A CODE DIVISION MULTIPLE ACCESS SPREAD SPECTRUM LOCAL AREA NETWORK

Mark Alastair Mattingley-Scott, B.Sc.

A thesis submitted in accordance with the regulation for the degree of Doctor of Philosophy in the University of Durham, Department of Applied Physics and Electronics

1989

A Code Division Multiple Access Spread Spectrum Local Area Network

Mark Alastair Mattingley-Scott

ABSTRACT

This thesis describes the application of Spread Spectrum techniques to Local Area Networks. Spread Spectrum communications were developed to provide a secure means of transmitting intelligence during times of hostility, and their applications have been increasing since they were first discovered. One of the main applications for Spread Spectrum techniques, and pseudo random binary sequences in particular, is in the optimum use of communication channels. A number of Spread Spectrum code types have been described in the open literature, such as Maximal Length codes, Gold Codes, Kasami Codes and Bent codes. Until now, Spread Spectrum systems always used these determinstic codes to spread and de-spread data. A new method based on an analogy between equilibrium thermodynamics and the solution of combinatorial minimisation problems of order O(n!), known as simulated annealing, is used to generate code sets. These code sets can then be used to modulate data in any Spread Spectrum system.

ACKNOWLEDGEMENTS

I would like to express my deeply felt gratitude for the excellent advice, consistent guidance and constructive criticism of my supervisor Professor C. T. Spracklen during my three years at Durham. I am indebted to him for his supervision and help in securing funding and equipment.

I would also like to thank the former head of the Applied Physics Department, Professor G. G. Roberts for allowing me to use the facilities of the department and also the current Head of Department Professor Mars. I am certain the my time at Durham was helped by the co-operation and goodwill of the technical staff at the Applied Physics Department and my thanks go to them also.

The other members of the Digital Electronics Research Group were all responsible in one way or another for the success of my work, because of their help and advice. My thanks also go to the staff of the Library and my college, Van Mildert College for their help.

Finally I am indebted to friends and particularly to my parents for providing the encouragement to complete this work.

Table of Contents

Table of Figures

CHAPTER 1

INTRODUCTION

The work described in this thesis is concerned with two aspects of communications: The combination of Local Area Network (LAN) and Spread spectrum (SS) coding techniques. In order to provide a background to the work contained, a brief history of the two subjects is included.

1.1 Historical Background

The use of electrical signals to communicate information dates from before the start of this century - In the 1830's the first telegraph system was invented by Morse, Wheatstone and Cooke and four decades later the telephone was invented by Alexander Graham Bell.

Towards the end of the nineteenth century the first telex system was introduced and, with it, a simple acknowledgement protocol which allowed operation of the equipment without human supervision. During the second world war the importance of security meant that much effort was devoted to making communications secure, and this resulted in advances in cryptography and information coding schemes. One such was Spread Spectrum, which attempted to prevent unauthorised reception of radio signals by changing the frequency of operation in a predetermined manner. If the receiver and transmitter knew the correct frequencies and the order in which they were to be used, the chance that an eavesdropper would correctly intercept a signal was low. Interestingley, one of the first patents on this method is held by the actress Hedey Lamarr [1].

In 1949 Claude Shannon defined the capacity of a communication channel to transfer information in terms of the Signal to Noise Ratio (SNR) of the channel. The effect of his work on the progress of communications was considerable since it showed that, by choosing a suitable coding method, the maximum channel information capacity could be approached [2]. The next few years saw a large effort to find suitable coding schemes that would realise the theoretical limit determined by Shannon and this resulted in the work of Viterbi, Bose, Chaudhuri, Hocquenghem, Reed, Solomon and others. The realisation of a coding method that would allow information to flow at the maximum channel capacity for a given noise level has never been reached, although some specialised codes have come close [3]. Using the concept of channel capacity, Spread Spectrum provided a way to utilise the channel efficiently, with the ability to adapt to changing conditions such as noise, jamming, or the presence of other signalling.

At the same time as Shannon derived the channel capacity, work was being carried out on the first electronic digital computers. These computers were used in government agencies, large corporations and some universities. Facilities were centralised with a maximum of two or three computers per installation. As the number of computers within an organisation grew, the need for computer to computer communications became important, allowing access to all the resources available with the consequent improvement in system reliability through redundancy. In the late 1950's the United States Advanced Research Projects Agency (ARPA) started work that led to the development of Wide Area Networks (WAN), Local Area Networks (LAN) and Computer Communications as a whole. This resulted in the construction of a communications network at the University of Hawaii to allow contractors to the U.S. Department of Defence to share computing resources.

The ARPA computer network thus formed was one of the first networks that connected many remote, autonomous computers. It is from the ARPA network that we will derive our definition of a distributed computer system [4]. **A distributed system is one in which the computing functions are dispersed among several physical computing elements** [5].

The issue of costs has been paramount in the development of computer networks; when the ARPA network was built, computers were very expensive compared to the communications systems used. The relative cost of computing resources compared to communications resources has dropped, and this has driven the development of computer networks greatly. It is now feasible to place small, inexpensive computers at remote sites, connected to one or more large central computers, to perform a variety of tasks, for example industrial process monitoring. In more recent years we have seen a proliferation in types of network, varying in physical size, usage and communications mechanism. Networks now exist that span the earth, while others cover only a few centimetres. Applications vary from inter-processor communication within multiprocessor machines to world-wide remote security monitoring. Communications mechanisms range from low bandwidth telephone circuits to very high bandwidth satellite links and fibre optic links.

The ARPA network transferred data by breaking it down into a set of constant sized lumps, or packets. These were routed across the network using a number of methods or protocols. In particular, at the University of Hawaii a method for the transmission of such packets over a satellite link was developed called ALOHA [6].

The principle of operation of ALOHA, based on the assumption that a satellite has a large bandwidth but a long fixed delay, was to allow any node to transmit at any time. Each node had to acknowledge the successful delivery of packets by waiting for them to be re-broadcast from the satellite on its downlink. If the sent message was not received within a fixed maximum time, or the message received had been corrupted by another message, then the transmitter waited for a time determined by a Poisson distribution before attempting re-transmission. Various improvements to the basic ALOHA protocol followed, resulting in Slotted ALOHA, in which all transmissions had to fall within well defined time slots. This latter method is the basis for many of current local area network access mechanisms, including the Ethernet.

Other types of LAN were developed in the 1970's and early 1980's, and many of these were based on the ring structure. The impetus for using a ring network came from the manufacturing industry, where cheap cabling and relatively long distances occur.

In all examples of ring networks, data is sent serially around the ring, each node receiving and re-transmitting in turn. A more detailed taxonomy of local area networks is included in chapter 2. Current LAN's can be categorised according to one or more constraints, for example speed of response, maximum message delay time, access mechanism, signalling method and maximum or minimum throughput. Among the more useful categorisations are ones which distinguish between the different methods for determining access to the network. This includes Demand Assignment (DA) networks, where access to the network is assigned based on a stated need by a controller; Polled and Poll-Response networks, where access to the network is allocated according to some ordering of nodes; and Contention networks, where access is not restricted.

1.2 Objectives

The background for the work contained in this thesis is the requirement for a high reliability local area network for use on ships. In 1976 the Ministry of Defence started work on a **Combat Systems Highway** (CSH) for use as a command and control system in modern frigates. This work proved to be highly successful and resulted in the specification of a defence standard defining the operation of all future naval networks [7]. As a result of the success of the first CSH, a series of documents [8] was prepared which studied the impact of future requirements and operational scenarios on computing and communication requirements in ships. The reports compared various conventional computer architectures and provided estimates of the likely communications bandwidth needed.

It was felt to be unrealistic to predict the future needs of the Navy with accuracy. In particular, bandwidth requirements were felt to be subject to two opposing trends. First, the need for increased bandwidth to accommodate teleconferencing type facilities and high speed raw data for gun-aiming etc. and second, the trend towards intelligent peripheral devices connected to networks. An example of the latter is the GKS [9] graphics standard, which communicates graphics information as high level graphics **primitives** such as squares, circles, lines, points etc., rather than as the raw video data. The future requirements lie in the middle of these two extremes, with the bandwidth requirements being largely determined by the technology.

The Combat Systems Highway is a Time Division Multiple Access (TDMA) LAN and has fixed capabilities which cannot easily be extended. Addition of further nodes to the polling list, or sudden demands for increased bandwidth from specific nodes will adversely affect the system performance, and this was highlighted as an important fault of present approaches. The only workable solution was to try and predict the requirements, and add further capabilities, where necessary, to allow for changes. TDMA LANs operate such that the network medium is quiet for a large part of the time, with short bursts of high bandwidth data appearing occasionally, unless the network is heavily loaded. This mode of operation places constraints on the network interface hardware in each node, because it has to be capable of operating at the full system data rate. In Ethernet for example, the system data rate is 10MBps but on a heavily loaded system the point to point data rate may only be 10's of KBps. This represents a poor utilisation of the available bandwidth.

The ideal system would adapt the performance of the network to the instantaneous network loading, providing a high throughput on demand, with a low throughput for highly loaded networks. In addition, the possibility of having a prioritised data rate at each node is a logical extension to this concept.

1.3 The Application of Spread Spectrum Techniques to Local Area Networks

The use of Spread Spectrum techniques to provide a multiple access mechanism had been known for some time [10] and other workers in the area were using Code Division Multiple Access techniques in Fibre-Optic networks. The use of direct modulated data, with a coaxial cable based network is a more evolutionary approach to the use of Spread Spectrum techniques. The use of fibre optic signalling has become feasible at very high data rates and it has been suggested that the only way to exploit the full benefits of the very large bandwidth of monomode coherent fibre systems is with some form of Spread Spectrum. Modulation of coherent light at 10's of Gigabits per second is feasible, but on a 10 THz fibre still represents a thousandfold waste in channel capacity. With the incorporation of direct optical modulators and correlators, Spread Spectrum would provide a method for using the full capacity.

1.4 Thesis Structure

In CHAPTER 2, the subject of Local Area Networks is covered in detail. The ISO OSI seven layer protocol standard is discussed. A short taxonomy of Local Area Networks is included for completeness.

CHAPTER 3 Introduces the requirements for spread spectrum systems and LAN's in particular. The chapter contains an overview of spread spectrum concepts such as process gain (PG), channel capacity (C), chip rate and finite field algebra. The large variety of deterministic spread spectrum sequences are discussed which can be used in CMDA applications, with derivations of relevant properties. Non-deterministic sequences are introduced and the technique of Simulated Annealing (SA) described with its application to code set design. Appendices are included which cover the specific problems of correlation bounds and the theory of Galois Fields.

CHAPTER 4 describes the work done on the simulation of correlation values for a sequence set of maximal-length sequences, and shows the results of these simulations. A listing of the simulator program is included in APPENDIX F, and a glossary of the functions in APPENDIX G.

CHAPTER 5 descibes the theory of simulated annealing, and explains how it is used to construct arbitrary length and size code sets for CDMA applications such as LAN's. A listing of the software is included in APPENDIX H

In CHAPTER 6 the Spread Spectrum Local Area Network is described, along with the construction of a prototype to demonstrate its viability. A listing of the Prototype network controller software is included in APPENDIX L, and a set of circuit diagrams for the Prototype in APPENDIX M

CHAPTER 7 completes the work on Spread Spectrum LAN's, with a description of the possible mechanisms for managing the network, and for coping with the problems inherent in this design. Security of code sequence distribution is described, using one of several possible encryption methods. APPENDIX E descibes three popular encryption methods.

CHAPTER 8 includes the results of the code set design program. Several code sets have been designed and they are illustrated graphically in this chapter.

CHAPTER 9 Concludes the main body of the thesis.

APPENDIX J and APPENDIX K are concerned with the Durham University Text Compressor. They describe the operation of the Text Compressor.

CHAPTER 2
LOCAL AREA NETWORKS

In this chapter we will produce a definition of the term Local Area Network and describe in detail the operation of certain types of Local Area Network. Particular attention will be drawn to the requirements of a LAN that supports the functions of command and control in a demanding environment. The intended application for the work described in this thesis is as the communications system for a ship-borne command and control system.

2.1 Types of Local Area Network

Local Area Networks can be classified in many ways, but it is convenient to consider the topology of the network as its classification feature. There are several possible topologies: Rings, Buses, Stars, Mesh, etc [11]. Star networks are not considered here because of their reliance on a single vulnerable central controller. Mesh networks have potentially the best reliability and resistance to damage because of the inherent redundancy of the communications paths. The problem of routing messages around a mesh network has not been solved except in a few special cases and it is because of this that they are little used in local area networks. The technique described in this thesis is applicable to mesh networks, but because of multiple signal pathway interactions (multipath) the efficiency of the network suffers, so they will not be described in detail here.

Of the remaining types of network topology (rings and buses), there are six main types of access mechanism:

2.1.1 Slotted Rings

This access mechanism was developed by Pierce [12] and work still continues at Cambridge University. A fixed number of slots circulate around the ring continuously. The first data bit of each slot marks that slot as either full or empty. If a node wishes to transmit data, it waits for an empty slot, marks the slot as full and sets the destination address, data bytes and status bits in the slot. Once the slot has returned to the originating node, it is marked as empty and can then be used by other nodes. The main advantage of the slotted ring is that it guarantees a maximum waiting time before a node can send data. The main disadvantages of the slotted ring are that it has low utilisation and the slot size is fundamentally related to the ring length. In the Cambridge Ring, of the thirty seven bits in a slot, sixteen are data, eight are source address, eight are destination address and there are five control bits; and it requires a centralised monitor service to handle initialisation of slots and removal of corrupted slots, see Figure 1 below.

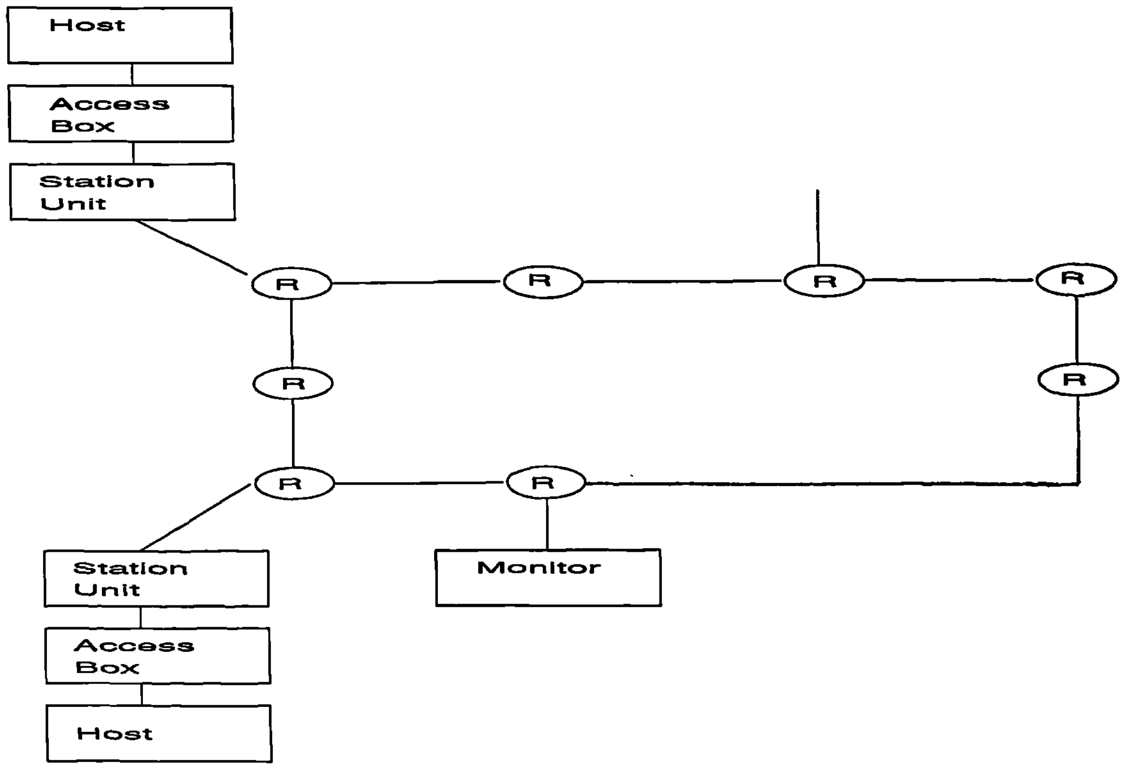


Figure 1 : Slotted Ring Network Topology

2.1.2 Register Insertion Rings

This type of ring was developed originally at Ohio State University [13]. Operation of the access mechanism is explained with reference to Figure 2 below.

First, consider the case in which the node has no data to send, but is passing packets on. When the ring is idle, the input pointer points to the rightmost position of the shift register, and as a packet is received it is inserted bit by bit into the shift register, with the input pointer shifting left, once for each bit. Packets begin with an address field, and as soon as the address is shifted in, the node decodes the address. If the address is valid (i.e. the message is destined for this node) the remainder of the packet is shifted into the shift register and copied into the node. The packet can then be removed, or passed on to further

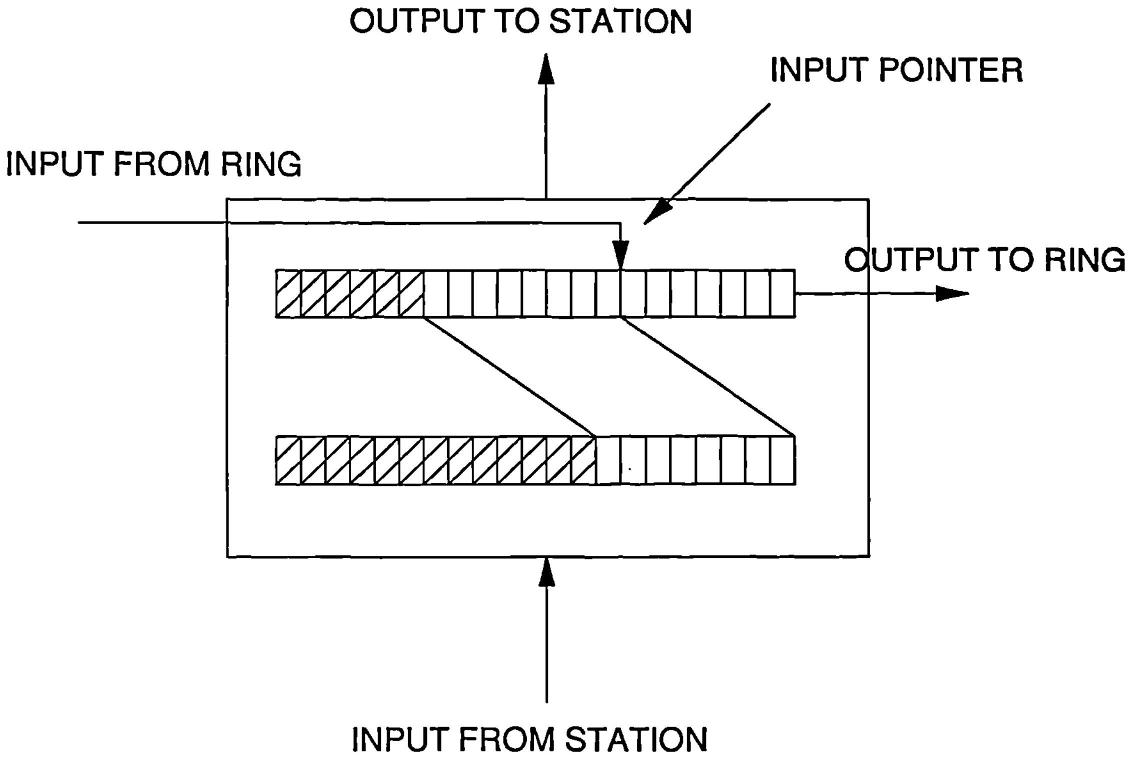


Figure 2 : Register Insertion Ring Topology

nodes if it is intended for broadcasting. If the packet was not destined for this node, then bits are shifted in at the input pointer and shifted out at the right hand side of the shift register, until no more bits are present or another packet arrives for decoding.

Second, consider the case when the node has data to send. A packet to be transmitted is placed in the output buffer. If the ring is idle, the packet can be shifted onto the ring. If the ring is not idle, then the node waits until a portion of the shift register equal to the length of the packet to be transmitted becomes free. The output buffer is then copied into the shift register and the input pointer adjusted to allow for the extra data to be sent. The main advantage of this access mechanism is that it achieves the maximum ring utilisation possible; the ring is never idle unless there is nothing to transmit. The main disadvantage is that address fields can become corrupted and never leave the ring. This problem is often overcome by the use of error correcting codes.

2.1.3 Token Rings

This is probably the oldest ring access mechanism, originally developed in 1969 and called the Newhall loop after its designer [14]. See Figure 3 below.

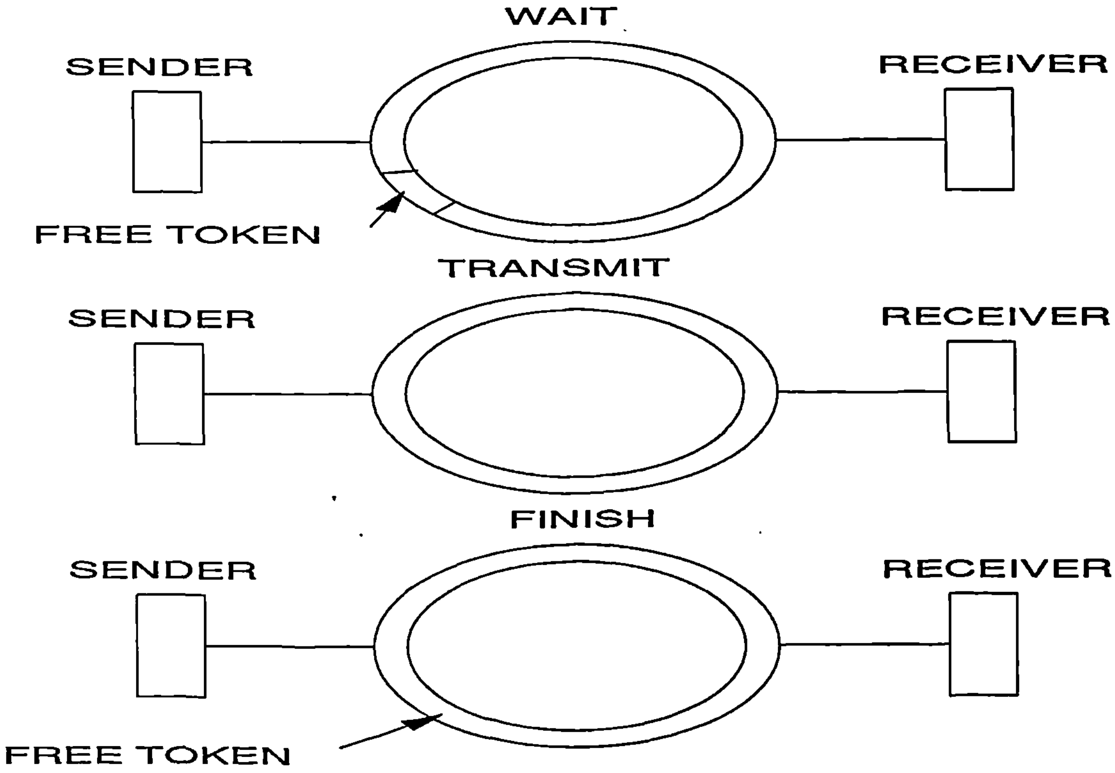


Figure 3 : Token Ring Topology

In the wait state, the Sender (marked S) waits for a free token. In the Transmit state, the sender changes the token status to busy and appends data to it. The Receiver (marked R) copies the data and passes the token on, setting status bits in the token on the fly. In the Finish state, the sender generates a free token once it has received the transmission header back from the receiver.

There are two error conditions that would stop the ring working: loss of the token and persistent busy token. To overcome this, one node is designated as acting master. This node handles recovery of missing tokens and removal of busy tokens after two consecutive circuits around the loop. The main advantage of the token ring is that traffic can be regulated by use of priorities and variable length packets. Its main disadvantages have already been mentioned.

2.1.4 Carrier Sense and Collision Detect Bus Networks

The most commonly used access mechanism for bus/tree topologies is Carrier Sense Multiple Access with Collision Detection (C.S.M.A./C.D.). To transmit data, a node listens to the medium until there is no activity. Transmission then begins until either the data is transferred successfully or a collision occurs. If another node accesses the medium during the end to end propagation delay of the medium this causes a collision. This normally catastrophic event is countered by jamming the network for a short period of time to let all the other nodes know that there has been a collision, and then waiting for a short random time before attempting transmission again. Packets must be at least twice as long as the propagation time of the network, in order that the collision detection works properly. See Figure 4 below.

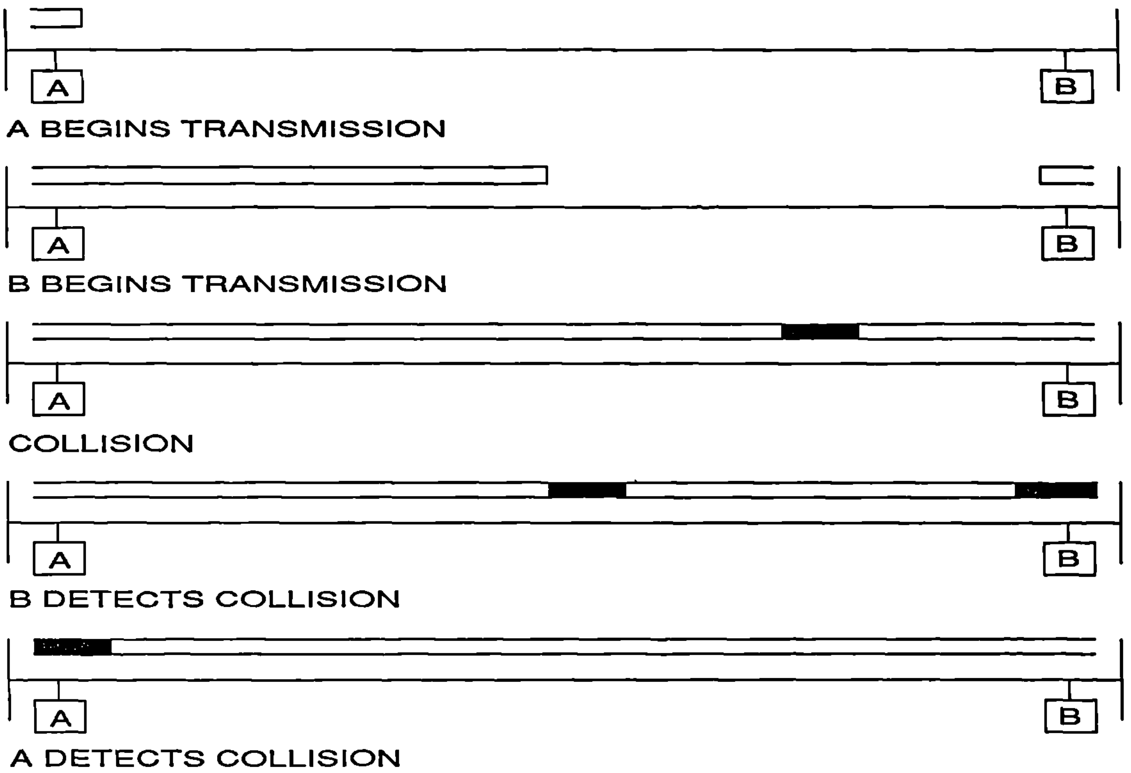


Figure 4 : Carrier Sense Multiple Access Topology

2.1.5 Token Bus Networks

With this access mechanism, the nodes on the bus form a logical ring so that bus access for a given node is assigned either on a round-robin basis, or prioritised. See Figure 5 below.

A control packet (token) is passed to each node in turn and this allows the node to transmit data. When it has finished, or run out of time, it passes the token to the next node in sequence. The problem with this mechanism is that it requires a sophisticated management function, for the following reasons: To add a node to the polling order, new nodes must be given the opportunity at regular intervals to make themselves known; To remove a

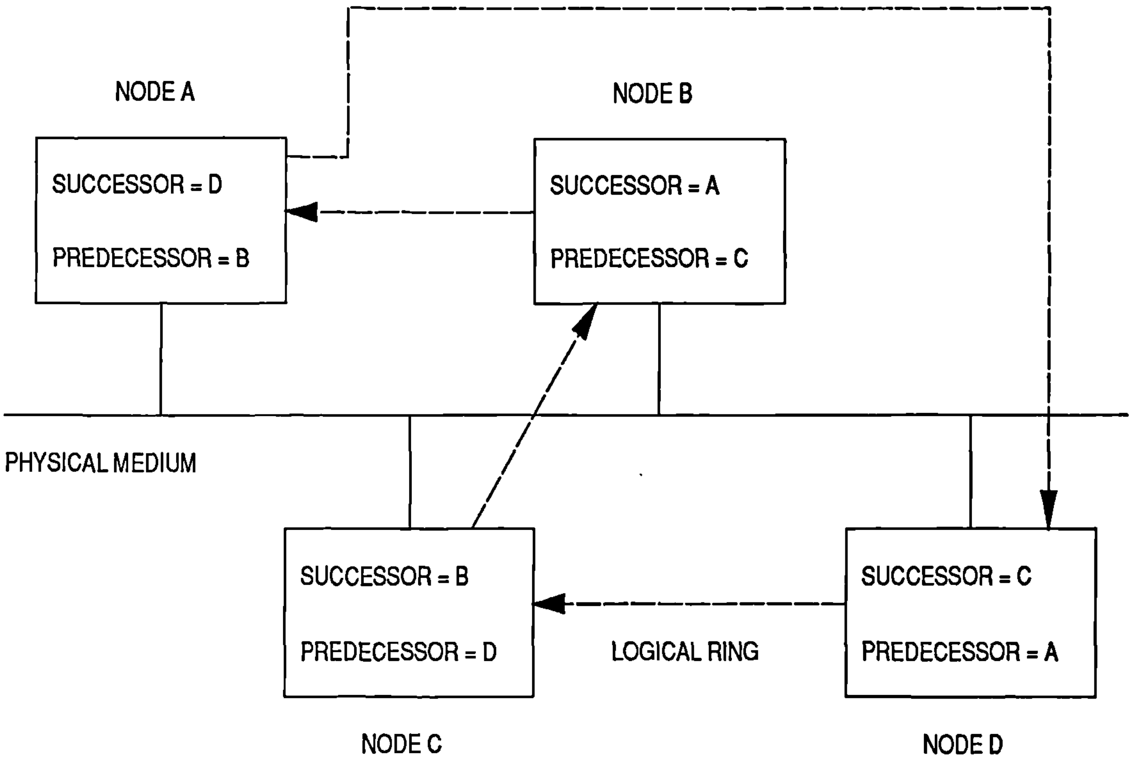


Figure 5 : Token Bus Topology

node from the polling order, the node has to inform its predecessor of the address of its successor to splice the poll; A cooperative decentralised mechanism for initialising the ring is necessary. Solutions to these problems are included in the IEEE 802 network standard on token buses [15].

2.1.6 Poll Response Bus Networks

In this type of network, a **Bus Master** provides overall control of the nodes and allocates permission to transmit to them, normally in some agreed order: round robin or priority based for instance. If the bus master fails, then one of the other nodes automatically becomes the new bus master after a suitable period of time.

2.2 The ARE Serial Highway

The current operational command and control system local area network used by the British Royal Navy is the Admiralty Research Establishment (A.R.E.) Serial highway (A.S.H.) [16]. This network is a poll response bus network. Although poll response buses have disadvantages in terms of delay, at the time that the ASH specification was drawn up, a poll response bus offered considerable advantages compared to the available networks. These are listed below:

Guaranteed maximum message delay time. Given that there are a fixed number of nodes on the network at any one time, and that each message is limited in length, then the maximum access delay time is equal to the product of the number of nodes and the maximum message length in seconds.

Redundancy of access medium. In the ASH, the physical medium is replicated up to three times for survivability. The routing of the three separate cables depends on the type of ship to which the network is to be fitted.

Redundancy of the Medium Access Controller (MAC). By duplicating the function of MAC in more than one place, the effects of damage to the active MAC or damage to the physical medium resulting in breakage can be minimised. Ideally each node would have the capability of being the MAC. If the active MAC was damaged in the course of hostilities, then the remaining nodes would decide which was to assume the responsibility of being the new MAC and continue after a short delay.

2.3 Standards

One of the problems that the great diversity of LAN's and WAN's created was that of compatibility between different types of network. As potential for confusion was high various standards bodies started work on suitable standards for communication between equipments (protocols) early in the evolution of LAN's and WAN's. This work lead to the ISO Open Systems Interconnection (ISO-OSI) seven layer protocol reference model [17] and the CCITT V- and X- protocol specifications. The ISO-OSI seven layer model defined protocol layers for data transmission based on the following principles [18].

A Layer should be created where a different level of abstraction is required.

Each layer should perform a well defined function.

The function of each layer should be easily achievable.

The interface between layers should be chosen to minimise the information flow across the interface.

The number of layers should be large enough to allow distinct functions to be in separate layers and small enough to prevent the whole being too unwieldy.

A description of the function of each of the seven layers is given below:

2.3.1 Physical Layer

This is concerned with the transmission of raw bits over the communication channel. Typical parameters are signalling speed, line voltages/currents, optical levels and interface specifications to computer equipment.

2.3.2 The Data Link Layer

This layer provides error free transmission to the next layer, the Network Layer. It does this by assembling input data into data frames and handling acknowledgement (ack) and negative acknowledgements (nack).

2.3.3 The Network Layer

This layer controls how packets are routed within a network. The routing information could be fixed within a given network, or dynamically varying because of external stimuli. The Network Layer is there to make this transparent to the next layer up.

2.3.4 The Transport Layer

The function of the Transport layer is to accept data from the Session layer and split it into packets for transmission by the Network Layer. It also determines what type of communication is to be carried out; normally this is an error free point to point connection that delivers packets in the order in which they are sent.

2.3.5 The Session Layer

This is the users' interface to the network. Initiation of connections between hosts is handled by this layer and once a connection is established it handles the dialogue between hosts.

2.3.6 The Presentation Layer

This layer performs functions that are sufficiently common to warrant finding a general solution for them, often available as operating system or library functions; such as text compression of large documents and encryption of sensitive data.

2.3.7 The Application Layer

The purpose of the Application layer is up to the individual user, but a good example is the problem of automatic partitioning of problems among computers on the network. This would be handled by the Application layer.

Examples of the CCITT V- and X- protocols are V24 which defines the interface between computers and terminals, X25 which defines the access protocol for packet switched networks(X25 corresponds to the lowest three layers of the OSI model) and X29 which defines the computer to Packet Assembler/Disassembler (P.A.D.) Interface.

2.4 The Future

In time, it is likely that only a few of the currently available networks will remain. The Ethernet, standardised as IEEE 802.3 is already firmly established as a commercial and academic network throughout the world. The Token Ring, or IEEE 802.5 has IBM as its main sponsor and is also well established. Other standards, such as Token Bus (802.4) and metropolitan area network (802.6) have yet to make their mark. As was mentioned in chapter 1, for ultra high speed networks using Terahertz bandwidth coherent optical fibre, the network delay completely dominates the performance. Characteristics are like the early ALOHA satellite links, but over a contained network. In this case CDMA and Spread Spectrum has much to offer with good capacity utilisation using relatively simple transmitters and receivers.

2.5 Spreadnet Operation

Each node on the network uses a binary code sequence to modulate the data to be transmitted, effectively spreading the bandwidth of the signal, and may transmit independently without first sensing the media. Interference between simultaneous transmissions is avoided by careful selection of the codes used, thus eliminating the problems of contention present in such systems as Ethernet. Nodes may select which messages they wish to receive by use of the appropriate locally generated code sequences to demodulate the data. A transmitter may broadcast to more than one node by transmitting on communal codes, or to a single node by choice of that nodes private code.

Conventional LAN systems all use TDMA in one way or another. Any node in a system that wishes to gain access to the medium must wait until such time as no other node is using the network. The node must then pass data into the network at a rate that utilises the full bandwidth of the network. This implies the use of high speed logic circuits in a system where the average data rate for each point to point link may be much lower. It is in this area that high speed networks are limited, so any scheme that provides maximum network throughput for a given bandwidth will be of benefit. For instance, in a 100 node system, with each node transmitting at 1 Mbit / second, a TDMA system would require at least 200 MHz bandwidth, which is clearly not realisable using present day technology. In Spreadnet a node may transmit continuously at a rate that closely resembles its average data rate, since simultaneous transmission by many nodes is allowed. This lowers the peak processing load on each node. At the receiver a phase shifted version of the code is generated and compared to the received signal to determine the phase of the incoming signal.

From the point of view of a single node, the signals from all the other network users appear as noise. As more users join the network, the noise power will increase and the channel capacity available to each link will be reduced.

CHAPTER 3

SPREAD SPECTRUM

3.1 Introduction

In this chapter we shall review the properties of spread spectrum communications techniques that are relevant to the work contained in the rest of this thesis.

Pickholtz defines Spread Spectrum as :

A means of transmission in which the signal occupies a bandwidth in excess of the minimum necessary to send the information; the band spread is accomplished by means of a code which is independent of the data, and a synchronised reception with the code at the receiver is used for de-spreading and subsequent data recovery.

The technique involves increasing the bandwidth of a signal or data stream by the use of coding techniques, in order to gain one or more of the advantages listed below:

Reduce the energy density of the data, i.e. disguise the information contained in the spread signal as **noise** to prevent detection by an unauthorised listener.

Allow reception of the embedded signal in the presence of noise, e.g. intentional **jamming** noise, electrical interference, or other signals using the same bandwidth during the same time interval.

The second condition includes three types of interference.

The ability of spread spectrum to decode information in the presence of jamming is especially useful in the military communications field.

Electrical interference from radio transmitters and high current switching is of a burst nature, and is a problem in a wire based communications system.

Large machinery starting and stopping and impulse interference caused by coupling between wires causes short bursts of noise across a large bandwidth. Spread spectrum techniques are able to cope with this type of problem.

In practice the total interference may be a combination of these three.

The applications described in this thesis require that more than one dialogue can be held at the same time and in the same bandwidth. Suitable codes can be chosen in order to make this possible.

3.2 Principles

A Spread Spectrum signal is one in which **a signal is spread over a frequency band larger than the minimum bandwidth necessary to transmit the information being sent**. The reference to the minimum bandwidth in the above refers to pioneering work done by Claude Shannon in the late 1940's at Bell Laboratories on information theory. As a result of studies of the entropy of a communications channel, Shannon demonstrated that the minimum bandwidth required to transmit a signal without any errors was given by the equation

$$C = W \log_2\left(1 + \frac{S}{N}\right)$$

where

C = Error free information rate in bits per second

W = Bandwidth in hertz

S = Signal power

N = Noise power

This equation relates the ability of a channel to transfer information without error within the signal to noise ratio of the channel to the bandwidth used to transmit the signal. It is clear from the above that an increase in W will result in an increase in C and it is this **increase in capacity** which lies at the heart of spread spectrum.

