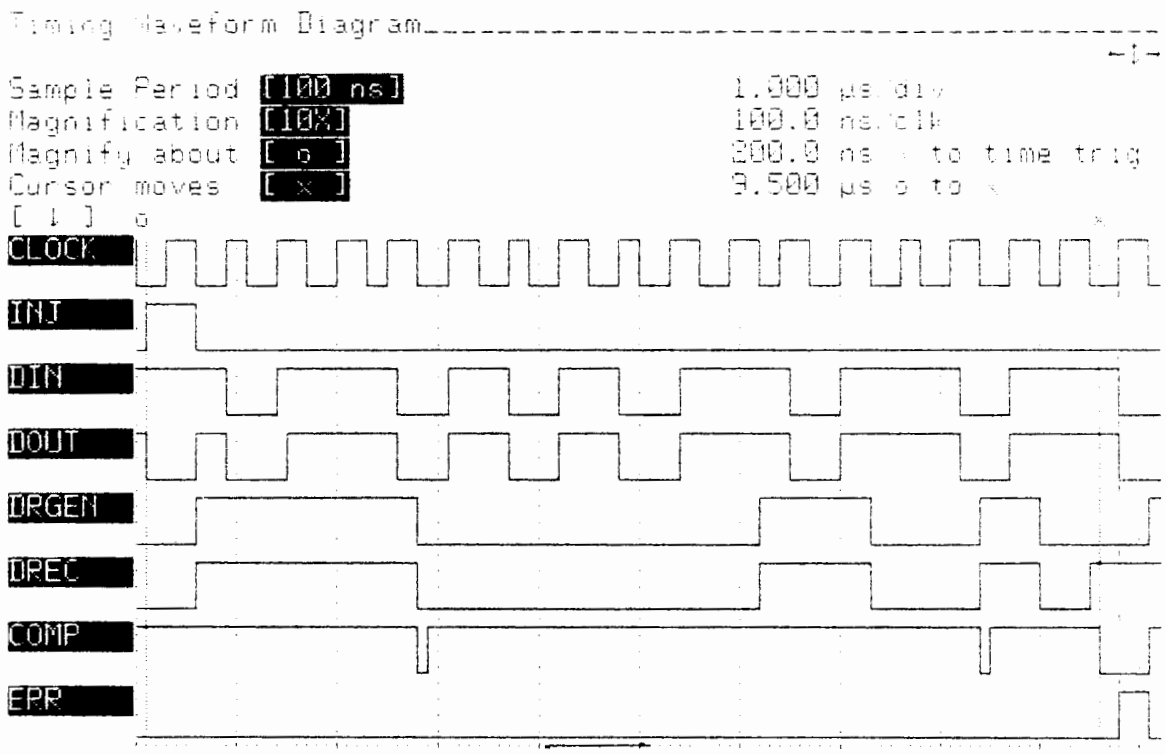


figure 5.3. Error injection and counting waveforms

5.3.4. Correlation

Figure 2 of Appendix I shows that the output of the reference generator and the inverted incoming data are compared by an EXNOR gate whose output is gated by the positive going clock pulse of the receive clock to prevent glitches (which could result from any phase difference between the received data and the reference data).

The detection of an injected error while operating in loopback mode can be seen in figure 5.3 where the error injecting pulse INJ forces a bit inversion error into the bit stream DIN which becomes DOUT (with signal naming as in the circuit shown in figure I.1 of Appendix I). The bit stream is transmitted, looped back and received. The received stream is DREC and 15 clock cycles after the error injection pulse INJ occurs in real time there is a difference between DREC (the received data) and DRGEN (the reference). The circuit for these signals may be seen in figure 2 of Appendix I. The difference between DREC an DRGEN is picked up by the comparator whose output is COMP and since the difference in streams occurs for a full bit duration and is not a glitch (as can be seen earlier on the COMP line resulting from a slight phase difference between DREC and DRGEN) an error count is pulsed by the positive going receive clock onto CLK1 (the error counter input clock).

5.3. SYSTEM INTERFACE

The object of this circuitry is to interface the bit error rate analyser card to the general I/O bus of a personal computer. Control and data lines are buffered and addresses are decoded to map the ports on the analyser board to

specific I/O system addresses so that they are accessible by programs written on the PC.

5.3.1. Buffering

The circuit diagram of the buffering of the system data bus and accompanying control lines is shown in figure 3 of Appendix I.

The data bus is buffered by a 74ls245 bidirectional octal transceiver with the direction of buffering enabled being selected by the I/O channel read line (IOR). Thus when the IOR line is pulsed low the CPU of the PC accesses the board.

There is also a buffer enable line which is only enabled when an address within the address space of the board is accessed.

Address lines that are not decoded (and thus utilised for addressing on board) and control signals are buffered by a 74ls244 which is permanently enabled.

5.3.2. Decoding

3 programmable counters to count errors, to software control injection of errors and to generate a general timing interrupt are required. A general 8 bit latch is used to control the functioning of the board.

The error rate analyser board I/O ports are mapped to the top end of the PC I/O addresses which has been allocated for board prototyping (i.e. the addresses from 318H to 31FH). The reset latch resides at 31BH and the registers of the 8254 programmable counter reside from 31CH to 31FH.

5.4. EXTERNAL INTERFACE

Physically the board is interfaced to the outside world by card mounting a 9 pin D connector with pin assignments shown in figure 5.4. The CCITT recommends using HDB3 encoding for data links at 2 Mbps [25] which is made available, taking four pins for the two balanced lines. In addition, it was found useful to have TTL level clock and signal available for a short cable length to a link which uses NRZ data or to a simple cable link. This requires another four lines for input, output and clocks.

pin 1:	HDB3 TRANSMIT
pin 2:	HDB3 TRANSMIT
pin 3:	HDB3 RECEIVE
pin 4:	HDB3 RECEIVE
pin 5:	GND
pin 6:	2 Mbps TTL transmit data stream
pin 7:	2 MHz TTL 50% duty cycle transmit clock
pin 8:	2 MHz TTL 50% duty cycle receive clock
pin 9:	2 Mbps TTL receive data stream

figure 5.4. 9 pin D connector pinout

The pinout does not conform to any standard. An adapter is also included to convert the balanced lines from D connector to 2 BNC sockets as shown in figure 8.1.

The external line driving circuit (as shown in figure 4 of Appendix I) supporting the MV1441 is basically as recommended in the application notes (see Appendix K) with the dual polarity outputs (amplified by the 2 transistors) driving either side of a balanced line transformer (which is wound by hand on an RS RM6 core) giving the required 3 level balanced line output.

For a full specification of HDB3 encoding see CCITT recommendation G.703 Annex A [26].

The input circuitry is symmetrical to the output with the balanced line transformer driving 2 transistors, which

provide the chip with the 2 constituent TTL parts of the HDB3 signal (the waveforms can be seen in Appendix K).

The complete external interface circuit is shown in figure 4 of Appendix I. Note that the loopback enable pin from the MV1441 is taken back to the reset latch to allow software switchable internal loopback on the card.

5.5. COMMUNICATIONS HARDWARE

A general purpose asynchronous communications card with 25 pin D connector RS232 output (only transmit data, receive data and ground lines are utilised) is used for the communications hardware. The following options are set:

1) The address space of the card is selected to be 3F8 hex to 3FF hex (being the primary serial communications address).

2) The interrupt request line used is selected as IRQ4 because the analyser card generates an interrupt on IRQ3. Interrupt contention is discussed in the relevant software section in chapter 7.

3) The baud rate is fixed at 2400 baud.

5.6. PHYSICAL DEVELOPMENT

The prototype hardware was built using a PC-35 wire wrap prototyping extension card. All the initial analyser hardware and low level programming was developed and tested on this prototype card.

The final version of the analyser card had its schematic diagrams (seen in Appendix I) captured using the PC-CAPS utility of the P-CAD packages [27].

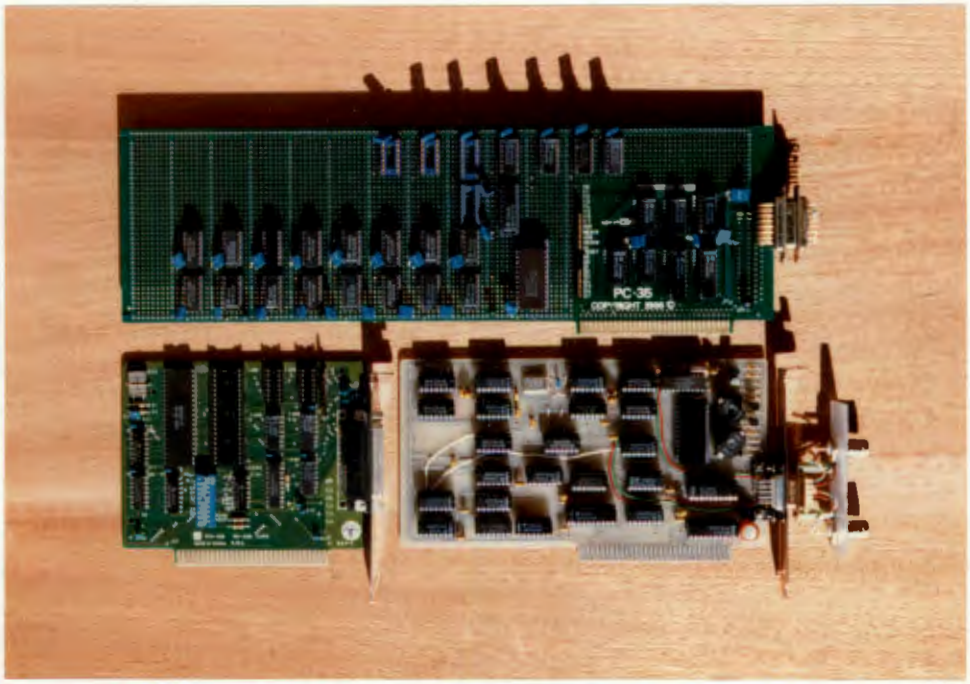


figure 5.5. Circuit cards used in the project.

The printed circuit board outline, taken from the PCB layout file designed using PC-CARDS, is included as figure 5 of Appendix I to show evidence of the work done in providing a final working model of the error analyser board. Four such boards were manufactured from plots (using PC-PLOTS) of the double sided pcb track layout.

The final printed circuit analyser board is seen in figure 5.5 alongside a serial communications card and below the prototype card. The PCB version is configured for the CCITT recommended sequence and has the CCITT recommended HDB3 encoded balanced line signals at the output. An adapter converts from a D-connector to 2 BNC sockets. NRZ signals are also present on the 9 pin D-connector.

CHAPTER 6

SOFTWARE DEVELOPMENT

The objective of the software is to allow the setting up of the test mode and display of bit error rate statistics.

There are 2 methods of implementing this software. The first is to write a large monolithic program which sequentially polls all the possible data inputs (eg. user keyboard entry, RS232 communications reception, bit error rate reading on the analyser card etc.) and takes the appropriate action.

This type of program would be extremely hard to maintain. It would also be difficult to find a particular bug in the program because trace execution of each particular function would be very difficult.

The second and more logical solution is to use a modular layered structure. Each procedure is developed and debugged separately and only when all the modules are tested and working then are they all linked together. The individual modules intercommunicate through a shared data structure. Furthermore the event driven procedures (reception of an RS232 communications byte, reading of the bit error rate

counter at a timed interval) are accessed by the interrupt service routines (ISRs), thus executing independently of the main user interfacing program.

6.1. DATA STRUCTURES

The following data structures are declared and accessed by more than one module:

1) A status and control structure for the analyser card.

2) An RS232 communications received buffer.

3) The overall unit operational status structure.

4) The bit error rate statistics data structure.

The data flow between the major functional modules is shown in figure 6.1.

6.2. SOFTWARE FUNCTIONAL MODULES

The overall objective of the software is thus to implement a menu-based controlling, recording and logging program to interface with the hardware developed and to implement easy

user manipulation and configuration of the device as a test instrument.

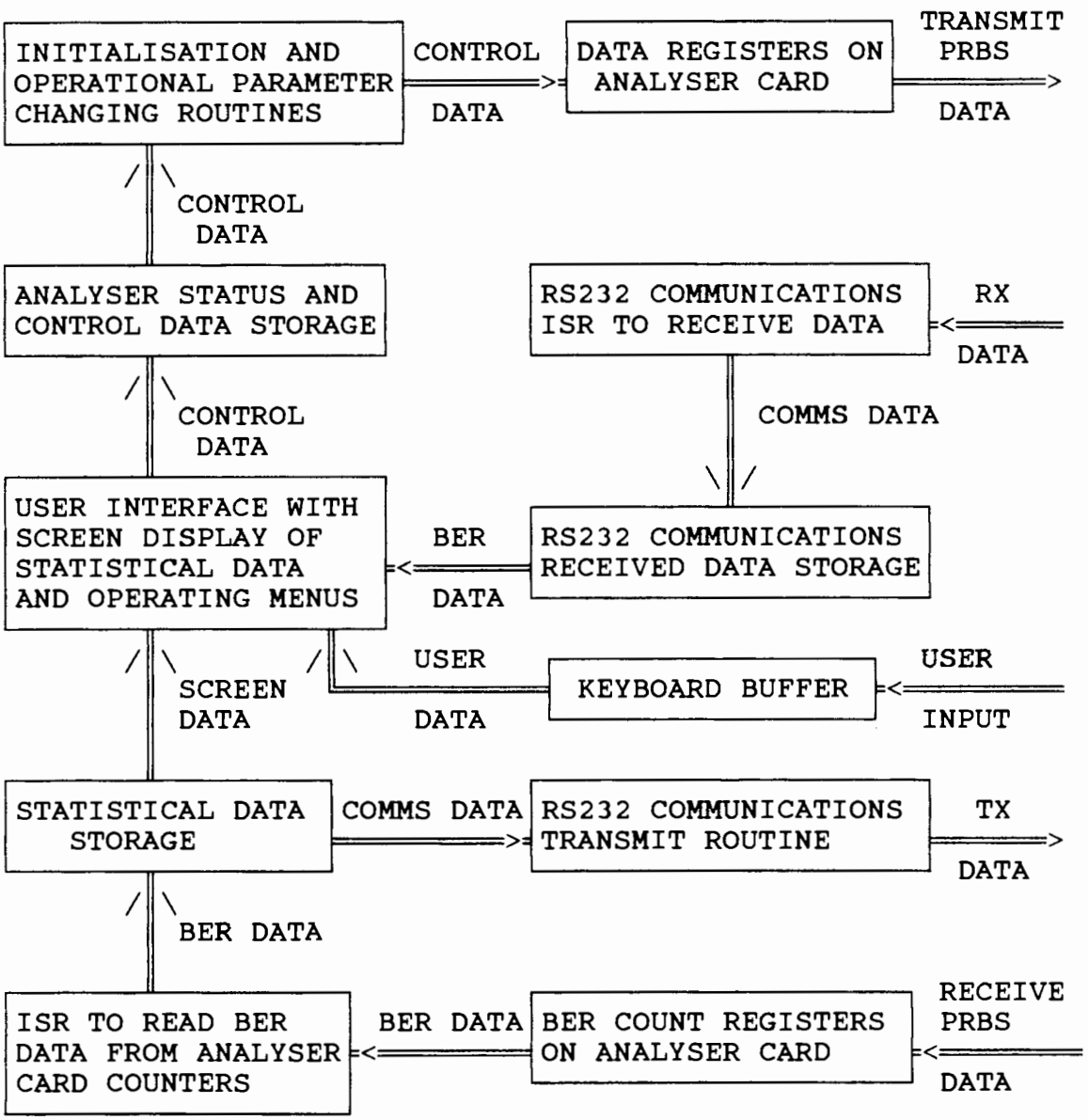


figure 6.1. Data Flow Diagram.

From figure 6.1 the software can be partitioned into 4 functional sections namely:

1) User interface routines which provide all the high level menus to access the control routines and generate the display of statistical information as shown in figure 6.3.

2) The controlling routines to provide the direct access to the test unit for initialisation and changing of the operating mode of the analyser. These involve bit or byte manipulation of the data transferred.

3) Hardware interrupt servicing routines (ISRs) to provide fixed timing for the reading of the error counter and to fetch and store data and commands received over the RS232 communications channel.

4) Across link communication routines which control the transfer of data and commands on the communications channel. These routines are often called from menu routines to implement commands on the opposite end of the link.

The development of the software is covered in the above order after a short section describing the compilers and libraries used.

6.3 ENVIRONMENT

The software was developed on a personal computer running under DOS version 3.20 [28] and it consists of programs written at three levels: applications, system and device drivers. The application layer consists of menu generating procedures, user input or output, file creation and logging, etc. The system layer controls the hardware designed or any other devices within the system. The drivers involve direct bit and byte manipulation and actual device addressing and control thereof.

When approaching all programming problems the following order of possible methods (as recommended by IBM [29]) is tried: using high level language library functions; using DOS interrupt functions supplied by the operating system; using BIOS function calls existing in ROM on the machine; if it is not possible to achieve by any of the other methods, direct Assembly Language programming of the specific devices to be accessed. The hierarchy of software levels is shown in figure 6.2.

With the high and intermediate level the C programming language is generally used; low level routines are 8086 assembly language based.

<table>
<tr><td colspan="3">HIGH LEVEL APPLICATION PROGRAMS</td></tr>
<tr><td>INTERMEDIATE LEVEL
APPLICATION SOFTWARE</td><td colspan="2">INTERMEDIATE LEVEL
LIBRARY ROUTINES</td></tr>
<tr><td colspan="3">DOS FUNCTIONS - TRANSIENTLY RESIDENT IN RAM</td></tr>
<tr><td>LOW LEVEL
APPLICATION SOFTWARE</td><td>LOW LEVEL
LIBRARY ROUTINES</td><td>BIOS ROUTINES
PERMANENT IN ROM</td></tr>
</table>

figure 6.2. Software Hierarchy.

The individual C language procedures (especially the high level routines) were developed and tested in the Turbo-C integrated environment [30] because of the ease and speed of recompiling, linking and running (for testing purposes) programs from the same development environment.

The intermediate and low level modules and the integrated modules, comprising C routines and 8086 assembly language routines, were compiled to object modules using the Microsoft C compiler version 4.0 [31] and the Microsoft macro assembler version 3.50 [32]. The final executable files were produced by the Microsoft linker version 4.0 [33]. The Microsoft system was used for the final linked versions of the software because of the powerful Microsoft windowed symbolic debugger program, Codeview [34]. This debugger was used extensively when developing the final linked version of the program due to the capability of tracing execution in

the C source code and the linked assembly source routines at the same time.

The basic libraries used in the C programs are from the standard small memory model C libraries with enhanced library functions such as windowing and file manipulation routines coming from the C TOOLS PLUS [35] libraries.

The screen generation of tables and menus requires the installation of the ANSI driver on system startup (this can be specified is the system CONFIG.SYS file) and a copy of ANSI.SYS must be present in the root directory [36].

The calls between the C modules and the assembly routines were done using standard C calling procedures and stack usage as defined by the Microsoft C manual [37].

The software was developed inthe following order:

a) The assembler initialising routines and analyser card accessing routines listed in Appendix F.

b) The main user interfacing routines listed in Appendix C.

c) The supplementary menu window routines listed in Appendix D.

d) The RS232 communications channel initialising and accessing assembler routines listed in Appendix G.

e) The calls to the RS232 communication routines from the user interfacing procedures in Appendix C.

f) The logging menu and file managing routines listed in Appendix E.

6.4. USER INTERFACING ROUTINES

User interfacing takes three forms, namely: the printing of the tables of statistical values (one table for the transmitter and one for the receiver) the pop-down menu windows for operating parameter alteration and the logging to disk or printer of specific results when necessary.

6.4.1. The tables

An example of the tables is seen in figure 6.3 and one can see at a glance that all the preferred formats of statistical bit error rate reporting are represented here at the same time.

The left hand table is defined as the bit error results for the received data stream. Similarly, the right hand side of

the screen is defined as being the bit error statistics for the transmitted data stream (received via the RS232 link).

There is also a message and status box in the lower part of the screen where messages related to current operations are displayed (errors are being injected, the program is logging results etc.) as well as general error messages (no analyser card in the system).

The source code for generating the tables uses ANSI escape sequences for cursor positioning and standard ASCII box drawing characters are used to generate the table outlines and divisions. The source is listed in Appendix D.

6.4.2. Menu windows

The windows are created and displayed using predefined structures and functions supplied with the C TOOLS PLUS supplementary libraries for Microsoft C. The structures for each window are first created in memory. They can then be written to, updated, displayed on the screen, removed from the screen and finally, if not needed again, destroyed.

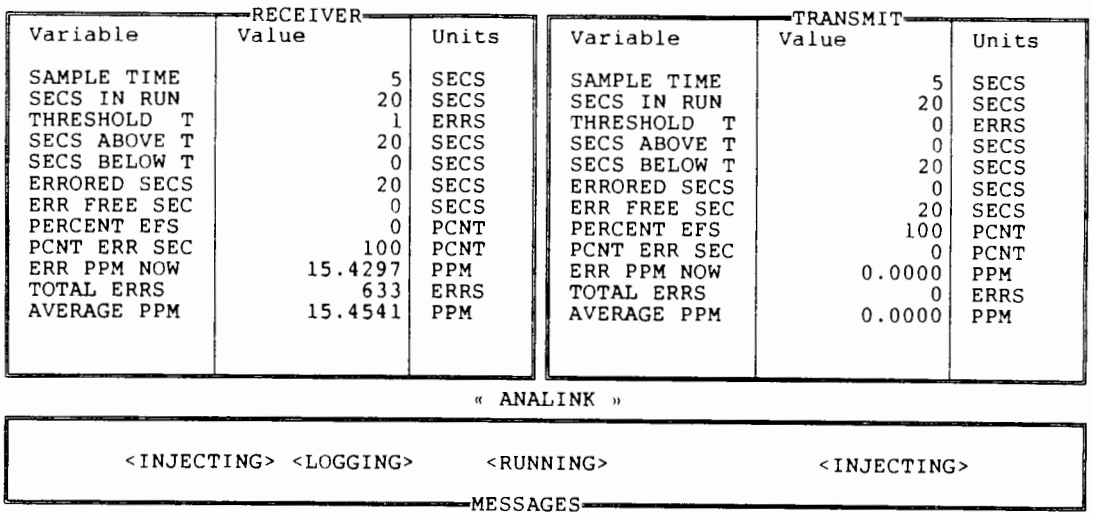


figure 6.3. Statistical tables screen.

Because each window is a separately defined structure and a record is kept of which windows are being displayed, the windows can open up over other windows on the screen and when removed, the previous contents of the screen will be redisplayed. The windows can also be written to and then displayed as the display command simply copies the window structure directly to the display memory of the current page.

Information is passed as parameters to the menu window routines as to which side of the screen to display the window on (transmitting or receiving). The options currently available to the user from that menu are displayed according to a set of globally defined flag variables. These flag variables all have names of the form "???*_flag" as can be seen in the source code for the menu routines in Appendices C and D.

The menu windows pop down over the tables as shown in figures 6.5 and 6.6 and each of the menus have a separate function. The first menu reached in the program is MENU 1, from which one can start or stop transmission or reception of pseudo-random data over the channel associated with that side of the screen. The program always starts up in MENU 1 RX (left hand side of the screen) and can only exit through one of the MENU 1's. The accessibility path between menus is shown in figure 6.4 which shows direction of accessibility.

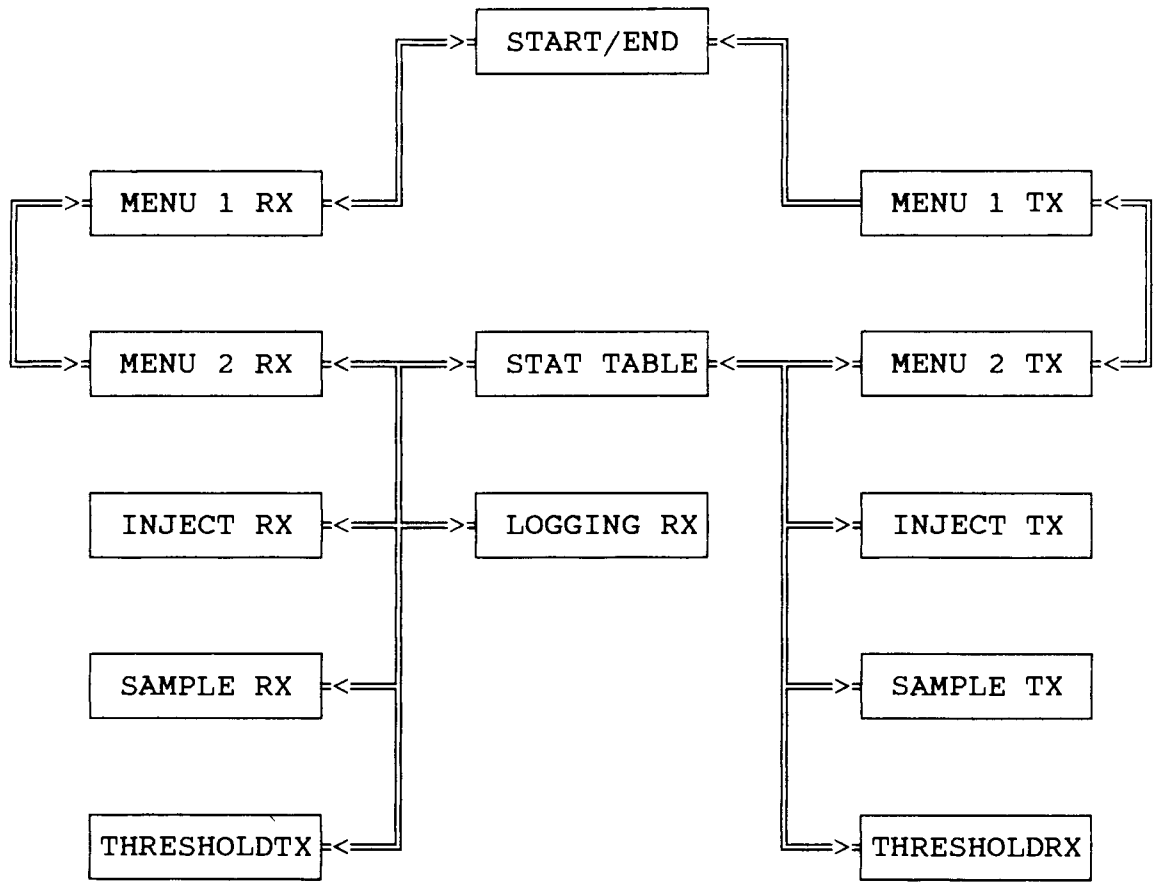


figure 6.4. Menu connectivity diagram.

MENU 2 controls the actual commands to display, inject, log, and toggle between sides of the screen. The other menus control functions such as setting of parameters for error threshold, sample time, error injection rate and logging file names.

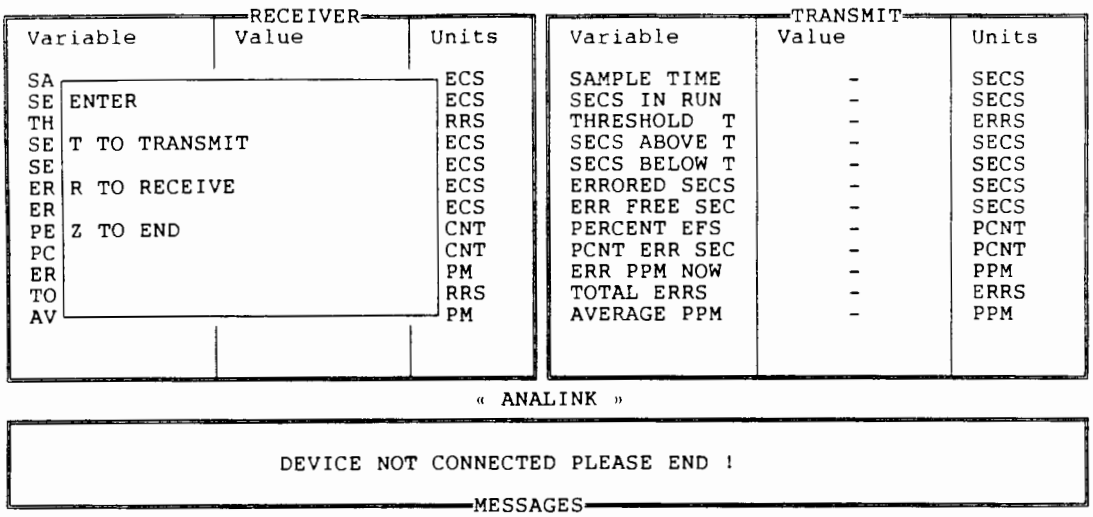


figure 6.5. Receiver window menu 1.

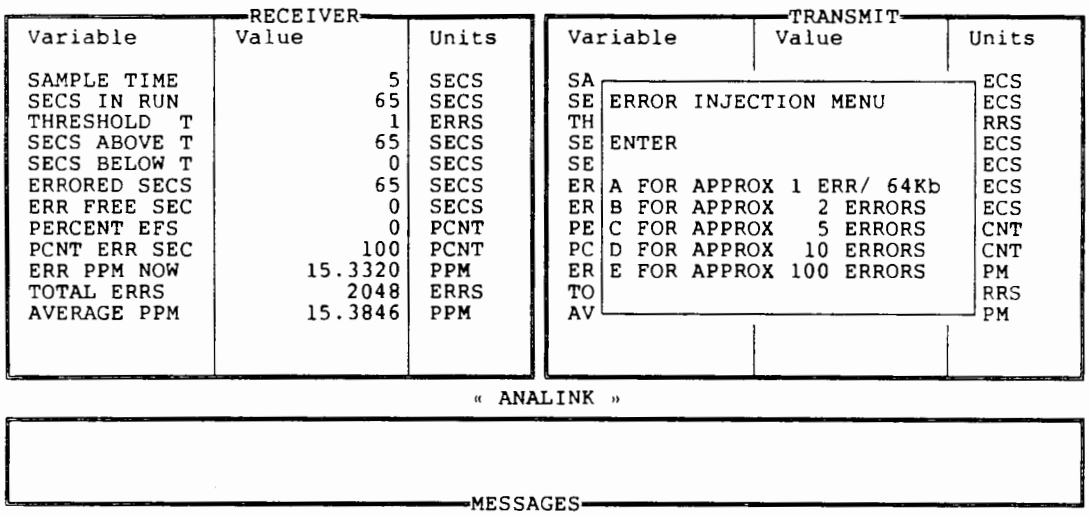


figure 6.6. Transmit window inject menu.

6.5. RS232 COMMUNICATION ROUTINES

As explained in chapter 5 the analyser must not only generate pseudo random test data, receive the test data and calculate and display the BER statistics for the 2 Mbps link being tested but also communicate the calculated statistics back to the transmitter where they will also be displayed. This configuration is shown in figure 6.7. In testing a duplex link this process is active in both directions at the same time.

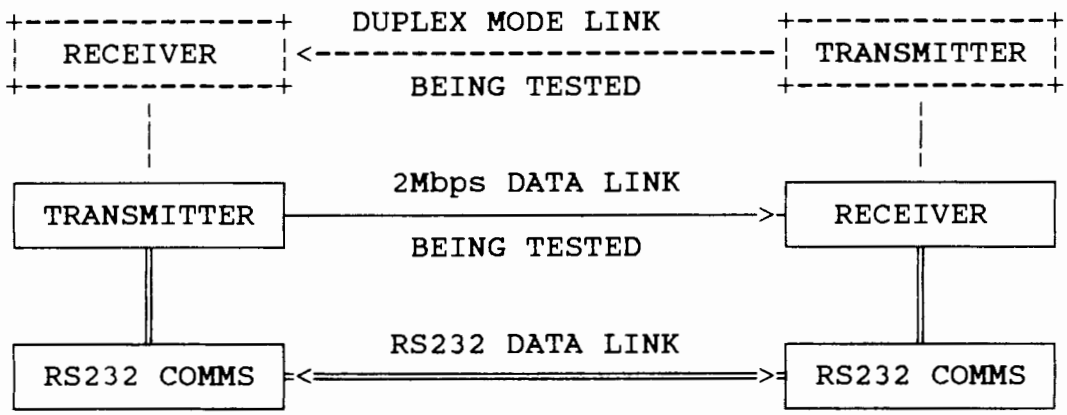


figure 6.7. Inter-System Communication Configuration

The three functions of the communications routines are initialisation, transmission and reception.

6.5.1 Initialisation

The initialisation of the serial communications card is done by this routine and the default communications format used at present is 2400 baud, no parity bit, one stop bit. However, the software is written so that the communications format can easily be changed.

The initialising routine also loads the address of the RS232 communications hardware interrupt service routine into the table of vectors in low memory and enables the interrupt, both on the communications card and on the 8259 communications controller.

The initialising routine is written in Assembly Language because low level programming is needed to access and program the serial communications chip on the RS232 card. It should be noted that the BIOS asynchronous communications routines available do not enable interrupts from the RS232 card.

The assembly language source code for the initialisation routine is listed in Appendix G as "init2". On exiting the program the vector table and interrupt status of the machine is restored to its original status by the "xit2" routine in the same appendix.

6.5.2. Transmitting

The RS232C based transmission of data or commands (to alter operating parameters e.g. such as error injection) from one analyser card to another is carried out by an Assembly Language routine. The complete message to be transmitted (in the message format shown in figure 6.9) is passed as two word parameters on the stack in the standard C parameter passing sequence as defined in the Microsoft C Manual [38].

The transmitting routine is called from every menu that passes commands to the other side of the link and also from the controlling statistical procedure which passes the error reading results as soon as they are available (once every second of measurement time).

The assembly language source code for the transmitting routine is listed in Appendix G as "trx_buf". The routine is called from menu routines in the main user interfacing module listed in Appendix C.

6.5.3. Receiving

The asynchronous communications receiver is programmed to interrupt on reception of a byte by the initialisation routine called from the main C module. The operation of the

hardware service interrupt routine servicing this interrupt is described in the section headed IRQ4 in this chapter and the assembly language source code is listed in Appendix G as "COMS_RD".

6.6. CONTROLLING ROUTINES

The control routines provide the direct access to the test unit for initialisation and changing of operating parameters and involve bit or byte manipulation of the data transferred.

The general reset latch is the address most often accessed by the control routines and the bit control of the register is broken up as in figure 6.8.

At start up the byte wide reset latch (or control register) is initially set to FF hex by default, as the data bus buffer output on the analyser card will be open circuit during the power up operation. The first operation is to ensure that the latch is reset to the all ones state.

The next task performed on startup is to test whether there is an analyser card in the system. The "dev_test" routine starts the transmitter clock on the card, enables the interrupt generating counter and continually reads the

counter to make sure it is counting. If it is not then the routine has a time-out after which a message is displayed in the message box on the screen informing the user of the fault.

BIT NO	CONTROL FUNCTION
0	TRANSMIT SHIFT REGISTER RESET
1	SYNCHRONISATION FLIP FLOP RESET
2	RECEIVER SHIFT REGISTER RESET
3	TRANSMIT CLOCK RESET
4	ERROR INJECTION RESET
5	INTERNAL LOOPBACK RESET
6	NOT USED
7	NOT USED

figure 6.8. Reset latch bit control

In the C global data definitions the variable used as the reset latch mask is declared as an unsigned char (i.e. one unsigned byte). The relevant byte values for setting specific control bits are all declared as constants in the main program header and these values are logically AND'ed or OR'ed with the current mask value to produce the required new byte value to be output to the reset latch by the general output port data transfer function outp().

The global latch mask is updated by either the menu functions listed in Appendix C or the RS232 communications receive ISR listed in Appendix G which updates the mask

according to information received from the opposite end of the link.

6.7. INTERRUPT SERVICE ROUTINES

There are two operating hardware interrupt requests generated by the extension cards in the system: IRQ3 (interrupt request 3) generated by the analyser card and IRQ4, generated by the asynchronous communications card.

6.7.1. IRQ3

IRQ3, generated by the timer chip on the extension card, is a request for the error counter on the analyser card to be read once every 10 mSec.

The value of 10 mSec is chosen on the basis of preventing overflow in the error counter from occurring when testing a 2 Mbps link. The hardware will ensure that every bit of test data received in error is counted.

If, as a worst case, every bit received is in error, the error counter will overflow every 30 mSec of real time. The solution of reading the counter every 10 mSec thus prevents overflow. The constant defining the 10 mSec count (which is

also used in calculations) can easily be changed to suit an 8 Mbps link should it be required.

Reading the error counters every 10 mSec also allows the error readings to be stored in the C data type of unsigned integer (which has a range of 0 to 65535 and is stored in 16 bits).

The advantage of using a data type of one 16-bit word, in the user interfacing C programs is that the data is accessed in one instruction cycle and an interrupt routine cannot corrupt the data as it is being written or read. Each complete value in the array of error readings is accessed within one instruction cycle and thus the pending interrupt cannot take control of the CPU until the instruction cycle is completed.

The IRQ3 service routine is written in assembly language for speed of processing. The code for the ISR, the routine which installs and enables the service routine in the system as the correct interrupt and the routine which returns the interrupt environment to its previous status on exit, is found in Appendix F.

6.7.2. IRQ4

This interrupt request is generated by the asynchronous communications card to signal the reception of a byte of data from the other end of the link.

The communications interrupt service routine (ISR) obtains the received byte and stores it in a buffer for analysis as the data sent can be either statistical data (calculated at the other end of the link) or commands sent from the other end of the link (pertaining to the operating parameters on this end of the link).

The communications ISR must thus determine the nature of the message and store the data or take the appropriate action.

Therefore, a scheme has been devised whereby the communications ISR receives bytes and stores them (passing control back to the controlling program) until the last byte in a message is received. Then, during the interrupt period while receiving the last byte of the message, the entire message is evaluated and the appropriate action is carried out before control is passed back to the user interfacing routines.

The format of the communications message is shown in figure 6.9.

BYTE 1	BYTE 2	BYTE 3	BYTE 4
ESCAPE	COM/DAT FLAG	MASK/VALUE	VALUE

figure 6.9. Message format

The escape character marks the beginning of a message and thus separates messages from one another. Thus, the pointer to the message received buffer is reset to point to the beginning of the buffer upon receiving this delimiter.

The next byte is a flag indicating whether the message contains data or a command.

On reception of the third byte after an escape character, i.e. the fourth byte of the message, the ISR looks at the flag byte to test whether the message is command or data.

case 1: data

If the flag indicates data then the next 2 bytes are the 16 bit value of the last error reading made by the receiver on the other end of the link being tested. This value is then

stored in the appropriate global variable on the transmitter end of the link.

case 2: command

If the flag indicates a command then the next byte is a mask byte for the reset latch of the analyser card (see chapter 6 for further details). The global mask variable is updated and stored and the new mask value is also output to the reset latch on the analyser card. The last byte in the message only has meaning if the command sent from the receiver end is to start the injection of errors into the pseudo random stream generated by the transmitter end. The final byte value is then a parameter indicating which value of initial count should be written to the error injection counter on the analyser card. The ISR thus writes the appropriate values to the 8254 registers and then returns control to the user interfacing program.

