

Durham E-Theses

Application of bit-slice microprocessors to digital correlation in spread spectrum communication systems

Ismail, Nabil Abd-el-wahid

How to cite:

Ismail, Nabil Abd-el-wahid (1983) *Application of bit-slice microprocessors to digital correlation in spread spectrum communication systems*, Durham theses, Durham University. Available at Durham E-Theses
Online: <http://etheses.dur.ac.uk/698/>

Use policy

The full-text may be used and/or reproduced, and given to third parties in any format or medium, without prior permission or charge, for personal research or study, educational, or not-for-profit purposes provided that:

- a full bibliographic reference is made to the original source
- a [link](#) is made to the metadata record in Durham E-Theses
- the full-text is not changed in any way

The full-text must not be sold in any format or medium without the formal permission of the copyright holders.

Please consult the [full Durham E-Theses policy](#) for further details.

Academic Support Office, Durham University, University Office, Old Elvet, Durham DH1 3HP
e-mail: e-theses.admin@dur.ac.uk Tel: +44 0191 334 6107
<http://etheses.dur.ac.uk>

**APPLICATIONS OF BIT-SLICE MICROPROCESSORS
TO DIGITAL CORRELATION
IN SPREAD SPECTRUM COMMUNICATION SYSTEMS**

by

Nabil Abd-el-wahid Ismail, B.Sc., M.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 & Electronics

1982

The copyright of this thesis rests with the author.
No quotation from it should be published without
his prior written consent and information derived
from it should be acknowledged.



**Applications of Bit-Slice Microprocessors to Digital
Correlation in Spread Spectrum Communication Systems**

Nabil Abd-el-wahid Ismail

ABSTRACT

This thesis describes the application of commercially available microprocessors and other VLSI devices to high-speed real-time digital correlation in spread spectrum and related communication applications. Spread spectrum communications are a wide-band secure communication system that generate a very broad spectral bandwidth signal that is therefore hard to detect in noise. They are capable of rejecting intentional or unintentional jamming, and are insensitive to the multipath and fading that affects conventional high frequency systems. The bandwidth of spread spectrum systems must be large to obtain a significant performance improvement. This means that the sequence rate must be fast and therefore very fast microprocessors will be required when they are used to perform spread spectrum correlation. Since multiplication cannot be performed efficiently by microprocessors considerable work, since 1974, has been published in the literature which is devoted to minimising the requirement of multiplications in digital correlation and other signal processing algorithms. These fast techniques are investigated and implemented using general-purpose microprocessors. The restricted-bandwidth problem in microprocessor-based digital correlator has been discussed. A new implementation is suggested which uses bit-slice devices to maintain the flexibility of microprocessor-based digital correlation without sacrificing speed. This microprocessor-based system has been found to be efficient in implementing the correlation process at the baseband in the digital domain as well as the post-correlation signal processing- demodulation, detection and tracking, especially for low rate signals. A charge coupled-device is used to obtain spectral density function. An all-digital technique which is programmable for any binary waveform and can be used for achieving initial acquisition and maintaining synchronisation in spread spectrum communications is described. Many of the practical implementation problems are discussed. The receiver performance, which is measured in terms of the acquisition time and the bit-error rate, is also presented and results are obtained which are close to those predicted in the system simulations.

ACKNOWLEDGEMENTS

I would like to express my gratitude for the consistent guidance and the constructive criticism of Dr C.T.Spracklen during the project. I am indebted to him for his supervision, encouragement, and helpful advice and for his help in securing equipment.

I would also like to thank Professor G.G.Roberts for allowing me to use the facilities of the Department of Applied Physics and Electronics, the University of Durham, and I am grateful to members of the workshop for their willingness to provide their skill and advice.

My thanks are also due to my colleagues in the Digital Electronics Group for many useful discussions and times spent together. The help of the staff of the Computer Unit of the University of Durham as well as the staff of the Science Library is also gratefully acknowledged.

My appreciation is also extended to the Egyptian Mission Department and the University of Menoufia for an award of a research studentship and for providing financial support over more than three years.

Finally, I am particularly indebted to my wife Afrah for her understanding, patience and support and to my parents, brothers and sisters for moral and financial support and to my daughter Marwa for her unceasing distraction which formed a source of much relaxation.

To Afrah, Marwa and Aeymn

CONTENTS

ABSTRACT

ACKNOWLEDGEMENTS

Glossary of Terms

CHAPTER 1 Introduction

- 1.1 History
- 1.2 Spread spectrum techniques
- 1.3 Problems of spread spectrum systems
- 1.4 Synchronisation problems
- 1.5 Practical implementation problems
- 1.6 Bit-slice microprocessors and spread spectrum systems
- 1.7 Conclusion

CHAPTER 2 Digital Correlation Techniques Using Microprocessors

- 2.1 Introduction
- 2.2 Digital correlation
- 2.3 Transform analysis
 - 2.3.1 Correlation using rectangular transforms
- 2.4 Implementations
 - 2.4.1 Implementation using an Intel-8080 microprocessor system
 - 2.4.2 Correlation on TMS9900 microcomputer
- 2.5 Binary correlation
- 2.6 Real-time power spectrum density
 - 2.6.1 Chirp-Z transform algorithm
 - 2.6.2 Hardware implementation
- 2.7 Conclusion

CHAPTER 3 Bit-Slice Microprocessor System

- 3.1 Introduction
- 3.2 System organisation
- 3.3 2901 ALU/Register slices
 - 3.3.1 Architecture
 - 3.3.2 2901-slices interconnection
- 3.4 Microprogram control
 - 3.4.1 Microprogram memory
 - 3.4.2 Microprogram counter
- 3.5 Condition code select logic
- 3.6 I/O external registers handling
 - 3.6.1 The 'To' decoder
 - 3.6.2 The 'From' decoder

- 3.6.3 The control decoder
- 3.7 Input/Output buffer memory
- 3.8 System clock
- 3.9 Conclusion

CHAPTER 4 System Microprogramming Features

- 4.1 Microprogramming
- 4.2 Microinstruction format
- 4.3 Microinstruction implementation
- 4.4 Microinstruction sequencing
 - 4.4.1 Sequential execution
 - 4.4.2 Skip control
 - 4.4.3 Multiple sequences
 - 4.4.4 Start address
- 4.5 Special microassembler
- 4.6 Software simulator
- 4.7 PROM programming
- 4.8 Development test equipments
- 4.9 Test software
 - 4.9.1 Control decoder test
 - 4.9.2 "To" and "From" decoder test
 - 4.9.3 2901-slices internal registers test
 - 4.9.4 Up shift test
 - 4.9.5 Down shift test
 - 4.9.6 2901 ALUs arithmetic operation test
 - 4.9.7 Carry control test
 - 4.9.8 FIFO control tests
 - 4.9.9 FIFO data tests
 - 4.9.10 Output enable test
- 4.10 Conclusion

CHAPTER 5 Implementation of Direct Sequences by Microprocessors

- 5.1 Introduction
- 5.2 Pseudo-noise sequences
 - 5.2.1 Generation and properties
 - 5.2.2 Correlation functions and power spectra of codes
- 5.3 Implementing the feedback shift register on a microprocessor
- 5.4 Sequence inversion keying (SIK) modulation
- 5.5 Synchronisation
 - 5.5.1 Initial acquisition techniques
 - 5.5.2 Correlation process
- 5.6 Tracking
 - 5.6.1 Delay-lock loop correlator
 - 5.6.2 Implementation
- 5.7 Conclusion

CHAPTER 6 Transmitter and Receiver Design

- 6.1 Introduction
- 6.2 Transmitter

- 6.2.1 Data acquisition
- 6.2.2 FIFO on transmit
- 6.3 Spreading
- 6.4 Transmitter software
- 6.5 Receiver
 - 6.5.1 FIFO on receive
 - 6.5.2 Search/lock strategy
 - 6.5.3 Receiver software
- 6.6 Clock frequency effects
- 6.7 Data recovery
- 6.8 Conclusion

CHAPTER 7 System Performance & Experimental Results

- 7.1 Introduction
- 7.2 System performance
 - 7.2.1 Acquisition time measurements
 - 7.2.2 Bit error-rate measurements
- 7.3 Noise channel simulation
 - 7.3.1 The microprocessor
 - 7.3.2 Hardware description
 - 7.3.3 Implementation
- 7.4 Experimental results
- 7.5 Conclusion

CHAPTER 8 Conclusion

APPENDIX A References

APPENDIX B Program Listings

Glossary of Terms

ADC	Analogue to Digital Converter
A/D	Analogue to Digital
AJ	Antijamming
ALU	Arithmetic Logic Unit
BER	Bit Error Rate
BPSK	Biphase Phase Shift Keying
CCD	Charge Coupled Device
CCP	Cyclic Convolution Property
CPE	Central Processing Element
CPU	Central Processing Unit
CRT	Chinese Remainder Theorem
CZT	Chirp-Z Transform
DAC	Digital to Analogue Converter
D/A	Digital to Analogue
DFT	Discrete Fourier Transform
DLL	Delay-Lock Loop
DMA	Direct Memory Access
FFT	Fast Fourier Transform
FIFO	First Input First Output
FIS	Fixed Instruction Set
G_p	Process Gain of Spread Spectrum System
IC	Integrated Circuit
I/O	Input/Output
LSB	Least Significant Bit
LSI	Large Scale Integration
m-sequence	Maximal Length Pseudonoise Sequence
MSI	Medium Scale Integration

NTT	Number Theoretic Transform
P_D	Probability of Detection
P_E	Probability of Error
PIA	Peripheral Interface Adapter
PN	Pseudo-Noise Sequence
PROM	Programmable Read Only Memory
PSK	Phase Shift Keying
QPSK	Quadrphase Phase Shift Keying
RAM	<i>Random</i> Access Memory
ROM	<i>Read</i> Only Memory
SIK	Sequence Inversion Keying
SAW	Surface Acoustic Wave
TDMA	Time-Division Multiple Access
VCO	Voltage Controlled Oscillator
VLSI	Very Large Scale Integration
WFTA	Winograd Fourier Transform Algorithm

CHAPTER 1

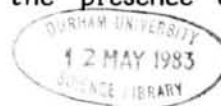
Introduction

1.1 History

The spread spectrum technique has evolved from a desire by communication system users to protect their messages against detection by unauthorised users and provide reasonable immunity to interference for the desired user. Spread spectrum is a means of transmission in which the basic signal characteristics are:

- (i) The carrier is a pseudonoise, wide-band signal.
- (ii) The bandwidth of the carrier is much wider than the minimum bandwidth required to transmit the information being sent. As a minimum, a voice signal can be sent with amplitude modulation (AM) in a bandwidth only twice that of the information itself. A spread spectrum system, on the other hand, has a modulated signal bandwidth that is at least 10 to 100 times that of the information bandwidth.
- (iii) Reception is achieved by crosscorrelation of the received wide-band signal with a synchronously generated replica of the wide-band carrier. This is used for despreading and subsequent data recovery. Furthermore, in a spread spectrum system, the information data rate does not dictate the bandwidth of the modulated signal.

The concept of spread spectrum technology has been known since Shannon's theorem (1) came to the light in 1940's. Costas work in 1959 (2) indicates that the idea of employing coded wide-band signals for communicating in the presence of noise



could be implemented in some systems. Golomb's work (3) in the area of codes used in communication and pseudonoise generation which was started in 1956 has offered a further recognition to the field. The first high performance electronic correlator by Lee which was described with other correlation techniques by Lange (4) in the early 1960's, was the important step towards the ability to mechanise the correlation operation precisely, which is essential in building high-performance spread spectrum systems. At that time, the initial application have been to military antijamming (AJ) communications, to guidance systems and other related applications which were employed with conventional vacuum tube technology. The prime advances in spread spectrum performance have come about primarily as a result of the availability of solid state components. The advent of high speed, high gain transistors in the 1960's gained the subject a new area of applications such as the navigation (ranging and direction finding) area and space exploration programs. Certain investigations and systems were carried out, mainly in the United States, during the 1960's and early 1970's (5)-(8), but were largely abandoned in favour of satellite and satellite-aircraft communications.

The recent advances in digital integrated circuits (IC) technology and VLSI (very large scale integration)/LSI (large scale integration) packages have enabled substantial reductions to be made in both the size and the cost of communication systems. At the same time, new analogue device developments, such as surface acoustic wave (SAW) and charge coupled devices (CCD),

have been introduced. It seems only logical that spread spectrum systems also benefit from such developments (9). On the other hand, much work (10), (11) has been performed in the area of developing special "acquirable" codes which have the required length for the system under question, but which also have synchronisation properties (excellent autocorrelation and crosscorrelation properties) that permit acquisition to be searched out without traversing the entire code length.

Although the current application for spread spectrum continue to be primarily for military communications, there is an increasing interest in the use of this techniques such as for mobile radio networks and some specialised applications in satellites. Most recently they have been successfully applied to multiple access situations involving many users simultaneously (12).

At present there is a limited amount of information (unclassified) in the published literature which outlines the applications of the spread spectrum concept to a communication system from an overall system viewpoint. The general details on practical performance are few with isolated theoretical investigations of some of the problems.

In this thesis we confine ourselves to principles related to the applications of VLSI technology to the design and analysis of those parts of a spread spectrum communications system concerned with synchronisation acquisition and tracking. For this complete transmitter and receiver systems were developed using the latest state-of-the-art technology, the bipolar bit-slice

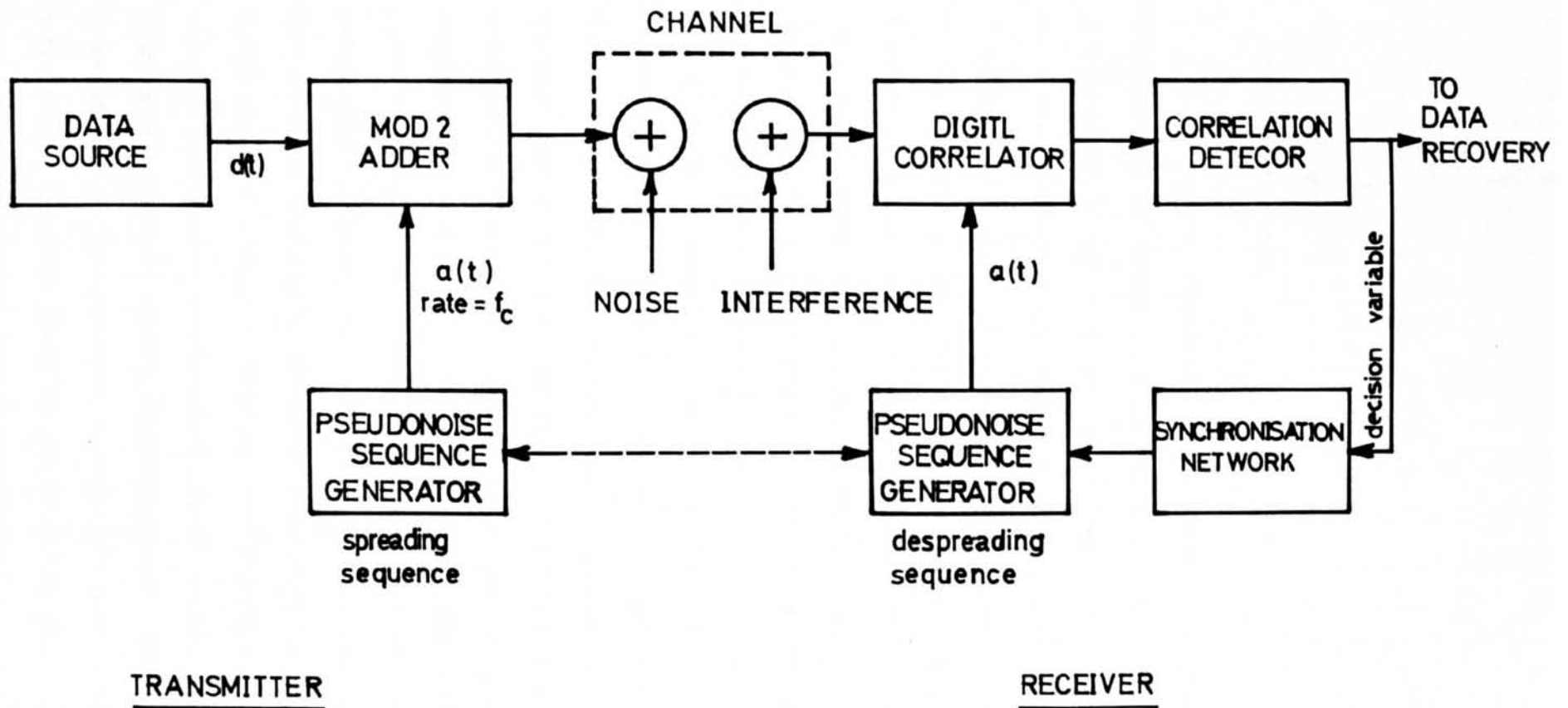
microprocessors. It is shown that those parts of the receiver which previously required large amounts of expensive analogue or discrete equipment can be realised at lower cost and with increased flexibility using all digital techniques.

1.2 Spread Spectrum Techniques

To illustrate the principle of a spread spectrum system the block diagram of a transmitter and receiver is shown in Figure (1.1). When viewed as a system composed of many sub-systems the individual units of a spread spectrum system are in many ways identical with sub-systems in conventional communication systems. From a theoretical viewpoint there is no reason why analogue waveforms should not be considered for bandwidth expansion in spread spectrum systems. There are, however, constraints on the desired correlation properties of spreading waveforms. In an ideal spread spectrum system, waveforms with good autocorrelation properties and orthogonality between the various waveforms are desired. It is generally accepted that in practical systems the best that can be achieved is waveforms which exhibit a two level autocorrelation function and low values of crosscorrelation.

The study of binary sequences is comprehensive in the literature (3), (10), (11). It is mainly due to this, and the ease of generation of maximal length pseudonoise sequences (m-sequence) using shift registers, that digital spreading waveforms are widely used. There are many techniques to achieve spectrum spreading (6), these are;

- (1) direct sequence modulated ✓
- (2) frequency hopping



FIGURE(1.1) DIRECT SEQUENCE SPREAD SPECTRUM SYSTEM FOR TRANSMITTING A BINARY DATA (BASEBAND).

- (3) time hopping
- (4) pulse-FM or chirp

In a direct sequence system (which is also called PN-sequence), as shown in Figure (1.1), the data information is combined with a high clock rate m-sequence before modulation on the carrier, resulting in direct bandwidth expansion. Frequency hopping (FH) has evolved from the idea that a good way to prevent an unintended receiver from receiving a message, or to prevent interference, is to move the carrier frequency of the information signal in a pseudorandom manner. Instead of directly modulating the carrier, the code sequence is used to switch the carrier frequency in a pseudorandom manner. The synchronisation acquisition in a frequency hopping scheme is faster due to a larger duration of the hopping chip. However, this is a disadvantage when the overall system requires any form of accurate time of arrival measurements. The hardware required to implement such schemes is always far more complicated and expensive to implement. Like frequency hopping, time hopping systems control their transmission time and period from a pseudonoise sequence. Time hopping is generally not used alone but is always employed in conjunction with frequency hopping and direct sequence methods to eliminate time dependent interference or allow time-division multiple-access (TDMA) system. Unlike the other spread spectrum systems, pulse-FM or chirp does not employ m-sequences. The operation is based on pulse compression achieved by frequency sweep at the transmitter and compression using a dispersive matched filter at the receiver.

Hybrid spread spectrum systems are possible by combining these basic techniques.

There are many advantages for spreading a signal's bandwidth and then collapsing it through correlation with a stored reference signal contained in the receiver:

- (a) selective addressing
- (b) low power density signals
- (c) inherent message privacy
- (d) code division multiple user access
- (e) high resolution ranging
- (f) interference rejection
- (g) possible operation with adverse transmission distortion
- (h) accurate universal timing

Selective addressing is possible through the assignment of a particular m-sequence (code) to a receiver. The low power density of spread spectrum signals results from the wideband for transmission and causes low interference to other users. The coded format of spread spectrum systems offers privacy in communication from the casual listener. The use of different codes allows multiple users in a spread spectrum communication system. The good correlation properties of m-sequences in conjunction with the wide bandwidth used for transmission allow accurate ranging of transmitter or receiver. The interference rejection occurs as a result of the despreading necessary for the operation of a spread spectrum receiver. In a particular system, the ratio of spread or transmitted bandwidth to the rate of the information sent is called the "process gain" (G_p) (7) of that

system. This factor is the measure of the interference rejection in that system. The large bandwidth of spread spectrum systems suggests that a form of frequency diversity is available in the system which may combat distortions due to the transmission medium. It should be noted that these advantages are not always available and rely on reasonable synchronisation of the receiver with respect to the transmitter.

1.3 Problems of Spread Spectrum Systems

Most of the problems discussed in this section are not unique to spread spectrum systems. Some of them are associated with communication via the propagation medium. The object is to provide an appreciation of the general problems relevant to the subsequent practical limitations in spread spectrum receiver system. The main problems are:

- (1) Interference and noise
- (2) Distortion due to the transmission medium
- (3) Synchronisation problems
- (4) Practical implementation problems

These problems are of equal concern in that; either they corrupt the received data or they affect the system performance. Interference and noise in spread spectrum systems are a result of:

(1) Interference due to other spread spectrum users; this is increases as more users utilise the same RF band. It is required to devise orthogonal spreading functions for the numerous users using the same frequency band.

(2) Interference due to the geometry of links; in certain

instants an interfering transmitter may be closer to a receiver than the desired transmitter. In this situation the wanted signal will be received in a high level of interference. This is known as the "near-far" problem.

(3) Interference from conventional radio systems; to a considerable extent a spread spectrum system has the ability to reject interference from narrowband systems. This is possible within the jamming margin of the spread spectrum. The jamming margin is the power level above the spread spectrum signal that a narrowband interferer can be discriminated against, for a desired output signal to noise ratio, including implementation losses (7).

(4) Man-made impulsive noise; this is produced from machinery, fluorescent lights, power switching appliances etc.

(5) Atmospheric and receiver noise; this may need consideration as in a conventional receiver system, depending on the frequency band of interest.

Distortions due to the transmission medium, on the other hand, are dependent on the propagation mechanisms of radio waves in a known environment.

1.4 Synchronisation Problems

The problem of synchronisation is of major concern in the design and implementation of spread spectrum systems. This is because the interference rejection capabilities rely on adequate synchronisation of the spreading and despread waveforms. By synchronisation we mean that, the signal seen by the receiver must be precisely correlated in time with a locally generated

reference signal.

The main sources of uncertainty, with respect to synchronisation, in spread spectrum systems are those that are time or frequency dependent. Time uncertainty includes any propagation time delay due to unknown range. Frequency uncertainty is due to the instability of the frequency sources used in both transmitter and receiver. Code, phase, and carrier frequency are the frequency uncertainty. Doppler-related frequency errors often cannot be predicted and may ~~be~~ affect both code rate and carrier frequency. Another consequence of frequency uncertainty may also exist, any clock rate offset is accumulated in code phase offset. These factors lead to a degradation of the synchronisation performance because; (i) not all main correlation peaks are detected, i.e., the detection probability P_D , and (ii) false set impulses occur, as false alarms are generated at instants at which correlation subpeaks are above a certain threshold away from the main peaks.

The time required for achieving synchronisation between transmitting and receiving units has become the major factor limiting usage of spread spectrum systems. Reduction of synchronisation time is limited by the maximum search rate a receiving unit is capable of achieving and the length of the m-sequence to be used. Maximum search rate, is limited by the recognition time of the receiver's correlation detection circuits. The receiver must be able to recognise correlation and stop the search process before the point of code synchronisation is passed. This requires that the bandwidth of the correlation

detectors must be commensurate with the autocorrelation requirements of the m-sequence used.

The synchronisation process is generally separated into two phases, initial synchronisation and tracking. The initial synchronisation phase determines the timing of an incoming signal and brings the receiver into initial alignment, the tracking phase holds it in alignment. Initial synchronisation is frequently achieved by means of a single synchronisation preamble at the beginning of each transmission. The structure of the preamble is known to all users, and is usually fixed. An alternative is to intersperse synchronising signals within the structure of the transmission, so that receipt of the beginning of the transmission is not necessary to achieve synchronisation and receivers which lose synchronisation during the transmission can reacquire. For security reasons and ease of implementation the transmitting signal itself can be used to achieve initial acquisition. Tracking is generally accomplished by a feedback loop which adjusts the receiver's time base to track the incoming signal.