The amount of spreading is usually measured by the Processing Gain G_p and this is defined as:

$$G_p = \frac{W}{D} = \frac{1}{Jamming\ Margin} \quad \text{where D is the data rate}$$

Where $W \gg D$ and the channel bandwidth $W_c \approx W$.

Data spreading and de-spreading can be considered as two successive multiplications by an identical spreading signal. Any jamming or interference is multiplied by the spreading signal only at the receiver and so its bandwidth will be spread to a minimum of W. As a result, the received energy density of the interfering signal or noise will be $E_n = \frac{P_j}{W}$ and the received energy density of the data will be $E_d = \frac{P_s}{D}$ where P_s is the signal power and P_j is the jamming or interference power. Since the bit error rate or BER is proportional to the ratio $\frac{E_d}{E_n}$, this ratio is important in defining the error rate of the system. Rewriting:

$$\frac{E_d}{E_n} = \frac{P_s}{P_j} \times \frac{W}{D}$$

and so the jamming power to signal power ratio is

$$\frac{P_s}{P_j} = \frac{D}{W} \times \frac{E_d}{E_n} = \frac{\frac{E_d}{E_n}}{G_p}$$

Thus relating the jamming margin to the minimum bit energy to noise density ratio.

3.3 Types of Spread Spectrum

There are four main types of spread spectrum modulation; direct sequence, frequency hopped, time hopped and chirped. Hybrid forms exist, and are basically combinations of the above four simple types of spread spectrum. The best known example is the backup communications link for the Space Shuttle, which uses a combination of direct sequence and frequency hopping [19]. An explanation of each of the simple types is given in the next section with, in the case of frequency hopping, details of two important variations.

3.3.1 Direct Sequence

This type of modulation usually employs a code sequence to phase modulate a carrier signal, or involves directly modulating the channel with the sequence itself. There are two main groups:

Binary Phase Shift Keying (PSK) and Quadrature Phase Shift Keying (QPSK).

M-ary modulation in which data is grouped into blocks, which are used to produce a code sequence for transmission.

Spectrum spreading is achieved by multiplying each digit of a stream of data digits d, by a pseudo-random spreading sequence, x. The bandwidth of the data stream is less than the bandwidth of the spreading sequence. Multiplication of two independent signals produces a signal whose spectrum is the convolution of the spectra of the two original signals and so the product signal will have approximately the bandwidth of the larger bandwidth spreading signal.

Spectrum de-spreading is achieved by multiplying the spread signal x.d by the spreading sequence x. The result is the signal x(x.d) = d. The choice of the sequence x that satisfies the above condition is crucial to the operation of a spread spectrum system, but to operate well it must have good auto correlation properties and cross correlation properties. These will be dealt with later in the chapter, and treated qualitatively.

The processing gain of the Direct Sequence Spread Spectrum (DSSS) system is independent of the frequency of the carrier signal and so the processing gain can be written as:

$$G_p = pulsewidthofsequence \times datarate$$

3.3.2 Frequency Hopping (FH)

The basic principle is to change the carrier frequency of a signal in a pseudo random way. The receiver knows the carrier frequencies and the sequence in which they will be visited and so can pick up data and re-assemble it. There are two basic types of frequency hopping systems: Fast FH transmits one or more data bits per frequency; slow FH transmits less than one data bit per frequency. Data is normally demodulated non coherently, i.e. with possible loss of phase coherence between the transmitter and the receiver, in contrast to DSSS, but there are techniques for retaining phase information at the expense of further bandwidth.

3.3.3 M-ARY or Multitone FH

There are two variations on this type of multi-bit FH. The first transmits data as blocks of length k mapped into sequences of length 2k, drawn from a set of 2k orthogonal sequences. The sequence specifies a series of frequencies which are multiplied by the previously transmitted sequence and sent over the channel. On reception the signal is again multiplied by the previous sequence and a maximum likelihood detector used to extract the original sequence. This is then used to decode the block of data bits.

The second type of M-ARY FH is multitone frequency shift keying FH used mainly in satellite and mobile radio communications. The transmitted data from each user is formed into blocks of bits and these subsequently generate a discrete time varying level Q. This is multiplied by the unique address of the receiver to produce a series of frequencies which are transmitted over the channel. On reception the received signal is multiplied by the address of the receiver to yield the original data block. This is illustrated in Figure 6 below.

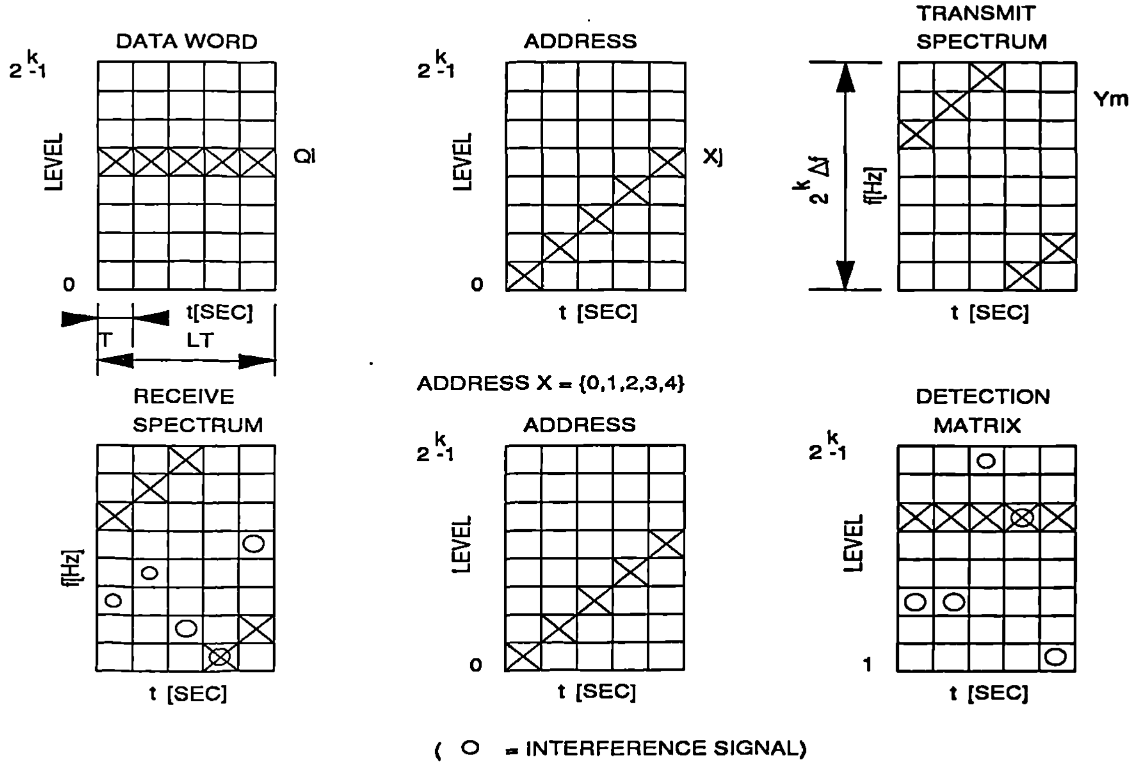


Figure 6 : Multitone Frequency Hopping

The processing gain of a frequency hopped system is given by

$$G_p = \frac{hoppedbandwidth}{databandwidth}$$

3.3.4 Time Hopping

In a time hopped system, the transmitter replaces each data bit to be transmitted by a short duration pulse whose time position in the data bit is determined by a pseudo random sequence. This type of modulation is very often used in OR-channel multiple access systems, i.e. ones in which the channel behaves like an or-gate (e.g. fibre optic systems).

3.3.5 Pulsed Frequency Modulation or Chirp

This technique is used mainly in radar systems where the requirements are for long range, or high peak power. If the change in frequency over time is discrete or it changes in a pseudo random manner, the modulation is spread spectrum.

3.4 Spectrum Measures

In a spread spectrum system, the whole bandwidth B_{tot} is used to transmit all the data sources. This is done by taking the data from each source and modulating it with a noise or noise like signal and transmitting this instead. To facilitate multiple access communications, all that is necessary is that the noise signal used by a given data source be chosen from a set of orthogonal noise signals, i.e. a set with suitable auto-correlation and cross-correlation properties.

3.4.1 Auto and Cross Correlation

The operation of correlation involves comparing two signals. In mathematical notation:

$$C_{xy} = \int_{t=-\infty}^{t=\infty} x(t)y(t)dt$$

i.e. the correlation of two signals x(t) and y(t) is the time-sum of their product. If x(t) and y(t) are different, then the operation is cross-correlation. If they represent the same signal, but with a time shift then the operation is auto-correlation. In this case the formula can be re-written:

$$C_{xx} = \int_{\tau=0}^{\tau=\infty} x(t)x(t-\tau)dt$$

Where τ is the relative shift between the signal and its replica. If the time period is restricted to a finite one, the auto-correlation of a single pulse for a time shift of $\tau = 0$ is shown below and as a function of time shift in Figure 7 .

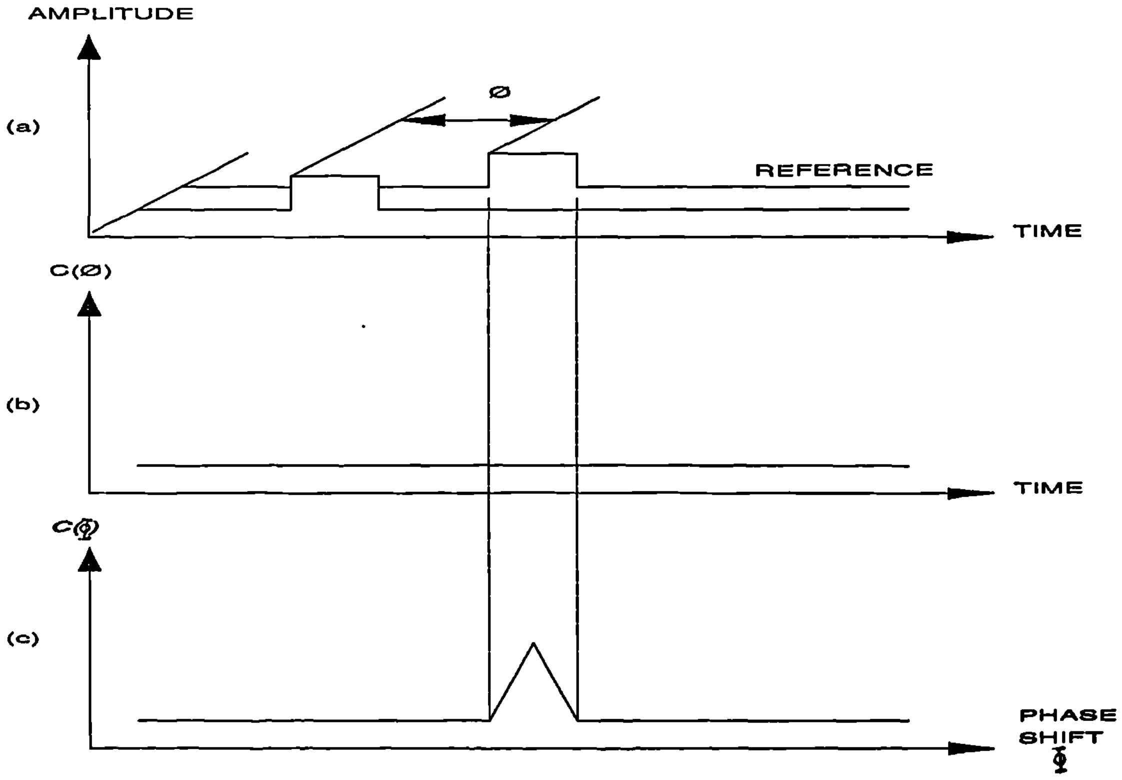


Figure 7 : Autocorrelation Function

As can be seen, the correlation value $C(\tau)$ is a measure of how much agreement there is between the signal and its replica. The peak value is attained when the signal and its replica have zero phase difference, or $\tau = 0$.

3.4.2 Criteria of Randomness

First, we shall see how the criteria of randomness can be extended to discrete sequences as well as continuous signals.

For a discrete random variable n(a,b), where p(a)=p(b)=1/2 the following conditions apply:

$$\underset{n \to \infty}{Lt} \frac{number of times a occurs}{number of times b occurs} = \frac{p(a)}{p(b)} = 1$$

and:

$$\underset{n \to \infty}{Lt} \frac{v(i)}{v(i+1)} = [p(a)^{i} p(b)]\ [p(a)^{i+1} p(b)] = p(a)^{-1} = 2$$

Where v(i) is the number of runs of i a's flanked by b, e.g.: baaaaa....aaaaab.

Suppose 1 is assigned to a and -1 to b, so any pair of tuples (1,1), (1,-1), (-1,1), (-1,-1) may occur with equal likelihood. If the product of two terms in this sequence [y]; $y_r \times y_{r+h}$ is formed, where h is a fixed integer, the auto-correlation is given by:

$$C(h) = \underset{j \to \infty}{Lt} \frac{1}{(2j-1)} \sum_{r=-j}^{r=+j} y_r y_{r+h} = \begin{matrix} 0 if h \neq 0 \\ 1 if h = 0 \end{matrix}$$

The function C(h) has a two level auto-correlation.

If we regard the sequence of 1's and -1's as a waveform y(t), the auto-correlation of y(t) is

$$\Phi(t) = \underset{T \to \infty}{Lt} \frac{2}{T} \int y(t)y(t+T)dt$$

If the clock period is δ, then $\Phi(t)$ is as shown below in Figure 8

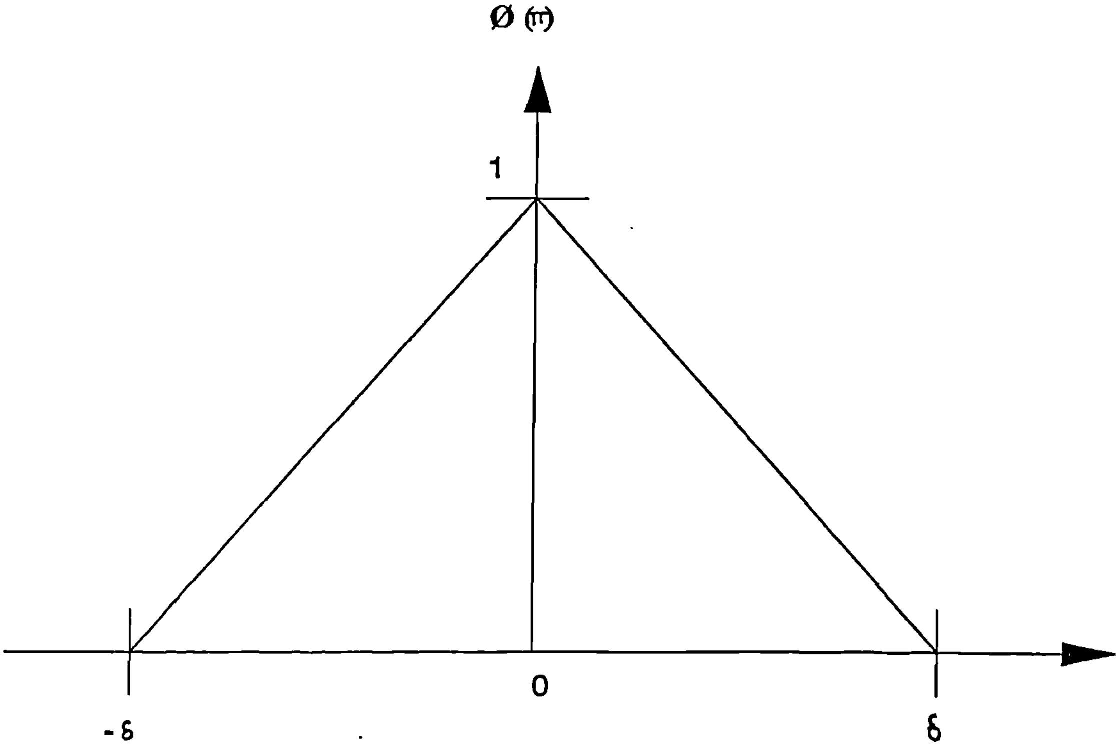


Figure 8 : Correlation Function

For the period $0 < \pi < \delta$:

$y(t)y(t+\pi)$ is +1 in the intervals A1A2, A3A4 and is equally likely to be +1 as -1 in the intervals A2A3, A4A5. Therefore the function $\phi(t)$ which is an average of the above product, equals $\frac{\delta-\pi}{\delta} = 1 - \frac{\pi}{\delta}$.

For the period $-\delta < \pi < 0$:

$\phi(t) = 1 + \frac{\pi}{\delta}$, so $\phi(t) = 1 - \frac{|\pi|}{\delta}$ for $|\, \pi \,| < \delta$. For $|\, \pi \,| > \delta$ there are no intervals of t over which the product is always +1. It is as likely to be +1 as -1 for all t, so $\phi(t) = 0$ for $|\, \pi \,| > \delta$. Since $\phi(t)$ is an average, its value does not depend on the position of the time axis and the autocorrelation function C is shown below in Figure 9 .

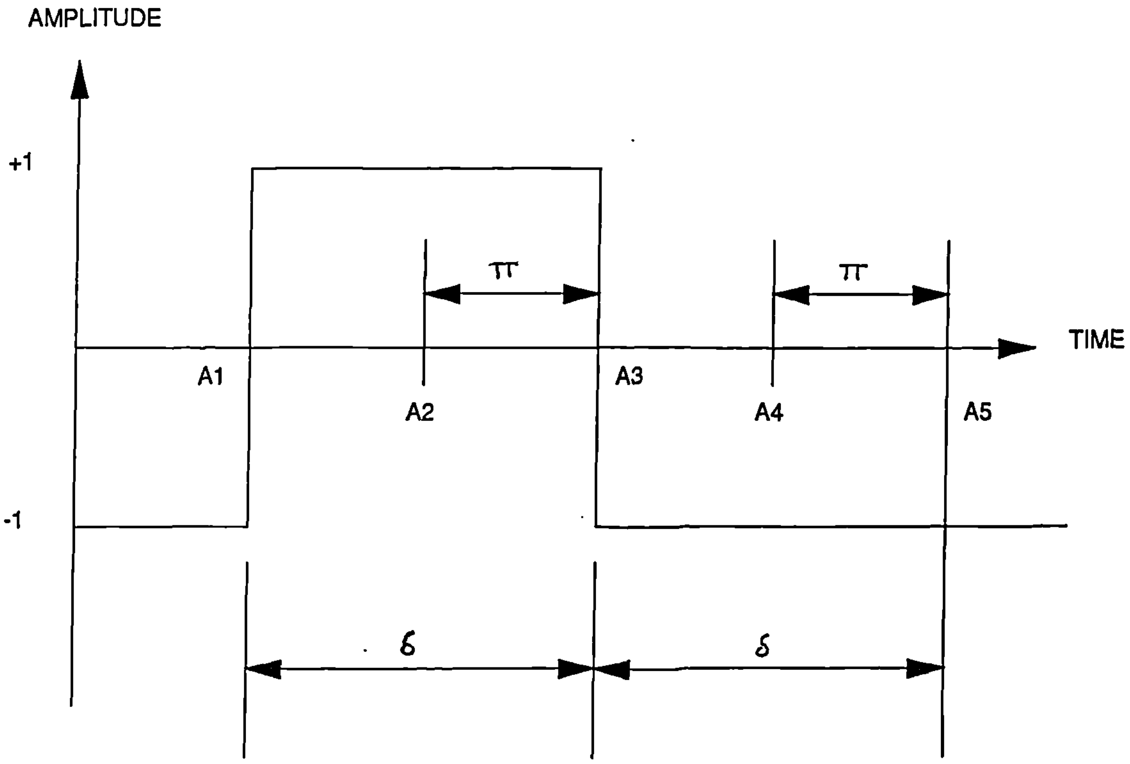


Figure 9 : Autocorrelation Function

If we consider pure white noise, i.e. a signal with all frequencies present and of the same mean amplitude, the instantaneous amplitude of the noise can change suddenly at any time, so the autocorrelation function will be zero for all phase shifts, except when the phase shift is zero, when it will be equal to the mean square value of the noise amplitude. This leads to one of the necessary conditions for a signal or sequence to be classed as noise-like: its auto-correlation must have a large value for zero phase shift, and a constant small value elsewhere.

The sequences used in spread spectrum communications are all based on a single type of ideal sequence called the m-sequence. A characteristic of this type of sequence is that it can be generated by a shift register using feedback taps, with the input to the shift register derived from a linear combination of the feedback taps. This type of sequence is not truly random, since the state of the register can be predicted given the starting value of the register, the feedback taps and the number of clock periods.

One of the properties of an m sequence generated by a shift register of length n is that as the generator cycles through its states, all the numbers between 1 and 2n-1 occur exactly once as n-tuples in the shift register. This leads to a further property: the number of ones in the m sequence is 2^{n-1} and the number of zeros is $2^{n-1}-1$, i.e. the difference between the number of ones and zeros is always 1 for an m sequence.

Except for the run of n ones and n-1 zeros the m sequence is identical to a truly random sequence n(a,b). To find out what effect this has on the noise properties of m sequences, consider a further property of an m sequence:

For any m sequence, a number m can be found so that

$$y_{r+m} = y_r \oplus y_{r+k}$$

where $\oplus$ is the linear addition operator

for all values of k except multiples of n.

For a periodic sequence of period r:

$$C(h) = \frac{1}{r} \sum_{r=1}^{r=n} y_r y_{r+h}$$

For $h \neq 0$ (mod n):

$$C(h) = \frac{1}{n} \sum_{r=1}^{r=n} y_r y_{r+l} = -n^{-1}$$

For h=0 (mod v):

$$C(h) = \frac{1}{n} \sum_{r=1}^{r=n} 1 = 1$$

Thus the m sequence has an auto-correlation value of 1 when in phase, and $-n^{-1}$ when out of phase. As n increases, the effect of the sequence length becomes less and the auto correlation approaches that of pure noise.

Periodic sequences having a two-level auto-correlation are called perfect sequences. If, in addition to this the number of ones and zeros differs by at most 1, then they are termed pseudo-random sequences, m sequences being an example. The m sequence is also the only type of sequence for which the number of runs of length i is twice the number of runs of length i+1 (c.f.).

3.4.3 Cross-Correlation of noise sequences

For two sequences of relatively prime lengths r and y, the cross correlation value must be constant since all possible relative phase shifts occur. They are statistically independent, i.e.

$$p(x_r = a, y_r = b) = p(x_r = a) \times p(y_r = b)$$

and the cross correlation value is v^{-1} where v is the product of the lengths of the two sequences. For m sequences of equal period, the cross correlation value is given below:

$$C_{xy}(h) = \frac{1}{n} \sum_{r=1}^{r=n} x_r y_{r+h}$$

The cross correlation takes on different values depending on the value of h. There is no formula for calculating the number of different cross correlations for a pair of sequences, but this number cannot exceed the number of m sequences of period n. $C_{xy}(h)$ depends on the coset to which h belongs, and so cannot have more values than cosets, i.e. m sequences.

3.5 Types of binary sequence

The subject of sequence generation is linked to that of number theory in mathematics with many of the theorems and results of the latter being applicable to the former. In considering the generation of maximal length linear sequences the theorems of Mobius and Euler are essential tools (see APPENDIX D).

3.5.1 Gold Codes

A set of Gold sequences of period $2^n - 1$ contains N + 2 sequences with $\theta_{max} = 1 + 2^{n+2/2}$.

Given a preferred pair of m-sequences of length $2^n - 1$ with peak periodic crosscorrelation M, it is possible to construct a set of N + 2 sequences with peak out of phase autocorrelation and peak crosscorrelation equal to M. This is done by taking the exclusive-or of the outputs of the two m-sequence generators. Different relative phase shifts between the two m-sequences will produce different Gold codes from the same set. Gold codes are optimal codes, because they satisfy the lower limit on correlation when the sequence length is related to an even power of 2 proved by Sideln'ikov [20] which states that for any set of N sequences, $\theta_{max} = \max\{\theta_c, \theta_a\} > \sqrt{2N-2}$

3.5.2 Small and Large Kasami Set

If a sequence and a cyclic shift of itself are combined when the exponent of the length is a power of 4, then the sequence set is called a small Kasami set, denoted $K_s(u)$ where the original sequence is denoted by u. Gold code sets and small Kasami sets are complimentary. Gold codes only exist when their length is related to an odd power of 2, small Kasami sets only exist when their length is related to a fourth power of 2.

When a preferred triple of codes is combined to generate a sequence, the set is denoted a large Kasami set $K_L(u)$. The set of large Kasami sequences contains as a subset the Gold code set of the same length and also the small Kasami set of the same length. The number of codes in the set is equal to $2^{n/2}(2^n + 1)$

3.5.3 Two Level Sequences

If $[\{y_r\}]$ is a binary sequence of period v with a +1's and b -1's, then a+b = v. In this case the auto correlation is given by

$$vC(h) = \sum_{r=1}^{r=v} y_r y_{r+h}$$

For h = 0,

$$vC(0) = \sum_{r=1}^{v} y_r^2 = v$$

$$C(0) = \frac{v}{v} = 1$$

If we form the sum $\sum_{h=1}^{v} vC(h)$ in two ways: First:

$$\sum_{h=1}^{v} vC(h) = \sum_{h=1}^{v}\sum_{r=1}^{v} y_r y_{r+h}$$

$$= \sum_{r=1}^{v}\sum_{h=1}^{v} y_r y_{r+h}$$

$$= \sum_{r=1}^{v} y_r \sum_{h=1}^{v} y_{r+h}$$

Since y_r is periodic, both these sums are equal to a-b, so

$$\sum_{h=1}^{v} vC(h) = (a-b)^2$$

Second:

$$\sum_{h=1}^{v} vC(h) = v \times C(1) + \ldots + C(v-1) + vC(v)$$

$$= v \times C(1) + \ldots + C(v-1) + v$$

since $c(v) = c(0)$. If the sequence is two level, the auto correlation for $h \neq 0$ is constant.

So

$$\sum_{h=1}^{v} vC(h) = k \times (v-1) + v$$

Since a+b=v, a-b=2a-v, so for (2a - v) 2 = w(v-1) + v to be satisfied by integers is a necessary condition for a sequence to be two level, i.e.

$$w = \frac{(2a - v)^2 - v}{v - 1}$$

If the difference between the number of 1's and -1's is one then w = -1 and the sequence is pseudo random.

3.5.4 Linear Feedback Shift Registers

The general linear feedback shift register consists of a number of registers in cascade. The input to the first store is the modulo-two sum of the output of each of the other stores, the last stores' output being known as the generator output.

A recurrence relation describing the operation of a linear feedback shift register will be designated by the position of the feedback taps. For instance, the recurrence relation

$$y^r \times (c_1 y^{r-1}(c_2 y^{r-2}(\ldots\ldots.c_{n-1} y^{r-n+1}(y^{r-n})))) = 0$$

$$1(c_1 D(c_2 D^2(\ldots..c_{n-1} D^{n-1}(D^n))))y^r = 0$$

will be referred to as the equation

$$(0, c_1, 2c_2, \ldots, (n-1)c_{n-1}, n)$$

Recently, much attention has been paid to sequences generated by shift registers with non linear feedback, because of advantages non linear sequences have from the point of view of cross and auto correlation. Non linear sequences can be classified by means of their effective linear span (ELS).

Any sequence (linear or non linear) of length n can be considered as part of a (possibly) longer linear m sequence, the length of the shortest such m sequence being the effective linear span of the original sequence.

For a linear m sequence generated by a shift register of length r, all numbers from 1 to r occur as tuples of the shift register in the course of one cycle of the m sequence. Because of this, there will be an m sequence of length 2^{r-1} which will include in it the original sequence. In the extreme, if the non-linear sequence consists of all 1's then the minimum value of r for the condition above to be satisfied will be n and the effective linear span will be 2^{n-1}, because this particular sequence is identical to the all 1's tuple of an m sequence of length 2^{n-1}. This classification gives a measure of the performance of the non linear code, based on the analysis of the performance of linear m sequences.

The investigation of sequences that have good correlation parameters (as discussed in APPENDIX A) has been given much attention in recent years, as shown by the number of papers published. The detailed discussion of the many type of sequences is not included here, but a brief overview of the types is included. The first category is composite sequences, typified by the well known Gold Codes [21] [22] which are linear combinations of m-sequences. One of their most important properties is that there is a much larger number of Gold codes of a given length than there are m-sequences. The autocorrelation and cross-correlation figures are also well defined in terms of their minimum and maximum values. The number of usable sequences of a given length is greater than for m-sequences, and their behaviour under non-ideal conditions is better known. The difference in phase between the two m-sequences define the Gold code to be generated, with different phases producing different codes.

In general, as a logical extension to the principle of Gold codes more than two m-sequences (or any other linear sequence) could be combined to give a set of sequences with any one of a number of parameters maximised for that set. All these types of code can be grouped under the heading combinational sequences.

The second group of non-maximal length codes is the set of non-linear codes, typified by the Bent [23] [24] code set. Many of the types of non-linear code have been designed with a particular performance requirement in mind, for instance a long linear span for good security, or a low cross correlation for good interference rejection.

3.6 Correlation Measurements

When considering the transmission of binary data modulated by a spread spectrum signal the autocorrelation and crosscorrelation of each sequence from the set is measured and used to provide a figure of merit or threshold for successful reception of data. Assuming that the sequence length is fixed at some value, and that each bit of data is modulated by a fixed integer number of spread spectrum sequences then two additional occurrences have to be considered. These are referred to as the aperiodic and odd cross and auto correlation [25] measures. Without loss of generality, assume that a data bit is modulated by a single period of the spreading sequence. Then eight possible combinations of spreading sequence are possible for the case of first order correlation. These are illustrated below in Figure 10 .

In all cases, a 1 refers to the spreading sequence and a 0 refers to the spreading sequence inverted. Two different cases have to be explained, first when the received sequence and transmitted sequence are the same, i.e. the receiver is expected to correctly interpret the value of the data bit; second when the received and transmitted spreading sequence are different and the receiver is expected to decode the data as being not valid. Each of the four combinations are explained below:

Combination 1, same sequence.

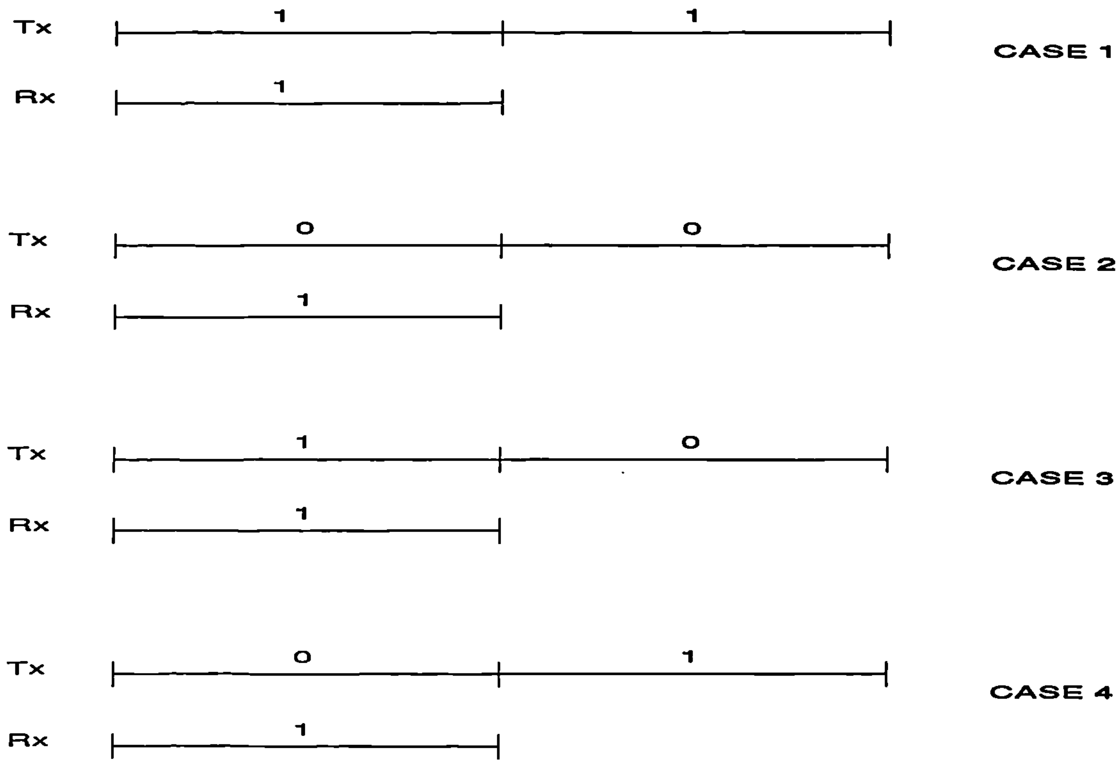


Figure 10 : Bit Combinations

The first case shows what happens when two consecutive 1 data bits are sent. The receiver is required to correctly discriminate the existence of two consecutive 1's by producing two correlation peaks at its output. This is interpreted as a correlation value near zero for all phase shifts except in-phase when the correlation should be high.

Combination 1, different sequences.

The receiver has to produce an output which indicates that no valid data is available on the network. This is normally interpreted as a correlation value near zero for all phase shifts.

Combination 2.

In this case, the in-phase autocorrelation peaks will be large and negative, but the autocorrelation for other phase shifts will be near zero. The cross correlation will be near zero.

Combination 3

In this case, the autocorrelation will be large and positive for zero phase shift, large and negative for 0+l phase shift and near zero for all other phase shifts. The crosscorrelation will again be close to zero.

Combination 4

This is similar to combination 3, except that the position of the positive and negative peaks is reversed.

All these combinations can be incorporated in a simple nomenclature. For the case of data bits 1 followed by 1, and 0 followed by 0, the correlation is designated by <1,11>. The goal of good code set design is to maximise the zero phase shift autocorrelation to either a large negative or positive value while minimising the non-zero phase shift autocorrelation and all values of the crosscorrelation. For the case of data bits 1 followed by 0 or 0 followed by 1 the correlation is designated <1,10> and similar goals apply. The goal for good code set design is very simple: to maximise the probability of detecting a valid spreading sequence. This is achieved by maximising the in phase autocorrelation and minimising all other correlations.

This has important implications for the design of code sets, because the two correlations <1,11> and <1,10> can be measured by cyclic convolutions using the fast fourier transform. If the codes are of length l, the <1,11> requires a length l convolution and <1,10> requires a length 21 convolution. Finding the index of discrimination, or correlation threshold is done by looking for the largest correlation value for all combinations, while ignoring the zero phase autocorrelation which will always be equal to +-1.

3.7 Higher Order Correlations

In all the open literature the performance of spread spectrum codes is measured by the auto and cross correlation of code pairs [26]. These measures are inherently incapable of measuring the performance of a multiple access system however, because in a real system multiple code interactions take place. A receiver rarely if ever has to decode a data bit in isolation. There are normally other exchanges of data between independent nodes in the network. This shortcoming of existing code design has not been addressed specifically, except in the measurement of average bit error rates for systems which show the effect of these multiple interactions. It is possible to include their effects in the measurement of the code set performance as described above by correlating a sequence with combinations of sequences from the code set.

The method of combination depends on the signalling method used. A non-coherent optical fibre network acts as an or-channel, while a differential voltage coaxial cable system acts as an adder and a coherent optical fibre system acts as an exclusive-or channel. Each of these networks will have different optimum code sets given a set code length and code set size.

The important parameters in determining a code set for a network are therefore:

Its code length, equal to the signalling rate divided by the data rate

The code set size, equal to the signalling rate divided by the data rate

The physical nature of the network medium

A program for designing code sets for use in spread spectrum systems (not just local area network ones) should use these and only these parameters. Traditional design methods assume that good first order correlation values will yield good higher order correlation values, and that the network designer is free to choose the physical characteristics of the network.

Traditional design methods also nearly always assume that codes are not of arbitrary length, and typically are of length $2^n - 1$ where n is some integer. This is because the analysis of these sequences in terms of their correlation performance is a tractable problem for linear fields, and for certain well defined non-linear operations. There are numerous review papers giving examples of performance measures for such sequences [27].

As was pointed out above, the sequence length should be a set by the requirements of the application, and not by the mathematical tractability of the correlation bounds. Similarly, the code set size should be determined by the network throughput and desired bit error rate.

CHAPTER 4
SIMULATOR

4.1 Introduction

Simulation covers a large number of methods for measuring the behaviour of systems within an environment. The important distinction between simulation of a system and its real world counterpart is the fact that the simulation relies on artificially provided boundaries which define where the system ends and where its environment starts. The environment normally represents aspects of the simulation that are not considered, or are assumed to be known quantities in their effect upon the system. One of the most important parameters in writing a simulator is therefore the desired accuracy of the programme. It is possible to develop a good model of a system without the programme knowing about the limitations of the description it uses. In the light of this fact, a definition of simulation could be: **The process of taking a set of descriptions of low level objects which are assumed to obey the rules provided for them and through manipulation of a number of these objects produce a model of what effects will be observed of the system as a whole.**

There are, within this definition, a whole range of types of simulation techniques available. Many of these relate to the statistical interpretation of the results, others to the manner in which the dependent variable (normally time) is progressed. In the latter case, two main types of simulation technique have evolved in response to the different needs of two types of problem:

The need to simulate the functional behaviour of a system, in terms of functional descriptions of its constituent parts.

Simulation of the parametric behaviour of a system in terms of parametric description of its constituent parts.

The former need has given rise to the so-called event driven simulators, while the latter has spawned the discrete time family of simulators. Both techniques are powerful, and it is only recently that simulators which provide a functional description in terms of parametric behaviour of components have appeared. They are referred to as mixed-mode simulators and have mainly been used in the VLSI design field.

The work contained in this thesis relates to the functional properties of a new type of local area network, and it was for this reason that an event driven simulator was chosen to examine the properties of codes used to communicate across the network. The computer used for the task was a machine running the Unix operating system. One of the most powerful features of the unix system is its support for a programming language called C. This is a very flexible language developed for writing large pieces of software to run quickly, and the simulator was therefore written in C.

4.2 Operation of the Network

The operation of the simulator is intended to reflect the operation of the demonstrator network in all important ways. This is reinforced by the point made earlier in the chapter; that one of the keys to successful simulation is knowing what to include and what to leave out of the simulation. In the case of the spread spectrum network the physical hardware operates on direct modulation of the network cabling by a polarised voltage. This is unlike other methods mentioned in CHAPTER 3, for instance phase shift keying and frequency shift keying. The reason for this choice of modulation method was that it allowed a very simple transmitter and receiver circuit to be used. This was felt to be of great importance considering the intended application.