A full communications protocol with retransmission in the case of corruption etc. is not implemented as it was felt that the development of RS232 communications processing should not detract from the primary functioning of the unit. A more developed protocol might involve considerable delays in reception of data and the objective is that as little

effect as possible should be made on the real time processing and display of the bit error statistics.

Thus, if any characters are received and not recognised for whatever reason, then no action is taken for that particular message and the next ESC character resets the pointer to the message buffer to the beginning of the buffer once more. This can result in a loss of statistical accuracy in the case of a data message not being received (or being corrupted) but since explicit comparison (of received commands against known commands) is always carried out on the command bytes received, the chances of false commands being carried out is very low.

The IRQ4 service routine is written in assembly language for speed of processing and the code for the ISR, as well as the routine which installs and enables the service routine in the system as the correct interrupt, as well as the routine which returns the interrupt environment to its previous status on exit, is found in Appendix G.

CHAPTER 7

RESULTS AND TESTING

The results are broken up into three main sections namely: hardware verification of the analyser card, checking of the application software and overall testing of the system as a bit error rate analyser.

7.1. HARDWARE VERIFICATION

The prototype board is hardwired for the alternative sequence recommended earlier in this thesis and has only NRZ signals available at the output pins for testing. Basic verification of hardware operation was carried out using this card and observing the test waveforms on a logic analyser. A slow clock was used to clock out the pseudorandom sequence and verify it against a print out of the modelled sequence.

The basic development and testing environment is shown in figure 7.2 and consists of two personal computers (with two analyser cards as well as two RS232 cards) and a Hewlett Packard HP 1630G logic analyser with a Thinkjet printer.

figure 7.2. The development and testing environment.

The hardware operation of the analyser card was tested and the timing signals verified using a sample time of 10 nSEC on the logic analyser to check for glitches which could cause improper states to occur. Timing signal printouts were shown in chapter 6.

7.2. SOFTWARE CHECKING

Proof of the correct compilation and linking of the 5 modules is shown by the included MAP file of the executable code (i.e. the map of all the symbols in all the five source modules) in Appendix C.

The correct program operation is checked by the inclusion in Appendix H of program logging output which reflects the statistical values displayed on the screen and can be checked (see figure 7.3) against the mathematically deduced values for a known rate of error injection.

7.3. OVERALL FUNCTIONING

The bit error rate analyser is verified under three basic test environments with the results logged to disk over a period of 250 seconds in each case.

1) The analyser card is configured for internal loopback and tested with 3 different fixed rates of internal error injection.

2) The analyser card is configured for normal transmission and reception with external error injection circuitry and external loopback circuitry implemented.

3) The analyser card is configured for normal external duplex mode operation over a link in a test environment.

The log files list columns of values in the following order:

1) The number of seconds elapsed in the test run.

2) The preset threshold of errors per second above which errored seconds should be counted.

3) The number of seconds with above threshold errors which have occurred in the test run.

4) The number of error free seconds in the test run.

5) The ppm error rate for the previous sample period.

6) The total number of errors counted in the run.

7) The average ppm error rate over the entire run.

7.3.1. Internal Loopback

Appendix H lists the following log files: no errors injected, 1 injected error per 64 kbits, 2 errors per 64 kbits, and 5 errors per 64 kbits. These injection figures correspond to the first three available values of the error injection menu.

The most important figures presented in the tables in Appendix H are the total errors counted and the long term ppm average. The long term ppm average is a function of the

total number of errors counted, the number of seconds in the test run and the rate of data transmission.

Since error injection rates are at a power of two one must convert the injected error rate to the decimal parts per million statistical reading required.

1 error per 65536 bits equals a 15.2588 ppm error rate.

These values are compared in figure 7.3 to the corresponding long term averages read from the log files in Appendix H for ppm errors over a period of 250 seconds (523 288 000 bits).

ERRS	/64Kb	THEORETICAL PPM	MEASURED PPM	% ACCURACY
0	0	0.0000	0.0000	0.00 %
LO	1	15.2588	15.3730	1.46 %
	2	30.5176	30.8867	1.57 %
	5	76.2940	74.8750	1.86 %
HI	10	152.5880	148.3265	2.81 %
	100	1525.8800	1344.8457	11.90 %

figure 7.3. Comparison of theoretical ppm to measured.

As can be seen in figure 7.3, the measurements made are accurate to within two percent, with low injection of errors. This small inaccuracy results from the fact that although the value for the total count of errors that have occurred is correct, the timing method of the software

(which updates the statistics once a second) is only accurate to approximately one percent. The values output to the screen (and disk) are a function of elapsed time as well as total count and at low error injection the timing inaccuracy accounts for most of the reading inaccuracy.

The measurements become less accurate as the injected error count becomes unrealistically high because the hardware is experiencing difficulty in synchronising to the incoming stream 100 % of the time. When the hardware does experience a problem synchronising no error measurements are made until resynchronisation occurs, thus intervening errors are not counted. Resynchronisation occurs 64 times per second, thus in the long term the error reading of higher error rates, while not being highly accurate, does give a general quantitative assessment of the errors on the channel.

The hardware can only fail to notice an error if it occurs on the last bit of the synch word which has a statistical probability of occurrence of 1 in 32767 (for 1 injected error per 65536 bits) which is very much less than the 1.46% inaccuracy recorded. Furthermore, since an error counted by hardware cannot fail to be noted by the software, the reason for the inaccuracy must be a timing one. This can be traced to the manner in which the software times the calculations (i.e. the timer tick).

The software has a minimum display time of 1 sec, thus the statistics are calculated once a second and displayed at a multiples of one second interval. Since the system real time clock is 'ticked' 18.2 times a second, when waiting for the seconds counter to increment through a BIOS routine, there are 18 sub-increments to a one second interval. There is thus a timing error induced of 0.2 / 18.2 which is 1.1 percent.

In addition to the timing error induced by the real time clock there is also a slight error induced by the fact that the statistics generating procedure is interrupted by the error reading and communications interrupts which adds a small amount of timing inaccuracy to the calculations.

7.3.2. External Loopback

In the external loopback mode the stream of data is externally inverted by an asynchronous pulse generated from the output of a counter whose clock frequency bears no relationship to the data clock. The circuit is a replica of the error injection circuitry on the analyser card except that the clock is derived from a separate source. The frequency is, however, lower than the data rate and thus the section of the stream that is inverted (the data is inverted to give a very definite error) is always longer than 1 bit.

The logged output is given in appendix H.3 for two rates of error injection, one being twice the rate of the other. The first is 1 clock period of a 894 886.25 Hz clock inversion per 52248 periods and the second is twice this amount.

These rates of inversion correspond to a theoretical error rate of 19.1395 ppm and 38.2790 ppm respectively although inversions can possibly occur in the middle of the data bit and thus cause glitch errors, leading to a small amount of inaccuracy.

For 1 injected error per 52248 (at 894886.25 Hz) the measured error rate of 18.8047 shown in Appendix H differs from the theoretical prediction of 19.1394 by 1.55 %.

For 2 injected errors per 52248 (at 894886.25 Hz) the measured error rate of 37.6855 differs from the theoretical prediction of 38.2790 by 1.75 %.

The external loopback error injection circuit was also connected to a commercial test unit although it was unfortunately not possible to obtain a print out of logging from the commercially available test unit but visual quantitative and qualitative assessments were carried out.

Comparing the performance of the commercial unit against that of the unit developed, the approximate accuracy characteristic shown in figure 7.3 was obtained.

The commercial unit has a tendency to over-read with high injection of errors and will read full scale before the entire bit stream is inverted. The unit developed, however under-reads with high injection of errors as it cannot synchronise to more and more sequences. When the entire bit stream is inverted the unit developed will read zero because it cannot synchronise to any sequences at all and the error counter remains permanently gated.

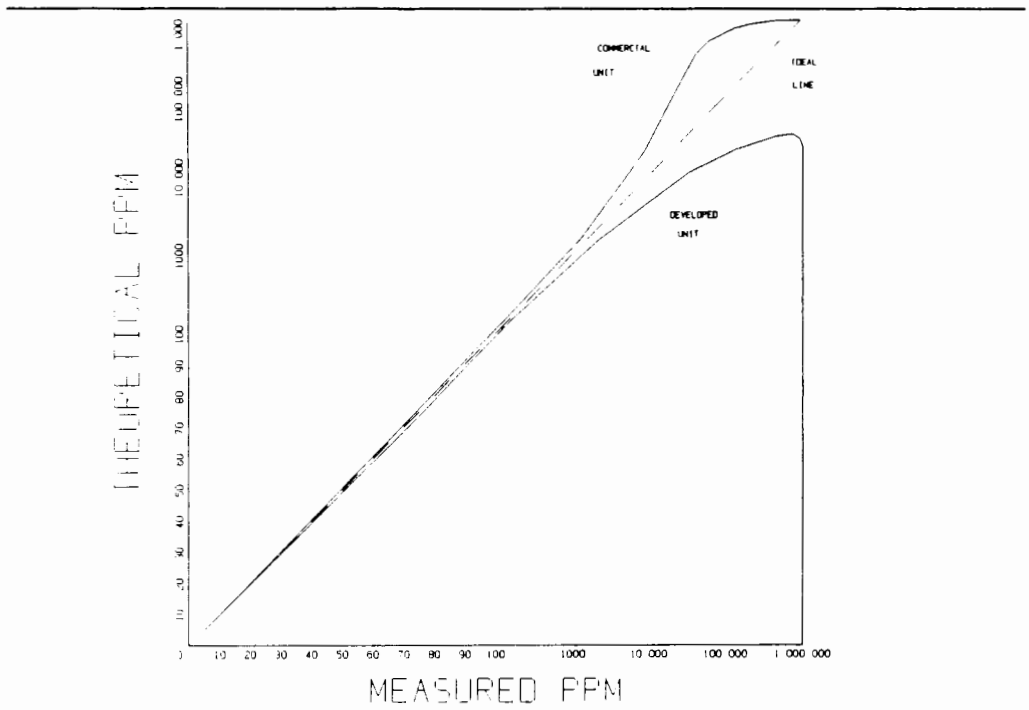


figure 7.3. Approximate measurement accuracies.

7.3.3 Normal Duplex Testing

The tester was connected to a 2 Mbps prototype link and compared to a commercially available tester testing the performance of the link in the laboratory environment.

Two sets of results are given in Appendix H.5, the first is the bit stream inadvertently inverted by the link data conditioning circuitry and the second is normal operation from two test receiver/transmitters with small horn antennae being utilised and a sheet of non-homogeneous absorptive/reflective foam between them (to simulate absorptive and scattering effects).

In the first instance of the bit stream being inverted the commercially available test unit gave a reading of exactly 1 000 000 parts per million being in error whilst the test unit developed gave a reading of zero. The reading of zero is explained by the fact that the test unit developed could not ever synchronise to the incoming stream as there was never a synchronisation word received, only a logically inverse synchronisation word. The reading of 1 000 000 ppm on the commercially available test unit, while being perfectly correct in this instance, is strange in that it would be illogical to assume that the commercial unit can sense that the entire bit stream is inverted. One assumes,

therefore, that the unit has been programmed to give a full scale reading of 1 million parts per million when the unit cannot synchronise to the incoming stream after a certain length of time.

Programming to implement the above feature could easily be included in the developed unit, provided that the appropriate hardware change is made to provide a sense bit that can be read by the software to confirm that data are being received at all. Such a sense bit could be taken from the MV1441 LOSS OF INPUT pin (see Appendix L) and mapped to an I/O address on the analyser card.

In the second instance of normal, low error reading operation the unit developed compared very favorably with the commercially available unit (in terms of error readings) with the advantage that all the possible display formats of the commercially available unit and more were present, all at the same time, on the unit developed. The error count of the link being analysed was low (due to the unit under test communicating over a short distance only) and thus the difference between the error count of the two units differed by approximately 1 % which was to be expected if the initial assumed timing inaccuracy of the developed unit is correct.

CHAPTER 8

CONCLUSIONS

In chapters 1 and 2 objectives and features of the error rate analyser are outlined and the chapters 3 to 7 explain the methods used to achieve them. From the work covered in these chapters and the results obtained in chapter 8 a number of conclusions can be drawn.

8.1. GENERAL CONCLUSIONS

Pseudo Random sequence generators can be modelled algebraically and characteristics can be predicted from the theory in terms of statistical properties (comparable to those of true random sequences) as well as physical properties such as the power spectrum of a pseudo random bit sequence.

The output from a pseudo random sequence generator can be simulated in a software program and characteristic patterns of the output sequence pattern can easily be observed.

From the simulation program output and references on current thinking on the subject, one can deduce that sequences exist which are better (in terms of being more random or more

suitable for use on certain links) than the CCITT recommended sequences.

The open architecture of the IBM Personal Computer facilitates the development of a compact and flexible piece of test equipment on an extension card. The test equipment can furthermore be produced for less than a tenth of the cost of a commercially available one.

8.2. DEVELOPMENT CONCLUSIONS

All the features selected from the list of features proposed by Feher and set out in chapter 2 are implemented and found to be useful in a bit error rate test instrument.

The test unit produces test data which conforms to the content and physical signal specifications of CCITT recommendation G.703, although the alternatives of TTL level outputs and different pseudo random sequences are made available.

The test unit developed compares well against a commercially available test unit in terms of ease of use and has the advantage of greater flexibility in terms of readout and logging facilities. Commercially available units have greater flexibility with respect to changing operating

frequency and corresponding sequence length compared to the test unit in its present stage of development.

An important shortfall of commercially available test units working in very noisy environments is that the accompanying timing glitches affect synchronisation and give erroneous readings. The test unit developed does, however, have one degree of protection against loss of synchronisation in that the resynchronisation used is a twofold action and the error counter is actually disabled if both phases of the resynchronisation are not received.

The unit developed has an accuracy characteristic which tends to under-read on links with very high error counts as opposed to the over-read of a commercial unit studied.

At low error statistics (less than about 200 ppm) the unit developed performs to within 2% of the theoretical figures and thus provides an accurate and cheap alternative to currently available commercial test equipment.

8.3. EXTENSIONS

Possible extensions to the project fall under two sections: software and hardware.

8.3.1. Software Extensions

The software as it stands is very versatile and user friendly. Possible improvements in the accuracy of the test unit would require additional changes to the hardware.

Firstly a precision timing algorithm could be implemented to ensure measurements are made at more exact time intervals than the 1% inaccuracy of the current routines.

Secondly a routine could be included to look for the signal condition of invalid HDB3 signal received or loss of signal input to the HDB3 codec chip. This would allow one to exclude the errors caused by these conditions from the error count which should reflect only the errors induced by the link itself.

Thirdly the one possible condition of false resynchronisation could be excluded by having a controlling routine look for a sudden large jump in the error count (within one hardware interrupt period) with no accompanying sudden loss of received signal on the MV1441.

8.3.2. Hardware Extensions

In hardware terms the most important extension would be to have a highly stable, temperature compensated oscillator with software controllable, calibrated jitter inducing capabilities for testing the clock recovery and bit synchronisation capabilities of a link. To facilitate this extension the software would have to be altered in the main user interfacing and menu modules as well as the communications module if remote jitter control is found to be necessary.

Another important possible extension is to have software switchable data rates (with corresponding switch in sequence configuration to the appropriate CCITT recommended one). The board as it stands is easily alterable to work at different data rates with corresponding small changes in the software.

The biggest obstacle to implementing switchable data rates is the question of output interfaces as the CCITT recommends different output standards for the different data rates and therefore all the necessary different output circuits would have to be present on the board which could be expensive and wasteful on space.

It would therefore be simpler, if one was implementing a variable data rate tester, to have only TTL outputs accessible and assume that the user would implement the user's own signal conditioning necessary for testing the link under examination. Otherwise one would have software switchable signal conditioning circuitry implementing each particular standard.

LIST OF REFERENCES

[1] G.C. Hartley, "PCM and Digital Networks" in Advanced Communication Systems, ed B.J. Halliwell, London: Newnes Butterworths, 1974, pp 102-103.

[2] Wayne Tomasi, Advanced Electronic Communication Systems, New Jersey: Prentice Hall, 1987, pp 172-174.

[3] T.W. Adams, "Data Services" in Telecommunications Networks, ed J.E. Flood, London: Peter Peregrinus, 1975, pp 247-248.

[4] CCITT Recommendation G.703, Yellow Book, Vol 3 Fascicle III.2, pp 47-50.

[5] K. Feher, Telecommunications Measurement, Analysis and Instrumentation, USA: Hewlett Packard, p 140.

[6] K. Feher, Telecommunications Measurement, Analysis and Instrumentation, USA: Hewlett Packard, pp 150-151.

[7] CCITT Recommendation O.151, Yellow Book, Vol 4 Fascicle IV.4, pp 87-91.

[8] CCITT Recommendation G.703, Yellow Book, Vol 3 Fascicle III.2, pp 48-49.

[9] CCITT Recommendation G.703, Yellow Book, Vol 3 Fascicle III.2, pp 49-50.

[10] CCITT Recommendation G.703, Yellow Book, Vol 3 Fascicle III.2, pp 48-49.

[11] CCITT Recommendation G.703, Yellow Book, Vol 3 Fascicle III.2, pp 49-50.

[12] Paul Horowitz and Winfield Hill, The Art of Electronics, Cambridge University Press, 1980, pp 406-407.

[13] CCITT Recommendation O.151, Yellow Book, Vol 4 Fascicle IV.4, pp 87-91.

[14] see reference [5].

[15] CCITT Recommendation G.703, Yellow Book, Vol 3 Fascicle III.2, Appendix 1, p 61.

[16] IBM Technical Reference, Florida, IBM, 1983, p 1-15.

[17] CCITT Recommendation G.703, Yellow Book, Vol 3 Fascicle III.2, p 48.

[18] Robert J. Matthys, Crystal Oscillator Circuits, New York: Wiley Interscience, 1983, pp 162-168.

[19] Robert J. Matthys, Crystal Oscillator Circuits, New York: Wiley Interscience, 1983, pp 201-206.

[20] IBM Technical Reference, Florida, IBM, 1983, p E-1.

[21] CCITT Recommendation G.703, Yellow Book, Vol 3 Fascicle III.2, p 50.

[22] CCITT Recommendation G.703, Yellow Book, Vol 3 Fascicle III.2, p 48.

[23] CCITT Recommendation G.703, Yellow Book, Vol 3 Fascicle III.2, p 48.

[24] CCITT Recommendation G.703, Yellow Book, Vol 3 Fascicle III.2, p 50.

[25] CCITT Recommendation G.703, Yellow Book, Vol 3 Fascicle III.2, Annex B, p 60.

[26] CCITT Recommendation G.703, Yellow Book, Vol 3 Fascicle III.2, Annex A, p 60.

[27] P-CAD USERS MANUAL, USA: Personal CAD Systems, 1984.

[28] DOS 3.20 Manual, New York: Microsoft Press, 1986.

[29] Peter Norton, A Programmers Guide to the IBM PC, New York: Microsoft Press, 1985, pp 242,243,17.

[30] Turbo-C Manual, USA: Borland International, 1987.

[31] Microsoft C Compiler Manual, USA: Microsoft, 1986.

[32] Microsoft Macro Assembler Users Guide, USA: Microsoft, 1986, pp 13-42.

[33] Microsoft Macro Assembler Users Guide, USA: Microsoft, 1986, pp 43-72.

[34] Microsoft Codeview Manual, USA: Microsoft, 1986.

[35] C-Tools Plus Manual, Berkeley: Blaise Computing, 1987.

[36] Microsoft C Compiler Users Guide, USA: Microsoft, 1986 pp 219,220.

[37] Microsoft C Compiler Users Guide, USA: Microsoft, 1986 p 219.

[38] Microsoft C Compiler Users Guide, USA: Microsoft, 1986 p 220.

[39] William W. Peterson, Error correcting codes, Cambridge Mass: MIT Press, 1972, pp 19-160.

[40] William W. Peterson, Error correcting codes, Cambridge Mass: MIT Press, 1972, pp 182-183.

[41] William W. Peterson, Error correcting codes, Cambridge Mass: MIT Press, 1972, pp 183-184.

[42] S.W. Golomb, Shift Register Sequences, London: Holden Day, 1967.

[43] William W. Peterson, Error correcting codes, Cambridge Mass: MIT Press, 1972, pp 222-223.

[44] Paul Horowitz and Winfield Hill, The Art of Electronics, Cambridge University Press, 1980, p 441.

[45] Charles H. Vincent, Random Pulse Trains, London: Peter Peregrinus, 1973, pp 211-213.

[46] Paul Horowitz and Winfield Hill, The Art of Electronics, Cambridge University Press, 1980, pp 441-442.

[47] Charles H. Vincent, Random Pulse Trains, London, Peter Peregrinus, 1973, p 213.

[48] Paul Horowitz and Winfield Hill, The Art of Electronics, Cambridge University Press, 1980, p 443.

[49] Paul Horowitz and Winfield Hill, The Art of Electronics, Cambridge University Press, 1980, p 446.

[50] Charles H. Vincent, Random Pulse Trains, London, Peter Peregrinus, 1973, p 213.

[51] Peterson, pp 472-490.

[52] Peterson, p 480.

BIBLIOGRAPHY

1. CCITT Recommendation G.703. Yellow Book. Vol 3 Fascicle III.2.

2. CCITT Recommendation O.151. Yellow Book. Vol 4 Fascicle IV.4.

3. C Tools Plus manual. Berkeley: Blaise Computing. 1987.

4. Disk Operating System Technical Reference. Florida: IBM. 1985.

5. Eagle PC-35 handbook. Cape Town: Eagle Electric. 1986.

6. Eggebrecht, Lewis C. Interfacing to the IBM Personal Computer. Indianapolis: Macmillan. 1983.

7. Feher, K. Telecommunications Measurement, Analysis and Instrumentation. USA. Hewlett Packard.

8. Flood, J.E. ed. Telecommunication Networks. London: Peter Peregrinus. 1975.

9. Freeman, Roger L. Telecommunication Transmission Handbook. New York: Wiley Interscience. 1981.

10. "Fundamental Limits of Jitter Tolerance In digital transmission systems." T. Crawford, G. Thow and P. Scott. Hewlett Packard Telecommunications symposium paper. 1983.

11. Halliwell, B.J. ed. Advanced Communication Systems. London: Newnes Butterworths. 1974.

12. Golomb, S.W. Shift Register Sequences. London. Holden Day. 1967.

13. Horowitz, Paul and Winfield Hill. The Art of Electronics. Cambridge University Press. 1980.

14. IBM PC Technical Reference. Florida: IBM. 1983.

15. IBM PC AT Technical Reference. Florida: IBM. 1984.

16. Kernighan, Brian W and Ritchie, Dennis M. The C Programming Language. New Jersey: Prentice Hall. 1978.

17. Kruglinski, David. A Guide to IBM PC Communications. Berkeley: McGraw Hill. 1984.

18. Matthys, Robert J. Crystal Oscillator Circuits. New York. Wiley Interscience. 1983.

19. McDowell, Ron. "Jitter Measurements in the Integrated Digital Network". Hewlett Packard Telecommunications Symposium paper. 1986.

20. McDowell, Ron. "Jitter Measurements in the Integrated Digital Network". Hewlett Packard Telecommunications Symposium paper. 1987.

21. Microsoft C manual. USA. Microsoft. 1983.

22. Microsoft Codeview manual. USA. Microsoft. 1983.

23. Microsoft Disk Operating System 3.20 manual. USA. Microsoft. 1987.

24. Microsoft Macro Assembler 4.0 manual. USA. Microsoft. 1987.

25. Microsystem Components Handbook vol 1 and vol 2. USA. Intel. 1986.

26. Norton, Peter. The Peter Norton Programmers Guide to the IBM PC. Washington: Microsoft Press. 1985.

27. Norton, Peter. Inside the IBM PC. New York: Robert J. Brady. 1986.

28. Peterson, W.W. Error correcting codes. New York. Wiley Interscience. 1961.

29. Peterson, William W. Error Correcting Codes. Cambridge Mass: MIT Press. 1972.

30. Stearns, S.D. Digital Signal Analysis. New Jersey: Hayden Book Company. 1975.

31. Tomasi, Wayne. Advanced Electronic Communications Systems. New Jersey: Prentice Hall. 1987.

32. TTL handbook. USA: National Semiconductor. 1986.

33. Turbo-C manual. USA: Borland International. 1987.

34. Turbo-Pascal 4.0 manual. USA: Borland International. 1988.

35. Vincent, C.H. Random Pulse Trains. London. Peter Peregrinus. 1973.

APPENDIX A

ALGEBRAIC BACKGROUND

The following algebraic systems form a background to the theory of pseudo random sequence generators and they follow the definitions which are condensed and adapted from Petersen [39].

1) A group is a system with one operation and its inverse defined.

2) A ring has two operations with an inverse operation for the first defined.

3) A field has two operations both with inverses defined.

A.1. GROUPS

Certain axioms must hold for a set of elements to be defined as a group.

1) (Closure) The operation defined can be applied to any 2 group elements to give a result which is also a group element.

2) (Associative law) For any three elements of the group the operation defined is associative.

3) There is an identity element.

4) Every element of the group has an inverse element.

If the group satisfies the commutative law as well the group is called Abelian or commutative.

A.2. RINGS

The following axioms must hold for a set of elements to be classified as a ring:

1) The set R is a commutative group under the first operation (this is normally termed addition whatever form it might take).

2) (Closure) For any two elements of R the second operation (which is usually termed multiplication) defines a third element of R.

3) (Associative law) For any three elements of R the second operation is associative.

4) (Distributive law) For any three elements of R the second operation is distributive over the first (multiplication is distributive over addition).

A.3. FIELDS

A field is defined as a ring where the second operation is commutative, there is a unit element and every nonzero element has an inverse.

A.4. VECTOR SPACES

A set V of elements is called a vector space if the following axioms hold (where field elements are called scalars and elements of V are called vectors):

1) The set V is a commutative group under addition.

2) For any vector v in V and any scalar c the product cv is a vector in V.

3) (Vector distributive law) If u and v are vectors in V and c is a scalar $c(u + v) = cu + cv$.

4) (Scalar distributive law) If v is a vector and c and d are scalars $(c + d)v = cv + dv$.

5) (Associative law) If v is a vector and c and d are scalars $(cd)v = c(dv)$.

Theorem 1

If a vector space V1 is contained in a vector space V2 and they have the same dimension k they are equal.

Proof

A basis for V1 is a set of k linearly independent vectors in V2. But the set of k linearly independent vectors in V2 is clearly a basis for V2. Therefore every vector in V2 is also in V1

A.5. IDEALS

An ideal I is a subset of elements of a ring R with the following properties:

1) I is a subgroup of the additive group of R.

2) For any element a of I and r of R, ar and ra are in I.

As an example in the ring of all positive and negative integers and zero, the set of all multiples of any particular integer is an ideal.

A.6. RESIDUE CLASSES

Since an ideal is a subgroup cosets can be formed, and with respect to rings these cosets are called residue classes. An ideal forms the first row of an array, then any element not in the ideal can be chosen as the leader of the first residue class on the next row below the ideal. The rest of this residue class is formed by adding the leader of this row to the corresponding element of the ideal above. The first element in each row must be a previously unused element and the row (or residue class) is built up as above.

The notation of residue classes is that $\{r\}$ is defined as the residue class containing the element r.

Following this notation addition and multiplication of residue classes can be defined as follows:

$$\{r\} + \{s\} = \{r + s\}$$

$$\{r\}\{s\} = \{rs\}$$

Since addition and multiplication are defined in this way for any pair of residue classes the residue classes of a ring with respect to an ideal form a ring and are called the residue class ring.

A few important theorems concerning residue classes and the terminology associated with them will be mentioned here.

Theorem 2

A set of integers is an ideal if and only if it consists of all multiples of some integer.

The ideal formed by taking all positive multiples of an integer m is denoted (m) and the residue class ring formed of residue classes of (m) is called the ring of integers modulo m.

Theorem 3

The residue class ring modulo m is a field if and only if m is a prime number.

The fields thus formed are known as prime or Galois fields of p elements and denoted GF(p).

A.7. POLYNOMIAL IDEALS

If one considers polynomials f(x) with one variable and with coefficients from any field F, the general form of the polynomials is:

$$f(x) = f_0 + f_1x + f_2x^2 + ... + f_nx^n.$$

The degree of a polynomial is the largest power of x in a term with nonzero coefficient. A polynomial is called monic if the coefficient of the highest power of x is 1. A polynomial of degree n which is not divisible by any polynomial of degree less than n but is greater than 0 is termed irreducible.

As in Theorem 2 with integers, a set of polynomials is an ideal if and only if it consists of all multiples of the same polynomial.Residue classes of polynomials can thus be formed and since addition and multiplication can be defined for polynomial residue classes (in the same way as in integers) we can conclude that:

Theorem 4

The residue classes of polynomials modulo a polynomial f(x) of degree n form a commutative linear algebra of dimension n over the coefficient field.

The notation for residue classes is typically that the symbol S is used to represent the residue class containing x, while the residue class containing a field element is the same representation as the field element.

Thus the coset that contains $a(x) = a_0 + a_1x + .. + a_{n-1}x^{n-1}$ is :

$$\{a_0 + a_1x + \ldots + a_{n-1}x^{n+1}\}$$

$$= \{a_0\} + \{a_1\}\{x\} + \ldots + \{a_{n-1}\}\{x\}^{n-1}$$

$$= a_0 + a_1S + \ldots + a_{n-1}S^{n-1} \ .$$

Thus every residue class equals a polynomial of degree less than n in S.

The definition of the multiplication of polynomials of degree n in the field is as follows:

Let $a(S) = a_0 + a_1S + \ldots + a_{n-1}S^{n-1}$;

$$b(S) = b_0 + b_1S + \ldots + b_{n-1}S^{n-1} .$$

Then the coefficient of S^j in $c(s) = a(s)b(s)$ is

$$c_j = a_0b_j + a_1b_{j-1} + \ldots + a_jb_0 + a_{j+1}b_{n-1} + a_{j+2}b_{n-2} +$$

$$\ldots + a_{n-1}b_{j+1} .$$

Where the terms involving $a_{j+1},\ldots,a_{n-1}$ come from the terms in the product of a(s) and b(s) which involve S^{n+j}, but since $S^n - 1 = 0$, $S^{n+j} = S^j$.

A definition of a null space is that a polynomial is said to be in the null space of an ideal J if $r(S)s(S) = 0$ for every polynomial s(S) in J. This is followed by an important theorem:

Theorem 5

Let f(x), g(x) be monic polynomials and let $f(x) = g(x)h(x)$. Then (a(x)) is in the null space of the ideal generated by h(x) if and only if it is in the ideal generated by g(x)

A.8. GALOIS FIELDS

Let p(x) be a polynomial with coefficients in a field F. If and only if p(x) is irreducible in F, that is, if and only if p(x) has no factors with coefficients in F, then the algebra of polynomials over F modulo p(x) is a field.

It has been stated above, in Theorem 3, that residue classes of integers modulo any prime number p form a field of p elements called the Galois field GF(p). It can be shown that the ring of polynomials over any finite field has at least one irreducible polynomial of every degree. The field of polynomials over GF(p) modulo an irreducible polynomial of degree m is called the Galois field of p^m elements or $GF(p^m)$.

This Galois field is a vector space of dimension m over GF(p) and hence has p^m elements. For any number $q = p^m$ that is a power of a prime number therefore there is a field GF(q), which has q elements.

A.9. ALGEBRAIC MODELLING OF FSR

According to Peterson [40] a feedback shift register can be modelled by the following recurrence relation or difference equation:

$$\Sigma_{j=0}^{k} h_j a_{i+j} = 0 \qquad (1)$$

or in terms of determining a_k from the values of $a_0, a_1, \ldots, a_{k-1}$

$$a_{i+k} = - \Sigma_{j=0}^{k-1} h_j a_{i+j} \qquad (2)$$

where h_0 is not = 0 and $h_k = 1$, and each h_j is an element of GF(q).

Because the equations are linear, any linear combination of solutions is a solution, and the solutions form a vector space.

The k solutions for which one of the symbols $a_0, a_1, \ldots, a_{k-1}$ is 1 and the rest are 0 span the space and therefore the space of solutions has dimension no greater than k. But since $a_0, a_1, \ldots, a_{k-1}$ are arbitrary the space has dimension at least k. Therefore the dimension is exactly k.

The solutions of the above difference equation are characterised in Theorem 6

Theorem 6

Let $h(x) = \Sigma_{j=0}^{k} h_j x^j$, h_0 not = 0, $h_k = 1$

and let n be the smallest positive integer for which $x^n - 1$ is divisible by h(x). Let $g(x) = (x^n - 1) / h(x)$, then the solutions of equation (1) above are periodic of period = n and the set made up of the first period of each possible solution, considered as polynomials modulo $(x^n - 1)$

$$a(x) = a_0 x^{n-1} + a_1 x^{n-2} + \ldots + a_{n-2} x + a_{n-1}$$

is the ideal generated by g(x) in the algebra of polynomials modulo $(x^n - 1)$.

A.10. MODELLING

A.10.1. THE SHIFT REGISTER GENERATOR

According to Peterson [41] a feedback shift register can be modelled by the following recurrence relation or difference equation:

$$\Sigma_{j=0}^{k} h_j a_{i+j} = 0 \qquad (1)$$

or in terms of determining a_k from the values of $a_0, a_1, \ldots, a_{k-1}$

$$a_{i+k} = - \Sigma_{j=0}^{k-1} h_j a_{i+j} \qquad (2)$$

where h_0 is not = 0 and $h_k = 1$, and each h_j is an element of GF(q).

Because the equations are linear, any linear combination of solutions is a solution, and the solutions form a vector space.

The k solutions for which one of the symbols $a_0, a_1, \ldots, a_{k-1}$ is 1 and the rest are 0 span the space and therefore the space of solutions has dimension no greater than k. But since $a_0, a_1, \ldots, a_{k-1}$ are arbitrary the space has dimension at least k. Therefore the dimension is exactly k.

The solutions of the above difference equation are characterised in Theorem 6. From this theorem one can deduce that some of the solutions may have period less than n (which is = 2^K-1), but there must be some that do not. In particular the solution obtained from (g(x)) has period exactly n.

In what one may take as a practical corollary, Golomb [42] states that the length of an output sequence of a linear feedback shift register circuit is the smallest positive integer n such that $(x^n - 1)$ is mod 2 divisible by the characteristic equation of the feedback configuration. If this smallest integer is $2^m - 1$ (where m is the number of stages in the shift register) then the sequence is referred to as a maximum length sequence. This follows because there are 2^m possible states that a binary m stage register can exist in, and the state of all zeroes is a self-perpetuating cyclic solution of one state (in a linear feedback configuration).

Mathematically the above can be expressed as follows as in Petersen [43].

If h(x) is the polynomial of degree k corresponding to the feedback connections then by the theorem proved above the period of the sequences is the smallest n such that h(x) divides $(x^q - 1)$, (where $q = p^m - 1$) and hence $n <= p^m - 1$.

But if h(x) is irreducible and a primitive polynomial h(x) divides $(x^n - 1)$ for no smaller value of n and therefore the period of the sequence produced is exactly $p^m - 1$.

Thus if the characteristic equation of a m stage linear feedback shift register is irreducible and primitive then the sequence produced is a maximum length sequence and the length of the sequence is equal to $2^m - 1$.

The set of vectors that are the first periods of all the solutions forms an ideal and hence a cyclic code. The nonzero solution with the most leading zeroes must be the generator polynomial of the code

$$\{(x^q - 1) / h(x)\} \quad \text{where } q = p^m - 1.$$

The vector contained in the shift register consists of m successive elements of the sequence. Since the contents of the shift register must be different for each element in one period, each nonzero vector of m components must appear in m successive positions of a maximum length sequence in one and only one place.

Finally in the set of all m component vectors every field element appears in 1/p of the p^m positions that is mp^{m-1} positions.

If the zero vector is omitted 0 appears only $m(p^{m-1} - 1)$ times. Since every m component vector appears once and only once in a maximum length sequence and in this way each

element of the sequence is counted m times, every nonzero element appears (in one period of a maximum length sequence) p^{m-1} times and zero appears $p^{m-1} - 1$ times.

In the case of binary numbers (i.e. where p = 2) this leads one to deduce the following statistical properties of maximum length sequences:

A.10.2. CHARACTERISTICS

1) A sequence generated by a m stage register contains 2^{m-1} ones and $(2^{m-1} - 1)$ zeroes.

2) Every possible sequence of m bits except all zeroes occurs exactly once in a period.

3) In every cycle there are exactly 2^{m-1} runs of all zeroes or all ones. Half of these runs have length 1, a quarter length 2, etc. with equal numbers of runs of zero and one in each case. This rule applies down to one run of zeroes of length m-2 and one run of ones of the same length. One run of zeroes of length m-1 and one run of ones of length m complete the list.

4) The autocorrelation function of the sequence is unity at zero displacement (or any number of complete periods distance) and has a small constant value elsewhere of

$$-1 / (2^m - 1)$$

The autocorrelation function is defined here as

$$c(n) = (4/q) \sum_{r=1}^{p} (a_r - 1/2) (a_{r+n} - 1/2)$$

where $q = 2^m - 1$, the a_r are the respective q bits of the sequence and where (r + n) is to be understood as (r + n) mod q.

The normalised autocorrelation function according to Horowitz [44] is shown in figure A.10.1 and is defined for an integer number of clock shifts only. For shifts that are not zero or a multiple of the sequence period q the function is a constant -1 value (due to the extra one in the sequence).

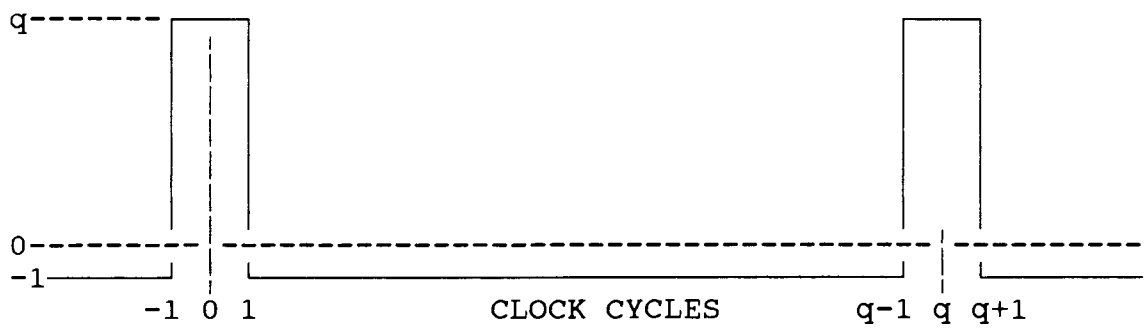


figure A.10.1. Full cycle discrete autocorrelation for PRBS

A.10.3. TRUE RANDOM SEQUENCES

The properties listed above for maximum length sequences have a striking correspondence to those of truly random sequences which are as follows, as stated by Vincent [45], for any q sequence length:

1) There are equal probabilities of 0 and 1 digits.