Most of these synchronisation methods, especially for low data rate systems, have been performed using digital techniques (5). The advent of analogue SAW devices and CCD technology has led to synchronisation schemes with fast acquisition characteristics. These are mainly used for very high data rate systems (13)-(15).

1.5 Practical Implementation Problems

The code sequences that are used for spectrum spreading must fulfill two criteria; (i) denying any information about future sequence k -tuples to the unintended user, and (ii) permitting practical implementation, including convenient code changes. Sometimes it is desirable for the sequence autocorrelation behaviour to have a high peak-to-sidelobe ratio, for acquisition synchronisation purposes. It is also desirable that the code sequence has a proper k -tuple statistics. Practical and efficient implementation techniques for PN sequences centre around use of shift-registers (3). High speed shift register implementation has been improved over the years from the use, in 1959, of large lumped-constant delay networks to present use of integrated circuits. A special LSI/MSI packages capable of operation at bit rates in excess of more than 300 Mbps has been developed especially for code sequence generation in spread spectrum systems. Increasing the code rate requires a significant improvement in the speed of integrated circuit technology. On the other hand, high speed logic circuits tend toward noise sensitivity and are more susceptible to error. This reason, in addition to the problems of spectrum occupancy, system synchronisation, and propagation constraints tend to limit the code rates used for spectrum spreading, and hence to improve system process gain.

In principle, it is possible for spread spectrum receivers to use matched filter or correlator structures to synchronise to the incoming signal. Sliding correlator (7) and sequential

estimation (16) methods have been used for acquisition which employ techniques to bring the transmitter and receiver code sequences into a range in which digital correlator or matched filter may be used. A time-complexity tradeoff exists. While using a bank of correlator or matched filters provides a means for rapid acquisition, a considerable reduction in complexity, size, and receiver cost can be achieved by using a single correlator or a single matched filter. However, these reductions are paid for by the increased acquisition time needed when performing a serial rather than a parallel operation. One obvious practical implementation problem is therefore the determination of the tradeoff between the number of parallel correlators (or matched filters) used and the cost and time to acquire. It is important to note that this tradeoff may become a major point, recently, as a result of the rapidly advancing VLSI technology.

Practical system considerations such as those encountered when operating at, HF, VHF, or UHF, and technology considerations, such as the role of surface acoustic wave devices and charge-coupled devices in the design of spread spectrum systems are not included in this work.

1.6 Bit-Slice Microprocessors and Spread Spectrum Systems

Microprocessors are one of the most significant products of VLSI technology previously mentioned. It is a monolithic device which can be obtained at low cost and which may be made to perform a wide range of instructions. The microprocessor system is configured such that it may perform most of the digital signal

processing tasks by appropriate choice of a sequence of instructions, 'software', stored in a read-only memory (ROM) space. Under the user control, the microprocessor may access the stored instructions and executes them sequentially. A microprocessor system may be made adaptive by determining that the order of execution of the instruction sequence is dependent on previous and/or present events. Because these devices are fabricated using MOS technologies, the instruction execution time is relatively long. In addition, their word length is limited and instructions are fixed. The inflexibility might prevent their use in applications where high speed or special instructions are essential.

A bit-slice microprocessor is a bipolar device which is designed to achieve high performance, flexible instruction format, and much longer word lengths. It is configured such that its control should be microprogrammed. A set of programmable read-only memory (PROM) or ROM are used to store the program instructions or 'microinstructions' which supervises the central processing unit (CPU) and the other auxiliary logic circuits. The CPU is where data is processed and it consists of one or more bit-slice microprocessors connected in cascade. A program counter may be used to access the stored microinstructions which are executed sequentially or in adaptive order. Usually the microinstruction is a dedicated user design.

This thesis describes the applications of bit-slice microprocessors to synchronisation and other aspects of digital spread spectrum communication systems.

The next chapter describes the different digital correlation techniques to be implemented with the aid of a microprocessor, and the implementation of other discrete-time signal processing techniques which are used in subsequent chapters in this thesis.

Chapter 3 describes the hardware configuration of the bit-slice microprocessor system, based on the 2901 bit-slice devices, that has been used in the subsequent chapters.

Chapter 4 continues the description of the microinstruction design of the system and introduces timing considerations. It describes the microprogram support tools; special assembler, software simulator, and other development and test equipments.

Chapter 5 discusses the analysis and implementation, in both software and hardware, of the functions which are concerned with direct sequence spread spectrum systems.

Chapter 6 describes how the 2901 microprocessor can be applied to perform the signal processing for the spreading, synchronising, and despreading of the transmitter and the receiver.

The ideas and results obtained from previous chapters in this thesis were combined in chapter 7 to discuss the performance of the receiving system in the presence of a channel noise simulator process. Formulas for estimating the synchronisation time have been given and results obtained using the equipment which was previously described are discussed.

1.7 Conclusion

Although the current applications for spread spectrum techniques continue to be primarily for military communications, there is a growing interest, during the last decade, in the use of these techniques for other commercial applications such as mobile radio networks, code division multiple access, and timing and positioning systems.

The problems associated with implementing this technique in data communications systems are considerable because of the cost, complexity, and the constraints on the information. Most of these problems are related to the technology to be used and the applications under question. One of the main tasks, which can be all digital, to be accomplished at the receiving end of a spread spectrum system is the synchronisation of the pseudonoise signal generated locally at the receiver with the pseudonoise signal contained in the received signal. This synchronisation process must be achieved in minimum time which requires high speed digital circuitry.

With the advent of microprocessors a relatively cheap and powerful digital signal processor has now become available. These microprocessors are well suited to communication systems which require adaptability since they are cheaper than analogue processing methods and take up less space.

This thesis describes the applications of these devices to synchronisation process and other digital signal processing requirements which are related to the present communication

systems. It shows that considerable savings in cost and hardware requirements may be made by using a primarily software-based approach to system design.

CHAPTER 2

Digital Correlation Techniques Using Microprocessors

2.1 Introduction

Correlation techniques have been widely used in signal processing systems such as spread spectrum communications, radar, and others. In all these systems correlation must be performed in real-time, requiring the use of electronic circuits that are compatible with the system in question. Electronic systems that perform correlation have been around for years, but they have been bulky and inefficient. The development of VLSI and microprocessors have changed this; now correlation can be performed efficiently with a minimum number of components (17). A digital correlation circuit should be able to achieve the three functions of correlation: time delay, multiplications, and summation, respectively. In binary correlation, on the other hand, the shift register, the exclusive NOR gates, and the summer fulfill the three functions.

A microprocessor has been found to be efficient in implementing digital correlation signal processing, especially for low rate signals. Recent work of Cooley, Tukey (18), (19), Winograd (20), (21), Agarwal, and Burrus (22), (23) has been devoted to minimising the requirement of multiplications in convolution and correlation algorithms to be implemented using microprocessors, because multiplication cannot be performed efficiently by microprocessors.

Many of the correlation signal processing requirements of spread spectrum communications systems may be realised using high speed digital techniques. Spread spectrum bandwidth must be large to obtain significant performance improvement. This means that the sequence rate must be fast and very fast microprocessors will be required when they are used to perform spread spectrum correlations. This is one of the reasons that the bit-slice technology is very attractive in this application.

This chapter introduces the different digital correlation techniques to be implemented with the aid of microprocessors. Digital correlation plays an important role in the analysis, the design, and the implementation of digital signal processing systems concerned with spread spectrum systems and is used in several of the parts described in following chapters. Software implementation of efficient algorithms for the computation of digital correlation is investigated. The possibility of applying the other alternative, binary correlation, using the bit-slice technology is also presented. The theory and hardware construction of a real-time spectral analyser based on the most recent charge coupled devices (CCD) technology is also included.

2.2 Digital Correlation

It is well known that when a received spread spectrum signal $r(t)$ is the transmitted signal $s(t)$ corrupted by additive white Gaussian noise, $n(t)$, the optimal receiver is a correlator receiver which computes correlation according to the equation

$$c(\tau) = 1/T \int_0^T r(t)s(t + \tau) dt \quad (2.1)$$

where $c(\tau)$ represents the crosscorrelation between the received signal and a replica of the transmitted signal. In many spread spectrum communications systems, the signal $s(t)$ is a pseudonoise (PN) sequence.

In general, correlation between two functions is a measure of their similarity, i.e., it is a comparison process. Equation (2.1) is determined by multiplying the received signal $r(t)$, by the transmitted signal shifted in time, $s(t+\tau)$, and then taking the integral of the product. Thus correlation involves time shifting, multiplication, and integration. The correlation of a function $s(t)$ with a time-delayed replica of itself is called autocorrelation.

Digital signal processing requires functions to be represented in discrete form, where the time scale and amplitude are quantised into discrete steps. The PN spread spectrum receiver, when implemented digitally, performs the correlation function as follows:

$$c(nT) = 1/N \sum_{i=0}^{N-1} r(iT)s((i+n)T) \quad n=0,1,\dots,N-1 \quad (2.2)$$

where the original time functions are approximated by sequences of length N . The N selected will depend on the durations of the two functions and of their sampled portions, and on their periodicities (if any). One guide often used in determining the sampling rate ' T_0 ' is the sampling theorem which states that an input signal with a highest frequency component of ' f ' can be

recovered without distortion using a sampling frequency $2f$ (24). A sampling rate (which is also known as Nyquist sampling rate) of $2f$ or greater will therefore minimise the likelihood that analogue information is being lost in the quantising process.

A microprocessor may ~~be~~ perform correlation, operating according to the discrete summation equation (2.2). Successive samples of an input voltage waveforms can be collected using an analogue-to-digital (A/D) converter. These data samples can either be put through some interface (input/output (I/O) ports or perhaps a peripheral interface adapter (PIA)) and sent to the microprocessor, or they can be stored in read access memory RAM directly by using each successive "conversion done" output of the analogue to digital converter (ADC) to initiate a direct memory access (DMA) cycle. After all the desired samples have collected, the data can be processed. For a fixed data record, the memory information is held for N complete recirculations before being replaced by a new record. With a varying input signal, after each recirculation the oldest memory sample is replaced by a new input sample. Since all the data samples are available for subsequent processing, multiplying each sample of the recirculating data with a fast reference signal and summing over N samples provides one point of the correlation function. Further points are obtained on successive recirculation. This method requires $2N$ memory space locations, N multiplications and N additions for each term of the correlation. If all terms of correlation function were desired, N^2 multiplications plus N^2 additions would be required. In a microprocessor system which

does not contain a hardware multiplier or employing a single hardware multiplier (rather than a bank of external multipliers) the multiplication operation can take up to 300 microseconds (u.sec). As a result of adopting this method, the signal bandwidth will be very limited.

2.3 Transform Analysis

Certain transforms possess the cyclic-convolution property (CCP) which may be stated as; the transform of cyclic convolution of two sequences is equal to the product of their transform. Transforms with the discrete Fourier transform (DFT) structure possess the CCP. Such transforms can be applied to the discrete correlation transform pair theorem (25) which stated as,

$$c(n) = \sum_{i=0}^{N-1} r(i).s(n+i) \quad n=0,1,\dots,N-1$$

and

$$C(k) = R^*(k) \times S(k) \quad k=0,1,\dots,N-1 \quad (2.3)$$

are transform pair, where 'x' denotes pointwise multiplication. This implies that a correlation can be calculated by

$$c(n) = T^{-1} (R^*(k) \times S(k)) \quad (2.4)$$

using two transforms, N multiplications, and one inverse transform. While the direct calculation of correlation according to the defining equation (2.2) would require a number of complex multiplications and additions proportional to N^2 , use of such transforms have been able to reduce this number tremendously.

Fast Fourier Transform (FFT) Correlation

Fast Fourier transform (FFT) is an algorithm for efficiently computing the discrete Fourier transform (DFT) of a finite length sequence. The development and the computation aspects of the FFT algorithm have taken a great stride since the Cooley-Tukey algorithm appeared in 1965 (18). The FFT derivation will not be discussed here (24), only technique for using the FFT for high speed correlation computation.

To apply the FFT to the computation of equation (2.2), N may be chosen to fulfill the required transform length, $N=2^v$. If the data sequence length is less than N , zeros are appended to $r(n)$ and $s(n)$ to eliminate the overlap or end effects. According to equation (2.4), we compute the following;

Compute the DFT of $r(n)$ and $s(n)$ using the FFT algorithm:

$$R(k) = \sum_{n=0}^{N-1} r(n) W^{nk} \quad k=0,1, \dots (N-1) \quad (2.5)$$

$$S(k) = \sum_{n=0}^{N-1} s(n) W^{nk} \quad (2.6)$$

Change the sign of the imaginary part of $R(k)$ to obtain $R^*(k)$.

Compute the product;

$$C(k) = R^*(k) \times S(k) \quad (2.7)$$

Compute the inverse transform using the forward transform;

$$c(n) = N^{-1} \sum_{k=0}^{N-1} C^*(k) W^{-nk} \quad (2.8)$$

where $W = e^{-j2\pi/N}$.

From the computation time point of view, the use of FFT correlation technique would require a time proportional to $(3.(N/2)\log N + N)$, complex multiplications, when N is a power of 2. It is generally faster to use this technique to compute digital correlation rather than computing equation (2.2) directly. Exactly how much faster the FFT approach is than the direct method depends on the microprocessor being employed and the extra supported hardware (i.e., single or parallel-processing scheme with either software or hardware multiplier). It should be noted that the efficient computation of correlation using FFT algorithm involves intermediate quantities, i.e., stored or generated sines and cosines, which are irrational numbers, so making exact results without roundoff errors is impossible on a microprocessor.

In 1975, Winograd (20) developed a new algorithm for computing short length DFT's known as the Winograd Fourier transform algorithm (WFTA). This algorithm uses fewer multiplications than the FFT, and about the same number of additions (26).

Correlation using Number Theoretic Transforms

Since 1972, Rader (27), Agarwal and Burrus (22), (28) have developed many transforms with the DFT structure (i.e. FFT and WFTA algorithms can be applied) which can be used for fast and exact calculation of finite digital convolution or correlation, and do not require storage of basis functions (sines and cosines). These transforms are collectively known as number theoretic transforms (NTT's), that are ideally compatible with

microprocessors. In these transforms an integer ' α ' of order N replaces $W = \exp(-j2\pi/N)$ used in the DFT, and both ' α ' and N are defined on finite fields and rings of integers with all the arithmetic operations to be carried out modulo an integer M , e.g. if we have a sequence of length N , $x(n)$ with modulo M we define the NTT of this sequence as:

$$X(k) = \sum_{n=0}^{N-1} x(n) \alpha^{nk} \mod (M) \quad k=0,1,\dots,N-1$$

and by analogy to DFT, the inverse NTT is;

$$x(n) = N^{-1} \sum_{k=0}^{N-1} X(k) \alpha^{-nk} \mod (M) \quad n=0,1,\dots,N-1$$

where the modulus, M , and the sequence length, N , have no common factors and where N is a divisor of $O(M)$ (the number of prime integers in M). α is chosen to be mutually prime to M and to have order N (22), (27), (28). These NTT's are truly digital transforms, taking into account the quantisation in amplitude and the finite precision of digital signals.

Microprocessors are becoming available with fast-multiply instructions, and for these that do not have this facility, fast hardware multiplier chips are available which allowing non-simple moduli and α 's and so many NTT's become practicable for microprocessor implementation (29). The main disadvantage of these transforms is that there is a relation between the sequence length N and the required word length that can require long word lengths for long sequence lengths.

2.3.1 Correlation Using Rectangular Transforms

Agrawal and Cooley (23) have derived a very efficient short term convolution algorithms ($N=2,3,\dots,9$) based on the recent work of Winograd (26), which can be used to generate a very useful tool to compute digital correlation. The new technique which was derived is called the **rectangular transform** technique. Like the FFT method, it significantly reduces the number of multiplications relative to the N^2 multiplies of the direct method. The authors have described a method, by which long length convolutions can be derived using two or more shorter convolutions, known as multidimensional convolutions. As an example, the derivation of a two-factor algorithm for cyclic $N=15$ correlation will be given here, according to this rectangular transformation technique.

Consider, in this example (to avoid any misleading due to symbol variations), that the correlation equation is

$$y_i = \sum_{k=0}^{14} h_{i+k} x_k \quad i=0,1,\dots,14$$

and let each of the vectors H , X contains the sequence elements h_i and x_i , and the vector Y contains the correlation sequence y_i . It should be noted that if the discussion in this section will be carried out on the discrete convolution equation only the h_i indices are needed to be taken in the backward direction to represent the discrete correlation equation.

Let N to be a composite number with mutually prime factor, $N=r_1 \cdot r_2$, where $r_1=5$ and $r_2=3$. By using the Chinese

Remainder Theorem (CRT) (23) to define the one-to-one mapping;

$$i \longrightarrow (i_1, i_2)$$

i.e.,

$$i = r_2 \cdot q_1 \cdot i_1 + r_1 \cdot q_2 \cdot i_2 \text{ mod } (15) \quad (2.9)$$

where q_1 and q_2 are given by

$$r_2 \cdot q_1 = 1 \text{ mod } r_1 \quad q_1 < r_1$$

and

$$r_1 \cdot q_2 = 1 \text{ mod } r_2 \quad q_2 < r_2$$

one would obtain

$$q_1 = 2 \text{ and } q_2 = 2$$

substituting q_1 and q_2 in equation (2.9), we get

$$i = 6i_1 + 10i_2 \text{ mod } (15) \quad (2.10)$$

Table (2.1) illustrates how the index i is mapped to (i_1, i_2) ,

		i_2		
		0	1	2
i_1	0	0	10	5
	1	6	1	11
	2	12	7	2
	3	3	13	8
	4	9	4	14

Table(2.1) Correspondence between one-and two-dimensional indexing in the prime factor algorithm for the case $i_1=5$, $i_2=3$ and $N=15$.

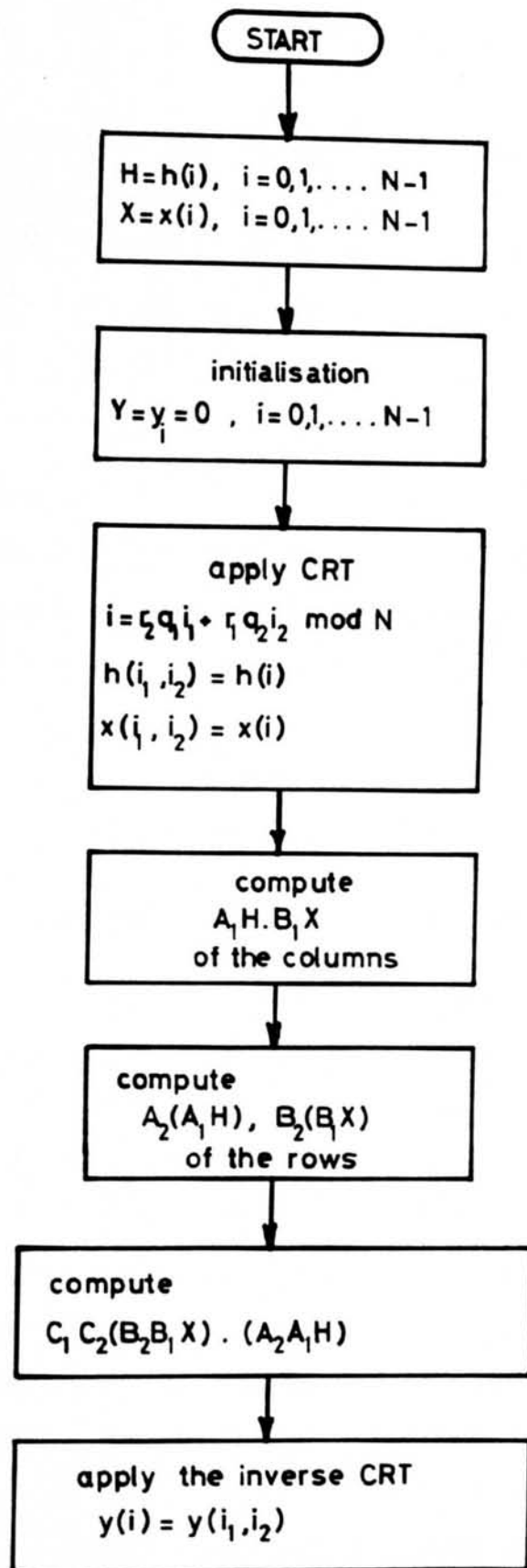
Let y_i , h_i , and x_i , respectively be indexed by the index pair (i_1, i_2) as shown in table (2.1). The two-dimensional algorithm can be represented, in this case, as

$$y_{i_1, i_2} = \sum_{k_2=0}^2 \sum_{k_1=0}^4 h_{i_1+k_1, i_2+k_2} x_{k_1 k_2} \quad (2.11)$$

In vector-matrix notation equation (2.11) may be written as

$$Y = C_5 C_3 (A_3 A_5 H) (B_3 B_5 X) \quad (2.12)$$

The notation $A_3 A_5 H$ means that, one computes the transform A_5 of the columns of H ; that is, each column contains 5-elements which can be computed using an optimal algorithm (23) of length $N=5$, the result is a rectangular array of 10×3 . Similarly A_3 denotes a rectangular transformation of length 3. The final result is then a rectangular transform of 10×4 . The notation $B_3 B_5 X$ means that, one computes the transform B_5 of the columns of X and then the transformation B_3 of the rows of the result. This will give a rectangular array of 10×4 . The element by element multiplication is also a rectangular array of 10×4 . In the same way the operator C_3 reduces the dimensionality, in reverse order, on the array on which it operates; that is, it transforms the 10×4 array to 10×3 , and the operator C_5 transforms the 10×3 array to 5×3 array whose elements are the sequences y_{i_1, i_2} . By applying the inverse CRT, this will yields the one-dimensional correlation of length 15. The above algorithm can be summarised in the flowchart shown in Figure (2.1).



FIGURE(2.1) TWO DIMENSIONAL RECTANGULAR TRANSFORM FLOWCHART.

The rectangular transform approach is applicable for both real and modular arithmetic, depending upon the sort of the transform which is used in each dimension. In most cases the h_i sequences represent a reference signal and remain fixed for many blocks of the x_i sequence, the received signal. Therefore, $A.H$ can be precomputed and used many times.

2.4 Implementations

Two methods for implementing the digital correlation using the direct technique and the rectangular transformation algorithm were investigated in software using the FORTH programming technique (30), (31):

- (i) implementation using Intel-8080 microprocessor system,
- (ii) implementation using TMS9900 microcomputer.