Another point that was made in CHAPTER 3 was that possible use of fibre optic techniques was investigated. The reason for this is the noise immunity and electromagnetic interference rejection capabilities of fibre optics. A study of the particular coding and communications problems likely to be encountered was not undertaken however, because there is much groundwork still to be done in this area. The work of Davies et al. [28], [29], [30], at Kent University is of great interest in this area though.

4.3 Simulator design strategy

One of the objectives of simulating the prototype spread spectrum local area network was to provide a detailed view of the correlation parameters, as discussed in appendix A, in a real network. The problem with the calculations outlined in appendix A is that they are not easily applied to real situations, and do not even produce particularly accurate results, even for the simplest and most well known type of sequence - the m sequence.

The choice of an event driven simulator to model the system, and the use of the programming language C to write the code was driven by the tools available at the time. The choice of an event driven simulator meant that a very large amount of data would be generated and for this reason the simulator was coded so that no long term storage of results within the environment of the programme was necessary. All results are disposed of immediately to the output file, with only intermediate results of individual correlations being kept within variables in the programme.

4.4 Program Structure

The basic data type in the simulator is called an event. It has attributes of time and amplitude. The time attribute denotes the time at which the event is to occur and the amplitude attribute its amplitude. Events are stored in an ordered linked list (see Figure

11 below). As more events are generated during the course of the simulation they are inserted at the appropriate point on the linked list. Each event points to another data type called a gen denoting a generator. For each transmitted sequence in the simulation there is an instance of gen. The linked list of events contain references to the generators that caused them.

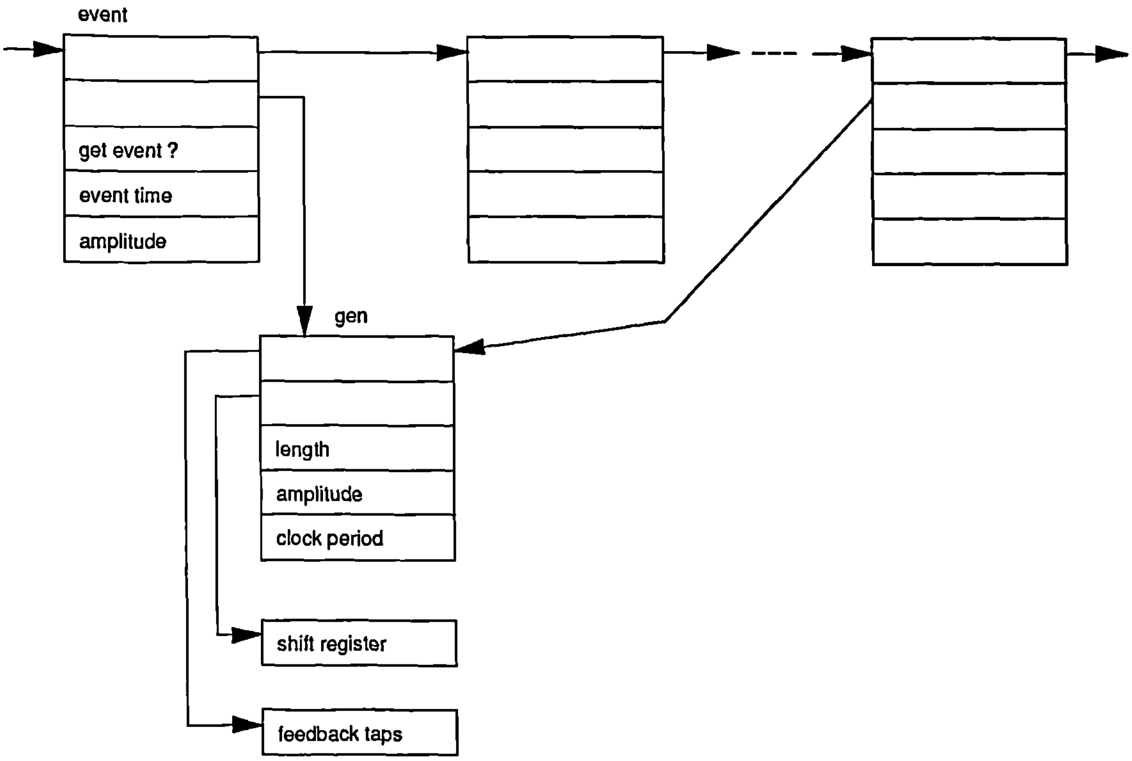


Figure 11 : Linked List of Events

The operation of the simulator, once initialised, is as follows. The list of events is searched for the event with the lowest time. This event is evaluated, causing the instantaneous voltage on the network to change. The generator which caused the event is evaluated, causing another event one clock period later. The linked list of events is then searched for the correct place to put this new event, and it is inserted there. The original event is then removed from the list and its storage space freed.

The whole exercise of generating events is only performed on demand by the part of the program which models the receiver. This is explained with reference to Figure 12 below.

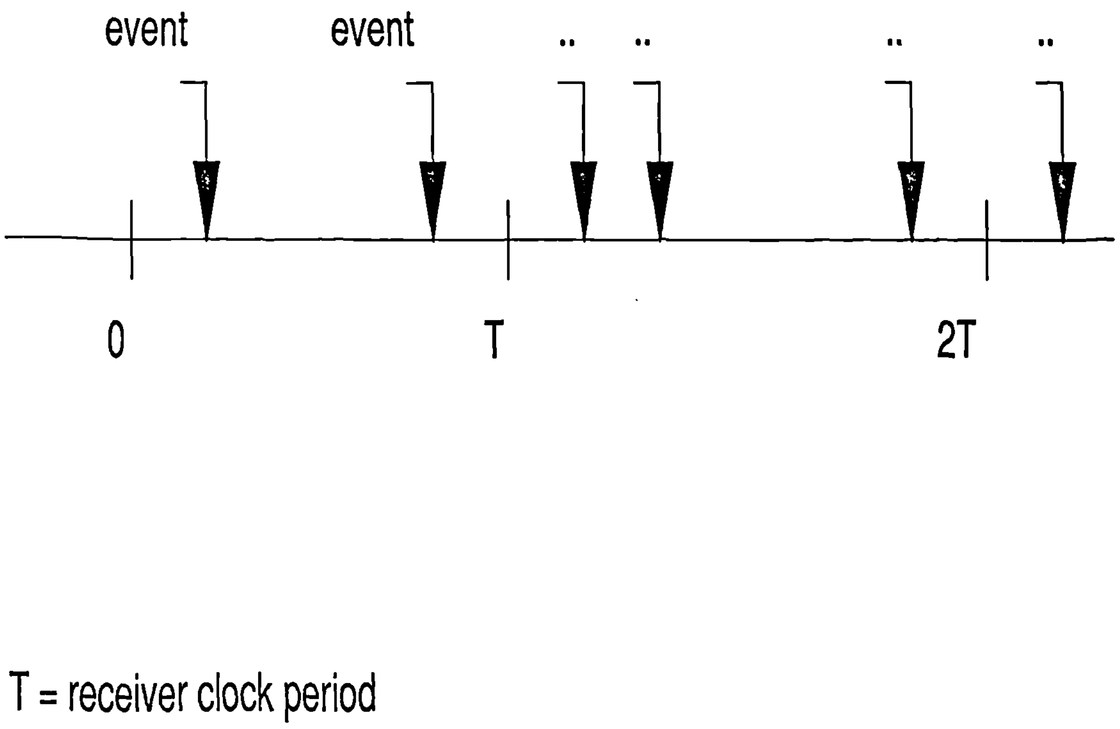


Figure 12 : Event Generation

The receiver looks ahead for one receiver clock period for any events which will change the network during that time. If it finds any they are evaluated as described above. Because evaluating an event always causes its successor to be created, the simulation will never stop. For this reason the receiver always checks to see if it has exceeded the maximum simulation time. The method used to calculate whether the received bit is positive or negative can be varied. In this version of the simulator there are two options, either to integrate or to sample. See Figure 13 below.

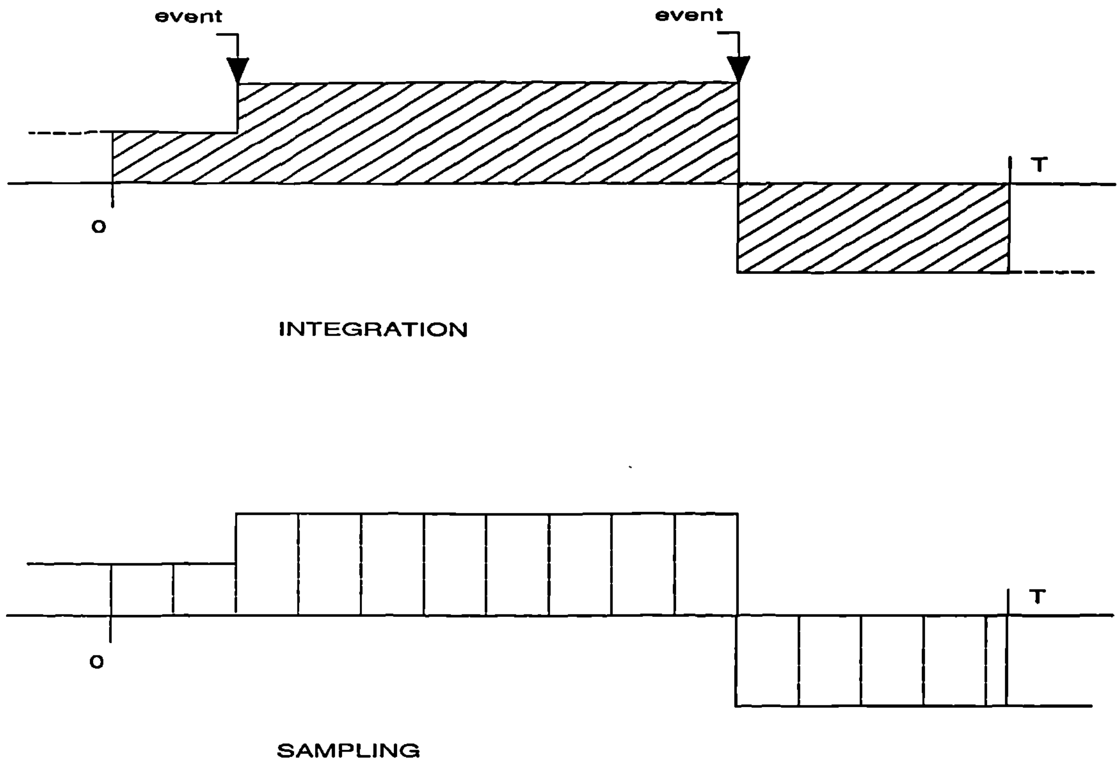


Figure 13 : Integration and Sampling in the Receiver

In integration mode the chip energy is measured by integrating the voltage over one period of the receivers clock. This corresponds to a perfect receiver. Alternatively the receiver can sample the voltage at a pre-determined number of points in its clock cycle, and use majority voting on the result. If one simulates with a single sample then the receiver is hard limited.

4.5 Results

Two facts emerged from the work done with the simulator. Both of these are of particular importance to a local area network where the number of information exchanges must be maximised and the use of a system wide clock synchronisation is not allowed. By using spread spectrum sequences to directly modulate data it was found that the number of simultaneous exchanges of data within a given bandwidth is much less than that predicted by theory, especially if the sequences are not synchronous. The second important result is that the effects of non-synchronous clocks at receiver and transmitter can be beneficial or a hindrance. Clock frequency difference can be compared with a Doppler shift of constant frequency, and the analogy with moving transmitters and receivers can be carried further to the point where a real network can be visualised as an area of space with each node moving at a constant velocity.

The net effect of the Doppler effect is that if it were possible to ensure that all nodes had a minimum and maximum relative frequency then the overall performance would be improved. In practice this is not possible because the desired spread in clock frequencies would prohibit the use of cost effective clocks. In a physical system this would correspond to all the nodes travelling away from each other at constant velocity, i.e. an "expanding universe" of nodes.

The maximum number of information exchanges allowable on a given network with a given length of code sequence is a very important parameter in local area network design. In a local area network with nodes producing data at rates of up to 1 MBps code sequences longer than 9 bits are at present unrealistic, because of the limitations of electronic circuits at high frequencies. GigaHertz sequence generators do exist, but their use in a local area network will have to wait until the availability of cheap, multi-drop fiber-optic networks. Using the equation discussed in appendix B, the number of simultaneous exchanges available is equal to the number of m-sequences of a particular length, and this is maximised when the sequence length is prime. This occurs for 127 and 8191 bit long sequences, corresponding to shift register lengths of 7 and 13 bits. These sequences are referred to as Mersenne Prime sequences.

The possibility that certain combinations of m-sequences would not be capable of extracting accurate synchronisation information was also made clear, and one example of this is shown in Figure 14 below.

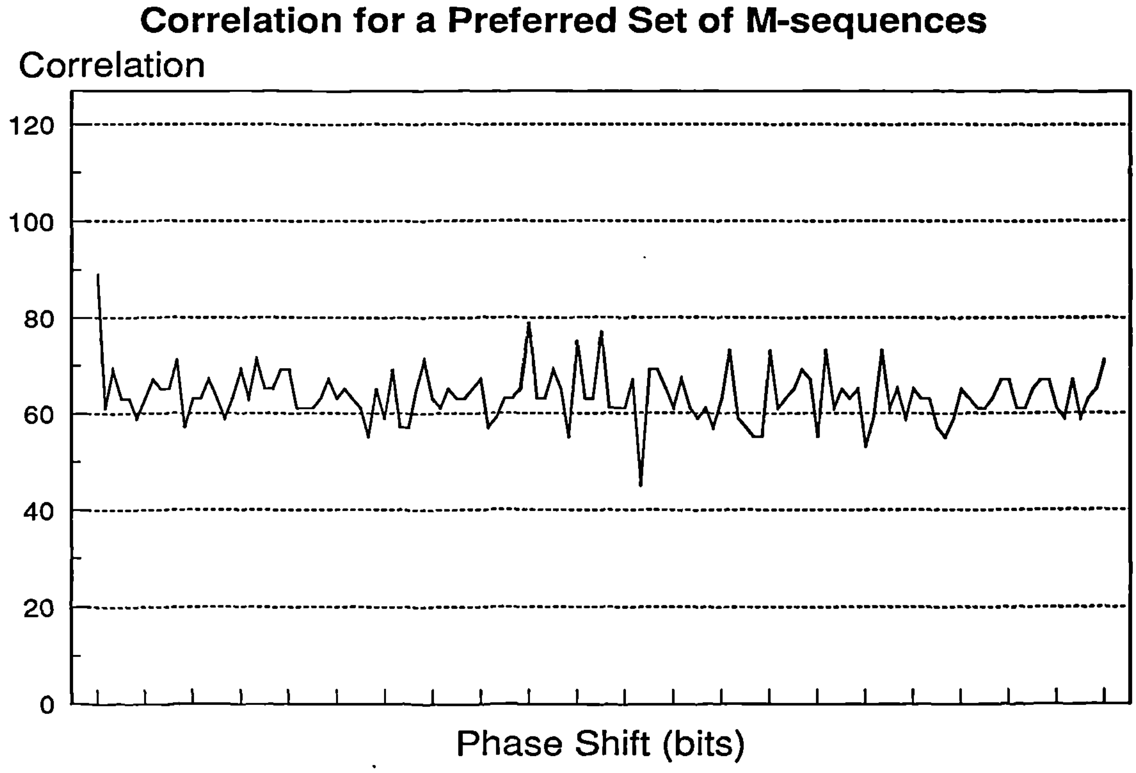


Figure 14 : Crosscorrelation of M Sequences

This shows the correlation function for a transmitter set of all 127 bit long m-sequences, correlated with the 127 bit m-sequence generated by the shift register with taps at bit positions 7, 4, 3, and 2. A value of correlation that would provide accurate reception with one sequence would not with another, it would merely result in a large number of false correlation peaks.

4.6 Conclusion

The simulations indicate that even the performance of a so called well known family of codes (m-sequences) is not fully understood.

Other types of deterministic sequence are better, for instance Gold codes, Kasami codes and Bent sequences, but these have restricted set of code lengths.

CHAPTER 5
SIMULATED ANNEALING

5.1 Genetic Algorithms and Simulated Annealing

Simulated Annealing is a stochastic computational technique derived from statistical mechanics for finding globally minimum cost solutions to large optimisation problems. Kirkpatrick [31] was the first to propose and demonstrate the application of simulation techniques from statistical physics to problems of combinatorial optimisation, specifically to the problems of wire routing and component placement in VLSI design. More recently other workers have applied Simulated Annealing to the design of source and channel codes [32]

In general, finding the global minimum value of an objective function with many degrees of freedom subject to conflicting constraints is an NP complete problem, since the objective function will tend to have many local minima. A procedure for solving hard optimisation problems should sample values of the objective function in such a way as to have a high probability of finding a near optimum solution and should also lend itself to efficient implementation. Over the past few years simulated annealing has emerged as a viable technique which meets the criteria.

To appreciate the relationship between techniques in statistical physics and the solution of large optimisation problems one must understand the basics of statistical mechanics. Statistical mechanics is the study of the behaviour of very large systems of interacting components, such as atoms in a fluid in thermal equilibrium at a finite temperature.

Suppose that the configuration of the system is identified with the set of spatial positions of the components. If the system is in thermal equilibrium at a temperature T then the probability $\pi_T(s)$ that the system is in a given configuration s depends on the energy E(s) of the configuration and follows the Boltzmann distribution:

$$\pi_T(s) = \frac{e^{\frac{-E(s)}{kT}}}{\sum_{\omega \in s} e^{\frac{-E(\omega)}{kT}}}$$

where k is Boltzmann's constant and s is the set of all possible configurations.

One can simulate the behaviour of a system of particles in thermal equilibrium at temperature T using a stochastic relaxation technique developed by Metropolis [33].

Suppose that at time t the system is in configuration q. A candidate r for the configuration at time t+1 is generated randomly. The criterion for selecting or rejecting configuration r depends on the difference between the energies of configurations r and q. Specifically the ratio of the probability of being in r and the probability of being in q is computed:

$$p = \frac{\pi_T(r)}{\pi_T(q)} = e^{\frac{-(E(r)-E(q))}{kT}} .$$

If p > 1 the energy of r is less than the energy of q then configuration r is automatically accepted as the new configuration for time t+1. If p < 1 the energy of r is greater than the energy of q then configuration r is accepted as the new configuration with probability p. Thus configurations of higher energy can be reached.

It can be shown that as $t \rightarrow \infty$ the probability that the system is in a given configuration s is $\pi_T(s)$ regardless of the starting configuration. The distribution of configurations therefore converges to the Boltzmann distribution [31]

In studying such systems of particles the nature of the low energy states is important. The system might be in a crystalline or glassy state. Very low energy states are not common when considering the set of all configurations. However, at low temperatures they predominate because of the nature of the Boltzmann distribution. To achieve low energy configurations it is not sufficient to simply lower the temperature. One must use an annealing process where the temperature of the system is elevated and then gradually lowered, spending enough time at each temperature to reach thermal equilibrium. If insufficient time is spent at each temperature then the probability of reaching a very low energy configuration is greatly reduced.

The simulation of annealing applied to optimisation problems involves the following steps. First, the analogue of physical parameters of energy and temperature must be found. The energy becomes the objective function, the particle configurations become the parameter values and finding a low energy configuration becomes seeking a near optimum solution, and temperature becomes the control parameter. Second, an annealing schedule consisting of decreasing temperatures and time to be spent at each temperature must be decided. Third, a method for generating and selecting new configurations must be chosen.

The annealing algorithm proposed by Kirkpatrick consists of running the Metropolis algorithm at each temperature in the annealing schedule for the amount of time prescribed by the schedule, and selecting the final configuration generated as a near optimum solution. The Metropolis algorithm, which accepts configurations that increase the cost as well as those that decrease the cost, is the mechanism for avoiding entrapment at a local minimum.

The annealing process is inherently slow. Geman and Geman [34] determined an annealing schedule sufficient for convergence. Specifically, for a given sequence of temperatures

$\{T_t\}$ such that $T_t \to 0$ as $t \to \infty$ and $T_t \geq \frac{c}{\log t}$ for a large constant c, then the probability that the system is in configuration s as $t \to \infty$ is equal to $\pi_0(s)$. Others have also improved this bound [35]

In practice it is often unnecessary to adhere to this conservative schedule in order to achieve acceptable results. Many applications of simulated annealing map naturally into a parallel processing implementation and hence the speed of execution can be substantially increased. Determining the proper annealing schedule for a given problem is frequently a matter of trial and error. Davis and Ritter [36] have used a genetic algorithm to determine the best annealing schedule.

5.2 Simulated Annealing for Code Set Design

In the case of spectrum code set design, the design parameters are the code length and the code set size. The performance measure is the correlation threshold for the first or higher order correlations. Assume it is the first order correlation threshold. This measures the in-phase autocorrelation against the out of phase auto correlation and cross correlation for the sets <1,11> and <1,10>. The in-phase autocorrelation is always fixed as +- code length, so the out of phase autocorrelation and crosscorrelation measure the performance. It is desired to minimise this value for a given code set in order to maximise the code set performance. Thus, this value is the objective function or energy in the annealing schedule.

The temperature parameter determines the amount of scrambling that is done to codes in the code set. A random variable dependent on the value of the temperature determines the number of bits that will be inverted in each code, with all codes in the set treated independently. After the codes have been scrambled the new configurations' energy is calculated using the cyclic convolution, and returned as the new energy value. If this is less than the previous energy then the new configuration is accepted, otherwise it is accepted with probability equal to $e^{\frac{-\delta E}{T}}$, so that a large increase in energy means a low probability of acceptance. This procedure is repeated many times for each temperature before the temperature is dropped, and when the temperature has reached a low enough value and the improvements in energy are insignificant the code set is accepted as the final configuration and its peak correlation parameters are determined for graphical output.

5.3 Program Design

The program that performs the simulated annealing is written in ANSI C. Two functions perform the annealing, others implement a fast correlation, output results and organise the overall flow of the program. The main function parses the command line, initialises data structures and then iterates throught the Metropolis algorithm until a stable configuration has been reached, or a maximum number of tries have been unsuccessful. The Metropolis algorithm applied to this problem consists of scrambling all the codes in the code set and then calculating their peak correlation. This is used as the energy which is to be minimised. The probability of being in the new configuration is calculated and it is either accepted or rejected based on a simple test. See Figure 14.A for details

5.3.1 Output of Results

The simulated annealing program produces three types of output. The codes that have been designed by the program are output to a file or to the screen. Second, diagnostic output showing the energy, temperature and number of bits scrambled can be output to show the annealing process at work. The third output consists of a pair of sequences which show the largest value of correlation at each phase shift for all code pairs in the code set. This provides an envelope of maximum correlation for visual verification of the results when used with a graph plotting routine. This format has been used in the results chapter.

The notation for graphs of peak correlation shows the code length and code set size. For instance, a code set of 20 codes, each of length 100 bits would be denoted {100,20}, i.e. {length, set size}.

5.3.2 Random Numbers

The scrambling of bits in the sequence and generation of random numbers for the probability test are all done using a very long period random number generator written especially for this application. It is based on three linear congruential generators. One is used for the most significant part of the output number, the second for the least significant part and the third to control a shuffling operation. A detailed description of the different operations is available in [37].

5.3.3 High Speed Correlation

If we denote the correlation between two sequences as C_{xy} then the discrete correlation theorem implies the Fourier transform pair

$$C_{xy} \Leftrightarrow X_k Y_k^*$$

This can be used to do very fast correlations. The two sequences to be compared are Fourier transformed, one transform is then mutiplied by the complex conjugate of the other, point by point. The resulting sequence is inverse transformed to produce the correlation sequence. The imaginary part of the transformed sequence will be zero because all the input samples were real, so this means that two simultaneous correlations of a given length can be done by one Fourier transform.

This method implements a circular correlation of the same length as the Fourier transform. The fastest implementation of a Fourier transform in C is using base 2 transforms, so the length of the circular correlation must be a power of 2. To implement correlations that are not a power of 2, the next highest power of 2 is used as the transform length and unused bits discarded after the transform. A simple Fourier transform program was written for this purpose, which exploits the symmetrical property of Fourier transforms to do two transforms at once [54]

The algorithm for calculating Y, the correlation parameter, is shown below:

```
SET Y to 0
FOR each sequence $a^i$
    FOR each sequence $a^j$
        calculate $C_{ij}$
        FOR each phase shift k
            IF $a^i = a^j$ and $k = 0$
                skip next steps
            IF $|\theta_{ij}(k)| > Y$
                $Y = |\theta_{ij}(k)|$
            IF $|\hat{\theta}_{ij}(k)| > Y$
                $Y = |\hat{\theta}_{ij}(k)|$
```

The Simulated Annealing Algorithm, which tries to reduce the correlation parameter, is also shown below. This *pseudo-code corresponds to the computer* program in Appendix H.

1. Given the code set size and code length, a set of random sequences is generated.

2. Y for this set is calculated.

3. A copy of the sequences is made, and randomly selected bits in the copy are inverted, the number of bits being a function of the temperature.

4. Y of the copy is calculated.

5. If Y for the copy is less than Y for the original, or Y for the copy is greater than Y for the original and the metropolis test is passed then the copy replaces the original and the temperature is lowered.

6. If the temperature has dropped to a preset level, then end.

7. Go to step 3.

Figure 14.A

CHAPTER 6

HARDWARE

This chapter will describe the design of the prototype Spread Spectrum network. The choice of a microprocessor to implement the control of the transceiver is justified on the basis of flexibility of design. The choice of the Intel 432 advanced microprocessor, as the first transceiver controller, is discussed later. Subsequent events prevented the use of the Intel microprocessor and so the Motorola 68000 32 bit microprocessor was used instead and its use as the controller is described. The demonstrator Spread Spectrum Local Area Network used several innovative hardware design techniques, and these are described in this chapter. The software that runs on each Node is also described in detail.

6.1 Objectives of the prototype system

As was mentioned in previous chapters, the use of direct sequence spread spectrum in local area networks is a novel idea. The type of sequence most suitable for this use is open to debate, but guidelines do exist in the form of desired upper and lower bounds for correlation parameters. One of the principal objectives of this work was to verify the theoretical work experimentally. This led to interesting results concerning the effects of clock drift between receiver and transmitter and the problems associated with hard limiting the signal. It was against this background that the decision to build a prototype network with limited performance was reached.

The specification for the production local area network included many facilities that were considered non-essential to the demonstration of the principles of operation of such a network. For example, many of the higher level protocols involved in code distribution and network management would only be necessary on a final design or as part of the testing of such protocols. The type of prototype to be built was therefore a simple set of receivers and transmitters, with the capability of being upgraded to provide a test-bed for the management functions when that part of the work was completed.

To recap, the design issues to be confronted were as listed below:

To find the best type of codes or code sets for use in this network.

The long term effects (compared to sequence duration) of loss of synchronism between receiver and transmitter clocks.

The impact that the use of a fiber optic based network would have on the design of the hardware.

The effect of using hard limiting techniques at the receiver, for simplicity.

In order to study these problems in detail, it was decided to proceed with the prototype in two stages: first, the construction of a single transmitter and receiver to investigate the problem of fiber optics and second, to build a multiple transmitter single receiver network using either fiber optics or cable technology to investigate the remaining three points.

6.2 Host Computer

The transmitter and receiver units for the network require a considerable amount of intelligence, so that code sequence changes, code sequence distribution, encryption and decryption and other higher level protocol activities could be performed at the critical place in the network, i.e. in the node unit. The choice of a microprocessor to perform this task was made because these electronic devices are being used increasingly for adaptability in similar applications.

One of the most important attributes of the control unit is the ability to manage more than one spread spectrum channel hardware unit at a time, which will be the case when a node has more than one transmitter or receiver. This requirement amounts to providing a multi-tasking facility in the node controller. At the time that this work was in progress, a promising new microprocessor was released by Intel, based on the work on Object Oriented Programming [38] that had been done in the U.S.A. and in the U.K. at the Royal Signals and Radar Establishment [39], [40], [41], [42], and resulted in the release of an object oriented microprocessor, the Intel iAPX432.

6.2.1 The Intel iAPX432

The issues of which computer to use as the controller for an access unit were to be based on the use of the Intel iAPX432 advanced microprocessor. Unfortunately the use of this processor was not feasible, because Intel stopped production of the 432. The alternative decided upon in this case was the Motorolla MC68000 (68K). The most important reason was the fact that the digital research group already had seven development boards based on the 68K, and also a disk based 68K computer. Avoiding the need to build or purchase new equipment on this scale was a major consideration. The interfacing requirements of the development board and the computer were slightly different, the former being based on a Motorolla standard called the Motorolla 68 K Development Module (K.D.M.) and the latter a fully asynchronous bus specification called V.M.E. The implications on the final design of having two types of interface are reduced to a minimum however, requiring only minor changes in the controller software to distinguish between the two.

6.2.2 The MC68000 microprocessor

This is an advanced design 32 bit microprocessor with fully asynchronous bus control based on the use of a common acknowledge signal called DTACK (not data transfer acknowledge). During normal operation of the microprocessor, devices connected to the bus such as memory, input output ports, floppy disks hard disks and the spreadnet controller, have to acknowledge reception of data written by the microprocessor or validity of data read by the microprocessor through DTACK.

6.3 Transmitter Design

There were several basic options for the transmitter design. These are listed below:

> Use a dedicated shift register with switchable feedback taps to allow different sequences to be generated.
>
> Use a counter to address a memory containing the value of each chip of the sequence.
>
> Use a memory in combination with a shift register to operate as a state machine.

In the first case, the type of sequence generated is limited to linear sequences. The delay between the start of a chip and the arrival of the next bit at the start of the shift register depends on the number of feedback taps used. At speeds approaching the limit of operability of the logic this is a crucial factor.

In the second case, sequences are limited to a fixed length (the modulus of the counter) unless some sort of reset capability is included in the counter. In this case there will be a delay, again limited by the speed of the logic, while the counter is reset.

In the third case, the delay between each chip is equal for all chips and is limited by the access time of the memory and the clock speed of the shift register. This design also allows variable length sequences to be generated, as well as non-linear sequences.

Based upon these considerations, the third method was chosen as the basis for the transmitter design. A simplified diagram of the sequence generation part of the transmitter is shown below in Figure 15

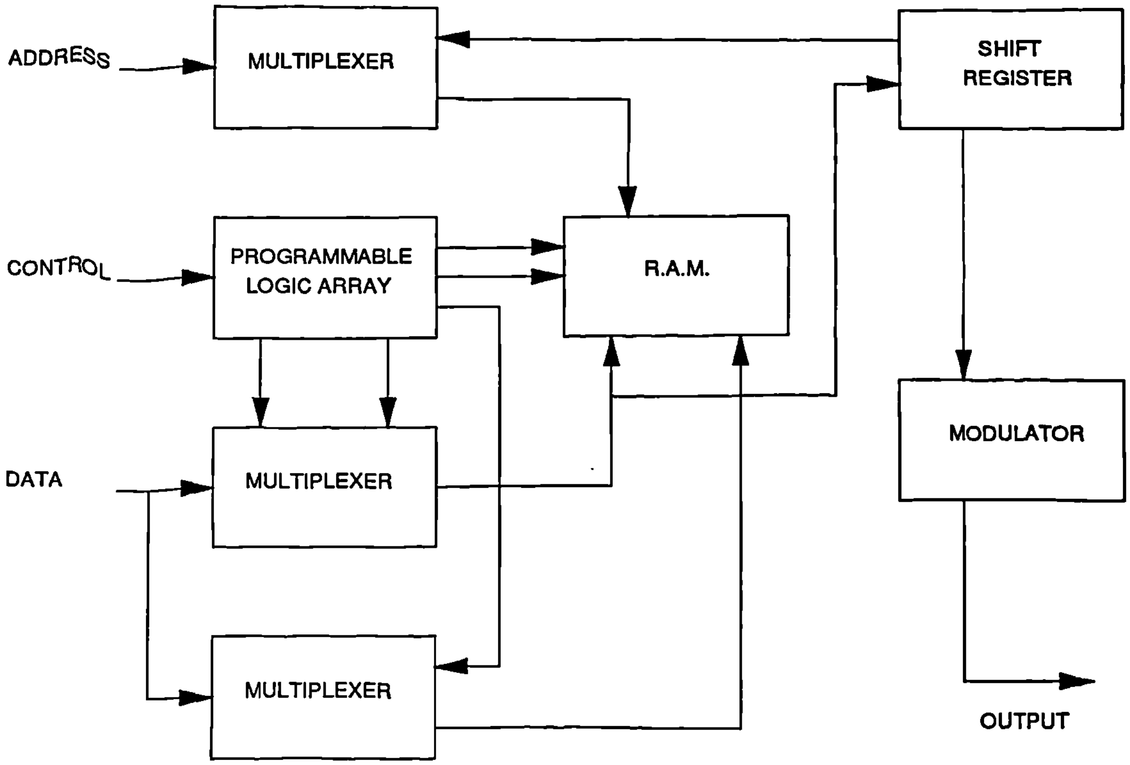


Figure 15 : Block Diagram of Transmitter

The operation of the state machine is as follows. The outputs of each stage of the shift register are used as addresses into the memory, the location addressed by them contains the next bit to be shifted into the shift register. To assemble the contents of the memory, each state of the desired sequence is isolated and set up as an address. The value of the next chip in the sequence is then the contents of the addressed location. Since this is an almost literal implementation of the concept of a pseudo random sequence, equivalent states are truly equivalent and so are degenerate sequences. Sequences of any length (up to the modulus of the shift register) and non-linear sequences can be generated.

Although the above presents a workable solution to the problem of generating sequences, the type of memory and the interfacing of this circuit to the transmitter have yet to be discussed. The use of fixed Read Only Memories (R.O.M.'s) was considered, since these are available with shorter access delays than most read/write memories. A very fast Random Access Memory (R.A.M.) was found however, that had access delays in the order of 35 nano-seconds. By using a R.A.M., sequences could be changed very quickly by re-writing the contents of the R.A.M. during operation of the transmitter. The output of the generator connects to a suitable driver circuit for whichever access medium is being used. Setting up the memory is slightly more complicated however, since the interfacing requirements of the MC68000 have to be taken into account. With the use of P.A.L.'s as mentioned earlier, this reduced the complexity of this part of the circuit. A more detailed block diagram of the circuit is shown below in Figure 16 .

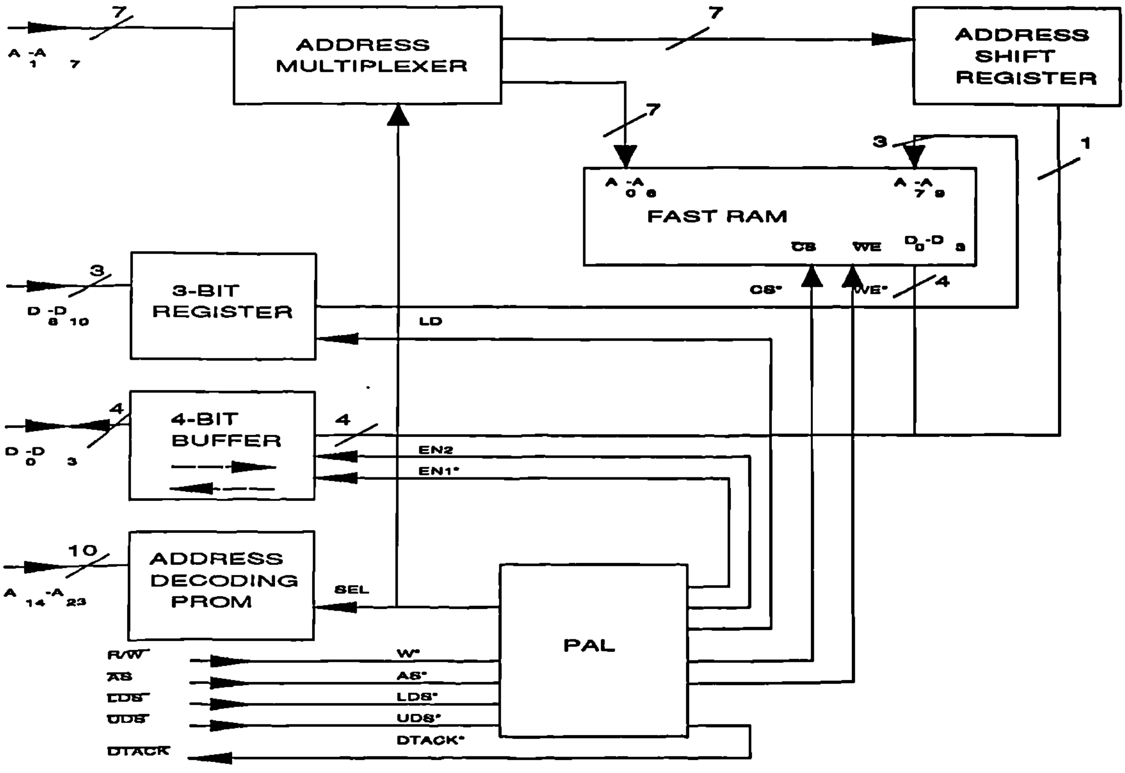


Figure 16 : Detailed Block Diagram of Transmitter

The operation of the circuit is as follows. There are two separate states that the transmitter can be in: a load state in which codes are being loaded into the R.A.M. from the computer;

and a running state in which the transmitter is generating sequences. For the purpose of the prototype, the second state is limited to the generation of continuous sequences. The second state can only be halted at the intervention of the computer, i.e. to load new codes. By using a programmable logic array, it is arranged that the first state is entered automatically when the transmitter is referenced by the computer, i.e. when a valid address is decoded which represents a location in the state-machine R.A.M. In this way, codes are loaded by the computer when it writes to the state-machine R.A.M. At all other times the state-machine is running at the clock speed of the shift register. The signals that drive the multiplexors are driven by the P.A.L. appropriately so that routing of the address and data buses on the computer and the data and address lines on the state-machine R.A.M. depends on whether the R.A.M. is being addressed by the computer or not.

In the running state, the address presented to the state-machine R.A.M. (SMR) is generated by the SR. The input to the SR comes from one of the four data bits available on the SMR. The most significant address bit of the SMR is also the output chip and is sent to the modulation circuitry. The lower of the three multiplexors provides the capability for pre-loaded codes to be used with a very rapid changeover. The codes are all loaded into the SMR, and when a change of code to one of the others present is desired, the contents of one memory location are changed.

In the load state, the multiplexors are switched to select the computer address bus and data bus as the source of address and data for the SMR. The PAL uses the various control signals provided by the 68K computer to decode addresses and produce the acknowledgement signal /DTACK. The PAL also contains logic to convert the computer read/write line into write information for the SMR.

6.3.1 The Receiver

The receiver is based on a correlator which has the ability to receive data out of synchronisation. A choice can be made between traditional acquire and track systems, or a form of continuous correlation. Because the continuous correlator is available in vlsi chip form, this makes this option much more attractive and its disadvantages are countered by the simplicity of design and its tolerance to errors. The particular chip in this design is the TRW 64 bit correlator TDC1023J. The TDC1023J uses emitter coupled logic (ECL) to realise a very fast correlation speed. A block diagram of the correlator is shown below in Figure 17 .