2) There are equal probabilities for any specific sequence of k digits.

3) The probabilities for runs of all ones or all zeroes of different length in a truly random sequence are very similar to those above in that the probability of the run occurring is inversely proportional to the length of the run.

4) Without the cyclic qualification, the autocorrelation function of a truly random sequence is very similar to the above figure A.10.1, being unity at zero displacement and zero expected value with $p^{-1/2}$ standard deviation elsewhere (as can be seen in figure A.10.2).

The truly random stream used in the discrete autocorrelation function (shown in figure A.10.2) is defined as a truly random sequence of bits clocked at a uniform rate and thus the correlation is only defined for an integer number of clock cycle shifts.

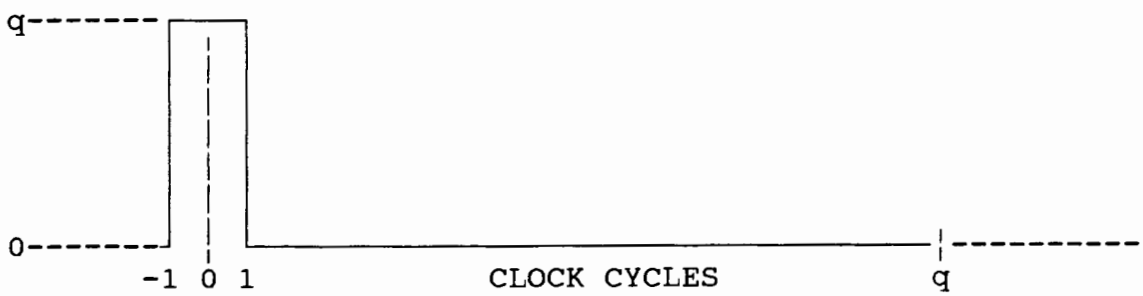


figure A.10.2. Discrete autocorrelation for a true random sequence

A.10.4. POWER SPECTRUM

If we consider the output of the sequence generator to be an analog signal whose value corresponds to +a or -a volts only the normalised continuous autocorrelation function is similar to figure A.10.1 taken from Horowitz [46] but the

shape of the peaks is triangular instead of being rectangular at q clock period spacings, as can be seen in figure A.10.3.

The power spectrum of this triangular shaped autocorrelation function has a well known power spectrum, shown in figure A.10.4, and consists of a series of delta functions spaced at f_{clock}/q and with an amplitude envelope governed by the function

$$P(f) = \frac{\sin^2 (\pi . f_{clock}/q)}{(\pi . f_{clock}/q)^2}$$

This function has the property that there is no power at the clock frequency or its harmonics. Thus a pseudo random sequence clocked out at f_{clock} has no power at f_{clock} or its harmonics.

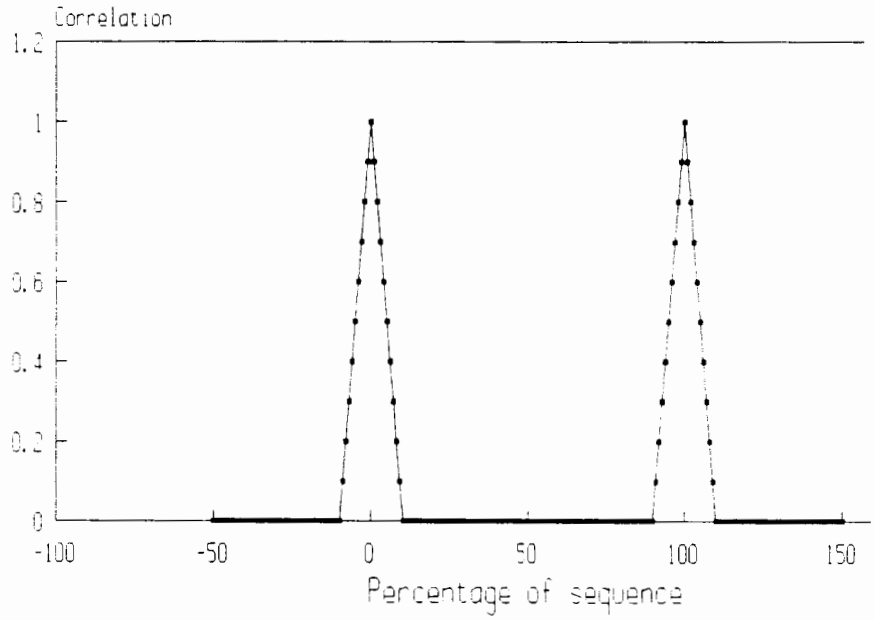


figure A.10.3. Continuous Autocorrelation of PRBS Sequence.

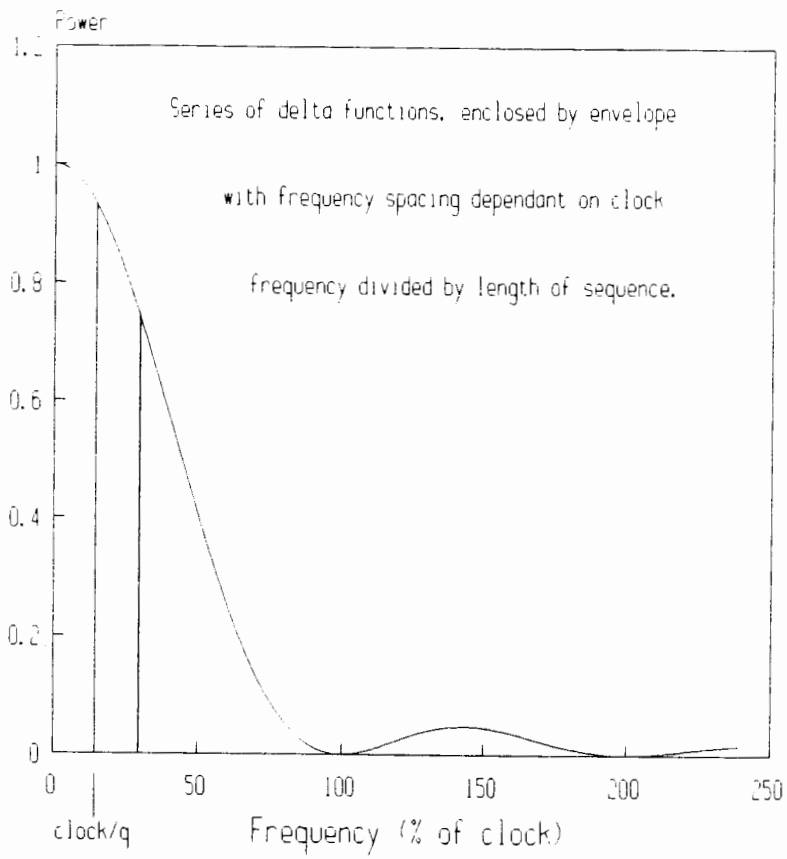


figure A.10.4. Power Spectrum of PRBS

A.10.5. COMPARISON TO TRUE RANDOM SEQUENCE

The first property of a pseudo random bit sequence (PRBS) that is completely foreign to truly random sequences is of course the cyclic one as can immediately be seen in the comparison of the discrete autocorrelation functions figures A.10.1 and A.10.2.

Secondly, Vincent [47] states that the probability that a truly random sequence would have the exact proportions of the runs of various lengths (as specified in the above properties of PRBS's) is very low, and these runs would therefore be highly uncharacteristic. Some runs would, in general, occur more often while others might not occur at all in a run of q bits.

Horowitz [48] uses an analogy to a "random walk" to compare PRBS's and truly random sequences. In moving one step forward for a 1 and one step backward for a 0 one would end up exactly one step away from where one started after a complete sequence of q bits, a result which is definitely not random.

The analogy is taken further to say that if one assumes the major nonrandom property of the PRBS is its exact equality of zeroes and ones, it can be shown that the random walk as described should reach an average distance from the starting point of

$$x = [\, r\,(q - r)\,(q - 1)\,]^{1/2}$$

after r draws from a population of p/2 ones and p/2 zeroes.

In a completely random walk x will equal the square root of r, thus the factor $[(q-1)(q-1)]^{1/2}$ expresses the effect of being limited to equal ones and zeroes generated in a finite sequence length of q.

Since Horowitz [49] in his discussion of PRBS's is primarily concerned with the generation of analog noise from the filtered digital output he advocates setting r very much less than q (where r is the number of samples used from a possible sequence length of q). The randomness of the walk is only slightly reduced from the completely random case (more and more so as q tends to infinity) and thus the section of the PRBS is indistinguishable from a true random sequence of this length.

Vincent [50] is primarily concerned with the generation of m-bit pseudo random numbers for use in Monte Carlo neutron physics calculations. He similarly advocates that in the generation of m-bit numbers by means of an n stage shift register, to simulate truly random numbers it is desirable for n to be sufficiently large for $2^n >> 2^m$.

In the above two recommendations the main drawback of PRBS's is their finite sequence length but this factor is useful in the practical measurement of bit error rates using these sequences. This is because the received sequence can be

synchronised against the standard sequence every period to ensure correct correlation (the correlation in this instance is looking for the peak at zero displacement of 100% correlation).

In practical use therefore, the length of the sequence (and hence its limit on randomness) must be balanced against the maximum allowable number of bits that can be processed before resynchronisation is deemed necessary for statistical purposes. This topic is covered further in the section on resynchronisation (chapter 6) and in the section on the design of the exact sequence used (chapter 4).

When looking at sequences in terms of spectral and statistical properties,and using more sophisticated tests of randomness such as measured by high order correlations, current research indicates that the use of many feedback taps (greater than or equal to half the number of stages) in the feedback network generates "better" sequences in this respect. There is more discussion on this matter in the section comparing CCITT recommended sequences to others, which is chapter 4.

A.11. PROOF OF MAXIMUM LENGTH OF SEQUENCES

In Marsh's tables [51] of irreducible polynomials over GF(2) the primitive polynomial with the least number of coefficients of the order 15 is listed in the octal format used in the tables as

1 0 0 0 0 3 F

which corresponds to a polynomial of the form

$$f(x) = x^{15} + x + 1$$

and according to its classification letter (F) the equation is primitive, the roots of the equation are linearly dependant while the roots for its reciprocal polynomial f*(x) are linearly independent.

It is known that the reciprocal polynomial of an irreducible polynomial is also irreducible and the reciprocal polynomial of a primitive polynomial is also primitive and thus Marsh only lists in his tables one of all reciprocal pairs.

One can therefore deduce that the polynomial recommended by the CCITT in figure A.10.1.,

$$g(x) = x^{15} + x^{14} + 1$$

which is the reciprocal of

$$f(x) = x^{15} + x + 1$$

is irreducible and primitive over the Galois field of GF(2) and since the degree of the polynomial is 15, it has linearly independent roots over the extension field of $GF(2^{15})$ or GF(32768) as can be seen from the field theory covered in chapter 2.

By the definition of a primitive polynomial one knows that the above polynomial must divide $(x^n - 1)$ for no n less than $(2^{15} - 1)$.

Thus by theorem 7 one finds that the solutions of the recurrence equation defined by the above polynomial are periodic of period $(2^{15} - 1)$ and since the polynomial is of degree 15 it can produce no more than 2^{15} different solutions (the zero vector being one of them), one finds that this is a maximum length sequence generator polynomial. Furthermore, from the properties of maximum length sequences

deduced in the preceding chapters, one finds the feedback shift register connected according to the above generator polynomial to be a pseudo random bit sequence generator.

In Marsh's tables [52] one also finds the octal formatted equation

1 0 5 5 5 5 E

which corresponds to the algebraic equation

$$f(x) = x^{15} + x^{11} + x^{9} + x^{8} + x^{6} + x^{5} + x^{3} + x^{2} + 1 \ .$$

This is the alternative configuration equation proposed above. The equation is, according to its classification letter (E), primitive and has linearly dependant roots.

One can thus deduce, as for the CCITT equation above, that this equation is also a generator polynomial for a pseudo random sequence.

APPENDIX B

SIMULATION PROGRAM AND OUTPUT

The simulation program included in this appendix is written in Turbo Pascal version 4.0 and the commented procedures are listed in alphabetical order.

The program will prompt the user for input of shift register length (between 2 and 23 stages), followed by bit by bit starting seed value, and finally bit by bit feedback connection value (enter a 1 if the particular stage is included int he feedback network or a 0 if not). Only an even number of feedback stages will be accepted as the program automatically assigns the linear feedback connections.

The program output, examples of which are included after the source code, is written to a file named SIM.OUT in the current directory and any files in the current directory of the same name will be overwritten.

```
{*********************************************************}
{ PSEUDO RANDOM SEQUENCE GENERATOR SIMULATOR PROGRAM      }
{*********************************************************}

PROGRAM SIMUL;

        var                                        { global var}
            filvar                  : TEXT;        {output file}
                                { arrays denoting binary value  }
                                { and feedback connection of SR }
            stage, flag             : ARRAY [0..22] OF BOOLEAN;
            finish1,finish2,
            sand,newstage           : BOOLEAN;     { flags }
            I,tempflag,stage_len    : INTEGER;     { counters }
            in_char                 : CHAR;        { replies  }
            count,countone,
            countzero,seq_len       : REAL;        { counters }

{*********************************************************}
{     PROCEDURE : Check                                   }
{     FUNCTION  : Checks for end of sequence of output.   }
{     PARAMETERS: none                                    }
{     CALLS     : none                                    }
{*********************************************************}

Procedure Check;
   begin
          I := 0;
          finish1 := true;   { necessary condition to end }
          repeat
                finish1 := finish1 AND stage [I];
                I := I + 1;
          until I >= stage_len + 1;     { AND every stage }
   end;

{*********************************************************}
{     PROCEDURE : Instage                                 }
{     FUNCTION  : To prompt the user for the number of    }
{                 stages in the shift register and the    }
{                 starting seed for every stage.          }
{     PARAMETERS: none.                                   }
{     CALLS     : readln,writeln.                         }
{*********************************************************}

Procedure Instage;

   begin
          repeat              { until no. of stages is 2-23}
            writeln;
            writeln ('ENTER NO. OF STAGES (2-16)');
            readln (stage_len);           { user response }
            stage_len := stage_len - 1;     { use 0 value }
          until (stage_len > 2) AND (stage_len < 22);
          seq_len := 1;              { find max len of seq }
```

```
          for I := 0 to stage_len do
              seq_len := 2 * seq_len;
          for I := 0 to stage_len do
          begin                                    { enter seed }
               writeln ('ENTER STAGE  ',I,' AS A 0 OR 1');
               readln  (tempflag);                { read integer }
               if tempflag = 0                     { convert to  }
                  then stage[I] := false;          { Boolean for }
               if tempflag = 1                     { stage array }
                  then stage[I] := true
          end
   end;

{*********************************************************}
{     PROCEDURE : Inflags                                 }
{     FUNCTION  : To prompt the user for feedback connec- }
{                 tivity of each stage and store in array.}
{     PARAMETERS: none                                    }
{     CALLS     : writeln,readln,odd.                     }
{*********************************************************}

Procedure Inflags;

   var
      count_flag : integer;          { no. of connections }

   begin
      repeat
          count_flag := 1;           { because we use the }
          writeln;                   { odd test function  }
          for I := 0 to stage_len do
          begin
               writeln ('ENTER FLAG FOR STAGE  ',
                         I,' AS A 0 OR 1');
               readln  (tempflag);          { read integer }
               if tempflag = 0              { set flags    }
                  then flag[I] := false;
               if tempflag = 1 then
               begin
                  flag[I] := true;            { count one  }
                  count_flag := count_flag + 1  { more     }
               end                            { connection }
          end;
      until odd (count_flag);          { even no. of taps }
   end;
```

```
{*********************************************************}
{     PROCEDURE : Logic                                   }
{     FUNCTION  : To implement the feedback network accor-}
{                 ding to the connection array and the    }
{                 current SR vector. The function uses an }
{                 array of booleans to calculate the next }
{                 bit value to be shifted into the reg.   }
{     PARAMETERS: none                                    }
{     CALLS     : none                                    }
{*********************************************************}

Procedure Logic;

   var
      sum : array [1..8] of boolean;   { feedback network }
      J   : integer;

   begin
        for I := 1 to 8 do             { max of 8 fback taps }
            sum[I] := false;           { initial values are 0}
        I := 0;
        J := 1;                        { counter initialise  }
        repeat
              if flag[I] then          { there is fback tap  }
              begin
                   sum[J] := stage[I];   { set first value}
                   repeat
                         I := I + 1
                   until flag[I];      { look for next tap }
                   sum[J] := sum[J] XOR stage[I]; { sum it}
                   J := J + 1          { point to next stage}
              end;
              I := I + 1;              { point to next stage}
        until I >= stage_len;          { until end of reg   }
        sum[1] := sum[1] XOR sum[2];
        sum[2] := sum[3] XOR sum[4];      { sum down array }
        sum[3] := sum[5] XOR sum[6];      { to half array  }
        sum[4] := sum[7] XOR sum[8];
        sum[1] := sum[1] XOR sum[2];      { sum down array }
        sum[2] := sum[3] XOR sum[4];      { to half array  }
        newstage := sum[1] XOR sum[2];    { final sum      }
   end;
```

```
{*********************************************************}
{     PROCEDURE : Output                                  }
{     FUNCTION  : To write to the output file the first 20}
{                 vectors a blank line and then the last  }
{                 twenty vectors.                         }
{     PARAMETERS: none                                    }
{     CALLS     : write,writeln,readln.                   }
{*********************************************************}

procedure output;

    var
        I,J:integer;

    begin
        if count = 22 then
           writeln (filvar);    { output blank line on 22 }
                                { if within the first }
                                { 20 or the last 20   }
        if (count < 21) OR (count > seq_len - 22) then
        begin
            for I:=0 to stage_len do  { output each stage }
            begin                     { for this vector   }
                if stage[I] then write (filvar,'1');
                if stage[I] = false then write (filvar,'0');
            end;
            writeln (filvar);         { output blank line }
        end;
    end;

{*********************************************************}
{     PROCEDURE : Printcon                                }
{     FUNCTION  : This procedure prints the configuration }
{                 word of the shift register to the output}
{                 file to identify the output.            }
{     PARAMETERS: none.                                   }
{     CALLS     : assign,rewrite,write,writeln.           }
{*********************************************************}

Procedure printcon;

   begin
        assign (filvar,'sim.out');        { set file name }
        rewrite (filvar);                 { open it for wr}
        writeln (filvar);               { output blank line}
        write (filvar,'THE SHIFT REG CONFIGURATION IS: ');
        for I := 0 to stage_len do
        begin
            if flag[I]                { test flag of array}
               then write (filvar,'1')      { write approp}
               else write (filvar,'0')      {-riate value }
        end;
        writeln (filvar);
        writeln (filvar);                    { blank lines }
```

```
        for I := 0 to stage_len do      { for every stage }
            write (filvar,'1');         { write seed value}
        writeln (filvar)
    end;


{*********************************************************}
{     PROCEDURE : Sequend                                 }
{     FUNCTION  : To output the number of steps in the    }
{                 sequence, the no. of ones and the no. of}
{                 zeroes to the output file and to prompt }
{                 on the terminal for program continuation}
{                 or termination.                         }
{     PARAMETERS: none                                    }
{     CALLS     : writeln,readln, close.                  }
{*********************************************************}

Procedure Sequend;

   begin
          writeln (filvar);
          writeln (filvar,
                   'THE NUMBER OF STEPS IN THE SEQUENCE WAS '
                         ,count);
          writeln (filvar,
                   'THE NUMBER OF ONES  IN THE SEQUENCE WAS '
                         ,countone);
          writeln (filvar,
                   'THE NUMBER OF ZEROS IN THE SEQUENCE WAS '
                         ,countzero);
          close (filvar);              { close output file }
          writeln ('ENTER Y/N TO CONTINUE');
          readln  (in_char);           { fetch user reply  }
          if in_char = 'y'
             then finish2 := false { reset flag for ending}
             else finish2 := true  {   set flag for ending}
   end;
{*********************************************************}
{     PROCEDURE : Shift                                   }
{     FUNCTION  : To simulate the practical clocking of   }
{                 the shift register model by re-arrange- }
{                 ment of the array.                      }
{     PARAMETERS: none.                                   }
{     CALLS     : none.                                   }
{*********************************************************}

Procedure Shift;

   begin
          I := stage_len;                 { no. of stages }
          if stage[stage_len] = true            { count the}
             then countone := countone + 1      { ones and }
             else countzero := countzero + 1;   { zeroes   }
          repeat
```

```
            stage[I] := stage[I-1];     { shift stages to}
            I := I - 1                  { become previous}
         until I = 0;                   { stage value.   }
         stage[0] := newstage       { shift in new stage }
   end;

{*********************************************************}
{     PROCEDURE : Main                                    }
{     FUNCTION  : To call all the relevant procedures in  }
{                 the correct order and to control the    }
{                 looping of the shift register model and }
{                 continuation of the program.            }
{     PARAMETERS: none.                                   }
{     CALLS     : check,inflags,instage,logic,output,     }
{                 printcon,sequend,shift.                 }
{*********************************************************}

   Begin
         repeat
               count := 0;                { initialise }
               countone := 0;
               countzero := 0;
               Instage;                   { get seed   }
               Inflags;                   { get fbk net}
               printcon;                  { o/p config }
               repeat
                     count := count + 1; { no. in seq }
                     Logic;              { calc newstg}
                     Shift;              { shift array}
                     output;             { o/p vector }
                     Check;              { tst for end}
               until finish1;            { flg for end}
               Sequend;                  { o/p results}
         until finish2                   { flag to fin}
    end.
```

THE SIMULATION PROGRAM OUTPUT FOR THE FEEDBACK SHIFT REGISTER CONFIGURED ACCORDING TO THE EQUATION

$F(X) = X^{15} + X^{14} + 1$

THE SHIFT REG CONFIGURATION IS: 000000000000011

```
111111111111111
011111111111111
001111111111111
000111111111111
000011111111111
000001111111111
000000111111111
000000011111111
000000001111111
000000000111111
000000000011111          FIRST TWENTY VECTORS
000000000001111
000000000000111
000000000000011
000000000000001
100000000000000
010000000000000
001000000000000
000100000000000
000010000000000
000001000000000

101010101001100
010101010100110
101010101010011
010101010101001
101010101010100
010101010101010
101010101010101
110101010101010
111010101010101
111101010101010
111110101010101          LAST TWENTY VECTORS
111111010101010
111111101010101
111111110101010
111111111010101
111111111101010
111111111110101
111111111111010
111111111111101
111111111111110
111111111111111
```

THE NUMBER OF STEPS IN THE SEQUENCE WAS 32767
THE NUMBER OF ONES IN THE SEQUENCE WAS 16384
THE NUMBER OF ZEROS IN THE SEQUENCE WAS 16383

THE SIMULATION PROGRAM OUTPUT FOR THE FEEDBACK SHIFT REGISTER CONFIGURED ACCORDING TO THE EQUATION

$F(X) = X^{15} + X^{1} + 1$

THE SHIFT REG CONFIGURATION IS: 100000000000001

```
111111111111111
011111111111111
101111111111111
010111111111111
101011111111111
010101111111111
101010111111111
010101011111111
101010101111111
010101010111111
101010101011111          FIRST TWENTY VECTORS
010101010101111
101010101010111
010101010101011
101010101010101
010101010101010
001010101010101
100101010101010
110010101010101
011001010101010
001100101010101

000000000100000
000000000010000
000000000001000
000000000000100
000000000000010
000000000000001
100000000000000
110000000000000
111000000000000
111100000000000
111110000000000          LAST TWENTY VECTORS
111111000000000
111111100000000
111111110000000
111111111000000
111111111100000
111111111110000
111111111111000
111111111111100
111111111111110
111111111111111
```

THE NUMBER OF STEPS IN THE SEQUENCE WAS 32767
THE NUMBER OF ONES IN THE SEQUENCE WAS 16384
THE NUMBER OF ZEROS IN THE SEQUENCE WAS 16383

THE SIMULATION PROGRAM OUTPUT FOR THE FEEDBACK SHIFT REGISTER CONFIGURED ACCORDING TO THE EQUATION

$F(X) = X^{15} + X^{13} + X^{10} + X^{8} + 1$

THE SHIFT REG CONFIGURATION IS: 000000010100101

111111111111111
011111111111111
001111111111111
000111111111111
000011111111111
000001111111111
000000111111111
000000011111111
000000001111111
100000000111111
110000000011111 FIRST TWENTY VECTORS
011000000001111
001100000000111
000110000000011
100011000000001
110001100000000
011000110000000
101100011000000
110110001100000
111011000110000
111101100011000

101011001100001
010101100110000
101010110011000
110101011001100
011010101100110
001101010110011
100110101011001
110011010101100
111001101010110
111100110101011
111110011010101 LAST TWENTY VECTORS
111111001101010
111111100110101
111111110011010
111111111001101
111111111100110
111111111110011
111111111111001
111111111111100
111111111111110
111111111111111

THE NUMBER OF STEPS IN THE SEQUENCE WAS 32767
THE NUMBER OF ONES IN THE SEQUENCE WAS 16384
THE NUMBER OF ZEROS IN THE SEQUENCE WAS 16383

THE SIMULATION PROGRAM OUTPUT FOR THE FEEDBACK SHIFT REGISTER CONFIGURED ACCORDING TO THE EQUATION

$F(X) = X^{15} + X^{12} + X^{3} + X^{1} + 1$

THE SHIFT REG CONFIGURATION IS: 101000000001001

111111111111111
011111111111111
101111111111111
010111111111111
001011111111111
100101111111111
110010111111111
111001011111111
011100101111111
101110010111111
010111001011111 FIRST TWENTY VECTORS
001011100101111
100101110010111
010010111001011
001001011100101
000100101110010
000010010111001
000001001011100
100000100101110
010000010010111
101000001001011

100011111000000
110001111100000
111000111110000
011100011111000
001110001111100
000111000111110
100011100011111
110001110001111
111000111000111
111100011100011
111110001110001 LAST TWENTY VECTORS
111111000111000
111111100011100
111111110001110
111111111000111
111111111100011
111111111110001
111111111111000
111111111111100
111111111111110
111111111111111

THE NUMBER OF STEPS IN THE SEQUENCE WAS 32767
THE NUMBER OF ONES IN THE SEQUENCE WAS 16384
THE NUMBER OF ZEROS IN THE SEQUENCE WAS 16383

THE SIMULATION PROGRAM OUTPUT FOR THE FEEDBACK SHIFT REGISTER CONFIGURED ACCORDING TO THE EQUATION

$F(X) = X^{15} + X^{10} + X^{5} + X^{1} + 1$

THE SHIFT REG CONFIGURATION IS: 100010000100001

111111111111111
011111111111111
101111111111111
010111111111111
101011111111111
010101111111111
001010111111111
100101011111111
110010101111111
011001010111111
001100101011111 FIRST TWENTY VECTORS
100110010101111
010011001010111
001001100101011
000100110010101
100010011001010
010001001100101
001000100110010
100100010011001
010010001001100
101001000100110

111111111000000
011111111100000
001111111110000
000111111111000
000011111111100
000001111111110
100000111111111
110000011111111
111000001111111
111100000111111
111110000011111 LAST TWENTY VECTORS
111111000001111
111111100000111
111111110000011
111111111000001
111111111100000
111111111110000
111111111111000
111111111111100
111111111111110
111111111111111

THE NUMBER OF STEPS IN THE SEQUENCE WAS 32767
THE NUMBER OF ONES IN THE SEQUENCE WAS 16384
THE NUMBER OF ZEROS IN THE SEQUENCE WAS 16383

THE SIMULATION PROGRAM OUTPUT FOR THE FEEDBACK SHIFT REGISTER CONFIGURED ACCORDING TO THE EQUATION

$$F(X) = X^{15} + X^{11} + X^{9} + X^{8} + X^{6} + X^{5} + X^{3} + X^{2} + 1$$

THE SHIFT REG CONFIGURATION IS: 011011011010001

111111111111111
011111111111111
001111111111111
100111111111111
010011111111111
101001111111111
010100111111111
101010011111111
010101001111111
101010100111111
010101010011111 FIRST TWENTY VECTORS
101010101001111
010101010100111
001010101010011
100101010101001
110010101010100
011001010101010
001100101010101
000110010101010
000011001010101
100001100101010

101010010000110
110101001000011
011010100100001
001101010010000
000110101001000
000011010100100
100001101010010
110000110101001
111000011010100
111100001101010
111110000110101 LAST TWENTY VECTORS
111111000011010
111111100001101
111111110000110
111111111000011
111111111100001
111111111110000
111111111111000
111111111111100
111111111111110
111111111111111

THE NUMBER OF STEPS IN THE SEQUENCE WAS 32767
THE NUMBER OF ONES IN THE SEQUENCE WAS 16384
THE NUMBER OF ZEROS IN THE SEQUENCE WAS 16383

THE SIMULATION PROGRAM OUTPUT FOR THE FEEDBACK SHIFT REGISTER CONFIGURED ACCORDING TO THE EQUATION

$$F(X) = X^{15} + X^{13} + X^{12} + X^{10} + X^{9} + X^{6} + X^{3} + X^{1} + 1$$

THE SHIFT REG CONFIGURATION IS: 101001001101101

111111111111111
011111111111111
101111111111111
010111111111111
001011111111111
100101111111111
110010111111111
011001011111111
101100101111111
110110010111111
111011001011111
111101100101111
111110110010111
011111011001011
101111101100101
110111110110010
111011111011001
011101111101100
001110111110110
000111011111011
100011101111101

010010001110110
101001000111011
010100100011101
101010010001110
010101001000111
001010100100011
100101010010001
110010101001000
111001010100100
111100101010010
111110010101001
111111001010100
111111100101010
111111110010101
111111111001010
111111111100101
111111111110010
111111111111001
111111111111100
111111111111110
111111111111111

THE NUMBER OF STEPS IN THE SEQUENCE WAS 32767
THE NUMBER OF ONES IN THE SEQUENCE WAS 16384
THE NUMBER OF ZEROS IN THE SEQUENCE WAS 16383

THE SIMULATION PROGRAM OUTPUT FOR THE FEEDBACK SHIFT REGISTER CONFIGURED ACCORDING TO THE EQUATION

$$F(X) = X^{15} + X^{12} + X^{11} + X^{8} + X^{7} + X^{6} + X^{4} + X^{2} + 1$$

THE SHIFT REG CONFIGURATION IS: 010101110011001

111111111111111
011111111111111
001111111111111
100111111111111
110011111111111
111001111111111
111100111111111
111110011111111
011111001111111
001111100111111
000111110011111 FIRST TWENTY VECTORS
100011111001111
110001111100111
111000111110011
111100011111001
011110001111100
001111000111110
000111100011111
000011110001111
100001111000111
010000111100011

010001111111010
001000111111101
100100011111110
010010001111111
001001000111111
000100100011111
100010010001111
110001001000111
111000100100011
111100010010001
111110001001000 LAST TWENTY VECTORS
111111000100100
111111100010010
111111110001001
111111111000100
111111111100010
111111111110001
111111111111000
111111111111100
111111111111110
111111111111111

THE NUMBER OF STEPS IN THE SEQUENCE WAS 32767
THE NUMBER OF ONES IN THE SEQUENCE WAS 16384
THE NUMBER OF ZEROS IN THE SEQUENCE WAS 16383

THE SIMULATION PROGRAM OUTPUT FOR THE FEEDBACK SHIFT REGISTER CONFIGURED ACCORDING TO THE EQUATION

$$F(X) = X^{15} + X^{12} + X^{11} + X^{8} + X^{5} + X^{4} + X^{2} + X^{1} + 1$$

THE SHIFT REG CONFIGURATION IS: 110110010011001

```
111111111111111
011111111111111
101111111111111
110111111111111
011011111111111
001101111111111
100110111111111
110011011111111
111001101111111
111100110111111
111110011011111          FIRST TWENTY VECTORS
011111001101111
101111100110111
110111110011011
011011111001101
101101111100110
110110111110011
111011011111001
111101101111100
111110110111110
111111011011111

010100000010011
001010000001001
100101000000100
010010100000010
001001010000001
000100101000000
100010010100000
110001001010000
111000100101000
111100010010100
111110001001010          LAST TWENTY VECTORS
111111000100101
111111100010010
111111110001001
111111111000100
111111111100010
111111111110001
111111111111000
111111111111100
111111111111110
111111111111111
```

THE NUMBER OF STEPS IN THE SEQUENCE WAS 32767
THE NUMBER OF ONES IN THE SEQUENCE WAS 16384
THE NUMBER OF ZEROS IN THE SEQUENCE WAS 16383

APPENDIX C

THE USER INTERFACE SOURCE CODE

C.1. INTRODUCTION TO ANALYSER PROGRAM

Appendices C to G contain the source code modules of the software developed for this project in the following order:

C) The map file generated by the compilation and linking of the five source modules and the C source code for the user interfacing routines and initialising routines. This code contains the linked program entry point.

D) The C source code for the screen generation and pop down window menus.

E) The C source code for the creation of a logging file and the pop down window menu for the naming of this file.

F) The Assembler source code for the IRQ3 interrupt service routine (which reads data from the analyser card) as well as the routines for installing and un-installing the service routine.

G) The Assembler source code for the IRQ4 interrupt service routine (which reads received data from the RS232 communications card) as well as the routines for installing and un-installing the service routine.