2.4.1 Implementation Using an Intel-8080 microprocessor system

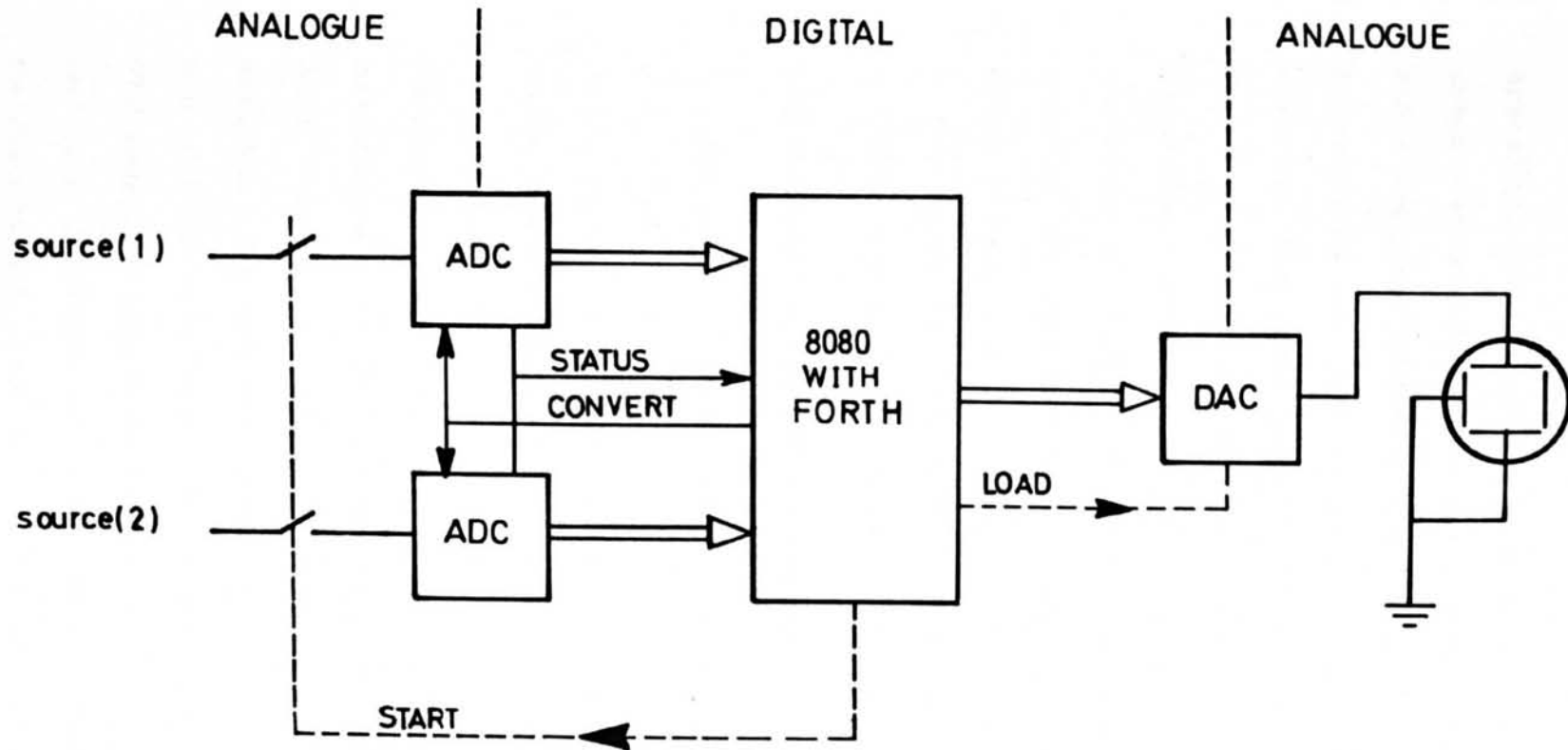
The Intel-8080 microprocessor system (32) which was used is an 8-bit microprocessor with some instructions which operate on 16-bit data. The 8080 system has an instruction cycle about 2 u.sec, it does not contain multiply or divide instructions, and these functions must be performed using software which takes about 250 u.sec. Since it is capable of performing 16-bit arithmetic, the FORTH programming technique was used. This permits the routines to be made very flexible and efficient. In performing arithmetic with reasonably complex expressions it is convenient to use reverse Polish Notation (33) (as in FORTH) which requires a stack to store temporary variables and to pass arguments. Such a stack need not be large, but there should be a reasonable set of instructions for transferring and manipulating

data in conjunction with it, which was available in the Intel-8080 system. FORTH programming is a very efficient technique, since it is an interactive, high level language compact with high speed performance that was suitable to use on the system. However, it was very attractive because the other alternatives, 8080 cross-assembler or any other high level language interpreter, were not readily available at that time.

An 8-bit ADC (Ferranti ZN425E type) was used to convert the analogue signals into sequence as shown in the system block diagram of Figure (2.2). In operation a 'start conversion' signal, a negative going pulse of at least 500 nanoseconds (n.sec) duration, was sent to the ADC from the microprocessor. The conversion takes a finite time and only when it is complete can the digital output be read. The converter produces a 'status' signal, which when high informs the microprocessor that a conversion is in progress, and when the data^{is} valid informs the microprocessor that the converter's output latches certain valid data. ← *contains*
One output port and two input ports were required in this case.

A simple digital-to-analogue converter (DAC) is incorporated in the Ferranti ZN425E chip. This was used to display the output using an oscilloscope. Two output ports were necessary, in this case, one for the converter data (8-bit) and the other for the 'load' pulse.

Real-time correlation programs were written for an Intel-8080 using FORTH programming language, in which two special



FIGURE(2.2) BLOCK DIAGRAM OF THE CORRELATION MEASUREMENTS SYSTEM USING INTEL 8080 MICROPROCESSOR.

operations were developed in order to keep the correlation computation accurate. These are; a 16-bit by 16-bit multiply and divide the result (32-bit) by a 16-bit number, and the second is a routine to store the summation of the multiplication of two sequences. A complete list of the FORTH programs on the 8080 system is shown in the listing of programs in Appendix B. The correlation function scaling was necessary in order to get a resolution of 8-bits. The execution time for an example requiring 100 correlation points using the direct technique was estimated to be about 4.5 seconds. Hence the speed is important in this application, even using the low level feature (assembly) of FORTH language, the system was impractically slow.

2.4.2 Correlation on TMS9900 microcomputer

The TMS9900 microcomputer is an efficient 16-bit machine (34), since it includes the capabilities offered by a full minicomputer. Its powerful instruction set including multiply and divide providing the possibility of computing correlation using fast transformation algorithms, such as the rectangular transforms, in short execution time. In addition, it is highly compatible with the FORTH programming technique, especially since during that time there was no cross-assembler for the 9900 system available. The main block diagram which was used is similar to that of Figure (2.2), except that the 8080 system was replaced by 9900 system. A program was written using the FORTH programming technique to compute 100 equally spaced correlation points using the direct method. Each point required two memory words to have sufficiently accurate results. The approximate speed of the

execution was estimated (excluding the input/output overhead time) by determining the total instruction-execution time. This time was found to be approximately 1 second.

When computing a 15 point correlation by using the rectangular transforms, (note that 16-bit modular operations was used) according to the flow-chart of Figure (2.1), it was found that the execution time is about 15 milliseconds (m.sec). Although there was a great improvements of the execution time when using the TMS9900, the overall requirements cannot be fulfilled by using a single microprocessor system implemented using software only. However, it was envisaged that using binary correlation implemented with the aid of fast bipolar bit-slice technology would fulfill the speed required.

2.5 Binary Correlation

In contrast to general-purpose microprocessors, bit-slice microprocessors (35) can be dedicated to the execution of a special task, for which they may be then prove very efficient. This procedure is especially powerful in combination with microprogramming. The bit-sliced processors are microprogrammed devices that can be realised with two basic types of devices: cascadable bit-slices with the arithmetic/logic unit and the register file on one hand, and a microprogram control memory, which may be arranged to constitute a microprocessor with almost any instruction set, on the other. This gives us the possibility of writing the required algorithms as close as possible to their hardware realisation and to get very high performance but with a 'hard-to-write' microprogram.

An alternative digital correlator using such a bit-slice processor was implemented, which demonstrates the feasibility of using bit-slice microprocessors for digital spread spectrum signal processing. In contrast to the previous methods, the realised processor was tailored for this application, which it therefore fulfills very efficiently. The received signal is normally a binary modulated sequence on which the information was embedded. Therefore, equation (2.2) simply implies a comparison process between the respective bits in the received sequence, $r(i)$, and the shifted stored sequence, $s(i+n)$. The number of agreement bits can be obtained by an exclusive-NOR operation and a Hamming weight function generator, whose outputs are summed. So, the main three operations in the correlation process are replaced by shifter, exclusive-NOR, and summer operations which were implemented at very high speed using the bit-slice approach. Thus for a digital correlator to be effective in this application it must be expandable to accomodate variations in the sequence length. The next chapter will introduce the bit-slice microprocessor chosen for the subsequent work in this thesis.

2.6 Real-time Power Spectral Density

In spread spectrum communication it is desirable to determine the spectral content of signals in real-time. It is very expensive to do this on general-purpose microprocessors, and only special array processors can provide the required digital computing power. However, analogue circuit technology, such as charge-coupled devices, have been widely used in such cases. An evaluation module containing the Reticon R5601 quad chirped

transversal filter (36) was available, which included additional circuitry necessary to compute the power spectrum of an analogue input signal by the Chirp-z transform algorithm (37). Simply, the device and interface system form a discrete-time spectrum analyser, selecting and outputting the magnitude and frequencies of the spectral components of an analogue input signal. The analysis band in the normal situation extends from zero to the Nyquist frequency (one-half the sample frequency). A mirror image also appears extending from the sample frequency (equivalent to dc) down to the Nyquist frequency. The resolution bandwidth in general is approximately (1/512) of the sample frequency. The overall performance is limited to obtaining the power spectral density and to a maximum sample rate of 200 KHz.

2.6.1 Chirp-Z Transform Algorithm

In 1969, Rabiner and Schafer (37) derived an algorithm for evaluating the DFT, which was called the "chirp-z" transform (CZT), in which the bulk of the computation is performed in a chirp transversal filter, and for this reason it is particularly attractive for CCD implementation (38). When implemented digitally, the CZT has no advantages over the conventional FFT algorithm (39).

The CZT algorithm can be derived by starting with the definition of the DFT

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nk/N} \quad , \quad k=0,1, \dots N-1$$

where either or both $x(n)$ and $X(k)$ may be complex.

Using the substitution

$$2nk = n^2 + k^2 - (n-k)^2$$

the following equation results:

$$\begin{aligned} X(k) &= e^{-j2\pi k^2/N} \sum_{n=0}^{N-1} (x(n)e^{-j\pi n^2/N}) e^{j\pi(k-n)^2/N} \\ &= e^{-j\pi k^2/N} \sum_{n=0}^{N-1} g(n)e^{-j\pi(k-n)^2/N} \end{aligned} \quad (2.13)$$

Equation (2.13) represents the CZT. Three operations are required

- (i) Multiply each term, $x(n)$, by the complex factor, $\exp(-j\pi n^2/N)$ to produce a new sequence $g(n)$.
- (ii) Perform a discrete convolution between the sequence $g(n)$ and the sequence $\exp(j\pi n^2/N)$.
- (iii) Multiply the resulting output sequence by the factor $\exp(-j\pi k^2/N)$ for each point of $X(k)$.

The CZT gets its name from the fact that; the sequences $\exp(-j\pi n^2/N)$ and $\exp(j\pi n^2/N)$ can be thought of as complex exponential sequences with linearly increasing frequency. Such signals are called "chirp" (linear FM) signals.

2.6.2 Hardware Implementation

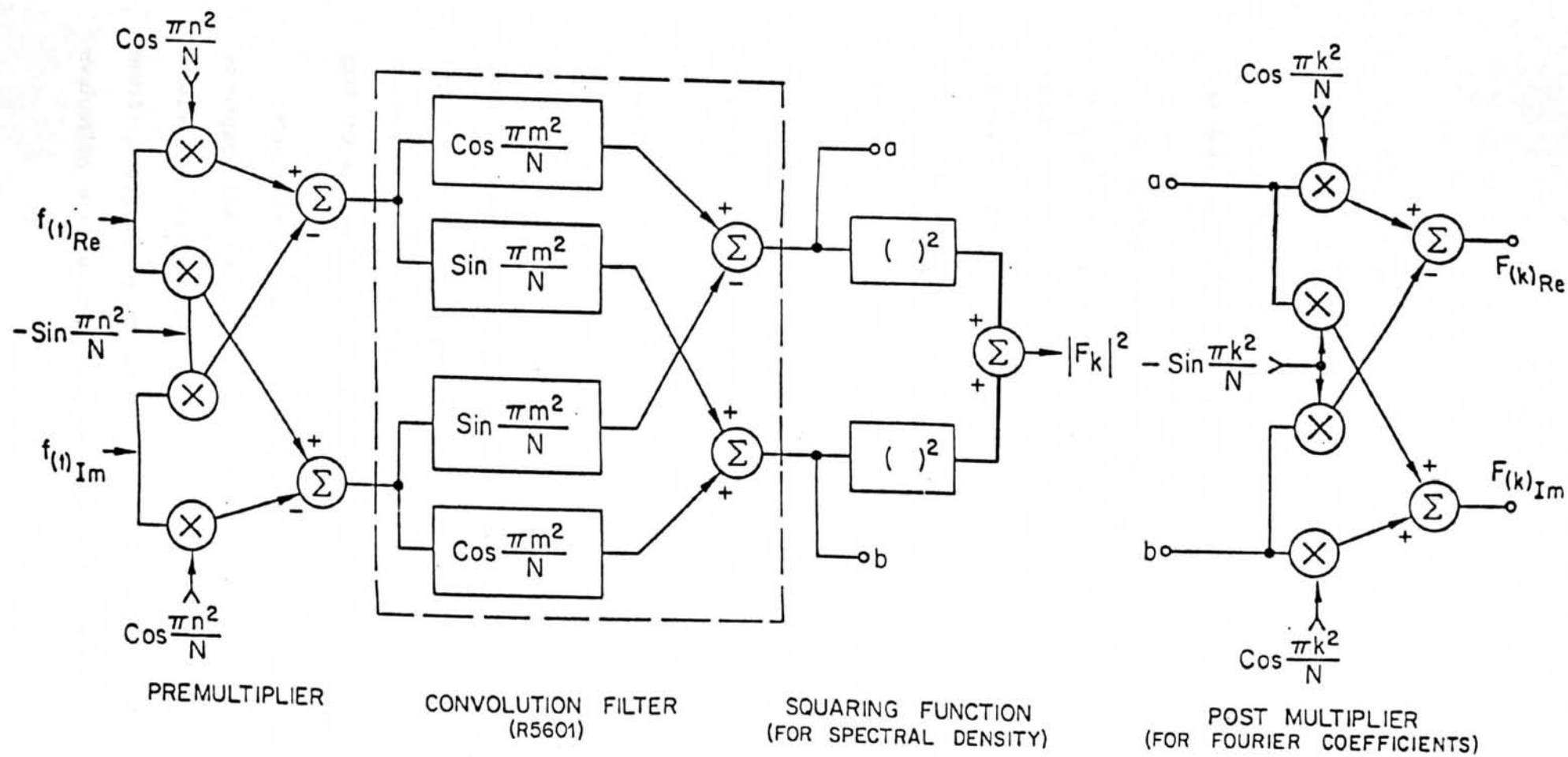
The above discussion shows that the CZT algorithm involves three stages of computation: pre-multiplication, convolution, and

post-multiplication. The block diagram of a complete transform based on the CZT algorithm of equation (2.13) is shown in Figure (2.3). Pre-multiplication is accomplished by the multipliers to the left in Figure (2.3) and post-multiplication by those on the right. The major computing task is the convolution portion; this task is performed by the Reticon R5601 quad chirped transversal filter (36). This device contains two separate 512-stage MOS charge-coupled devices which are used to implement four transversal filters using a split-electrode technique (40). The filter weighting coefficients and internal circuit connections are configured so that the device, in conjunction with additional off-chip components, can implement the CZT algorithm to calculate a 512-point DFT (38), (41).

The evaluation module which contained the R5601 device can be used to compute the power spectrum of an analogue signal. No phase information is obtainable with this module, as the post-multiplier unit is replaced with a hypotenuse function which recovers the spectral amplitude from the component cosine and sine terms. From equation (2.13), the squared spectral amplitude of a sequence $x(n)$ can be expressed as

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\pi n^2/N} e^{j\pi (k-n)^2/N} \quad (2.14)$$

The final phase multiplier term, $e^{-j\pi k^2/N}$, has been deleted because it has unit magnitude and so does not affect the amplitude. The input data is stepped each time a new spectral component is calculated. Equation (2.14) then becomes:



FIGURE(2.3) BLOCK DIAGRAM OF THE CHIRP-Z TRANSFORM ALGORITHM.

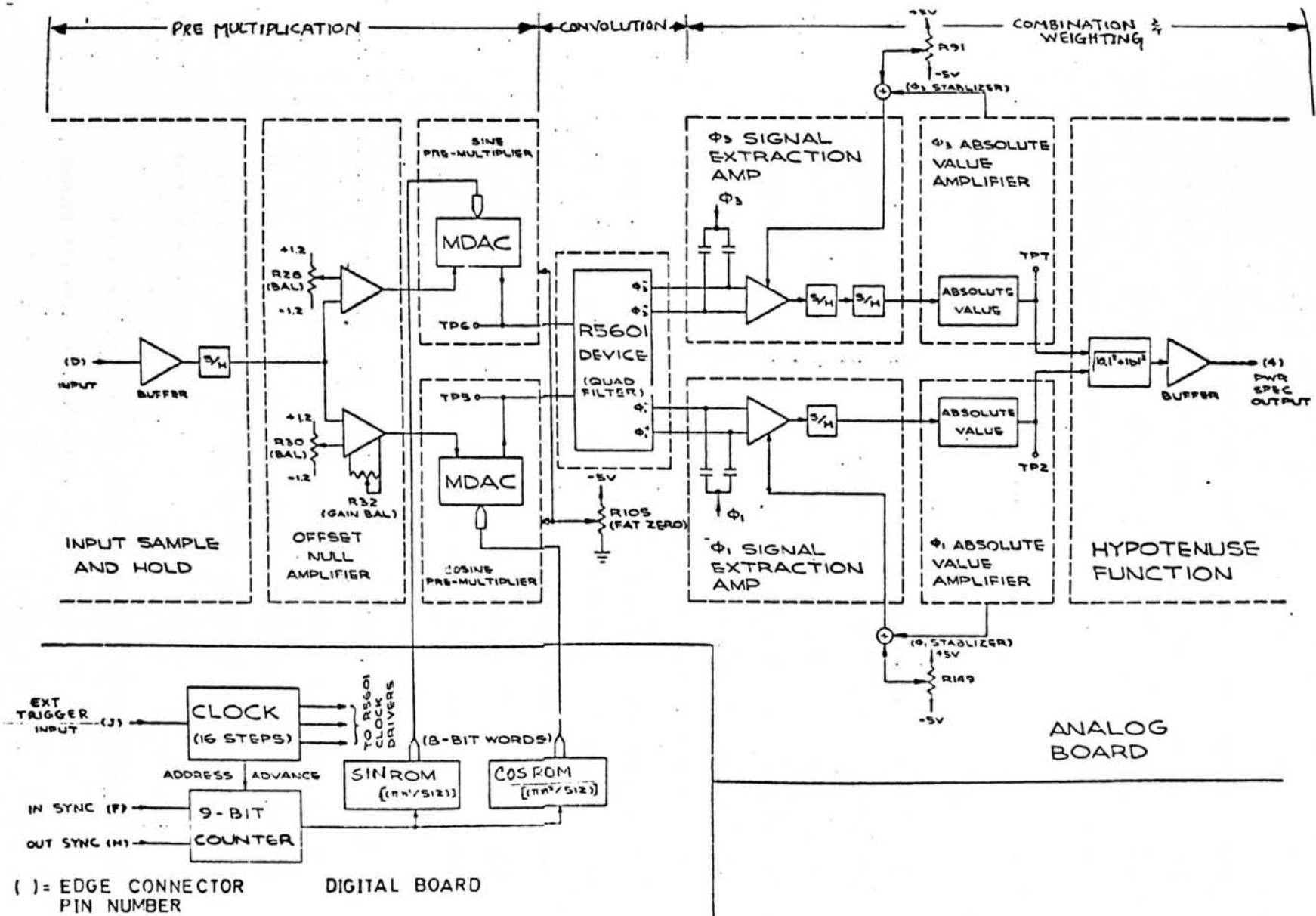
$$X_s(k) = \sum_{n=0}^{N-1} x(n+k) e^{-j\pi n^2/N} e^{-j\pi(k-n)^2/N}$$

The notation $X_s(k)$ indicates a "sliding" CZT.

A further simplification in implementation is possible if the input is purely real, as it is in this case. The imaginary input is always zero so that two of the input multipliers may be deleted and the input circuit simplified.

A block diagram of the evaluation module is shown in Figure (2.4). The analogue (real) input signal is buffered and converted to discrete-time samples by the input sample-and-hold, then split into the direct and quadrature (real and imaginary) channels. The sample values are multiplied by the appropriate chirped waveform using multiplying digital-to-analogue converters. The digital inputs to these converters are derived from two 512-by-8 bit ROMs which contain the sampled chirped sine and cosine waveforms. The sampled analogue products are then used for the input to the R5601 four-channel convolution filter. Outputs from the filter are sampled and held to give time coincidence of all outputs, and then combined on an rms basis to give the spectral density of the input waveform.

Four clock phases are required by the filter device to propagate the discrete signal packets through the CCD channels. These are designated $\Phi_1 - \Phi_4$ and are generated by a multi-phase clock generator circuit incorporated in the evaluation module which may be driven either from a 1.6 MHz



FIGURE(2.4) BLOCK DIAGRAM OF POWER SPECTRUM EVALUATION MODULE.

internal oscillator or from an external trigger source. The sample rate with the on-board oscillator is a nominal 100 KHz, but lower rates are attainable with external triggering. The "address advance" pulse increments a 9-bit counter which addresses the weighting factor ROMs.

2.7 Conclusion

To apply digital techniques directly to the correlation process would seem to require high speed circuitry, in contrast to the rather slow FIS microprocessor systems. Much ongoing research is devoted to minimising the requirement of multiplications in signal processing algorithms, because multiplications cannot be performed efficiently by microprocessors. The applications of efficient algorithms such as FFT, WFTA, and NTTs for digital correlation have been described. The idea of using a general-purpose microprocessor system rather than dedicated processors for digital correlation computation using a fast transform techniques, such as a rectangular transforms, has been implemented and investigated, this will not lead to a very practical bandwidth capability. The use of a dedicated bit-slice microprocessor has been found very efficient in implementing binary correlation and other signal processing applications related to PN spread spectrum system described elsewhere in this thesis.

An investigation into power spectrum using charge-coupled devices has been demonstrated.

CHAPTER 3

Bit-Slice Microprocessor System

3.1 Introduction

In the late 1970's, bipolar LSI devices including the four-bit microprocessor slice became readily available (42), (43), (44). These devices have been used in the design of 4-bit, 8-bit, 16-bit, 32-bit, and even larger CPU's (45). The structures function under the control of a **microprogrammed** memory. The microprogram memory is an N word by M bit memory used to hold the microinstructions, e.g. 1K x 32 bits in the present system. The data output from the microprogram memory are distributed to most parts of the system and these constitute the control signals.

The bit-slice approach requires each central processing element (CPE) chip to contain a 2-bit or 4-bit slice of every register in the CPU of the system. For a CPU constructed of bipolar microprocessor slices, the difference between a CPE and a CPU is that the CPE is the bit-sliced element that is used to form the complete CPU by paralleling two or more CPE's in order to obtain the desired microprocessor word length. A bit-sliced CPE contains a bit group of the working register set or RAM, a very high-speed ALU and status indicators. Multiple buses are used to interconnect the parallel bit-sliced chips and form the microprocessor system. Bipolar microprocessors of this type can be used to form systems with 125 n.sec cycle times. MOS

microprocessor equivalents are slower, with cycle times of the order of 1-2 u.sec. When instruction times are given for a MOS microprocessor, the instruction is a machine level instruction. To compare this with a bit-slice system **macroinstruction** execution times must be used, where a **macroinstruction** is a machine instruction which the microprogram supports. The bit-slice microprocessor developed for this project has an effective macroinstruction time of 330 n.sec or less.

This chapter describes the hardware of the bit-slice microprocessor system that has been used in the following chapters.

3.2 System Organisation

Since the required speed cannot be obtained using MOS microprocessors, a bit-slice approach was chosen for this project.