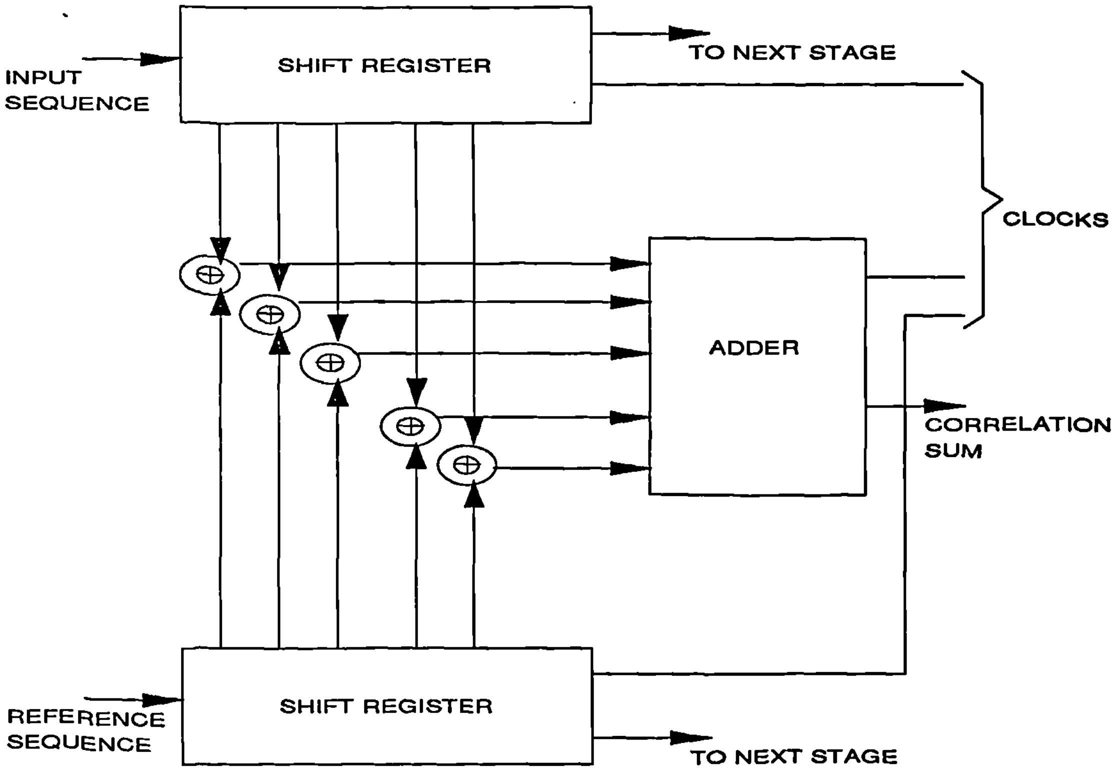


Figure 17 : Block Diagram of Correlator

Operation is as follows. The two shift registers marked 'A' and 'B' are compared continuously in the exclusive-or gates, if two bits in each shift register are the same then the output of the gate is a logic 0, otherwise they are different and the output is a logic 1. The outputs of all the exclusive or gates are taken to a three stage carry look-ahead adder to produce a correlation sum. The maximum clock frequency of the shift registers is 40MHz, that of the adder 25MHz. The chip is therefore capable of continuous operation up to 25MHz and can be cascaded to produce correlation of sequences of lengths which are a multiple of 64 bits- in the prototype design two were used per receiver. Which of the two shift registers is used as the input and which as the reference sequence is arbitary since the chip operates symmetrically. In this case, the 'A' shift register is the input. A general block diagram of the receiver is shown below in Figure 18

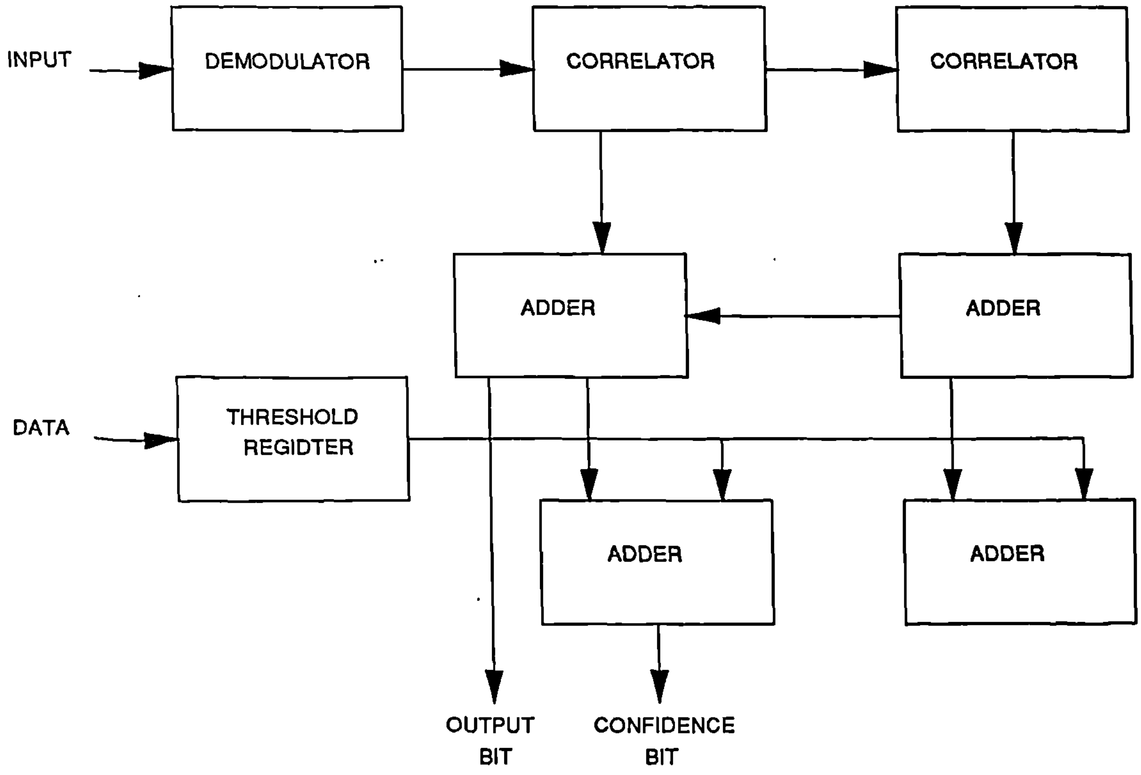


Figure 18 : Block Diagram of Receiver

Interface circuits connect to the access medium to the correlators and the outputs from the two correlator chips are added in a simple carry look ahead adder. The correlation sum is then compared with a threshold value in another adder and if the correlation value exceeds the threshold then a signal denoting a valid data bit is cleared. This flag, together with the value of the data bit decoded are used to produce a measure of the discrimination properties of the codes used. This is measured by a high speed logic analyser, which is capable of detecting given correlation levels and recording up to 2048 correlation scores in total.

6.3.2 Host Computer Software

In the prototype network, each host computer can control one transmitter and one receiver. The software that runs on the node host computer consists of three simple routines and initialisation software. The three routines perform the following functions:

CALC	Mimics a shift register SETUP
TRANS	Sets up and starts the Transmitter
SETUP REC	Sets up and starts the Receiver

The following three sections give detailed description of each of these routines.

6.3.3 Shift Register Operation

The principle of operation of this routine is to mimic a shift register with feedback taps. It does this by using one of the microprocessor's registers as the shift register and calculating a feedback bit based on the contents of a mask (which determines which bits are significant in the feedback), and a temporary copy of the existing shift register. By calling a suitable piece of software during the calculation of the feedback bit, various non-linear operations can be implemented in order to generate non-linear sequences [43].

6.3.4 Setting up the Transmitter

Sets up the transmitter RAM by reading the feedback taps for each code and then selecting code number 0 as the current code. This allows fast reloading of codes during transmission. The use of dual port memory would allow true dynamic code swapping without interrupting the output of the transmitter, although this was not tried.

6.3.5 Setting up the Receiver

This is done by sending each bit of the received sequence to the correlator A register, along with its correspong mask bit. Bit 0 is the sequence value while bit 1 is the mask value.

6.4 Conclusion

In this chapter the detailed design and operation of the prototype spread spectrum network was described. The use of a state machine based sequence generator for the transmitter was detailed, along with the computer software required to drive it. In the receiver, use was made of the high speed correlator chip produced by TRW- the TDC 1023J and the operation of this device was explained in depth, including its operation by the computer.

CHAPTER 7

NETWORK OPERATION

In this chapter the operation of the ISO model level 2 and 3 network layers in the Spread Spectrum Local Area Network (SSLAN) will be described, and compared to the operation of some of the more conventional local area networks described in chapter 2. The various communications methods available will be described, along with the mechanisms used to manage the network. The problem of network security and in particular code sequence allocation and distribution will be addressed, and a solution to this problem using public key cryptographic techniques will be described.

The use of spread spectrum techniques has, up to now, been limited to anti-jam situations in the military environment and situations where noise is a great problem, as was seen in chapter 3. In local area networks, noise and intentional jamming are not so much of a problem and the advantages offered by SS are utilised in other ways. As was shown in chapter 4, one of the properties of SS is its immunity to noise, and in a LAN the perceived noise at a node consists of the data being generated by other users of the network.

A common problem with conventional LAN's is that of network management, e.g. keeping the network working when something goes wrong. In many cases the problem may be minor, for instance the loss of a packet of data due to a burst of noise, or collision between two packets in a CSMA type network and in these cases individual nodes are able to cope. Other types of problem will occur which will require drastic action on the part of one or more of the nodes in order to rectify them. The solution to the class of non trivial problems and to the continued smooth operation of the network is known as network management.

The SSLAN has a major advantage as far as real time LAN's are concerned, and that is its guaranteed maximum message delivery time for any user of the network at any time. An expression for the maximum message delivery time based on signalling rate, propagation delays and software overhead will be derived later in this chapter. This property is one of the prime considerations when dealing with critical systems in the military environment, and is one of the reasons that the SSLAN has been chosen as the basis for the future command system for the Royal Navy in the 1990's.

7.1 Multiple Access

Because the codes used in the SSLAN are chosen to have a high dimensionality (i.e. they are orthogonal or as nearly so as possible), the presence of one code sequence modulating data on the network has the same effect on another sequence as would coloured noise [44].

Referring to chapter 4, the number of m sequences that can be generated by a shift register of length n was seen to be $\frac{2n-1}{n}$. For the case n=7, or sequences of length 127, the number of m sequences is 18. Therefore, if m sequences were to be used in the SSLAN, the maximum number of simultaneous conversations (SC's) able to exist on the network at the same time, ignoring problems such as cabling attenuation and interference, is 18.

When attenuation is considered, along with other types of signal losses, the number of possible SC's is reduced. This reduction in numbers depends on the length of the sequences and their cross and auto correlation properties, and it is the intention of a later chapter to describe work done to calculate exactly what effect attenuation and non-synchronisation will have on the SSLAN.

7.2 Transmission Mechanisms

Given that a set of codes (possibly of differing lengths) has been chosen for the SSLAN code set, there are several methods for transmission of data between nodes. The choice of method is dependent on the type of application, but does not depend on the codes themselves. For example, a broadcast mode of operation where a single node sends data to all other nodes is often provided on existing LAN's and proves to be useful. A study of the requirements of a military network combined with the strengths and weaknesses of the SSLAN suggested six possible operating modes, described below. The six data transmission methods are called point to point, broadcast, group, function, message and protocol.

7.2.1 Point to Point

In this method, each receiver node is allocated a code or group of codes. Thus, each receiver is allocated a unique code and so only one receiver can receive data using that code. To transmit to a node the transmitter must use that node's code to modulate its data. Since the receiver is listening to the network at all times, it will pick up any data with its address. There is a problem if more than one transmitter sends data to the receiver node, but this may be overcome by the use of multiple receivers at each node and through the coordination of their code synchronisation at a higher level. The diagram below illustrates this transmission method. The box marked tx holds the code being transmitted by a node, and the box marked rx holds the code being received by a node. See Figure 19 below.

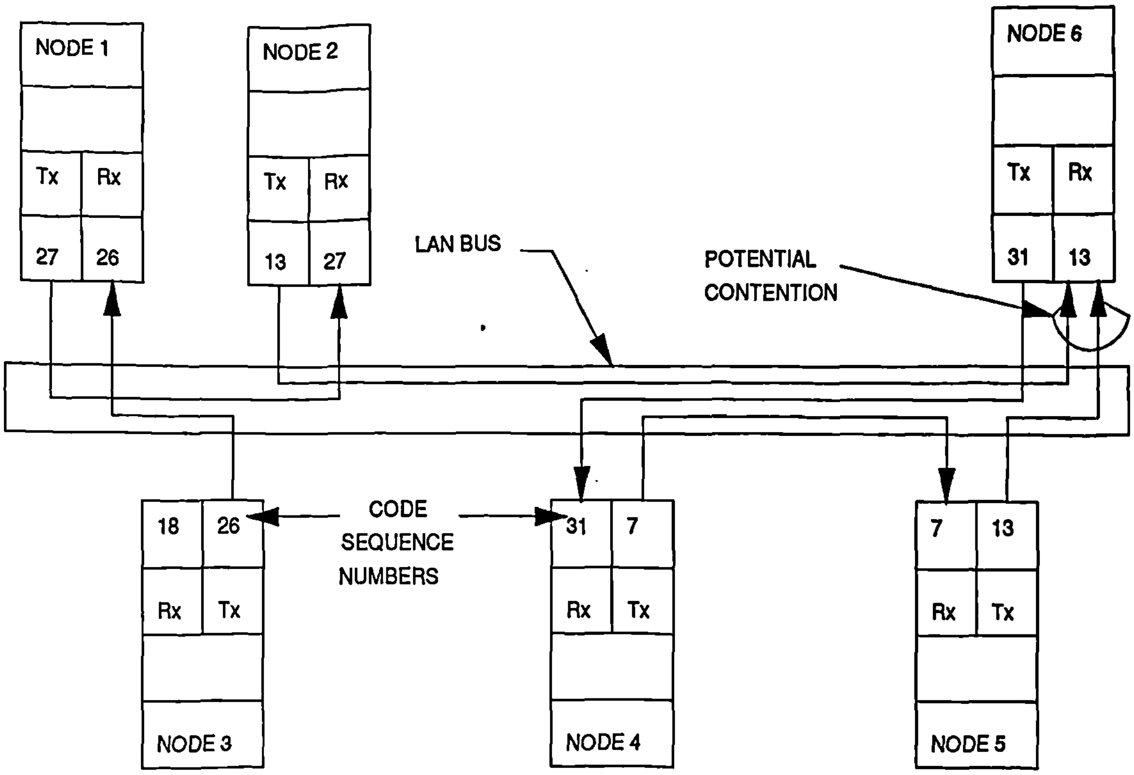


Figure 19 : Point to Point Protocol

7.2.2 Broadcast

In this method a code or codes are allocated to each transmitter node rather than to each receiver node in point to point mode. When the node wishes to transmit data it uses its code to modulate the data. The receiver must be listening for this particular code for its reception to be assured. This method is termed broadcast because it is possible to transmit to all the receiver nodes using only one code. A major advantage from the point of view of the receiver hardware is that there cannot be a contention, since there can only be one transmitter originating a particular code. The main disadvantage with this method is that the receiver must be capable of receiving multiple codes for it to be of any use in a multiple access network. A possible solution to this is to have one receiver scanning a group of codes for activity, where it is assumed that the transmitters using these codes are implementing a data available protocol. This of course negates one of the advantages of the SSLAN, that of guaranteed minimum message delivery time.

The diagram below, Figure 20, illustrates the principle of this method, again using the nomenclature used in the previous figure.

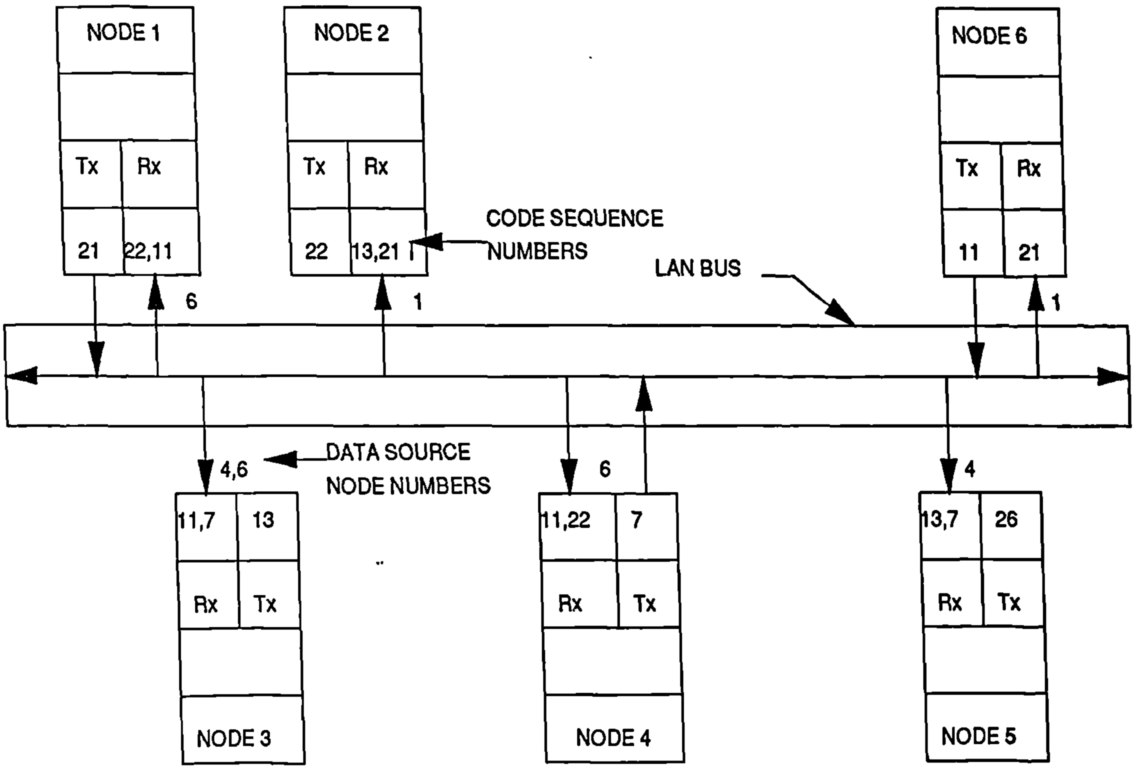


Figure 20 : Broadcast Protocol

7.2.3 Group

This method uses a logical grouping of a set of nodes, into a zone or domain [45] [46] These zones consist of nodes which regularly communicate with each other to perform a common function, distributed across the network. Within each zone a set of code sequences is allocated and communication amongst zone members is limited to the use of these codes. Different zones may use different types of code, depending on the type of application they are performing. Similarly, a node could be a member of more than one zone.

The problem of nodes external to a zone communicating with nodes within the zone can be resolved by using common code sets, or utilising a higher level protocol to tell nodes in a zone to listen out for externally generated data. The use of common code sets produces the same problem as the point to point method, that of contention, and thus the management channel solution is normally preferred.

The diagram below (Figure 21) illustrates the operation of the Group method.

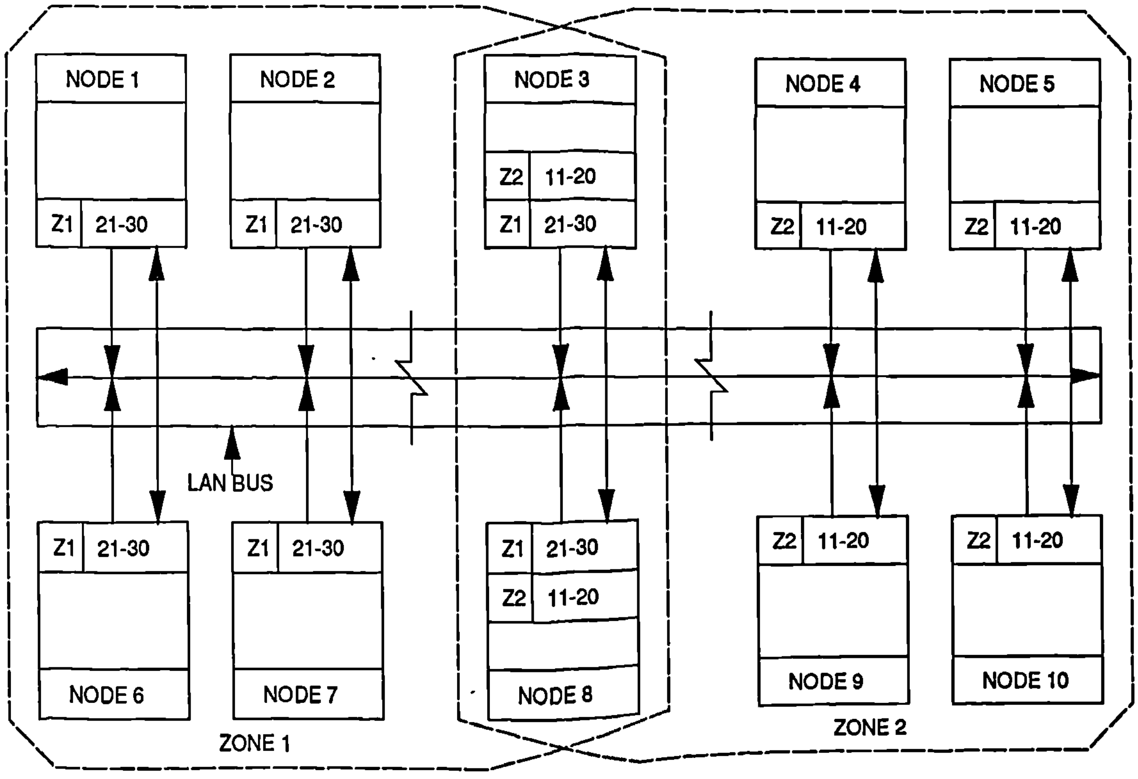


Figure 21 : Group Protocol

7.2.4 Function

This method is perhaps best seen as an extension of the previous method, where each group is assigned a specific function, and where each group is ideally small in size. Thus, a functional element consists of a series of permanently established links between pairs of nodes, all of which can operate simultaneously. The problem of contention is removed by the condition that each functional group has no duplicated code for transmission. Security of the system is improved since particular knowledge of which nodes are using a code is restricted to within the functional group.

This method is used when nodes are grouped together in applications such as industrial command and control, where it is advantageous that the number of wires used to read data from a large number of sensors is minimised. Sensors would report to individual semi-intelligent controllers, while a central management device would be used to report faults, etc. Figure 22 below illustrates the operation of the function method.

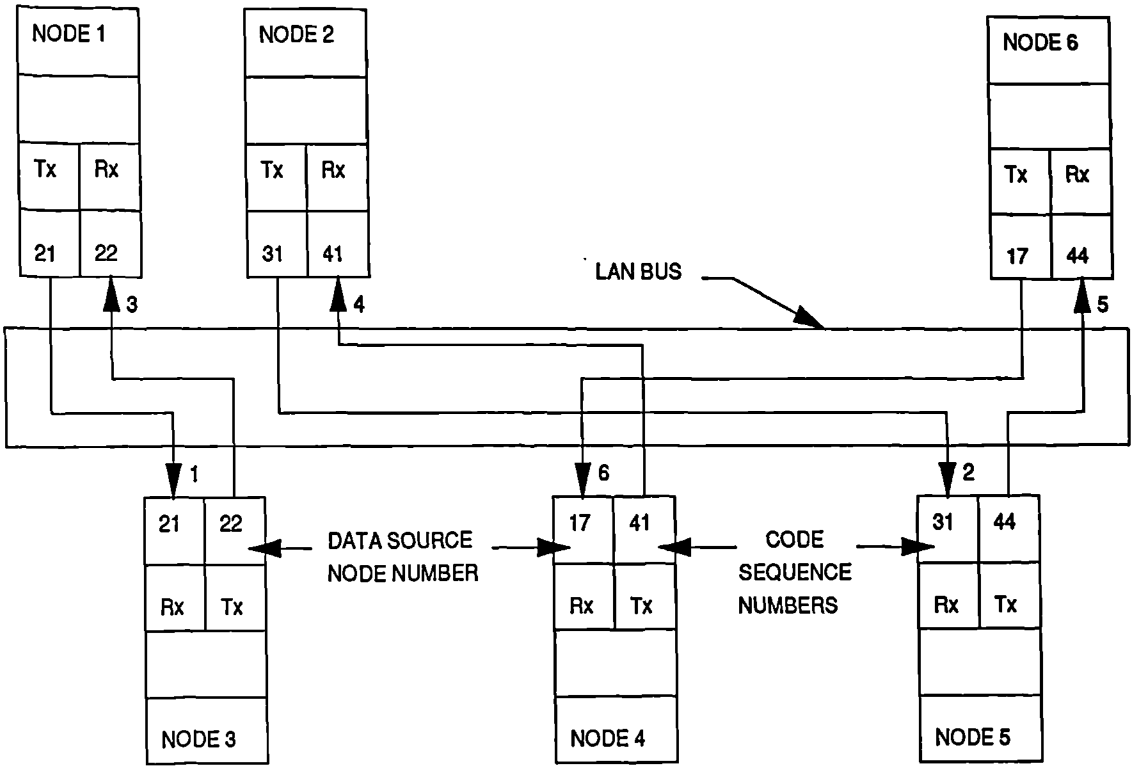


Figure 22 : Function Protocol

7.2.5 Message

This method assigns code sequences to messages according to the type of data contained in the message. A receiver must listen to all codes for which it expects to receive a given data type. Before transmitting, the transmitter must ascertain from its internal code pool which code is allocated to that particular type of message. There is a high probability of contention using this method, especially if messages are of long duration. The ideal application is therefore one in which the messages are all of short duration, for instance in a system which implements a remote procedure call facility. Figure 23 below illustrates the operation of this method.

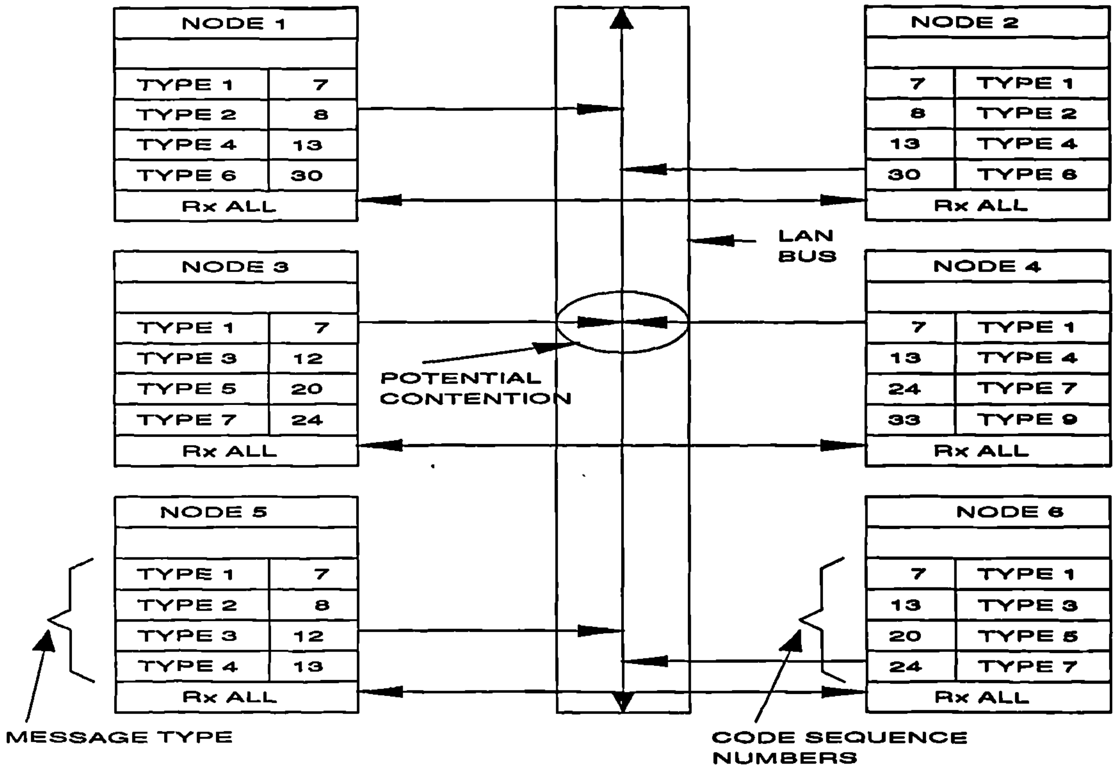


Figure 23 : Message Protocol

7.2.6 Protocol

The final method relies on the use of one of the protocol standards. Codes are allocated either on the protocol level at which they are to be used or to the states possible within a particular protocol level. Errors within a protocol level would indicate loss of code sequence synchronisation, and hence recovery from message loss would be simplified, relying only on the re-synchronisation of the code sequence. A receiver need only listen for the code or codes associated with a given protocol level. There is a probability of contention if more than one node steps across a protocol boundary at the same time as another, but this type of fault would be handled by the next layer in the hierarchy. This is illustrated in Figure 24 below.

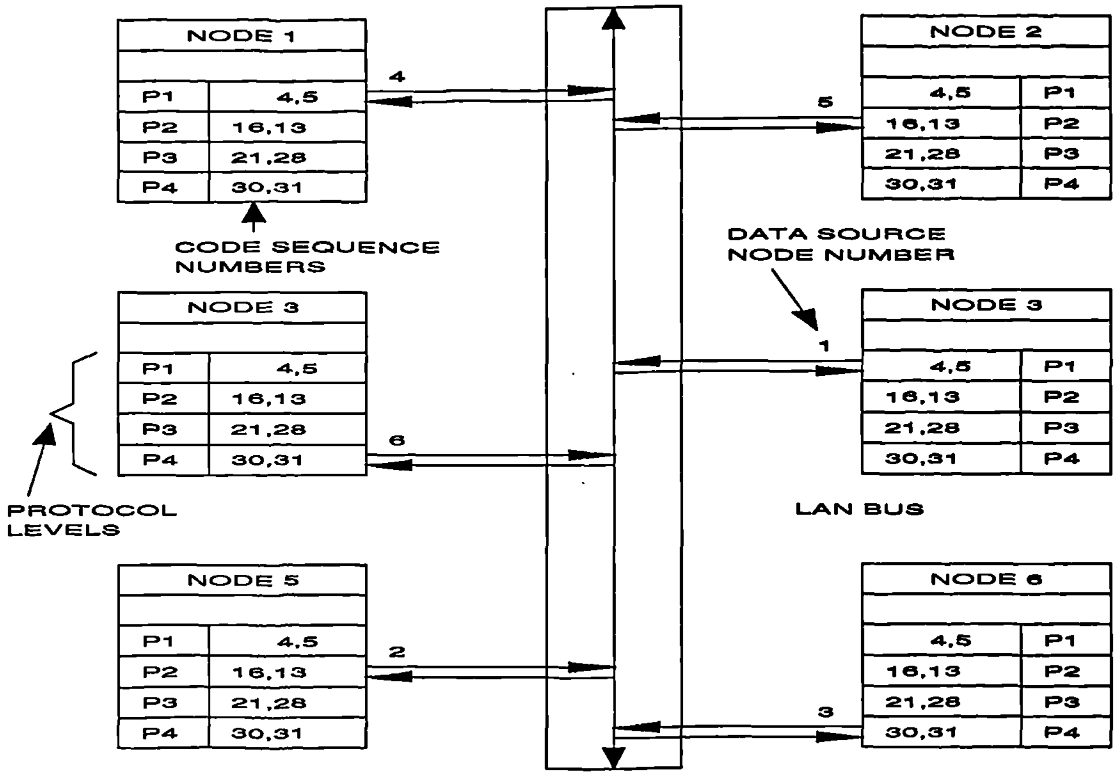


Figure 24 : Protocol Mode

7.3 Limitations

As with any communications system, the SSLAN has limitations which can lead to failures of the network under certain conditions. This section is intended to illustrate these problems, mainly the low point to point data rate, problems of synchronisation without the use of common clocks or synch pulses and the problems of implementing protocols as defined by the ISO and the IEEE.

7.3.1 Low Point to Point Data Rate

Most LAN's operate at point to point data rates between 1 and 10 Mbits/S. The limitations of TDMA systems and the overheads introduced by implementing the physical layer protocol on the networks. The point to point data rate averaged over a long time is normally at least 10 times lower than the quoted figure. In the prototype implementation of the SSLAN, using 127 bit m sequences with a chip frequency of 10 MBps , the point to point data rate was close to 100KBps . In the proposed working version of the SSLAN the point to point data rate will be an order of magnitude higher on average, due to the increased chip frequency and use of optimal codes sequences for a particular application.

Another aspect of this problem is the effect that the type of signalling medium has on the point to point data rate. In the prototype implementation, coaxial cable was used, but experiments with optical fibers suggested that the large bandwidth they offered would allow even larger increases in the data rate. There are problems with running high speed logic at rates close to GHz frequencies, but with the introduction of Gallium Arsenide (GaAs) logic devices with operating frequencies in the 10's of GHz these problems are being resolved.

The type of data source that is connected to the Spread Spectrum LAN should reflect the advantages offered by the LAN. The data source should produce data at a steady rate, with as few bursts as possible. In conventional LAN's a data source of this type requires a large amount of buffering at the source node in order to cope with variations in the availability of the network.

7.3.2 Protocol Incompatibilities

The standardisation bodies - the International Standards Organisation (ISO) and the Institute of Electrical and Electronic Engineers (IEEE) produced a set of proposals for protocols for local and wide area networks, resulting in the ISO Open Systems Interconnect seven layer model for wide area networks and the IEEE 802 series of proposals for local area networks. As was mentioned in chapter 2, the principle of operation of a protocol is that each layer should be isolated as much as possible from the layers to which it interfaces, each layer providing a service to the layer above it and using the service provided by the layer below it.

With the protocol type of code utilisation described in this chapter, knowledge of the type of protocol message being sent between nodes is required at the lowest level, in order for the appropriate code sequence to be used. This is contrary to the recommendations of both the IEEE and ISO.

7.4 Security and Code Distribution

In the applications for which the SSLAN was originally designed, the security of the data being transmitted was considered to be important. If the network is operating in the function or group mode, then the allocation of a set of codes within a group is known only to members within the group, so any eavesdropper would have to listen to all the codes used by a group in order to extract information about a particular member of the group. A corrolary is that the more nodes there are within a group, the harder it will be to eavesdrop.

The distribution of code sequences within a group, or across the whole network is obviously of great importance, since if this operation can be guaranteed to be secure, the security of the network is greatly increased. The use of various cryptographic techniques was investigated, and some of the conclusions and methods found are presented here. These include Public Key Cryptography [47] the Data Encryption Standard (D.E.S.) [48] and Knapsack algorithms [49] although the latter has recently been discounted because it is possible to decode Knapsack codes rather easily using Discrete Fourier Transform techniques.

7.5 Authentication and Signatures

When data encryption methods are being used to distribute spread spectrum sequences, as has been proposed here, the problem of authenticating the distributor of the codes is important. It is essential that the receiver of the new codes can verify that the sequences have been sent by the claimed person and that the sender knows about the allocation as well. The requirements of a large number of codes and authenticability are, however, conflicting because a good code requires a large code space (code space = potential number of codes) while an authenticable code requires a small code space in order to validate the code and prevent intrusion.

CHAPTER 8

RESULTS OF CODE SET DESIGN

The performance of the code sets designed with the simulated annealing programme show that the peak correlation is nearly as good as other types of deterministic sequence, with the added advantage of being able to use arbitrary length codes. For comparison with deterministic sequences, code sets of the same length as the deterministic set were generated, and performance was measured by the peak correlation.

APPENDIX I shows the results of a typical simulated annealing run on the computer. Each line in the listing shows the annealing temperature, the change in energy and the number of bit changes. At the end of the run the codes in the code set are printed out, along with the maximum correlation value.

Defining the peak out of phase autocorrelation to be θ_a, the peak cross correlation to be θ_c, the peak aperiodic out of phase autocorrelation to be C_a and the peak aperiodic cross correlation to be C_c, the following graphs plot two seperate curves for each code set. The first is $\max\{\theta_a, \theta_c\}$ taken over all code pairs in the code set and plotted over half the x axis. The second is $\max\{| C_a |, | C_c |\}$ again taken over all code pairs in the code set and plotted over the whole x axis. The important point to note is that all the curves have two large peaks at zero phase shift and full length phase shift. The zero phase shift peak shows the ability of the codes to extract the correct phase information, while the peak at full length phase shift shows their ability to discriminate between positive and negative modulation of data.

Figure 25 illustrates a code set of 18 codes of length 127 bits. The code set size and length are the same as the m-sequence set of length 127 bits. This particular m-sequence set has some very good properties, such as a very large code set size (for m-sequences) of 18 codes. The peak cross correlation for all pairs of m-sequences of length 127 is 41, while for a preferred sub set of 6 codes it is 17. Using simulated annealing, a peak correlation of 30 can be acheived, which is better than that of the m-sequence set.