C.2. MAP LISTING OF PUBLIC SYMBOLS

Stack Allocation = 4000 bytes

Start	Stop	Length	Name	Class
00000H	090ADH	090AEH	_TEXT	CODE
090B0H	0B3C1H	02312H	EMULATOR_TEXT	CODE
0B3C2H	0B3C2H	00000H	C_ETEXT	ENDCODE
0B3D0H	0B6F1H	00322H	_STACK	STACK
0B700H	0B82FH	00130H	EMULATOR_DATA	FAR_DATA
0B830H	0B865H	00036H	NULL	BEGDATA
0B866H	0CB13H	012AEH	_DATA	DATA
0CB14H	0CB21H	0000EH	CDATA	DATA
0CB22H	0CB22H	00000H	XIB	DATA
0CB22H	0CB22H	00000H	XI	DATA
0CB22H	0CB22H	00000H	XIE	DATA
0CB22H	0CB22H	00000H	XPB	DATA
0CB22H	0CB23H	00002H	XP	DATA
0CB24H	0CB24H	00000H	XPE	DATA
0CB24H	0CB24H	00000H	XCB	DATA
0CB24H	0CB24H	00000H	XC	DATA
0CB24H	0CB24H	00000H	XCE	DATA
0CB24H	0CB43H	00020H	CONST	CONST
0CB44H	0CB4BH	00008H	HDR	MSG
0CB4CH	0CC1CH	000D1H	MSG	MSG
0CC1DH	0CC1EH	00002H	PAD	MSG
0CC1FH	0CC1FH	00001H	EPAD	MSG
0CC20H	0CC4DH	0002EH	_BSS	BSS
0CC4EH	0CC4EH	00000H	XOB	BSS
0CC4EH	0CC4EH	00000H	XO	BSS
0CC4EH	0CC4EH	00000H	XOE	BSS
0CC50H	0D21BH	005CCH	c_common	BSS
0D220H	0E1BFH	00FA0H	STACK	STACK

Origin	Group
0B83:0	DGROUP

Address		Publics by Name
0B83:12B1		$i8_implicit_exp
0000:8D0E		$i8_input
0B83:12B0		$i8_input_ws
0000:6A1B		$i8_output
0000:6CDC		$i8_tpwr10
0000:FE32	Abs	FIARQQ
0000:0E32	Abs	FICRQQ
0000:5C32	Abs	FIDRQQ
0000:1632	Abs	FIERQQ
0000:0632	Abs	FISRQQ
0000:A23D	Abs	FIWRQQ
0000:4000	Abs	FJARQQ
0000:C000	Abs	FJCRQQ
0000:8000	Abs	FJSRQQ
0B83:0EDE		STKHQQ

```
0B83:003E       _above_thresh
0000:6D87       _atof
0000:8A64       _atol
0B83:19E2       _average
0000:2F43       _bios
0B83:006C       _breakstat
0000:8518       _brkctl
0B83:0D58       _b_adap_state
0B83:0D36       _b_bp
0B83:0D3C       _b_cga
0B83:0D0E       _b_curknown
0B83:0CEE       _b_curoff
0B83:0CAC       _b_curpage
0B83:0CAE       _b_curtype
0B83:0D4E       _b_device
0B83:0D3E       _b_ega
0B83:0D34       _b_es
0B83:0D38       _b_know_hw
0B83:0D3A       _b_mdpa
0B83:0D42       _b_mem_ega
0B83:0DEA       _b_pactnode
0B83:0E54       _b_pcmodel
0B83:0E42       _b_pcurwin
0B83:0D40       _b_pgc
0B83:0D44       _b_sw_ega
0000:3821       _b_vidcpy
0B83:0D26       _b_wnerr
0B83:0E1A       _b_wnlist
0B83:142A       _c1
0B83:142B       _c2
0B83:142C       _c3
0000:6DC8       _calloc
0B83:006E       _channel_flag
0000:004C       _check
0000:77B4       _close
0000:8A67       _creat
0000:1B23       _cr_file
0000:00B0       _curs_off
0000:00DF       _curs_on
0B83:1256       _daylight
0000:010E       _dev_test
0000:3010       _dos
0000:0192       _dta_reset
0000:01A8       _dta_set
0B83:006D       _dump_flag
0B83:13F0       _edata
0B83:19F0       _end
0000:01E7       _end_rec
0000:021A       _end_trx
0B83:0F72       _environ
0000:023F       _erase_line
0B83:1638       _erray
0B83:0F4E       _errno
0B83:0042       _err_sec
```

```
0000:6F81       _exit
0000:6FEC       _fclose
0000:7810       _fflush
0B83:1708       _file_name
0000:1F69       _flretdta
0000:1F9C       _flsetdta
0000:8AED       _flushall
0000:708F       _fopen
0B83:19E8       _fpoint
0000:70B5       _fprintf
0000:7226       _free
0B83:0036       _frequency
0000:70EC       _getch
0000:8B1E       _getenv
0B83:19DE       _grand_total
0000:558A       _gvrdrect
0000:4C08       _gvwrrect
0000:1CB6       _init
0000:1D5D       _init2
0000:027D       _inj_err
0000:038F       _inj_err_rx
0000:0298       _inj_err_tx
0B83:19E6       _inj_flag
0B83:19EA       _inj_flag_tx
0000:1688       _inj_wind
0000:7108       _inp
0000:7115       _int86
0000:85DC       _isatty
0000:7878       _itoa
0000:7196       _kbhit
0000:30BB       _kbin
0B83:003A       _last_rd
0000:1B7C       _log_wind
0B83:1702       _loop_count
0B83:162E       _ltime
0000:0010       _main
0000:7234       _malloc
0B83:0070       _mask
0000:71A3       _memcpy
0000:71F5       _memset
0000:0445       _menu1
0000:0531       _menu1_rx
0000:04AE       _menu1_tx
0000:1475       _menu1_wind
0000:05B4       _menu2
0000:06C6       _menu2_rx
0000:05D6       _menu2_tx
0000:126A       _menu2_wind
0B83:176C       _message
0B83:0064       _mess_buf
0000:1233       _mess_table
0B83:1422       _new_dta_ads
0B83:1632       _old_dta_ads
0B83:1636       _old_off2
```

```
0B83:1700       _old_seg2
0000:85FE       _open
0000:727A       _outp
0B83:1706       _pbuffer
0B83:1428       _pointmessage
0000:7288       _printf
0000:4C2F       _qymodel
0000:72CF       _rand
0000:83E0       _remove
0B83:1976       _response1
0000:07A1       _restrt
0000:07DC       _running
0000:09CC       _run_print
0B83:19EB       _rx_flag
0B83:19DB       _rx_flag_tx
0000:1120       _rx_table
0B83:0066       _sample_time
0000:1811       _sample_wind
0000:310B       _scattrib
0000:1FC3       _scbox
0000:4C60       _scchgdev
0000:325E       _sccurpos
0000:2256       _sccurset
0000:22EF       _sccurst
0000:32BE       _scequip
0000:23EE       _scmode
0000:3573       _scnewdev
0000:247F       _scpage
0000:38FA       _scpages
0000:24AE       _scpgcur
0000:55AB       _scpscrol
0000:5ACD       _scrdbuf
0000:57FB       _scread
0000:39F8       _scrows
0000:5C98       _scscroll
0000:4D94       _scttywin
0000:509D       _scttywrt
0000:5D9A       _scwrbuf
0000:585B       _scwrite
0000:0D9B       _set_sample
0000:0E28       _set_threshold
0000:0E98       _set_tx_threshold
0000:0910       _single
0000:72BE       _srand
0B83:19DA       _start
0000:0F08       _start_err_count
0000:0F4C       _start_rec
0000:0FC7       _start_trx
0000:835C       _strcat
0000:72FB       _strcmp
0000:7330       _strcpy
0000:7355       _strlen
0000:8BEB       _strncmp
0000:8B6E       _strncpy
```

```
0B83:0068       _threshold
0000:199A       _thresh_wind
0000:7370       _time
0B83:1252       _timezone
0B83:1426       _tm_now
0000:73C7       _tolower
0B83:004E       _total2
0000:1DDC       _trx_buf
0B83:0046       _tx_above_thresh
0B83:005A       _tx_average
0B83:004A       _tx_err_sec
0B83:1420       _tx_flag
0B83:19DC       _tx_flag_tx
0B83:0056       _tx_grand_total
0B83:005E       _tx_ppm
0000:1000       _tx_table
0B83:006A       _tx_threshold
0B83:0062       _tx_total
0B83:0052       _tx_total2
0B83:1258       _tzname
0000:8791       _tzset
0000:8923       _ultoa
0000:83C4       _ungetch
0000:83E0       _unlink
0000:26D0       _utabsptr
0000:5200       _utintoff
0000:5211       _utinton
0000:5222       _utmove
0000:3A49       _utslmove
0000:3A6B       _utsreg
0000:6044       _viads
0000:63AA       _vidirect
0000:640E       _virdrect
0000:59A0       _viwrrect
0000:5247       _wncover
0000:2713       _wncreate
0000:3A87       _wncurmov
0000:3B14       _wncurpos
0000:3B55       _wncursor
0000:27FD       _wndsplay
0000:2A6B       _wndstroy
0000:3C86       _wnforget
0000:5344       _wngetimg
0000:3D38       _wnhide
0000:41C0       _wnmkimg
0000:53CB       _wnovrlap
0000:4214       _wnpgadd
0000:42D1       _wnpgrem
0000:5423       _wnputbor
0000:54D5       _wnputimg
0000:2AFD       _wnquery
0000:2CAC       _wnremove
0000:43C5       _wnseldev
0000:4456       _wnselect
```

```
0000:44B3       _wnunhide
0000:463F       _wnupdate
0000:5560       _wnvalnod
0000:472D       _wnvalwin
0000:2D28       _wnwrap
0000:4757       _wnwrbuf
0000:4965       _wnwrtty
0000:892D       _write
0000:1CE2       _xit
0000:1DAE       _xit2
0B83:0F34       __abrkp
0B83:0EE4       __abrktb
0B83:0F34       __abrktbe
0000:9876  Abs  __acrtmsg
0000:9876  Abs  __acrtused
0B83:0F42       __aintdiv
0000:763C       __amalloc
0000:7779       __amallocbrk
0B83:11B6       __amblksiz
0000:771D       __amexpand
0000:7757       __amlink
0000:6EC7       __amsg_exit
0000:8C25       __aNlmul
0000:8C51       __aNuldiv
0000:8C25       __aNulmul
0B83:11B0       __aseg1
0B83:1088       __asegds
0B83:11B2       __asegn
0B83:11B4       __asegr
0B83:0EE0       __asizds
0000:6E28       __astart
0B83:0EE2       __atopsp
0B83:1776       __bufin
0B83:142E       __bufout
0B83:11E4       __byte
0B83:1247       __cappend
0000:8B96       __catox
0000:6924       __cfltcvt
0B83:11E6       __cflush
0000:670E       __cftoe
0000:67FC       __cftof
0000:68C4       __cftog
0B83:0F74       __child
0000:6E12       __chkstk
0000:6ED2       __cinit
0000:6EC4       __cintDIV
0000:8A07       __cltoasub
0000:8603       __copensub
0000:6659       __cropzeros
0000:8A83       __csetmode
0B83:0F76       __csigtab
0000:6FC4       __ctermsub
0B83:0F7E       __ctype
0B83:0F7E       __ctype_
```

```
0000:8ADA        __cXENIXtoDOSmode
0000:8A13        __cxtoa
0B83:1202        __days
0B83:0F58        __doserrno
0000:77C8        __dosret0
0000:77D0        __dosretax
0B83:0F56        __dosvermajor
0B83:0F57        __dosverminor
0000:81E0        __dtoxtime
0000:6F98        __exit
0B83:0F46        __fac
0000:66CB        __fassign
0000:6516        __fcmp
0000:83ED        __flsbuf
0000:8CC9        __fltin
0000:8CB6        __fltinf
0000:6993        __fltout
0000:9876  Abs   __fltused
0B83:129E        __fmode
0000:6620        __forcdecpt
090B:0723        __fpemulator
090B:0200        __FPEXCEPTION87
0B83:12E6        __fpinit
090B:2203        __FPINSTALL87
090B:0024        __fpmath
0000:65E3        __fpsignal
0B70:012E        __fptaskdata
090B:227F        __FPTERMINATE87
0000:6D19        __fptostr
0000:73E2        __freebuf
0000:75B3        __ftbuf
0000:838D        __getstream
0B83:1096        __iob
0B83:1136        __iob2
0B83:129E        __iomode
0000:8847        __isindst
0B83:11AE        __lastiob
0B83:11E8        __lpdays
0000:77DC        __maperror
0000:874F        __myalloc
0000:7226        __nfree
0000:7234        __nmalloc
0000:7893        __NMSG_TEXT
0000:78C3        __NMSG_WRITE
0000:7798        __nullcheck
0000:7411        __openfile
0B83:0F58        __oserr
0B83:0F5A        __osfile
0B83:0F56        __osmajor
0B83:0F57        __osminor
0000:78EC        __output
0000:66AD        __positive
0B83:0F54        __psp
0B83:0F52        __pspadr
```

```
 0000:80FB       __setargv
 0000:82F6       __setenvp
 0000:7514       __stbuf
 0B83:0F50       __umaskval
 0B83:0F6E       ___argc
 0B83:0F70       ___argv
 0B83:125C       ___dnames
 0B83:1272       ___mnames

Program entry point at 0000:6E28
```