The architecture of the bit-slice microprocessor system is shown in Figure (3.1). It is an 8-bit microprogrammed processor made up of two 4-bit 2901 bit-slice devices with a microprogram control unit constructed from a PROM and a counter. Other subsystems consist of auxiliary logic control circuits which support the execution of the microinstructions; these are the carry control, the skip select, and the skip control. The system also contains various decoders and external registers which were used for interfacing the system to the external world through an 8-bit data bus and eight control flags. The system operates synchronously under the control of a clock which runs at 3 MHz

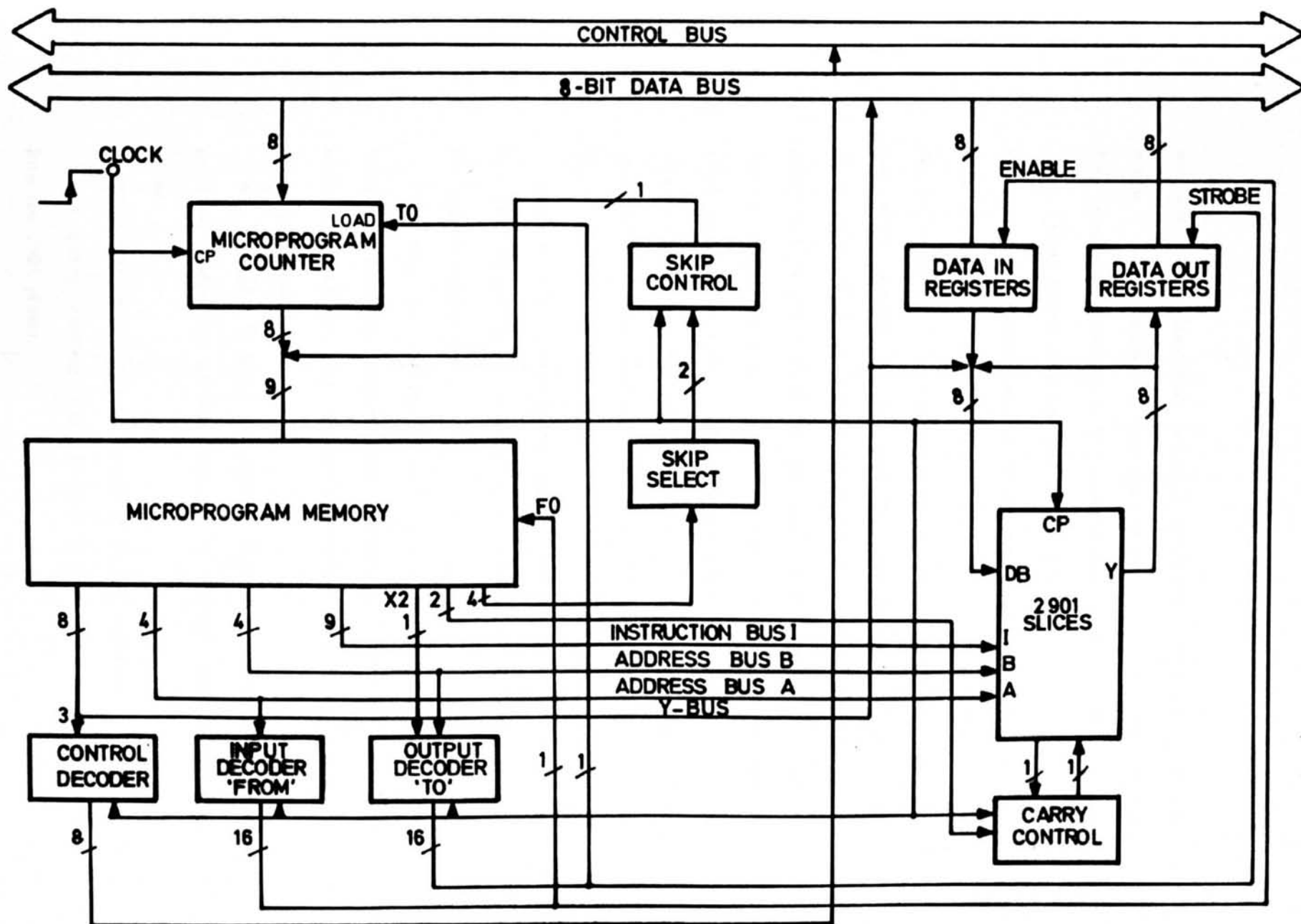


FIGURE (3.1) ARCHITECTURE OF BIT-SLICE MICROPROCESSOR SYSTEM.

and produces a low level for 83.3 n.sec and a high level for 250 n.sec. Before operation the microprogram is loaded into PROM. The size of the PROM is 512 words, with each word being 32 bits long (one microinstruction in length). In operation the microprogram counter outputs an address to the PROM memory, and this address is used to fetch the next microinstruction that is to be executed (a microinstruction will be assumed to execute in one clock cycle). In this case the next microinstruction address is always equal to the current microinstruction address plus 1. After a time delay equal to the read access time of the memory, the memory outputs the control signals to the rest of the system. Each microinstruction contains information blocked out in fields, where each microinstruction field directs or controls one or more specific hardware elements in the system, as shown in Figure (3.1).

The 'Y' field (8-bits) is used to provide constant parameters for the microprogram as well as the address of the destination in the branch instruction.

Two four bit fields, A and B, are used for addressing the internal registers, source and destination. A and B are also used to address the 'From' (data-in) and 'To' (data-out) registers, respectively.

An 'I' field (9-bits) is used to control the source, function, and the destination of any external or internal data in the 2901 slices.

X2 (1-bit) when low, enables one of 16 'To' registers.

The carry control field (2-bits) is used to control the carry into the 2901 slices.

The skip control field (4-bit) is used to control the LSB of the microprogram counter. It is worth mentioning here that the two flags 'T0' and 'F0' have special uses in the system which will be discussed later.

The following sections of this chapter will describe the connection of each IC used in this design.

3.3 2901 ALU/Register slices

The Am2901 bipolar 4-bit microprocessor slice is designed to be used in microprogrammed systems (46), (47). It was first produced by Advanced Micro Devices and is now second-sourced by many other firms. It is the most widely used bit-slice device, because of the flexible structure of the slice's microinstruction. The 9-bit microinstruction code consists of three 3-bit groups that either control or determine the internal arithmetic-logic unit's source operand, ALU function and destination register. This breakdown reduces delays; it permits parallel decoding of different groups of the same microinstruction. The three groups lead to 512 possible microinstructions.

3.3.1 Architecture

The architecture of the 2901 is shown in Figure (3.2) (46). All data paths are 4-bits wide. One key element is the 16-word RAM forming a bank of 16 4-bit registers. It is a 2-port RAM, meaning that two words (registers) can be selected simultaneously. Data in any of the 16 registers of the RAM can be read from the A-port which is controlled by the 4-bit A address

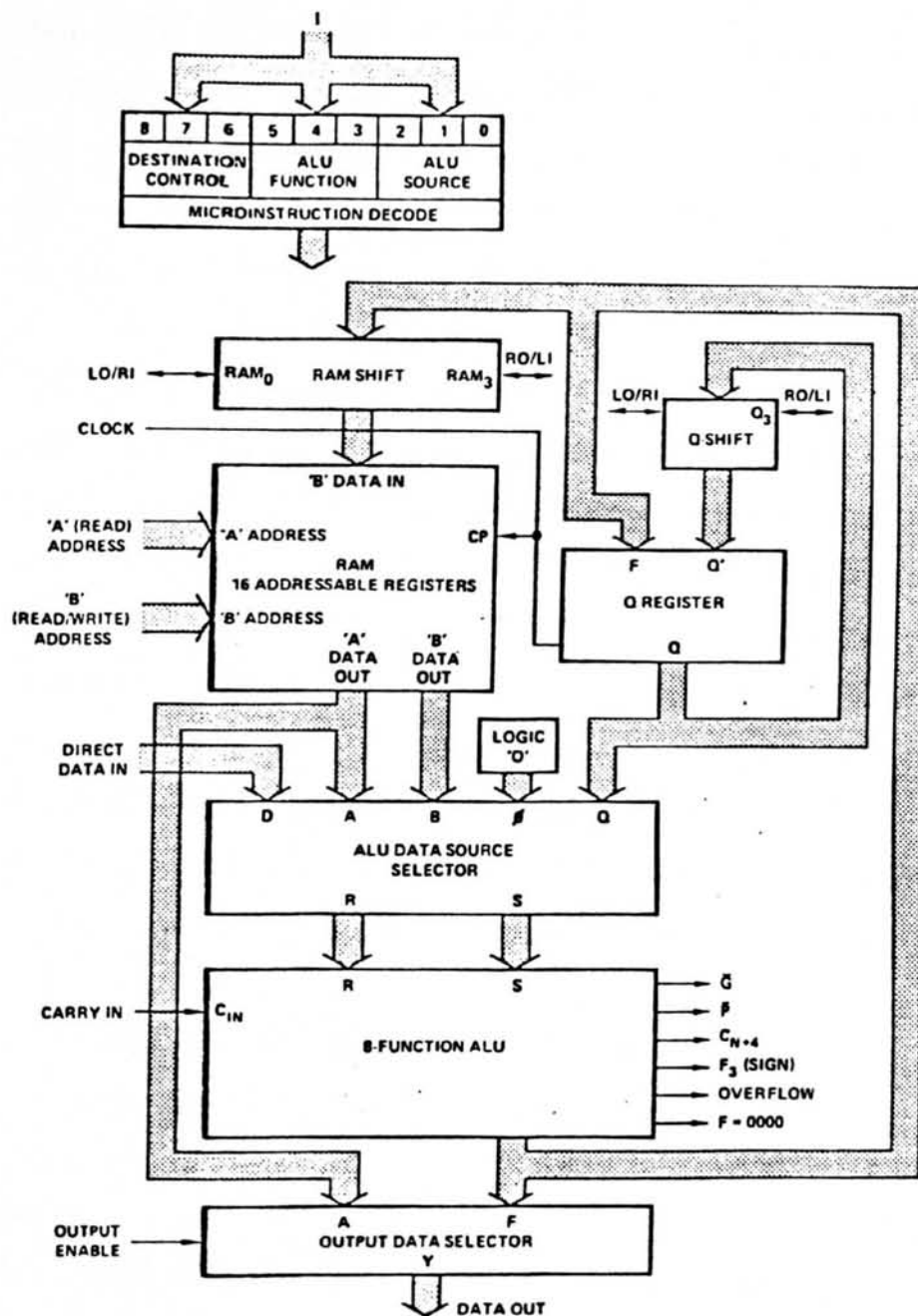


FIGURE (3.2) THE AM 2901 MICROPROCESSOR SLICE.

field input. Likewise, data in any of the 16 registers of the RAM as defined by the B address field input can be simultaneously read from the B-port of the RAM. The A and B busses feed two latches. When the clock input to the slice is HIGH, the selected registers are enabled into the A and B busses and pass through the latches. When the clock input is LOW, the latches hold the RAM data. This eliminates any possible race conditions that could occur while new data is being written into the RAM. The 4-bit high-speed ALU can perform three binary arithmetic and five logic operations. The R port of the ALU is fed from a multiplexer, allowing us to gate the A register, the D bus (an external bus coming into the 2901), or zeros into the R port. Likewise, the S port of the ALU is fed by a multiplexer, allowing us to gate the A register, B register, Q register, or zeros into the S port. These multiplexers and the characteristics of the register array allow us to perform operations such as:

$$\begin{array}{ll} R3 = R2 + R3 + 0/1 & R3 = D + Q + 0/1 \\ R3 = R3 + R3 + 0/1 & R3 = D + 0 + 0/1 \\ R3 = D + R2 + 0/1 & R3 = 0 + R2 + 0/1 \end{array}$$

but not

$$\begin{array}{ll} R4 = R2 + R3 + 0/1 & R3 = Q + Q + 0/1 \end{array}$$

where the meaning of 0/1 is that the carry condition can be added to the operation.

The ALU has three other status outputs. These are F3, F=0, and the overflow (OVR). The F3 output is the sign bit. F=0 output is used for zero detect, F=0 is HIGH when all outputs are

LOW. The overflow (OVR) output is used to flag arithmetic operations that exceed the available two's complement number range. The chip also contains another register, the Q register. It can be used for 8-bit shift up or down operations.

The output of the ALU can be gated to several destinations. A 3-state output bus (Y) can be fed with the ALU output (the F bus) or with the value of the register selected as the A register. The ALU output can also be gated into the register array (the register currently selected as the B register), passing through a shifter as well as being gated into the Q register, passing first through another shifter.

The nine I inputs control the source operands, the ALU function, the shifters, and the routing of data. The microinstruction inputs used to select the ALU source operands are I_0 , I_1 , and I_2 . The I_3 , I_4 , and I_5 microinstruction inputs are used to specify the function of the ALU. The remaining three microinstruction inputs, I_6 , I_7 and I_8 control the two shifters, the Q-register multiplexer, and the Y-bus multiplexer.

The clock input to the 2901 controls the registers array, the Q register, and the A and B latches to the ALU. Data is clocked into the Q register on the LOW-to-HIGH transition of the clock. When the clock input is HIGH, the latches are open and pass the values of the registers selected as the A and B registers. When the clock input is LOW, the latches close and retain the last data entered. New data can be fed into the B

register when the clock input is LOW. Figure (3.3) is a simplified view of the timing of the 2901, the clock timing of the system will be described in the following sections. Notice that the control inputs must be stabilised at their required states at the beginning of the cycle. These times are called set-up times; these are expressed relative to the transitions of the clock input. As an example, the I signals from the current microinstruction must be present at the 2901's pins at least 80 n.sec before the LOW-to-HIGH transition of the clock pulse. Another timing consideration is propagation delays, the time from when an input signal is established to when a particular output is stable (46).

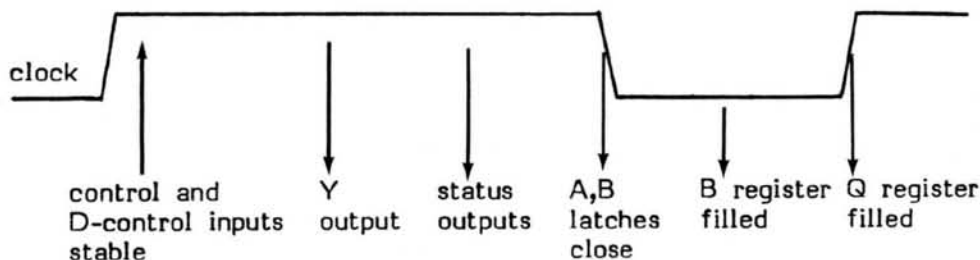


FIGURE (3.3) SIMPLIFIED VIEW OF 2901 TIMING.

3.3.2 2901-Slices Interconnection

Two 2901's were connected to form a CPU with a data-path width of eight bits. The 16 registers and the Q register are 8-bits wide and reside in the 2901's, a half in each 2901 as shown in Figure (3.4). An 8-bit data-in bus feeds both 2901's in parallel, and the 2901's feed an 8-bit data-out bus. Figure (3.4) also shows the connection of the control signals and the

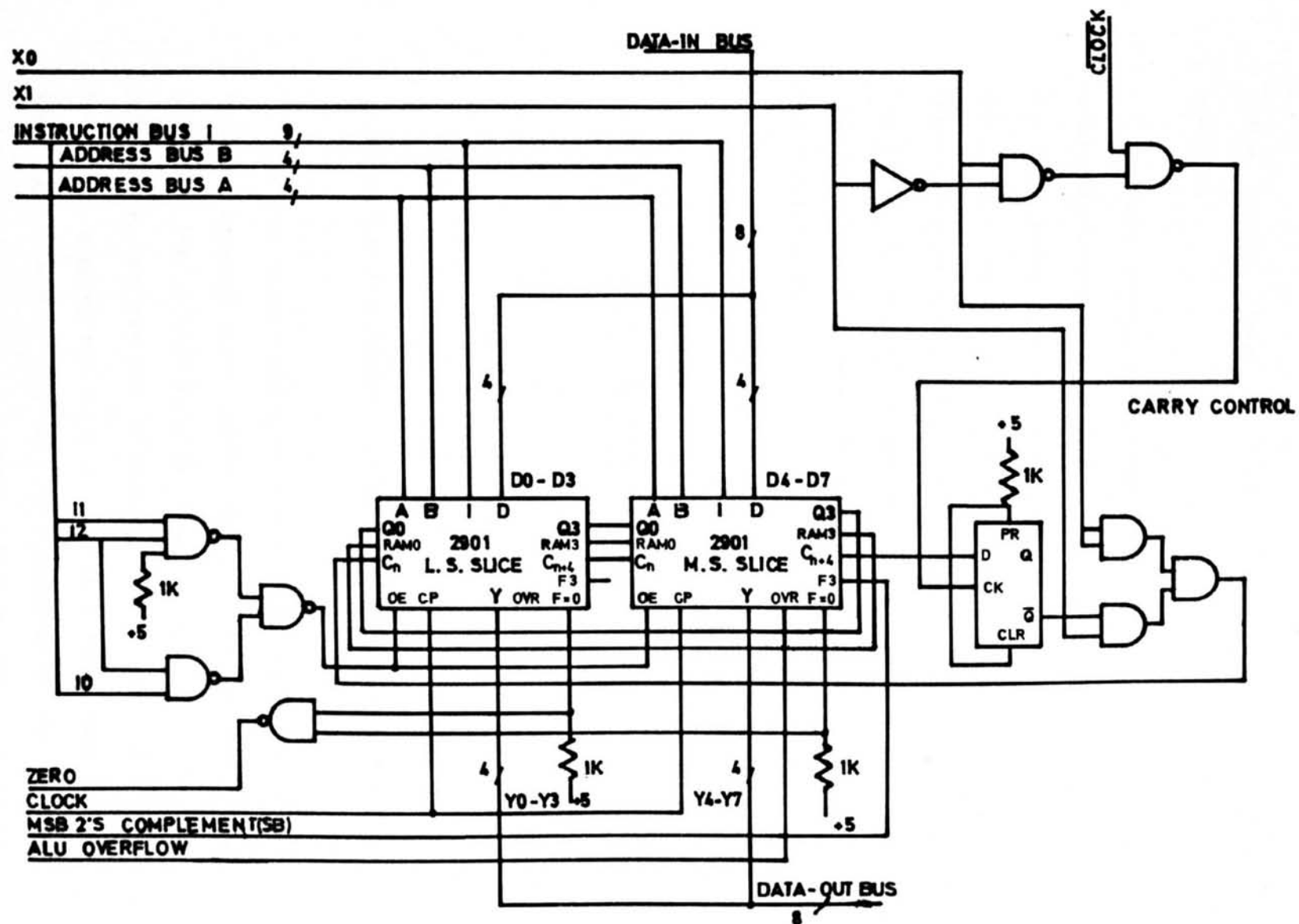


FIGURE (3.4) TWO 2901'S USE TO CONSTRUCT 8-BIT CPU WITH CARRY CONTROL.

status outputs. Most of the control signals feed the 2901's in parallel.

It was mentioned in the previous section that the microinstruction inputs, I_0 , I_1 , and I_2 are used to select the ALU source. One of these source operands is the direct data input (D). To select D the data output (Y) must be in the high-impedance state. This can be done by using the group inputs, I_0 , I_1 , and I_2 to control the output enable (OE) as shown in Figure (3.5), when OE is HIGH, the Y outputs are in the high-impedance state.

mnemonic	micro code			octal code	ALU source operand		OE
	I_2	I_1	I_0		R	S	
DA	H	L	H	5	D	A	H
DQ	H	H	L	6	D	Q	H
DZ	H	H	H	7	D	O	H

FIGURE (3.5) ALU DIRECT INPUTS (D) SOURCE SELECT CONTROL.

On the least-significant slice, the carry-in is an input from an external carry control source. Two bits X0 and X1 determine the carry-in state as shown in Figure (3.6). On the other slice, the carry-in is connected to the carry-out of the first slice, enabling the ALUs to work as a single, ripple carry, 8-bit ALU. Notice also the interconnection of the shifters, enabling the Q shifter and RAM shifter to act as two 8-bit

shifters. Most of the status output are taken only from the most-significant slice. The F=0 output is an open-collector output, meaning that it can be wire-AND'ed, with a pull-up resistor, between slices to indicate whether the output from both ALUs is zero. The look-ahead carry pins on both slices were not used, since the look-ahead carry logic was not used in this design.

X1	X0	carry-in	
0	0	1	carry set
0	1	0	carry hold
1	0	C_{n+4}	carry propagate
1	1	0	carry clear

FIGURE (3.6) CARRY CONTROL LOGIC.

From Figure (3.4) we can analyse the minimum microcycle time for this system as follows:

The guaranteed, or worst-case, propagation times for the Am2901B slice are (43), (42);

From inputs A, B to output Y	60 n.sec
From inputs A, B to last status output	70 n.sec
From inputs A, B to C_{n+4}	59 n.sec
From input C_n to last status output	37 n.sec
From input C_n to output Y	30 n.sec

The propagation delay due to the ripple carry between the slices (i.e. the carry-in to the most-significant slice is not stable until $t + 59$ n.sec) means that the output of this slice will not

stabilise until $t + 59 + 37$ n.sec. By adding the propagation and set-up times of the external carry control (60 n.sec) this system could not operate faster than one microcycle per 160 n.sec.

3.4 Microprogram Control

The microprogram control unit is the part of the system that controls the other subsystems, synchronises the internal and external events and fetches and decodes the microprogram residing in the microprogram memory. A microprogram control unit consists of the microprogram memory and the structure required to determine the address of the next microinstruction; in our case this structure is the microprogram counter. The logic diagram of the microprogram control together with the skip control and the skip select is shown in Figure (3.7).

Unlike the main memory in MOS microprocessor systems, the microprogram memory is referred to once each microcycle during the execution of a microinstruction. Therefore, to gain the necessary speed, the microprogram memory is always implemented using bipolar memory devices. This memory contains sequences of microinstructions, 32 bits wide, which apply the proper control signals to the 2901's and the other subsystems, to execute the desired operation. The address lines of the microprogram memory are driven from the microprogram counter. This counter has facilities for storing an address, incrementing an address, and jumping to any address. The microprogram counter is controlled by bits from the microprogram memory.

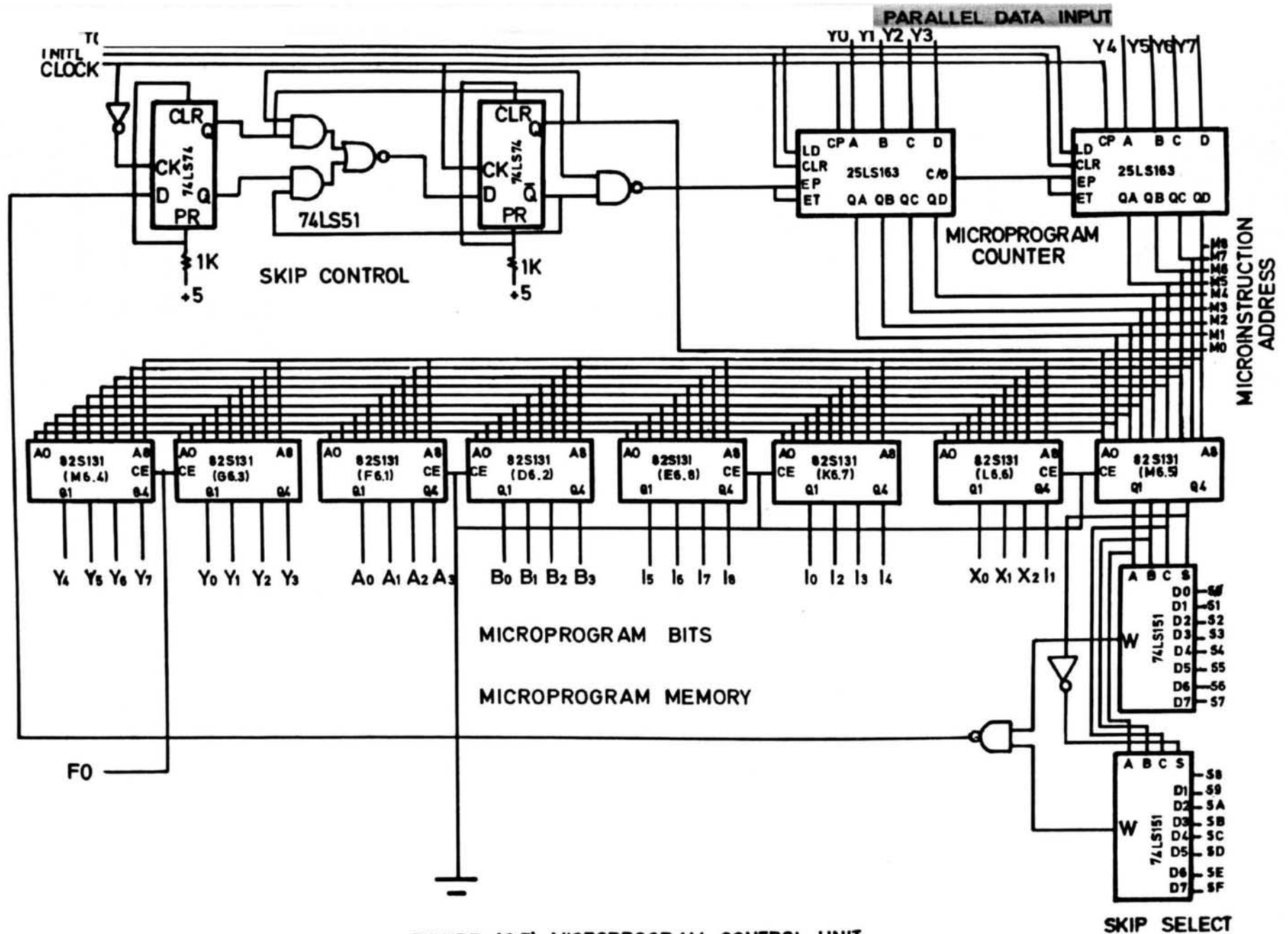


FIGURE (3.7) MICROPROGRAM CONTROL UNIT.