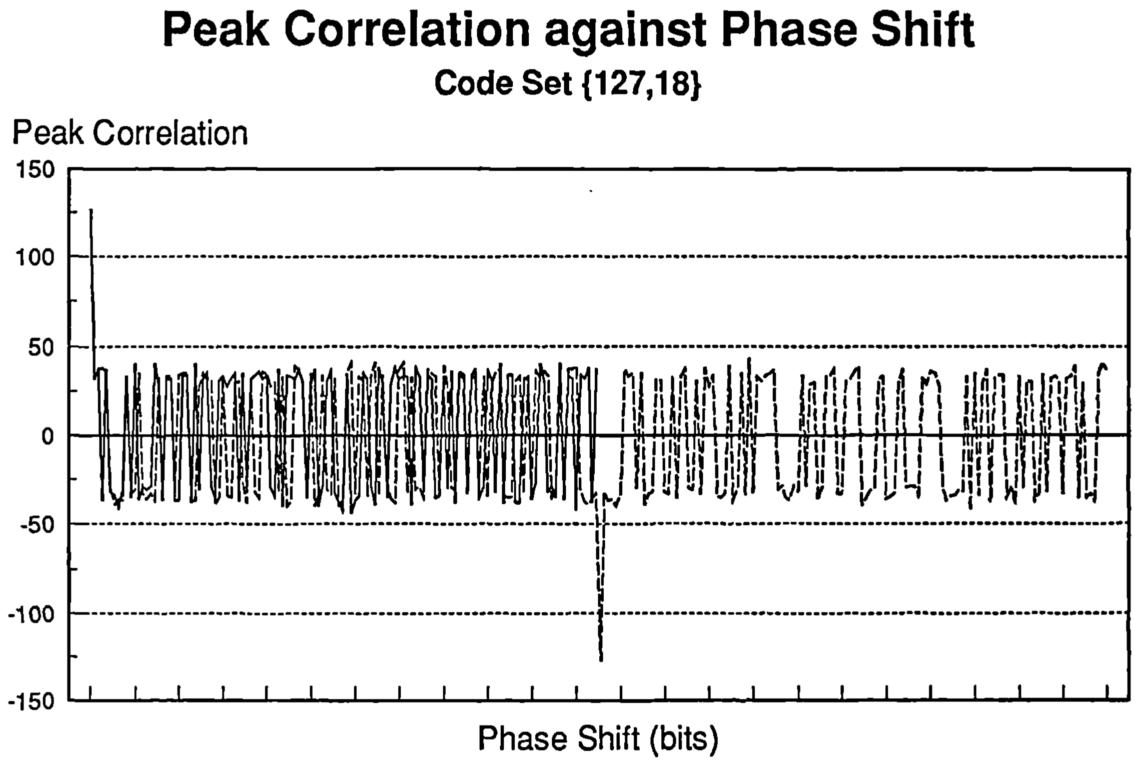


Figure 25 : Correlation for Code Set {127,18}

An important property of the simulated annealing method is the ability to design code sets with arbitrary length, and the next two figures illustrate this. Figure 26 illustrates the correlation for the code set {50,50}. While Figure 27 showd the code set {25,25}. Code set {25,25} also demonstrates the existence of reasonably large code sets of very short length, an important point in the design of networks.

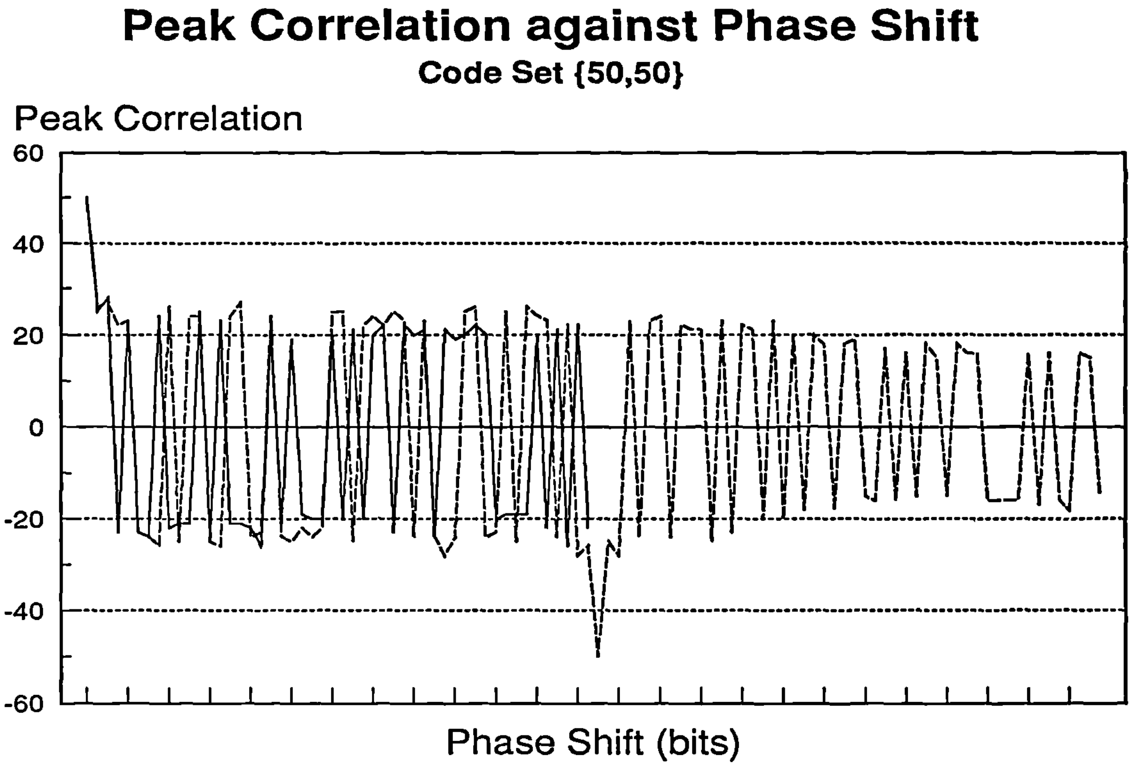


Figure 26 : Peak Correlation for Code Set {50,50}

The final Figure, 28 illustrates the correlation for code set {100,40}. This shows that the correlation threshold is over 50% of the code length for a large code set of medium length.

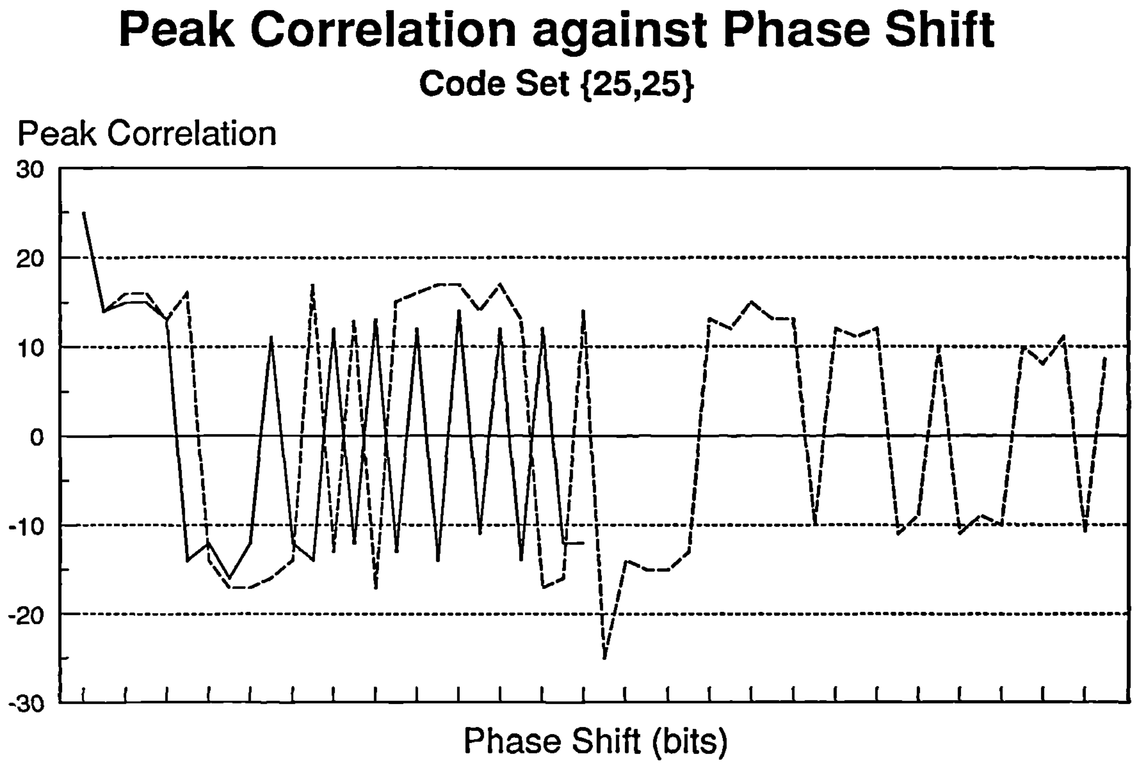


Figure 27 : Peak Correlation for Code Set {25,25}

One of the limitations of this method is the time taken to design a code set. The codes {100,40} took approximately.5 hours to design on a high performance personal computer. Given that a code set only needs to be designed once however, the overhead of code design is almost certainly worth the penalty.

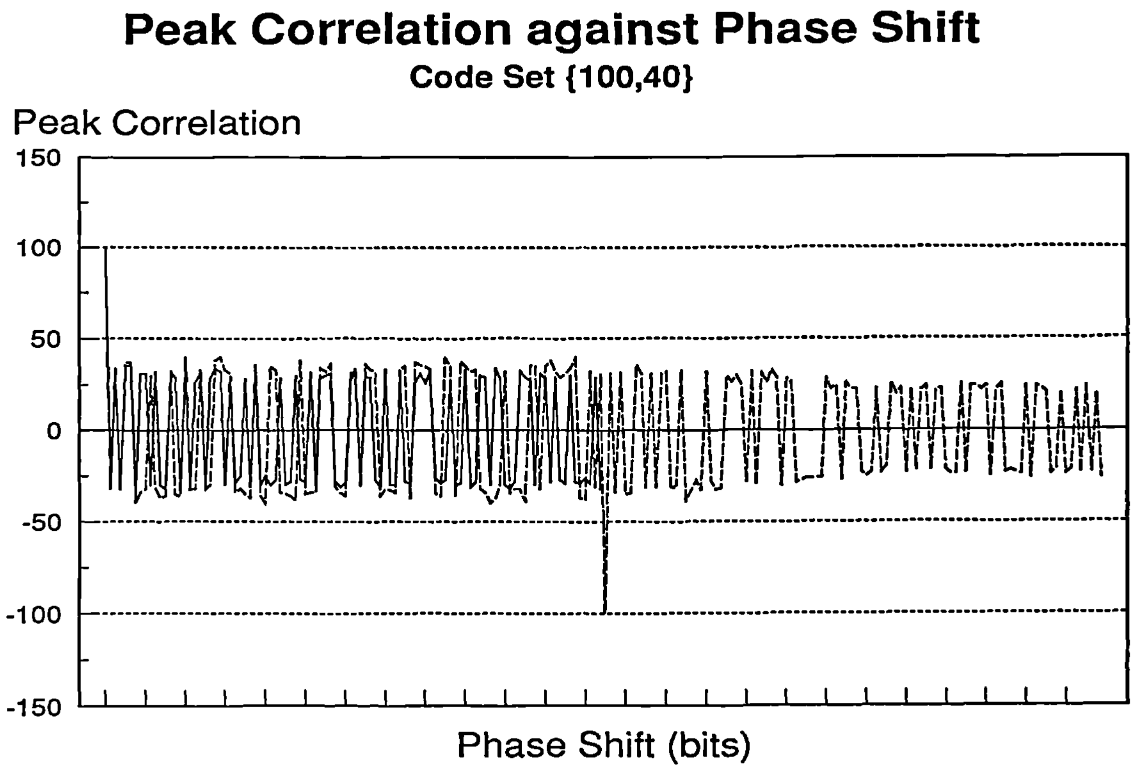


Figure 28 : Peak Correlation for Code Set {100,40}

CHAPTER 9

CONCLUSION

The use of direct sequence spread spectrum signalling in local area networks is a new technique which has been demonstrated by the author to be feasible. The type of sequence that is used is crucial to the performance of the network. The physical realisation of a spread spectrum network such as has been described in this thesis is limited by two factors:

The performance of correlators. The state of the art in correlators is a device that continuously correlates at 25 million bits per second at one bit resolution and with 64 register stages. Optical correlators are a theoretical possibility, with clock speeds measured in Gigahertz. Combined with optical mixers for code modulation high speed optical correlators could provide the basis for an ultra high speed network.

Code restrictions. One of the limitations of presently available spread spectrum systems is the restriction on code length. Most methods for generating sequences involve using Galois Field Theory, or the use of fourier transforms. Optimal sequences can be designed using these methods, but two major problems exist. A prototype network was constructed to demonstrate the principle of the netowrk, and although the results were encouraging this showed the limitations of using m-sequences.

First, their length has to be related to an integer power of a prime. This means that the clock rate and data rate cannot be independent.

Second, the performance of the codes in a true multiple access environment is not taken into account in their design.

As was mentioned in CHAPTER 3 the network signalling mechanism affects the way codes combine and interfere with each other. There is a large difference between using code division in an optical network and in an electrical network. Therefore any method for generating codes should try to minimise the signal to noise ratio of the network as a whole based on the signalling mechanism. A code set that has a good performance for one type of signalling mechanism will not neccessarily have good performance for another signalling mechanism.

The thesis has described the concept of a code division multiple access spread spectrum local area network. A discussion of finite field algebra has shown that good code sets are available, but that they have some limitations. A new method for generating arbitrary length code sets which uses the method of Simulated Annealing has been described, and several examples of code sets designed by it have been included. The simulated annealing program does not optimise code sets based on higher order correlations, and this would be a good starting point for further work.

APPENDIX A

SEQUENCE SETS WITH SMALL CROSS CORRELATION

A communications system that is intended to support a number of users requires a distinct code for the exclusive use of each user. Thus, the selection of code sets with high in-phase auto-correlation and low cross correlation is of great importance. The automatic generation of such code sets is, unfortunately, impossible however several criteria exist for judging how good a particular set of codes is. Much effort has been devoted to establishing upper and lower bounds on the correlation performance of code sets [50] and [51] These treatments are restricted to the aperiodic and odd cross correlation functions, and the realisation of bounds by these methods are impractical due to the large number of calculations necessary to scan all possible phases of the sequences in the set. The choice of code sets is therefore mostly based on efficient methods for searching by computer, but for long sequences these methods are not realistic either. In summary, a set of sequences for spread spectrum multiple access communications must satisfy the following criteria [52].

Each sequence $x \in K$ possesses a small out of phase periodic auto correlation.

Each pair of sequences $x, y \in K$ possess a small periodic cross correlation.

The number of members of the set K which satisfy the two previous rules must be large.

Each sequence $x \in K$ must be easy to generate.

Each sequence $x \in K$ must be secure.

Although Shaar lists these criteria as being of direct relevance to selection of code sets, the method of using other means (such as formal encryption) to lessen the constraints on a code set is not mentioned. If we examine the problem of selecting a first guess bound on the auto correlation and cross correlation of a set of sequences, the following equations will yield the desired result: In general

$$\sum_{l=0}^{N-1} \theta_{w,y}(l)\,[\theta_{x,z}(l+n)]^* = \sum_{l=0}^{N-1} \theta_{w,x}(l)\,[\theta_{y,z}(l+n)]^*$$

The first bound on $\theta_{x,y}(l)$ is

$$|\,\theta_{x,y}(l)| \leq \|x\| \cdot \|y\|$$

From this, setting n=0, z=y and w=x,

$$\sum_{l=0}^{N-1} |\,\theta_{x,y}(l)|^2 = \sum_{l=0}^{N-1} \theta_x(l) \times \theta_y^*(l)$$

Applying the Cauchy Inequality

$$\sum_{l=0}^{N-1} |\,\theta_{x,y}(l)|^2 \leq \sqrt{\sum_{l=0}^{N-1} |\,\theta_x(l)|^2} \times \sqrt{\sum_{i=0}^{N-1} |\,\theta_y(l)|^2}$$

Rewriting 3 as

$$\sum_{l=0}^{N-1} |\,\theta_{x,y}(l)|^2 = \theta_x(0) \times \theta_y(0) + \sum_{l=1}^{N-1} \theta_x(l) \times \theta_y^*(l)$$

and re-applying the Cauchy Inequality to the RHS to find an upper and lower bound,

$$\sum_{l=0}^{N-1} |\,\theta_{x,y}(l)|^2 <> \theta_x(0) \times \theta_y(0) \pm \sqrt{\sum_{l=1}^{N-1} |\,\theta_x(l)|^2} \times \sqrt{\sum_{l=1}^{N-1} |\,\theta_y(l)|^2}$$

If θ_c is the peak cross correlation magnitude of a set of sequences, and θ_a is the peak out of phase auto correlation magnitude of the same set, then

$$\theta_c = \max|\,\theta_{x,y}(l)| : 0 \leq l \leq N-1, x \in \chi, y \in \chi, x \neq y$$

$$\theta_a = \max|\,\theta_x(l)|:1 \le l \le N-1, x \in \chi$$

Sarwate [53] has shown that

$$\left(\frac{\theta_c^2}{N}\right) + \frac{N-1}{N \times (K-1)} \times \left(\frac{\theta_a^2}{N}\right) \ge 1$$

Which can be re-arranged to give

$$\theta_{\max} \equiv \max(\theta_a, \theta_c) \ge N \times \sqrt{\frac{K-1}{N \times K - 1}}$$

which is the correlation parameter and represents the lowest possible crosscorrelation and out of phase autocorrelation value for a given code set. Similar limits apply for aperiodic correlation parameters.

APPENDIX B

THEORY OF FINITE FIELDS

A finite field GF(p) is a finite set of elements with addition (+), subtraction (-), multiplication (×) and division (/) defined. The number (p) of field elements denotes the order of the field. Addition and multiplication satisfy commutation, distribution and association as defined below:

For all $\alpha, \beta, \gamma \in GF(p)$

$$\alpha + \beta = \beta + \alpha$$

$$\alpha \times \beta = \beta \times \alpha$$

$$\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$$

$$\alpha \times (\beta \times \gamma) = (\alpha \times \beta) \times \gamma$$

$$\alpha \times (\beta + \gamma) = \alpha \times \beta + \alpha \times \gamma$$

Also, elements $0, 1, -\alpha, \alpha^{-1}$ for all $\alpha \in GF(p)$ must exist such that:

$$0 + \alpha = \alpha$$

$$(-\alpha) + \alpha = 0$$

$$0 \times \alpha = 0$$

$$1 \times \alpha = \alpha$$

if $\alpha \neq 0$ then $(\alpha^{-}1) \times \alpha = 1$

One of the fundamental theorems of the algebra of finite fields is:

There exists a finite field GF(p) only for p prime or a power of a prime.

There are therefore two cases of interest, first when p is prime the field is known as the ground field, GF(p) and when p is the n^{th} power of a prime the field is known as the extension field GF(n).

2.1 The Ground Field GF(p)

When p is a prime number, arithmetic is performed modulo p. The result of an arithmetic operation is the usual result modulo p.

A non zero element $\alpha \in GF(p)$ is said to be of multiplicative order λ if λ is the lowest non-zero integer such that $\alpha^{\lambda} = 1$. An element with λ = p-1 is called a primitive element. A primitive element has the property that its powers generate all the non-zero field elements. The number of primitive elements in the field is $\phi(\phi(p)) = \phi(p-1)$ where $\phi(p)$ is the Euler function [1] or totient.

2.2 The Extension Field GF(N)

Operations in GF(n) [2] are carried out modulo a primitive polynomial q(t) (whose coefficients are from GF(p)) of degree n. For the same reason that $2 \equiv 0$ in GF(2), $q(t) \equiv 0$ in GF(n). A primitive polynomial is irreducible and divides t^{m-1} for $m = 2^{n-1}$ and for no

[1] The Euler function has the value of the number of numbers that are relatively prime to its argument

[2] In the case of spread spectrum communications, where two valued functions are used the prime p is normally 2, and the remainder of this section will assume that p=2.

smaller m. Elements of GF(n) are all polynomials of degree n-1 or less with coefficients from GF(2). Representing each polynomial element by its coefficients gives an n-tuple version of the element, e.g. $t^2 + 1 \equiv 101$. Elements from GF(n) can also be represented as powers of a primitive element α. A primitive element is defined in a similar way to one in GF(p). The number of primitive elements is therefore $\phi(p^n - 1)$.

APPENDIX C

THE THEORY OF INFORMATION

This Appendix contains a derivation of the Channel Capacity equation derived by Claude Shannon in 1949.

3.1 The Discrete Channel without Noise

The capacity of a discrete channel is given by

$$C = \underset{T \to \infty}{Lt} \left\{ \log_2 \left(\frac{N(T)}{T} \right) \right\}$$

Where N(T) is the number of allowed signals of duration T.

Theorem 1: If the sequences of symbols allowed can be described as a state machine then the channel capacity can be found.

Theorem 2: If b_{ij}^{s} is the duration of the s^{th} symbol which is allowable in state i and leads to state j then

$$C = \log_2(W)$$

Where W is the largest real root of

$$\int W(-b_{ij}^{s}) - \delta_{ij} = 0$$

δ being the identity matrix. A discrete source can be modelled by a stochastic process, more specifically by a Markov process.

Definition 1: An Ergodic process is one in which every sequence produced by the process is identical in statistical properties. Relating this to state machines, a state machine is ergodic if it does not consist of more than one disjoint part and if and only if the circuit lengths of the state machine are relatively prime.

If the first condition is satisfied but not the second, the sequences have periodicity, and fall into d classes (d being the G.C.D. of the circuit lengths). If the first condition is violated, we have a set of pure Ergodic sources, only one of which will be chosen. If p_i is the probability of state i and $P_i(j)$ is the transition probability to state j, for a stationary process

$$P_j = \sum_i p_i P_i(j)$$

For an Ergodic source, $P_j(N)$ of being in state j after N symbols approaches equilibrium value as $N \rightarrow \infty$.

3.2 Entropy

If there are a set of events with probabilities $p1, p2, p3, \ldots pn$ we can assign a measure $H(p_1, p_2, p_3, \ldots p_n)$ of the choice of the outcome of an event. H will then satisfy the following conditions:

H should be continuous with respect to p_i

If $p_1 = p_2 = \ldots\ldots p_n = \frac{1}{n}$ then H should be monotonic and increasing (the more events, the more uncertainty).

If a choice is broken down into two successive choices, the original H should be the weighted sum of the individual values of H.

From this we can decompose a choice from s^m equally likely possibilities into a series of m choices each from s equally likely possibilities and obtain

$$A(s^m) = mA(s)$$

$$A(t^n) = nA(t)$$

Choosing n arbitrarily large, take m such that

$$s^m \leq t^n < s^{m+1}$$

Taking logs and dividing by n log s,

$$\frac{m}{n} \leq \frac{\log t}{\log s} < \frac{m}{n+1}$$

or

$$\frac{m}{n} - \frac{\log t}{\log s} < e, e \approx \frac{1}{n}$$

where e is arbitrarily small. From the monotonic property of A(n),

$$A(s^m) \leq A(t^n) \leq A(s^{m+1})$$

$$mA(s) \leq nA(t) \leq (m+1)A(s)$$

Dividing by nA(s),

$$\frac{m}{n} \leq \frac{A(t)}{A(s)} \leq \frac{m}{n+1}$$

or

$$\frac{m}{n}-\frac{A(t)}{A(s)}<e$$

Combining with 7,

$$\frac{A(t)}{A(s)}-\frac{\log t}{\log s}<2e$$

or

$$A(t)=-K\log(t)$$

With K positive to satisfy (2). If we have a choice from n possibilities with probabilities $p_i=\frac{ni}{\Sigma(ni)}$. We can break down a choice from $\Sigma(ni)$ possibilities into a choice from n possibilities with probabilities $p_1,p_2,p_3,\ldots.p_n$ and then if the i^{th} was chosen, a choice from n_i with equal probabilities. Using the third condition,

$$K\log\Sigma(ni)=H(p_1,p_2,p_3,\ldots p_n)+K\,\Sigma(p_i\log n_i)$$

So

$$H=K\,\Sigma(p_i\log\Sigma(ni))-\Sigma(p_i\log ni)$$

$$H=-K\,\Sigma(p_i)\log\left(\frac{n_i}{\Sigma(n_i)}\right)$$

$$H=-K\,\Sigma(p_i)\log p_i.$$

3.3 Properties of H

H vanishes if and only if all p_i but one are zero, the remaining one having a probability of one.

For a given n, H is a maximum when all $p_i = \frac{1}{n}$ so that H = log (n).

Assume two events x and y, with m possibilities for x and n for y. If p(i,j) is the prob. that x=i and y=j,

$$H(x,y) = -\sum_i, j p(i,j) \log p(i,j)$$

So H(x,y) <= H(x) + H(y), with equality only when x and y are independent. Any change towards equalising the $p_1, \ldots, p_n$ increases H. Assume two interdependent events x and y. If $p_i(j)$ is the probability that y=j given that x=i,

$$p_i(j) = P \frac{(i,j)}{\sum_j (p(i,j))}$$

The conditional entropy H_y is the average of the entropy of y for each value of x, weighted according to p(x).

$$H_x(y) = -\sum_i, jP(i,j) \log P(i,j) + \sum_i, jP(i,j) \log \sum_j P(i,j)$$

$$H(x,y) = H(x) + H_x(y)$$

If we consider a long message of N symbols, it will contain with high probability $p_1 N$ occurrences of the first symbol ,....$p_n N$ occurrences of the n^{th} symbol. The probability of this message is

$$p = p_1 p_1 N \ldots\ldots p_n p_n N$$

or

$$\log p = N \sum_i p_i \log p_i$$

$$\log p = -NH$$

$$H = \frac{\log \frac{1}{p}}{N}$$

Relative entropy is defined as the ratio of the entropy of a source to the maximum value it can have for the same set of symbols. The redundancy is defined as (1 - relative entropy).

3.3.1 Representing the encoding and decoding operations

Both transmitter and receiver are known as discrete transducers. They may have internal finite memory in which case there are a finite number of states that the transducer can be in. The output and next state of the transducer will be functions of the present state and the input:

$$y_n = f(x_n, a_n)$$

$$a_{n+1} = g(x_n, a_n)$$

Where x_n is the n^{th} input symbol, a_n is the state of the transducer when the n^{th} input symbol is introduced, y_n is the output symbol. If there is a second transducer which operates on the output of the first and recovers its original input, the first transducer is non-singular and the second is its inverse.

Theorem 3: The output of a finite state transducer driven by a finite state source is a finite state source with entropy less than or equal to that of the input. If the transducer is non-singular they are equal.

Theorem 4: If a source has entropy H (bits per symbol) and a channel has a capacity C then it is possible to encode the output of the source so to transmit $\frac{C}{H} - e$ symbols per second with e arbitrarily small. It is not possible to transmit at an average rate greater than $\frac{C}{H}$.

3.4 The Discrete Channel with Noise

Two types of noise may be identified: if a given transmission always produces the same received signal then the noise is distortion, otherwise the noise is an independent variable N.

$$E = f(S,N)$$

Where E is the received signal, S is the transmitted signal and N is the noise. In general, the noise may be considered as a stochastic process. The probability that if the channel is in state a and symbol i is transmitted, that symbol j will be received and the channel left in state b is

$$p_{a,i}(b,j)$$

a and b range over all the possible states and i over the possible transmitted signals, j over the possible received signals. If successive symbols are independently perturbed there is only one state and the channel is described by a set of transition probabilities $p_i(j)$. If the entropy of the source is H(x), the entropy of the output of the channel is H(y), the joint entropy of the input and output H(x,y) and the conditional entropies $H_x(y)$ and $H_y(x)$ then

$$H(x,y) = H(x) + H_x(y) = H(y) + H_y(x).$$

The rate of transmission R is the rate of production (entropy of the source) minus the average rate of conditional entropy:

$$R = H(x) - H_y(x)$$

The conditional entropy $H_y(x)$ is known as the equivocation and it measures the average ambiguity of the received signal. If we imagine an observer who notes errors in the received data and transmits this information over a correction channel back to the transmitter, then

Theorem 5: If the correction channel has a capacity $H_y(x)$ it is possible to so encode the correction data as to send it over the correction channel and correct all but an arbitrarily small fraction e of the errors. This is not possible if the correction channel capacity is less than $H_y(x)$.

$H_y(x)$ is the amount of additional information that must be supplied per second at the receiving point to correct the message. Moreover, it possible to send information over the channel at the rate C with as small a frequency of errors or equivocation as desired by the use of suitable coding.

APPENDIX D

THE THEORIES OF MOBIUS AND EULER

4.1 The Mobius Function

The mobius function of a number n, $\mu(n)$ is defined as follows

$\mu(n) = 1$ if n = 1

$\mu(n) = 0$ if n contains a prime factor raised to a power greater than 1

$\mu(n) = (-1)^k$ if n is the product of k distinct primes

4.1.1 Properties of the Mobius Formula

Except in the case n = 1, $\sum_{\frac{d}{n}} \mu(d) = 0$ where the summation limit $\frac{d}{n}$ means for all divisors d

of n.

$$\sum_{\frac{d}{n}} \mu \frac{(d)}{\left(\frac{d}{n}\right)} = 1$$

where div(n) is the number of divisors of n

$$\lambda(n) = \sum_{\frac{d}{n}} \mu\left(\frac{n}{d}\right)\log(d)$$

where

$\lambda(n) = \log(\text{p})$ if n is a power of a single prime p or 0 if n contains different primes

This function is used when one is calculating the inverse of a function given by $F(n) = \sum_{\frac{d}{n}} f(d)$ where n is an integer and d is a divisor of n. The values of F for n up to 8 are shown below.

n	d	F(n)
1	1	f(1)
2	1,2	f(1) + f(2)
3	1,3	f(1) + f(3)
4	1,2,4	f(1) + f(2) + f(4)
5	1,5	f(1) + f(5)
6	1,2,3,6	f(1) + f(2) + f(3) + f(6)
7	1,7	f(1) + f(7)
8	1,2,4,8	f(1) + f(2) + f(4) + f(8)

solving for f(n):

$$f(1) = F(1)$$
$$f(2) = F(2) - F(1)$$
$$f(3) = F(3) - F(1)$$
$$f(4) = F(4) - F(2)$$
$$f(5) = F(5) - F(1)$$
$$f(6) = F(6) - F(3) - F(2) + F(1)$$
$$f(7) = F(7) - F(1)$$
$$f(8) = F(8) - F(4)$$

In general,

$$f(n) = \sum_{\frac{d}{n}} \mu\left(\frac{n}{d}\right) F(d) = \sum_{\frac{d}{n}} \mu(d) F\left(\frac{n}{d}\right)$$

4.2 The Euler Function

The Euler function $\phi(n)$ gives the number of numbers less than n

and prime to it. So

$$\phi(1) = 1, \phi(2) = 1, \phi(3) = 2, \phi(4) = 2, \phi(5) = 4, \phi(6) = 2 \ldots$$

An important property of the Euler function is the one mentioned above, namely $n = \sum_{\frac{d}{n}} \phi(d)$

Applying the mobius inversion formula to this gives

$$\phi(n) = \sum_{\frac{d}{n}} \mu(d) \left(\frac{n}{d}\right)$$

$$\frac{\phi(n)}{n} = \sum_{\frac{d}{n}} \frac{\mu(d)}{d}$$

For a shift register of length n, the number of m sequences that it can generate is, in general,

$$\frac{\phi(2n-1)}{n}$$

and if 2n-1 is prime this is equal to $\frac{(2^n-2)}{n}$, e.g. for n=7, this is 18.

The work so far in this chapter has used linear feedback shift register sequences as its examples, and m sequences in particular as a special case, using the properties of these types of sequence to classify them.

A more general type of linear sequence is the two-level sequence, i.e. one in which the auto-correlation has two values, as discussed before.

APPENDIX E

ENCRYPTION METHODS

5.1 The R.S.A. Public Key Cryptographic Method

Both this method and the Knapsack method rely for their success on the principle that a two-key code, where the two keys are not related in any way, simplifies the problems associated with keeping the keys secret. In other words, it is possible for anyone to encrypt data using one of the keys (the public key), but only the receiver has a copy of the private key and so only he can decrypt the data. In fact, the public key should be made as widely available as possible. In more concise form:

DECRYPT(ENCRYPT(DATA))=DATA

It is difficult to deduce D from E

E cannot be broken given the data and its encrypted form.

The problem with this method is finding good candidates for DECRYPT and ENCRYPT. The method proposed by Rivest et al. provides a solution that is widely used today:

Choose two large prime numbers with more than 100 digits.

Find the product of these prime numbers: N = P1 * P2. Compute another number, Q = (P1 - 1) * (P2 - 1).

Find a number DECRYPT such that GCD(DECRYPT,Q)=1, i.e. pick DECRYPT so that it is relatively prime to Q.

Find ENCRYPT such that ENCRYPT * DECRYPT = 1 mod Q.

In order to encrypt data, it is split into blocks of N or less bit-equivalents, and compute

SECRET = MESSAGE BITS ^ ENCRYPT

Where ^ refers to exponentiation modulo Q. To decrypt the encrypted message, compute

RECEIVED = SECRET ^ DECRYPT

The public key then consists of both ENCRYPT and N, and the private key DECRYPT and N. Given that finding large prime numbers is a formidable task, finding a suitable DECRYPT given all the other parameters of the system is a non-trivial task. In fact, one of the inventors of the RSA algorithm commented that factoring a 500 bit binary number would take approximately 10^{25} years.

5.2 The Data Encryption Standard

The principle behind this standard is to use a simple cryptographic method (in this case bit permutation) many times in succession to encrypt the data. This philosophy of applying a simple method many times is contrasted by the method used in the RSA algorithm described elsewhere in this chapter. If a number of data bits (say 3) are used to activate one of eight outputs ($2^3 = 8$) and those eight bits are permuted before being re-encoded as a three bit number, a cipher can be built simply and easily. The DES standard uses this method, with a series of 16 key-dependent and 3 key-independent transpositions operating on 64 bits of data. The three key-independent transposition are:

The first transposition.

The next to last transposition, which exchanges the upper and lower 32 bits.

The last transposition, which is the exact inverse of the first.

The operation of the 16 intermediate stages involves taking the upper and lower 32 bits, setting the upper 32 bit output to be the lower 32 bits input, and the lower 32 bits output to be the exclusive-or of the upper 32 bits input and a function of the lower 32 bits input. This is illustrated in Figure 29 below.

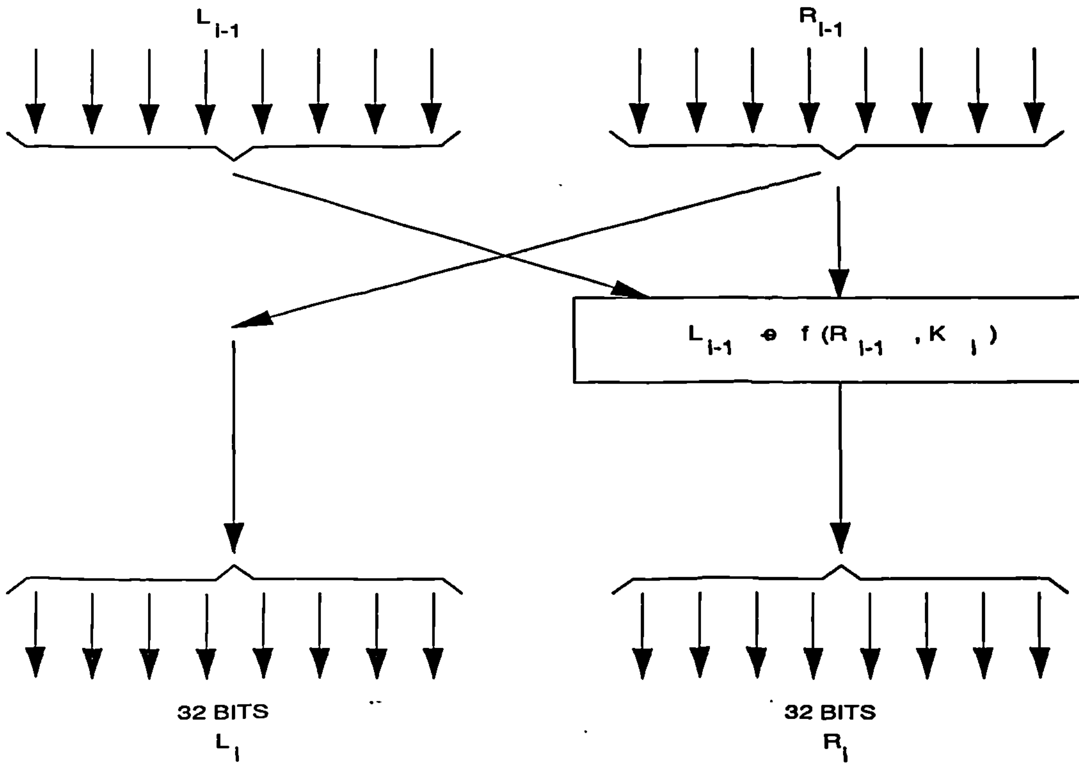


Figure 29 : Data Encryption Standard

The secret to DES lies in the function which acts on one 32 bit input. This function operates in five parts, as shown in the table below:

A 48 bit number (N) is derived from the 32 bit input by applying a fixed transposition and duplication rule

and the stage-key are exclusive-or'ed together

The result is partitioned into eight groups of 6 bits.

Each 6 bit word is permuted 4 bits

The eight groups of four bits are permuted together to produce a 32 bit result.

The stage-key is derived from the master key as follows:

A 56 bit transposition is applied to the key

The result of this is split into two 28 bit parts, and these parts are logically rotated a number of bits n, which depends on the iteration number

The result is again permuted as 56 bits and becomes the final 48 bit stage key.

Despite the apparent complexity, DES is a single alphabet encryption method, which is breakable in a finite time using modern computers.

5.3 The Knapsack Algorithm

This method of encryption relies on an analogy for its explanation. If we imagine a set of objects, with specified weights, then a bag containing a subset of the objects will have a total weight equal to their combined weight. It is exceedingly difficult to determine which objects are in the bag, given the total weight of the bag and the weights of the individual objects.