C.3. USER INTERFACING C ROUTINES

```
/*****************************************************/
/*    COMLINK:CONTROLLING PROGRAM FOR ANALINK        */
/*****************************************************/

/************************************/
/*         INCLUDE FILES            */
/************************************/

#include   <stdio.h>                  /* file for fprint    */
#include   <conio.h>                  /* file for outport   */
#include   <stdlib.h>                 /* file for tolower   */
#include   <time.h>                   /* file for time      */
#include   <dos.h>                    /* file for c-int86   */
#include   <process.h>                /* file for exit etc  */
#include   <malloc.h>                 /* file for malloc    */
#include   <bfile.h>                  /* file for fl functns*/
#include   <butility.h>               /* file for general   */

/************************************/
/*         CONSTANT DEFINITIONS     */
/************************************/

#define    FREQUENCY2  2048000    /* 2MHz bits / sec       */
#define    FREQUENCY8  8000000    /* 8MHz bit count        */
#define    INT_CNT_LO  0x00       /* count for  2MHz       */
#define    INT_CNT_HI  0x28       /*   10 mSec  interrupt  */
#define    INT_CNT_LO8 0xFF       /* count for 8 MHz       */
#define    INT_CNT_HI8 0xFF       /*    10 mSec interrupt  */
#define    ACNT_LO     0xFE       /* count for 1 injected  */
#define    ACNT_HI     0xFF       /*   error per 64kbits   */
#define    BCNT_LO     0xFF       /* count for 2 injected  */
#define    BCNT_HI     0x7F       /*   errors per 64kbits  */
#define    CCNT_LO     0x33       /* count for 5 injected  */
#define    CCNT_HI     0x33       /*   errors per 64kbits  */
#define    DCNT_LO     0x9A       /* count for 10 injected */
#define    DCNT_HI     0x19       /*   errors per 64kbits  */
#define    ECNT_LO     0x8F       /* count for 100injected */
#define    ECNT_HI     0x02       /*   errors per 64kbits  */

#define    CNT0_MODE   0x34       /* no.0 m&lsb mode2,bin */
#define    CNT1_MODE   0x78       /* no.1 m&lsb mode4,bin */
#define    CNT2_MODE   0xB4       /* no.2 m&lsb mode2,bin */
#define    TEST_MODE   0x18       /* no.0 lsb   mode4,bin */

#define    RESET       0xFF       /* reset byte format  */
#define    REC_BITS    0xFC       /* receiver reset mask  */
#define    TRX_BIT     0xFB       /* transmitter reset mask*/
#define    INJ_BIT     0xEF       /* inject err reset mask*/
#define    STRT_CLK    0xF7       /* clock reset bit       */
```

```
#define     RST_LCH      0x31B      /* reset latch address */
#define     CNTER0       0x31C      /* counter0 latch address*/
#define     CNTER1       0x31D      /* counter1 latch address*/
#define     CNTER2       0x31E      /* counter2 latch address*/
#define     CNTRL1       0x31F      /* control latch address */
#define     MAX_SAMPLE   100        /* no. of samples per sec*/
#define     FULL         0xFF       /* fullcount bit pattern */
#define     VIDEO        0x10       /* int no. for bios video*/
#define     ESC          0x1B       /* char for communication*/
#define     ESC_COM      0xF01B     /* char for communication*/
#define     ESC_DATA     0x0F1B     /* char for communication*/

/************************************/
/*         GLOBAL DATA AREA         */
/************************************/

/*************************************************************/
/*   This section includes all data being accessed by        */
/*   more than one function as well as public data to        */
/*   be accessed by assembler routines as well as the        */
/*   declarations of the assembler routines called           */
/*   and the 'c' routines called externally.                 */
/*************************************************************/

struct tm  *tm_now;                 /* needed for time calls */

FILE *fpoint,*fopen();              /* needed for file access*/

ADS  old_dta_ads,new_dta_ads;       /* needed for DTA setup*/

unsigned long   frequency = FREQUENCY2,
                last_rd = 0xFFFF,loop_count,
                above_thresh = 0,err_sec = 0,
                tx_above_thresh = 0,tx_err_sec = 0,
                total2 = 0,tx_total2 = 0;

long        ltime;                  /* needed for time call */

float       grand_total,tx_grand_total = 0,        /* stat */
            average,tx_average = 0,tx_ppm = 0; /* variables*/

unsigned    pointmessage,old_off2,old_seg2,
            erray[MAX_SAMPLE],
            tx_total = 0,mess_buf = 0,
            sample_time = 1,threshold = 0,
            tx_threshold = 0;

char     c1,c2,c3,breakstat = 0,message[10],
         *file_name[50],*response1[50],*pbuffer,    /* flag */
         start,tx_flag,rx_flag,inj_flag,            /* and  */
         tx_flag_tx,rx_flag_tx,inj_flag_tx,         /* char */
         dump_flag = 0;channel_flag = 0,            /* bytes*/
         mask = RESET;
```

```
extern  trx_buf(),init(),xit(),init2(),
        xit2();                              /* ASM8088 procs */

extern  tx_table(),rx_table(),
        menu1_wind(),menu2_wind(),                /* c procs */
        inj_wind(),sample_wind(),
        thresh_wind(),cr_file();


/************************************/
/*         MAIN FUNCTION            */
/************************************/

/*************************************************************/
/*   PROCEDURE:main()                                        */
/*   FUNCTION :This procedure calls the initialising         */
/*           routines, calls the main interactive            */
/*           menu and then resets the enviroment             */
/*           with control always returning to it.            */
/*   GLOBAL VARS:none.                                       */
/*   CALLS:check(),curs_off(),curs_on(),dev_test(),          */
/*        dta_set(),dta_reset(),init(),init2(),              */
/*        menu1(),restrt(),xit(),xit2().                     */
/*         start_err_count.                                  */
/*   PARAMETERS:none.                                        */
/*   RETURNS:none.                                           */
/*************************************************************/

main()                                  /*  'c' prog entry point*/
{
   restrt();                            /*  reset device if not */
   dev_test();                          /* test if dev in system*/
   init();                              /* interrupt initialise */
   init2();                             /* initialise comms intr*/
   dta_set();                           /* setup disk trans buf */
   start_err_count();                   /* start intrpt counter */
   curs_off();                          /* cursor off for menu  */
   menu1();                             /* goto menu-driven part*/
   curs_on();                           /* turn cursor back on  */
   check();                             /* total reset request? */
   dta_reset();                         /* restore default DTA  */
   xit2();                              /* disable comms intrpt */
   xit();                               /* disable interrupts   */
   exit(0);                             /* exit to DOS return 0 */
}

/************************************/
/*          OTHER FUNCTIONS         */
/************************************/
```

```
/*************************************************************/
/*   PROCEDURE:check()                                       */
/*   FUNCTION:this function asks the user if a total devic*/
/*        reset is required on exit from the program or   */
/*        only a partial one. in the case of a partial    */
/*        reset the device will continue transmitting as  */
/*        long as the board is powered up or until the dev*/
/*        device is reset by re-entering the program.     */
/*        the control latch mask is tested and the user is*/
/*        informed if the device is transmitting on exit  */
/*   GLOBAL VARS:c3,mask.                                    */
/*   CALLS:end_rec(),restrt().                               */
/*   PARAMETERS:none.                                        */
/*   RETURNS:none.                                           */
/*************************************************************/

check()
{
   char reply;

   printf
   ("\x1b[23,10f                                          ");
   printf("\x1b[2J");              /* clear screen escape */
   printf
   ("\x1b[19;27fDO YOU WISH TO RESET CARD ON EXIT ? (Y/N)");
   reply = tolower(reply = getch());     /* get response  */
   if (reply == 'y')
      restrt();                  /* total reset of device */
   else                          /*       else            */
      end_rec();                 /* continue transmitting */
   if (mask  != RESET)
      printf
      ("\x1b[23;27fTHE DEVICE IS STILL TRANSMITTING !\n");
   printf("\x1b[23;1f");           /* position exit cursor */
}

/*************************************************************/
/*   PROCEDURE:curs_off()                                    */
/*   FUNCTION:This function makes a DOS system call throug*/
/*         the int86 library function and sets the cursor */
/*         top line to be below the cursor bottom line to */
/*         turn it off for the duration of menu displays  */
/*   GLOBAL VARS:none.                                       */
/*   CALLS:none                                              */
/*   PARAMETERS:none                                         */
/*   RETURNS:none                                            */
/*************************************************************/
```

```
curs_off()
{
   union REGS regs;
                                        /* set the register */
   regs.h.ah = 01;                      /* values through   */
   regs.h.al = 00;                      /* the union regs'  */
   regs.h.ch = 0x10;                    /* fields           */
   regs.h.cl = 00;
   int86(VIDEO,&regs,&regs);
}

/*************************************************************/
/*   PROCEDURE:curs_on()                                     */
/*   FUNCTION:this function makes a dos system call throug*/
/*         the int86 library function to set the cursor      */
/*         back to the DOS standard.                         */
/*   GLOBAL VARS:none.                                       */
/*   CALLS:none                                              */
/*   PARAMETERS:none                                         */
/*   RETURNS:none                                            */
/*************************************************************/

curs_on()
{
   union REGS regs;

   regs.h.ah = 01;                  /* set the register      */
   regs.h.al = 00;                  /* parameters passed in  */
   regs.h.ch = 06;                  /* the union regs'fields */
   regs.h.cl = 07;
   int86(VIDEO,&regs,&regs);
}

/*************************************************************/
/*   PROCEDURE:dev_test()                                    */
/*   FUNCTION:This function tests,at program start,whether*/
/*         a device is connected in the system or not and */
/*         informing the user in the case of one not found*/
/*         the test is done by enabling the clock and the */
/*         interrupt counter (in test mode) and seeing     */
/*         whether the counter is present by reading the  */
/*         count and looking for a change from FF which is*/
/*         the open circuit bus count ( no I/O device )    */
/*   GLOBAL VARS:c2,mask,loop_count,breakstat.               */
/*   CALLS:none                                              */
/*   PARAMETERS:none                                         */
/*   RETURNS:none                                            */
/*************************************************************/
```

```
dev_test()
{
   outp(RST_LCH,mask &= STRT_CLK);/*start the ck on board */
   outp(CNTRL1,TEST_MODE);        /* set count0 to testmode*/
   outp(CNTER0,FULL);                /* write full count */
   loop_count = 0;
   do
   {
       if (loop_count++ >= 1000) /* max no. of test loops */
       {
           breakstat = 1;        /* set flag to show break*/
           break;                          /* leave loop */
       }
   }
   while ((c2 = inp(CNTER0)) ^ FULL == 0);/*tst0 counting?*/

   outp(CNTRL1,CNT0_MODE);
}

/*************************************************************/
/*   PROCEDURE:dta_reset()                                   */
/*   FUNCTION:Resets the disk transfer area to what it was*/
/*         origrnally, in the program segment prefix.      */
/*   GLOBAL VARS:none                                        */
/*   CALLS:none                                              */
/*   PARAMETERS:none                                         */
/*   RETURNS:none                                            */
/*************************************************************/

dta_reset()
{
   flsetdta(&old_dta_ads);/* setup dta address to default */
}

/*************************************************************/
/*   PROCEDURE:dta_set()                                     */
/*   FUNCTION:The procedure allocates a buffer from memory*/
/*         and sets up the disk transfer address to point */
/*         to this after saving the old dta.               */
/*   GLOBAL VARS:old_dta_ads,new_dta_ads,pbuffer.            */
/*   CALLS:none                                              */
/*   PARAMETERS:none                                         */
/*   RETURNS:none                                            */
/*************************************************************/

dta_set()
{
   flretdta(&old_dta_ads);           /* fetch DTA address */
   if (NIL != (pbuffer = malloc(512))) /* allocate mem buf*/
   {
       utabsptr(pbuffer,&new_dta_ads);/* set pnter to buf */
       flsetdta(&new_dta_ads);    /* write pointer to DTA */
   }
}
```

```
/*************************************************************/
/*   PROCEDURE:end_rec()                                     */
/*   FUNCTION:this function resets the receiving sectionof*/
/*         the device and the interrupt counter.             */
/*   GLOBAL VARS:mask,rx_flag                                */
/*   CALLS:none                                              */
/*   PARAMETERS:none                                         */
/*   RETURNS:none                                            */
/*************************************************************/

end_rec()
{
   outp(CNTRL1,CNT0_MODE);          /* reset int counter   */
   outp(RST_LCH,mask |= ~REC_BITS);/*reset receiver device*/
   rx_flag = 0;
}

/*************************************************************/
/*   PROCEDURE:end_trx()                                     */
/*   FUNCTION:this function resets the transmitter section*/
/*               of the device.                              */
/*   GLOBAL VARS:mask,tx_fag_tx.                             */
/*   CALLS:none                                              */
/*   PARAMETERS:none                                         */
/*   RETURNS:none                                            */
/*************************************************************/

end_trx()
{
   outp(RST_LCH,mask |= ~TRX_BIT);/* reset transmit device*/
   tx_flag_tx = 0;
}

/*************************************************************/
/*   PROCEDURE:erase_line()                                  */
/*   FUNCTION:this function is a general purpose screen   */
/*         utility function to erase all characters betwn */
/*         'col1' and 'col2' on 'rownum' using variable   */
/*         ANSI escape sequences and outputting blank     */
/*         characters.                                     */
/*   GLOBAL VARS:none.                                       */
/*   CALLS:none                                              */
/*   PARAMETERS:rownum,col1,col2.                            */
/*   RETURNS:none                                            */
/*************************************************************/
```

```
erase_line(rownum,col1,col2)

    int rownum,col1,col2;
{
    int i;

    printf("\x1b[%d;%df",rownum,col1);/* position curscol1*/
    for (i = col1;i <= col2;i++) /* output blanks until   */
        printf(" ");            /* column 2 is reached    */
}

/*********************************************************/
/*   PROCEDURE:curs_on()                                 */
/*   FUNCTION:This procedure checks which channel is being*/
/*         used at the present time and selects which    */
/*         menu procedure to call.                       */
/*   GLOBAL VARS:channel_flag.                           */
/*   CALLS:none                                          */
/*   PARAMETERS:none                                     */
/*   RETURNS:none                                        */
/*********************************************************/

inj_err()
{
   if (channel_flag != 0)
      inj_err_tx();              /* if tx channel active */
   else
      inj_err_rx();  ,           /* if rx channel active */
}

/*********************************************************/
/*   PROCEDURE:inj_err_tx()                              */
/*   FUNCTION:this function displays a menu for injection */
/*        of errors and allows user choice as to what rate*/
/*        of error injection to use. the function then   */
/*        writes the mode and count to the error injection*/
/*        counter on the board.                          */
/*   GLOBAL VARS:c3.                                     */
/*   CALLS:none                                          */
/*   PARAMETERS:none                                     */
/*   RETURNS:none                                        */
/*********************************************************/

inj_err_tx()
{
   c3 = inj_wind(5,45);
   c3 = tolower(c3);             /* get user response char*/
   outp(CNTRL1,CNT2_MODE);       /* output mode to count2 */
   switch (c3)                   /* test user response    */
   {
      case 'a':
           outp(CNTER2,ACNT_LO);/* output count for 1    */
           outp(CNTER2,ACNT_HI);/* error injection / 64k */
           break;
```

```
        case 'b':
              outp(CNTER2,BCNT_LO);/* output count for 2    */
              outp(CNTER2,BCNT_HI);/* error injection / 64k */
              break;
        case 'c':
              outp(CNTER2,CCNT_LO);/* output count for 5    */
              outp(CNTER2,CCNT_HI);/* error injection / 64k */
              break;
        case 'd':
              outp(CNTER2,DCNT_LO);/* output count for 10   */
              outp(CNTER2,DCNT_HI);/* error injection / 64k */
              break;
        case 'e':
              outp(CNTER2,ECNT_LO);/* output count for 100  */
              outp(CNTER2,ECNT_HI);/* error injection / 64k */
              break;
         }
}

/*************************************************************/
/*   PROCEDURE:inj_err_rx()                                  */
/*   FUNCTION:this function displays a menu for injection */
/*        of errors and allows user choice as to what rate*/
/*        of error injection to use. the function then    */
/*        tansmits the formatted commands across the link */
/*        to implement the error injection on that channel*/
/*   GLOBAL VARS:c3,mess_buf.                                */
/*   CALLS:inj_wind(),trx_buf().                             */
/*   PARAMETERS:none                                         */
/*   RETURNS:none                                            */
/*************************************************************/

inj_err_rx()
{
   c3 = inj_wind(5,6);
   c3 = tolower(c3);              /* get user response char*/
   switch (c3)                    /* test user response    */
   {
      case 'a':
            mess_buf = INJ_BIT * 256 + 'a'; /* create word*/
            trx_buf(ESC_COM,mess_buf); /* trx comd message*/
            break;
      case 'b':
            mess_buf = INJ_BIT * 256 + 'b';/* create word */
            trx_buf(ESC_COM,mess_buf); /* trx comd message*/
            break;
      case 'c':
            mess_buf = INJ_BIT * 256 + 'c';/* create word */
            trx_buf(ESC_COM,mess_buf); /* trx comd message*/
            break;
      case 'd':
            mess_buf = INJ_BIT * 256 + 'd';/* create word */
            trx_buf(ESC_COM,mess_buf); /* trx comm message*/
            break;
```

```
      case 'e':
            mess_buf = INJ_BIT * 256 + 'e';/* create word */
            trx_buf(ESC_COM,mess_buf); /* trx comm message*/
            break;
       }
}

/*********************************************************/
/*   PROCEDURE:menu1()                                   */
/*   FUNCTION:This procedure tests which channel is being */
/*        worked with and calls the appropriate menu proc */
/*        It also calls the procs which create the screen */
/*        display and prints an appropriate message if    */
/*        a device is not connected.                      */
/*   GLBLVARS:c1,breakstat,channel_flag,rx_flag_tx,rx_flag*/
/*   CALLS:tx_table(),rx_table(),mess_table(),menu1_rx(), */
/*        menu1_tx().                                     */
/*   PARAMETERS:none                                     */
/*   RETURNS:none                                        */
/*********************************************************/

menu1()
{
   int i;

   tx_table();                      /*tx channel stat table*/
   rx_table();                      /*rx channel stat table*/
   mess_table();                    /* screen message area */
   printf("\x1b[20;36f\xAE ANALINK \xAF");/*table heading */
   if (breakstat != 0)                        /* no device! */
     printf("\x1b[23;22fDEVICE NOT CONNECTED PLEASE END !");
   do
   {
       if (channel_flag != 0)
       {
           menu1_tx();           /* if tx channel active & */
           if ((rx_flag_tx != 0) && (c1 != 'z')) /* rxing */
           menu2();                     /* goto menu2_tx   */
       }
       else
       {
           menu1_rx();             /* if rx channel active */
           if ((rx_flag != 0) && (c1 != 'z')) /* if rxing */
           menu2();                     /* goto menu2_rx  */
       }
   }
   while (c1 != 'z');                   /* no 'z' entered */
}
```

```
/*************************************************************/
/*   PROCEDURE:menu1_tx()                                    */
/*   FUNCTION:this function displays the initial menu for */
/*        the transmit channel side of the screen and     */
/*        offers the user the choice of operation modes   */
/*        such as starting or stopping transmitting,start-*/
/*        ing or stopping receiving or exiting the prog   */
/*        the status of the flags is tested to offer      */
/*        only valid choices to the user. once the recv   */
/*        mode has been enabled the function calls        */
/*        menu2 automaticallly thus the transmitter on    */
/*        the board should be started before the receiver */
/*        The receiver started is the one affecting       */
/*        this channel i.e. other side the link and thus  */
/*        the appropriate command is formatted and sent.  */
/*   GLOBAL VARS:c1,mess_buf,rx_flag_tx.                   */
/*   CALLS:menu1_wind(),start_trx(),end_trx(),trx_buf().   */
/*   PARAMETERS:none                                       */
/*   RETURNS:none                                          */
/*************************************************************/

menu1_tx()
{

   c1 = menu1_wind(5,45);          /* display menu tx side */
   c1 = tolower(c1);                      /* fetch response*/
   switch(c1)
   {
       case 't':                          /* transmit start*/
            start_trx();                  /* if t entered  */
            break;
       case 'h':
            end_trx();                    /* end transmit  */
            break;                        /*if so required */
       case 'r':
            mess_buf = REC_BITS * 256 + 0;/* transmit com*/
            trx_buf(ESC_COM,mess_buf);    /* over link to */
            rx_flag_tx = 1;               /* begin receiv */
            break;                        /* if r entered */
       case 'c':
            mess_buf = REC_BITS * 256 + RESET;/* trx com */
            trx_buf(ESC_COM,mess_buf);    /* over link to  */
            rx_flag_tx = 0;               /* end receiving */
            break;                        /* if c entered  */
       }
}
```

```
/*********************************************************/
/*   PROCEDURE:menu1_rx()                                */
/*   FUNCTION:this function displays the initial menu for */
/*        the transmit channel side of the screen and    */
/*        offers the user the choice of operation modes  */
/*        such as starting or stopping transmitting,start-*/
/*        ing or stopping receiving or exiting the prog  */
/*        the status of the flags is tested to offer     */
/*        only valid choices to the user. once the recv  */
/*        mode has been enabled the function calls       */
/*        menu2 automaticallly thus the transmitter on   */
/*        the board should be started before the receiver */
/*        The transmitter started is the one affecting   */
/*        this channel i.e. other side the link and thus */
/*        the appropriate command is formatted and sent. */
/*   GLOBAL VARS:c1,mess_buf,tx_flag.                    */
/*   CALLS:menu1_wind(),trx_buf(),start_res(),end_rec().  */
/*   PARAMETERS:none                                     */
/*   RETURNS:none                                        */
/*********************************************************/

menu1_rx()
{

   c1 = menu1_wind(5,6);
   c1 = tolower(c1);                    /* fetch response */
   switch(c1)
   {
       case 't':                        /* transmt start  */
            mess_buf = TRX_BIT * 256 + 0; /* transmit com*/
            trx_buf(ESC_COM,mess_buf); /* over link to   */
            tx_flag = 1;               /* start transmt  */
            break;                     /* if t enetered  */
       case 'h':
            mess_buf = TRX_BIT * 256 + RESET; /* trx com */
            trx_buf(ESC_COM,mess_buf); /* over link to   */
            tx_flag = 0;               /* end transmit   */
            break;                     /* if h entered   */
       case 'r':
            start_rec();               /* start receiving*/
            break;                     /* if r entered   */
       case 'c':
            end_rec();                 /* end receive if */
            break;                     /* c entered      */
   }
}
```

```
/*************************************************************/
/*   PROCEDURE:menu2()                                       */
/*   FUNCTION:This procedure determines which channel is     */
/*        active at present and calls the appropriate menu   */
/*        routine until an ending character is entered.      */
/*   GLOBAL VARS:c2,channel_flag.                            */
/*   CALLS:menu2_rx(),menu2_tx().                            */
/*   PARAMETERS:none                                         */
/*   RETURNS:none                                            */
/*************************************************************/

menu2()
{
   do
   {
      if (channel_flag != 0)
         menu2_tx();                    /* if tx channel  */
      else
         menu2_rx();                    /* if rx channel  */
   }
   while (c2 != 'z');                   /* no 'z' entered */
}

/*************************************************************/
/*   PROCEDURE:menu2_tx()                                    */
/*   FUNCTION:This function displays the display control     */
/*        menu for the transmitting channnel side            */
/*        and offers the user display options such as        */
/*        sample time selection, threshold selection,        */
/*        continuous analysis display and also allows        */
/*        access to the error injection menu from which      */
/*        error rates can be selected to inject into the     */
/*        transmit data stream and logging of results to     */
/*        disk is also accessed from this menu.              */
/*   GLOBAL VARS:c2,dump_flag,inj_flag_tx,channel_flag.      */
/*   CALLS:menu2_wind(),cr_file(),set_sample(),inj_err()     */
/*         set_tx_threshold(),running().                     */
/*   PARAMETERS:none                                         */
/*   RETURNS:none                                            */
/*************************************************************/

menu2_tx()
{
   int i;

   do
   {
       c2 = menu2_wind(5,45);       /* display window menu*/
       c2 = tolower(c2);            /* and fetch response */
```

```
        switch (c2)
        {
           case 'd':
                 cr_file();            /* create a log file */
                 dump_flag = 1;         /* set logging flag */
                 break;
           case 'e':
                 outp(RST_LCH,mask |= ~INJ_BIT);
                 inj_flag_tx = 0;          /* end error     */
                 break;                    /* inject mode   */
           case 'i':
                 inj_err();                /* begin the err */
                 outp(RST_LCH,mask &= INJ_BIT);/* injection*/
                 inj_flag_tx = 1;
                 break;
           case 'k':
                 fclose(fpoint);      /* close log file     */
                 dump_flag = 0;       /* reset logging flag */
                 break;
           case 'r':
                 running();        /* start running display */
                 break;
           case 's':
                 set_sample();          /* sample time menu */
                 break;
           case 't':
                 set_tx_threshold();      /* threshold menu */
                 break;
           case 'x':
                 channel_flag = 0;           /* switch flag */
                 break;                      /* for channel */
           case 'y':
                 channel_flag = 1;           /* switch flag */
                 break;                      /* for channel */
        }
     }
     while ((c2 != 'z') && (channel_flag != 0));/* end menu */
}                                               /* on exit  */
```

```
/*********************************************************/
/*   PROCEDURE:menu2_rx()                                */
/*   FUNCTION:This function displays the display control */
/*        menu for the receiving channnel side.          */
/*        and offers the user display options such as     */
/*        sample time selection, threshold selection,    */
/*        continuous analysis display and also allows    */
/*        access to the error injection menu from which  */
/*        error rates can be selected to inject into the */
/*        transmit data stream and logging of results to */
/*        disk is also accessed from this menu.          */
/*   GLOBAL VARS:c2,dumpflag,inj_flag,channel_flag.      */
/*   CALLS:menu2_wind(),cr_file(),trx_buf(),inj_err(),   */
/*         running(),set_sample(),set_threshold()        */
/*   PARAMETERS:none                                     */
/*   RETURNS:none                                        */
/*********************************************************/

menu2_rx()
{
   int i;

   do
   {
       c2 = menu2_wind(5,6);       /* display window menu2 */
       c2 = tolower(c2);           /* get response to menu */
       switch (c2)
       {
       case 'd':
             cr_file();                /* openlogging file */
             dump_flag = 1;            /* set logging flag */
             break;
       case 'e':
             mess_buf = INJ_BIT * 256 + RESET;/*format com*/
             trx_buf(ESC_COM,mess_buf);   /* transmit com */
             inj_flag = 0;                /* set flag     */
             break;                       /* inject mode  */
       case 'i' :
             inj_err();                   /* begin the err*/
             inj_flag = 1;                /* injection    */
             break;
       case 'k':
             fclose(fpoint);         /* close logging file */
             dump_flag = 0;          /* reset logging flag */
             break;
       case 'r':
             running();            /* start running display*/
             break;
       case 's':
             set_sample();              /* set sample time */
             break;
       case 't':
             set_threshold();           /* set threshold*/
             break;
```

```
        case 'x':
              channel_flag = 0;              /* switch flag */
              break;                        /* for channel  */
        case 'y':
              channel_flag = 1;             /* switch flag  */
              break;                        /* for static   */

        }
    }
    while ((c2 != 'z') && (channel_flag == 0));/* end menu */
}                                                /* on exit  */

/*************************************************************/
/*    PROCEDURE:restrt()                                     */
/*    FUNCTION:This procedure resets the device reset latch*/
/*         as well as resettimg the error count interrupt  */
/*         generating counter                              */
/*    GLOBAL VARS:mask.                                      */
/*    CALLS:none                                             */
/*    PARAMETERS:none                                        */
/*    RETURNS:none                                           */
/*************************************************************/

restrt()
{
   outp(RST_LCH,mask |= RESET);  /* reset all on board    */
   outp(CNTRL1,CNT0_MODE);       /* reset intrp counter   */
   outp(CNTRL1,CNT2_MODE);       /* reset inj err counter */
}

/*************************************************************/
/*    PROCEDURE:running()                                    */
/*    FUNCTION:this function implements running displays of*/
/*         the error statistics and calls the functions to */
/*         print the statistics table as well as the single*/
/*         analysis routine repeatedly at timed intervals  */
/*         which are gauged by calls to the time function  */
/*    GLBL VAR:c2 average,tx_average,loop_count,grand_total*/
/*         tx_grand_total,above_thresh,errsec,tx_err_sec,  */
/*         tx_total,total2,tx_total2,sample_time,dump_flag */
/*         inj_flag.                                       */
/*    CALLS:single(),runprint(),erase_line().             */
/*    PARAMETERS:none                                        */
/*    RETURNS:none                                           */
/*************************************************************/
```

```
running()
{
   unsigned long    sec1,sec2;

   average = 0;
   tx_average = 0;            /* initialise all variables */
   loop_count = 0;            /* at start of each run     */
   grand_total = 0;
   tx_grand_total = 0;
   above_thresh = 0;
   tx_above_thresh = 0;
   err_sec = 0;
   tx_err_sec = 0;
   tx_total = 0;

   printf("\x1b[23;37f<RUNNING>");  /* messages to screen */
   if (inj_flag_tx != 0)
      printf ("\x1b[23;61f<INJECTING>");
   if (inj_flag != 0)
      printf ("\x1b[23;11f<INJECTING>");
   if (sample_time >= 2)
      if (dump_flag != 0)
         printf("\x1b[23;23f<LOGGING>");
   do
   {
      total2 = 0;                  /* initialise on each */
      tx_total2 = 0;               /* second's measuring */
      sec1 = time(&ltime);         /* get system time    */
      sec2 = sec1;                 /* both times the same*/
      do
      {

         do
         {
         }
         while (sec2 == time(&ltime));/* till 1 sec later */
         single();                /* do 1sec err analysis */
         loop_count++;
         sec2 = time(&ltime);      /* fetch system time 2 */
      }
      while (((sec2 - sec1) < sample_time) &&
                  (kbhit() == 0));
      run_print();                        /* print analysis */
   }
   while ((kbhit()) == 0);         /* do until keyb entry */
   c2 = getch();
   erase_line(23,10,78);           /* erase message line */
}
```

```
/*************************************************************/
/*   PROCEDURE:single()                                      */
/*   FUNCTION:this function performs an analysis of last     */
/*        second of errors by accessing the array that the   */
/*        machine coded interrupt service routine is         */
/*        writing to and, using the counters and frequency   */
/*        variables,provides a statistical basis for         */
/*        runprint.                                          */
/*   GLOBAL VARS:total,total2,tx_total2,err_sec,tx_err_sec   */
/*               above_thresh,tx_above_thresh,erray[].       */
/*   CALLS:trx_buf().                                        */
/*   PARAMETERS:none                                         */
/*   RETURNS:none                                            */
/*************************************************************/

single()
{
   int             i;
   unsigned   total;

   total = 0;
   for (i = 0; i < MAX_SAMPLE; i++)
   {
        total += erray[i];          /*initialise the array*/
   }
   total2 += total;                 /* rx running total   */
   tx_total2 += tx_total;           /* tx running total   */
   if (total > 0) err_sec++;        /* count errored secs */
   if (tx_total > 0) tx_err_sec++;
   if (total > threshold) above_thresh++;
   if (tx_total > tx_threshold) tx_above_thresh++;
      trx_buf(ESC_DATA,total);
}

/*************************************************************/
/*   PROCEDURE:run_print()                                   */
/*   FUNCTION:This procedure calsulates and prints the       */
/*        various statistical values for the error counts    */
/*        and dumps to screen and also if necessary, disk    */
/*   GLBL VARS:grand_total,total2,tx_grand_total,tx_total2   */
/*        loop_count,average,frequency,tx_average            */
/*        sample_time,ppm,tx_ppm,fpoint,dumpflag,            */
/*        above_thresh,tx_above_thresh,threshold,            */
/*        tx_threshold.                                      */
/*   CALLS:none                                              */
/*   PARAMETERS:none                                         */
/*   RETURNS:none                                            */
/*************************************************************/
```

```
run_print()
{
   float    ppm;

   grand_total += total2;           /* accumulative total */
   tx_grand_total += tx_total2;
   if (loop_count == 0) loop_count = 1;/* prevent /0 error*/
   average = 1E6 * grand_total / loop_count / frequency;
   tx_average = 1E6 * tx_grand_total / loop_count /
                frequency;
   ppm = 1E6 * total2 / frequency / sample_time;/* ppm err*/
   tx_ppm = 1E6 * tx_total2 / frequency / sample_time;

   if (tx_ppm > 2E6) tx_ppm = 2E6;
   if (tx_average > 2E6) tx_average = 2E6;

   printf("\x1b[5;62f %7d",sample_time);
   printf("\x1b[6;62f %7d",loop_count);
   printf("\x1b[7;62f %7d",tx_threshold);
   printf("\x1b[8;62f %7d",tx_above_thresh);
   printf("\x1b[9;62f %7d",loop_count - tx_above_thresh);
   printf("\x1b[10;62f %7d",tx_err_sec);
   printf("\x1b[11;62f %7d",loop_count - tx_err_sec);
   printf("\x1b[12;62f %7d",
          100 - 100*tx_err_sec/loop_count);
   printf("\x1b[13;62f %7d",100*tx_err_sec/loop_count);
   printf("\x1b[14;58f %11.4f",tx_ppm);
   printf("\x1b[15;58f %11.0f",tx_grand_total);
   printf("\x1b[16;58f %11.4f",tx_average);

   printf("\x1b[5;23f %7d",sample_time);
   printf("\x1b[6;23H %7d",loop_count);
   printf("\x1b[7;23H %7d",threshold);
   printf("\x1b[8;23H %7d",above_thresh);
   printf("\x1b[9;23H %7d",loop_count - above_thresh);
   printf("\x1b[10;23H %7d",err_sec);
   printf("\x1b[11;23H %7d",loop_count - err_sec);
   printf("\x1b[12;23H %7d",100 - 100*err_sec/loop_count);
   printf("\x1b[13;23H %7d",100*err_sec/loop_count);
   printf("\x1b[14;19H %11.4f",ppm);
   printf("\x1b[15;19H %11.0f",grand_total);
   printf("\x1b[16;19H %11.4f",average);
   if (sample_time >= 2)
      if (dump_flag != 0)
      {
         fprintf(fpoint, "%9d %10d %10d %10d",
                 loop_count,threshold,above_thresh,
                 loop_count-err_sec);
         fprintf(fpoint, " %11.4f %11.0f %11.4f\n",
                 ppm,grand_total,average);
      }
}
```

```
/*************************************************************/
/*   PROCEDURE:set_sample()                                  */
/*   FUNCTION:Allows user to select sample time from range*/
/*        of possibilities.                                  */
/*   GLOBAL VARS:c2,mask,loop_count,breakstat.               */
/*   CALLS:sample_wind().                                    */
/*   PARAMETERS:none                                         */
/*   RETURNS:none                                            */
/*************************************************************/

set_sample()
{
   if (channel_flag != 0)
   c3 = sample_wind(5,45);
   else
       c3 = sample_wind(5,6);
   c3 = tolower(c3);              /* get user response char*/
   switch (c3)                    /* test user response    */
   {
      case 'a':
            sample_time = 1;
            break;
      case 'b':
          sample_time = 2;
            break;
      case 'c':
          sample_time = 5;
            break;
      case 'd':
          sample_time = 10;
            break;
      case 'e':
          sample_time = 60;
            break;
   }
}

/*************************************************************/
/*   PROCEDURE:set_thresh()                                  */
/*   FUNCTION:Allows the user to select a new threshold of*/
/*        errors to report on for the receiving channel.     */
/*   GLOBAL VARS:c3.                                         */
/*   CALLS:thresh_wind()                                     */
/*   PARAMETERS:none                                         */
/*   RETURNS:none                                            */
/*************************************************************/
```

```
set_threshold()
{
   c3 = thresh_wind(5,6);
   c3 = tolower(c3);              /* get user response char*/
   switch (c3)                    /* test user response    */
   {
      case 'a':
           threshold = 0;
           break;
      case 'b':
           threshold = 1;
           break;
      case 'c':
           threshold = 5;
           break;
      case 'd':
           threshold = 10;
           break;
      case 'e':
           threshold = 100;
           break;
   }
}

/*************************************************************/
/*   PROCEDURE:set_tx_threshold()                            */
/*   FUNCTION:Allows the user to select a new threshold of*/
/*        errors to report on for the transmit channel.      */
/*   GLOBAL VARS:c3.                                         */
/*   CALLS:thresh_wind().                                    */
/*   PARAMETERS:none                                         */
/*   RETURNS:none                                            */
/*************************************************************/

set_tx_threshold()
{
   c3 = thresh_wind(5,45);
   c3 = tolower(c3);              /* get user response char*/
   switch (c3)                    /* test user response    */
   {
      case 'a':
           tx_threshold = 0;
           break;
      case 'b':
           tx_threshold = 1;
           break;
      case 'c':
           tx_threshold = 5;
           break;
      case 'd':
           tx_threshold = 10;
           break;
```

```
      case 'e':
           tx_threshold = 100;
           break;
   }
}

/*********************************************************/
/*   PROCEDURE:start_err_count()                         */
/*   FUNCTION:This function sets up the error counter i.e.*/
/*         counter 1 on the 8254 on the board and writes  */
/*         the initial 'full' count to it.                */
/*   GLOBAL VARS:none.                                   */
/*   CALLS:none                                          */
/*   PARAMETERS:none                                     */
/*   RETURNS:none                                        */
/*********************************************************/

start_err_count()
{
   outp(RST_LCH,RESET);          /* reset device completely*/
   outp(CNTRL1,CNT1_MODE);       /* error counter mode     */
   outp(CNTER1,FULL);            /* initial count for error*/
   outp(CNTER1,FULL);            /* counter                */
}

/*********************************************************/
/*   PROCEDURE:start_rec()                               */
/*   FUNCTION:this function starts the receiver on board  */
/*         by setting the appropriate bits in the reset   */
/*         latch and sets the interrupt counter accord to */
/*         the frequency of the chip installed (counter 0)*/
/*   GLOBAL VARS:rx_flag,mask,frequency.                 */
/*   CALLS:none                                          */
/*   PARAMETERS:none                                     */
/*   RETURNS:none                                        */
/*********************************************************/

start_rec()
{
   outp(RST_LCH,mask &= REC_BITS);  /* set receiver bits  */
   outp(CNTRL1,CNT0_MODE);         /* setmode of counter 0 */
   if (frequency == FREQUENCY8)
   {                               /* set interrupt count  */
       outp(CNTER0,INT_CNT_LO8);   /* according to the freq*/
       outp(CNTER0,INT_CNT_HI8);   /* -uency variable      */
   }                               /* with least sig byte  */
   else                            /* first and most sig   */
   {                               /* byte last            */
       outp(CNTER0,INT_CNT_LO);
       outp(CNTER0,INT_CNT_HI);
   }
   rx_flag = 1;
}
```

```
/*************************************************************/
/*   PROCEDURE:start_trx()                                   */
/*   FUNCTION:this function enables the transmitter          */
/*            pseudrandom bit sequence generator that is     */
/*            the transmitter by setting the appropriate     */
/*            bit in the reset latch as well as enabling     */
/*            the on board clock to reach the output         */
/*            circuitry.                                     */
/*   GLOBAL VARS:mask,tx_flag_tx.                            */
/*   CALLS:none                                              */
/*   PARAMETERS:none                                         */
/*   RETURNS:none                                            */
/*************************************************************/

start_trx()
{
   outp(RST_LCH,mask &= STRT_CLK);     /* clock reset bit  */
   outp(RST_LCH,mask &= TRX_BIT);    /* transmit reset bit */
   tx_flag_tx = 1;
}
```

```
/************************************************/
/*   COMMENU:MENU PROGRAM FOR ANALINK           */
/************************************************/

/************************************************/
/*   INCLUDE FILES                              */
/************************************************/

#include <bscreen.h>
#include <bwindow.h>
#include <stdio.h>

/************************************************/
/*   DEFINES                                    */
/************************************************/

#define       TRX_BIT         0xFB
#define       REC_BITS        0xFC
#define       INJ_BIT         0xEF

/************************************************/
/*   GLOBAL VARIABLES                           */
/************************************************/

extern  unsigned char c1,c2,c3,mask,channel_flag,
        tx_flag,rx_flag,inj_flag,dump_flag,
        tx_flag_tx,rx_flag_tx,inj_flag_tx;

/************************************************/
/*   PROCEDURES                                 */
/************************************************/

/******************************************************/
/*   PROCEDURE:tx_table()                             */
/*   FUNCTION:To print out the transmit channel table */
/*        for statistical analysis using ANSI         */
/*        escape sequences defined in the IBM         */
/*        TECHNICAL REFERENCE.                        */
/*   GLOBAL VARS:none                                 */
/*   CALLS:none                                       */
/*   PARAMETERS:none                                  */
/*   RETURNS:none                                     */
/******************************************************/

tx_table()
{
    int i;

    printf("\x1b[2J");                  /* clear screen   */
    scbox(1,40,18,79,15,0,NORMAL);      /* create box     */
    printf("\x1b[2;59fTRANSMIT");       /* print heading  */
    for (i=3;i<=9;i++)                  /* draw columns   */
    printf("\x1b[%1d;56f\xb3\x1b[%1d;70f\xb3",i,i);
```

```
    for (i=10;i<=18;i++)
        printf("\x1b[%2d;56f\xb3\x1b[%1d;70f\xb3",i,i);
    printf("\x1b[2;56f\xd1\x1b[19;56f\xcf");
    printf("\x1b[2;70f\xd1\x1b[19;70f\xcf");
    printf

    ("\x1b[3;43fVariable\x1b[3;58fValue\x1b[3;72fUnits");
    printf
    ("\x1b[5;43fSAMPLE TIME \x1b[5;63f-\x1b[5;72fSECS  ");
    printf
    ("\x1b[6;43fSECS IN RUN \x1b[6;63f-\x1b[6;72fSECS  ");
    printf
    ("\x1b[7;43fTHRESHOLD  T\x1b[7;63f-\x1b[7;72fERRS  ");
    printf
    ("\x1b[8;43fSECS ABOVE T\x1b[8;63f-\x1b[8;72fSECS  ");
    printf
    ("\x1b[9;43fSECS BELOW T\x1b[9;63f-\x1b[9;72fSECS  ");
    printf
    ("\x1b[10;43fERRORED SECS\x1b[10;63f-\x1b[10;72fSECS");
    printf
    ("\x1b[11;43fERR FREE SEC\x1b[11;63f-\x1b[11;72fSECS");
    printf
    ("\x1b[12;43fPERCENT EFS \x1b[12;63f-\x1b[12;72fPCNT");
    printf
    ("\x1b[13;43fPCNT ERR SEC\x1b[13;63f-\x1b[13;72fPCNT");
    printf
    ("\x1b[14;43fERR PPM NOW \x1b[14;63f-\x1b[14;72fPPM ");
    printf
    ("\x1b[15;43fTOTAL ERRS  \x1b[15;63f-\x1b[15;72fERRS");
    printf
    ("\x1b[16;43fAVERAGE PPM \x1b[16;63f-\x1b[16;72fPPM ");
}

/*********************************************************/
/*   PROCEDURE:rx_table()                                */
/*   FUNCTION:To print out the receive channel table     */
/*        for statistical analysis using ANSI            */
/*        escape sequences defined in the IBM            */
/*        TECHNICAL REFERENCE.                           */
/*   GLOBAL VARS:none                                    */
/*   CALLS:none                                          */
/*   PARAMETERS:none                                     */
/*   RETURNS:none                                        */
/*********************************************************/

rx_table()
{
    int i;

    scbox(1,1,18,39,15,0,NORMAL);       /* create box    */
    printf("\x1b[2;20fRECEIVER");       /* print heading */
    for (i=3;i<=9;i++)                  /* draw columns  */
    printf("\x1b[%1d;17f\xb3\x1b[%1d;31f\xb3",i,i);
```

```
    for (i=10;i<=18;i++)
        printf("\x1b[%2d;17f\xb3\x1b[%1d;31f\xb3",i,i);
    printf("\x1b[2;17f\xd1\x1b[19;17f\xcf");
    printf("\x1b[2;31f\xd1\x1b[19;31f\xcf");

    printf
    ("\x1b[3;4fVariable\x1b[3;19fValue\x1b[3;33fUnits");

    printf
    ("\x1b[5;4fSAMPLE TIME \x1b[5;24f-\x1b[5;33fSECS  ");
    printf
    ("\x1b[6;4fSECS IN RUN \x1b[6;24f-\x1b[6;33fSECS  ");
    printf
    ("\x1b[7;4fTHRESHOLD  T\x1b[7;24f-\x1b[7;33fERRS  ");
    printf
    ("\x1b[8;4fSECS ABOVE T\x1b[8;24f-\x1b[8;33fSECS  ");
    printf
    ("\x1b[9;4fSECS BELOW T\x1b[9;24f-\x1b[9;33fSECS  ");
    printf
    ("\x1b[10;4fERRORED SECS\x1b[10;24f-\x1b[10;33fSECS");
    printf
    ("\x1b[11;4fERR FREE SEC\x1b[11;24f-\x1b[11;33fSECS");
    printf
    ("\x1b[12;4fPERCENT EFS \x1b[12;24f-\x1b[12;33fPCNT");
    printf
    ("\x1b[13;4fPCNT ERR SEC\x1b[13;24f-\x1b[13;33fPCNT");
    printf
    ("\x1b[14;4fERR PPM NOW \x1b[14;24f-\x1b[14;33fPPM ");
    printf
    ("\x1b[15;4fTOTAL ERRS  \x1b[15;24f-\x1b[15;33fERRS");
    printf
    ("\x1b[16;4fAVERAGE PPM \x1b[16;24f-\x1b[16;33fPPM ");
}

/*********************************************************/
/*   PROCEDURE:mess_table()                              */
/*   FUNCTION:To print out the message table section     */
/*        for program messages etc. using ANSI           */
/*        escape sequences defined in the IBM            */
/*        TECHNICAL REFERENCE.                           */
/*   GLOBAL VARS:none                                    */
/*   CALLS:none                                          */
/*   PARAMETERS:none                                     */
/*   RETURNS:none                                        */
/*********************************************************/

mess_table()
{
    scbox(20,1,24,79,15,0,NORMAL);      /* create box   */
    printf("\x1b[25;36fMESSAGES");      /* print heading*/
}
```

```
/*******************************************************/
/*   PROCEDURE:menu2_wind()                            */
/*   FUNCTION:To create and print out the window and   */
/*        menu for the display  menu on either         */
/*        side of the screen, to return a user         */
/*        response or to return an error value.        */
/*   GLOBAL VARS:channel_flag,inj_flag,inj_fag_tx,     */
/*               dump_flag.                            */
/*   CALLS:none                                        */
/*   PARAMETERS:c_row,c_col                            *
/*   RETURNS:b_wnerr or *response                      */
/*******************************************************/

char menu2_wind(c_row,c_col)

int c_row,c_col;
{
    BWINDOW *pwin;
    BORDER  bord;
    WHERE   location;
    int     mode,columns,act_page;
    int     cursor_was_off,row,col,high,low;
    char    response[2];
    int     scan,bnum = 5;

                /* Create the window structure in memory.*/

    pwin = wncreate(10,          /* Height of data area   */
              26,                /* Width    of data area */
              REVERSE);        /* Attributes of data area */
    if (pwin == NIL)
    return b_wnerr;                      /* Quit if failure */
                            /* (b_wnerr records the most */
                                /* recent window error.) */

                              /* Choose style of border.  */
    bord.type = 1;          /* Box drawn with single lines */
    bord.attr = RED;                       /* Red on black */

                /* Choose where to display the window:the */
                /* active page on current display device. */

    location.dev    = scmode(&mode,&columns,&act_page);
    location.page   = act_page;
    location.corner.row =  c_row;
    location.corner.col = c_col;

              /* Retrieve former cursor position and size*/
    scpage(act_page);
                               /* Actually display window*/
    if (NIL == wndsplay(pwin,&location,&bord))
       return b_wnerr;                 /* Quit if failure*/
```

```
                    /* Display a message on the window*/
    wnwrap(0,"\tENTER\12\r\12\r",
            -1,-1,CHARS_ONLY);                  /* heading */
    wnwrap(0," R FOR RUNNING DISPLAY\12\r",
            -1,-1,CHARS_ONLY);
    if (channel_flag == 0)                      /* rx channel */
    {
    if (inj_flag == 0)                          /* injecting */
          wnwrap(0," I FOR INJECT ERR MODE\12\r",
                   -1,-1,CHARS_ONLY);
    else                                       /* not inject */
          wnwrap(0," E TO END INJ ERR MODE\12\r",
                  -1,-1,CHARS_ONLY);
    }
    else                                       /* tx channel */
    {
    if (inj_flag_tx == 0)                      /* injecting  */
          wnwrap(0," I FOR INJECT ERR MODE\12\r",
                  -1,-1,CHARS_ONLY);
    else                                  /* not injecting */
          wnwrap(0," E TO END INJ ERR MODE\12\r",
                  -1,-1,CHARS_ONLY);
    }
    if (channel_flag != 0)                     /* tx channel */
    wnwrap(0," X FOR RECEIVER MENU\12\r",-1,-1,CHARS_ONLY);
    else                                       /* rx channel */
    wnwrap(0," Y FOR TRANSMIT MENU\12\r",-1,-1,CHARS_ONLY);

    wnwrap(0," T FOR NEW  THRESHOLD \12\r",
            -1,-1,CHARS_ONLY);
    wnwrap(0," S FOR NEW SAMPLE TIME\12\r",
            -1,-1,CHARS_ONLY);
    if (dump_flag == 0)                          /* if logging */
       wnwrap(0," D TO LOG RESULTS TO DISK\12\r",
            -1,-1,CHARS_ONLY);
    if (dump_flag != 0)                    /* if not logging*/
       wnwrap(0," K TO END LOGGING TO DISK\12\r",
              -1,-1,CHARS_ONLY);

    wnwrap(0," Z FOR ENDING\12\r",-1,-1,CHARS_ONLY);
    if (b_wnerr)
       return b_wnerr;                  /* Quit if failure */

                    /* Await a response from the user,*/
                    /* echoing keystrokes to window   */
    wnquery(response,sizeof(response),&scan);
    if (b_wnerr)
       return b_wnerr;                   /* Quit if failure*/
                  /* Remove the window & restore the*/
                  /* screen and cursor.             */
    wnremove(pwin);
    wndstroy(pwin);
    return *response;                /* return user char*/
}
```

```
/*******************************************************/
/*   PROCEDURE:menu1_wind()                            */
/*   FUNCTION:To create and print out the window and   */
/*        menu for the start up menu on either         */
/*        side of the screen, to return a user         */
/*        response or to return an error value.        */
/*   GLOBAL VARS:channel_flag,tx_flag,tx_flag_tx,      */
/*               rx_flag,rx_flag_tx.                   */
/*   CALLS:none                                        */
/*   PARAMETERS:c_row,c_col.                           */
/*   RETURNS:b_wnerr or *response.                     */
/*******************************************************/

char menu1_wind(c_row,c_col)

int c_row,c_col;
{
    BWINDOW *pwin;
    BORDER  bord;
    WHERE   location;
    int     mode,columns,act_page;
    int     cursor_was_off,row,col,high,low;
    char    response[2];
    int     scan,bnum = 5;

              /* Create the window structure in memory*/

    pwin = wncreate(10,      /* Height of data area    */
            26,              /* Width of data area     */
            REVERSE);        /* Attributes data area   */
    if (pwin == NIL)
       return b_wnerr;         /* Quit if failure.     */

    bord.type = 1;       /* Box drawn with single lines */
    bord.attr = RED;     /* Red on black when colour    */

    location.dev    = scmode(&mode,&columns,&act_page);
    location.page   = act_page;
    location.corner.row =  c_row;
    location.corner.col = c_col;

            /* Retrieve former cursor position & size*/
    scpage(act_page);
    cursor_was_off = sccurst(&row,&col,&high,&low);

                        /* Actually display the window */
    if (NIL == wndsplay(pwin,&location,&bord))
       return b_wnerr;              /* Quit if failure. */
                         /* Display a message on window*/
    wnwrap(0,"\tENTER\12\r\12\r",-1,-1,CHARS_ONLY);
```

```
    if (channel_flag == 0)              /* if tx channel*/
    {
    if (tx_flag == 0)                      /* if trx-ing*/
          wnwrap(0," T TO TRANSMIT\12\12\r",
                   -1,-1,CHARS_ONLY);
    else                                   /* if not -trx  */
          wnwrap(0," H TO STOP TX\12\12\r",
                   -1,-1,CHARS_ONLY);
    }
    else                                    /* if rx channel*/
    {
    if (tx_flag_tx == 0)                   /* if trx-ing  */
          wnwrap(0," T TO TRANSMIT\12\12\r",
                   -1,-1,CHARS_ONLY);
    else                                  /* if not trx- */
          wnwrap(0," H TO STOP TX\12\12\r",
                   -1,-1,CHARS_ONLY);
    }
    if (channel_flag == 0)             /* if tx channel */
    {
    if (rx_flag == 0)                 /* if receiving  */
     wnwrap(0,"R TO RECEIVE\12\12\r",-1,-1,CHARS_ONLY);
     else                                  /* if not rec */
     wnwrap(0,"C TO STOP RX\12\12\r",-1,-1,CHARS_ONLY);
    }
    else                                   /* if rx channel*/
    {
    if (rx_flag_tx == 0)                /* if receiving   */
       wnwrap(0,"R TO RECEIVE\12\12\r",-1,-1,CHARS_ONLY);
    else                                  /* if not recv   */
       wnwrap(0,"C TO STOP RX\12\12\r",-1,-1,CHARS_ONLY);
    }
    wnwrap(0," Z TO END\12\12\r",-1,-1,CHARS_ONLY);
    if (b_wnerr)
       return b_wnerr;                  /* Quit if failure*/

                         /* Await a response from user,*/
                         /* echoing keystrokes to the  */
                         /* window.                    */
    wnquery(response,sizeof(response),&scan);
    if (b_wnerr)
       return b_wnerr;                  /* Quit if failure*/
                        /* Remove the window & restore */
                        /* the screen and cursor.      */
    wnremove(pwin);
    sccurset(row,col);
    scpgcur(cursor_was_off,high,low,CUR_NO_ADJUST);
    if (b_wnerr)
       return b_wnerr;                /* Quit if failure*/

     wndstroy(pwin);
     return *response;
}
```

```
/*******************************************************/
/*   PROCEDURE:inj_wind()                              */
/*   FUNCTION:To create and print out the window and   */
/*        menu for the inject error menu on either     */
/*        side of the screen, to return a user         */
/*        response or to return an error value.        */
/*   GLOBAL VARS:none                                  */
/*   CALLS:none                                        */
/*   PARAMETERS:c_row,c_col                            */
/*   RETURNS:b_wnerr or *response                      */
/*******************************************************/

char inj_wind(c_row,c_col)

int c_row,c_col;
{
    BWINDOW *pwin;
    BORDER  bord;
    WHERE   location;
    int     mode,columns,act_page;
    int     cursor_was_off,row,col,high,low;
    char    response[2];
    int     scan,bnum = 5;

                /* Create the window structure in memory*/

    pwin = wncreate(10,           /* Height of data area */
                    26,           /* Width  of data area */
              REVERSE);       /* Attributes of data area */
    if (pwin == NIL)
       return b_wnerr;              /* Quit if failure. */

    bord.type = 1;        /* Box drawn with single lines */
    bord.attr = RED;                    /* Red on black */

    location.dev    = scmode(&mode,&columns,&act_page);
    location.page   = act_page;
    location.corner.row =  c_row;
    location.corner.col = c_col;

              /* Retrieve former cursor position & size*/
    scpage(act_page);
    cursor_was_off = sccurst(&row,&col,&high,&low);

                           /* Actually display window*/

    if (NIL == wndsplay(pwin,&location,&bord))
       return b_wnerr;             /* Quit if failure. */

                     /* Display a message on the window*/
    wnwrap(0,"\tERROR INJECTION MENU\12\12\rENTER\12\12\r ",
              -1,-1,CHARS_ONLY);
       wnwrap(0,"A FOR APPROX 1 ERR/ 64Kb\12\r",
              -1,-1,CHARS_ONLY);
```

```
        wnwrap(0,"B FOR APPROX   2 ERRORS \12\r",
               -1,-1,CHARS_ONLY);
        wnwrap(0,"C FOR APPROX   5 ERRORS \12\r",
               -1,-1,CHARS_ONLY);
        wnwrap(0,"D FOR APPROX  10 ERRORS \12\r",
               -1,-1,CHARS_ONLY);
        wnwrap(0,"E FOR APPROX 100 ERRORS \12\r",
               -1,-1,CHARS_ONLY);
    if (b_wnerr)
        return b_wnerr;                 /* Quit if failure */

                        /* Await a response from the user,*/
                        /* echoing keystrokes to window   */
    wnquery(response,sizeof(response),&scan);
    if (b_wnerr)
        return b_wnerr;                  /* Quit if failure*/

                        /* Remove the window & restore the*/
                        /* screen and cursor.             */
    wnremove(pwin);
    sccurset(row,col);
    scpgcur(cursor_was_off,high,low,CUR_NO_ADJUST);

    if (b_wnerr)
.       return b_wnerr;                  /* Quit if failure*/

     wndstroy(pwin);
     return *response;
}

/*********************************************************/
/*   PROCEDURE:sample_wind()                             */
/*   FUNCTION:To create and print out the window and     */
/*        menu to set up sample time on either           */
/*        side of the screen, to return a user           */
/*        response or to return an error value.          */
/*   GLOBAL VARS:none                                    */
/*   CALLS:none                                          */
/*   PARAMETERS:c_row,C_col                              */
/*   RETURNS:b_wnerr or *response                        */
/*********************************************************/

char sample_wind(c_row,c_col)

int c_row,c_col;
{
    BWINDOW *pwin;
    BORDER  bord;
    WHERE   location;
    int     mode,columns,act_page;
    int     cursor_was_off,row,col,high,low;
    char    response[2];
    int     scan,bnum = 5;
```

```
                /* Create the window structure in memory*/

    pwin = wncreate(10,             /* Height of data area */
                26,                 /* Width  of data area */
                REVERSE);       /* Attributes of data area */
    if (pwin == NIL)
       return b_wnerr;                  /* Quit if failure. */

    bord.type = 1;          /* Box drawn with single lines */
    bord.attr = RED;                        /* Red on black */

    location.dev    = scmode(&mode,&columns,&act_page);
    location.page   = act_page;
    location.corner.row =  c_row;
    location.corner.col = c_col;

              /* Retrieve former cursor position and size*/
    scpage(act_page);
    cursor_was_off = sccurst(&row,&col,&high,&low);

                              /* Actually display window */
    if (NIL == wndsplay(pwin,&location,&bord))
       return b_wnerr;                /* Quit if failure. */

                     /* Display a message on the window */
    wnwrap(0,"\tSET SAMPLE TIME MENU\12\12\rENTER\12\12\r ",
           -1,-1,CHARS_ONLY);
       wnwrap(0,"A FOR APPROX   1 SECOND \12\r",
              -1,-1,CHARS_ONLY);
       wnwrap(0,"B FOR APPROX   2 SECONDS\12\r",
              -1,-1,CHARS_ONLY);
       wnwrap(0,"C FOR APPROX   5 SECONDS\12\r",
              -1,-1,CHARS_ONLY);
       wnwrap(0,"D FOR APPROX  10 SECONDS\12\r",
              -1,-1,CHARS_ONLY);
       wnwrap(0,"E FOR APPROX  60 SECONDS\12\r",
              -1,-1,CHARS_ONLY);
    if (b_wnerr)
       return b_wnerr;                 /* Quit if failure*/
                    /* Await a response from the user, */
                    /* echoing keystrokes to window    */
    wnquery(response,sizeof(response),&scan);
    if (b_wnerr)
       return b_wnerr;                /* Quit if failure */
                    /* Remove the window & restore the */
                    /* screen and cursor.              */
    wnremove(pwin);
    sccurset(row,col);
    scpgcur(cursor_was_off,high,low,CUR_NO_ADJUST);
    if (b_wnerr)
       return b_wnerr;                /* Quit if failure */
     wndstroy(pwin);
     return *response;
}
```

```
/*******************************************************/
/*   PROCEDURE:thresh_wind()                           */
/*   FUNCTION:To create and print out the window and   */
/*        menu to set the threshold  on either         */
/*        side of the screen, to return a user         */
/*        response or to return an error value.        */
/*   GLOBAL VARS:none                                  */
/*   CALLS:none                                        */
/*   PARAMETERS:c_row,c_col                            */
/*   RETURNS:b_wnerr or *response                      */
/*******************************************************/

char thresh_wind(c_row,c_col)

int c_row,c_col;
{
    BWINDOW *pwin;
    BORDER  bord;
    WHERE   location;
    int     mode,columns,act_page;
    int     cursor_was_off,row,col,high,low;
    char    response[2];
    int     scan,bnum = 5;

              /* Create the window structure in memory.*/

    pwin = wncreate(10,           /* Height of data area */
                    26,           /* Width of data area */
             REVERSE);       /* Attributes of data area */
    if (pwin == NIL)
       return b_wnerr;              /* Quit if failure. */

    bord.type = 1;         /* Box drawn with single lines */
    bord.attr = RED;                    /* Red on black */

    location.dev    = scmode(&mode,&columns,&act_page);
    location.page   = act_page;
    location.corner.row =  c_row;
    location.corner.col = c_col;

           /* Retrieve former cursor position and size*/

    scpage(act_page);
    cursor_was_off = sccurst(&row,&col,&high,&low);

                           /* Actually display window */

    if (NIL == wndsplay(pwin,&location,&bord))
       return b_wnerr;             /* Quit if failure.*/
                  /* Display a message on the window.*/
    wnwrap(0,"\tSET THRESHOLD MENU\12\12\rENTER\12\12\r ",
           -1,-1,CHARS_ONLY);
```

```
    wnwrap(0,"A FOR APPROX   0 PPM/SEC\12\r",
           -1,-1,CHARS_ONLY);
    wnwrap(0,"B FOR APPROX   1 PPM/SEC\12\r",
           -1,-1,CHARS_ONLY);
    wnwrap(0,"C FOR APPROX   5 PPM/SEC\12\r",
           -1,-1,CHARS_ONLY);
    wnwrap(0,"D FOR APPROX  10 PPM/SEC\12\r",
           -1,-1,CHARS_ONLY);
    wnwrap(0,"E FOR APPROX 100 PPM/SEC\12\r",
           -1,-1,CHARS_ONLY);
    if (b_wnerr)
       return b_wnerr;              /* Quit if failure. */

                    /* Await a response from the user, */
                    /* echoing keystrokes to window    */
    wnquery(response,sizeof(response),&scan);
    if (b_wnerr)
       return b_wnerr;               /* Quit if failure.*/

                    /* Remove the window & restore the */
                    /* screen and cursor               */

    wnremove(pwin);
    sccurset(row,col);
    scpgcur(cursor_was_off,high,low,CUR_NO_ADJUST);

    if (b_wnerr)
       return b_wnerr;               /* Quit if failure*/

     wndstroy(pwin);
     return *response;
}
```

```
/*********************************************************/
/*   COMLOG:LOGGING PROGRAM FOR ANALINK                  */
/*********************************************************/

/*********************************************************/
/*   INCLUDE FILES                                       */
/*********************************************************/

#include <stdio.h>
#include <bfile.h>
#include <bwindow.h>
#include <bscreen.h>
#include <butility.h>

/*********************************************************/
/*   GLOBAL VARIABLES                                    */
/*********************************************************/

extern char *file_name,*response1;
extern FILE  *fpoint,*fopen();

/*********************************************************/
/*   PROCEDURES                                          */
/*********************************************************/

/*****************************************************/
/*   PROCEDURE:cr_file()                             */
/*   FUNCTION:To create or open a log file for appe*/
/*        nding to,after checking the file and path*/
/*        names and writing the general heading    */
/*        and column headings to the file          */
/*   GLOBAL VARS:fpoint                              */
/*   CALLS:log_wind()                                */
/*   PARAMETERS:none                                 */
/*   RETURNS:none                                    */
/*****************************************************/

cr_file()
{
    int    i,ercode,*plast;

                                        /* display menu */
     ercode = log_wind(5,6);            /* get response */
     ercode = flnorm(response1,file_name,plast);
                                        /* check it        */
    if (ercode != 0)                    /* if no error     */
       fpoint = fopen(file_name,"a");/* open file and   */
                          /* get handle & write headings*/
    fprintf(fpoint,"LOG FILE FOR LINK ANALYSIS\n\n");
    fprintf(fpoint,"SECS IN RUN--THRESHOLD T--SEC ABOVE T--
ER FREE SEC2");
    fprintf(fpoint,"--PPM ERR NOW--TOTAL ERROR--PPM
 AVERAGE\n\n");
}
```

```
/************************************************/
/*   PROCEDURE:log_wind()                       */
/*   FUNCTION:To create and display the window and */
/*        menu for opening a logging file and to  */
/*        store the response in a global string   */
/*        variable,response1,for later use,as well */
/*        as returning an error value if necessary */
/*   GLOBAL VARS:response1                      */
/*   CALLS:none                                 */
/*   PARAMETERS:c_row,c_col                     */
/*   RETURNS:b_wnerr                            */
/************************************************/

int log_wind(c_row,c_col)

int c_row,c_col;
{
    BWINDOW *pwin;
    BORDER  bord;
    WHERE   location;
    int     mode,columns,act_page;
    int     cursor_was_off,row,col,high,low;
    char    response[67];
    int     scan,bnum = 5;

              /* Create the window structure in memory */

    pwin = wncreate(10,        /* Height of data area   */
                    26,        /* Width of data area    */
              REVERSE);        /* Attribs of data area  */
    if (pwin == NIL)
.      return b_wnerr;              /* Quit if failure. */

    bord.type = 1;       /* Box drawn with single lines */
    bord.attr = RED;     /* Red on black                */

                       /* define location current page  */
    location.dev    = scmode(&mode,&columns,&act_page);
    location.page   = act_page;
    location.corner.row = c_row;/* define row left top  */
    location.corner.col = c_col;/* define col left top  */

             /* Retrieve former cursor position & size  */
    scpage(act_page);
    cursor_was_off = sccurst(&row,&col,&high,&low);

                            /* Actually display window  */
    if (NIL == wndsplay(pwin,&location,&bord))
       return b_wnerr;             /* Quit if failure   */
```

```
                    /* Display a message on the window  */
    wnwrap(0,"\tENTER LOG FILENAME WITH EXTENSION:\12\12\r ",
           -1,-1,CHARS_ONLY);
    wnwrap(0,"\(SAMPLE TIME MUST BE > 2 SEC TO LOG)\12\12\r",
           -1,-1,CHARS_ONLY);

     if (b_wnerr)
        return b_wnerr;             /* Quit if failure   */

                    /* Await a response from the user,  */
                    /* echoing keystrokes to window     */
     wnquery(response,sizeof(response),&scan);
     if (b_wnerr)
        return b_wnerr;              /* Quit if failure.  */

                    /* Remove the window & restore the  */
                    /* screen and cursor.               */
     wnremove(pwin);
     sccurset(row,col);
     scpgcur(cursor_was_off,high,low,CUR_NO_ADJUST);

     if (b_wnerr)
        return b_wnerr;              /* Quit if failure.  */

      wndstroy(pwin);
      return b_wnerr;                /* quit if failure.  */
      response1 = response;          /* store user reply  */
}
```

```
;*******************************************************;
;    COMINT:INTERRUPT SERVICE ROUTINE FOR DEVICE        ;
;*******************************************************;

;*********************************;
;            EQUATES              ;
;*********************************;

CNTRL1        EQU    31FH                ;address control reg 8254
CNTER1        EQU    31DH                ;add counter1 8254
CNT0_MODE     EQU    30H                 ;mode for int counter
RDBACK        EQU    0D4H                ;readback command 8254
MAX_SAMPLE    EQU    63H                 ;no. of ints per sec
INTA1         EQU    21H                 ;address of 8259 int reg
INTA0         EQU    20H                 ;address of 8259 int control
EOI           EQU    20H                 ;end of interrupt command
INT_CNT_LO    EQU    0F5H                ;count for 10 mSec int lsb
INT_CNT_HI    EQU    22H                 ;count msb
FULL_COUNT    EQU    0FFFFH              ;top count of 8254
INT_LEN       EQU    2

;*********************************;
;            VECTORS              ;
;*********************************;

VECTOR        SEGMENT AT 0
              ORG        0BH*4            ;INT TYPE 3 AVAILABLE ON 8259;
OFFTIMER      DW         ?                ;vector to int routine offset
SEGTIMER      DW         ?                ;int routine segment
VECTOR        ENDS

;***********************************************;
;            DATA STRUCTURE STORAGE             ;
;***********************************************;

DGROUP        GROUP    _DATA
              ASSUME  CS:_TEXT, DS:DGROUP, ES:VECTOR ,SS:DGROUP

EXTRN         _erray:NEAR                  ;array of errors in 'c' prog

_DATA         SEGMENT WORD PUBLIC 'DATA'

LAST_RD       DW         FULL_COUNT        ;last 8254 count read
PNTERR        DW         0000              ;pointer to erray
OLD_OFF       DW         0000              ;old vector at int3
OLD_SEG       DW         0000              ;old vector segment

_DATA         ENDS

_STACK        SEGMENT PARA STACK 'STACK'
              DW         200 DUP(?)
STCK_TP       DW         00
_STACK        ENDS
```

```
;*********************************************;
;            INITIALISATION OF VECTORS        ;
;*********************************************;

_TEXT       SEGMENT BYTE PUBLIC 'CODE'

;*****************************************************;
;    ROUTINE:_init                                    ;
;    FUNCTION:To save the old interrupt enviroment    ;
;         and to set up a new one to allow IRQ3       ;
;         to be generated by the board to call the    ;
;         TICK_RD routine.                            ;
;    GLOBAL VARS:none                                 ;
;    CALLS:none                                       ;
;    PARAMETERS:none                                  ;
;    RETURNS:none                                     ;
;    DESTROYS:AX                                      ;
;*****************************************************;

            PUBLIC _init                    ;to be called by 'c' prog

_init       PROC   NEAR                     ;writes vectors to table
                                            ;and unmasks int3 on 8259
            PUSH   BP                       ;std entry from 'c' prog
            MOV    BP,SP
            MOV    AX,VECTOR                ;segment of vector table
            MOV    ES,AX                    ;set ES to vector

            MOV    AX,WORD PTR OFFTIMER     ;read old vect from table
            MOV    WORD PTR OLD_OFF,AX      ;store old vector
            MOV    AX,WORD PTR OFFTIMER+2
            MOV    WORD PTR OLD_SEG,AX

            MOV    AX,OFFSET TICK_RD        ;offset of int routine
            MOV    WORD PTR OFFTIMER,AX     ;write to vector table
            MOV    WORD PTR OFFTIMER+2,CS   ;write segment prefix

            IN     AL,INTA1                 ;read 8259 mask
            AND    AL,11110111B             ;set int3 bit
            OUT    INTA1,AL                 ;write new 8259 mask

            MOV    SP,BP                    ;std exit to 'c' prog
            POP    BP
            RET
_init       ENDP
```

```
;*******************************************************;
;    ROUTINE:_xit                                       ;
;    FUNCTION:To restore the old interrupt enviroment   ;
;             and to disable IRQ3                       ;
;    GLOBAL VARS:none                                   ;
;    CALLS:none                                         ;
;    PARAMETERS:none                                    ;
;    RETURNS:none                                       ;
;    DESTROYS:AX                                        ;
;*******************************************************;

            PUBLIC _xit                         ;resets enviro to what
                                                ;it was before 'c' prog
_xit        PROC   NEAR
            PUSH   BP
            MOV    BP,SP
            MOV    AX,VECTOR                    ;segment of vector table
            MOV    ES,AX                        ;set ES to vector

            MOV    AX,WORD PTR OLD_OFF          ;fetch old vector
            MOV    WORD PTR OFFTIMER,AX         ;write to vector table
            MOV    AX,WORD PTR OLD_SEG
            MOV    WORD PTR OFFTIMER+2,AX

            IN     AL,INTA1                      ;read 8259 mask
            OR     AL,08                         ;reset int3 bit
            OUT    INTA1,AL                      ;write new 8259 mask

            MOV    SP,BP
            POP    BP
            RET
_xit        ENDP


;*******************************************************;
;    ROUTINE:TICK_RD                                    ;
;    FUNCTION:This routine handles the interrupt        ;
;             generated by the timer on the             ;
;             device and reads the latest error         ;
;             count and updates the 'c' program's       ;
;             error array without changing it's         ;
;             enviroment.                               ;
;    GLOBAL VARS:_erray                                 ;
;    CALLS:none                                         ;
;    PARAMETERS:none                                    ;
;    RETURNS:none                                       ;
;    DESTROYS:none                                      ;
;*******************************************************;
```

```
TICK_RD     PROC    NEAR

            STI                             ;allow higher level ints

            PUSH    AX
            PUSH    BX
            PUSH    CX                      ;save enviroment i.e.
            PUSH    DX                      ;all registers affected
            PUSH    DS                      ;by int routine
            PUSH    ES
            PUSH    DI
            PUSH    BP

            MOV     AX,DGROUP               ;value of data segment
            MOV     DS,AX                   ;pertinent to routine

            MOV     AL,RDBACK               ;readback command for 8254
            MOV     DX,CNTRL1               ;address of control reg
            OUT     DX,AL                   ;write RB command to 8254

            DEC     DX                      ;change address to
            DEC     DX                      ;point to count1 reg

            IN      AL,DX                   ;read lsb of count
            MOV     CL,AL                   ;store it
            IN      AL,DX                   ;read msb of count
            MOV     CH,AL                   ;form 16 bit count CX

            MOV     AX,WORD PTR LAST_RD     ;LAST_RD IN
            CMP     CX,AX                   ;comp this read and last
            JBE     LESSER                  ;if read < last
            MOV     DX,FULL_COUNT           ;work out difference
            SUB     DX,CX                   ;between this read
            INC     DX                      ;and last and store
            ADD     DX,AX                   ; in DX
            JMP     SHORT STORE             ; goto store value

LESSER:     SUB     AX,CX                   ;ELSE subtract this
            MOV     DX,AX                   ;value from last

STORE:      MOV     WORD PTR LAST_RD,CX     ;this read becomes lastrd
            MOV     BX,WORD PTR PNTERR      ;find pointer to erray
            MOV     DI,OFFSET DGROUP:_erray ;start address of erray
            MOV     WORD PTR [BX+DI],DX     ;write new error count
            ADD     BX,INT_LEN              ;point to next erray elemnt
            CMP     BX,MAX_SAMPLE * INT_LEN ;point to end of erray?
            JBE     NO_FULL                 ;if so then
            MOV     BX,0                    ;point to begin of erray
NO_FULL:    MOV     WORD PTR PNTERR,BX      ;else write new pointer

            MOV     AL,EOI                  ;end of interrupt command
            OUT     INTA0,AL                ;write to 8259 OCW2
```

```
                POP     BP
                POP     DI
                POP     ES
                POP     DS                      ;restore enviroment for
                POP     DX                      ;'c' control program
                POP     CX                      ;with original values of
                POP     BX
                POP     AX                      ;all registers affected

                IRET                            ;return to 'c' program

TICK_RD         ENDP

_TEXT           ENDS

                END
```

```
;*****************************************************************;
;    COMCOMMS:INTERRUPT SERVICE ROUTINE FOR COMMUNICATIONS       ;
;*****************************************************************;

;*******************************;
;            EQUATES            ;
;*******************************;

RDSTAT        EQU    3FDH                ;readback status 8250
MOD_CNTRL     EQU    3FCH                ;modem control reg 8250
CNTRL2        EQU    3FBH                ;line control reg 8250
INT_REG       EQU    3F9H                ;INTRPT reg 8250
DATAR1        EQU    3F8H                ;bidir data reg 8250
MODE_DIV      EQU    80H                 ;mode for div counter
COM_MODE      EQU    0FH                 ;mode of coms on 8250
INTA1         EQU    21H                 ;address of 8259 int reg
INTA0         EQU    20H                 ;address of 8259 int control
EOI           EQU    20H                 ;end of interrupt command
DIV_CNT_LO    EQU    30H                 ;count for comms baud rate
DIV_CNT_HI    EQU    00H                 ;count msb
MESS_LEN      EQU    02H                 ;length of comms message
TIMEOUT       EQU    0FFFFH              ;count for comms timeout
COMSET        EQU    0F0H                ;flag for comms command

ACNT_LO       EQU    0FEH                ; count for 1 injected
ACNT_HI       EQU    0FFH                ; error per 64kbits
BCNT_LO       EQU    0FFH                ; count for 2 injected
BCNT_HI       EQU    7FH                 ; errors per 64kbits
CCNT_LO       EQU    33H                 ; count for 5 injected
CCNT_HI       EQU    33H                 ; errors per 64 kbits
DCNT_LO       EQU    9AH                 ; count for 10 injected
DCNT_HI       EQU    19H                 ; errors per 64kbits
ECNT_LO       EQU    8FH                 ; count for 100injected
ECNT_HI       EQU    02H                 ; errors per 64 kbits

CNT2_MODE     EQU    0B4H                ; no.2 m&lsb mode2,bin

RESET         EQU    0FFH                ; reset byte format
REC_BITS      EQU    0FCH                ; receiver reset mask
TRX_BIT       EQU    0FBH                ; transmitter reset mask
INJ_BIT       EQU    0EFH                ; inject err reset mask

CNTRL1        EQU    31FH                ; control latch address
CNTER2        EQU    31EH                ; counter2 latch address
RST_LCH       EQU    31BH                ; reset latch address
```

```
;*******************************;
;            VECTORS            ;
;*******************************;

VECTOR      SEGMENT AT 0
            ORG       0CH*4              ;INT TYPE 4 AVAILABLE ON 8259;
OFFTIMER    DW        ?                  ;vector to int routine offset
SEGTIMER    DW        ?                  ;int routine segment

VECTOR      ENDS


;***********************************************;
;            DATA STRUCTURE STORAGE             ;
;***********************************************;

DGROUP      GROUP   _DATA
            ASSUME  CS:_TEXT, DS:DGROUP, ES:VECTOR,SS:DGROUP

EXTRN       _tx_total    :NEAR
EXTRN       _message     :NEAR
EXTRN       _pointmessage:NEAR
EXTRN       _start       :NEAR
EXTRN       _old_off2    :NEAR
EXTRN       _old_seg2    :NEAR
EXTRN       _mask        :NEAR


_DATA       SEGMENT WORD PUBLIC 'DATA'

_DATA       ENDS

_STACK      SEGMENT PARA STACK 'STACK'
            DW   200 DUP(0)
_STACK      ENDS

;***********************************************;
;            ROUTINES                           ;
;***********************************************;

_TEXT       SEGMENT BYTE PUBLIC 'CODE'
```

```
;*****************************************************;
;     ROUTINE:_init2                                  ;
;     FUNCTION:To save the old interrupt enviroment   ;
;          and to set up a new one to allow the       ;
;          asynchronous communications card to use    ;
;          IRQ4 to signal data received.The asynch    ;
;          card is also enabled by this code          ;
;     GLOBAL VARS:old_off2,old_seg2                   ;
;     CALLS:none                                      ;
;     PARAMETERS:none                                 ;
;     RETURNS:none                                    ;
;     DESTROYS:AX,DX,ES                               ;
;*****************************************************;

              PUBLIC _init2                    ;to be called by 'c' prog

_init2        PROC   NEAR                      ;writes vectors to table
                                               ;and unmasks int3 on 8259
              PUSH   BP                        ;std entry from 'c' prog
              MOV    BP,SP

              MOV    DX,CNTRL2                 ;control reg 8250
              MOV    AL,MODE_DIV               ;access baud counter
              OUT    DX,AL
              DEC    DX
              DEC    DX                        ;point to counter msb

              MOV    AL,DIV_CNT_HI
              OUT    DX,AL                     ;msb of count
              DEC    DX                        ;point to counter lsb
              MOV    AL,DIV_CNT_LO
              OUT    DX,AL                     ;lsb of count
              MOV    DX,CNTRL2                 ;control reg
              MOV    AL,COM_MODE               ;even parity
              OUT    DX,AL                     ;and 2 stop bits

              MOV    AX,VECTOR                 ;segment of vector table
              MOV    ES,AX                     ;set ES to vector

              MOV    AX,WORD PTR OFFTIMER      ;read old vector from table
              MOV    WORD PTR _old_off2,AX     ;store old vector
              MOV    AX,WORD PTR OFFTIMER+2
              MOV    WORD PTR _old_seg2,AX

              CLI
              MOV    AX,OFFSET COMS_RD1          ;offset of int routine
              MOV    WORD PTR OFFTIMER,AX        ;write to vector table
              MOV    WORD PTR OFFTIMER+2,CS      ;write segment prefix
              STI

              MOV    DX,MOD_CNTRL                ;modem cntrl reg
              MOV    AL,08                       ;enable OUT2
              OUT    DX,AL                       ;enable asynch intrpts
```

```
            MOV     DX,INT_REG              ;intrpt mask reg
            MOV     AL,1
            OUT     DX,AL                   ;enable data rdy intrpt

            IN      AL,INTA1                ;read 8259 mask
            AND     AL,11101111B            ;set int3 bit
            JMP     $+2                     ;delay
            OUT     INTA1,AL                ;write new 8259 mask

            MOV     SP,BP                   ;std exit to 'c' prog
            POP     BP
            RET
_init2      ENDP


;*******************************************************;
;     ROUTINE:_xit2                                     ;
;     FUNCTION:To restore the old interrupt enviroment  ;
;          and to disablethe asynchronous communi-      ;
;          cations card from using IRQ4 to signal       ;
;          data received.                               ;
;     GLOBAL VARS:old_off2,old_seg2                     ;
;     CALLS:none                                        ;
;     PARAMETERS:none                                   ;
;     RETURNS:none                                      ;
;     DESTROYS:AX,DX,ES                                 ;
;*******************************************************;

            PUBLIC _xit2                        ;resets enviro to what
                                                ;it was before 'c' prog
_xit2       PROC   NEAR

            PUSH   BP
            MOV    BP,SP
            MOV    AX,VECTOR                ;segment of vector table
            MOV    ES,AX                    ;set ES to vector

            CLI
            MOV    AX,WORD PTR _old_off2    ;fetch old vector
            MOV    WORD PTR OFFTIMER,AX     ;write to vector table
            MOV    AX,WORD PTR _old_seg2
            MOV    WORD PTR OFFTIMER+2,AX
            STI

            IN     AL,INTA1                 ;read 8259 mask
            OR     AL,00010000B             ;reset int3 bit
            JMP    $+2
            OUT    INTA1,AL                 ;write new 8259 mask

            MOV    DX,INT_REG
            MOV    AL,0
            OUT    DX,AL                    ;reset intrpt 8250 mask
```

```
            MOV     DX,3FCH
            OUT     DX,AL                   ;disable asynch intrpts

            MOV     SP,BP
            POP     BP
            RET

_xit2       ENDP


;*******************************************************;
;    ROUTINE:_trx_buf                                   ;
;    FUNCTION:To transmit a buffer of words passed as   ;
;         parameters from a 'c' routine across the      ;
;         link as an RS232 block message checking       ;
;         for transmitter ready and quitting if         ;
;         timeout                                       ;
;    GLOBAL VARS:none                                   ;
;    CALLS:none                                         ;
;    PARAMETERS:2 words on stack.[SP+4],[SP+6]          ;
;    RETURNS:error in AX                                ;
;    DESTROYS:AX,BX,DX                                  ;
;*******************************************************;

            PUBLIC    _trx_buf

_trx_buf    PROC NEAR

            PUSH    BP
            MOV     BP,SP                   ;std entry from 'c' prog
            SUB     SP,4
            PUSH    DI
            MOV     DI,4

T7:         MOV     BX,TIMEOUT              ;count for timeout
T3:         MOV     DX,RDSTAT               ;status register 8250
            DEC     BX
            JZ      T5                      ;jump if timeout
            IN      AL,DX
            AND     AL,20H                  ;check transmitter rdy
            JZ      T3                      ;if not ready

T5:         MOV     AX,WORD PTR [BP+DI]     ;fetch word parameter
            MOV     DX,DATAR1
            OUT     DX,AL                   ;transmit lsb

            MOV     BX,TIMEOUT              ;repeat for msb
T4:         MOV     DX,RDSTAT
            DEC     BX
            JZ      T6
            IN      AL,DX
            AND     AL,20H
            JZ      T4
```

```
T6:       MOV     DX,DATAR1
          MOV     AL,AH                          ;transfer msb-lsb
          OUT     DX,AL                          ;transmit msb

          ADD     DI,2                           ;end of block?
          CMP     DI,8
          JB      T7                             ;if not end

          POP     DI
          MOV     SP,BP                          ;std return to 'c' prog
          POP     BP
          RET
_trx_buf  ENDP


;*******************************************************;
;    ROUTINE:COMS_RD                                    ;
;    FUNCTION:This routine handles the interrupt        ;
;             generated by the 8250 on the              ;
;             coms card and reads the latest            ;
;             data and updates the 'c' program's        ;
;             enviroment and data automatically         ;
;             while not disturbing program flow.        ;
;    GLOBAL VARS:_mask,_tx_total,_start,_message        ;
;                _pointmessage.                         ;
;    CALLS:none                                         ;
;    PARAMETERS:none                                    ;
;    RETURNS:none                                       ;
;    DESTROYS:none                                      ;
;*******************************************************;

COMS_RD1     PROC    NEAR

             STI                                    ;allow higher level ints

             PUSH    BP
             PUSH    AX
             PUSH    BX
             PUSH    CX
             PUSH    DX                             ;save all regs affected
             PUSH    DI
             PUSH    SI
             PUSH    ES
             PUSH    DS

             MOV     AX,DGROUP                      ;value of data segment
             MOV     DS,AX                          ;pertinent to routine

             MOV     AL,EOI                         ;end of interrupt command
             OUT     INTA0,AL                       ;write to 8259 OCW2

             MOV     DX,DATAR1                      ;holding reg 8250
             IN      AL,DX
```

```
                CMP     AL,1BH                  ;<esc> start of message
                JE      ESC1                    ;if start
                CMP     BYTE PTR _start,1       ;if not started already
                JNE     ENDD                    ;then abort
                JMP     STORE                   ;else must be data

ESC1:           CMP     BYTE PTR _start,1       ;get start flag
                JE      ESC2                    ;if already started
                MOV     BYTE PTR _start,1       ;else set start flag
                JMP     ENDD                    ;and leave

STORE:          MOV     BX,OFFSET _message      ;start address of buffer
                MOV     DI,WORD PTR _pointmessage ;pointer
                MOV     BYTE PTR [BX+DI],AL     ;store latest byte
                INC     DI                      ;increment pointer
                MOV     WORD PTR _pointmessage,DI ;store it

ESC2:           CMP     WORD PTR _pointmessage,MESS_LEN  ;end of message?
                JA      PRT                     ;goto writing routine

ENDD:           POP     DS
                POP     ES
                POP     SI
                POP     DI                      ;restore environment for
                POP     DX                      ;'c' control program
                POP     CX
                POP     BX                      ;with original values of
                POP     AX                      ;all registers affected
                POP     BP

                IRET                            ;return to 'c' program

PRT:            MOV     BYTE PTR _start,0       ;reset start flag
                MOV     WORD PTR _pointmessage,0  ;reset pointer
                MOV     BX,OFFSET _message      ;start of buffer
                MOV     SI,1

                MOV     AL,BYTE PTR [BX]        ;first byte received
                CMP     AL,COMSET               ;flag for command/data
                JE      PRT2                    ;if command

                MOV     BP,OFFSET _tx_total     ;address global tx_total
                MOV     DI,0
PRT1:
                MOV     AL,BYTE PTR [BX+SI]     ;byte of integer value
                MOV     BYTE PTR [BP+DI],AL     ;store it
                INC     SI
                INC     DI                      ;point to next byte
                CMP     DI,2                    ;double byte integer
                JNZ     PRT1
                JMP     ENDD                    ;exit
```

```
PRT2:   MOV     DX,CNTRL1               ;address of 8254 reg
        MOV     AL,CNT2_MODE            ;counter mode for inject
        OUT     DX,AL                   ;set up counter2

                                        ;look at 2nd byte of com
        CMP     BYTE PTR [BX+1],0       ;is it = 0?
        JNE     PRT3                    ;no, then test next value
        MOV     AL,BYTE PTR _mask
        MOV     DX,RST_LCH              ;dev reset latch
        AND     AL,BYTE PTR [BX+2]      ;set new mask value
        OUT     DX,AL                   ;output it
        MOV     BYTE PTR _mask,AL       ;save it
        JMP     ENDD

PRT3:   CMP     BYTE PTR [BX+1],0FFH    ;2nd byte = ff?
        JNE     PRT9                    ;no, test next value
        MOV     AL,BYTE PTR [BX+2]      ;get reset bit value
        NOT     AL
        MOV     DX,RST_LCH              ;dev reset latch
        OR      AL,BYTE PTR _mask       ;reset bit of mask
        OUT     DX,AL
        MOV     BYTE PTR _mask,AL       ;save new mask
        JMP     ENDD

PRT9:   CMP     BYTE PTR [BX+1],'a'     ;compare 2nd byte for
        JE      PRT4                    ;counter value and
        CMP     BYTE PTR [BX+1],'b'     ;goto appropriate
        JE      PRT5                    ;routine
        CMP     BYTE PTR [BX+1],'c'
        JE      PRT6
        CMP     BYTE PTR [BX+1],'d'
        JE      PRT7
        CMP     BYTE PTR [BX+1],'e'
        JE      PRT8
        JMP     ENDD

PRT4:   MOV     DX,CNTER2               ;8254 counter2 address
        MOV     AL,ACNT_LO              ;lsb of count
        OUT     DX,AL
        MOV     AL,ACNT_HI              ;msb of count
        OUT     DX,AL
        MOV     AL,BYTE PTR _mask       ;set inj bit on mask
        MOV     DX,RST_LCH
        AND     AL,BYTE PTR [BX+2]
        OUT     DX,AL
        MOV     BYTE PTR _mask,AL
        JMP     ENDD
```

```
PRT5:       MOV     DX,CNTER2               ;8254 counter address
            MOV     AL,BCNT_LO              ;lsb of count
            OUT     DX,AL
            MOV     AL,BCNT_HI              ;msb of count
            OUT     DX,AL
            MOV     AL,BYTE PTR _mask       ;set inj bit on mask
            MOV     DX,RST_LCH
            AND     AL,BYTE PTR [BX+2]
            OUT     DX,AL
            MOV     BYTE PTR _mask,AL
            JMP     ENDD

PRT6:       MOV     DX,CNTER2                ;8254 counter address
            MOV     AL,CCNT_LO               ;lsb of count
            OUT     DX,AL
            MOV     AL,CCNT_HI               ;msb of count
            OUT     DX,AL
            MOV     AL,BYTE PTR _mask        ;set inj bit on mask
            MOV     DX,RST_LCH
            AND     AL,BYTE PTR [BX+2]
            OUT     DX,AL
            MOV     BYTE PTR _mask,AL
            JMP     ENDD

PRT7:       MOV     DX,CNTER2               ;8254 counter address
            MOV     AL,DCNT_LO              ;lsb of count
            OUT     DX,AL
            MOV     AL,DCNT_HI              ;msb of count
            OUT     DX,AL
            MOV     AL,BYTE PTR _mask       ;set inj bit on mask
            MOV     DX,RST_LCH
            AND     AL,BYTE PTR [BX+2]
            OUT     DX,AL
            MOV     BYTE PTR _mask,AL
            JMP     ENDD

PRT8:       MOV     DX,CNTER2               ;8254 counter address
            MOV     AL,ECNT_LO              ;lsb of count
            OUT     DX,AL
            MOV     AL,ECNT_HI              ;msb of count
            OUT     DX,AL
            MOV     AL,BYTE PTR _mask       ;set inj bit of mask
            MOV     DX,RST_LCH
            AND     AL,BYTE PTR [BX+2]
            OUT     DX,AL
            MOV     BYTE PTR _mask,AL
            JMP     ENDD

COMS_RD1    ENDP

_TEXT       ENDS

            END
```