3.4.1 Microprogram Memory

The microprogram memory was implemented in PROMs. A 512 by 32 memory was constructed using the 82S131 device (tristate) (48). The 82S131 is a bipolar PROM, organized as 512 words by 4 bits per word, with nine address lines and an enable line (65 n.sec access time). Eight chips were placed in parallel with all address lines common (Figure (3.7)). The address lines are loaded with 8 loads, under the maximum load limit (50 loads) that could be driven by the counter, therefore no buffer drivers were required. These are driven by the microprogram counter (9-bit). It was mentioned before that the PROM outputs (the microinstructions) are the microprogram bits required to control the rest of the system, these are stable before the next clock pulse. Figure (3.7) shows a typical construction of the output control bits.

The PROM chips are always enabled (active LOW) except for the two chips that are used to store the fixed constants field, the 'Y' field, these are enabled by the strobe 'F0'. In this case the 'Y' field outputs are used as an 8-bit external register to store fixed parameters and the destination address of a branch microinstruction within the microprogram.

Programs were developed to take the microprogram to be placed in the microprogram memory and slice it up among the 8-PROMs. The microprogram support tools will be described in the next chapter.

3.4.2 Microprogram Counter

The address information to the microprogram control is derived from the data bus. The microprogram counter stores the 9-bit address of the current microinstruction to be fetched from the PROM. It consists of two parts, counter, and skip control logic. The counter stores the most significant 8-bits of the address, this consists of two 25LS163 types connected in cascade (49), and it increments on the positive transition of the clock pulse unless the load or clear lines are activated. Two D-type flip-flops were used for constructing the skip control logic, this generates the least significant bit (LSB) of the address that indicates if a skip is required or not. On the negative transition of the clock a selected skip state is strobed into the flip-flop and if the output is LOW the LSB of the microprogram counter is held and count enable to the counter is activated (HIGH), so that on the positive transition of the clock the microprogram counter contents are increased by two instead of being incremented. If a branch microinstruction is taking place then the skip control is used to determine if the LSB of the new microinstruction address is odd or even (1 or 0), while an active LOW strobe 'T0' can be used for loading the counter by 8-bit data on the data bus. In fact, that is the main use for the strobe 'T0' in the system, and it should not be used elsewhere. A LOW level (INITL) at the clear inputs sets the microprogram counter outputs LOW after the next positive clock transition. This facility, reset the microinstruction address to zero on startup, is very efficient in writing a microprogram for the system. Usually the microprogram, to be loaded on PROM, starts with a

microinstruction, which by loading the microprogram counter 'T0' and skipping to give a 0 LSB, and not skipping to give a 1 LSB allows a branch to any location in the PROM.

3.5 Condition code select logic

The skip select logic was added to the system to allow a microinstruction to test conditions generated within its own microcycle. This consists of two 74LS151 types (50) which, under control of 4 microprogram bits, route one of sixteen lines from the various flags to the D input of a flip-flop, for use in determining the next microinstruction address ("skip on result of condition"). The microprogram counter skips one microinstruction if the state of the flag specified is 'TRUE' (HIGH). For microprogram simplicity, the flags were designated S0 to SF, and these assignments will correspond to predefined flags within the system. Typical flag assignments that will be used in the following chapters are shown in figure (3.8).

skip field (Hex)	flag assignments	function
0	S0	never skip (connected to +5 volt through 1k resistance)
1	S1	always skip (connected to 0-volt)
6	S6	test flag
7	S7	test flag
8	S8	output register empty (buffer (FIFO) output)
B	SB	the most significant ALU (F3) output bit

D	SD	input register full (buffer (FIFO) output)
F	SF	the result of an ALU operation is zero (F=0)

FIGURE (3.8) SKIP FLAG ASSIGNMENTS.

The other flag assignments could be used for interfacing the system with different computer systems and connecting it to test equipment.

3.6 I/O External Registers Handling

The I/O subsystem provides the communication between the processor and the outside world. There are three types of I/O used according to the method of controlling the data transfer (50), (51), (52):

- (i) microprogram controlled I/O
- (ii) interrupt controlled I/O
- (iii) direct-memory-access I/O

Interrupt controlled and direct-memory-access I/Os require a complete hardware interface circuit; fortunately the system can provide the suitable data paths and the control signals which are needed for this interfacing. I/O paths can either be bidirectional, in which case the external data will be sent and received via the same lines, or the input and output lines can be separate. Microprogram controlled I/O with separate input and output lines was used in this project.

Two sets of I/O registers were used to interface the external I/O data to the data bus of the system under the control of the microprogram memory through input and output decoders as shown in Figure (3.9). By using 4-to-16 line decoders, any one of the sixteen I/O read or the sixteen I/O write external registers can be selected. Because only one output from the decoders can be activated at a time, in a given microcycle no more than one I/O register in the group can be used. Two four bit fields, A and B, are decoded as input (From) and output (To) decoders respectively. If the microinstruction indicates that external register contents are required, then the 'A' (From) field is decoded, to gate data from a register external to the 2901 slices onto the data bus, in which case the 'B' field is used to select the destination register in the 2901 internal RAM. If the microinstruction indicates an internal source of data, then the 'B' (To) field is decoded and routes either the contents of the 'Y' field or the output from the 2901 slices to the data bus, to be strobed into the external register selected by the B field. In the later case, if the 2901 slices output was chosen to be the source of the data, the 'A' field is used to select the register in the 2901 internal register array. Also it is possible to store data in the 2901 internal RAM and an external register simultaneously, in which case the 'B' field is used to address these registers.

3.6.1 The "To" Decoder

The 'To' decoder was used to provide the system with sixteen data-out registers, one of these registers was assigned as the

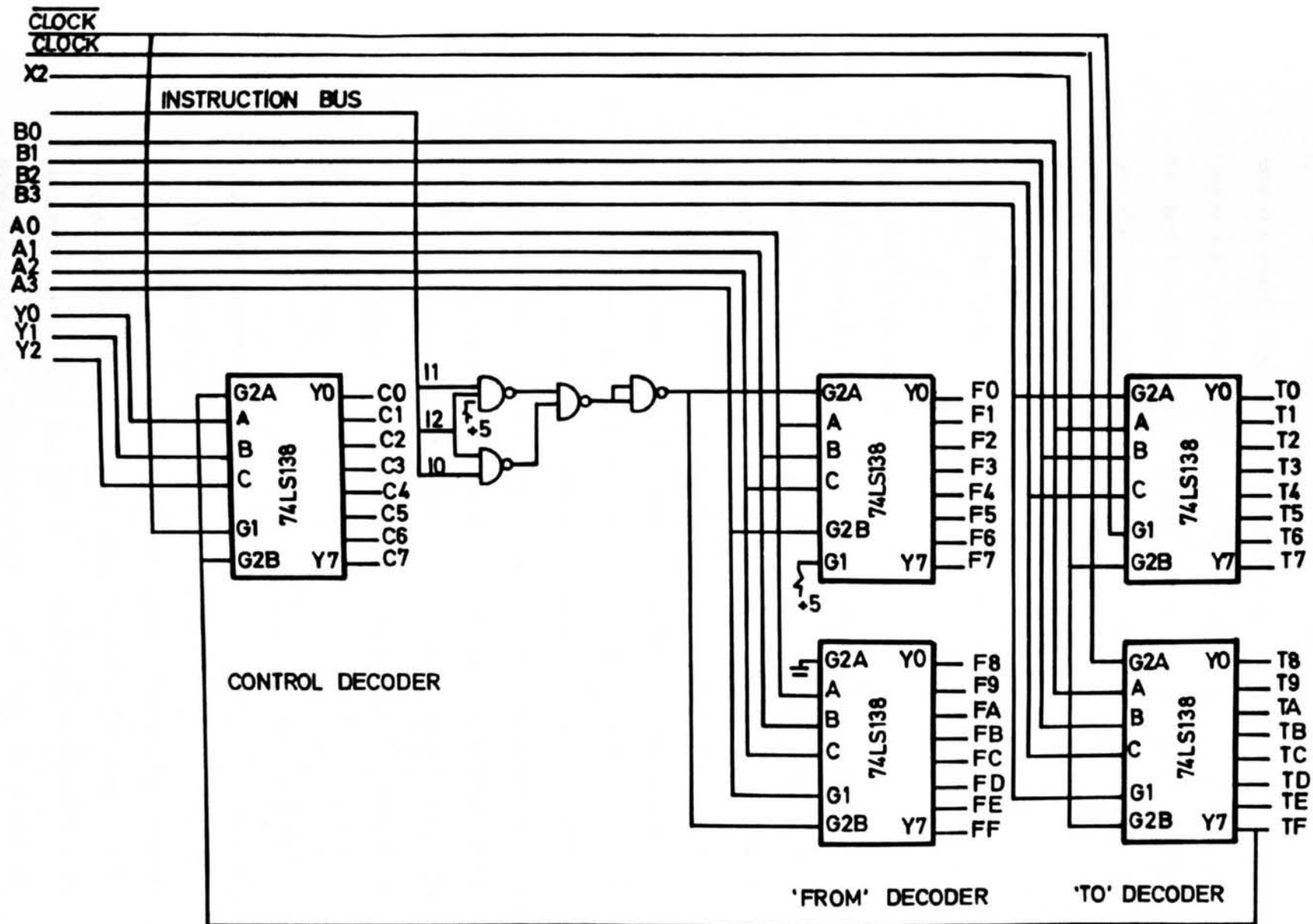


FIGURE (3.9) I/O AND CONTROL DECODERS.

microprogram counter (T0). This decoder consists of two 74LS138 (three-line to eight-line) decoders, enabled by the 'X2' control bit and the clock of the system, and use the output 'B' field, of the PROM to generate the 'To' flags. These flags determine which of the data-out registers in the system is to receive the data on the bus and to generate a strobe for that register to store the data in the second half of the microcycle.

3.6.2 The "From" Decoder

A sixteen-line decoder was implemented using two 74LS138 decoders to provide the system with a sixteen port external source of data. The 'From' decoder decodes one of the sixteen lines, dependent on the conditions of the four binary select inputs and the 'A' field, and is enabled by the I_0 , I_1 and I_2 microinstruction code (the source operand of the 2901) output from the PROM. The 'From' decoder outputs are used to select data-in registers within the system and cause them to output their data onto the common 8-bit data bus while the Y outputs of the 2901 slices are OFF. One of these registers is the 8-bit constant field output of the PROM which is enabled by the 'F0' line.

3.6.3 The Control Decoder

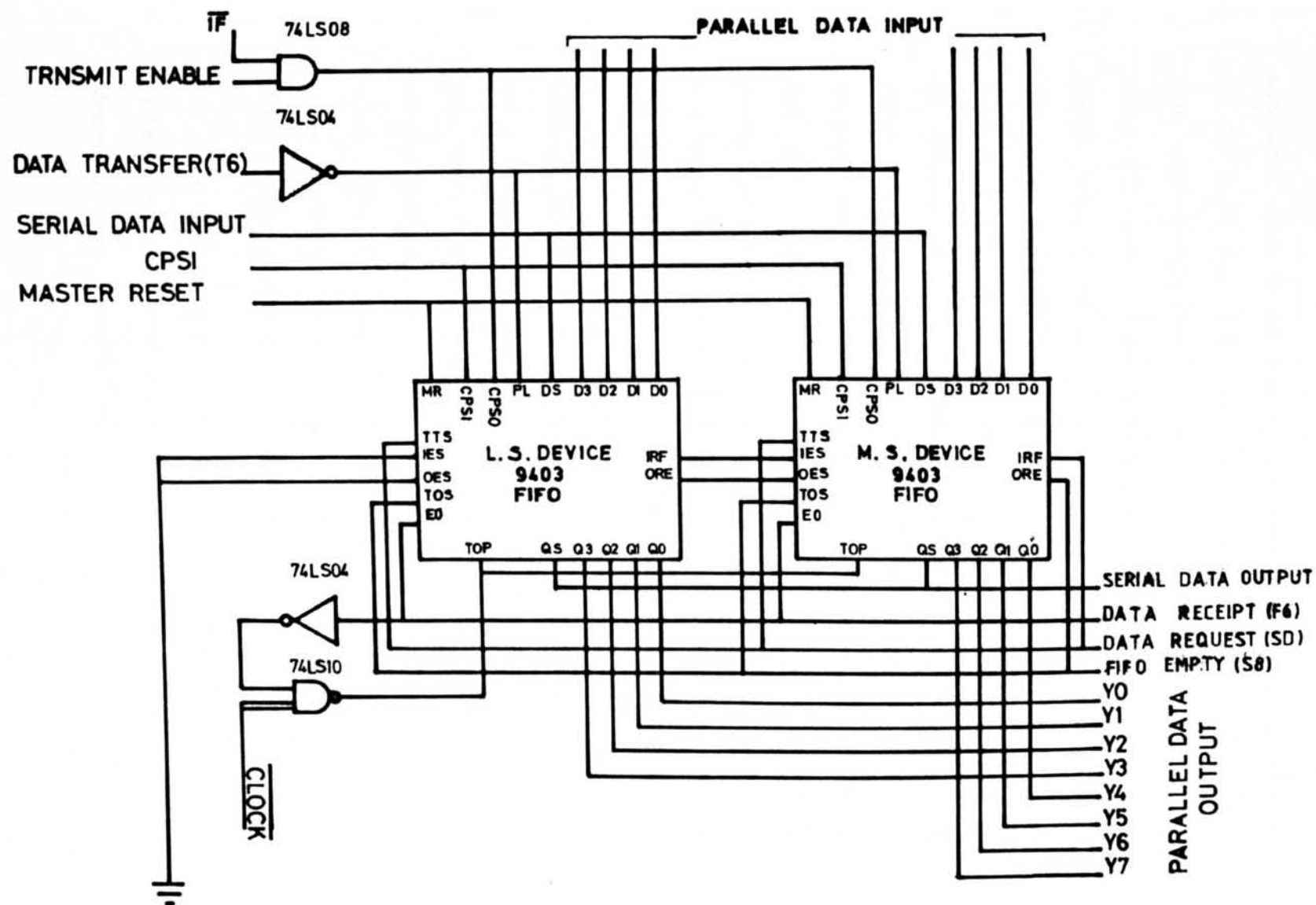
To generate the control signals (that are seldom all needed within the same microcycle) would require a greater microinstruction length than is really needed. By using a three to eight line decoder, any one of an eight control signals can be generated with only three microprogram bits. The advantages and

disadvantages of this technique will be described in the next chapter. The control decoder is a 74LS138 type, enabled by the clock of the system and an output from the 'To' decoder (which in this case is 'TF') and uses the Y0, Y1, and Y2 data outputs of the 'Y' field to select one of the 'C' lines (C0 to C7) to be active LOW in the second half of the microcycle. These lines are used to determine which flag within the other subsystem is to be set or cleared.

3.7 Input/Output Buffer Memory

The input/output buffer memory provides the means for the microprocessor system to interact with the external data medium as characterised by a communication link. First-in, first-out (FIFO) register stacks were used, these allow data transfers that are continuous and do not require the processor to wait during the communication operation. Transferring data from the processor to the link is normally executed as a single microinstruction that loads data, either from an external memory or the 2901 internal RAM registers, into the FIFOs and then issues the necessary flags to initiate the transmitting operation. The subsystem logic circuitry provides the autonomous timing and sequencing signals necessary to perform the shifting operation associated with sending or receiving the serial data. Data may be accepted in serial or parallel at one data rate and extracted at another rate.

The interconnections necessary to form a 16-word by 8-bit FIFO, using the Fairchild's 9403 (16 words by four bits) type, are shown in Figure (3.10). In operation, on the input side,



FIGURE(3.10) 16 WORD BY 8-BIT BUFFER MEMORY SYSTEM.

successive words (8-bits) can be loaded into the FIFO by a LOW on T6 for each one. This can be continued for up to 16 words, if none are removed from the output during the process of loading these 16 words. More generally, the FIFO can continue to be loaded until it no longer raises its SD (data request) line. At this point, the FIFO has accepted the last word but is indicating that it is full and cannot accept further data. Serial data can be entered on each HIGH-to-LOW transition of the CPSI clock input, once loaded into the FIFO, the successive data bits "fall through" the FIFO structure and line up in order at the output. The clock required for the output from the FIFO is completely independent of that on the input. Data words can be extracted by a LOW on F6 line, this can be continued with successive words in the FIFO until S8 (FIFO empty) no longer rises, indicating that the FIFO is empty. Data is serially shifted out on the HIGH-to-LOW transition of CPSO. An important characteristic time of a FIFO, for our purposes, is the "fall through time". This is the time it takes for an input-data word to appear at the output of the initially empty FIFO. This time, in our case, is 450 n.sec (53); this means that it is not possible to extract the same input-data word in two successive microinstructions without using an intermediate microinstruction to test the line S8.

The timing sources for the shifting operation associated with the serial entering or extracting of the data will be described in the following chapters.

3.8 System Clock

The main source of the timing signals is the system clock. The clock's output frequency is controlled by a stable crystal oscillator, MC300 (12 MHz) type. A single phase clock was used in the microprocessor system. The oscillator output drives a binary counter, 25LS169 type, with outputs logically combined to form a set of repetitive signals. Figure (3.11) shows the block diagram of the clock circuit and the clock pulses. The clock pulse width, called the microcycle, is determined from:

$$c_p = t_1 + t_2 + t_3 + t_4 \quad (3.1)$$

where

t_1 : counter clock to output time (n.sec)

t_2 : PROM read access time (n.sec)

t_3 : 2901 ALU execution time (n.sec)

t_4 : other propagation delay in the system (n.sec)

For the 2901 system a microcycle is measured from one rising edge of the clock to the next (42). All input signals to the 2901 slices from the system data bus are captured on the rising edge of the clock (the set-up time prior to the clock LOW-to-HIGH transition is about 70 n.sec for the 2901B). A timing diagram is given in Figure (3.12) showing a series of sequential microprogram steps. During each microcycle, one microinstruction is fetched and executed. At each rising edge of the clock, the microprogram counter increments and settles, and the counter outputs an address to the PROM, whose access time is greater than the counter settling time. As soon as the outputs are stable at

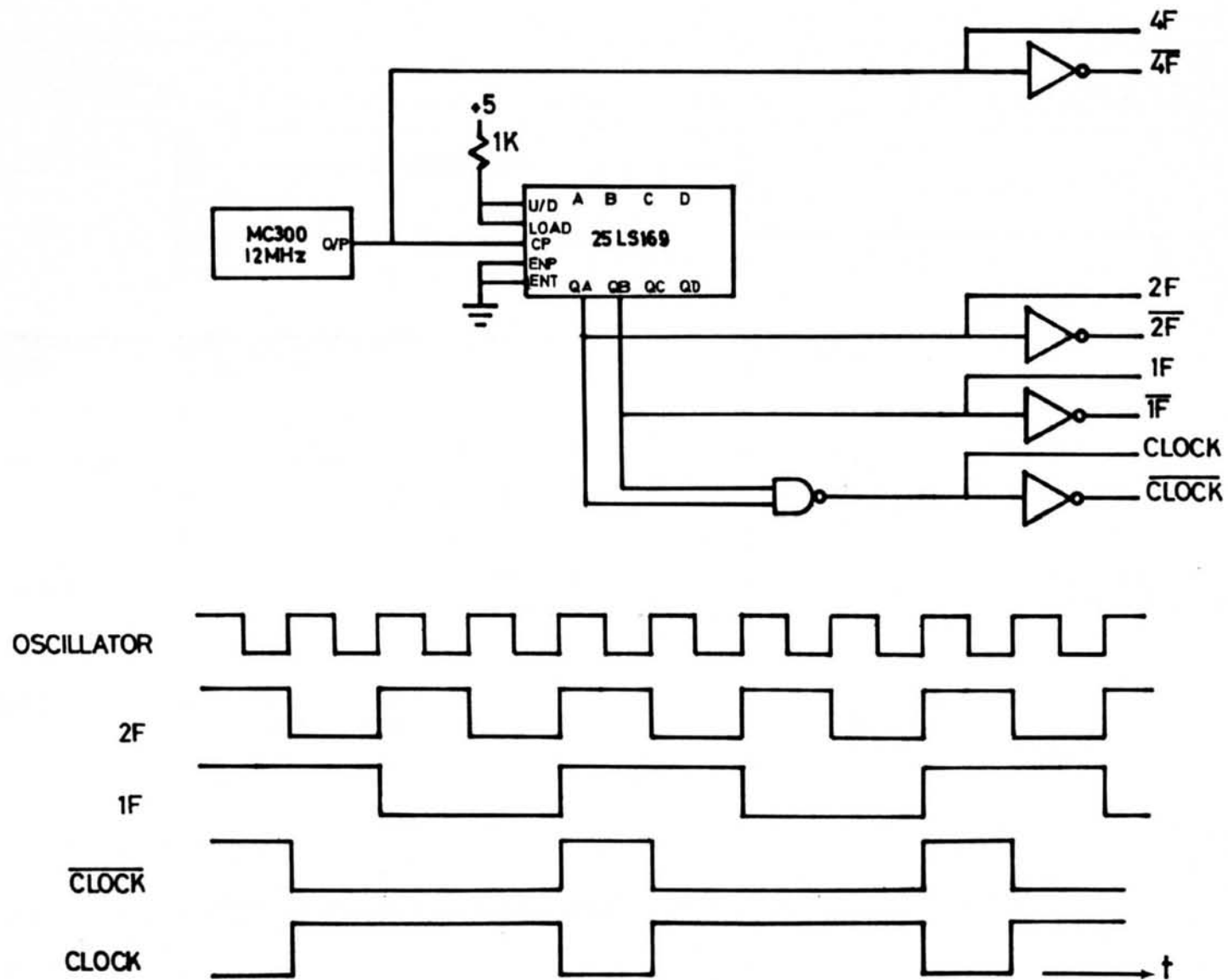
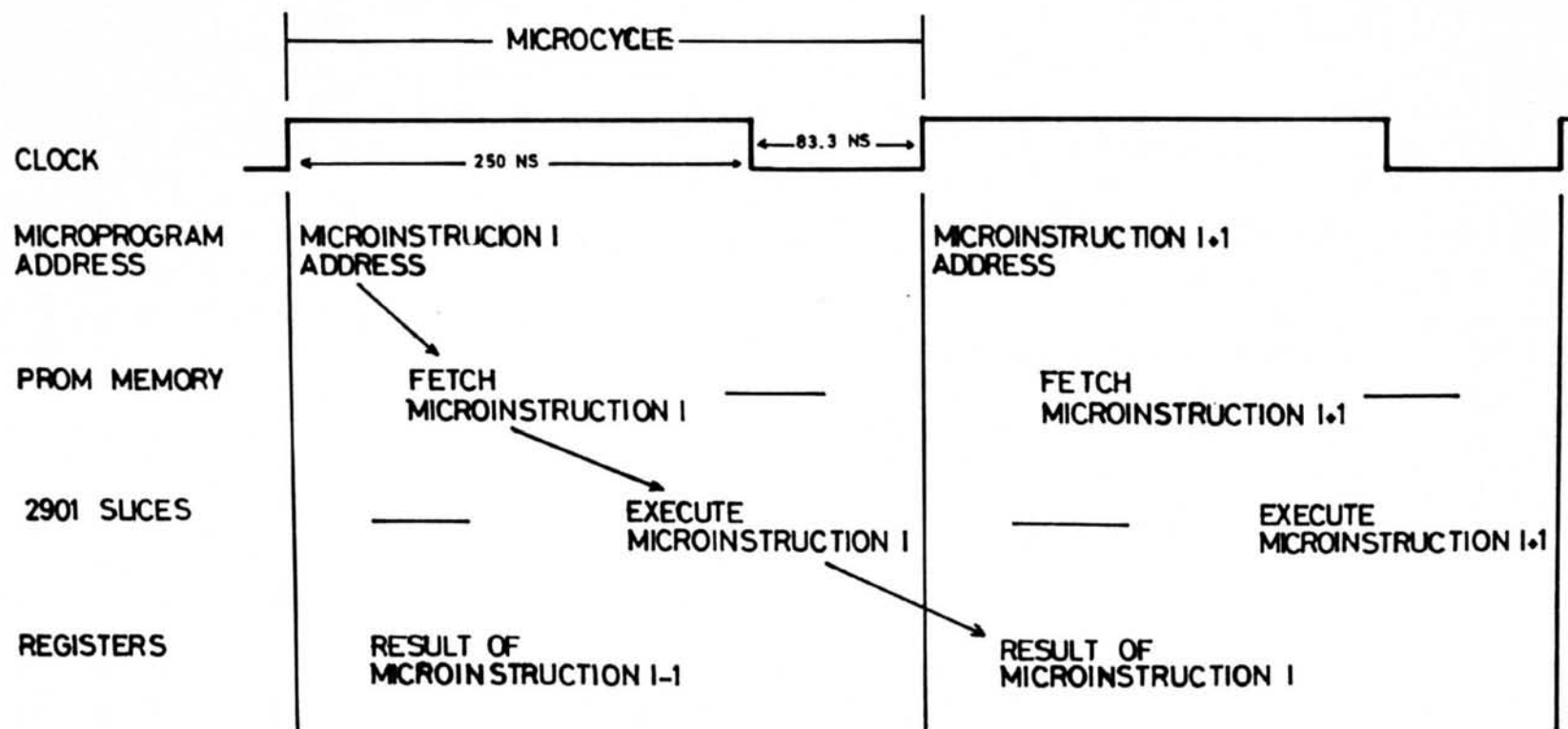


FIGURE (3.11) CLOCK CIRCUIT AND CLOCK PULSES.