In order to use this method for encryption, a message is split into a number of bits, and each bit multiplied by a binary weight W. The encoded message is the sum of the weighted values of each of the bits. If each bit weight W is larger than the sum of all the previous weights, the coded message can be decrypted by subtracting the largest weight less than the current total weight. The code user then chooses two large numbers such that:

GCD(N1,N2) = 1

and the weight vector is chosen so that

WEIGHTVEC = N2 * W (mod N1)

This Weight vector is published, and data encrypted by computing

ENCRYPT = WEIGHTVEC * DATA

To decrypt the data, the receiver computes

RECEIVED = 1/N2 * ENCRYPT (mod N1)

APPENDIX F

SIMULATOR LISTINGS

```
#include <stdio.h>
#define TRUE 1
#define FALSE 0
#define max(a,b) (((a)>(b))?(a):(b))
#define min(a,b) (((a)<(b))?(a):(b))
struct transition {
   struct transition  *next;
   double  t_amplitude,
         t_time;
};
struct gen {
   unsigned  *mask,
           *shift_reg,
           length;
   double  clock_period,
         amplitude;
};
struct event {
   struct event  *nxt_event;
   int     get_event;
   struct gen *source;
   double  event_time,
         amplitude;
};
/*@(#)sim.c    1.6*/
#include <math.h>
#include "sim.h"
/*-
This program simulates the operation of a communications
channel connected to a number of spread spectrum sequence
generators or sources.
The number of seperate sequences is not limited,
neither is the length, amplitude and clock period of any one
sequence.
The simulation is event driven, on a demand basis.
This means that new events are generated only when needed, as opposed to
for all time.
This slows the simulation down somewhat, but allows
very long effective duration of simulation (potentially infinite).
The program offers the choice of either a perfect integrator at the
receiver, or more realistically a sampling receiver.
If the number of samples is set at 1, then the receiver becomes hard limited.
The simulator can be run either from the command line or from a data file.
The mode of operation is determined from the command line itself, i.e. if
there is a source file then non-interactive mode is chosen.
In interactive mode, the program prompts for the following information
for each transmitted sequence:
sequence polynomial in octal <octal integer>
sequence start time <double real>
sequence clock period <double real>
sequence amplitude <double real>
another sequence? <y|n>
This is repeated for as long as the answer to the last question is y, then
the following questions are asked:
receiver polynomial in octal <octal integer>
```

```
 receiver clock period <double real>
 simulation duration <double real>
 The results of the simulation are output either on the standard output
 or to a output file specified on the command line
 */
 /*- global variables */
FILE * s_file, *d_file;
struct event   *event_header;
int     do_integrate,
        interactive;
/*  receiver registers */
int     r_length,
       *r_sequence,
       *r_reference;
/* receiver clock, total simulation time and global amplitude */
double  r_clock,
        s_time,
        g_amplitude = 0.0;
/*- memory allocation primitives for the two types of structure */
struct gen *galloc () {
                return ((struct gen *) malloc (size of (struct gen)));
};
struct event   *ealloc () {
                    return ((struct event  *) malloc (size of (struct event)));
};
/*- used when EOF is prematurely read */
void read error () {
    fprintf (stderr, "Unable to read file: %s\n", s_file);
    exit (2);
};
/*- simple integer arithmetic routines necessary for spread spectrum sequences */
int     two_power (power)
int     power;
{
    int     count;
    int     i = 1;
    for (count = 1; count != power + 1; count++)
        i *= 2;
    return (i);
};
int     exor (val1, val2)
int     val1,
        val2;
{
    switch (val1) {
        case 0:
            if (val2 == 0)
                return (0);
            else
                return (1);
        default:
            if (val2 == 0)
                return (1);
            else
                return (0);
    }
};
/*
 inserts an event in the event list, pointed to by event_header. Sets the
 pointer to the generator, the event time, amplitude etc.
```

```
  on entry:
  initiator:   pointer to the generator that caused this event
  time:        time at which the transition is to occur
  amplitude:   amplitude of the transition
  more:        true if there are more transitions to be
  generated for this bit, false otherwise

  on exit:
  the global variable event_header will point to the modified
  event list with the new event inserted in chronological order

  global variables referenced:
  event_header

  global variables altered:

  functions called:
  */
void insert_event (initiator, time, amplitude, more)
struct gen *initiator;
int    more;
double  time,
        amplitude;
{
   struct event   *first,
                  *next;
/* start at the head of the event list */
   first = next = event_header;
/* step through events until the correct time slot is found */
   for (;;) {
   /* insert events in chronological order */
      if (next -> event_time > time) {
         next = first;
         break;
      }
   /* unless at the end of the event list */
      if (next -> nxt_event == NULL)
         break;
   /* otherwise check the next event in the list */
      first = next;
      next = next -> nxt_event;
   }
/* insert the new event */
   first = next -> nxt_event;
   next -> nxt_event = ealloc ();
   next -> nxt_event -> nxt_event = first;
   next = next -> nxt_event;
/* fill in the fields of the event record */
   next -> event_time = time;
   next -> amplitude = amplitude;
   next -> get_event = more;
   next -> source = initiator;
};
/*
 insert a new generator, with corresponding first event in the event list.
 This routine is used in the read_params function
     on entry:
     on exit:
        adds the first event associated with a generator to the
        event list. All parameters of the generator and first event
        are set up as read in.
        global variables referenced:
           s_file
        global variables altered:
        functions called:
           galloc()
           calloc()
           insert_event
*/
void insert_gen () {
   struct gen *this;
   int    val,
          len;
   unsigned    poly;
/* get an uninitialised structure */
   this = galloc ();
/* read in the feedback taps for the shift register */
   if (interactive == TRUE)
      fprintf (stderr, "enter generator polynomial: ");
   fscanf (s_file, "%o", &poly);
   len = (int) floor (log ((double) poly) / log (2.0));
   poly /= 2;
   this -> shift_reg = (unsigned *) calloc ((unsigned) len,
         size of (unsigned));
   this -> mask = (unsigned *) calloc ((unsigned) len,
         size of (unsigned));
/* initialise the shift register to all 1's (see below) */
   for (val = 0; val < len; val++)
      this -> shift_reg[val] = 1;
/* read in the taps */
   for (val = 0; val < len; val++) {
      this -> mask[val] = poly % 2;
      poly /= 2;
   }
/* read in the rest of the fields */
   this -> length = len;
   if (interactive == TRUE)
      fprintf (stderr, "start time: ");
   if (fscanf (s_file, "%lf", &s_time) == EOF)
      read error ();
   if (interactive == TRUE)
      fprintf (stderr, "clock period: ");
   if (fscanf (s_file, "%lf", &(this -> clock_period)) == EOF)
      readerror ();
   if (interactive == TRUE)
      fprintf (stderr, "amplitude: ");
   if (fscanf (s_file, "%lf", &(this -> amplitude)) == EOF)
      readerror ();
/* insert the first transition into the event list. This line of code
   assumes that all sequences start with a 1 (see above), hence the 1/2
   amplitude */
   insert_event (this, s_time, this -> amplitude / 2, TRUE);
};
/*
  read the communication channel specification, including as many transmitters
  as are wanted, and the receiver spec.
        on entry:
        on exit:
           has set up the data structures necessary for the simulation
        global variables referenced:
           event_header
           s_file
```

```
    global variables altered:
      r_length
      r_sequence
      r_reference
    functions called:
      ealloc
      calloc
      two_power
      exor
      cfree
*/
read_params () {
  int    i,
         j,
         bit,
         r_poly,
         *r_shiftreg,
         *r_mask;
  char   reply;
/* insert a dummy event at zero time into the event list */
  event_header = ealloc ();
  event_header -> get_event = FALSE;
  event_header -> event_time = 0.0;
  event_header -> amplitude = 0.0;
/* read sequence parameters until no more are required */
  do {
    insert_gen ();
    if (interactive == TRUE)
      fprintf (stderr, "Another sequence?: ");
  } while (fscanf (s_file, "%1s", &reply) != EOF &&
      reply == 'y');
/* read in the feedback taps etc. for the receiver, and initialise the
  receiver registers */
  if (interactive == TRUE)
    fprintf (stderr, "Enter the receiver generator polynomial: ");
  if (fscanf (s_file, "%o", &r_poly) != EOF) {
    r_length = (int) floor (log ((double) r_poly) / log (2.0));
    r_shiftreg = (int *) calloc ((unsigned) r_length, sizeof (int));
    r_mask = (int *) calloc ((unsigned) r_length, sizeof (int));
    r_poly /= 2;
  /* read in the taps */
    for (i = 0; i < r_length; i++) {
      r_mask[i] = r_poly % 2;
      r_poly /= 2;
    }
  }
  else
    readerror ();
/* set up the reference sequence in the receiver correlator and
  initialise the receiver to all 0's (not strictly speaking necessary)
*/
  r_reference = (int *) calloc ((unsigned) two_power (r_length) - 1,
        sizeof (int));
  r_sequence = (int *) calloc ((unsigned) two_power (r_length) - 1,
        sizeof (int));
  for (i = 0; i < two_power (r_length) - 1; i++)
    r_sequence[i] = 0;
/* start off with all 1's in the shift register (see earlier) */
  for (i = 0; i < r_length; i++)
    r_shiftreg[i] = 1;
/* generate the reference */
```

```
    for (j = 0; j < two_power (r_length) - 1; j++) {
      bit = r_reference[j] = r_shiftreg[r_length - 1];
      for (i = r_length - 2; i >= 0; i--)
        if (r_mask[i] == 1)
          bit = exor (bit, r_shiftreg[i]);
      for (i = r_length - 1; i > 0; i--)
        r_shiftreg[i] = r_shiftreg[i - 1];
      r_shiftreg[0] = bit;
    }
  /* dispose of all unwanted space */
    cfree ((char *) r_mask);
    cfree ((char *) r_shiftreg);
  /* read in the remaining parameter */
    if (interactive == TRUE)
      fprintf (stderr, "Enter the receiver clock period: ");
    if (fscanf (s_file, "%lf", &r_clock) == EOF)
      readerror ();
 };
 /*  use the information in an event and its corresponding source to
    generate the next voltage transition, by sequencing the shift
    register through states until a change in output is observed
      on entry:
        eptr:      pointer that is to be referenced
      on exit:
        leaves a new
        event in the event list at the time of the next change in output of
        the sequence contained in the generator
      global variables referenced:
      global variables altered:
      functions called:
        exor
        insert_event
*/
new_event (eptr)
struct event   *eptr;
{
  struct gen *gptr;
  double  time;
  unsigned    bit,
              obit;
  int     i;
/* generate a new event if the flag is TRUE. This will always be the
  case, unless Manchester coding or some variant is used which results
  in multiple transitions per bit */
  if (eptr -> get_event == TRUE) {
    time = eptr -> event_time;
    gptr = eptr -> source;
  /* generate bits from the shift register */
    do {
      bit = obit = gptr -> shift_reg[gptr -> length - 1];
    /* exor bits corresponding to those set in the mask register */
      for (i = 0; i < gptr -> length - 1; i++)
        if (gptr -> mask[i] == 1)
          bit = exor (bit, gptr -> shift_reg[i]);
    /* shift */
      for (i = gptr -> length - 1; i > 0; i--)
        gptr -> shift_reg[i] = gptr -> shift_reg[i - 1];
      gptr -> shift_reg[0] = bit;
    /* step the time period and save the new output value */
      bit = gptr -> shift_reg[gptr -> length - 1];
      time += gptr -> clock_period;
```

```
        } while (bit == obit);
    /* having found the event time, insert the transition */
        insert_event (gptr, time,
            bit == 0 ? -(gptr -> amplitude) : gptr -> amplitude, TRUE);
        eptr -> get_event = FALSE;
    }
};
/*
        sample the output waveform a given number of times, and decide on a
        majority basis whether a bit is 1 or 0. If a single sample is used
        then reception is hard limited.
        on entry:
            clock:         receiver clock period
            start_time:    start of sample time period
            n_samples:     number of samples
        on exit:
            returns the bit value of the bit period
        global variables referenced:
            event_header
            g_amplitude
        global variables altered:
            g_amplitude
            event_header
        functions called:
            new_event
            cfree
*/
int    sample (clock, start_time, n_samples)
double  clock,
        start_time;
int    n_samples;
{
    struct event   *even;
    int    sample,
           j;
    double  t_inc,
            t;
    even = event_header;
    t_inc = clock / n_samples;
    sample = (g_amplitude >= 0.0 ? 1 : -1);
    t = start_time;
    for (j = 1; j < n_samples; j++) {
        while ((t + t_inc >= even -> event_time)) {
            g_amplitude += even -> amplitude;
            new_event (even);
            event_header = even -> nxt_event;
            cfree ((char *) even);
            even = event_header;
        }
        t += t_inc;
        sample += (g_amplitude >= 0.0 ? 1 : -1);
    }
    while (even -> event_time <= start_time + clock) {
        g_amplitude += even -> amplitude;
        new_event (even);
        event_header = even -> nxt_event;
        cfree ((char *) even);
        even = event_header;
    };
    return (sample >= 0 ? 1 : 0);
};
```

```
/*
perform a continuous integration over one receiver bit period. This
type of reception is ideal, as opposed to the sampling method used
above which is quantised
    on entry:
        clock:      denotes the time period over which the
                    integration is calculated.
        start_time:    denotes the starting time for the integration
    on exit:
        returns the value of the single bit based on a simple
        quantisation of receiving a 1 if the integral is >=1,
        otherwise retuning a zero.
    global variables referenced:
        event_header
        g_amplitude
    global variables altered:
        event_header
        g_amplitude
    functions called:
        new_event
        cfree
*/
int    integrate (clock, start_time)
double  clock,
    start_time;
{
   struct event   *even;
   double  integral_sum,
        t,
        t_last;
 /* set up the initial conditions */
   even = event_header;
   integral_sum = 0.0;
   t = start_time;
 /* do this for all events before the end of this time slot including any
   transitions that occur exactly at the end of the integration period */
   while ((even -> event_time) <= (start_time + clock)) {
      t_last = t;
      t = even -> event_time;
   /* increment the integral */
      integral_sum += g_amplitude * (t - t_last);
   /* increment the global amplitude */
      g_amplitude += even -> amplitude;
   /* and dispose of this transition */
      new_event (even);
      event_header = even -> nxt_event;
      cfree ((char *) even);
      even = event_header;
    }
    integral_sum += g_amplitude * (start_time + clock - t);
    return (integral_sum >= 0.0 ? 1 : 0);
 };
 main (argc, argv)
 char   argc,
      **argv;
 {
    int    csum,
         i,
         n_samples;
    char   *s;
    double
```

```
         s_time,
         time;
   interactive = TRUE;
   r_clock = 0.0;
   s_file = stdin;
   d_file = stdout;
/*- parse the command line invocation for parameters */
   switch (argc) {
      case 4:
         if ((d_file = fopen (argv[3], "w")) == NULL) {
            fprintf (stderr, "Can't open %s\n", argv[3]);
            exit (1);
         }
      case 3:
         if ((s_file = fopen (argv[2], "r")) == NULL) {
            fprintf (stderr, "Can't open %s\n", argv[2]);
            exit (1);
         }
         interactive = FALSE;
      case 2:
         s = argv[1];
         switch (*s) {
            case 'i':
               do_integrate = TRUE;
               break;
            case 's':
               do_integrate = FALSE;
               break;
            default:
               fprintf (stderr,
                     ":Usage: sim i/s {<sourcefile>} {<plotfile>}\n");
               exit (2);
         }
         break;
      default:
         fprintf (stderr,
               "Usage: sim i/s {<sourcefile>} {<plotfile>}\n");
         exit (2);
   }
/*- read in the remaining global variables */
   if (interactive == FALSE)
      fprintf (stderr, "Reading simulation parameters\n");
   read_params ();
   if (interactive == TRUE)
      fprintf (stderr, "Enter simulation time: ");
   if (fscanf (s_file, "%lf", &s_time) == NULL)
      readerror ();
   if (do_integrate == FALSE) {
      if (interactive == TRUE)
         fprintf (stderr, "Enter number of samples: ");
      if (fscanf (s_file, "%d", &n_samples) == NULL)
         readerror ();
   }
   time = 0.0;
/* main program loop */
   while (time < s_time) {
   /* shift the received sequence and read in the next bit */
      for (i = 0; i < two_power (r_length) - 2; i++)
         r_sequence[i] = r_sequence[i + 1];
      r_sequence[two_power (r_length) - 2] =
         do_integrate == TRUE ? integrate (r_clock, time) :
```

```
      sample (r_clock, time, n_samples);
  /* calculate the correlation sum */
    csum = 0;
    for (i = 0; i < two_power (r_length) - 1; i++)
      if (r_reference[i] == r_sequence[i])
        csum++;
  /* output the correlation value for further processing or display */
    fprintf (d_file, "%d\n", csum);
  /* move on to the next chip period */
    time += r_clock;
  };
}
```

APPENDIX G

GLOSSARY OF SIMULATOR FUNCTIONS

7.1 Main

This part of the programme simulates the operation of a communications channel connected to a number of spread spectrum sequence generators or sources.

The number of separate sequences is not limited, neither is the length, amplitude and clock period of any one sequence. The simulation is event driven, on a demand basis. This means that new events are generated only when needed, as opposed to for all time. This slows the simulation down somewhat, but allows very long effective duration of simulation (potentially infinite). The programme offers the choice of either a perfect integrator at the receiver, or more realistically a sampling receiver. If the number of samples is set at 1, then the receiver becomes hard limited.

7.2 Insert Event

Inserts an event in the event list, pointed to by event_header. Sets the pointer to the generator, the event time, amplitude etc.

On Entry	initiator	pointer to the generator that caused this event
	time	time at which the transition occurred
	amplitude	amplitude of the transition
	more	true if there are more transitions to be generated for this bit, false otherwise
On Exit		the global variable event_header will point to the modified event list with the new event inserted in chronological order
Global Variables Referenced	event_header	
Global Variables Altered		
Functions Called		

7.3 Insert Gen

Insert a new generator, with corresponding first event in the event list. This routine is used in the read_params function

On Entry		
On Exit		adds the first event associated with a generator to the event list. All parameters of the generator and first event are set up as read in
Global Variables Referenced	s_file	
Global Variables Altered		
Functions Called	galloc calloc insert_event	

7.4 Read Params

Read the communication channel specification, including as many transmitters as are wanted, and the receiver spec.

On Entry		
On Exit		Has set up the data structures necessary for the simulation
Global Variables Referenced	event_head s_file	
Global Variables Altered	r_length r_sequence r_reference	
Functions Called	ealloc calloc two_power exor cfree	

7.5 Next Event

Use the information in an event and its corresponding source to generate the next voltage transition, by sequencing the shift register through states until a change in output is observed

On Entry	eptr	Pointer that is to be referenced
On Exit		Leaves a new event in the event list at the time of the next change in output of the sequence contained in the generator
Global Variables Referenced		
Global Variable Altered		
Functions Called	exor insert_event	

7.6 Sample

Sample the output waveform a given number of times, and decide on a majority basis whether a bit is 1 or 0. If a single sample is used then reception is hard limited.

7.7 Integrate

Perform a continuous integration over one receiver bit period. This type of reception is ideal, as opposed to the sampling method used above which is quantised.

APPENDIX H

SIMULATED ANNEALING LISTING

The simulated annealing program is included in this appendix. It was written on an IBM AT using a proprietary C compiler which conforms to the ANSI C standard. The first file is a sourcefile for the "make" utility which combines the other files to create the executable code. The program has been written to be portable and should run on any machine using unix.

8.1 File Anneal

```
CFLAGS=/I..\crecipies /c /Dfloat=double /Dran1=ran3
F=..\crecipie\files

four1.obj: $(F)\four1.c
   cl $(CFLAGS) $(F)\four1.c

realft.obj: $(F)\realft.c four1.obj
   cl $(CFLAGS) $(F)\realft.c

twofft.obj: $(F)\twofft.c four1.obj
   cl $(CFLAGS) $(F)\twofft.c

correl.obj: $(F)\correl.c twofft.obj realft.obj
   cl $(CFLAGS) $(F)\correl.c

ran3.obj: $(F)\ran3.c
   cl $(CFLAGS) $(F)\ran3.c

irbit1.obj: $(F)\irbit1.c
   cl $(CFLAGS) $(F)\irbit1.c

nrutil.obj: ..\crecipie\nrutil.c
   cl $(CFLAGS) ..\crecipie\nrutil.c

util.obj: util.c anneal.h
   cl $(CFLAGS) util.c

scramble.obj: scramble.c ran3.obj anneal.h
   cl $(CFLAGS) scramble.c

energy.obj: energy.c correl.obj util.obj anneal.h
   cl $(CFLAGS) energy.c
```

```
output.obj: output.c util.obj anneal.h
   cl $(CFLAGS) output.c

anneal.obj: anneal.c nrutil.obj irbit1.obj ran3.obj util.obj output.obj scramble.obj energy.obj anneal.h
   cl $(CFLAGS) anneal.c

anneal.exe: anneal.obj
   cl /I..\crecipies /Feanneal.exe *.obj
```

8.2 File Anneal.c

```
/* first attempt at a simulated annealing program for designing cdma
code sets */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "anneal.h"

int          idum = -1;

int metrop(de,t)
double de,t;
{
             static int gljdum=1;

             return de < 0.0 || ran3(&gljdum) < exp(-de/t);
}

int          main (int argc, char *argv[]) {
   int    i,
                j,
                length,
                no_of_codes,
                ntries,
                try_limit,
          successes,
          success_limit;

   unsigned long int   iseed;

   double  energy,
          energy_prev,
          de,
          temperature,
          cooling_factor,
                **sequence,
         **backup;

   FILE   *s_file,
                *code_file,
                *plot_file;
```

```
/* read in the program variables */
   s_file = stdin;
   code_file = stdout;
   plot_file = stdout;


/*- parse the command line invocation for parameters */
   switch (argc) {
            case 3:
               if ((plot_file = fopen (argv[2], "w")) == NULL) {
                        fprintf (stderr, "Can't open %s\n", argv[2]);
                        exit (1);
               }
               fprintf(stderr,"Opened %s as plot file\n",argv[2]);
            case 2:
               if ((code_file = fopen (argv[1], "w")) == NULL) {
                        fprintf (stderr, "Can't open %s\n", argv[1]);
                        exit (1);
               }
               fprintf(stderr,"Opened %s as code file\n",argv[1]);
            case 1:
               break;
            default:
               fprintf (stderr,
                                   "Usage: anneal {<codefile>} {<plotfile>}\n");
               exit (2);
   }
   fprintf (stderr, "enter the code length: ");
   fscanf (s_file, "%d", &length);
   fprintf (stderr, "enter the code set size: ");
   fscanf (s_file, "%d", &no_of_codes);


/* initialise the data structures */
   cooling_factor = 0.9;
   energy_prev = (double) length;
   successes = 0;
   ntries = 0;
   temperature = (double) length;
   success_limit = (int) floor (-log ((double) length) / log (cooling_factor));
   try_limit = length;
   sequence = dmat (1, no_of_codes, 1, near2(length));


 /* initialise the sequence set to be completely random on the first try
   of this algorithm */
   for (i = 1; i <= no_of_codes; i++)
            for (j = 1; j <= length; j++)
               sequence[i][j] = (irbit1 (&iseed) == 1 ? 1.0 : -1.0);


/* main program loop */
   do {
            /* save a copy of the current version of the sequence in case the
               scrambled copy fails the metrop test */
               backup = dmat (1, no_of_codes, 1, near2(length));
               for (i = 1; i <= no_of_codes; i++)
                        for (j = 1; j <= length; j++)
                           backup[i][j] = sequence[i][j];


   /* scramble the sequences and then calculate the new energy value */
            i = scramble_sequences (temperature,sequence,no_of_codes,length);
#if defined(DEBUG)
            fprintf(code_file,"number:%d\t",i);
#endif

            energy = calculate_energy (sequence,no_of_codes,length);

  /* decide whether to update the sequence set based on the current
     value of temperature and the improvement in energy */
            de = energy - energy_prev;

#if defined(DEBUG)
            fprintf (code_file,
                        "e: %7.4lf\tde: %7.4lf\ttemp: %7.4lf\t",
                        energy, de, temperature);
#endif

            if (metrop (5.0 * de, temperature))
            {
              successes++;
              ntries = 0;
              temperature *= cooling_factor;
              energy_prev = energy;
              free_dmatrix( backup, 1, no_of_codes, 1, near2(length));
            }
            else
            {
              ntries++;
              free_dmatrix( sequence, 1, no_of_codes, 1, near2(length));
              sequence = backup;
            }
            if (ntries >= try_limit)
            {
              successes++;
              ntries = 0;
              temperature *= cooling_factor;
            }

#if defined(DEBUG)
            fprintf (code_file, "\n");
#endif

  }

/* if we have tried to improve this one and it doesn't work then give up */
  while (ntries < length && successes < success_limit);

/* now output the results */
  fprintf (code_file, "the figure of merit is: %lf", energy_prev);
  for (i = 1; i <= no_of_codes; i++) {
            fprintf (code_file, "\nsequence %d: ", i);
            for (j = 1; j <= length; j++)
              fprintf (code_file, "%d", (sequence[i][j] == 1.0 ? 1 : 0));
  }
  fprintf (code_file, "\n");
```

```
/* output the correlation values */
  output_results(sequence,no_of_codes,length,plot_file);

  return (1);
}
```

8.3 File Scramble.c

```
/* scramble up the sequences in the two dimensional array sequence. The
amount of scrambling ot be done is based on the temperture. High temp
means more scrambling */


#include <math.h>
#include "anneal.h"


int scramble_sequences (double temperature, double **sequence,
                             int no_of_codes, int length)
{
/* scrambling is done on the basis of getting a portion of the full
    sequence and inverting it. This is done repeatedly, the number of
    times being dependent on the annealing temperature. The position
    and length of each portion are determined randomly */

extern int idum;

  int    i,
            iter = 0,
            j;

          for (i = 1; i <= no_of_codes; i++) {
            for (j = 1; j <= length; j++) {
                   if (ran3(&idum)<temperature/(2.0*(double)length)){
                     iter++;
                     sequence[i][j] = (sequence[i][j] == 1.0 ? -1.0 : 1.0);
                   }
            }
          }
  return(iter);

}
```

8.4 File Energy.c

```
/* routine to calculate the energy for the simulated annealing program
based on the auto and cross correlation properties of the code set.
there are two global variables, length and no_of_codes. length is the size
of the sequences, and no_of_codes is the desired code set size */


#include <math.h>
#include "anneal.h"
#include <stdio.h>
```

```
double      calculate_energy (double **sequence, int no_of_codes, int length) {

    int    i,
           j,
           k;

    double *ans,
                   *a,
                   *b,
           thap;

    thap = 0.0;
    ans = dvec             (1, 4 * near2(length));
    a = dvec  (1, 2 * near2(length));
    b = dvec  (1, 2 * near2(length));

    /* find the <1,11> autocorrelation */
    for (i = 1; i <= no_of_codes; i++) {
              correl (sequence[i],sequence[i],near2(length),ans);
              for (j = 2; j <= length; j++)
                 if (fabs (ans[j]) > thap) thap = fabs (ans[j]);
    }

    /* find the <1,10> autocorrelation */
    for (i = 1; i <= no_of_codes; i++) {
              for (j = 1; j <= length; j++) {
                 a[j] = sequence[i][j];
                 a[j + length] = -sequence[i][j];
                 b[j] = sequence[i][j];
                 b[j + length] = 0.0;
              }
              correl (a, b, 2 * near2(length), ans);
              for (j = 2; j <= length; j++) {
                 if (fabs (ans[j]) > thap) thap = fabs (ans[j]);
                 if (fabs (ans[j + length]) > thap) thap = fabs (ans[j + length]);
              }
    }

    /* do the <1,11> cross correlation */
    for (i = 1; i <= no_of_codes; i++) {
              for (j = 1; j <= no_of_codes; j++) {
                 if (i != j) {
                           correl (sequence[i],sequence[j],near2(length),ans);
                           for (k = 1; k <= length; k++)
                              if (fabs (ans[k]) > thap)
                                        thap = fabs (ans[k]);
                 }
              }
    }

    /* do the <1,10> cross correlation */
    for (i = 1; i <= no_of_codes; i++) {
              for (j = 1; j <= no_of_codes; j++) {
                 if (i != j) {
                           for (k = 1; k <= length; k++) {
                              a[k] = sequence[i][k];
                              a[k + length] = -sequence[i][k];
```

```
                      b[k] = sequence[j][k];
                      b[k + length] = 0.0;
                    }
                    correl (a, b, 2 * near2(length), ans);
                    for (k = 1; k <= length; k++) {
                      if (fabs (ans[k]) > thap)
                              thap = fabs (ans[k]);
                      if (fabs (ans[k + length]) > thap)
                              thap = fabs (ans[k + length]);
                    }
            }
          }
   }


   free_dvector  (ans, 1, 4*near2(length));
   free_dvector (a, 1, 2*near2(length));
   free_dvector (b, 1, 2*near2(length));

   return thap;

}
```

8.5 File Output.c

```
#include <math.h>
#include <stdio.h>
#include "anneal.h"

void  output_results (double **sequence, int no_of_codes, int length, FILE *plot_file) {

  int    i,
      j,
           k;

  double *ans,
           *a,
           *b,
           *m;

  ans = dvec            (1, 4 * near2(length));
  a = dvec  (1, 2 * near2(length));
  b = dvec  (1, 2 * near2(length));
  m = dvec  (1, 2 * near2(length));

  /* print the <1,11> correlation */
  for (i = 1; i <= no_of_codes; i++) {
          for (j = 1; j <= no_of_codes; j++) {
                     correl (sequence[i],sequence[j],near2(length),ans);
                     for (k = 1; k <= length; k++)
                        if(fabs(ans[k]) > fabs(m[k])) m[k] = ans[k];
          }
  }
  for(k=1;k<=length;k++)
          fprintf(plot_file,"%d ",(int)m[k]);
  fprintf(plot_file,"\n");
```

```
    for(k=1;k<=length;k++)
            m[k]=0.0;

    /* print the <1,10> correlation */
    for (i = 1; i <= no_of_codes; i++) {
            for (j = 1; j <= no_of_codes; j++) {
                        for (k = 1; k <= length; k++) {
                          a[k] = sequence[i][k];
                          a[k + length] = -sequence[i][k];
                          b[k] = sequence[j][k];
                          b[k + length] = 0.0;
                        }
                        correl (a, b, 2 * near2(length), ans);
                        for(k=1;k<=2*length;k++)
                          if(fabs(ans[k])>fabs(m[k])) m[k] = ans[k];
            }
    }

    for(k=1;k<=2*length;k++)
            fprintf(plot_file,"%d ",(int)m[k]);
    fprintf(plot_file,"\n");

    free_dvector (ans, 1, 4*near2(length));
    free_dvector (a, 1, 2*near2(length));
    free_dvector (b, 1, 2*near2(length));
    free_dvector (m, 1, 2*near2(length));

    return;
  }
```

8.6 File Util.c

```
  #include <math.h>
  #include <nrutil.h>
  #include "anneal.h"

  double *dvec(int low, int high) {
    int i;
    double *ptr;
    ptr=dvector(low,high);
    for(i=low;i<=high;i++)
            ptr[i]=0.0;
    return ptr;
  }

  double **dmat(int lowrow, int highrow, int lowcol, int highcol) {
    int i,j;
    double **ptr;
    ptr=dmatrix(lowrow,highrow,lowcol,highcol);
    for(i=lowrow;i<=highrow;i++)
            for(j=lowcol;j<=highcol;j++)
              ptr[i][j]=0.0;
    return ptr;
  }
```

```
int near2(int num) {
   return (int) pow(2.0,ceil(log( (double) num)/log(2.0) ));
}
```

8.7 File Anneal.h

```
#include "nrutil.h"
#include <stdio.h>


#define DEBUG
#if defined (MSDOS)
extern      int main(int argc, char *argv[]);
extern  void correl(double *data1,double *data2,int n,double *ans);
extern      double calculate_energy(double **sequence,int no_of_codes,int length);
extern  void four1(double *data,int nn,int isign);
extern  int irbit1(unsigned long *iseed);
extern  int metrop(double de,double t);
extern  double ran3(int *idum);
extern      void realft(double *data,int n,int isign);
extern      double *dvec(int low, int high);
extern      double **dmat(int lowrow, int highrow, int lowcol, int highcol);
extern      int near2(int num);
extern      int scramble_sequences(double temperature,double **sequence,int no_of_codes, int
length);
extern      void twofft(double *data1,double *data2,double *fft1,double *fft2,int n);
extern      void output_results(double **sequence,int no_of_codes,int length, FILE *plot_file);
#else
int main();
void correl();
double calculate_energy();
void four1();
int irbit1();
int metrop();
double ran3();
void realft();
double *dvec();
double **dmat();
int near2();
int scramble_sequences();
void twofft();
void output_results();
#endif
```

8.8 File Four1.c

```
#include <math.h>


#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr


void four1(data,nn,isign)
float data[];
int nn,isign;
{
    int n,mmax,m,j,istep,i;
    double wtemp,wr,wpr,wpi,wi,theta;
    float tempr,tempi;

    n=nn << 1;
    j=1;
    for (i=1;i<n;i+=2) {
        if (j > i) {
            SWAP(data[j],data[i]);
            SWAP(data[j+1],data[i+1]);
        }
        m=n >> 1;
        while (m >= 2 && j > m) {
            j -= m;
            m >>= 1;
        }
        j += m;
    }
    mmax=2;
    while (n > mmax) {
        istep=2*mmax;
        theta=6.28318530717959/(isign*mmax);
        wtemp=sin(0.5*theta);
        wpr = -2.0*wtemp*wtemp;
        wpi=sin(theta);
        wr=1.0;
        wi=0.0;
        for (m=1;m<mmax;m+=2) {
            for (i=m;i<=n;i+=istep) {
                j=i+mmax;
                tempr=wr*data[j]-wi*data[j+1];
                tempi=wr*data[j+1]+wi*data[j];
                data[j]=data[i]-tempr;
                data[j+1]=data[i+1]-tempi;
                data[i] += tempr;
                data[i+1] += tempi;
            }
            wr=(wtemp=wr)*wpr-wi*wpi+wr;
            wi=wi*wpr+wtemp*wpi+wi;
        }
        mmax=istep;
    }
}

#undef SWAP
```

8.9 File Realft.c

```
#include <math.h>

void realft(data,n,isign)
float data[];
int n,isign;
{
    int i,i1,i2,i3,i4,n2p3;
```

```
        float c1=0.5,c2,h1r,h1i,h2r,h2i;
        double wr,wi,wpr,wpi,wtemp,theta;
        void four1();


        theta=3.141592653589793/(double) n;
        if (isign == 1) {
                c2 = -0.5;
                four1(data,n,1);
        } else {
                c2=0.5;
                theta = -theta;
        }
        wtemp=sin(0.5*theta);
        wpr = -2.0*wtemp*wtemp;
        wpi=sin(theta);
        wr=1.0+wpr;
        wi=wpi;
        n2p3=2*n+3;
        for (i=2;i<=n/2;i++) {
                i4=1+(i3=n2p3-(i2=1+(i1=i+i-1)));
                h1r=c1*(data[i1]+data[i3]);
                h1i=c1*(data[i2]-data[i4]);
                h2r = -c2*(data[i2]+data[i4]);
                h2i=c2*(data[i1]-data[i3]);
                data[i1]=h1r+wr*h2r-wi*h2i;
                data[i2]=h1i+wr*h2i+wi*h2r;
                data[i3]=h1r-wr*h2r+wi*h2i;
                data[i4] = -h1i+wr*h2i+wi*h2r;
                wr=(wtemp=wr)*wpr-wi*wpi+wr;
                wi=wi*wpr+wtemp*wpi+wi;
        }
        if (isign == 1) {
                data[1] = (h1r=data[1])+data[2];
                data[2] = h1r-data[2];
        } else {
                data[1]=c1*((h1r=data[1])+data[2]);
                data[2]=c1*(h1r-data[2]);
                four1(data,n,-1);
        }
}
```

8.10 File Twofft.c

```
void twofft(data1,data2,fft1,fft2,n)
float data1[],data2[],fft1[],fft2[];
int n;
{
        int nn3,nn2,jj,j;
        float rep,rem,aip,aim;
        void four1();


        nn3=1+(nn2=2+n+n);
        for (j=1,jj=2;j<=n;j++,jj+=2) {
                fft1[jj-1]=data1[j];
                fft1[jj]=data2[j];
        }
        four1(fft1,n,1);
        fft2[1]=fft1[2];
        fft1[2]=fft2[2]=0.0;
        for (j=3;j<=n+1;j+=2) {
                rep=0.5*(fft1[j]+fft1[nn2-j]);
                rem=0.5*(fft1[j]-fft1[nn2-j]);
                aip=0.5*(fft1[j+1]+fft1[nn3-j]);
                aim=0.5*(fft1[j+1]-fft1[nn3-j]);
                fft1[j]=rep;
                fft1[j+1]=aim;
                fft1[nn2-j]=rep;
                fft1[nn3-j] = -aim;
                fft2[j]=aip;
                fft2[j+1] = -rem;
                fft2[nn2-j]=aip;
                fft2[nn3-j]=rem;
        }
}
```

8.11 File Correl.c

```
void correl(data1,data2,n,ans)
float data1[],data2[],ans[];
int n;
{
        int no2,i;
        float dum,*fft,*vector();
        void twofft(),realft(),free_vector();

        fft=vector(1,2*n);
        twofft(data1,data2,fft,ans,n);
        no2=n/2;
        for (i=2;i<=n+2;i+=2) {
                ans[i-1]=(fft[i-1]*(dum=ans[i-1])+fft[i]*ans[i])/no2;
                ans[i]=(fft[i]*dum-fft[i-1]*ans[i])/no2;
        }
        ans[2]=ans[n+1];
        realft(ans,no2,-1);
        free_vector(fft,1,2*n);
}
```

8.12 File Ran.c

```
#define MBIG 1000000000
#define MSEED 161803398
#define MZ 0
#define FAC (1.0/MBIG)

float ran3(idum)
int *idum;
{
        static int inext,inextp;
        static long ma[56];
        static int iff=0;
        long mj,mk;
        int i,ii,k;
        if (*idum < 0 || iff == 0) {
                iff=1;
                mj=MSEED-(*idum < 0 ? -*idum : *idum);
                mj %= MBIG;
                ma[55]=mj;
                mk=1;
                for (i=1;i<=54;i++) {
                        ii=(21*i) % 55;
                        ma[ii]=mk;
                        mk=mj-mk;
                        if (mk < MZ) mk += MBIG;
                        mj=ma[ii];
                }
                for (k=1;k<=4;k++)
                        for (i=1;i<=55;i++) {
                                ma[i] -= ma[1+(i+30) % 55];
                                if (ma[i] < MZ) ma[i] += MBIG;
                        }
                inext=0;
                inextp=31;
                *idum=1;
        }
        if (++inext == 56) inext=1;
        if (++inextp == 56) inextp=1;
        mj=ma[inext]-ma[inextp];
        if (mj < MZ) mj += MBIG;
        ma[inext]=mj;
        return mj*FAC;
}

#undef MBIG
#undef MSEED
#undef MZ
#undef FAC
```

8.13 File Ranbit.c

```
#define IB1 1
#define IB2 2
#define IB5 16
#define IB18 131072

int irbit1(iseed)
unsigned long *iseed;
{
        unsigned long newbit;

        newbit = (*iseed & IB18) >> 17
                ^ (*iseed & IB5) >> 4
                ^ (*iseed & IB2) >> 1
                ^ (*iseed & IB1);
        *isced=(*iseed << 1) | newbit;
        return (int) newbit;
}
```

```
#undef IB1
#undef IB2
#undef IB5
#undef IB18
```

8.14 File Nutil.c

```
#include <malloc.h>
#include <stdio.h>
#include <nrutil.h>


void nrerror(error_text)
char error_text[];
{
        void exit();

        fprintf(stderr,"Run-time error...\n");
        fprintf(stderr,"%s\n",error_text);
        fprintf(stderr,"...now exiting to system...\n");
        exit(1);
}



double *dvector(nl,nh)
int nl,nh;
{
        double *v;

        v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
        if (!v) nrerror("allocation failure in dvector()");
        return v-nl;
}



double **dmatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
        int i;
        double **m;

        m=(double **) malloc((unsigned) (nrh-nrl+1)*sizeof(double*));
        if (!m) nrerror("allocation failure 1 in dmatrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=(double *) malloc((unsigned) (nch-ncl+1)*sizeof(double));
                if (!m[i]) nrerror("allocation failure 2 in dmatrix()");
                m[i] -= ncl;
        }
        return m;
}
```

```
void free_dvector(v,nl,nh)
double *v;
int nl,nh;
{
        free((char*) (v+nl));
}


void free_dmatrix(m,nrl,nrh,ncl,nch)
double **m;
int nrl,nrh,ncl,nch;
{
        int i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}
```

APPENDIX I

OUTPUT FROM CODE SET DESIGN

```
number:302     e: 23.0000     de: -2.0000     temp: 25.0000
number:265     e: 19.0000     de: -4.0000     temp: 22.5000
number:248     e: 17.0000     de: -2.0000     temp: 20.2500
number:245     e: 17.0000     de:  0.0000     temp: 18.2250
number:210     e: 17.0000     de: -0.0000     temp: 16.4025
number:193     e: 19.0000     de:  2.0000     temp: 14.7623
number:177     e: 17.0000     de: -2.0000     temp: 13.2860
number:143     e: 18.0000     de:  1.0000     temp: 11.9574
number:157     e: 19.0000     de:  2.0000     temp: 11.9574
number:142     e: 17.0000     de: -2.0000     temp: 10.7617
number:114     e: 20.0000     de:  3.0000     temp:  9.6855
number:126     e: 23.0000     de:  6.0000     temp:  9.6855
number:109     e: 21.0000     de:  4.0000     temp:  9.6855
number:149     e: 21.0000     de:  4.0000     temp:  9.6855
number:117     e: 19.0000     de:  2.0000     temp:  9.6855
number:142     e: 19.0000     de:  2.0000     temp:  9.6855
number:115     e: 21.0000     de:  4.0000     temp:  9.6855
number:116     e: 19.0000     de:  2.0000     temp:  9.6855
number:122     e: 19.0000     de:  2.0000     temp:  9.6855
number:103     e: 19.0000     de:  0.0000     temp:  8.7170
number:94      e: 21.0000     de:  2.0000     temp:  7.8453
number:96      e: 19.0000     de:  0.0000     temp:  7.8453
number:82      e: 17.0000     de: -2.0000     temp:  7.0607
number:68      e: 17.0000     de:  0.0000     temp:  6.3547
number:64      e: 19.0000     de:  2.0000     temp:  5.7192
number:76      e: 17.0000     de: -0.0000     temp:  5.7192
number:82      e: 19.0000     de:  2.0000     temp:  5.1473
number:70      e: 19.0000     de:  2.0000     temp:  5.1473
number:60      e: 19.0000     de:  2.0000     temp:  5.1473
number:60      e: 19.0000     de:  2.0000     temp:  5.1473
number:59      e: 17.0000     de: -0.0000     temp:  5.1473
number:65      e: 19.0000     de:  2.0000     temp:  4.6326
number:50      e: 19.0000     de:  2.0000     temp:  4.6326
number:49      e: 17.0000     de:  0.0000     temp:  4.6326
number:47      e: 19.0000     de:  2.0000     temp:  4.1693
number:54      e: 19.0000     de:  2.0000     temp:  4.1693
number:55      e: 19.0000     de:  2.0000     temp:  4.1693
number:46      e: 17.0000     de:  0.0000     temp:  4.1693
number:45      e: 19.0000     de:  2.0000     temp:  3.7524
number:55      e: 21.0000     de:  2.0000     temp:  3.3771
number:47      e: 18.0000     de: -1.0000     temp:  3.3771
number:45      e: 18.0000     de:  0.0000     temp:  3.0394
number:39      e: 19.0000     de:  1.0000     temp:  2.7355
number:30      e: 18.0000     de: -1.0000     temp:  2.4619
number:27      e: 19.0000     de:  1.0000     temp:  2.2157
number:23      e: 19.0000     de: -0.0000     temp:  1.9942
number:43      e: 17.0000     de: -2.0000     temp:  1.7947
number:14      e: 19.0000     de:  2.0000     temp:  1.6153
number:23      e: 17.0000     de: -0.0000     temp:  1.6153
number:22      e: 17.0000     de: -0.0000     temp:  1.4537
number:15      e: 17.0000     de: -0.0000     temp:  1.3084
number:15      e: 17.0000     de:  0.0000     temp:  1.1775
peak correlation is: 17.000000
sequence 1: 1001011111100000000011110
sequence 2: 1101110010110001110101101
```

sequence 3: 0000110011000010010001111
sequence 4: 1000001100101110000111111
sequence 5: 0000101101010000100011111
sequence 6: 0010111011100000110011010
sequence 7: 0000101001101011000011111
sequence 8: 1111111111011101101110000
sequence 9: 1111010111000111101101111
sequence 10: 1111111110011001110111001
sequence 11: 0001110111110001111101001
sequence 12: 0110010011101001011000011
sequence 13: 1101100011101001010111101
sequence 14: 0101001000101000111101111
sequence 15: 1011011010101000000011011
sequence 16: 1110011101011100000100011
sequence 17: 1100100010010101111110100
sequence 18: 1011001000000010010000111
sequence 19: 1010110001100110011101100
sequence 20: 1101010000101100011100011
sequence 21: 0011000101011001101101010
sequence 22: 1100000101101011001010111
sequence 23: 1001110010100010010100011
sequence 24: 1001110111100110010100110
sequence 25: 0101100001101111111011100

APPENDIX J

REAL TIME ADAPTIVE TEXT COMPRESSION

ARE suggested a study for real time text compressor, suitable for use on RATT links. The purpose of this section is to describe a novel, adaptive, real-time text compressor. The system uses a widely available single board microprocessor system to perform the transmitter and receiver functions and achieves substantial compression at a realistic cost using ready made components. The system is completely self contained, small and can adapt to any text system automatically. The compression efficiency depends on the redundancy in the text being sent - it is clear that a completely random stream of words could not be compressed at all. Bearing this in mind, a compression efficiency of 30% may be obtained in average text, while heavy use of abbreviations, as found in military signals, will reduce this efficiency to around 25%. Compression efficiency is defined as

$$\eta = \frac{N_{raw} - N_c}{N_{raw}} \times 100$$

Where N_{raw} is the number of bits required to transmit the raw information, N_c is the number of bits required to transmit the compressed information.