APPENDIX H

THE LOGGED PROGRAM OUTPUT FILES

This appendix consists of six sections:

H.1. The log files for internal loopback error injection.

H.2. The calculations for internal loopback statistics.

H.3. The log files for external loopback error injection.

H.4. The calculations for external loopback statistics.

H.5. The log files for normal operation.

H.6. The explanation of normal operation.

The log files are as they were printed to disk by the program and headings are not given as the files take a complete page for eachfile.

H.1. INTERNAL LOOPBACK LOG FILES

LOG FILE FOR LINK ANALYSIS FOR 1 INJECTED ERROR PER 64 KBITS

RUN SECS	THRESHOLD-SEC	ABOVE-FREE SEC	PPM ERR NOW	TOTAL ERR	PPM AVERAG
5	0	0	14.9414	153	14.9414
10	0	0	15.1367	308	15.0391
15	0	0	15.0391	462	15.0391
20	0	0	14.7461	613	14.9658
25	0	0	14.8438	765	14.9414
30	0	0	15.1367	920	14.9740
35	0	0	14.9414	1073	14.9693
40	0	0	15.1367	1228	14.9902
45	0	0	15.1367	1383	15.0065
50	0	0	15.0391	1537	15.0098
55	0	0	15.1367	1692	15.0213
60	0	0	15.0391	1846	15.0228
65	0	0	14.8438	1998	15.0090
70	0	0	15.1367	2153	15.0181
75	0	0	15.1367	2308	15.0260
80	0	0	15.0391	2462	15.0269
85	0	0	15.1367	2617	15.0333
90	0	0	15.0391	2771	15.0336
95	0	0	15.0391	2925	15.0339
100	0	0	15.0391	3079	15.0342
105	0	0	14.9414	3232	15.0298
110	0	0	15.0391	3386	15.0302
115	0	0	14.9414	3539	15.0263
120	0	0	14.9414	3692	15.0228
125	0	0	15.1367	3847	15.0273
130	0	0	15.0391	4001	15.0278
135	0	0	15.0391	4155	15.0282
140	0	0	15.0391	4309	15.0286
145	0	0	15.1367	4464	15.0323
150	0	0	15.1367	4619	15.0358
155	0	0	15.1367	4774	15.0391
160	0	0	14.9414	4927	15.0360
165	0	0	15.0391	5081	15.0361
170	0	0	14.9414	5234	15.0333
175	0	0	15.0391	5388	15.0335
180	0	0	15.1367	5543	15.0364
185	0	0	14.8438	5695	15.0311
190	0	0	14.9414	5848	15.0288
195	0	0	15.0391	6002	15.0290
200	0	0	15.0391	6156	15.0293
205	0	0	15.1367	6311	15.0319
210	0	0	15.0391	6465	15.0321
215	0	0	14.9414	6618	15.0300
220	0	0	14.9414	6771	15.0280
225	0	0	15.0391	6925	15.0282
230	0	0	15.1367	7080	15.0306
235	0	0	15.1367	7235	15.0328
240	0	0	15.1367	7390	15.0350
245	0	0	15.0391	7544	15.0351
250	0	0	15.0391	7698	15.0352

LOG FILE FOR LINK ANALYSIS FOR 2 INJECTED ERRORS PER 64 KBITS

RUN SECS	THRESHOLD-SEC	ABOVE-FREE	SEC-PPM ERR NOW	TOTAL ERR	PPM AVERAG
5	0	0	0	30.0781	308	30.0781
10	0	0	0	30.0781	616	30.0781
15	0	0	0	30.0781	924	30.0781
20	0	0	0	29.7852	1229	30.0049
25	0	0	0	29.8828	1535	29.9805
30	0	0	0	30.1758	1844	30.0130
35	0	0	0	29.8828	2150	29.9944
40	0	0	0	30.1758	2459	30.0171
45	0	0	0	30.2734	2769	30.0456
50	0	0	0	29.8828	3075	30.0293
55	0	0	0	30.2734	3385	30.0515
60	0	0	0	29.8828	3691	30.0374
65	0	0	0	29.8828	3997	30.0255
70	0	0	0	30.0781	4305	30.0293
75	0	0	0	30.0781	4613	30.0326
80	0	0	0	29.9805	4920	30.0293
85	0	0	0	30.1758	5229	30.0379
90	0	0	0	29.8828	5535	30.0293
95	0	0	0	30.0781	5843	30.0319
100	0	0	0	30.0781	6151	30.0342
105	0	0	0	30.0781	6459	30.0363
110	0	0	0	29.9805	6766	30.0337
115	0	0	0	30.0781	7074	30.0357
120	0	0	0	29.9805	7381	30.0334
125	0	0	0	30.0781	7689	30.0352
130	0	0	0	29.9805	7996	30.0331
135	0	0	0	30.0781	8304	30.0347
140	0	0	0	30.0781	8612	30.0363
145	0	0	0	29.8828	8918	30.0310
150	0	0	0	30.0781	9226	30.0326
155	0	0	0	30.0781	9534	30.0340
160	0	0	0	30.0781	9842	30.0354
165	0	0	0	30.1758	10151	30.0397
170	0	0	0	29.9805	10458	30.0379
175	0	0	0	29.8828	10764	30.0335
180	0	0	0	30.2734	11074	30.0401
185	0	0	0	29.8828	11380	30.0359
190	0	0	0	29.9805	11687	30.0344
195	0	0	0	30.0781	11995	30.0356
200	0	0	0	29.8828	12301	30.0317
205	0	0	0	30.0781	12609	30.0329
210	0	0	0	30.1758	12918	30.0363
215	0	0	0	29.8828	13224	30.0327
220	0	0	0	29.9805	13531	30.0315
225	0	0	0	29.9805	13838	30.0304
230	0	0	0	30.0781	14146	30.0314
235	0	0	0	30.0781	14454	30.0324
240	0	0	0	30.1758	14763	30.0354
245	0	0	0	30.0781	15071	30.0363
250	0	0	0	30.0781	15379	30.0371

LOG FILE FOR LINK ANALYSIS FOR 5 INJECTED ERRORS PER 64 KBITS

RUN SECS	THRESHOLD-SEC	ABOVE-FREE	SEC-PPM ERR NOW	TOTAL ERR	PPM AVERAG
5	0	0	74.7070	765	74.7070
10	0	0	75.0977	1534	74.9023
15	0	0	74.9023	2301	74.9023
20	0	0	74.7070	3066	74.8535
25	0	0	74.9023	3833	74.8633
30	0	0	74.7070	4598	74.8372
35	0	0	74.7070	5363	74.8186
40	0	0	74.7070	6128	74.8047
45	0	0	75.2930	6899	74.8589
50	0	0	74.9023	7666	74.8633
55	0	0	75.2930	8437	74.9023
60	0	0	74.9023	9204	74.9023
65	0	0	74.7070	9969	74.8873
70	0	0	74.9023	10736	74.8884
75	0	0	74.9023	11503	74.8893
80	0	0	74.9023	12270	74.8901
85	0	0	74.7070	13035	74.8794
90	0	0	74.7070	13800	74.8698
95	0	0	74.9023	14567	74.8715
100	0	0	74.7070	15332	74.8633
105	0	0	75.0977	16101	74.8744
110	0	0	74.9023	16868	74.8757
115	0	0	75.0977	17637	74.8854
120	0	0	74.7070	18402	74.8779
125	0	0	74.7070	19167	74.8711
130	0	0	74.7070	19932	74.8648
135	0	0	74.9023	20699	74.8662
140	0	0	75.0977	21468	74.8744
145	0	0	74.9023	22235	74.8754
150	0	0	74.9023	23002	74.8763
155	0	0	74.9023	23769	74.8771
160	0	0	75.0977	24538	74.8840
165	0	0	74.9023	25305	74.8846
170	0	0	74.9023	26072	74.8851
175	0	0	74.9023	26839	74.8856
180	0	0	74.9023	27606	74.8861
185	0	0	74.7070	28371	74.8812
190	0	0	74.9023	29138	74.8818
195	0	0	74.7070	29903	74.8773
200	0	0	74.7070	30668	74.8730
205	0	0	74.7070	31433	74.8690
210	0	0	75.0977	32202	74.8744
215	0	0	74.9023	32969	74.8751
220	0	0	74.9023	33736	74.8757
225	0	0	75.0977	34505	74.8806
230	0	0	74.7070	35270	74.8769
235	0	0	74.9023	36037	74.8774
240	0	0	74.9023	36804	74.8779
245	0	0	74.9023	37571	74.8784
250	0	0	74.7070	38336	74.8750

H.2. INTERNAL LOOPBACK CALCULATIONS

The theoretical parts per million error rate for the above rates of error injection are calculated as follows:

1 error per 65536 at a data rate of 2048 Mbps gives

$$1 * \frac{2\ 048\ 000}{65536} * \frac{1\ 000\ 000}{2\ 048\ 000} = \frac{1\ 000\ 000}{65\ 536} = 15.2588 \text{ ppm}$$

thus for 2 injected errors per 65536 bits (independent of frequency)

$$2 * \frac{1\ 000\ 000}{65536} = 30.5176 \text{ ppm}$$

and for 5 injected errors per 65536 bits

$$5 * \frac{1\ 000\ 000}{65536} = 76.2939 \text{ ppm}$$

The measured ppm error statistic for 1 injected error per 65536 is

15.0352 ppm which is an error of

$$\frac{(15.2588 - 15.0352)}{15.2588} * \frac{100}{1} = 1.47\ \%$$

The measured ppm error statistic for 2 injected errors per 65536 is

30.0371 ppm which is an error of

$$\frac{(30.5176 - 30.0371)}{30.5176} * \frac{100}{1} = 1.57\ \%$$

The measured ppm error statistic for 5 injected errors per 65536 is

15.0352 ppm which is an error of

$$\frac{(76.2939 - 74.8750)}{76.2939} * \frac{100}{1} = 1.86\ \%$$

H.3. EXTERNAL LOOPBACK LOG FILES

LOG FILE FOR LINK ANALYSIS 1 err per 52248 at 894886.25 Hz

RUN SECS	THRESHOLD-SEC	ABOVE-FREE SEC	PPM ERR NOW	TOTAL ERR	PPM AVERAG
5	0	0	19.2383	197	19.2383
10	0	0	18.3594	385	18.7988
15	0	0	18.6523	576	18.7500
20	0	0	18.6523	767	18.7256
25	0	0	18.9453	961	18.7695
30	0	0	18.6523	1152	18.7500
35	0	0	18.9453	1346	18.7779
40	0	0	19.2383	1543	18.8354
45	0	0	18.9453	1737	18.8477
50	0	0	18.9453	1931	18.8574
55	0	0	18.6523	2122	18.8388
60	0	0	18.3594	2310	18.7988
65	0	0	19.5313	2510	18.8552
70	0	0	18.6523	2701	18.8407
75	0	0	18.9453	2895	18.8477
80	0	0	18.6523	3086	18.8354
85	0	0	18.9453	3280	18.8419
90	0	0	18.9453	3474	18.8477
95	0	0	18.3594	3662	18.8220
100	0	0	18.6523	3853	18.8135
105	0	0	19.2383	4050	18.8337
110	0	0	18.3594	4238	18.8121
115	0	0	18.6523	4429	18.8052
120	0	0	18.9453	4623	18.8110
125	0	0	18.9453	4817	18.8164
130	0	0	18.6523	5008	18.8101
135	0	0	19.5313	5208	18.8368
140	0	0	18.9453	5402	18.8407
145	0	0	18.3594	5590	18.8241
150	0	0	18.9453	5784	18.8281
155	0	0	18.6523	5975	18.8225
160	0	0	18.3594	6163	18.8080
165	0	0	18.6523	6354	18.8033
170	0	0	18.3594	6542	18.7902
175	0	0	18.6523	6733	18.7863
180	0	0	18.6523	6924	18.7826
185	0	0	18.9453	7118	18.7870
190	0	0	19.5313	7318	18.8065
195	0	0	19.5313	7518	18.8251
200	0	0	18.9453	7712	18.8281
205	0	0	18.3594	7900	18.8167
210	0	0	18.9453	8094	18.8198
215	0	0	18.9453	8288	18.8227
220	0	0	18.9453	8482	18.8255
225	0	0	18.6523	8673	18.8216
230	0	0	18.0664	8858	18.8052
235	0	0	18.6523	9049	18.8019
240	0	0	18.6523	9240	18.7988
245	0	0	19.2383	9437	18.8078
250	0	0	18.6523	9628	18.8047

LOG FILE FOR LINK ANALYSIS 2 errs per 52248 at 894886.25 Hz

RUN SECS	THRESHOLD-SEC	ABOVE-FREE	SEC-PPM ERR NOW	TOTAL ERR	PPM AVERAG
5	0	0	37.5977	385	37.5977
10	0	0	37.5977	770	37.5977
15	0	0	37.5977	1155	37.5977
20	0	0	37.5977	1540	37.5977
25	0	0	37.8906	1928	37.6563
30	0	0	37.8906	2316	37.6953
35	0	0	37.5977	2701	37.6814
40	0	0	37.8906	3089	37.7075
45	0	0	37.5977	3474	37.6953
50	0	0	37.5977	3859	37.6855
55	0	0	37.5977	4244	37.6776
60	0	0	37.5977	4629	37.6709
65	0	0	37.8906	5017	37.6878
70	0	0	37.5977	5402	37.6814
75	0	0	37.8906	5790	37.6953
80	0	0	37.5977	6175	37.6892
85	0	0	37.5977	6560	37.6838
90	0	0	37.5977	6945	37.6790
95	0	0	37.5977	7330	37.6748
100	0	0	37.5977	7715	37.6709
105	0	0	37.5977	8100	37.6674
110	0	0	37.5977	8485	37.6642
115	0	0	37.8906	8873	37.6741
120	0	0	37.5977	9258	37.6709
125	0	0	37.5977	9643	37.6680
130	0	0	37.5977	10028	37.6653
135	0	0	37.5977	10413	37.6628
140	0	0	37.5977	10798	37.6604
145	0	0	37.5977	11183	37.6583
150	0	0	37.8906	11571	37.6660
155	0	0	37.8906	11959	37.6733
160	0	0	37.5977	12344	37.6709
165	0	0	37.8906	12732	37.6776
170	0	0	37.5977	13117	37.6752
175	0	0	37.5977	13502	37.6730
180	0	0	37.5977	13887	37.6709
185	0	0	37.8906	14275	37.6768
190	0	0	37.5977	14660	37.6748
195	0	0	37.8906	15048	37.6803
200	0	0	37.8906	15436	37.6855
205	0	0	37.5977	15821	37.6834
210	0	0	37.5977	16206	37.6814
215	0	0	37.5977	16591	37.6794
220	0	0	37.8906	16979	37.6842
225	0	0	37.5977	17364	37.6823
230	0	0	37.5977	17749	37.6805
235	0	0	37.8906	18137	37.6849
240	0	0	37.8906	18525	37.6892
245	0	0	37.5977	18910	37.6873
250	0	0	37.5977	19295	37.6855

H.4. EXTERNAL LOOPBACK CALCULATIONS

The theoretical parts per million error rate for the above rates of error injection are calculated as follows:

1 error per 52248 at a data rate of 894886.25 Hz gives

$$1 * \frac{894\ 886.25}{52248} * \frac{1\ 000\ 000}{894\ 886.25} = \frac{1\ 000\ 000}{52248} = 19.1394 \text{ ppm}$$

thus for 2 injected errors per 52248 bits (independant of frequency)

$$2 * \frac{1\ 000\ 000}{52248} = 38.2790 \text{ ppm}$$

The measured ppm error statistic for 1 injected error per 52248 is

18.8047 ppm which is an error of

$$\frac{(19.1394 - 18.8047)}{19.1394} * \frac{100}{1} = 1.75\ \%$$

The measured ppm error statistic for 2 injected errors per 52248 is

37.6855 ppm which is an error of

$$\frac{(38.2790 - 37.6855)}{38.2790} * \frac{100}{1} = 1.55\ \%$$

H.5. EXTERNAL DUPLEX MODE LOG FILES

LOG FILE FOR LINK ANALYSIS FOR BIT STREAM INVERTED.