FIGURE(3.12) MICROCYCLE TIMING FOR THE SYSTEM OF FIGURE(3.1).

the PROM output, execution begins in the 2901 ALU slices. On the next rising edge of the clock, the 2901 ALU result is gated into the registers and the status signals which are being input to the subsystem circuits are assumed to be stable. If an unconditional branch microinstruction is to be executed then when the outputs are available from the PROM memory, the control signals are sent to the counter to cause it to load the branch address. No 2901 ALU activity occurs. On the next rising edge of the clock, the branch address enters the counter and the address is input to the PROM. The execution proceeds as before. There is no difference in the microcycle of a branch and nonbranch microinstruction in this system. However, while the PROM memory is being accessed, the 2901 ALU must remain idle, and while the 2901 ALU executes, the PROM memory must remain idle.

In the 2901 ALU, some internal operations require longer time to execute than others. One or more of these operations requires the maximum length of the time to complete. This is called the worst case delay path. The minimum total width of the microcycle, c_p , is the sum of the worst case fetch and execute times.

3.9 Conclusion

The hardware construction of an 8-bit microprogrammed processor which is constructed with 4-bit bipolar microprocessor slices has been described. A bit-slice approach was chosen for this project, since the flexibility and speed required cannot be obtained using a single fixed instruction set microprocessor. The system is configured such that it can support a

microinstruction cycle of up to 250 n.sec.

CHAPTER 4

System Microprogramming Features

4.1 Microprogramming

Microprogramming was first suggested by Wilks in the early 1950s (54), (55). With the development of fast, inexpensive LSI devices, commercial use of microprogramming spread into the microprocessor-based systems domain. Present microprocessors employ microprogramming in two ways. The first is the traditional method of using microprograms to perform machine instructions (56). The second is to combine bipolar bit-slice devices to synthesize a microprocessor or controller system with a particular architecture (57), (42). One can describe microprogramming as (58) the use of a program language (microprogram) that explicitly and directly controls the sequence of internal machine-hardware functions (e.g., registers, ALU's, counters, busses, memory). In this way, microinstructions specify control terms that cause the machine hardware to perform an elemental function such as transferring data from one register to another. The device, in this case, is completely software driven, having no predetermined sequence of operation implemented in hardware, i.e., linkages of microinstructions cause the machine to perform the desired function.

Microprogramming is considered to be the best approach to control a bit-slice system for the following reasons (59), (42), (60), (61):

- 1- A memory (ROM or PROM or related devices) is a substitute

for random sequential control logic circuits. This leads to a more structured organisation of the design.

2- Software test routines can be developed and included in the PROMs or, the normal PROM memory could be swapped with a special test memory by substituting PROMs.

3- Variation of the initial design can be implemented by substituting one or more PROMs (i.e., changing the microprogram), and also adding PROMs expands the system.

4- The microprogram, documented in the definition file and in the assembly source file, serves as the principle documentation of the 'firmware' (62) (because such microprograms have been placed in PROM or ROM, they have been called firmware, i.e., software modules that are "firmly protected" from being changed), this provides a clearer documentation than multipaged schematics can provide.

5- Subsystems can be upgraded by replacing the appropriate PROM more easily than hardwiring or patching new components onto a crowded printed circuit board (PCB), with all of associated difficulty that this entails.

Three measures are useful for defining the microinstruction characteristics (63):

A- Monophase-polyphase characteristic

A monophase microinstruction would generate the control signals used during one clock pulse. A polyphase microinstruction would generate control levels and signals used during two or more clock pulses.

B- Encoding characteristic

This measure refers to the degree of encoding in the microinstruction word. There are two different types. The first is direct encoding, in which case the mutually exclusive signals can be grouping together into fields. These fields are then decoded to produce the corresponding control signals. This type of encoding reduces the size of the microinstruction word. The other type of encoding is known as indirect encoding where the meaning of a field is made to depend on the value of a control field in the microinstruction.

C- Serial-parallel characteristic

This refers to the method used to determine the next microinstruction to be executed. In the serial approach, the generation of the address for the next microinstruction to be executed does not begin until the execution of the current microinstruction terminates. In the parallel microprogram approach, the addressing of the PROM for the next microinstruction is overlapped with the execution of the current microinstruction.

The microprogram size can be expanded in two ways (55), horizontally and vertically. A horizontal expansion of the microinstruction, is implemented by adding more control bits to each and every microinstruction in the microprogram for controlling additional hardware elements. A vertical expansion means that you increase the actual number of microinstructions in the microprogram to perform new functions. Although horizontal expansion allows the microprocessor system to perform more

parallel operations in each microcycle, it has the disadvantage that each bit is dedicated to a single function and, consequently, a maximum number of bits is required, i.e., a much larger amount of microprogram memory is needed. On the other hand, a vertical expansion can increase the capability of the CPU, but the amount of sequential control logic in the system increases. A vertical microinstruction usually involves little parallel operation within the microcycle; instead it initiates a single sequence of events, and hence, the microcycle time increases and the speed is decreased. A combination of horizontal and vertical microprogramming schemes is normally used to meet the specific speed and control memory limitations.

For long microprograms (> 48 microinstructions in length or with microinstructions > 16 bits wide), software development systems are required. These systems allow each field to be defined with symbolic definitions, which is a documentation method. Once the fields are defined, the microcode (microprogram, microinstruction) can be written in symbolic language, similar to a pseudoassembly language, that will provide human-readable documentation. The development system may be used to assemble the microprogram thus written and to create the input to a PROM programmer. Since microprograms, like programs, seldom run properly when first executed, the development system provides simulators and debuggers which allow users to interact with, and monitor the execution of, a microprogram as it is being run on the system. Simulators usually run on a different machine and simulate the actions of the system for which the microprograms

were written.

Given that the microprogramming concept is closely related with the bit-slice microprocessor system, this chapter will describe the microinstruction characteristics and discuss tools and facilities for the development of microprograms.

4.2 Microinstruction Format

The microinstruction has two primary parts. These are:

1- the definition and control of all micro-operations to be carried out

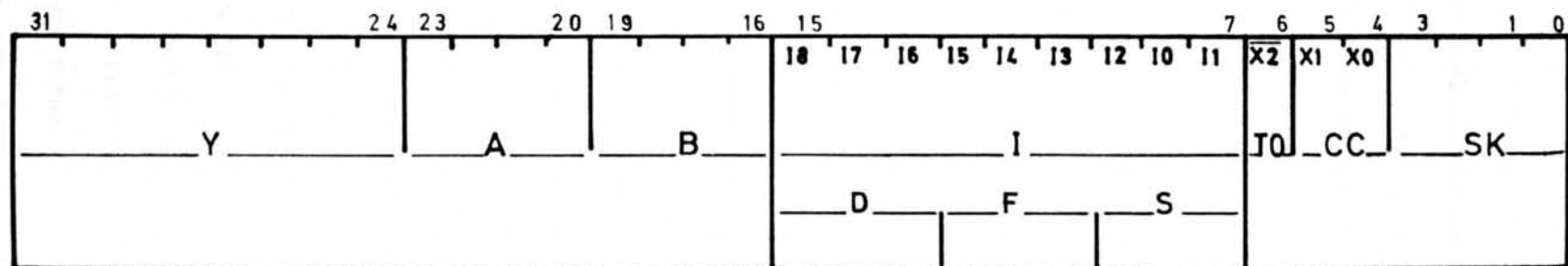
2- the definition and control of the address of the next microinstruction to be executed

The definition of the micro-operations to be carried out includes such things as ALU source operand selection, ALU function, ALU destination, carry control, shift control, and data-in and data-out control. The definition of the next microinstruction function includes identifying the source selection of the next microinstruction address or supplying the actual value of that microinstruction address.

The thirty-two bit microinstructions of the 2901 microprocessor system used in this investigation consist of ten fields which provide some parallel operation as illustrated in Figure (4.1).

SK (4 bits) is the test and skip control field for selecting one-of-sixteen skip flags denoted by 0 through F, these values will correspond to predefined bits within the system, as described before.

CC (2 bits), the carry-select control field, determines the



FIGURE(4.1) FORMAT OF 32-BIT INSTRUCTION WORD
FOR THE 2901 SYSTEM OF FIGURE(3.1).

carry-in state as illustrated in Figure (3.6).

TO (1 bit), the external write-only registers strobe, enables the current data value on the data bus to be strobed into an external register when it is '0'. This field also activates the LOAD control line of the microprogram counter during branch operations.

I (9 bits), is the 2901 instruction control lines. Also shown in Figure (4.1) is

S (3 bits), source operand field, used to determine what data sources will be applied to the ALU-slices.

F (3 bits), function field, used to determine what function the ALU will perform.

D (3 bits), destination format field, used to determine what data is to be deposited in the Q-register or the internal register array.

B (4 bits), is the B address field, the four address inputs to the internal register array used to select one register whose contents are displayed through the B-port and into which new data can be written when the clock goes LOW. This field also selects one-of-sixteen external write-only registers (TO-registers) to be loaded from the data bus.

A (4 bits), is the A address field, The four address inputs to the internal register array used to select one register whose contents are displayed through the A-port. It also selects one-of-sixteen external read-only registers (From registers) whose contents are output onto the data bus.

Y (8 bits), the control store literal field, an 8-bit data word which represents a number or an address. It is used for

assignment to a register or to indicate the address of the next microinstruction to be executed. This is rather like the immediate field used in some machine language instructions.

4.3 Microinstruction Implementation

The system microinstructions have the capability to perform two distinct operations simultaneously- An ALU/shifter operation and a conditional branch or skip operation. The additional capability to perform other operations simultaneously (such as external register handling, and carry control) suffices to classify these microinstructions as horizontal. Direct (or one level) encoding was implemented in the representation of microinstructions, this is due to the fact that most of the micro-operations that a particular subsystem can perform were represented in the microinstructions as a field rather than as individual bits. Since the micro-operations that were combined into a field are mutually exclusive, no information is lost in this single level encoding scheme. There is only one hardware subsystem, the control decoder, in which indirect (two level) encoding was used. The flag TF, generated by the TO decoder (it is not a direct control bit output), and the Y-field (Y0 - Y2) are combined to select the control flags (C0 - C7).

When no control signals are to be enabled by a given set of bits, the bits are all placed in the 0 state. In this case, a unique binary code must be assigned to this condition since it represents a legitimate control pattern for a control field. When the all-0s bit pattern is decoded, no action is generated by that field during that microcycle. Typically, this all-0s bit

pattern is used to represent a microcycle no-operation (NOP). Microinstruction implementation is serial (fetching the next microinstruction to be executed is started after the execution phase of the present microinstruction). The basic system clock cycle is 330 n.sec, and in normal operation, a microinstruction is read from the PROM to the subsystem and executed in one clock cycle (there are no suboperations performed, and all operations specified by a microinstruction are executed simultaneously).

4.4 Microinstruction Sequencing

Three techniques were combined for accomplishing this microinstruction sequencing. These are:

- 1- sequential execution
- 2- skip control
- 3- multiple sequences

4.4.1 Sequential execution

In this case the PROM address of the next microinstruction to be executed is one greater than the address of the microinstruction being executed. The microprogram counter increments by one on each clock cycle.

4.4.2 Skip control

The sequential execution of microinstructions may be altered by the skip micro-operation. The microinstruction has a skip micro-operation in which the microprogram counter is incremented by two instead of one. A conditional skip micro-operation facilitates an efficient one microinstruction subroutine.

4.4.3 Multiple Sequences

The sequential execution of microinstructions may be altered by unconditional branch micro-operations. The load control of the counter is a single bit (T0) defined by the microinstruction. Whenever this bit is at logic '0' a load will be enabled. If the load is enabled, the new (branch) address contained within the PROM will be parallel loaded into the counter. The branch address originates from two sources; the literal (addressing) field of the microinstruction, in which case it is the field supplying the actual value of the address. The other source is the external read-only registers (From registers), in which case, the data inputs to the counter receive the start address.

Another useful facility combines sequential and skip execution, by assigning an address to a label or register. This facility, with the unconditional branch micro-operation can be used to implement loops in microprograms.

4.4.4 Start Address

The microprogram counter is reset to zero on startup, at which the microinstruction must be in the form:

$$T0 = F7 + 0, S7$$

This loads the microprogram counter (T0) with the data on the data bus which is the output of the 'From' register (F7). Skipping to give a 0 LSB, when S7 is true '1', and not skipping to give a 1 LSB, when S7 is '0'. Start addresses may now be assigned to any location in the PROM. The 'From' register 'F7' contains address lines driven by switches to allow all the

different states to be valid.

4.5 Special Microassembler

One of the difficulties of using the 2901 bit-slice system is the difficulty with software support for the completed system. It is evident that the system described in the previous chapter is unique and a special assembler and simulator will be required. The following two sections will describe the assembler and simulator that have been used in this work for developing the microprograms.

The architecture of the 2901-system is designed for maximum speed of operation in its application, which differ from those of MOS microprocessors. As a result, the 2901-system assembler contains features which are unique to this system application, e.g. the assembly language differs from standard assembly languages, as described below.

A microassembler (we will call it an assembler from now on) was developed for the microinstruction format described in the previous two sections and illustrated in Figure (4.1). It assembles a microprogram written in a symbolic language into the bit patterns for subsequent use in microprogram PROMs, and provides various convenient features for use in writing microprograms. The assembler was written in the CORAL programming language (64) and was implemented using a CORAL compiler which runs under the UNIX operating system for PDP-11 computers (65), (66). The language comprises several fields correspond to microinstruction fields (operators, operands, shift, skip, and carry control), label field, and comments. Operations specify

transfers of information among registers. Unary and binary operations can be written in algebraic notation, and unconditional execution can be represented by simple 'branch' statements.

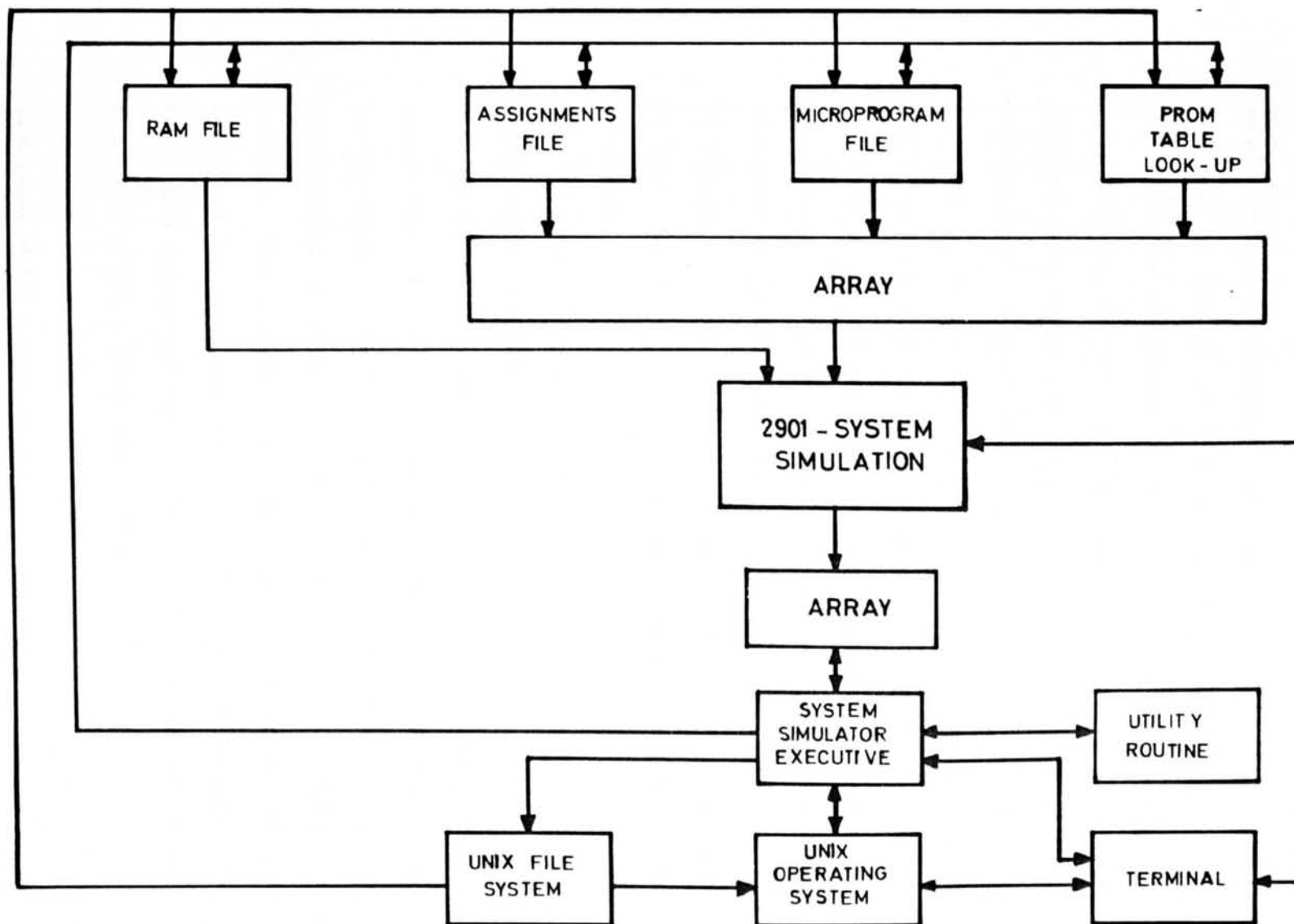
The address space is 512 words. This is achieved using the 8 bit literal field plus the skip option. In consequence a branch instruction requires a knowledge of whether the executing instruction is at an odd or even address and whether the target is at an odd or even address. There are two assembler directives which force these conditions, *EVEN and *ODD. The addressing given in the generated list is a 3-digit number, the numbers being hex, hex, binary (the least significant being binary). The source program contains function assignments and the symbolic microprogram itself. The assembler makes two passes through the source. On the first pass it makes entries into the label table for each label, assigning it to a value equal to its address, starting at location 000. On the second pass it converts the symbolic values to the binary encoding of the microinstructions and stores these in internal memory (binary file) for subsequent processing. In addition to the binary output, an assembled listing file 'ass.lst' can be produced. This file is the output medium by which the assembler communicates its results to the 2901-system microprogrammer. This file is prepared by re-reading the input source microprogram and matching each line to the assembled code. It shows, for the microinstruction, each symbolic representation, its translated binary representation, and its assigned PROM address.

Associated with the assembler is a 'converter' program. It takes the binary output of the assembler and, given the description of each PROM location, produces the proper output (for input to the PROM programmer) for each PROM.

4.6 Software Simulator

Another development aid for the microprogram is a software simulator. It was decided to use a software simulator in order to test, debug, and optimise the microprogram before 'burning' a set of PROMs. The simulator being used is specifically for the present system (67), and runs under the UNIX operating system. The simulator provides an interactive microprogram development and debugging facility which operates exclusively in UNIX with no need for the 2901-system or any associated hardware. It includes input/output handling and has the ability to access registers, set breakpoints (break on condition mode), and single step execution mode. Execution can be halted at any time for observation of the register contents, change in the breakpoint conditions, after which execution can be continued without any loss. Preparation of microprograms is achieved by using the assembler (and converter) which generates a file that the simulator can load directly into its microprogram memory. Diagnostic messages are printed in response to erroneous operations and special system conditions.

A block diagram of simulator is shown in Figure (4.2). The system box represents the simulation of the 2901-system architecture as described in the previous chapter. The operation of the simulator is controlled by the simulator executive system



FIGURE(4.2) BLOCK DIAGRAM OF SYSTEM SIMULATOR.

which interprets commands and invokes required utility routines (macros). A number of files are associated with the simulator. The RAM file corresponds to the random access memory which is interfaced to the system. The microprogram file contains data that are to be loaded into the PROMs of the simulator. The table look-up files and signal sources contain data to be read into the internal registers. The simulator has thirty seven control/skip status and register assignments which correspond to dedicated hardware in the system. The simulator handles these assignments through its communications links with the terminal or the UNIX file system in the following ways:

- (i) Default. A request is printed on the terminal for the value of the assignments. Execution resumes when the assignment value is entered.
- (ii) Optional. The assignments can be read from a UNIX file specified as an assignments file.

The simulator can access files in the UNIX file system, so that system files can be loaded from and written to UNIX files. Data to be entered into registers directly from the terminal may be hexadecimal or binary. Files for the system are arrays in memory. They are four different types in accordance with the length of the data they store. The file types, characterised by their data format and their use by the simulator, are as follows:

- 1- 6 types of 10K x 8 bits data RAM file
- 2- 4 types of 15 x 8 bits data assignments file (option)
- 3- 8 types of 512 x 4 bits data microprogram file
- 4- 8 types of 512 x 4 bits data table look-up files

The UNIX files are in ASCII format. They contain a filetype

declaration which must match the binary type required in this system.

One can be constantly interacting with the simulator and this interaction will be controlled by the use of the 'command library'. The command library includes commands for re-initialisation the simulator, setting and reading 2901-system registers, resetting monitor points, and transfer of files between the simulator and UNIX environments. Another group of commands are used to perform checks on breakpoint variables during execution of the microprogram and print messages when breakpoint conditions are met. A breakpoint can be set on a microinstruction address, on a register value, and on the number of clock cycles executed. The run (rn) command initiates execution of the microprogram until a break condition is met or for a specified number of clock cycles. While the run command provides for continuous execution of a microprogram, the single step command (ss) executes only one microinstruction.