10.1 Fixed Dictionary Compression

As far as the author is aware, very few attempts have been made at text compression for real time applications. Library applications have in general centred around the use of fixed application dictionaries. For every application a predefined dictionary is maintained of commonly occurring words which, when encountered in the text, are assigned specific reserved codes that are not allowed to occur otherwise. It is clear that such a system could easily be implemented for use in a military environment.

Brushing aside for the moment potential problems that might be encountered (capital letters, plurals, words followed by punctuation parentheses etc.), such a system could be implemented as shown below. At the transmitter the processor is required to delimit the incoming text stream into isolated words, which may be checked against the fixed dictionary held in permanent storage. If a match is found, the appropriate code is sent over the link, otherwise the uncompressed word is transmitted.

Advantages:

Simple implementation

Rapid Search of fixed dictionary - no updating

Fixed codes - no need to transfer code words

Disadvantages:

Fixed, application dependent dictionary

Large dictionary needed to get good compression

An alternative to this scheme is suggested below. Here the processor examines the incoming text stream and notes frequently occurring words. These are assigned the codes which are used to achieve compression. These two requirements pose problems:

How to determine frequently occurring words.

How to convey to the receiver the codes mappings.

In order to determine the most frequently occurring words, the processor must keep a record of all different words that it identifies in the incoming text stream and their usage count. Words attracting a heavy usage are viewed by the system as potential candidates for code allocation. Clearly, the total number of different words that can be received is very large, but it is likely that in any particular situation or piece of text the number encountered will be much smaller. In any case the present system can store over 100,000 letters, or three hours worth of text at 100 baud. When a word is received by the processor it first searches for it in the dictionary, which is organised as a tree structure to allow for rapid searching. If a match is found the processor updates the usage count and examines a flag that indicates whether the received word is coded or not. If it is coded then the code is transmitted, otherwise the original word is used. This part of the program operates as an interrupt driven routine.

If a word is not found in the dictionary, it is placed in an update queue. A background process on the computer continuously updates the dictionary and selects words for coding. To avoid possible deadlock, or out of sync codes being used, a system of semaphores is used to communicate between the interrupt process and the background process. After updating the dictionaries tree structure to incorporate new words, the processor performs a sequential search through the entire dictionary. It notes:

The word with the lowest usage count $N_{c,low}$

The uncoded word with the highest usage count $N_{u,high}$

If $N_{u,high}$ is greater than $N_{c,low}$ the code for $N_{c,low}$ is removed and used instead for $N_{u,high}$, unless the total sum of all the codes used so far is less than the maximum allowed, in which case $N_{u,high}$ is allocated a code anyway. This latter situation will usually occur at start up when no codes have been allocated, however, the subject of pre-defined dictionaries will be covered later.

Since the dictionary search routine (for code identification) is interrupt driven, it is not possible for the update routine to modify the codes during a search operation and thus no contention is possible.

After a code change has occurred, it is essential that the processor should inform the receiver before any more text is transmitted. This is easily accomplished by having the background task place code updates in a queue. Before the interrupt routine transmits a word it checks the status of the code queue to see if it is full. If a new code word is discovered it is sent, prior to the outstanding text word, since it is conceivable that the update occurred during reception of the current text word. This is particularly true if data is being sent relatively slowly, allowing updates to occur between the reception of words. Under normal circumstances the code queue will be short because there will be sufficient time to update the dictionary between received words, and therefore only the last word received is a candidate for updating. If text is received rapidly, dictionary updating may fall behind reception of data, although any code updates placed in the queue will be cleared rapidly. If the queue grows long, this implies that there has been a break in a rapid period of text transmission, allowing dictionary updating to catch up. As described, the system operates with an interrupt driven input/output system and a background updating operation. If the source of the text should attempt to send text to the processor very rapidly there will be little or no remaining time to perform dictionary updating. It will never be possible to overrun the receiver since DTR and Xon Xoff can be used. As soon as a character is removed from the serial interface, the text source may place another character there. This will cause the interrupt routine to be immediately re-entered with the new character, as soon as the old one has been processed. In general this should not be a problem, since we assume the text is transferred at a sufficiently slow baud rate. For a given system the maximum baud rate will be highly dependent upon the dictionary size, compression efficiency, etc., which will be text content dependent.

A bottle neck exists in the previously described information when the uncoded word is transmitted. As shown, the processor transmits this word while it is in the interrupt routine.

This means that it will poll the transmit data register of the serial interface whilst waiting for characters to be transmitted. At 100 baud, it might take one second to transmit a complete word. To overcome this problem the transmit data routine is also interrupt driven, using the transmit data register empty flag from the serial interface to initiate an interrupt. The processor used for the prototype system supports autovectoring of interrupts from many sources - it is therefore not necessary to determine which serial interface caused the interrupt since this is implicit in the address vectored to. The receive routine is afforded a higher priority interrupt so that the transmit data routine may be interrupted at appropriate moments. Thus, the data to be transmitted is placed in a queue which is gradually emptied by the transmit data routine. At any time the processor may be:

Receiving data and searching the dictionary

Updating the dictionary

Transmitting data.

Routine (3) is very short and consumes negligible time. Routine (2) consumes the majority of the processing time at low baud rates, since the dictionary may be searched very rapidly. The sequential search used in (1) must access the dictionary for every word present, which is a very time consuming task.

The system described so far works efficiently once equilibrium has been reached. Unfortunately, at start-up, the dictionary is empty and so no codes exist, giving rise to an initial compression efficiency of worse than 0%. This may not be a problem in some applications, but the present system supports an optional start-up dictionary which may be application dependent. This start-up dictionary is transferred from back-up storage at start-up. Consideration is also given to providing a memory with battery back-up to hold dictionaries that may be transferred from the dictionary currently being used, ready for when the resident dictionary is required again. This back-up memory would not need to be very large since it would only hold the coded words, unlike the processors memory which must also hold the uncoded words in order to maintain a usage count. The use of a back-up would also be very useful if the subject of the text is suddenly changed - since it might be possible to call up the new dictionary immediately - the old dictionary being placed in the back-up memory. This facility would require some intelligence on the part of the text source, and would therefore be included only as an optional enhancement for those devices able to take advantages of it.

10.2 Updating the Control Codes

In conventional text compression systems with fixed application dictionaries, the size of the dictionary will need to be large to maximise the chance of finding words that match. A small dictionary, even for a relatively well defined application, will lead eventually to a low compression efficiency. In general, the format for such a system will be as shown below, and it will be necessary to use multi-byte codes. This is because the number of words in the dictionary is likely to be in excess of 2^N where N is the number of bits used in the transmission process.

In the adaptive text compressor the number of coded dictionary words can be very much smaller than in the fixed dictionary scheme, since the codes represent the most frequently used words at that moment in time. The distribution of the most frequently occurring words will be text context dependent, and it is therefore impossible to predict the optimum dictionary size exactly. Also, the way in which the data is transmitted over the communications link will affect matters. In conventional serial communications systems ASCII (ASCII: American Standard Code for Information Interchange) characters are mapped into an 8 bit byte for convenience, with the most significant bit as a don't care a ninth parity bit is often added and removed by the serial interface and is transparent to the processor unless an error occurs. However, some serial systems utilise only seven bits (even though the data will probably end up in byte format, the serial interface will only transmit seven bits per character), or 5 bits (BAUDOT code with letter and numeric shift characters).

It is desirable that any text compressor should be able to cope with all these formats and it is certain that a software selectable scheme could be implemented. Military communications still utilise the older BAUDOT scheme for RATT. Modern computer systems do not normally support BAUDOT. This is not a serious problem since it is possible to simulate a BAUDOT scheme with an eighth or seven bit scheme. The present system adopts this method.

Given that we expect the number of words in the coded dictionary to be relatively small and heavily used, it is not worthwhile considering the 7 bits of the first code and the 8 bits of the second to constitute a 15 bit code (32768 words), since this clearly exceeds our likely usage. It is better to concentrate on a single byte code, with a possible extension if necessary, since we might expect heavy use to be made of relatively few code words. The following diagrams indicate the protocol adopted for both 8-bit and 7-bit ASCII. In the eight bit code, bit 7 is used to separate code words and ASCII characters. If bit 7 = 0 then the character is considered as straight ASCII - this means that there is no restriction whatsoever on the ASCII character used in the text. If bit 7 = 1 then the following 7 bits constitute either a code address (03-7F) or one of three control characters. Thus, the 125 most frequently encountered words may be compressed into a single byte. If the control character = 00, then this indicates that the following byte (01-FF) constitutes a secondary address, allowing an extension of a further 255 addresses. To define a new code at the receiver, the transmitter first sends a 01 control code (ie 81 is transmitted, since bit 7 = 1) followed by the appropriate address, extended if need be, followed by the word to be expanded in normal ASCII text, followed by an 02 (ie 82), which is code word end.

During transmission of 8-bit characters, bit 7 will be 1 to indicate code word start (rather than an ASCII character) but following codes, addresses will have bit 7 = don't care, since they can be unambiguously separated from the ASCII text by their position in the control stream. This has been done in the present system for reasons of software compatibility with the 7-bit code. However, code word end must have bit 7 = 1 to separate it from the preceeding ASCII text.

The 7-bit protocol is similar to the 8-bit, except that the user must now identify a reserved control code to separate ASCII characters and control. Thus, the shortest possible sequence is two bytes long, the reserved code followed by a code address (03-127, as in the previous case). A similar scheme is used for address extensions etc, except that their addresses are limited to 01-127 for extensions.

10.3 The Effect of Errors

It is clearly possible (and with RATT links, even likely) that errors will occur in the received text. This section considers the possible implications for compressed text transmission of this state of affairs for two conditions:

Errors occur, and the presence of those errors is detected (but not corrected) by some undefined mechanism (ie parity check) at the receiver.

In normal text transmission a ? is printed at the receiver.

Errors occur, and their presence is not detected at the receiver (ie either no parity check etc exists, or possibly it was ineffective).

Depends on the faulty character - however, it is likely that a faulty character will be printed.

Since only complete words are candidates for coding, and we define a complete work as one delimited by either spaces on either side or some combination of spaces and carriage return symbols we will have the situation shown below.

Here the previous character must have either been 20, 0D or 0A. In this case we are unable to distinguish whether this was a code or the first letter of a new word (uncoded). However, if the previous character was not 20, 0D or 0A and we are not in a code sequence, then it must be a letter. In the text, if the previous character was 20, 0D or 0A, a ? is printed to draw attention to the possibility of code word corruption (? underlined), otherwise a ? is printed (non underlined). The worst possibility is that the corrupted byte marked the start of a new code word definition, or occurred during sensitive parts of the definition (address extension or particularly address definition). If the ASCII text part of a code definition is corrupted, the effects will not be too serious - possibility one letter out of place. If the code terminating character is corrupted, but still has bit 7 = 1, this has no effect. If bit 7 is corrupted to 0, then it is still likely the effect will not be too serious, since 20, 0D or 0A will follow and the processor will terminate the code word definition and delete the character from the code word that occurred last.

The serious effect is when a new code word definition has a corrupted address and the processor then does not know what code to use. In this situation the processor at the receiver will search its (small) code word dictionary for the coded word with the lowest usage count and replace this, on the basis that the transmitter would not have selected any other candidate for replacing, unless the complete of codes is as yet unused, when the next code in sequence will be allocated. This mechanism is used as a check at every update, since for systems with no parity check, this will represent the only checking mechanism available. If, when the check is completed, a discrepancy is found (and no parity error or similar was detected) then a serious error has occurred and the user is informed. Since it is likely, when a new code is defined, that is soon followed by a use of that code, the processor waits for the next occurrence of a code. If the code is either of the two codes above (usage count generated code address, or incoming code) then this is the one selected for updating. This mechanism provides a powerful resistance to false updating of the codes in the receiver, potentially the most dangerous area.

If we receive a code address incorrectly (say a primary address), then the processor assumes that only one bit of the received data was in error, and indicates the seven possible alternative code words, or the appropriate ASCII character for selection by the user.

Note that a large number of these types of errors will upset the receiver usage count, possibly causing serious problems. However, since we might expect the errors to be random in nature, this is unlikely to happen.

If we have no parity check facility in our date transmission system, or if the parity system in use is overcome (two errors in one byte) then, except for the update operation mentioned above, the processor will be unable to determine that an error has occurred. One incorrect ASCII character, or one incorrect code word will be printed - however, this will be an isolated event - corruption of a code word definition is checked for, since this is potentially a dangerous area, with the possibility that many subsequent words would be misprinted.

10.4 System Design

The prototype uses a powerful microprocessor to implement the adaptive coding algorithms, basing the demonstrator on a readily available development system, the Force

GmbH 68K Profi Kit with built in software development tools. The code for the text compressor was written in assembly language (see APPENDIX K for a complete listing). The text compressor consists of three main modules.

The first module performs *initialisation of the code pool if necessary, and sets up* communication ports as appropriate. The second module is a tree manager which is used in all references to the code pool and code mapping linked list. The third module performs input and output through buffers and is interrupt driven.

10.5 Conclusion

The previous section has described the design and development of a real-time adaptive text compressor suitable for RATT use. The variable dictionary approach ensures that the compressor adapts to the current text content, and therefore requires only a short code word, resulting in high efficiency. Careful thought has been given to the protection from corruption by errors of the system. A standard single-board microprocessor system utilising the Motorola M68000 has been used both at the receiver and the transmitter, resulting in a very economical and compact design.

APPENDIX K

TEXT COMPRESSOR LISTING

```
/************************************************************************/
/*              TEXT  COMPRESSOR                  */
/************************************************************************/
BASE:   EQU   @800;             /*START OF THE PROGRAM*/
TACIA:  EQU   @C0080;             /*ACIA ADDRESSES*/
HACIA:  EQU   @$C0041;
TDRE:   EQU   #1;               /*READY FLAGS IN THE ACIA'S*/
RDRF:   EQU   #0;
SPACE:  EQU   ' ';              /*CHARACTER CONSTANTS*/
CRET:   EQU   #13;
LFEED:  EQU   #10;
/************************************************************************/
/* REGISTER EQUATES                               */
/************************************************************************/
FLAGS:  EQU   D7;               /*D7 IS THE FLAG REGISTER*/
EOL:    EQU   #0;
TEXT:   EQU   #1;
HSHAK:  EQU   #2;
BPTR:   EQU   A0;
DP:     EQU   A4;
SP:     EQU   A3;
SLP:    EQU   A6;
LEP:    EQU   A5;
CODE:   EQU   D2;
CHCTR1: EQU   D1;
CHCTR2: EQU   D3;
ENEXT:  EQU   A0;
PPREV:  EQU   A1;
PREV:   EQU   A2;
CRNT:   EQU   A3;
RECCH:  EQU   D5;
TRCH:   EQU   D6;
/************************************************************************/
/* THE TEXT BUFFER AND DICTIONARY START AT THE BOTTOM OF MEMORY     */
/************************************************************************/
     ORG   BASE;
     DS    10;    /*STACK*/
     ORG   BASE;
     BRA.L  BOOT;
SYSSTK:
BUFF:  DS    $20;
DICT:  DS    1;
/************************************************************************/
/* TABLE OF MESSAGES TO BE SENT TO THE TERMINAL              */
/************************************************************************/
     ORG   @$1000;
     BRA.L  BOOT;
     BRA.L  BOOTD;
     DC    #SYSSTK;  /*INITIAL STACK POINTER*/
     DC    #BOOT;    /*INITIAL PROGRAM COUNTER*/
MSSG:  DC.B CRET; DC.B LFEED; DC.B LFEED; DC.B LFEED; DC.B LFEED; DC.B LFEED;
     DC '        DURHAM UNIVERSITY TEXT COMPRESSOR'; DC.B CRET;
     DC.B LFEED; DC.B LFEED; DC.B LFEED;
     DC '                VERSION  1.0'; DC.B CRET; DC.B LFEED;
     DC.B LFEED;
     DC 'TCOMP:  IS THIS DEVICE A TRANSMITTER OR A RECEIVER ?'; DC.B CRET;
     DC.B LFEED;
     DC 'TCOMP:  TYPE T FOR TRANSMIT, R FOR RECEIVE';
     DC.B CRET; DC.B LFEED;
     DC.B #0;
/************************************************************************/
TMSG:   DC 'TCOMP:  TRANSMITTER-'; DC.B CRET; DC.B LFEED; DC.B #0;
/************************************************************************/
RMSG:   DC 'TCOMP:  RECEIVER-'; DC.B CRET; DC.B LFEED; DC.B #0; DC.B #0;
/************************************************************************/
/* THE PROGRAM STARTS HERE AND IS FULLY RELOCATABLE. IT IS NORMALLY  */
/* RELOCATED TO THE SYSTEM EPROM, SO THE REAL START ADDRESS IS $28146*/
/************************************************************************/
/*
BOOT:   MOVEM.W D0,@$406;
     LEA.L  $8,A0;
STRP:  LEA.L  #@BOOT,A1;
     MOVE.L  A1,(A0)+;
     CMP.L  #$400,A0;
     BMI.S  STRP;  */
BOOT:   BSET   HSHAK,FLAGS;
     BRA.S  TIGH;
BOOTD:  BCLR   HSHAK,FLAGS;
TIGH:   MOVE   #$FF,D0;            /*CLEAR MEMORY*/
     MOVE   #DICT,A0;
CLRM:  MOVE   #0,(A0)+;
     DBF    D0,CLRM;
INIT:  MOVE   #0,FLAGS;            /*CLEAR ALL FLAGS*/
     MOVE   #DICT,DP;             /*INITIALISE DICT POINTER*/
     MOVE   #DICT,SP;             /*INITIALISE SEARCH POINTER*/
     MOVE   #DICT,SLP;             /*INIT START OF LIST PTR*/
     MOVE   #DICT,LEP;             /*INIT LAST ENTRY POINTER */
     CLR    RECCH;              /*CLEAR COUNT RECEIVED CHARS*/
     CLR    TRCH;              /*CLEAR COUNT OF TX'ED CHARS*/
     CLR    CODE;              /*THE FIRST CODE*/
     CLR    CHCTR1;             /*CLEAR THE COUNT REGISTERS*/
     CLR    CHCTR2;
      MOVE.B  #$43,HACIA;          /*RESET THE HOST ACIA*/
      MOVE.B  #$43,TACIA;          /*RESET THE TERMINAL ACIA*/
      MOVE.B  #$15,TACIA;          /*ENABLE ACIA FOR MESSAGES*/
      LEA    MSSG,A0;           /*PRINT THE STARTUP MESSAGE*/
      BSR.L  TYPE;
 INC:   BTST   RDRF,TACIA;          /*GET THE RESPONSE*/
      BEQ.S  INC;
      MOVE.B  TACIA+2,D0;          /*TRANSMIT OR RECEIVE MODE */
      BCLR   #7,D0;             /*MAKE IT A LEGAL CHARACTER*/
      CMP.B  'R',D0;            /*RECEIVE ?*/
      BEQ.L  RECEIVE;            /*YES GO TO RECEIVE ROUTINE*/
      CMP.B  'T',D0;            /*TRANSMIT ?*/
      BNE.S  INIT;             /*NO SO ASK AGAIN*/
 /*********************************************************************/
 /*          TRANSMIT ROUTINE                  */
 /*********************************************************************/
      LEA    TMSG,A0;           /*PRINT THE TRANSMIT MESSAGE*/
      BSR.L  TYPE;
      MOVE   #BUFF,BPTR;          /*INITIALISE CHARACTER BUFF*/
 /*****************************************/
 /*    IS EOL SET ?          */
 /*****************************************/
      MOVE.B  #$55,TACIA;
      MOVE.B  #$19,HACIA;
```

```
TRANSM: BTST   #EOL,FLAGS;          /*IS END OF LINE SET*/
     BEQ.S  NOCR;
/******************************************/
/* TRANSMIT A CRLF                  */
/******************************************/
HSRWAIT: MOVE   #$FFF,D0;
WAITJ:  DBF   D0,WAITJ;
     CLR   D0;
     BSR.L  HSEND;              /*WAIT FOR THE LINE TO CLEAR*/
     MOVE.B CRET,HACIA+2;          /*TRANSMIT A CARRIAGE RETURN*/
     BCLR   #EOL,FLAGS;          /*CLEAR EOL FLAG*/
/******************************************/
/* SET A0 TO START OF BUFFER           */
/******************************************/
NOCR:  MOVE   #BUFF,BPTR;          /*SET A0 TO START OF BUFF*/
/******************************************/
/* READ A CHARACTER FROM TERMINAL INTO D0 */
/******************************************/
TSRWAIT: BTST   HSHAK,FLAGS;
     BEQ.S  TWAIT;
WT:   BTST  TDRE,TACIA;
     BEQ.S  WT;
     MOVE.B '*',TACIA+2;
TWAIT:  BTST   RDRF,TACIA;
     BNE.S  GETCH;
     MOVE.B #$15,TACIA;          /*AWAIT INCOMING CHARACTER*/
WAITR:  BTST   RDRF,TACIA;
     BEQ.S  WAITR;
     MOVE.B #$55,TACIA;
GETCH:  MOVE.B TACIA+2,D0;          /*GET THE CHARACTER INTO D0*/
     BTST   HSHAK,FLAGS;
     BNE.S  SKP;
     MOVE.B TACIA+2,TACIA+2;       /*AND ECHO THE CHARACTER*/
SKP:   BCLR   #7,D0;           /*MAKE IT A LEGAL CHARACTER*/
/******************************************/
/* IS THE CHARACTER < '!'           */
/******************************************/
     CMP.B  '!',D0;           /*IS IT A SPACE OR LESS*/
     BMI.S  CNTRL;             /*YES SO CHECK IT*/
/******************************************/
/* SET THE TEXT FLAG                */
/******************************************/
     BSET   TEXT,FLAGS;          /*SET TEXT MODE*/
/******************************************/
/* PUT THE CHARACTER INTO THE BUFFER    */
/******************************************/
     MOVE.B D0,(BPTR)+;          /*PUT CHARACTER INTO BUFF*/
     BRA.S  TSRWAIT;           /*AND GET THE NEXT CHARACTER*/
/******************************************/
/* IS THE CHARACTER CR OR LF ?        */
/******************************************/
CNTRL:  CMP.B  CRET,D0;           /*WAS CHARACTER A CR OR LF*/
     BNE.S  QTXT;            /*YES SO SEND A CR*/
/******************************************/
/* SET THE EOL FLAG                 */
/******************************************/
     BSET   EOL,FLAGS;          /*SET THE EOL FLAG*/
/******************************************/
/* IS THE TEXT FLAG SET              */
/******************************************/
QTXT:  BTST   TEXT,FLAGS;          /*ARE WE IN TEXT MODE*/
```

```
     BEQ.L  TRANSM;              /*NO SO IGNORE THE CHARACTER*/
/******************************************/
/* GET THE LENGTH OF THE WORD INTO D1    */
/******************************************/
     MOVE   BPTR,CHCTR1;          /*GET THE LENGTH INTO D1*/
     SUB    #BUFF,CHCTR1;
/******************************************/
/* CLEAR THE TEXT FLAG                   */
/******************************************/
     BCLR   TEXT,FLAGS;           /*NO LONGER IN TEXT MODE*/
/******************************************/
/* GET THE ADDRESS OF THE START OF THE   */
/* START OF THE DICT INTO A3             */
/******************************************/
     MOVE   SLP,SP;              /*START SEARCHING*/
/******************************************/
/* ARE THE LENGTHS EQUAL                 */
/******************************************/
COMP:  CMP.B  $E(SP),CHCTR1;        /*ARE THE LENGTHS EQUAL*/
     BNE.S  NEXT;                /*NO SO CHECK THE NEXT ENTRY*/
/******************************************/
/* SET A1 TO BUFF FOR START OF COMPARISON */
/******************************************/
     MOVE   #BUFF,A1;            /*SET UP FOR COMPARISON*/
/******************************************/
/* SET D3 TO THE LENGTH OF THE WORD      */
/******************************************/
     MOVE.B CHCTR1,CHCTR2;        /*GET THE LENGTH INTO D3*/
     SUBQ.B #1,CHCTR2;            /*FOR DBF INSTRUCTION*/
     MOVE   SP,A2;
     ADD    #$F,A2;              /*OFFSET TO FIRST CHARACTER*/
/******************************************/
/* COMPARE CHARACTERS                    */
/******************************************/
CCOMP:  CMPM.B (A1)+,(A2)+;        /*COMPARE CHARACTERS*/
     BNE.S  NEXT;
     DBF    CHCTR2,CCOMP;
     CLR    CHCTR2;              /*RESTORE D3*/
/******************************************/
/* INCREMENT THE USAGE COUNT             */
/******************************************/
     ADDI   #1,8(SP);            /*FOUND IT INC USAGE COUNT*/
/******************************************/
/* SEND THE CODE                         */
/******************************************/
HTWAIT: BSR.L  HSEND;              /*SEND FIRST BYTE OF CODE*/
     MOVE.B $C(SP),D0;
     ADD.B  #$80,D0;
     MOVE.B D0,HACIA+2;
SINGLE: BSR.L  HSEND;              /*SEND THE SECOND BYTE*/
     MOVE.B $D(SP),D0;
     ADD.B  #$80,D0;
     MOVE.B D0,HACIA+2;
     BSR.L  SWP;
     BRA.L  TRANSM;              /*GET MORE CHARACTERS*/
/******************************************/
/* IS THE NEXT ADDRESS ZERO              */
/******************************************/
NEXT:  CMPI   #0,(SP);             /*IS THE NEXT ADDRESS ZERO*/
     BEQ.S  FILL;                /*YES SO MAKE A NEW ENTRY*/
/******************************************/
```

```
/* SET A3 TO THE NEXT ADDRESS          */
/*******************************************/
     MOVE   (SP),SP;              /*NO SO GET THE NEXT ENTRY*/
     BRA.S  COMP;                 /*TRY THE NEXT ENTRY*/
/*******************************************/
/* MAKE A NEW ENTRY IN THE LIST        */
/*******************************************/
FILL:  MOVE   DP,(SP)+;            /*LINK*/
     MOVE   #0,(DP)+;             /*NEXT ADDRESS*/
     MOVE   SP,(DP)+;
     MOVE   #1,(DP)+;             /*USAGE COUNT*/
     MOVE.W CODE,(DP)+;            /*CODE*/
     MOVE.B CHCTR1,(DP)+;           /*CHARACTER COUNT*/
     SUBQ.B #1,CHCTR1;             /*FOR DBF*/
     MOVE   #BUFF,A1;             /*START THE SEARCH AT BUFF*/
/*******************************************/
/* SEND A CHARACTER TO THE ACIA         */
/*******************************************/
HERE:  BSR.S  HSEND;
     MOVE.B (A1),HACIA+2;          /*SEND THE CHARACTER*/
/*******************************************/
/* PUT THE CHARACTER INTO THE DICTIONARY */
/*******************************************/
     MOVE.B (A1)+,(DP)+;           /*AND PUT IT IN THE DICT*/
/*******************************************/
/* MORE CHARACTERS ?                    */
/*******************************************/
     DBF    CHCTR1,HERE;           /*MORE TO DO?*/
/*******************************************/
/* ROUND A4 UP                  */
/*******************************************/
     CLR    CHCTR1;               /*ZERO D1*/
     MOVE   DP,D0;
     ADDQ   #1,D0;                /*ROUND A4 UP*/
     BCLR   #0,D0;
     MOVE   D0,DP;
/*******************************************/
/*  SEND THE CODE JUST ALLOCATED        */
/*******************************************/
     BSET   #7,CODE;
     BSET   #15,CODE;
     ROR    #8,CODE;
WAIT9:  BSR.S  HSEND;
     MOVE.B CODE,HACIA+2;
     ROL    #8,CODE;
WAIT8:  BSR.S  HSEND;
     MOVE.B CODE,HACIA+2;
     BCLR   #7,CODE;
     BCLR   #15,CODE;
     ADDQ.B #1,CODE;
     BCLR   #7,CODE;
     BEQ.L  TRANSM;
     ADD.W  #$100,CODE;
     BRA.L  TRANSM;               /*AND GET THE NEXT CHARACTER*/
/******************************************************************/
/* THIS SUBROUTINE SENDS A CHARACTER TO THE HOST PORT          */
/******************************************************************/
HSEND:  BTST   TDRE,HACIA;
     BEQ.S  HSEND;
     RTS;0
/******************************************************************/
```

```
/* THIS SUBROUTINE PERFORMS THE HANDSHAKING WITH THE HOST PORT     */
/*********************************************************************/
HSHK:  BTST  RDRF,HACIA;
     BNE.S  NOREAD;
     MOVE.B  #$19,HACIA;
WAIT:  BTST  RDRF,HACIA;
     BEQ.S  WAIT;
     MOVE.B  #$59,HACIA;
NOREAD: ADDQ  #1,RECCH;
     RTS;
/*********************************************************************/
/* THIS ROUTINE PERFORMS THE HANDSHAKING WITH THE TERMINAL PORT    */
/*********************************************************************/
TSHK:  BTST  TDRE,TACIA;
     BEQ.S  TSHK;
     ADDQ  #1,TRCH;
     RTS;
/*********************************************************************/
/*        RECEIVER ROUTINE                       */
/*********************************************************************/
RECEIVE: LEA  RMSG,A0;
     BSR.L  TYPE;
     MOVE  #BUFF,BPTR;
/*****************************************/
/* READ A CHARACTER FROM THE ACIA      */
/*****************************************/
CHRWAIT: BSR.S  HSHK;
     MOVE.B  HACIA+2,D0;
/*****************************************/
/* IS IT A CARRIAGE RETURN          */
/*****************************************/
START:  CMP.B  CRET,D0;             /*IS IT A CARRIAGE RETURN*/
     BNE.S  NOCRET;
/*****************************************/
/* PRINT A CARRIAGE RETURN LINEFEED    */
/*****************************************/
     BSR.L  DISPST;
WAIT1:  BSR.S  TSHK;              /*PRINT A CRLF*/
     MOVE.B  #CRET,TACIA+2;
     BSR.S  TSHK;
     MOVE.B  #LFEED,TACIA+2;
     BRA.S  QTEXT;            /*CHECK FOR END OF WORD*/
/*****************************************/
/* IS THE CHARACTER A CODE          */
/*****************************************/
NOCRET:  BTST  #7,D0;            /*IS IT A CODE*/
     BNE.S  QTEXT;            /*CHECK FOR AND OF WORD*/
/*****************************************/
/* SET THE TEXT FLAG, BUFFER THE       */
/* CHARACTER AND READ ANOTHER         */
/*****************************************/
     BSET  TEXT,FLAGS;         /*READING CHARS SO FLAG IT*/
     MOVE.B  D0,(BPTR)+;          /*AND BUFFER THE CHARACTER*/
WAIT3:  BSR.S  HSHK;
     MOVE.B  HACIA+2,D0;
     BRA.S  START;            /*AND CHECK IT*/
/*****************************************/
/* IS THE TEXT FLAG SET            */
/*****************************************/
QTEXT:  BTST  TEXT,FLAGS;          /*ARE WE IN TEXT MODE*/
     BEQ.L  QCODE;            /*NO SO DECODE THE CODE WORD*/
     BCLR   TEXT,FLAGS;
/******************************************/
/* MAKE A NEW ENTRY, READ IN THE REST OF */
/* THE CODE AND COMPARE WITH THE ONE JUST */
/* ALLOCATED                        */
/******************************************/
     MOVE   DP,(LEP)+;          /*WAS TEXT MAKE A NEW ENTRY*/
     MOVE   #0,(DP)+;           /*NEXT ADDRESS*/
     MOVE   LEP,(DP)+;
     MOVE   DP,LEP;
     SUB    #8,LEP;
     MOVE   #1,(DP)+;           /*USAGE COUNT*/
     MOVE.W CODE,(DP)+;          /*CODE*/
     MOVE   BPTR,CHCTR1;          /*GET THE LENGTH INTO D1*/
     SUB    #BUFF,CHCTR1;
     MOVE.B CHCTR1,(DP)+;         /*AND PUT IT IN THE DICT*/
     SUBQ.B #1,CHCTR1;           /*FOR THE DBF*/
     MOVE   #BUFF,BPTR;          /*START MOVING CHARACTERS*/
STCHAR: BSR.L  TSHK;
     MOVE.B (BPTR),TACIA+2;
     MOVE.B (BPTR)+,(DP)+;        /*COPY A CHARACTER*/
     DBF    CHCTR1,STCHAR;
     CLR    CHCTR1;             /*CLEAR CHARACTER COUNTER*/
     MOVE   DP,D3;              /*ROUND A4 UP*/
     ADDQ   #1,D3;
     BCLR   #0,D3;
     MOVE   D3,DP;
/******************************************/
/* RESET THE INPUT BUFFER           */
/******************************************/
     ROL    #8,D0;
     BSR.L  HSHK;
     MOVE.B HACIA+2,D0;          /*GET RECEIVED CODE INTO D0*/
     BCLR   #7,D0;
     BCLR   #15,D0;
     CMP.W  D0,CODE;
     BEQ.S  NRROR;
     MOVE.W CODE,D0;
     LEA    ERMSG1,A0;
     BSR.L  TYPE;
NRROR:  ADDQ.B #1,CODE;
     BCLR   #7,CODE;
     BEQ.S  WAITB;
     ADD.W  #$100,CODE;
WAITB:  BSR.L  TSHK;
     MOVE.B #SPACE,TACIA+2;
     MOVE   #BUFF,BPTR;
     BRA.L  WAIT3;              /*AND TEST LAST CHARACTER*/
/*****************************************************************/
ERMSG1: DC.B CRET; DC.B LFEED;
     DC 'TCOMP: !!ERROR!! CODE OUT OF SEQUENCE ON NEW WORD';
     DC.B CRET; DC.B LFEED; DC.B #0;
/******************************************/
/* IS THE CHARACTER A CODE          */
/******************************************/
QCODE:  BTST   #7,D0;           /*IS THE CHARACTER A CODE*/
     BEQ.L  WAIT3;              /*NO SO IGNORE IT*/
/******************************************/
/* READ THE REST OF THE CODE          */
/******************************************/
     BSR.L  HSHK;
```

```
    ROL    #8,D0;
    MOVE.B  HACIA+2,D0;
    BCLR   #7,D0;              /*GET RID OF THE CODE OFFSET*/
    BCLR   #15,D0;
    CMP.W  CODE,D0;            /*IS THE CODE IN RANGE*/
    BLE.S  NERR;
    LEA    ERMSG2,A0;
    BSR.L  TYPE;
    BRA.L  INIT;
/***********************************************************************/
ERMSG2: DC.B CRET; DC.B LFEED;
    DC 'TCOMP: !!FATAL ERROR!! CODE OUT OF RANGE ON WORD';
    DC.B CRET; DC.B LFEED; DC.B #0; DC.B #0;
/******************************************/
/* SEARCH THE DICTIONARY AND PRINT THE    */
/* CHARACTERS                             */
/******************************************/
NERR:  MOVE   SLP,SP;             /*AND START SEARCHING FOR IT*/
    SUBQ.W #1,D0;              /*IT MAY BE THE FIRST ENTRY*/
    BMI.S  FOUND;
NXTENT: MOVE  (SP),SP;            /*SKIP UNTIL WE REACH IT*/
    DBF    D0,NXTENT;
FOUND: ADDQ.W #1,D0;              /*ZERO D0*/
/******************************************/
/* UPDATE THE USAGE COUNT AND SWAP IF     */
/* NECESSARY                              */
/******************************************/
    ADDI   #1,8(SP);           /*INCREMENT THE USAGE COUNT*/
    MOVE.B $E(SP),CHCTR1;      /*GET THE NUMBER OF CHARACTERS*/
    SUBQ.B #1,CHCTR1;          /*FOR THE DBF*/
    MOVE   #$F,BPTR;           /*ADDRESS OF THE CHARACTERS*/
    ADD    SP,BPTR;
WAIT5: BSR.L  TSHK;               /*PRINT THE NEXT CHARACTER*/
    MOVE.B (BPTR)+,TACIA+2;
    DBF    CHCTR1,WAIT5;       /*TILL NO MORE LEFT*/
    CLR    CHCTR1;             /*ZERO D1*/
    BSR.L  TSHK;               /*PRINT A FOLLOWING SPACE*/
    MOVE.B #SPACE,TACIA+2;
    BSR.S  SWP;
    MOVE   #BUFF,BPTR;
    BRA.L  WAIT3;              /*AND READ ANOTHER CHARACTER*/
/******************************************/
/*THIS IS THE SUBROUTINE THAT SWAPS       */
/* DICTIONARY ENTRIES                     */
/******************************************/
SWP:   MOVE   SP,(DP)+;
STRT:  MOVE   4(CRNT),PREV;       /*GET ADDRESS OF PREVIOUS */
                               /*ENTRY INTO A2 */
    MOVE   CRNT,D0;            /*CHECK FOR START OF LIST */
    ADDQ   #4,D0;
    CMP    4(CRNT),D0;
    BNE.S  SW;                 /*IT IS NOT START SO SWAP*/
    MOVE   CRNT,SLP;           /*UPDATE START POINTER */
    BRA.S  NOSWAP;             /*AND EXIT */
SW:    MOVE   8(CRNT),D0;         /*IS A SWAP NECESSARY */
    CMP    4(PREV),D0;
    BLE.S  NOSWAP;             /*NO */
    MOVE   (CRNT),ENEXT;       /*YES SO GET OF NEXT ENTRY */
    MOVE   (PREV),PPREV;       /*AND PREV PREV ENTRY */
    MOVE   PPREV,4(CRNT);      /*ALTER BACK LINK */
    MOVE   ENEXT,-4(PREV);     /*FWD LINK OF PREV ENTRY */
```

```
      MOVE   PREV,(CRNT);          /*BACK LINK OF THIS ENTRY */
      SUBI   #4,(CRNT);
      CMP    #0,ENEXT;          /*ENEXT=0 IF LAST ENTRY*/
      BNE.S  NLSTENT;
      MOVE   (CRNT),LEP;
      BRA.S  LASTENT;
NLSTENT: MOVE   PREV,4(ENEXT);        /*BACK LINK OF NEXT ENTRY */
LASTENT: MOVE   CRNT,(PREV);        /*FORWARD LINK OF PREV */
      ADDI   #4,(PREV);
      CMP    PREV,PPREV;          /*PREV=PPREV IF IT IS FIRST */
      BNE.S  NFRST;            /*DONT ALTER THE START LINKS*/
      MOVE   CRNT,4(CRNT);          /*SET UP SO THAT BACK PTR*/
      ADDI   #4,4(CRNT);          /*POINTS TO ITSELF*/
      BRA.S  FRST;            /*AVOID THE BACK LINK ADJUST*/
NFRST:  MOVE   CRNT,-4(PPREV);       /*FORWARD LINK OF PREV PREV */
FRST:   MOVE.W $C(CRNT),D0;         /*SWAP THE CODES*/
      MOVE.W 8(PREV),$C(CRNT);
      MOVE.W D0,8(PREV);
      BRA.S  STRT;
NOSWAP: MOVE   -(DP),SP;          /*RESTORE THE REGISTERS */
      RTS;               /*AND RETURN */
/* THE NEXT SUBROUTINE PRINTS A MESSAGE STARTING AT ADDRESS IN A0 ONTO
THE TERMINAL . THE STRING IS TERMINATED BY A NULL */
TYPE:   BTST   TDRE,TACIA;
      BEQ.S  TYPE;
      MOVE.B (A0)+,TACIA+2;
      BNE.S  TYPE;
      RTS;
/*THE NEXT SUBROUTINE DISPLAYS THE COMPRESSION STATISTICS*/
DISPST: MOVE   TRCH,D0;          /*GET THE DIFFERENCE INTO D0*/
      SUB    RECCH,D0;
      BGE.S  NONEG;            /*CHECK THE SIGN OF THIS*/
NUP:    BTST   TDRE,TACIA;          /*FLAG A NEGATIVE NUMBER*/
      BEQ.S  NUP;
      MOVE.B '-',TACIA+2;
      NEG    D0;             /*AND MAKE IT POSITIVE*/
NONEG:  MULU   #100,D0;           /*MAKE THE DIFFERENCE INTO*/
      DIVU   TRCH,D0;           /*A PERCENTAGE*/
      AND    #$FF,D0;           /*MASK OUT THE LOW BYTE*/
      CMP.B  #100,D0;           /*CHECK IF 100% !!!!!*/
      BNE.S  NOONE;
WAITF:  BTST   TDRE,TACIA;          /*IF WAS A 1 THEN PRINT IT*/
      BEQ.S  WAITF;
      MOVE.B '1',TACIA+2;
      SUB    #100,D0;
NOONE:  DIVU   #10,D0;           /*PRINT REST OF THE NUMBER*/
      ADD.B  '0',D0;
WAITE:  BTST   TDRE,TACIA;
      BEQ.S  WAITE;
      MOVE.B D0,TACIA+2;
      SWAP   D0;             /*THE UNITS*/
      ADD    '0',D0;
WAITD:  BTST   TDRE,TACIA;
      BEQ.S  WAITD;
      MOVE.B D0,TACIA+2;
MARK:   BTST   TDRE,TACIA;          /*AND RETURN*/
      BEQ.S  MARK;
      MOVE.B '%',TACIA+2;
      RTS;
```