RUN SECS	THRESHOLD-SEC	ABOVE-FREE	SEC-PPM	ERR NOW	TOTAL ERR	PPM AVERAG
5	0	0	5	0.0000	0	0.0000
10	0	0	10	0.0000	0	0.0000
15	0	0	15	0.0000	0	0.0000
20	0	0	20	0.0000	0	0.0000
25	0	0	25	0.0000	0	0.0000
30	0	0	30	0.0000	0	0.0000
35	0	0	35	0.0000	0	0.0000
40	0	0	40	0.0000	0	0.0000
45	0	0	45	0.0000	0	0.0000
50	0	0	50	0.0000	0	0.0000
55	0	0	55	0.0000	0	0.0000
60	0	0	60	0.0000	0	0.0000
65	0	0	65	0.0000	0	0.0000
70	0	0	70	0.0000	0	0.0000
75	0	0	75	0.0000	0	0.0000
80	0	0	80	0.0000	0	0.0000
85	0	0	85	0.0000	0	0.0000
90	0	0	90	0.0000	0	0.0000
95	0	0	95	0.0000	0	0.0000
100	0	0	100	0.0000	0	0.0000
105	0	0	105	0.0000	0	0.0000
110	0	0	110	0.0000	0	0.0000
115	0	0	115	0.0000	0	0.0000
120	0	0	120	0.0000	0	0.0000
125	0	0	125	0.0000	0	0.0000
130	0	0	130	0.0000	0	0.0000
135	0	0	135	0.0000	0	0.0000
140	0	0	140	0.0000	0	0.0000
145	0	0	145	0.0000	0	0.0000
150	0	0	150	0.0000	0	0.0000
155	0	0	155	0.0000	0	0.0000
160	0	0	160	0.0000	0	0.0000
165	0	0	165	0.0000	0	0.0000
170	0	0	170	0.0000	0	0.0000
175	0	0	175	0.0000	0	0.0000
180	0	0	180	0.0000	0	0.0000
185	0	0	185	0.0000	0	0.0000
190	0	0	190	0.0000	0	0.0000
195	0	0	195	0.0000	0	0.0000
200	0	0	200	0.0000	0	0.0000
205	0	0	205	0.0000	0	0.0000
210	0	0	210	0.0000	0	0.0000
215	0	0	215	0.0000	0	0.0000
220	0	0	220	0.0000	0	0.0000
225	0	0	225	0.0000	0	0.0000
230	0	0	230	0.0000	0	0.0000
235	0	0	235	0.0000	0	0.0000
240	0	0	240	0.0000	0	0.0000
245	0	0	245	0.0000	0	0.0000
250	0	0	250	0.0000	0	0.0000

LOG FILE FOR LINK ANALYSIS FOR NORMAL DUPLEX MODE TESTING

RUN SECS	THRESHOLD-SEC	ABOVE-FREE SEC	PPM ERR NOW	TOTAL ERR	PPM AVERAG
5	1	1	4	0.2930	3	0.2930
10	1	4	6	0.6836	10	0.4883
15	1	6	8	0.6836	17	0.5534
20	1	7	10	0.4883	22	0.5371
25	1	9	12	0.6836	29	0.5664
30	1	10	16	0.2930	32	0.5208
35	1	11	20	0.4883	37	0.5162
40	1	11	25	0.0000	37	0.4517
45	1	15	26	0.9766	47	0.5100
50	1	16	29	0.3906	51	0.4980
55	1	17	33	0.2930	54	0.4794
60	1	19	35	0.6836	61	0.4964
65	1	22	36	0.9766	71	0.5334
70	1	25	38	0.9766	81	0.5650
75	1	27	40	0.5859	87	0.5664
80	1	27	45	0.0000	87	0.5310
85	1	28	48	0.3906	91	0.5227
90	1	30	51	0.4883	96	0.5208
95	1	34	51	1.0742	107	0.5500
100	1	35	55	0.3906	111	0.5420
105	1	37	56	0.6836	118	0.5487
110	1	40	58	0.9766	128	0.5682
115	1	40	63	0.0000	128	0.5435
120	1	43	65	0.7813	136	0.5534
125	1	45	67	0.6836	143	0.5586
130	1	48	69	0.7813	151	0.5672
135	1	52	69	1.2695	164	0.5932
140	1	55	71	0.8789	173	0.6034
145	1	56	75	0.2930	176	0.5927
150	1	58	76	0.7813	184	0.5990
155	1	60	79	0.5859	190	0.5985
160	1	62	81	0.9766	200	0.6104
165	1	64	84	0.4883	205	0.6067
170	1	67	85	0.8789	214	0.6147
175	1	68	89	0.3906	218	0.6083
180	1	71	90	0.8789	227	0.6158
185	1	72	94	0.2930	230	0.6071
190	1	73	98	0.1953	232	0.5962
195	1	75	100	0.4883	237	0.5934
200	1	76	102	0.4883	242	0.5908
205	1	76	106	0.0977	243	0.5788
210	1	77	108	0.3906	247	0.5743
215	1	78	112	0.2930	250	0.5678
220	1	79	114	0.6836	257	0.5704
225	1	80	117	0.3906	261	0.5664
230	1	81	120	0.2930	264	0.5605
235	1	82	123	0.2930	267	0.5548
240	1	84	126	0.4883	272	0.5534
245	1	85	129	0.3906	276	0.5501
250	1	87	131	0.6836	283	0.5527

H.6. FULL DUPLEX EXPLANATION

The file on page H-9 above (with the incoming data stream inverted) shows the inability of the test unit to measure streams where synchronisation is impossible to achieve. The measurement is thus 0.0000 ppm when strictly the theoretical measurement should be 1 000 000.0000 ppm.

The second instance is a normal duplex link with general noise and signal loss accounting for the errors measured. It should be noted that the threshold has been set to 1 ppm per second for this test to show the validity of the measurements. The measurement of seconds above the threshold, SECS ABOVE does not bear a relationship to the FREE SECONDS measurement which is a count of the number of seconds in the run during which no errors were counted.

APPENDIX I

CIRCUIT DIAGRAMS

The circuit diagrams were prepared using the PC-CAPS schematic capture utility of the P-CAD system and printed on an AST TURBO laser printer.

The printed circuit board outline (taken from the PCB layout file captured using PC-CARDS) is included to show evidence of the work done in providing a final working model of the error analyser board. Four such boards were manufactured from photoplots (using PC-PLOTS) of the double sided pcb track layout and are now in use.

The diagrams are presented in the following order:

1) TRANSMIT CLOCK AND PRBS GENERATOR
2) RECEIVER
3) BUFFERING AND DECODING
4) EXTERNAL INTERFACE
5) PCB LAYOUT

An explanation for some of the gate configurations is given on the intervening pages.

I.1. THE TRANSMITTER

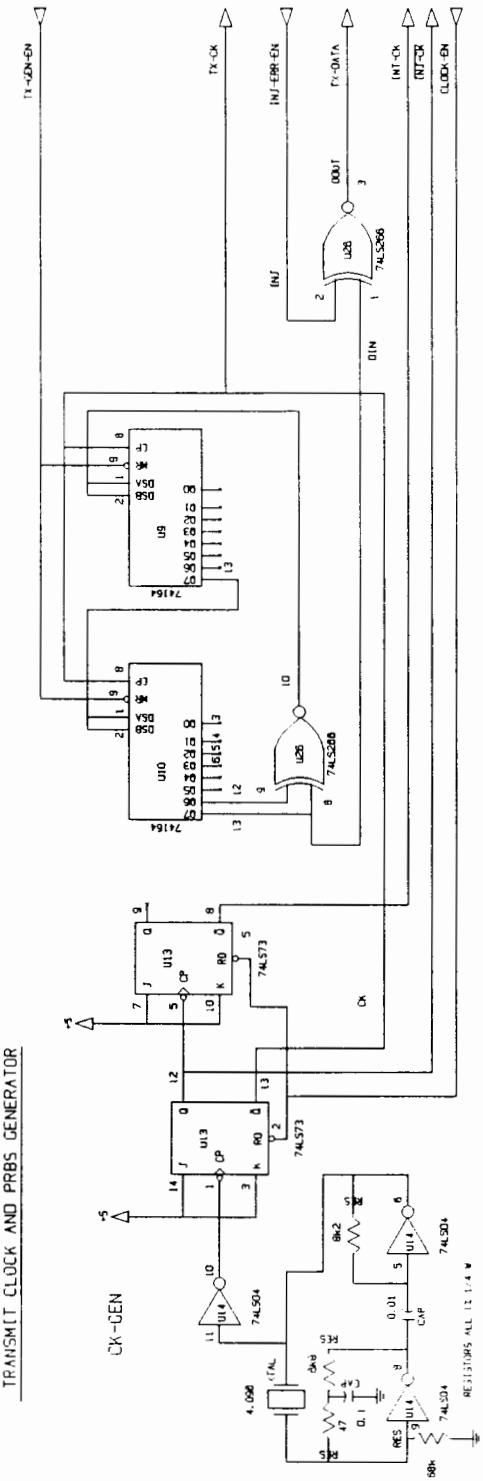


FIGURE I.1.

TRANSMIT CLOCK AND PRBS GENERATOR

I.2. THE RECEIVER

In the synchronisation circuitry the equation for the reset input of the reference PRBS generator is:

$$\overline{\text{RESET}} = Q14.Q13.Q12.Q11.Q10.Q9.Q8.Q7.Q6.Q5.Q4.Q3.Q2.Q1.Q0 + Q15 + \overline{\text{CLOCK}}$$

and the equation for the set pulse is:

$$\overline{\text{SET}} = Q15.Q14.Q13.Q12.Q11.Q10.Q9.Q8.Q7.Q6.Q5.Q4.Q3.Q2.Q1 + Q0 + \text{CLOCK}.$$

But since one wants to generate both pulses with common circuitry one extracts a common term

$$\overline{\text{COM}} = Q14.Q13.Q12.Q11.Q10.Q9.Q8.Q7.Q6.Q5.Q4.Q3.Q2.Q1$$

and thus reset changes to become

$$\overline{\text{RESET}} = \overline{\text{COM}} + \overline{Q0} + Q15 + \overline{\text{CLOCK}}$$

and the set pulse becomes

$$\overline{\text{SET}} = \overline{\text{COM}} + \overline{Q15} + Q0 + \text{CLOCK}$$

as can be seen in figure 2.

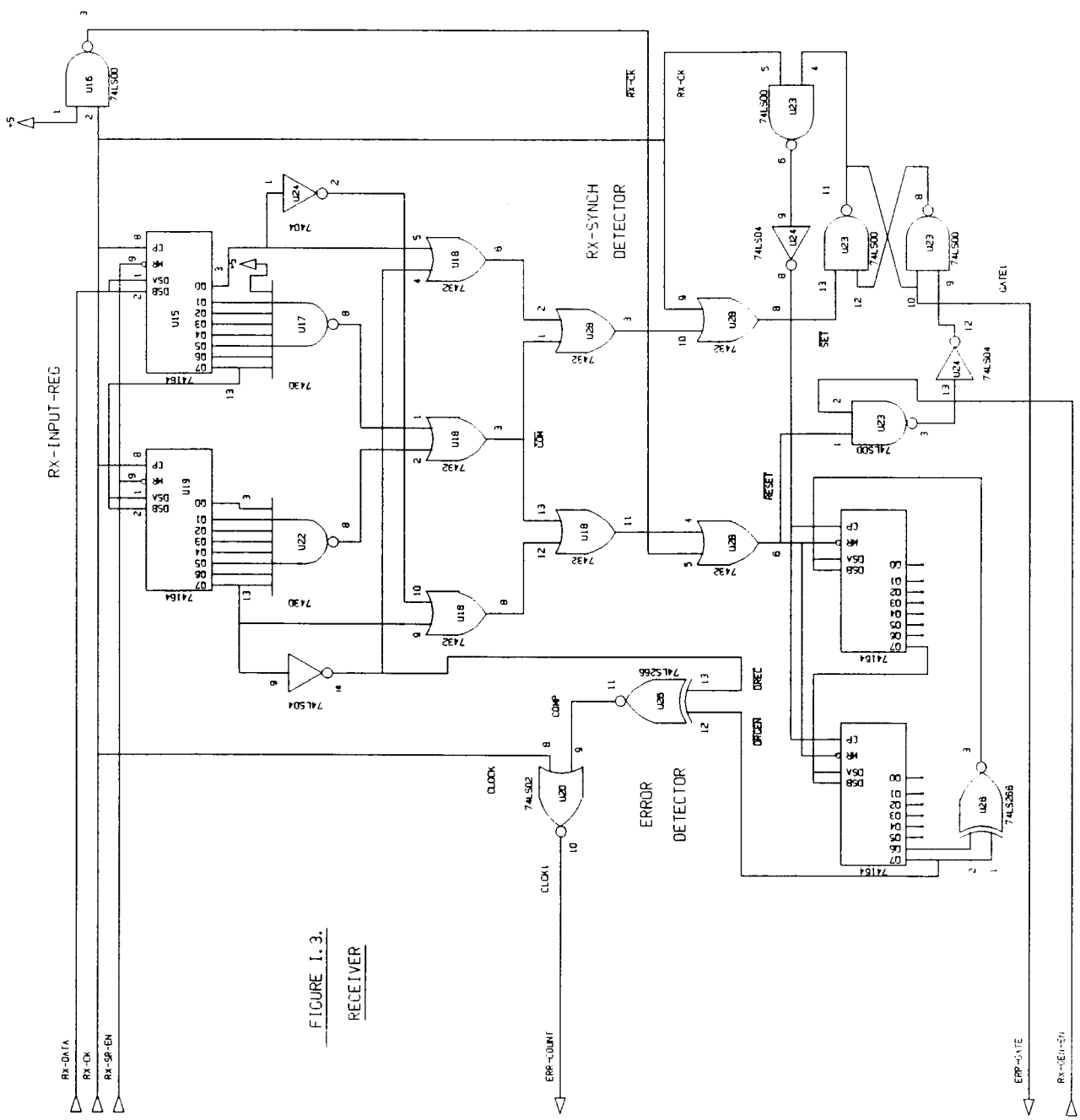


FIGURE I.3.

RECEIVER

I.3. BUFFERING AND DECODING

The general address decode output function is:

$$\overline{ADEC} = \overline{AEN}.A9.A8.\overline{A7}.\overline{A6}.\overline{A5}.A4.A3 + \overline{IOW}.\overline{IOR}$$

The first term is OR'ed with the (IOW AND IOR) term to ensure that only when the address generated is an I/O address is the board enabled.

Thus the 8254 chip select line,

$$\overline{8254CS} = \overline{ADEC} + A2$$

while the latch which is a 74ls373 has

$$\overline{373CS} = \overline{ADEC} + A2 + \overline{A1.A0}$$

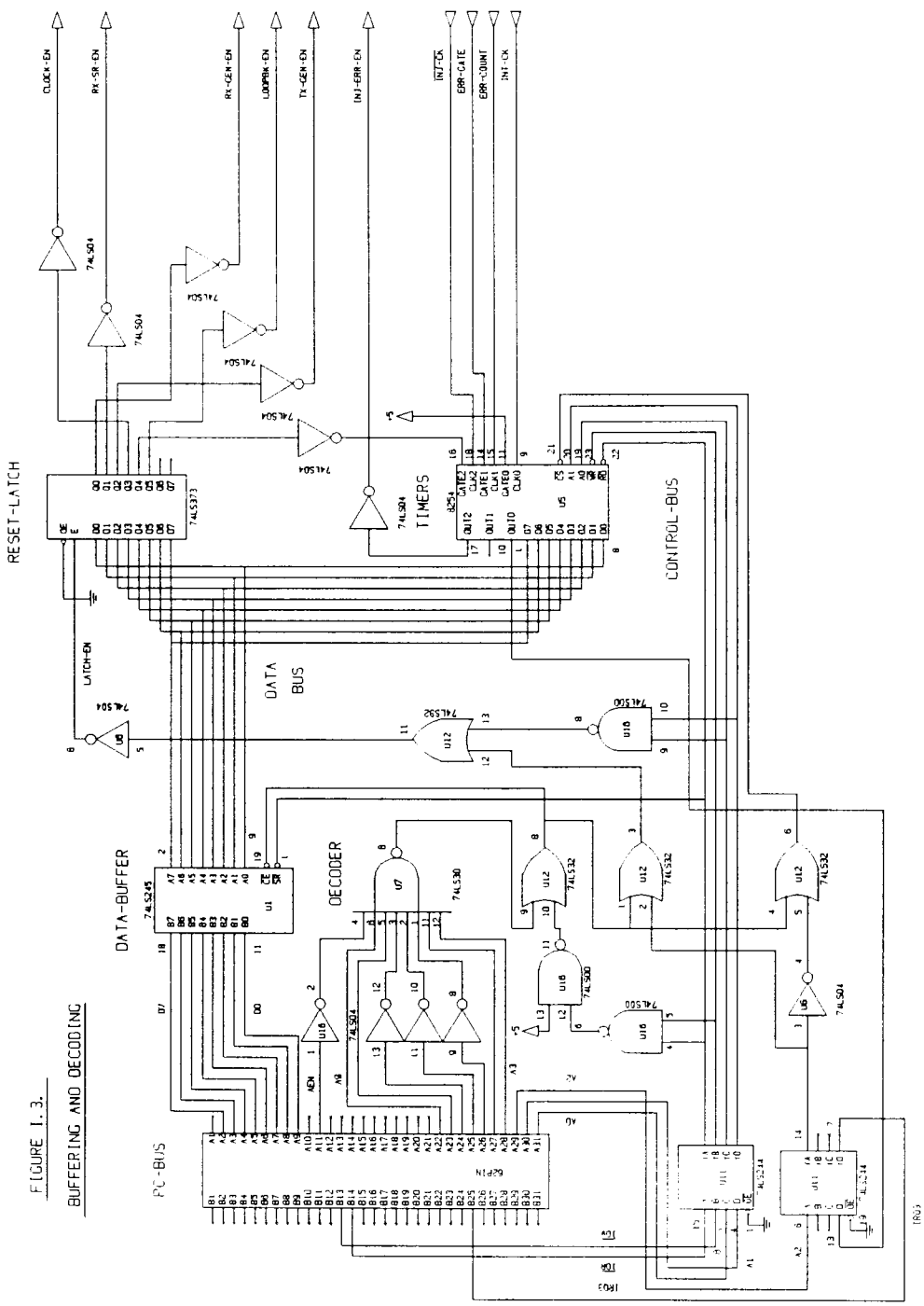


FIGURE I.3.

BUFFERING AND DECODING

I.4. THE EXTERNAL INTERFACE

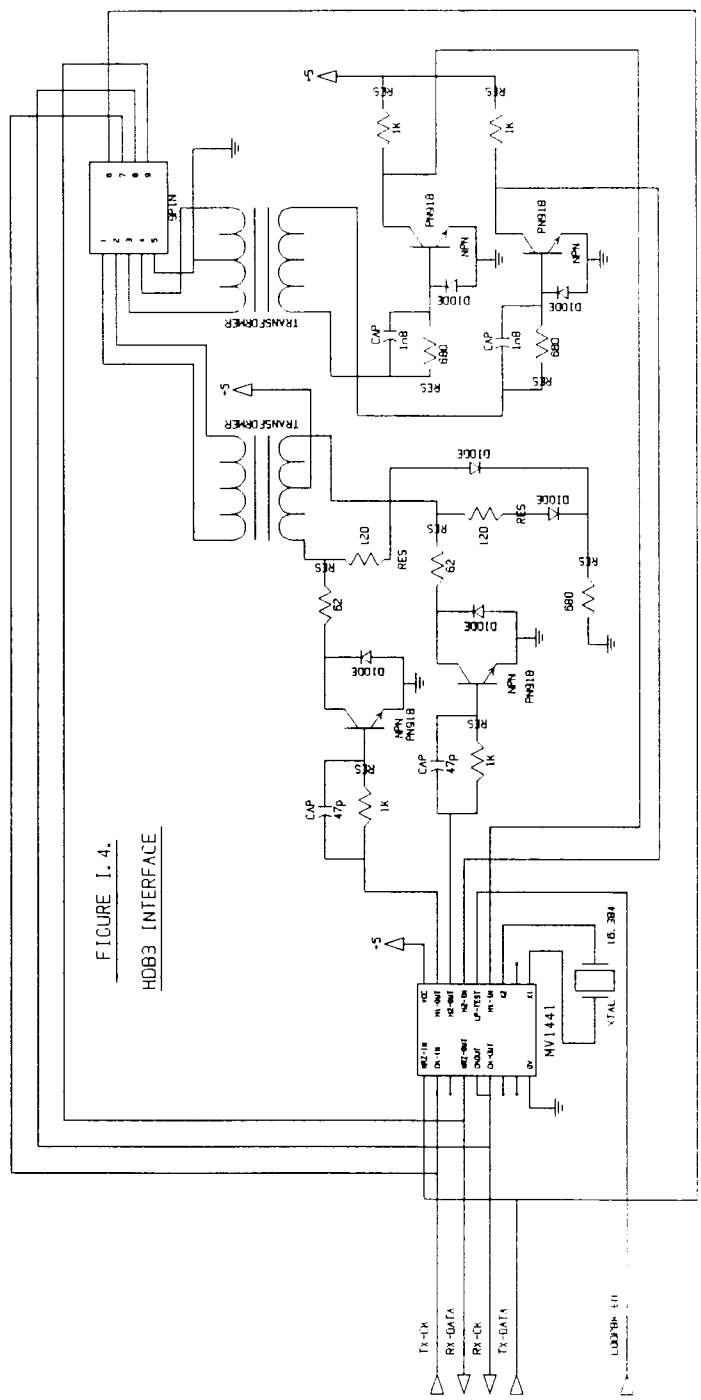
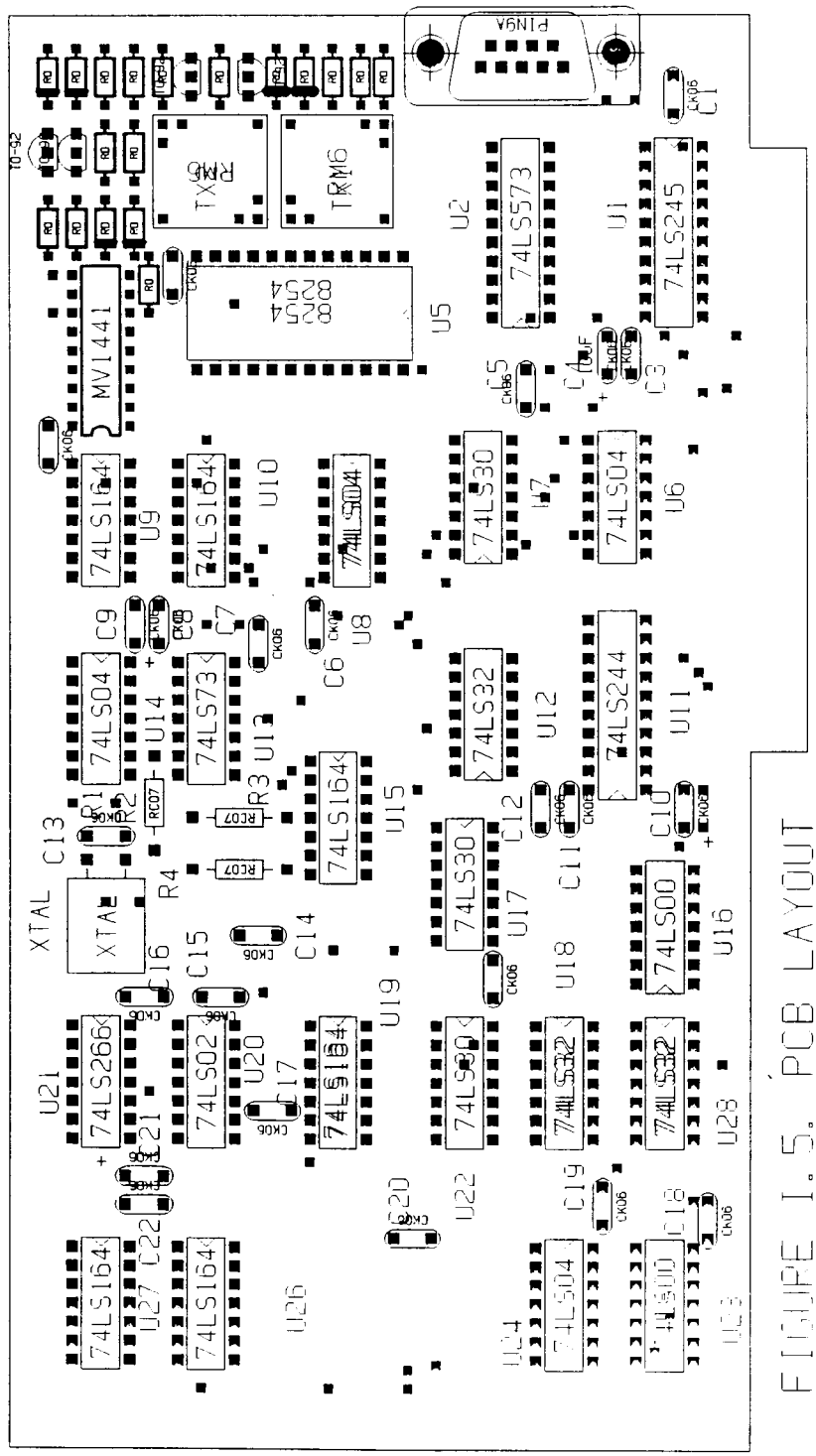


FIGURE I.4.
HDB3 INTERFACE

I.5. THE PCB LAYOUT



APPENDIX J

THE USE OF ANALINK

1 A BASIC OVERVIEW

The device is designed to analyse the error rate on a microwave or other data link by providing the hardware and software interface between the link and a personal computer (PC). The hardware of the device is designed to generate and receive a pseudorandom data stream at 2048 kbit/s (although this is easily alterable on the present board between DC and 8 Mbit/s).

There is furthermore an RS232 communications link between the two boards used on either end of the link and data and commands are transferred across the link while it is being tested.

The software is designed to allow easy control of all the possible options available on the device such as starting and stopping the transmit and receive sections, viewing the analysis in different modes and injecting errors to verify operation. The communications part of the software is transparent to the user and analysis of each channel of the

duplex link is displayed seperately on each terminal and independently on each terminal.

The software is also designed to allow the device to be modified to run at 8 MHz by including bi-stable variables selectable at run time to suit the crystal on board.

2 INTERFACE SPECIFICATIONS

2.1 PC INTERFACE

The board plugs into a 62 pin PC edge connector socket on a PC motherboard and presents one LS TTL load to the following lines:

D0 - D7 , A0 - A9 , IOR , IOW , IRQ3 .

In addition the board draws 385 mA from the +5 rail of the PC supply.

There must be an asynchronous communications card present in the system and it must be configured to be COM1 (addresses 3F8Hex to 3FFHex) and to use IRQ4.

2.2 EXTERNAL INTERFACE

There is a 9 pin D connector on the back panel of the PC with the following signals on it:

pin 1:	HDB3 TRANSMIT
pin 2:	HDB3 TRANSMIT
pin 3:	HDB3 RECEIVE
pin 4:	HDB3 RECEIVE
pin 5:	GND
pin 6:	2 Mbs TTL transmit data stream
pin 7:	2 MHz TTL 50% duty cycle transmit clock
pin 8:	2 MHz TTL 50% duty cycle receive clock
pin 9:	2 Mbs TTL receive data stream

table 1.1. 9 pin D connector pinout

2.3 SOFTWARE INTERFACE

The program requires an IBM PC or compatible with hardware interrupt vector table addresses compatible with standard IBM bios although this is easily modifiable to suit any change. No graphics card or graphics monitor is needed and the only enviroment configuration that is necessary is that the CONFIG.SYS file for the system must contain the specification 'DEVICE = ANSI.SYS' and a copy of the ANSI.SYS file must be in the root directory of the booted disk.

The supplied disk with the device controller software on it is also a bootable disk and has suitable AUTOEXEC.BAT,

ANSI.SYS and CONFIG.SYS files on it to automatically start the controller program if the system is booted on this drive.

3 PROGRAM USE

3.1 STARTING

The program is located in source code and executable form on the appropriate floppy disk. Type 'ANALINK' and the program will execute. If no board is plugged into the system the program will display a message to that effect and ask the user to end normally (the first options menu will appear and a 'z' is required to be entered to end).

The board test routine will respond with the same message if there is a malfunction in the reset latch, clock circuitry or counter.

Alternatively the system can be booted with the supplied diskette in drive A and the controlling program will automatically start.

As an extra the program will prompt for presence of a 16 MHz crystal to run the device at 8 MHz although the default is 2 MHz and no harm will be done if a 'Y' is entered by mistake

(the error statistics will be out although the total error count will still be valid).

It should be noted that the left half of the screen on each terminal is dedicated to the receiving channnel and all commands issued on that side of the screen will affect that channel only. The same applies to the right hand side of the screen and the transmitting channel. The left or right side of the screen is changed by the entry of 'x' or 'y' from the display options menu.

3.2 OPERATION OPTIONS MENU

This is the first menu to be displayed and from this menu transmitting and receiving can be switched on or off. An example of screen menu format is shown in fig 1.

3.2.1 Transmitting

A 't' is entered to start transmitting and an 'h' to stop. As the program automatically starts on the receiving side of the screen the transmitter that is started by this command is actually on the other end of the link. To start the transmitter on the same side of the link as the program the operation options menu for transmitting must be reached through the transmitting display options menu.

Transmitting can be done actively while in the program and receiving or transparently while using the computer for another task. This can be done by starting transmission and then ending the program without resetting the device. This can be useful for one way testing as it only uses one PC.

NB! To end transmission the program must be restarted and the program will automatically reset the device before offering options as to whether the user wishes to start any of the sections once more.

3.2.2 Receiving

An 'r' is entered to start receiving and a 'c' to stop. As soon as receiving is enabled on either side the display options menu will be displayed, therefore, if loopback testing or full duplex testing is needed the transmitter should be enabled before the receiver otherwise the user will have to end the 'display options' menu to once again access the 'basic options' menu.

3.3 DISPLAY OPTIONS MENU

This menu allows swapping between sides of the display, using continuous analysis with presettable sample time, threshold of errors, optional dumping of data to disk and

finally the option of injecting errors into the stream purposefully.

3.3.1 Swapping sides

To change sides one enters a 'y' or an 'x'in the appropriate display options menu and this is the only route from side to side thus one must always return to this menu.

3.3.2 Running Mode

The running mode display is obtained by entering an 'r' and provides a continuous analysis once per sample time of the last sample time of errors displaying parts per million errors over the last sample time , total number of errors since the running mode was selected and the average parts per million errors since the running mode was selected, amongst various other results as can be seen in figure 3.

Note that the reception of data from across the link is only possible when the opposite side of the link is in running mode and thus if the operator on the other side is changing the configuration on that side no new data will be received across the link.

It was decided not to facilitate controlling of display modes across the link because the communicated commands would then visibly interfere with an operator on the other side whereas the mask changing commands occur invisibly over the link.

The running mode display can be exited by entering any key on the keyboard and the display will return to the display options menu.

3.3.3 Setting Sample Time

By entering 's' the sample time menu is entered and a new sample time for the running display can be selected from the choice. An invalid choice results in no change to the sample time.

3.3.4 Setting Threshold

By entering 't' the threshold menu is entered and a new threshold for the running display can be selected from the choice. An invalid choice results in no change to the threshold.

3.3.5 Logging To Disk

If a 'd' is entered in the display options menu then the dump or log menu is entered and one is prompted for a filename. If an invalid path or file name is entered the menu will re-appear until a valid one is entered. The total path plus filename and extension can be up to 50 characters long. When one re-enters the display options menu the sample time should be set to greater than or equal to 2 seconds or the results of the running display will not be logged to disk.

If a log file of the same name as the one entered already exists the program appends the new data on the end of the old file with a new heading seperating the two lots of data.

To end logging a 'k' is entered which also closes the log file for safety in case of program abortion.

3.3.6 Error Injection

This option offered on the display options menu (selected by entering 'i' and exited with 'e') allows the user to deliberately inject errors into the outgoing data stream to monitor their progress on the receiver.

When selected, this option offers a further menu of choices of injected error percentage ('a' - 'e').

When the error injection is entered for the receive channel the results of the injection may take a while to appear on the running display. This is because the command must be processed and acted upon on the other side of the link. The injected errors may also take a while to affect the error results as these values are averages of a great deal of measurements.

When the error injection is entered from the transmit side the errors are injected almost immediately but they must still be collected and processed on the other side of the link before being transmitted back to be displayed thus introducing a delay.

To end error injection an 'e' is entered.

3.4 ENDING

To exit from any menu enter 'z' and the previous menu will be displayed. The menus have automatic resets on exit such as stopping reception on exit from the basic options menu, disabling the interrupt counter from the display options

menu but default display modes are remembered on re-entry to a menu.

On exit from the operation options menu the user will be prompted on whether or not total device reset is required. If the transmitter is not reset a message to that effect will be displayed on program exit.

3.5 GENERAL

The source code supplied was developed using MICROSOFT C and TURBO-C compilers although the final executable file was compiled and linked using MICROSOFT and C-TOOLS-PLUS libraries. Modification to the source code can be done using either compiler although when using the TURBO-C version the library routines 'inp()' and 'outp()' may have to be altered to 'inportb()' and 'outportb()'.

The diskette accompanying the report also contains the schematic of the circuit for the board, the printed circuit board layout and the manual documentation (created in MS-WORD). The files are respectively: DEV.SCH, DEV.PCB and DEV.DOC. The circuit files were both created in P-CAD.

APPENDIX K

SPECIFICATION SHEET FOR MV1441

The specification sheet for the Plessey Semiconductors 2 Mbps HDB3 codec chip is included as an appendix because it is a special function chip which is not mass produced and thus the specification sheet is not normally easily available.

The main body of the thesis refers to the appendix on two subjects:

1) The clock recovery from the balanced line HDB3 signals.

2) The recommended external line driving circuit with hand-wound balanced line transformers.

PLESSEY
Semiconductors

ADVANCE INFORMATION

Advance information is issued to advise Customers of new additions to the Plessey Semiconductors range which, nevertheless, still have *pre-production* status. Details given may therefore change without notice although we would expect this performance data to be representative of *full production* status product in most cases. Please contact your local Plessey Semiconductors Sales Office for details of current status.

2MBIT PCM SIGNALLING CIRCUIT

MV1441

HDB3 ENCODER/DECODER/CLOCK REGENERATOR

The 2.048MBit PCM Signalling Circuits comprise a group of circuits which will perform the common signalling and error detection functions for a 2.048MBit PCM transmission link operating to the appropriate CCITT recommendations. The circuits are fabricated in CMOS and operate from a single 5 volt supply with relevant inputs and outputs TTL compatible.

The MV1441 is an encoder/decoder for the pseudo-ternary transmission code, HDB3 (CCITT Orange Book Vol.III.2 Annex to Rec. G703). The device encodes and decodes simultaneously and asynchronously. Error monitoring functions are provided to detect violations of HDB3 coding, all ones detection and loss of input (all zeros detection). In addition a loop back function is provided for terminal testing. A clock recovery circuit is provided using a 16.384MHz crystal (12.352MHz for 1.544MHz operation), which may be shared between several separate devices.

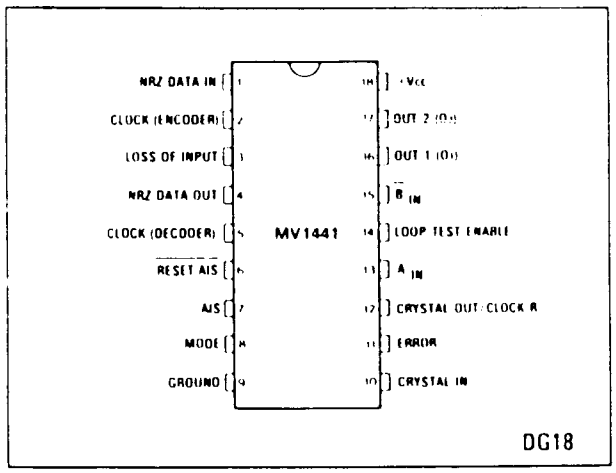


Fig.1 Pin connections - top view

FEATURES

- On-Chip Digital Clock Regenerator
- HDB3 Encoding and Decoding to CCITT rec. G703
- Asynchronous Operation
- Simultaneous Encoding and Decoding
- Clock Recovery Signal allows Off-Chip Clock Regeneration
- Loop Back Control
- HDB3 Error Monitor
- 'All Ones' Error Monitor
- Loss of Input Alarm (All Zeros Detector)
- Decode Data in NRZ Form
- Low Power Operation
- 2.048MHz or 1.544MHz Operation

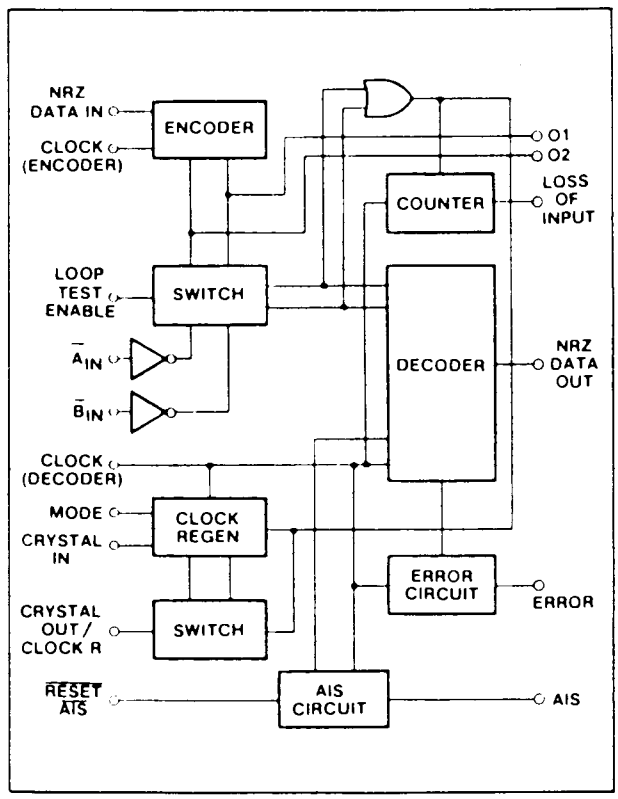


Fig.2 Block diagram

ABSOLUTE MAXIMUM RATINGS

The absolute maximum ratings are limiting values above which operating life may be shortened or specified parameters may be degraded.