The simulator makes full use of the connected terminal being used. When one invokes the simulator, the terminal displays a "start-up" frame. The simulator command level is indicated by a ":" prompt character. First, the simulator's microprogram memory is loaded, using **pc** or **pt** commands, with the binary object file, which was generated previously by the 2901-assembler from a source program. Next, assignments, RAM, breakpoints, and any number of monitor points up to 18 points are set. Various actions may be taken when the breakpoint is reached; these are:

- 1- Print the monitor points on the terminal for investigation

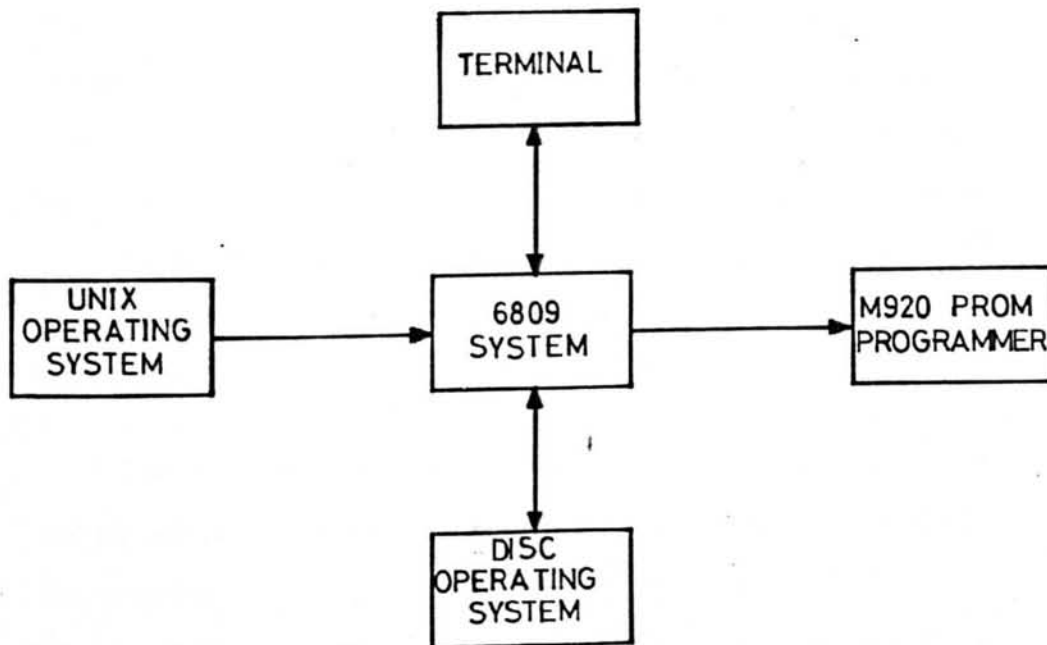
2- Write into the RAM file and stop execution, or

3- Initialise the execution.

This gives a general idea of the simulator operation.

4.7 PROM Programming

A multi-interface system has been used for programming the PROMs. The block diagram for this interface is shown in Figure (4.3). The PROM programmer used was the PRO-LOG M920 (68), which permits entry from either the 6809 system (69), or copied from another PROM. This PROM programmer puts successive pulses onto each bit at the recommended rates (current specification is usual for "fusible-link" bipolar PROMs) of the PROM manufacturer.



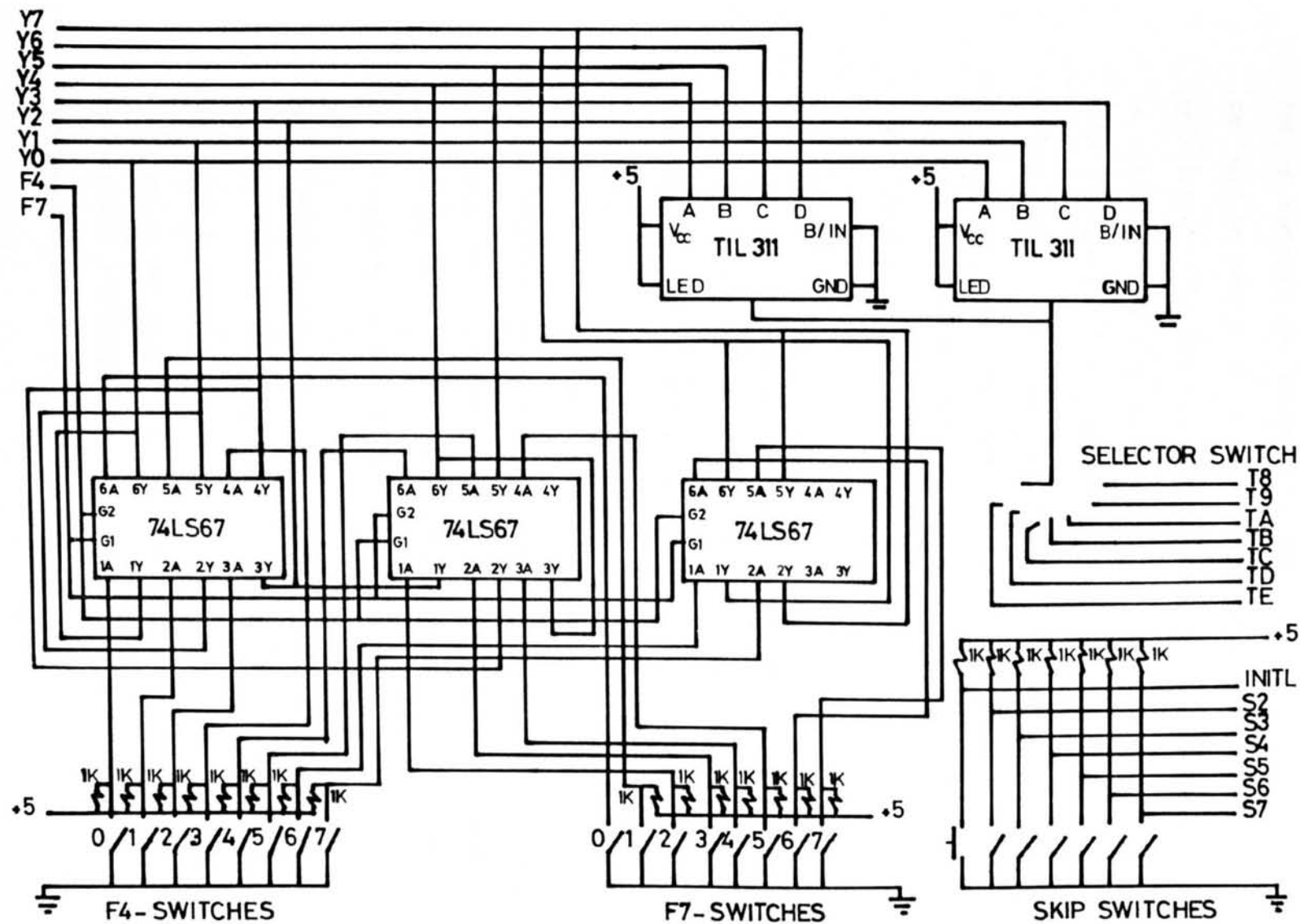
FIGURE(4.3) PROM PROGRAMMING INTERFACE.

The assembler output files are transferred from UNIX to the 6809 microprocessor system using a special program. These files, in turn, can be maintained in 6809 system discetes for subsequent use. The 6809 system is connected to the parallel interface of the M920 via a 25-pin, D-type connector. The parallel interface provides eight parallel input data lines, eight parallel output data lines, seven handshake control lines, and an internal handshake program (this PROM programmer uses an Intel 4004 microprocessor to provide this and other features).

4.8 Development Test Equipments

A sophisticated microprogram requires special equipment for testing the system functions and circuits. These supporting tools should be efficient and easy to implement. The test tools that have been used in this system are classified into two groups. Software test microprograms which enable us to examine the system functions (these will be described in the next section) and external test equipment which allow us to control and investigate the software routines. The external test equipment that was used was a **test box**, a logic analyser, and an oscilloscope.

The software test routines requires a test box which allow us to select a particular test, to set parameters, and to display the results. The circuit diagram of the test box is shown in Figure (4.4). It consists of two TIL 311 types (an hexadecimal display with integral TTL circuit to accept, store, and display 4-bit binary data) (70) and three 74LS367 types, hex bus drivers. Three rows of address switches, are included that can be used to



FIGURE(4.4) TEST BOX CIRCUIT.

set up addresses or parameters on the F4 and F7 registers. It can also be used to set or clear any of seven different types of skip flags. The test box contains also a selector switch which is used for displaying the contents of any of the general use 'TO' registers. The test box is directly interfaced to the system via an 8-bit data bus, and the 'TO' and 'From' strobes. It should be noted that the initialisation control line 'INITL' is generated from the test box and supplied to the rest of the system circuits.

4.9 Test Software

It was mentioned before that one considerable advantage which is derived from the use of microprogram control of the 2901-system is that software test routine can be developed and included in a special test memory by substituting the normal PROMs. A set of software microprograms were used to test the system hardware, any of which could be selected by setting up the appropriate address on the test box registers. Each microprogram contains one or more tests for a particular part of the hardware. These microprograms are described below.

4.9.1 Control Decoder Test

This microprogram generates control pulses corresponding to the value to which F4 is set on the test box. The pulses are observed on an oscilloscope to test for correct decoding. A continuously changing value on display TE indicates that the microprogram is running.

4.9.2 "To" and "From" Decoder Test

This microprogram generates pulses on lines T1 through TF in sequence and on lines F1 through FF in sequence (notice that T0 and F0 are special purpose control lines). The pulses are observed on an oscilloscope to establish the correct operation of the decoders.

4.9.3 2901-Slices Internal Registers Test

This microprogram tests the 'R' registers on the 2901-slices by incrementing the 'Q' register, loading its value into registers R0 through RF in turn, and then comparing each register to 'Q'. A fault causes the microprogram to loop indefinitely in an error loop. This is indicated by the display TA. A successful run through the microprogram causes the 'Q' register to be incremented and the microprogram repeated.

4.9.4 Up Shift Test

This microprogram tests the upward shift function on the 2901-slices. A 'ONE' is shifted around registers Q and R0 using the up-shift function. A second 'ONE' is shifted around registers R1 and R2 using the carry. Errors are checked by comparing R1 and Q, and R2 and R0. R3 is an error counter displayed on T8.

4.9.5 Down Shift Test

This microprogram tests the downward shift function on the 2901-slices. The value 'hex 80' is loaded into registers Q and R1, and then shifted down one bit at a time, comparing its value to the pre-defined value in register R3 between shifts. An error

will increment RA while a successful shift will increment R9. RA is displayed on TA, while R9 is displayed on T9. Resetting S6 introduces a delay loop which allows displays to be read.

4.9.6 2901 ALUs Arithmetic Operation Test

This microprogram tests the 2901 ALU's by carrying out its eight different operations on two registers which are loaded from the test box. The results of the operations are displayed on T8 through TE. Switch S6 selects either R-S or S-R operation.

4.9.7 Carry Control Test

This microprogram tests the carry generator circuit for correct operation. If F7 is set to 'hex FF' and F4 to any value, then TA will display (the contents of F4 + 1) if S6 is clear and contents of F4 if it is set. T9 will always display the contents of F4 while T8 will display 'FF' if S6 is set, and '01' if it is clear.

4.9.8 FIFO control tests

This microprogram tests the FIFO control system and consists of three consecutive tests. For the first test, the FIFO is loaded with 'hex 55' and a reset pulse is generated. If the FIFO fails to clear, a branch is made to an error routine, otherwise the second test is entered. In this second test, the FIFO is loaded with sixteen characters, a check for 'FIFO full' being made before each entry, and a check for 'FIFO empty' made after each entry. When sixteen characters have been loaded, a check for 'FIFO full' is made. If the flag is set, the third test is entered, otherwise the error routine is entered. The error

routine is also entered if any of the previous checks fails. In the third test, the FIFO is unloaded one character at a time, a check for a 'FIFO empty' being made before unloading each character, and a check for 'FIFO full' made after unloading a character. When sixteen characters have been unloaded, a check for 'FIFO empty' is made. If the flag is set, the program loops back to the first test, otherwise the error routine is entered. The error routine is also entered if any of the previous checks fails. A failure in the first test is indicated by T8 displaying the value 'FF'. R9 is an error counter, and T9 displays the total error count.

4.9.9 FIFO Data Tests

This microprogram tests the FIFO for correct retention of data. The FIFO is initially cleared as are registers R0 through R3. The FIFO is then loaded with the contents of registers R0, R1, R2 and R3 in sequence. The data is then read back and compared with the register contents. An error increments R9 and restarts the program. A successful pass causes R0 to be incremented and the cycle repeated. Every time R0 reaches 'hex FF' R1 is incremented, and when this reaches 'hex FF' R2 is incremented. R3 is likewise incremented when R2 reaches 'hex FF'. The FIFO is therefore tested for all possible combinations of data. T9 displays the value of the error counter R9, while T8 displays either the value of R3 if S6 is clear, or the value of R2 if S6 is set.

4.9.10 Output Enable Test

This microprogram tests the output enable on the 2901-slices for correct operation. F4 and F7 are set to any value. TC and TD should display the value of F4. T8 and T9 should display the value of F7 while TA and TB should display (the value of F7) AND (the value of F4).

The FIFO transmit/receive tests will be described in the subsequent chapters. The test microprograms were written such that they will loop continuously permitting diagnosis with a logic analyser and an oscilloscope, that will isolate faults to particular areas.

4.10 Conclusion

The characteristics and implementation of a 32-bit microinstruction for the system of Figure (3.1) has been described. Microprogram development aids have been discussed. One of these aids is a software simulator, which can be used without access to the 2901-system hardware environment. This simulator allows us to monitor run-time characteristics of microprograms which cannot be observed using the system itself. The assembler and simulator discussed in this chapter have been used extensively in applications described elsewhere in this thesis. The method was used for programming the PROMs has also described.

CHAPTER 5

Implementation of Direct Sequences By Microprocessors

5.1 Introduction

It has been mentioned in the introductory chapter that the object of this work is the realisation of the signal processing requirements of a spread spectrum communication system using digital techniques implemented with the aid of fast microprocessors. In all spread spectrum systems synchronisation acquisition and tracking is of prime importance. The behaviour of synchronisation systems in the presence of multipath propagation, unacceptable levels of interference, and secondary cross-correlation peaks of the sub-sequences used for rapid acquisition have to be studied to set thresholds on performance in practical systems. The cause of thresholding lies in such things as tracking loss, and thresholding of the correlation detector. In general, direct sequence spread spectrum communication systems are the most widely used spread spectrum systems (3), (6), (7), (71), (72). In direct sequence modulation the baseband information is added (modulo-2) to a digital code sequence whose bit rate is much higher than the information signal bandwidth. This process has the effect of "spreading" the signal energy over a bandwidth equal to twice R_c , the system's code clock rate (7). For good correlation properties and ease of generation, "maximal" length pseudo-noise sequences (m-sequences) were used. Despreading, at the other end, is obtained by correlating the received spread spectrum signal with a similar

local reference signal (code). When the signals are matched (i.e. synchronisation occurs), the baseband signal collapses to its original bandwidth before spreading. Synchronisation is generally achieved in two stages

- a) Acquisition or coarse synchronisation
- b) Tracking or fine synchronisation

The spread spectrum system is required to accomplish synchronisation in the presence of interference and transmission distortion.

This chapter discusses the analysis and implementation, in both software and hardware, of the functions which are concerned with direct sequence spread spectrum systems. In particular, maximal length sequences, and sequence-inversion keying (SIK) modulation are considered. The difficult problem of initial synchronisation of the system and the methods by which synchronisation can be maintained are then examined.

5.2 Pseudo-noise Sequences

Binary pseudo-noise (PN) sequences (which are also called shift-register sequences or m-sequences) are the basis for the direct sequence spread spectrum implementation. These imply a deterministic string of binary digits that repeat only after a relatively long period and have statistical properties similar to those of true random numbers. These sequences can be reproduced by any authorised terminal. Pseudo-noise sequences have been known for more than twenty-five years. During that time, results have been obtained on the structural properties and are used in many applications such as range-finding, modulation, and

synchronisation (72),(3). Recently, MacWilliams, Sloane, Sarwate, and Pursley (10), (11) have been examining certain properties of these sequences and their applications in spread spectrum communications. The sequences that have received the most attention in the literature are the binary maximal-length linear feedback shift register sequences which are known as m-sequences. In m-sequences, which are the type used in this work, the maximum length sequence L is $2^n - 1$ bits, where n is the number of stages in the shift register.

5.2.1 Generation and Properties

At present, MSI pseudo-noise sequence generators are available which allow limited code generation at rates up to several Mbps (73). In this project, software methods were developed for the 2901 system which use a few bytes of PROM and do not require any R/W RAM.

A particularly convenient method for generating the m-sequences is by using an n-stage shift register with a feedback term formed by the modulo-2 addition of several stages which can be specified by its feedback polynomial,

$$h(x) = h_0 + h_1x + \dots + h_nx^n \quad (5.1)$$

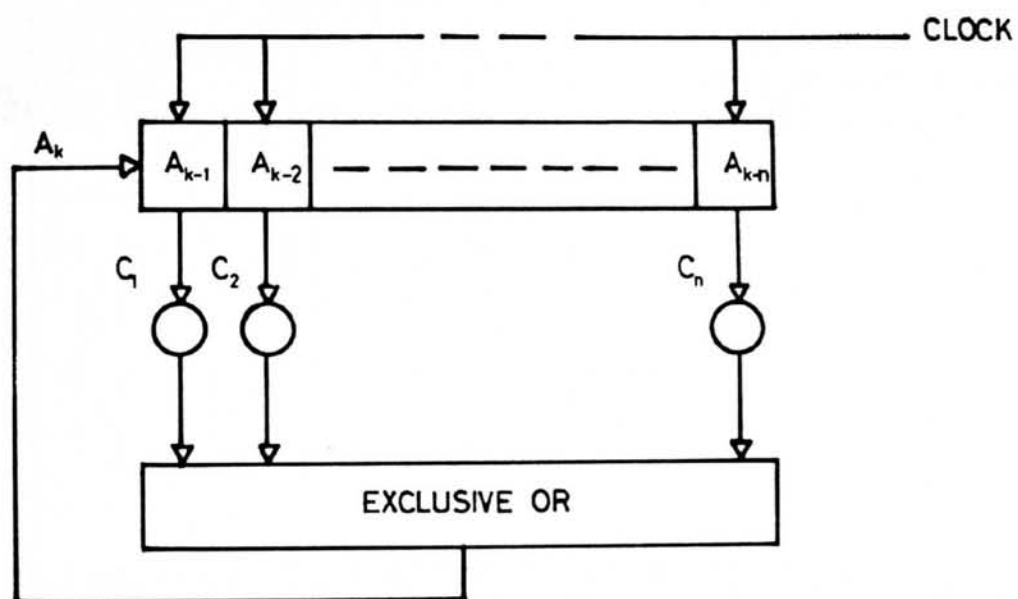
where the degree n of the feedback polynomial is the length of the shift-register generator and the binary coefficient h_i (0 or 1) represents the feedback tap on the generator. The equivalent of this operation can also be performed efficiently using microprocessors (see Appendix B).

Mathematically, the generation of a binary m-sequence $\{a_k\}$ is defined by the operation

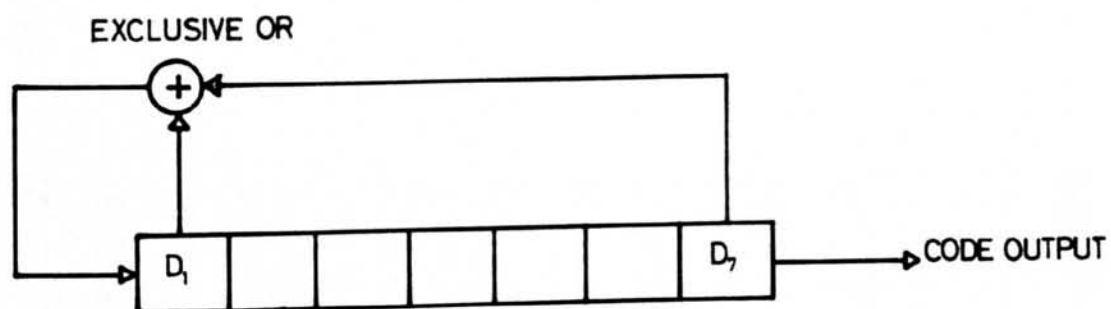
$$a_k = \sum_{i=1}^n c_i a_{k-i} \pmod{2} \quad k=0,1,\dots,L-1 \quad (5.2)$$

where the sum is modulo 2 addition and both c_i and a_k take the values 0 or 1. The coefficients c_i , $i=1,2,\dots,n$ do not depend on n and must be known in order to specify an m-sequence. The k th state of the m-sequence generator is, therefore, defined by the past n terms of the sequence a_{k-i} , $i=1,2,\dots,n$ as shown in Figure (5.1).

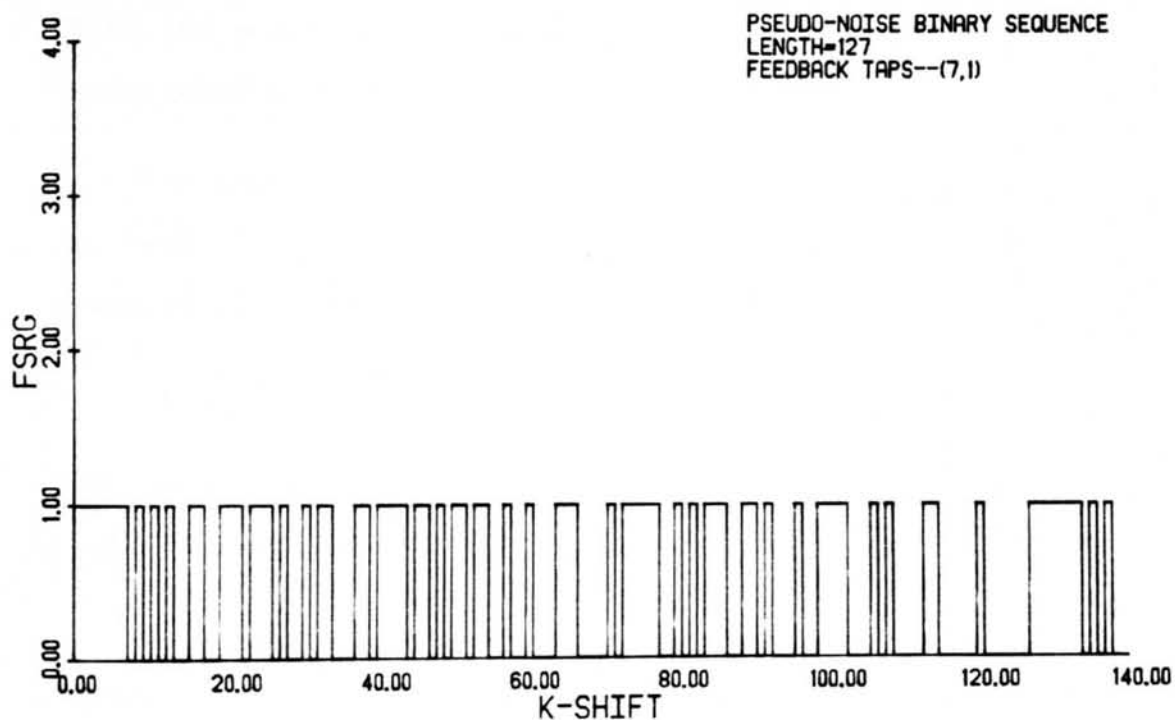
Figure (5.2) illustrates a shift register consisting of 7 stages, representing memory elements or flip-flops, each containing a 0 or 1 (all-zeros state is not allowed to exist). Outputs from the last stage (D_7) and an intermediate stage (D_1) are combined in a modulo-2 adder or EXCLUSIVE-OR gate, defined by $0 + 0 = 1 + 1 = 0$, $0 + 1 = 1 + 0 = 1$, and fed-back to the input of the first stage. At each clock cycle the contents of the stages are shifted one place to the right. In this particular example the code sequence generated is shown in Figure (5.3), which is cyclic with a total period 127 times the period of a single flip-flop output pulse. This is the longest code sequence that can be generated by 7 stages in a shift register; that is for n stages the longest sequence that can be generated is $2^n - 1$. For an n -stage register, there are $\phi(L)/n$ maximal sequences that can be generated by using different linear combinations of feedback taps (where $\phi(L)$, is the Euler ϕ -function, i.e., the number of positive integers including 1



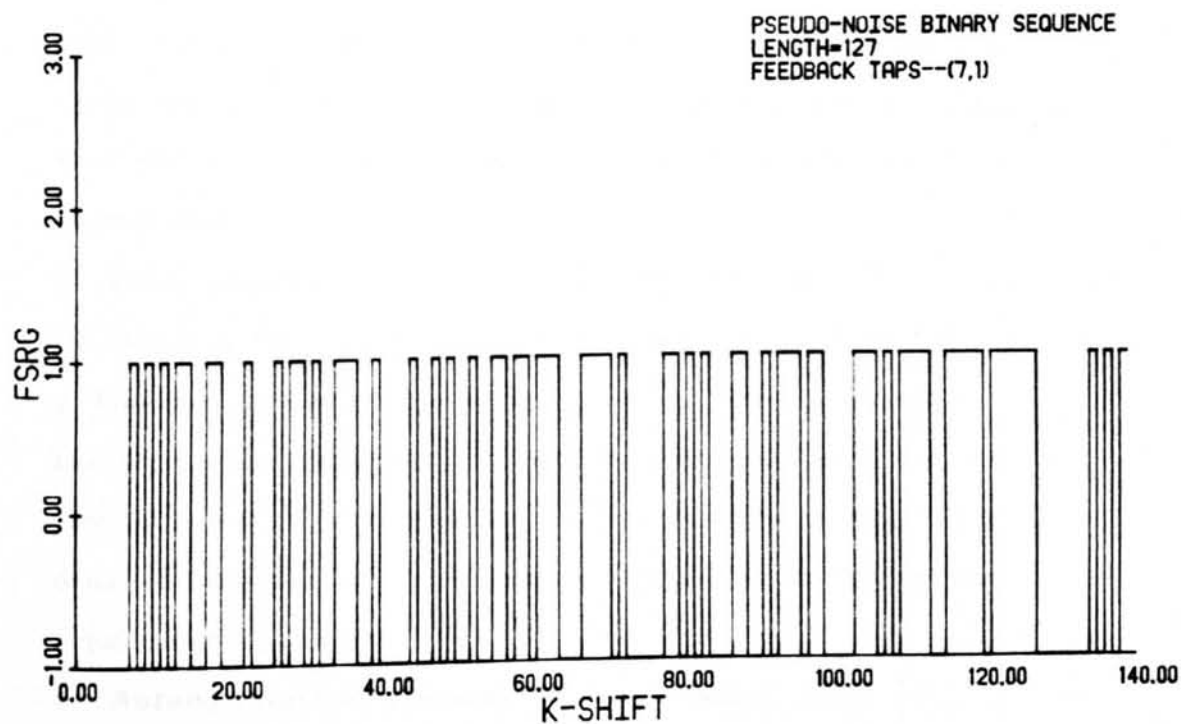
FIGURE(5.1) BINARY SHIFT REGISTER.