APPENDIX L

SPREAD SPECTRUM NETWORK SOFTWARE LISTING

```
/*
PROGRAM TO SET UP THE PROTOTYPE SPREAD SPECTRUM NETWORK RECEIVER AND
TRANSMITTER. IN THE EXAMPLE SHOWN BELOW, THE ADDRESSES OF THE
TRANSMITTER MEMORY, THE RECEIVER MEMORY AND THE ASYNCHRONOUS
COMMUNICATIONS INTERFACE ARE FOR THE VME BUS BASED VERSION. FOR THE
KDM
BASED VERSION IT IS ONLY NECCESSARY TO CHANGE THESE VALUES.
*/
TXRAM:  EQU   @$500000;    TRANSMITTER MEMORY
RXRAM:  EQU   @$504000;    RECEIVER MEMORY
STACK:  EQU   @$A00;       PROCESSOR RETURN STACK
TACIA:  EQU   @$0C0080;    COMMUNICATIONS INTERFACE
ORIGIN: EQU   @$1000;      START OF THE PROGRAM
/*
INITIALISE PROGRAM VARIABLES
*/
PROG:: ORG   ORIGIN;
     LEA   STACK,A7;
     LEA   MSG0,A0;
/*
PRINT STARTUP MESSAGE ON THE VDU
*/
     BSR   TYPE;
     LEA   MSG1,A0;
     BSR   TYPE;
/*
SETUP THE TRANSMITTER AND CONFIRM ON VDU
*/
     BSR   TXSET;
     LEA   MSG2,A0;
     BSR   TYPE;
/*
SET UP THE RECEIVER AND CONFIRM ON VDU
*/
     BSR   RXSET;
     LEA   MSG3,A0;
     BSR   TYPE;
/*
MONITOR ERRORS
*/
     BSR   ERRORS;;
/*
STARTUP MESSAGE
*/
MSG0:
     DC 'CODE MASKS FOR 127 BIT M-SEQUENCES ARE GIVEN BELOW';
     DC.W $0D0A;
     DC '7 1 0:        03';DC.W $0D0A;
     DC '7 3 0:        09';DC.W $0D0A;
     DC '7 3 2 1 0:    0F';DC.W $0D0A;
     DC '7 6 5 4 2 1 0:77';DC.W $0D0A;
     DC '7 6 5 2 0:    65';DC.W $0D0A;
     DC '7 6 3 1 0:    4B';DC.W $0D0A;
     DC '7 4 3 2 0:    1D';DC.W $0D0A;
```

```
    DC ’7 5 4 3 2 1 0:3F’;DC.W $0D0A;
    DC ’7 6 4 2 0:   55’;DC.W $0D0A;DC.W $0000;
/*

*/
MSG1: DC    ’SETTING UP TRANSMITTER RAM’;
    DC.W  $0D0A;
    DC.W  $0000;
/*

*/
MSG2: DC    ’SETTING UP RECEIVER CORRELATOR SEQUENCE ’;
    DC.W  $0D0A;
    DC.W  $0000;
/*

*/
MSG3:
DC    ’CODES SET UP, USE MEMORY LOCATION $500000 TO SELECT CODE’;
    DC.W  $0D0A;
    DC.W  $0000;;
/*

  GET A CHARACTER FROM THE ACIA INTO D0

*/
CHIN::
CHAR: EQU   D0;
    BTST  #0,TACIA;
    BEQ.S  CHIN;
    MOVE.B  TACIA+2,CHAR;
    RTS;;
/*

PRINT A CHARACTER HELD IN D0 TO THE TERMINAL ACIA

*/
CHOUT::
CHAR: EQU   D0;
    BTST  #1,TACIA;
    BEQ.S  CHOUT;
    MOVE.B  CHAR,TACIA+2;
    RTS;;
/*

TYPE A STRING AT THE ADDRESS HELD IN A0 TERMINATED BY A NULL

*/
TYPE::
CHAR: EQU   D0;
STRADD: EQU   A0;
    MOVE.B  (STRADD)+,CHAR;
    BEQ.S  EOSTR;
    BSR.S  CHOUT;
    BRA.S  TYPE;
EOSTR: RTS;;
/*

CALCULATE THE NEXT SHIFT REGISTER CONTENTS FROM THE OLD USING THE
FEEDBACK TAPS HELD IN D2, LEAVE THE RESULT IN D1
```

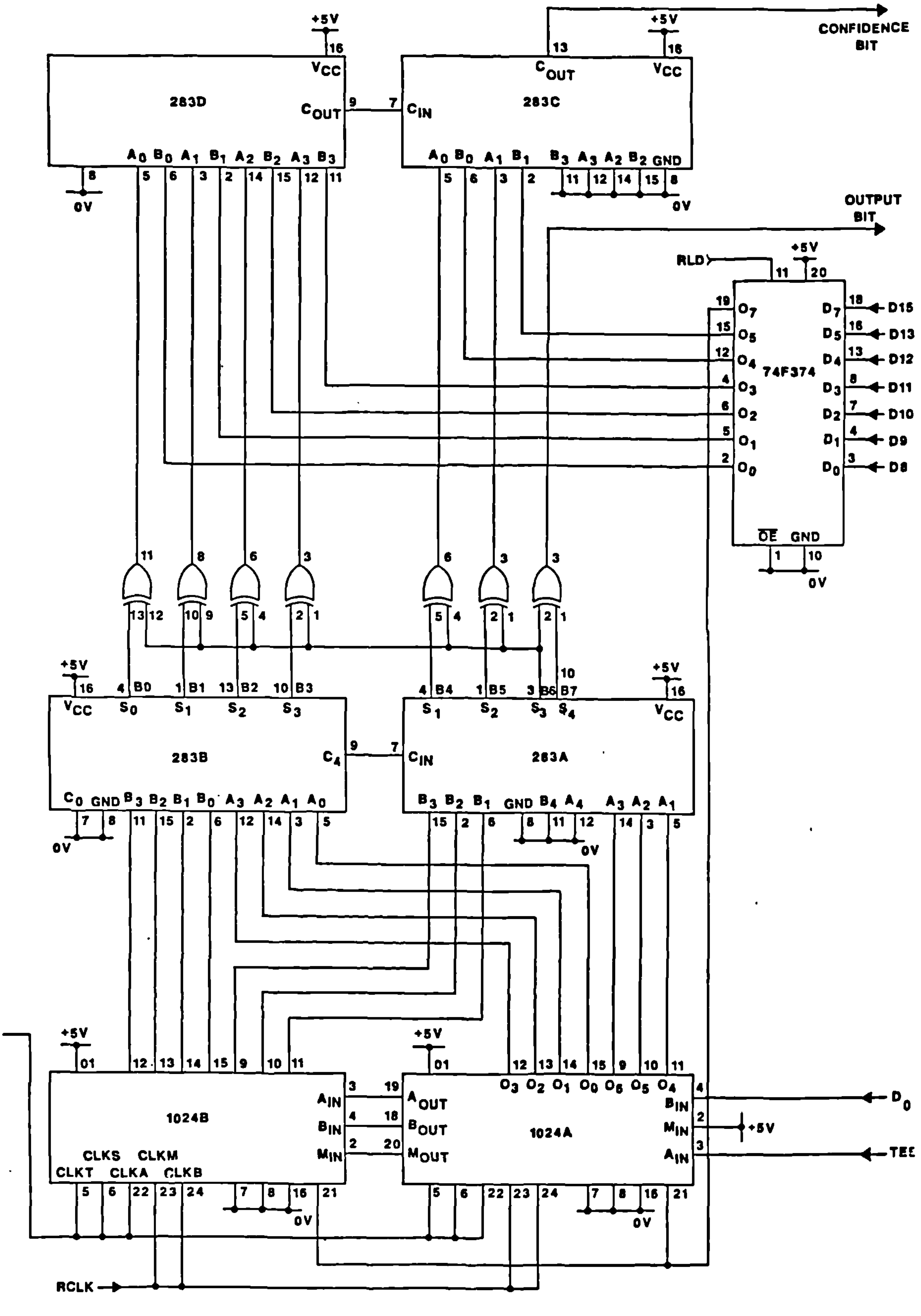
```
*/
CALC::
BITCTR: EQU   D0;
RESULT: EQU   D1;
MASK:  EQU   D2;
REG:  EQU   D3;
     MOVEM  D0/D3,-(A7);  /*GET REGISTERS FOR LOCAL VARIABLES*/

     AND   #$7F,RESULT;  /*CHECK FOR ZERO IN THE SHIFT REGISTER*/
     BNE.S  OK;
     MOVE.B #$7F,RESULT;  /*AND SET TO ALL 1'S IF ZERO*/
     BRA.S  EXIT;

OK:  MOVE.B RESULT,REG;  /*MAKE A LOCAL COPY OF THE SHIFT REG*/
     MOVEQ  #1,BITCTR;    /*INITIALISE THE BIT COUNTER*/
     AND.B  #1,RESULT;    /*AND ISOLATE THE ZERO'TH BIT OF THE REG*/

NEXTBIT:               /*FOR EACH BIT IN THE SHIFT REGISTER*/
     BTST   BITCTR,MASK;  /*TEST WHETHER TO CHECK THIS BIT*/
     BEQ.S  NOTAP;       /*AND SKIP IF NO TEST*/
     BTST   BITCTR,REG;   /*OTHERWISE, TEST THE VALUE OF THE BIT*/
     BEQ.S  CLEAR;

SET:  BTST   #0,RESULT;    /*SET, SO COMPARE WITH THE MOD 2 SUM*/
     BEQ.S  SSET;        /*REG BIT=1, MOD 2 SUM=0 SO SET TO 1*/

SCLEAR: BCLR  #0,RESULT;    /*CLEAR RESULT TO 0*/
     BRA.S  NOTAP;       /*AND TRY NEXT BIT*/

SSET:  BSET  #0,RESULT;    /*SET RESULT TO 1*/
     BRA.S  NOTAP;       /*AND TRY NEXT BIT*/

CLEAR:  BTST  #0,RESULT;    /*CLEAR, SO COMPARE WITH THE MOD 2 SUM*/
     BNE.S  SSET;        /*REG BIT=0, MOD 2 SUM=1 SO SET TO 1*/
     BRA.S  SCLEAR;       /*REG BIT=0, MOD 2 SUM=0 SO CLEAR TO 0*/

NOTAP: ADDQ  #1,BITCTR;    /*INCREMENT BIT COUNTER*/
     CMP.B  #7,BITCTR;    /*AND CHECK FOR ALL BITS TESTED*/
     BNE.S  NEXTBIT;      /*MORE BITS TO TEST SO DO THEM*/

     LSR.B  #1,REG;       /*SHIFT THE REGISTER AND INSERT BIT*/
     BCLR   #6,REG;
     BTST   #0,RESULT;
     BEQ.S  SKIP;
     BSET   #6,REG;

SKIP:  MOVE   REG,RESULT;   /*NEW SHIFT REGISTER IS IN RESULT*/
EXIT:  MOVEM  (A7)+,D0/D3;  /*RESTORE THE GLOBAL VARIABLES*/
     RTS;;
/*

SETUP THE TRANSMITTER RAM BY READING THE MASKS FOR EACH CODE FROM
MADD
AND THEN RESELECT CODE 0 AS THE CURRENT CODE

*/
TXSET::
CCTR:  EQU   D0;
SHREG:  EQU   D1;
MASK:  EQU   D2;
CODEPTR:EQU   D3;
```

```
DATA:  EQU   D4;
VALID: EQU   D5;
MPTR:  EQU   A0;
RAMPTR: EQU   A1;
    MOVEM  D0-D5/A0-A1,-(A7);/*MAKE SPACE FOR LOCAL VARIABLES*/
    CLR    CCTR;           /*AND INITIALISE THEM*/
    LEA    TXRAM,RAMPTR;
    LEA    MADD,MPTR;

NXCODE: MOVE.B (MPTR)+,MASK;    /*GET THE NEXT MASK*/
    MOVE.B CCTR,TXRAM;     /*SELECT CODE NUMBER*/
    MOVEQ  #1,CODEPTR;     /*AND INITIAL CODE OFFSET*/

OP:   MOVE   CODEPTR,SHREG;  /*NEXT BIT IS THIS PRODUCED BY THIS*/
    LSR    #1,SHREG;
    BSR    CALC;           /*CALCULATE THE NEXT BIT VALUE*/
    BTST   #6,SHREG;
    BEQ.S  CLEAR;
    MOVE.B #$F,DATA;
    BRA.S  SKIP;
CLEAR:  MOVE.B #0,DATA;
SKIP:   MOVE.B DATA,(RAMPTR,CODEPTR);/*SET MEMORY APPROPRIATELY*/

    MOVE.B (RAMPTR,CODEPTR),VALID;/*READ THE STORED VALUE */
    AND.B  #$F,VALID;
    CMP.B  DATA,VALID;
    BEQ.S  VDATA;
    TRAP   #13;            /*RETURN WITH REGISTER IF FAILURE*/

VDATA:  ADDQ   #2,CODEPTR;     /*ADDRESS NEXT STATE MACHINE*/
    CMP.B  #257,CODEPTR;   /*AND CHECK FOR END OF CODE*/
    BNE.S  OP;

    ADDQ   #1,CCTR;        /*SELECT NEXT CODE*/
    CMP    #8,CCTR;        /*AND CHECK FOR LAST CODE*/
    BNE.S  NXCODE;

    CLR.B  TXRAM;          /*SELECT CODE 0 AS OPERATIONAL CODE*/
    MOVEM  (A7)+,D0-D5/A0-A1;/*RESTORE VARIABLES*/
    RTS;;
/*

SET UP THE RECEIVER BY SENDING EACH BIT OF THE REQUIRED SEQUENCE TO
RXRAM LOWER BYTE, WITH THE CORRESPONDING MASK BIT ALONGSIDE IT. BIT 0
IS THE SEQUENCE BIT, BIT 1 IS THE MASK BIT. THE SHIFT REGISTER MASK
USED TO GENERATE THE RECEIVER SEQUENCE IS HELD IN LOCATION $808. */
RXSET::
BITCTR: EQU   D0;
SHREG: EQU   D1;
MASK:  EQU   D2;
    MOVEM  D0-D2,-(A7);    /*SAVE SPACE FOR LOCAL VARIABLES*/
    CLR    BITCTR;         /*BIT COUNTER*/
    MOVE.B RADD,MASK;      /*GET THE MASK*/
    CLR    SHREG;          /*AND CALCULATE THE FIRST BIT*/

NEXT:   MOVE.B SHREG,RXRAM+1;  /*STORE THE BIT AND */
    BSR    CALC;           /*AND CALCULATE THE NEXT BIT*/
```

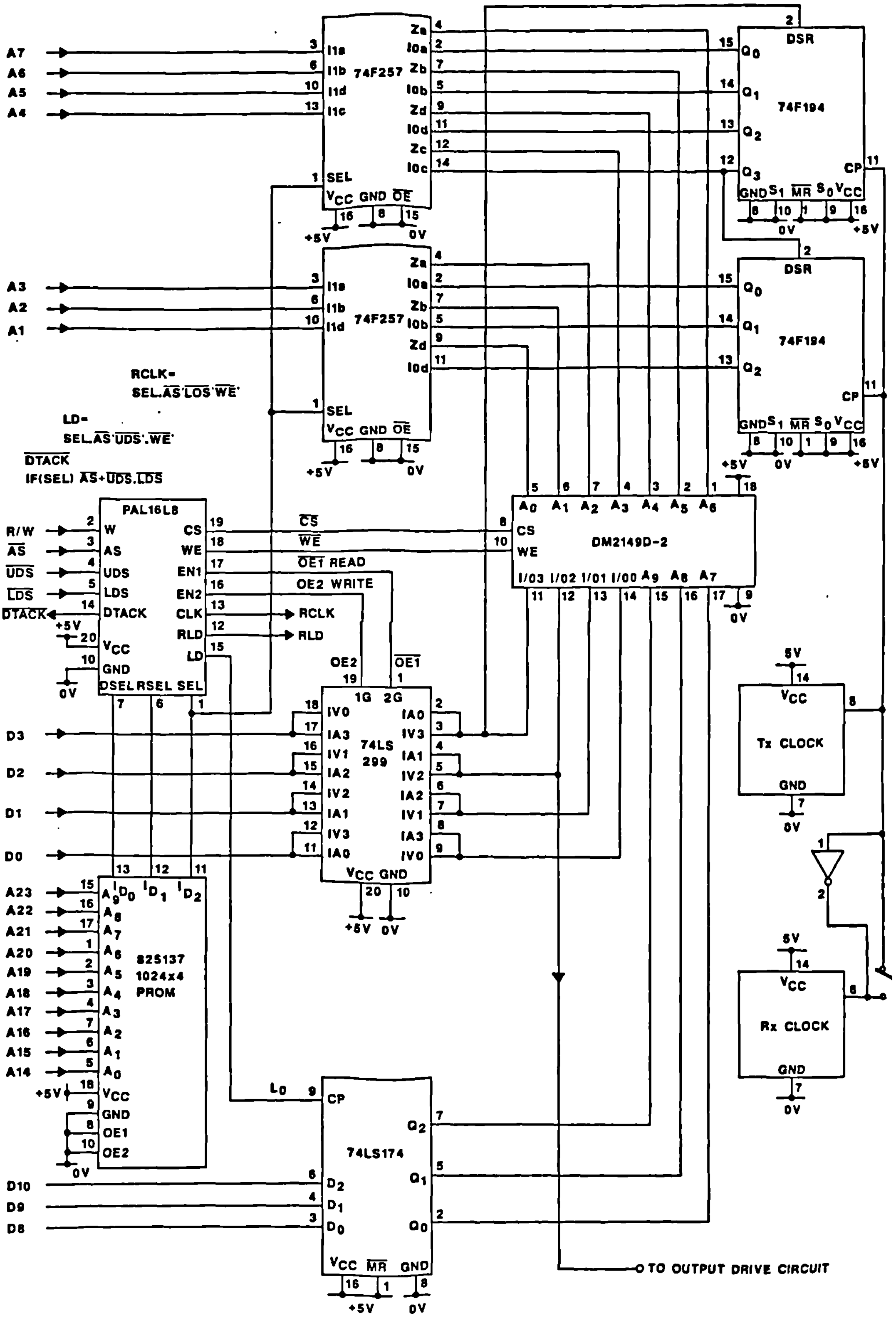
```
     ADDQ   #1,BITCTR;       /*MORE BITS?*/
     CMP    #129,BITCTR;
     BNE.S  NEXT;

     MOVE.B #$80,RXRAM;     /*LOAD THE SEQUENCE INTO REGISTER*/
     MOVE.B #$3F,RXRAM;     /*SET UP THE THRESHOLD TO 30+40=70*/

     MOVEM  (A7)+,D0-D2;    /*RESTORE VARIABLES*/
     RTS;;
/*
TRANSMITTER MASKS
*/
MADD: DC.B   $03;
     DC.B   $09;
     DC.B   $0F;
     DC.B   $77;
     DC.B   $65;
     DC.B   $4B;
     DC.B   $1D;
     DC.B   $3F;
/*
RECEIVER CODE MASK
*/
RADD: DC.B   $03;
     DC.B   $00;
/*

ERRORS COUNTS THE NUMBER OF MISSING BITS, BASED ON THE ASSUMPTION THAT
BITS ARE TRANSMITTED CONTINUOUSLY AND CONSECUTIVELY.
/*
ERRORS::
PIT:  EQU   @$E0000;
TCR:  EQU   $21;
TIVR: EQU   $23;
TSR:  EQU   $35;
CPRH: EQU   $27;
CPRM: EQU   $29;
CPRL: EQU   $2B;
     TRAP   14;
```

APPENDIX M

SPREAD SPECTRUM NETWORK CIRCUIT DIAGRAMS



+5V
16
VCC
283D
COUT
9
7
CIN
283C
13
COUT
CONFIDENCE BIT
8
0V
A0 B0 A1 B1 A2 B2 A3 B3
5 6 3 2 14 15 12 11
A0 B0 A1 B1 B3 A3 A2 B2 GND
5 6 3 2 11 12 14 15 8
OUTPUT BIT
RLD
11 20
74F374
O7 O5 O4 O3 O2 O1 O0
19 15 12 4 6 5 2
D7 D5 D4 D3 D2 D1 D0
18 16 13 8 7 4 3
D15
D13
D12
D11
D10
D9
D8
OE GND
1 10
11 8 6 3
13 12 10 9 5 4 2 1
6 3 3
5 4 2 1 2 1
10
4 B0 1 B1 13 B2 10 B3
4 B4 1 B5 3 B6 B7
S0 S1 S2 S3
S1 S2 S3 S4
283B
C4
283A
C0 GND B3 B2 B1 B0 A3 A2 A1 A0
7 8 11 15 2 6 12 14 3 5
B3 B2 B1 GND B4 A4 A3 A2 A1
15 2 6 8 11 12 14 3 5
01
12 13 14 15 9 10 11
12 13 14 15 9 10 11
O3 O2 O1 O0 O6 O5 O4
1024B
AIN
BIN
MIN
3 4 2
19 18 20
AOUT
BOUT
MOUT
1024A
BIN
MIN
AIN
4 2 3
D0
TEE
CLKS CLKM
CLKT CLKA CLKB
5 6 22 23 24 7 8 16 21
5 6 22 23 24 7 8 16 21
RCLK



74F257
74F257
74F194
74F194
DSR
DSR
PAL16L8
DM2149D-2
74LS 299
82S137 1024x4 PROM
74LS174
Tx CLOCK
Rx CLOCK
RCLK= SEL.AS LOS WE
LD= SEL.AS UDS.WE
DTACK IF(SEL) AS+UDS.LDS
CS
WE
OE1 READ
OE2 WRITE
RCLK
RLD
OE2
OE1
A7
A6
A5
A4
A3
A2
A1
R/W
AS
UDS
LDS
DTACK
D3
D2
D1
D0
A23
A22
A21
A20
A19
A18
A17
A16
A15
A14
D10
D9
D8
L0
+5V
0V
5V
TO OUTPUT DRIVE CIRCUIT

REFERENCES

[1] **Secret Communications System**, H. K. Markey, G. Antheil, August 1942, U.S. Patent Office, No. 2 292 387

[2] **Communication in the Presence of Noise**, Claude E. Shannon, January 1949, Proceedings of the IRE, pp. 10-21.

[3] **Theory and Practice of Error Control Codes**, Richard E. Blahut, 1983, New York, Addison-Wesley

[4] **An Introduction to Local Area Networks**, D. D. Clark, K. T. Pogran, D. P. Reed, November 1978, IEEE Press, Proceedings of the IEEE, pp. 1497-1517.

[5] **Computer Networks**, Andrew S. Tanenbaum, 1981, Prentice-Hall Inc., Englewood Cliffs, New Jersey

[6] **The ALOHA system - Another Alternative for Computer Communications**, H. Abramson, 1970, Proceedings FJCC, pp. 281-285

[7] **Defence Standard 00/19 Combat Systems Highway**, Ministry of Defence, 1975

[8] **Future Command Systems**, Admiralty Research Establishment, 1980

[9] **GKS Graphics Kernel System**, American National Standards Institute, 1980

[10] **Secret Telephony as a Historical Example of Spread-Spectrum Communication**, William R. Bennett, January 1983, IEEE Transactions on Communications, Volume 31, Pages 98-104

[11] **An Annotated Bibliography on Local Computer Networks**, J. F. Schoch, October 1979, Xerox PARC Technical Report SSL-79-5

[12] **Network for block switching of data,** J. R. Pierce, 1972, Bell System Technical Journal, Volume 51, Number 6

[13] **A digital loop communications system,** E. R. Hafner, Z. Nenadal, M. Tschanz, 1974 IEEE Transactions on Communications, Volume 22, Number 6, Pages 877-881

[14] **An experimental distributed switching system to handle bursty computer traffic,** W. D. Farmer, E. E. Newhall, 1969, Proceedings of the Association of Computing Machinery

[15] **IEEE project 802 Local Network Standards,** IEEE, 1983

[16] **Defence Standard 00-19/Issue 2, THE ASWE SERIAL HIGHWAY,** Ministry of Defence, June 1987

[17] **Data processing - open systems interconnection - basic reference model,** International Standards Organisation, Number 7498

[18] **OSI reference model- The ISO Model of Architecture for Open Systems Interconnection,** H. Zimmerman, April 1980, IEEE Transactions on Communications, Volume 28, Pages 425-432

[19] **Coherent Spread Spectrum Systems,** J. K. Holmes, Wiley Interscience

[20] **On Mutual Correlation of Sequences,** V. M. Sideln'ikov, 1971, Soviet Mathematical Dokl, Volume 12, Pages 197-201

[21] **Binary Sequence with a High Cross-correlation with each Member of a Subset of Gold Codes,** K. W. Yates, M. A. Beach, A. J. Copping, 28 August 1986, Electronics Letters, Volume 22, Number 18, Pages 930-932

[22] **Optimal Binary Sequences for Spread Spectrum Multiplexing,** Robert Gold, October 1967, IEEE Transactions on Information Theory, Volume 13, Number 4, Pages 619-621

[23] **Bent-Function Sequences**, John D. Olsen, Robert A. Scholtz, Lloyd R. Welch, November 1982, IEEE Transactions on Information Theory, Volume 28, Number 6, Pages 858-864

[24] **Maximal Families of Bent Sequences**, Abraham Lempel, Martin Cohn, November 1982, IEEE Transactions on Information Theory, Volume 28, Number 6, Pages 865-868

[25] **Evaluation of Correlation Parameters for Periodic Sequences**, M. B. Pursley, D. V. Sarwate, IEEE Transactions on Information Theory, July 1977, Pages 508-513

[26] **Crosscorrelation Properties of Pseudorandom and Related Sequences.** Dilip V. Sarwate, Michael B. Pursley, Proceedings of the IEEE, Volume 68, Number 5, May 1980, Pages 593-619

[27] **Comparison of Sequences for Local Area Networks Using Spread Spectrum Multiple Access**, Nabil A. Ismail, January 1988, Proceedings of the IEEE, Volume 76, Number 1, Pages 87-88

[28] **A 100 Mbs-1 Optical Fibre Ring Network**, F. Abdul Ghani, P. A. Davies, 2 June 1983, IEE Colloquium on Optical Fibre Local Networks, Pages 1-3

[29] **A Survey of One-Coincidence Sequences for Frequency Hopped Spread Spectrum Systems**, A. A. Shaar, P. A. Davies, Private Communication

[30] **Prime Sequences: quasi-optimal sequences for OR channel code division multiplexing**, A. A. Shaar, P. A. Davies

[31] **Optimisation by Simulated Annealing**, S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, Science, Number 220, Pages 671-680

[32] **Using Simulated Annealing to Design Good Codes**, Abbas A. Gamal, Lane A. Hemachandra, Itzhak Shperling, Victor K. Wei, IEEE Transactions on Information Theory, Volume 33, Number 1, Pages 116-123

[33] **Equations of State Calculations by Fast Computing Machines,** N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, Journal of Chemical Physics, Number 21, Pages 1087-1091

[34] **Stochastic Relaxation, Gibbs Distributions and the Beysian Restoration of Images,** S. Geman, D. Geman, IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 6, Pages 721-741

[35] **Non-stationary Markov Chains and Convergence of the Annealing Algorithm,** B. Gidas, Journal of Statistical Physics, Number 39, Pages 73-131

[36] **Schedule Optimisation with Probabilistic Search.** L. Davis, Frank Ritter, Proceedings of the 3rd IEEE Conference on Artificial Intelligence Applications

[37] **Seminumerical Algorithms, 2nd Edition of The Art of Computer Programming,** Donald E. Knuth, Section 3.2, 1981, Addison-Wesley

[38] **A Programmer's View of The Intel 432 System,** Elliot I. Organick, McGraw-Hill Book Company, New York, 1983.

[39] **CURT: The Command Interpreter Language for Flex,** I. M. Currie, J. M. Foster, September 1982, Procurement Executive, Ministry of Defence, RSRE, Memorandum 3522

[40] **Flex Firmware,** I. F. Currie, P. W. Edwards, J. M. Foster, September 1981, Procurement Executive, Ministry of Defence, Number 81009

[41] **FLEX: A Working Computer with an Architecture based on Procedure Values,** J. M. Foster, I. F. Currie, P. W. Edwards, July 1982, Procurement Executive, Ministry of Defence, RSRE Memorandum 3500

[42] **An Introduction to the Flex Computer System,** J. M. Foster, C. I. Moir, I. F. Currie, J. A. M. McDermid, P. W. Edwards, J. D. Morison, C. H. Pygott, October 1979, Procurement Executive, Ministry of Defence, Number 79016

[43] **Maximal Families of Bent Sequences,** Abraham Lempel, Martin Cohn, November 1982, IEEE Transactions on Information Theory, Volume 28, Number 6, Pages 865-868

[44] **Acquisition Time Performance of PN Spread-Spectrum Systems,** Jack K. Holmes, Chang C. Chen, August 1977, IEEE Transactions on Communications, Volume 25, Number 8, Pages 778-783

[45] **Spreadnet - A spread-spectrum local area network,** C. T. Spracklen, C. Smythe, M. A. Scott, N. A. Ismail, January 1987, Journal of the Institution of Electronic and Radio Engineers, Volume 57, Number 1, Pages 12-16

[46] **Direct Sequence Spread Spectrum Techniques in Local Area Networks,** Colin Smythe, 1985, Phd Thesis Durham University

[47] **A method for Obtaining Digital Signatures and Public Key Cryptosystem,** R. L. Rivest, A. Shamir, L. Adleman, Febuary 1987, Communications of the Association of Computing Machinery, Volume 21, Pages 120-126

[48] **Data Encryption Standard,** National Bureau of Standards, January 1977, Federal Information Processing Standard Publication No 46

[49] **Hiding Information and Receipts in Trap-Door Knapsacks,** R. C. Merkle, M. E. Hellman, September 1978, IEEE Transactions on Information Theory, Volume 24, Pages 525-530

[50] **Performance evaluation for phase-coded spread spectrum multiple access communication - part 1: system analysis,** M. B. Pursley, August 1977, IEEE Transactions on Communications, Volume 25, Number 8, Pages 795-799

[51] **Performance evaluation for phase-coded spread spectrum multiple access communication - part 2: code sequence analysis,** M. B. Pursley, August 1977, IEEE Transactions on Communications, Volume 25, Number 8, Pages 800-803

[52] **A Study of Code Division Multiple Access With Reference To fiber Optic Local Area Networks,** A. A. Shaar, April 1985, PhD Thesis, University of Kent

[53] **Bounds on Crosscorrelation and Autocorrelation of Sequences,** Dilip V. Sarwate, November 1979, IEEE Transactions on Information Theory, Volume 25, Number 6, Pages 720-724

[54] Fast Fourier Transform and Convolution Algorithms, H. J. Nussbaumer, Springer Verlag, New York, 1982