Electrical Ratings

+Vcc	-0.5V to +7V
Inputs	Vcc +0.5V Gnd -0.3V
Outputs	Vcc, Gnd -0.3V

ELECTRICAL CHARACTERISTICS

Test conditions (unless otherwise stated):
Supply voltage V_{CC} 5V ± 0.5V Ambient temperature T_{amb} 0°C to +70°C

Static characteristics

Characteristic	Symbol	Pins	Value			Units	Conditions
			Min	Typ	Max		
Low level input voltage	V_{IL}	All inputs	-0.3		0.8	V	
Low level input current	I_{IL}				50	μA	V_{IL} 0V
High level input voltage	V_{IH}		2.0		V_{CC}	V	
High level input current	I_{IH}				50	μA	V_{IH} 5V
Low level output voltage	V_{OL}	All outputs			0.4	V	I_{sink} 2.0mA
High level output voltage	V_{OH}		2.8			V	I_{source} 2mA) both
			V_{CC}-0.75			V	I_{source} 1mA) apply
Supply current	I_{CC}			2	4	mA	All inputs to 0V All outputs open circuit

Dynamic Characteristics

Characteristic	Symbol	Value			Units	Conditions
		Min	Typ	Max		
Max. Clock (Encoder) frequency	$f_{max(enc)}$	4.0	10		MHz	Figs.10,15
Max. Clock (Decoder) frequency	$f_{max(dec)}$	2.2	5		MHz	Figs.11,15
Propagation Delay Clock (Encoder) to O_1, O_2	$t_{pd1A,B}$			100	ns	Figs.8,10,15 See Note 1
Rise and Fall times O_1, O_2				20	ns	Figs.10,15
t_{pd1A} - t_{pd1B} difference				20	ns	Figs.10,15
Propagation Delay Clock (Encoder) to Clock Regenerate	t_{pd1A}			150	ns	Loop test enable '1', Figs.10,15
Setup time of NRZ data in to Clock (Encoder)	t_{s3}	75			ns	Figs.7,10,15
Hold time of NRZ data in	t_{h3}	55			ns	Figs.7,10,15
Propagation delay $\overline{A}_{IN}$, $\overline{B}_{IN}$ to Clock Regenerate	t_{pd2}			150	ns	Loop test enable '0' Figs.13,15
Propagation delay Clock (Decoder) to error	t_{pd4}			200	ns	Figs.12,15
Propagation delay $\overline{\text{Reset AIS}}$ falling edge to AIS output	t_{pd5}			200	ns	Loop test enable '0', Figs.14,15
Propagation delay Clock (Decoder) to NRZ data out	t_{pd6}			150	ns	Figs.7,11,15 See Note 2
Setup time of $\overline{A}_{IN}$, $\overline{B}_{IN}$ to Clock (Decoder)	t_{s1}	75			ns	Figs.7,11,15
Hold time of $\overline{A}_{IN}$, $\overline{B}_{IN}$ to Clock (Decoder)	t_{h1}	5			ns	Figs.7,11,15
Hold time of $\overline{\text{Reset AIS}}$ '0'	t_{h2}	30			ns	Figs.7,14,15
Setup time Clock (Decoder) to $\overline{\text{Reset AIS}}$	t_{s2}	100			ns	Figs.7,14,15
Setup time $\overline{\text{Reset AIS}}$ '1' to Clock (Decoder)	t_{s2}	0			ns	Figs.14,15
Propagation Delay Clock (Decoder) to LIP				150	ns	

NOTES

1. The Encoded ternary outputs O_1, O_2 are delayed by 3.5 clock periods from NRZ Data In ($\overline{A}_{IN}$, $\overline{B}_{IN}$ Fig.4).
2. The decoded NRZ output is delayed by 4 clock periods from the HDB3 inputs ($\overline{A}_{IN}$, $\overline{B}_{IN}$ Fig.4).

High Density Bipolar 3 (HDB3) is a pseudo-ternary signal in which the number of consecutive zeros that may occur is restricted to a maximum number of three. In any sequence of four consecutive binary zeros, the ultimate zero is substituted by a 'mark' (+ or -) of the same polarity as the previous mark, i.e. it violates AMI code (Alternate Mark Inversion) and is termed a 'violation'. To ensure parity between marks of opposite polarity, the first zero is substituted by an additional mark when there would otherwise be an even number of marks between 'violations'. Thus violations alternate in polarity.

FUNCTIONAL DESCRIPTION

Functions Listed by pin number

1. NRZ data in
Input data for encoding into ternary form. The data is clocked by the negative-going edge of the Clock (Encoder).

2. Clock (Encoder)
Clock for encoding data on pin 1.

3. LIP
Loss of input circuit detects eleven consecutive zeros at the decoder input and then gives an output high. Any logic 1 at the input ($\overline{A}_{IN}$ or $\overline{B}_{IN}$ = 0) resets this count.

4. NRZ data out
Decoded binary data from pseudo-ternary inputs $\overline{A}_{IN}$, $\overline{B}_{IN}$.

5. Clock (Decoder)
Clock for decoding data on $\overline{A}_{IN}$ and $\overline{B}_{IN}$, or O_1 and O_2 in loop test mode.

6,7. $\overline{\text{Reset AIS}}$, AIS
Logic 0 on $\overline{\text{Reset AIS}}$ resets a decoded zero counter and either resets AIS output to 0 provided 3 or more zeros have been decoded in the preceding $\overline{\text{Reset AIS}}$ = 1 period, or sets AIS to 1 if less than 3 zeros have been decoded in the preceding $\overline{\text{Reset AIS}}$ = 1 period to indicate loss of time slot Zero. Logic 1 on $\overline{\text{Reset AIS}}$ enables the internal decoded zero counter.

8. Mode
Mode at logic 1 selects internal crystal controlled clock regeneration and Mode at logic 0 selects external clock regeneration using, for example, a tuned circuit.

9. Ground
Zero volts.

10. Crystal In
Input to amplifier forming crystal oscillator when crystal is connected between pins 10 and 12. This pin may also be used as a 16.384MHz clock input if one oscillator is to be shared over several HDB3 encoders decoders.

11. Error
A logic 1 indicates that a violation of the HDB3 encoding law has been detected, e.g. 3 1s of the same polarity.

12. Clock R/Crystal Out
If pin 8 is at 0, pin 12 is Clock Regenerate, giving OR function of $\overline{A}_{IN}$, $\overline{B}_{IN}$, for clock regeneration when pin 14 = 0. OR function of O_1, O_2 when pin 14 = 1. If pin 8 is at 1, then pin 12 becomes Crystal Out and forms oscillator with pin 10.

13,15. $\overline{A}_{IN}$, $\overline{B}_{IN}$
Inputs representing the received ternary PCM signal. $\overline{A}_{IN}$ 0 represents a positive going 1, $\overline{B}_{IN}$ = 0 represents a negative going 1. $\overline{A}_{IN}$ and $\overline{B}_{IN}$ are sampled by the positive going edge of the clock decoder. $\overline{A}_{IN}$ and $\overline{B}_{IN}$ may be interchanged.

14. Loop test enable
TTL input to select normal or loop back operation. Pin 14 = 0 selects normal operation, encode and decode are independent and asynchronous. When pin 14 = 1 O_1 is connected internally to A_{IN} and O_2 to B_{IN}. Clock R becomes the OR function of O_1, O_2. N.B. A decode clock has to be supplied, or regenerated. The delay from NRZ in (pin 1) to NRZ out (pin 4) is about 7½ clock periods in loop back.

16,17. O_1, O_2
Outputs representing the ternary encoded PCM HDB3 signal for line transmission. O_1 and O_2 are in Return to zero form and are clocked out on the positive going edge of the encode clock. The length of O_1 and O_2 pulses is set by the positive clock pulse length. Use suitable line drivers from these two outputs such that O_1 gives positive going pulse and O_2 gives negative going pulse.

18. V_{CC}
Positive 5V ± 10% supply.

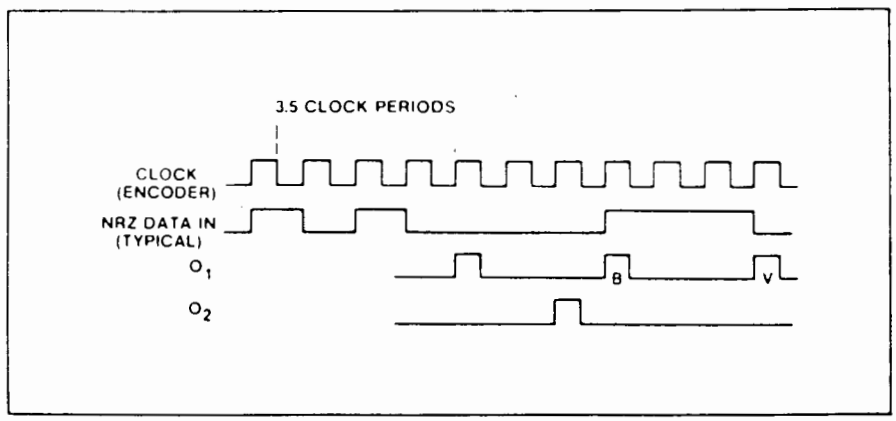


Fig.3 Encode waveforms

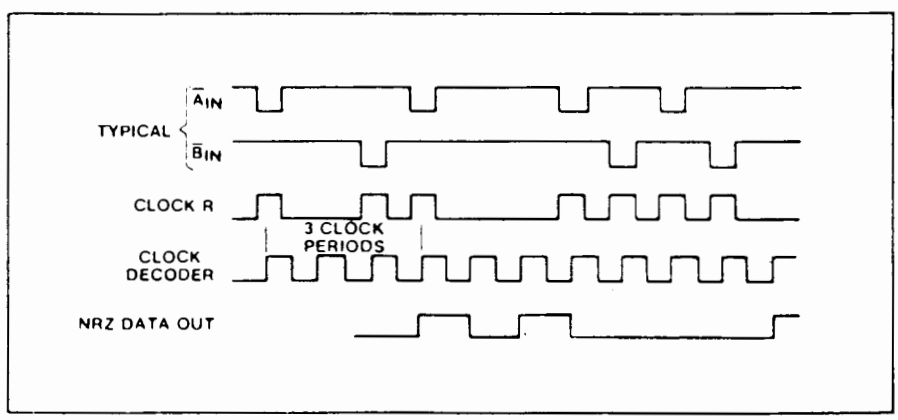


Fig.4 Decode waveforms

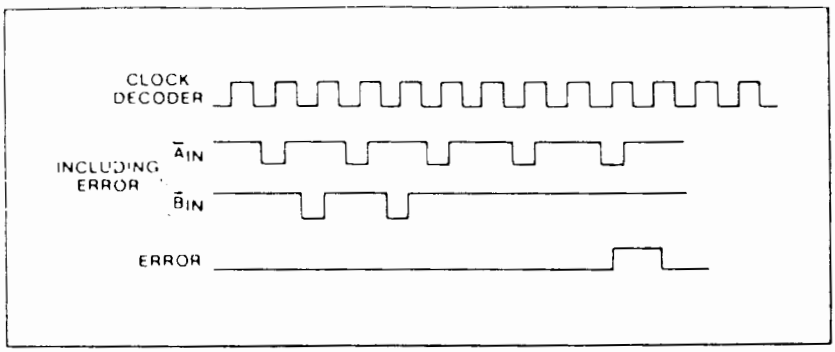


Fig 5 HDB3 error output waveforms

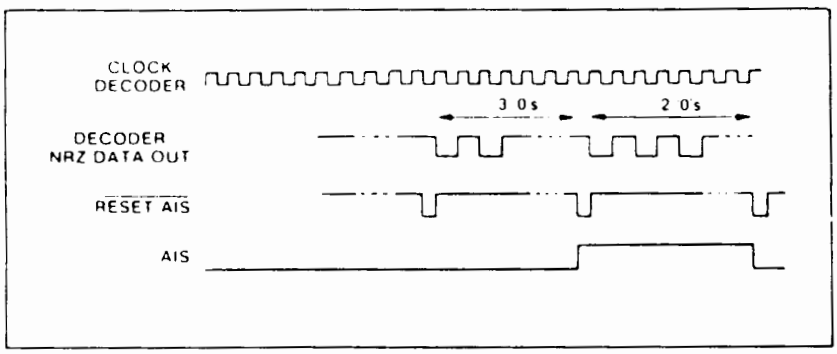


Fig 6 AIS error and Reset waveforms

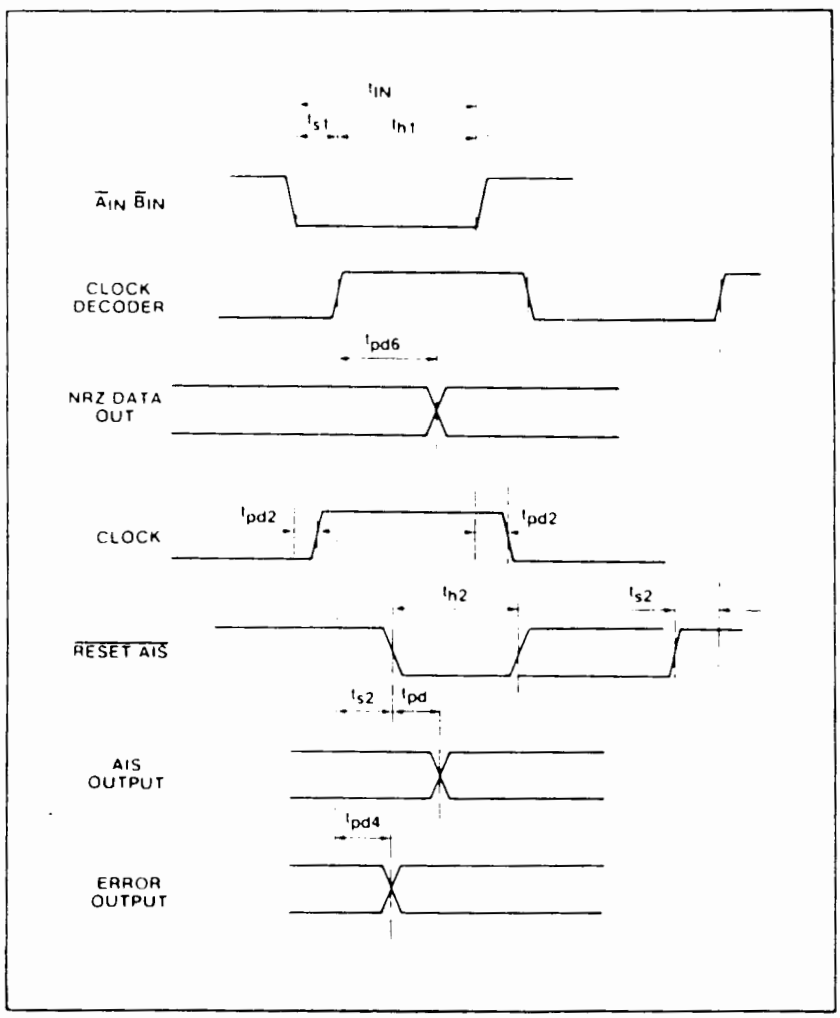


Fig 7 Decoder timing relationship

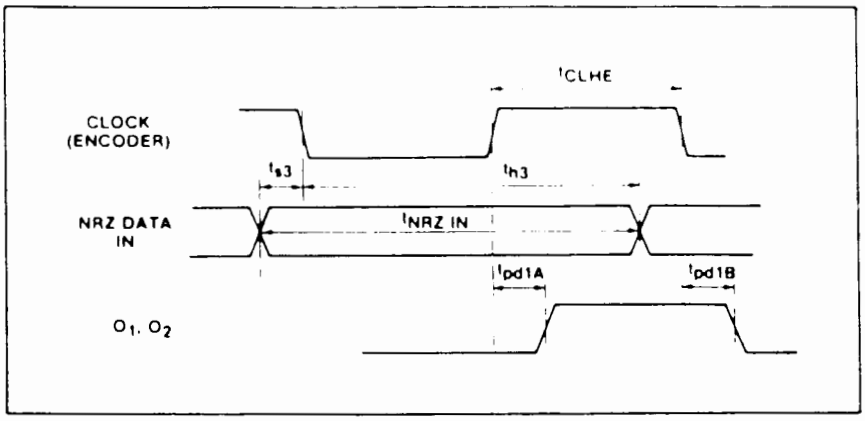


Fig.8 Encoder timing relationship

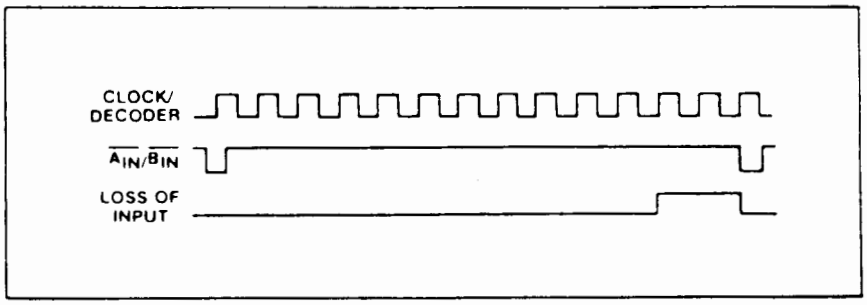


Fig.9 Loss of input waveforms

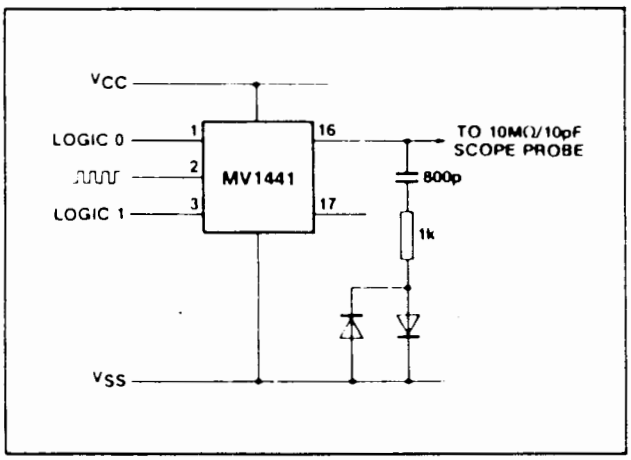


Fig.10

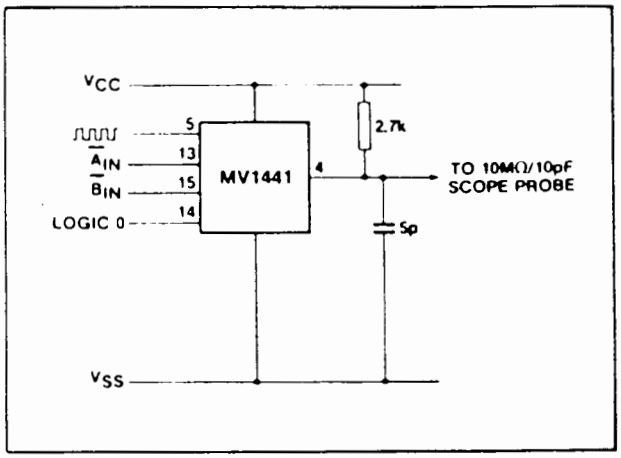


Fig.11

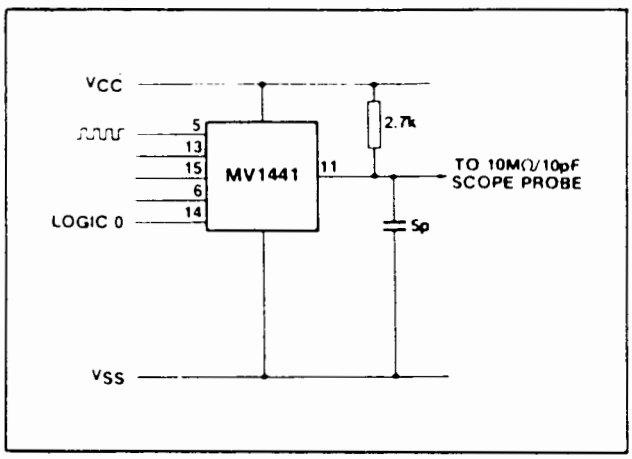


Fig.12

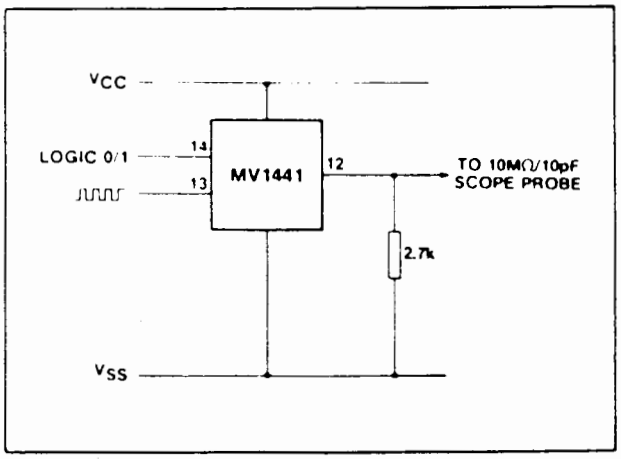


Fig.13

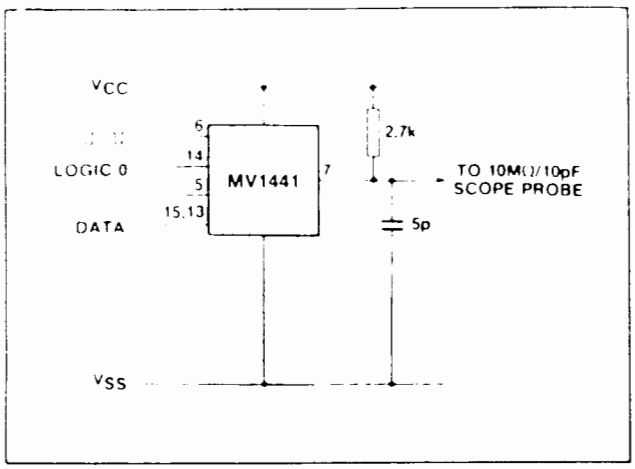


Fig 14

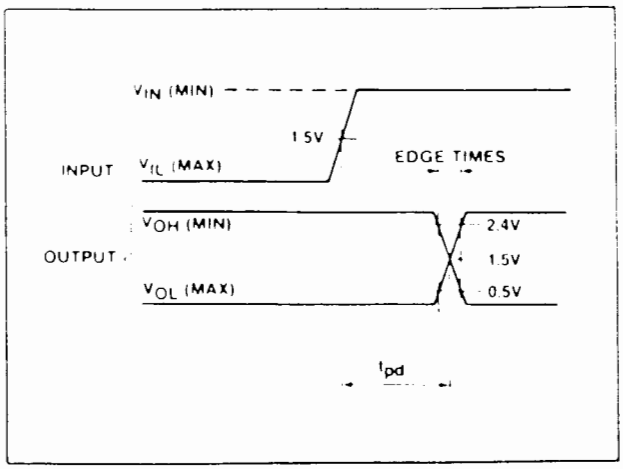


Fig 15 Test timing definitions

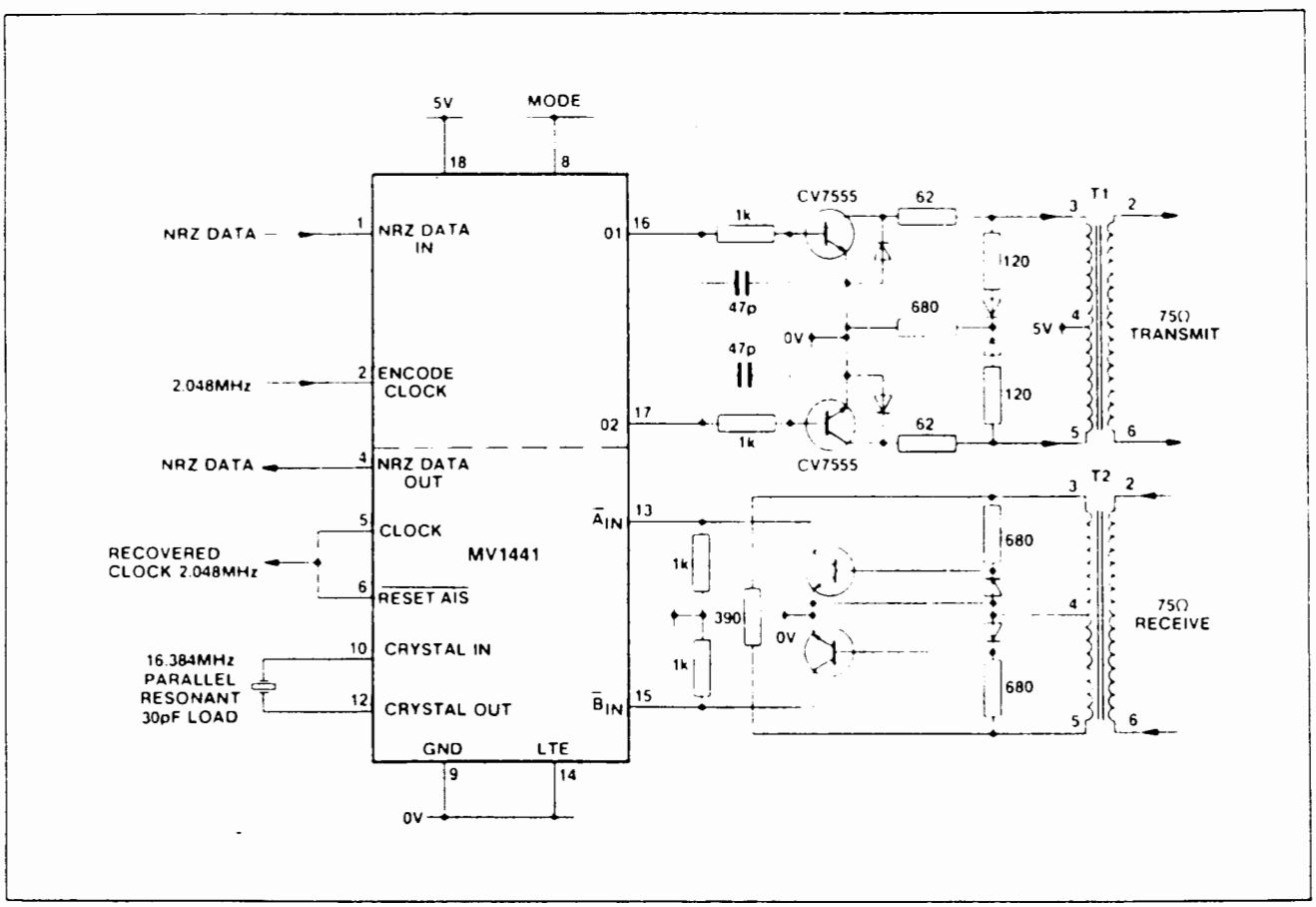


Fig 16 A typical application of the MV1441 with the interfacing to the transmission lines included

APPENDIX L

SPECIFICATION OF JITTER MEASUREMENT

The notes on jitter specification are taken from papers presented by Hewlett-Packard engineers at European Telecommunications Symposiums in 1983, 1986 and 1987. The complete references are as follows:

1) "Fundamental Limits of Jitter Tolerance In digital transmission systems." T. Crawford, G. Thow and P. Scott. Hewlett Packard Telecommunications symposium paper. 1983.

2) McDowell, Ron. "Jitter Measurements in the Integrated Digital Network". Hewlett Packard Telecommunications Symposium paper. 1986.

3) McDowell, Ron. "Jitter Measurements in the Integrated Digital Network". Hewlett Packard Telecommunications Symposium paper. 1987.

FUNDAMENTAL LIMITS OF JITTER TOLERANCE IN DIGITAL TRANSMISSION SYSTEMS

Tom Crawford, Geoff Thow and Peter Scott

INTRODUCTION

It is well known that digital transmission is affected by two main impairments:

i) additive noise which can cause erroneous decisions within regenerators.

ii) time displacement of the sampling point within the regenerator caused by phase modulation of the data stream called timing jitter.

Both impairments degrade the error-rate performance of the transmission system.

For error rate, the overall accumulation law in a chain of digital line sections is sufficiently well understood so that each regenerator section can be specified in order to meet the quality requirement on the overall connection. (1)

However, the jitter behaviour in a composite digital path, even if widely investigated by many authors (2), (3), is not yet completely known. There are some difficulties in determining specifications on jitter performance and further investigations, including experimental tests, are needed. The problem is becoming more urgent with the widespread installation of digital transmission and switching systems, which is leading to the introduction of integrated digital networks.

Effects of Jitter

Timing jitter introduces transmission impairments in two ways:

i) In the case of a digitally encoded analogue signal, jitter leads to the decoded analogue samples being irregularly spaced, thus introducing a distortion into the baseband filtered signal.

This distortion is particularly harmful in wideband coded signals, like FDM assemblies or TV signals.

Jitter can be classified as systematic or non-systematic, according to whether or not jitter sources degrade the pulse train in the same way. Since the former accumulates more rapidly, it will be predominant in sufficiently long digital paths.

ii) In the transmission of a digital pulse stream, timing irregularities introduce two effects;

a) In the regeneration process, the decision instant can be displaced from the centre of the signal eye, thus leading to a reduction in the noise margin and a degradation in the error rate performance.

b) At the output of asynchronous demultiplexers (or at the input of digital exchanges) slips can occur, due to overflow of the elastic store, which causes losses of frame alignment in the tributary signals.

This article analyses the fundamental jitter tolerance of a class of digital signal regeneration and explains an inherent source of jitter in digital multiplexing/demultiplexing. Then, taking account of these factors, illustrates how a consistent approach to the specification and measurement of jitter is emerging.

FUNDAMENTAL JITTER TOLERANCE IN DATA REGENERATION

Defining Jitter

We may define pk-pk jitter (in seconds, or bits, or unit intervals) as the maximum peak-to-peak displacement of the ith bit or symbol with respect to its position in a hypothetical unjittered reference stream.

More specifically for periodic jitter terms we may define it with the aid of FIGURE (1) as the maximum displacement of the n-th bit occurring in one-half cycle of the phase modulating function. In the second half of the cycle the phase will be lost relative to the reference to give a mean phase displacement of zero.

As an introduction let us derive an expression for the peak to peak jitter in seconds and in bits for this triangular case in terms of the modulating rate fj, the bit-rate f_b and the peak frequency deviation of the bit-rate, Δf.

Let n be the number of bits occurring in the positive half of the jitter modulating period $\frac{1}{2f_j}$

So $n = \frac{1}{2f_j} \times$ mean bit-rate

i.e. $n = \frac{1}{2f_j} \cdot \frac{(f_b + \Delta f) + f_b}{2}$ bits

Now n bits without jitter take $n \cdot \frac{1}{f_b}$ secs

i.e. $\frac{1}{2f_j} \frac{(f_b + \Delta f) + f_b}{2} \cdot \frac{1}{f_b}$ secs

So the peak to peak jitter is given by

which allows jitter at sufficiently low frequencies to be passed on to the demultiplexer where it is accommodated in the tributary store.

This leads to the definition of a performance standard for the minimum value of tolerance to input jitter that can be applied to all equipments, "The Lower limit of maximum tolerable input jitter" and when applied in conjunction with the output jitter specification ensures compatibility.

The lower limit of maximum tolerable input jitter (jitter mask) is tested by applying sinusoidal jitter to a PRBS simulating a traffic signal. The sinusoidal jitter used for test purposes does not simulate the noise-like jitter that is generated within the network, but it does provide information regarding the peak-peak alignment jitter tolerance (A_2), timing-recovery circuit bandwidth (f_5) and buffer storage capacity (A_2 or A_1).

The positioning of this lower limit of maximum tolerable input jitter mask is shown in FIGURE (20). To permit simple correlation between measured output jitter and input jitter tolerance, then the maximum permissable output jitter is measured in two frequency bands;

i) a high pass filter having a cut-off frequency f_5 hence the output jitter from the equipment under test will not cause errors at the decision circuit of the subsequent equipment, provided that the peak-peak jitter at the output of the filter is less than A_2 (FIGURE 20).

ii) a bandpass filter having cut-off frequencies f_1 and f_5, hence the output jitter from the equipment under test will not cause errors at the decision circuit of the subsequent equipment, provided that the peak-peak jitter at the output of the filter is less than A_1 (FIGURE 20)

A compromise is necessary when establishing the jitter amplitude (A_1). This is necessary because pattern dependent jitter from digital line sections can increase without limit as the number of regenerators is increased whereas, except at low frequencies, equipments such as muldexes have an upper limit to the amplitude of jitter that can be accommodated, which is determined by the size of the buffer stores. Peak-peak jitter amplitudes in excess of several unit intervals are possible from digital line sections having more than 100 regenerators.

The value of (A_1) should therefore represent a reasonable compromise between the lengths of digital line section that are likely to occur, the size of buffer stores in equipments and the additional cost of providing jitter reducers where these are shown to be necessary. (For the high rate digital line sections, however, jitter reducers can be incorporated in line terminal equipments with little or no cost penalty).

Having provided specifications for the control of jitter within component parts of a digital line section it only remains to specify the jitter transfer function of the section so that they may be cascaded in a controlled manner.

This jitter transfer function will in practice reflect the characteristic of the demultiplexer phase lock loop timing reconstruction circuit and therefore takes the form of FIGURE (21).

Brief Summary of test equipment requirements (

Essentially what is required is:

1) a jitter generator capable of:

   a) providing sinusoidal phase modulation on PRBS test patterns in order that:

      i) the lower limit of maximum tolerable input jitter can be performed easily.

      ii) the upper limit of maximum tolerable input jitter can be examined, i.e. the safety marg

      iii) a known input jitter amplitude can be supplied for jitter transfer function measurements

   b) The jitter frequency and jitter amplitude ranges should be wide enough to satisfy a).

2) a jitter measuring circuit capable of:

   a) recovering a jittered clock signa from the incoming jittered data without inadvertently filtering any of the jitter components that should be passed to the jitter demodulator and the measurement circuit.

   b) providing a measurement of the instantaneous peak-peak jitter amplitude, to perform a measurement of jitter transfer function on sinusoidal jitter stimulus.

   c) providing a variable threshold sc that the number of occasions that peak-peak jitter amplitudes excee this threshold can be counted.

   d) providing a measurement of the maximum peak-peak jitter amplitud over a specified gating interval to conform to the specification of maximum permissable output jitter for digital equipment.

   e) providing the filtering to fully specify measurement d).

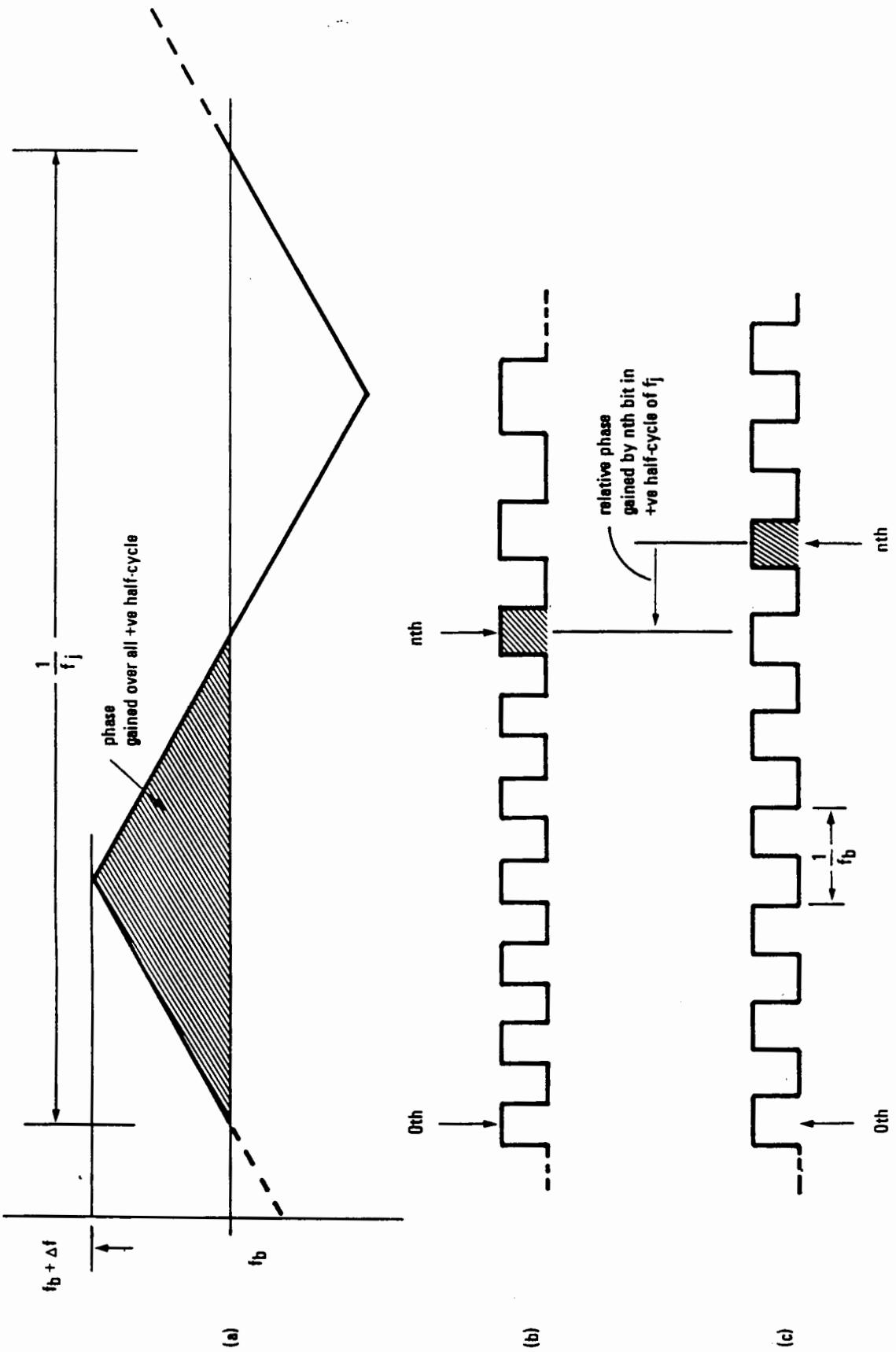


Figure 1. Triangular jitter modulation waveform