FIGURE(5.2) SHIFT REGISTER GENERATOR FOR 127-BIT m-SEQUENCE.



FIGURE(5.3a) TYPICAL m -SEQUENCE $a(0,1)$ (PERIOD =127).



FIGURE(5.3b) TYPICAL m -SEQUENCE $b(+1,-1)$ (PERIOD =127).

that are relatively prime to and less than L). Feedback connections have been tabulated for maximal code generators from 3 to 100 stages, so that any length from 7 through $2^{36}-1$ are readily available (7), (3).

For most cases it is convenient to consider the m -sequence as formed from the digits $\{+1, -1\}$ instead of $\{0, 1\}$, the $\{+1, -1\}$ sequence $\{b_i\}$ is related to the $\{0, 1\}$ sequence $\{a_i\}$ by

$$\{b_i\} = \{1 - 2a_i\} \quad (5.3)$$

this enables modulo-2 addition to be replaced by conventional multiplication or vice versa.

For convenience some relevant properties, for our application, of m -sequences are summarised below.

1- The one-zero balance property : in every period of the sequence the total number of ones (2^{n-1}) always exceeds the total number of zeros ($2^{n-1}-1$) by one. For a 127 bit code there are 64 ones and 63 zeros. This property has the effect that the DC component in a code or in a code-modulated signal can be neglected.

2- Runs property : in any code sequence there are $2^{n-(p+2)}$ runs of length p for both ones and zeros, where runs is defined to be a maximal string of consecutive identical symbols, except that there is only one run containing n ones, and only one run containing $n-1$ zeros. There are no runs of zeros of length n or ones of length $n-1$. This property is useful for testing code sequences of any length.

3- Autocorrelation property : if a maximal code $\{a_i\}$ is

correlated with a replica of itself during a complete sequence period L , then the normalised autocorrelation function varies linearly from 1 to 0 in the range $0 < \tau < T_c$ (the sequence chip) phase shift and equals to 0 for all other values of phase shift, i.e. for a complete period L the normalised autocorrelation function is given by

$$R_a(\tau) = \begin{cases} 1 - |\tau|/T & 0 < \tau < T \\ 0 & \text{elsewhere} \end{cases} \quad (5.4)$$

For convenience, the periodic unnormalised autocorrelation function is often used and is defined as

$$\begin{aligned} R_a(l) &= \sum_{k=1}^L a_k a_{k+l} \\ &= \begin{cases} L, & \text{if } l = 0 \pmod{L} \\ -1, & \text{if } l \neq 0 \pmod{L} \end{cases} \end{aligned} \quad (5.5)$$

Thus binary m -sequences have two-valued autocorrelation functions. This is the most important property and it will be discussed in detail in the following section.

4- Shift and add property : the modulo-2 addition of a maximal code and a cyclic shift of itself is another replica with a phase shift different from either of the originals. This property, which allows generation of any desired code phase, can be used in a multiple correlators scheme in order to reduce effective synchronisation time.

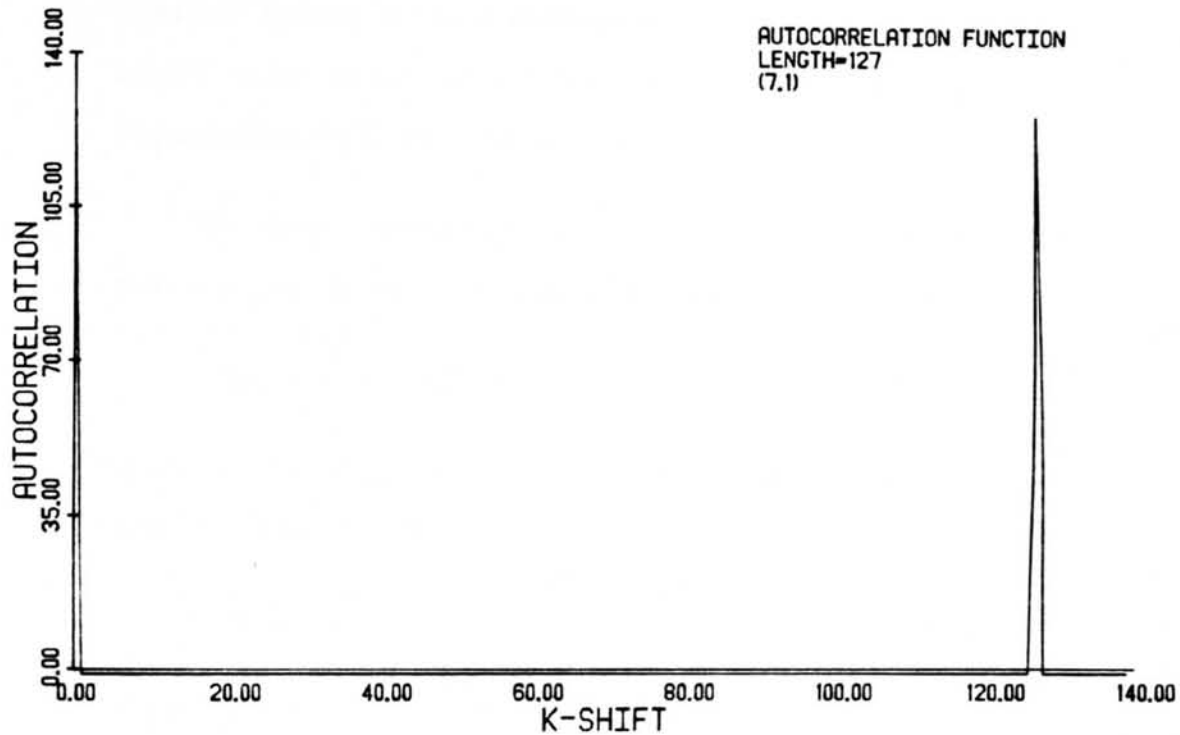
5- Window property : if a window of width n is slid along a complete code period, each of the $2^n - 1$ nonzero binary state n -tuples exists only once.

5.2.2 Correlation Functions and Power Spectra of Codes

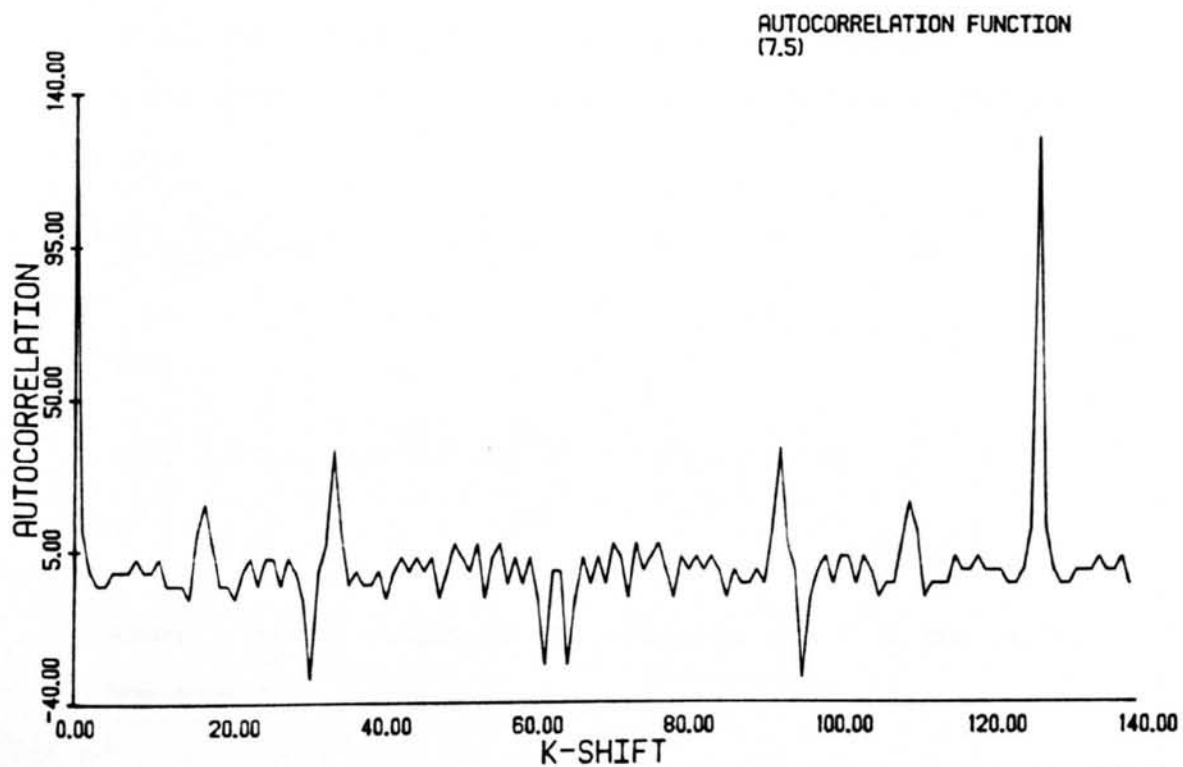
The autocorrelation properties of m-sequences are interesting from the point of view of synchronisation as the autocorrelation function is periodic and two valued, with a peak only at the zero shift point. This property is important in choosing code sequences that give the least probability of a false synchronisation. Code sequence (unnormalised) correlation can be expressed as the number of agreements (A) minus the number of disagreements (D) when the code and a phase-shifted replica of itself are compared bit by bit. The normalised correlation is then given by

$$R_a = (A - D) / L \quad (5.6)$$

The unnormalised autocorrelation (crosscorrelation) function of a sequence (two sequences) is the set of correlation values of the sequence (one sequence) with all cyclic permutations of itself (the other sequence). This is a generalised correlation definition which coincides with the two types of code sequence representation $\{0,1\}$ and $\{+1,-1\}$ as mentioned above. Figure (5.4a) shows the autocorrelation function of a 7-stage shift register generator, generating a 127-bit m-sequence of Figure (5.3a). Autocorrelation properties for nonmaximal sequences may be different from those of the m-sequences, an example in Figure (5.4b) shows the autocorrelation for a nonmaximal sequence generated from the same shift register but with different feedback taps. The Figure shows minor correlation peaks which are dependent on the code and are caused by partial correlations during the correlation process. When such minor correlations



FIGURE(5.4a) A 127-BIT m -SEQUENCE AUTOCORRELATION FUNCTION.



FIGURE(5.4b) NON-MAXIMAL SEQUENCE AUTOCORRELATION FUNCTION.

occur, the receiving system's ability to synchronise may be impaired because it must discriminate between the major (0+1 bit shift) and minor correlation peaks, and the margin of discrimination (index of discrimination (ID) (7)) is reduced.

The power spectrum of an m-sequence may be determined from the autocorrelation (74) function by application of the equation:

$$S(w) = \int_{-\infty}^{+\infty} R(\tau) e^{-jw\tau} d\tau \quad (5.7)$$

Defining the sequence autocorrelation function as equation (5.4), then its Fourier transform is

$$\mathcal{F}\{R_a(\tau)\} = T \text{sinc}^2 \pi f T \quad (5.8)$$

Since $R_a(\tau)$ is periodic it can be expressed as

$$R_a(\tau) = -1/L + (L+1/L)R_a(\tau) * \sum_{n=-\infty}^{\infty} \delta(\tau+nLT) \quad (5.9)$$

where the asterisk indicates the convolution operation. The power spectrum can now be obtained by applying equation (5.7) and using

$$\mathcal{F}\left\{\sum_{n=-\infty}^{\infty} \delta(\tau+nLT)\right\} = 1/LT \sum_{n=-\infty}^{\infty} \delta(f + n/LT) \quad (5.10)$$

Thus

$$S(w) = (L+1/L^2) \text{sinc}^2(w/2\pi f) \sum_{\substack{k=-\infty \\ k \neq 0}}^{\infty} \delta(w-(2\pi kf)/L) + 1/L^2 \delta(w) \quad (5.11)$$

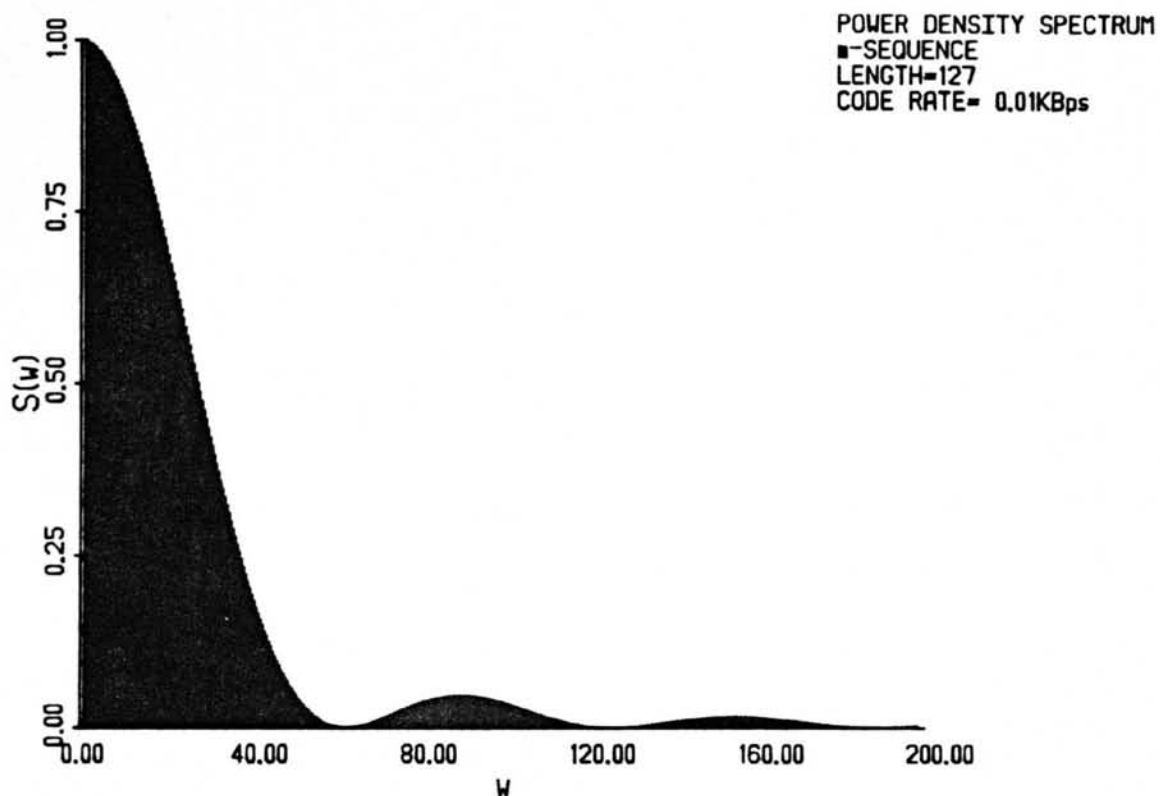
where L is the length of the sequence, and f is the clock frequency, and

$$\text{sinc}^2 x = (\sin \pi x / \pi x)^2$$

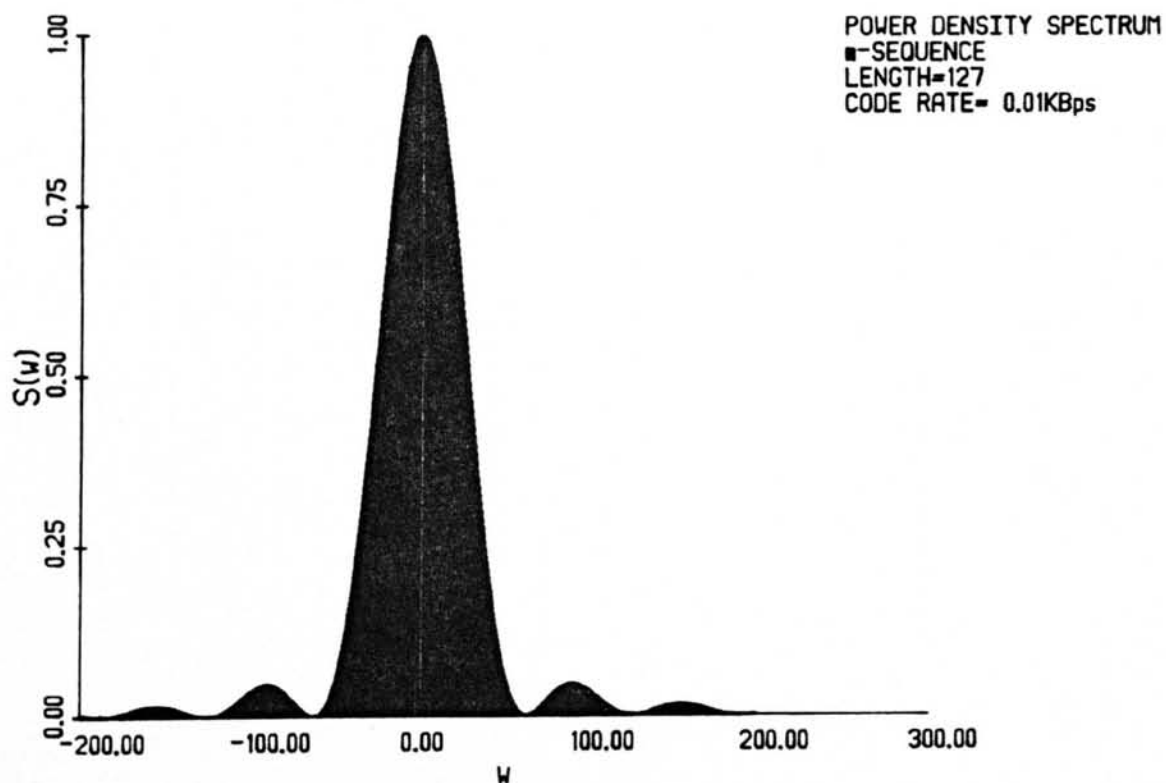
The spectrum has a $(\sin x / x)^2$ envelope. Figure (5.5) shows the power spectrum of the m-sequence waveform whose autocorrelation function was shown in Figure (5.4a). Observation of such a spectrum will show that it is a line spectrum with a line spacing equal to the code repetition rate, R_c/L . Because the pulse of shortest duration in the m-sequence is equal in length to the code sequence clock period, the spectrum will have a main lobe bandwidth such that its first nulls fall at the code bit rate. This is an interesting point in a direct sequence system, in which the transmission bandwidth is assumed to be equal to the bandwidth of the main lobe (i.e., is twice the code bit rate). From the one-zero distribution property, which is mentioned earlier, it is seen that the DC component is $\pm 1/L$ and the DC power is $1/L^2$.

5.3 Implementing the Feedback Shift Register on a Microprocessor

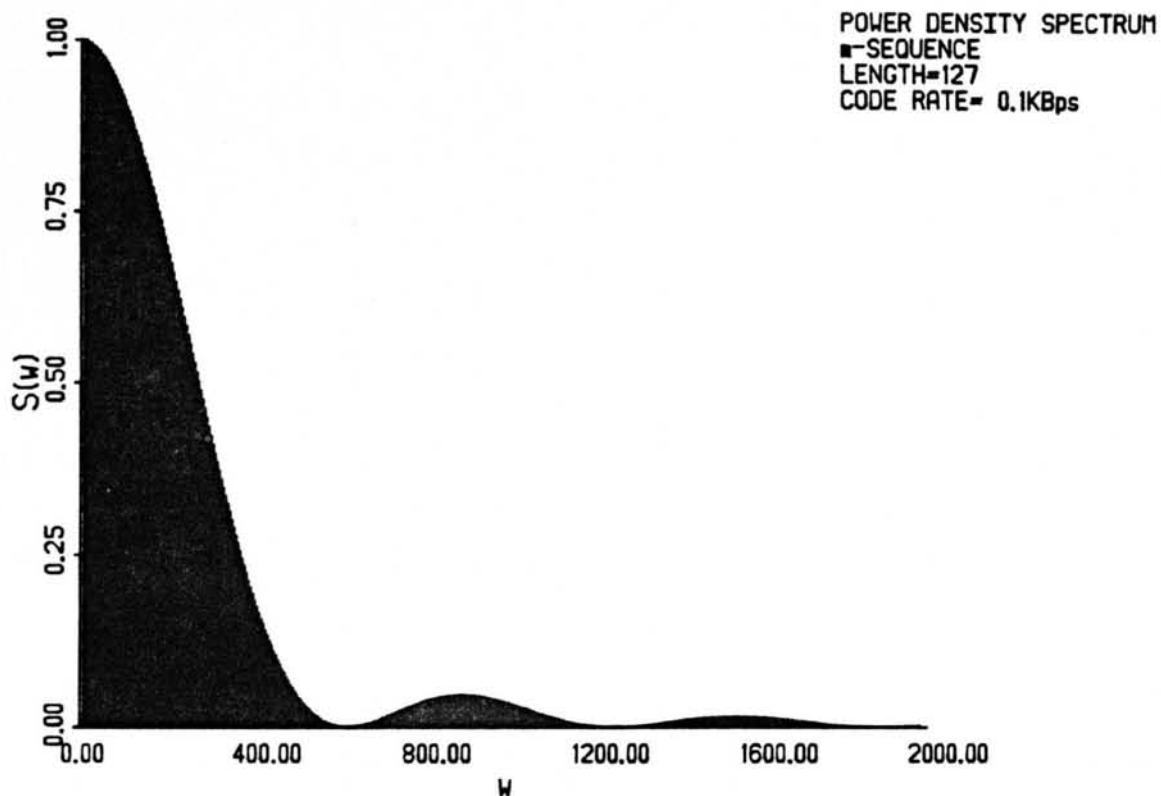
Although the linear congruential algorithms (75), (76) are the most successful approach for pseudonoise sequence generation in large computers, they are not well suited for use on microprocessors because they rely on multiplication of large integers. Instead, straightforward simulation of a linear feedback shift register is usually used to produce binary m-sequence using microprocessors. The term 'linear' here, means that only EXCLUSIVE-OR connections appear in the feedback logic (Figure (5.1)). M-sequences can be produced on the basis of equation (5.2) by simulating the hardware methods just discussed. A typical microcode program, when using a 2901 system, might



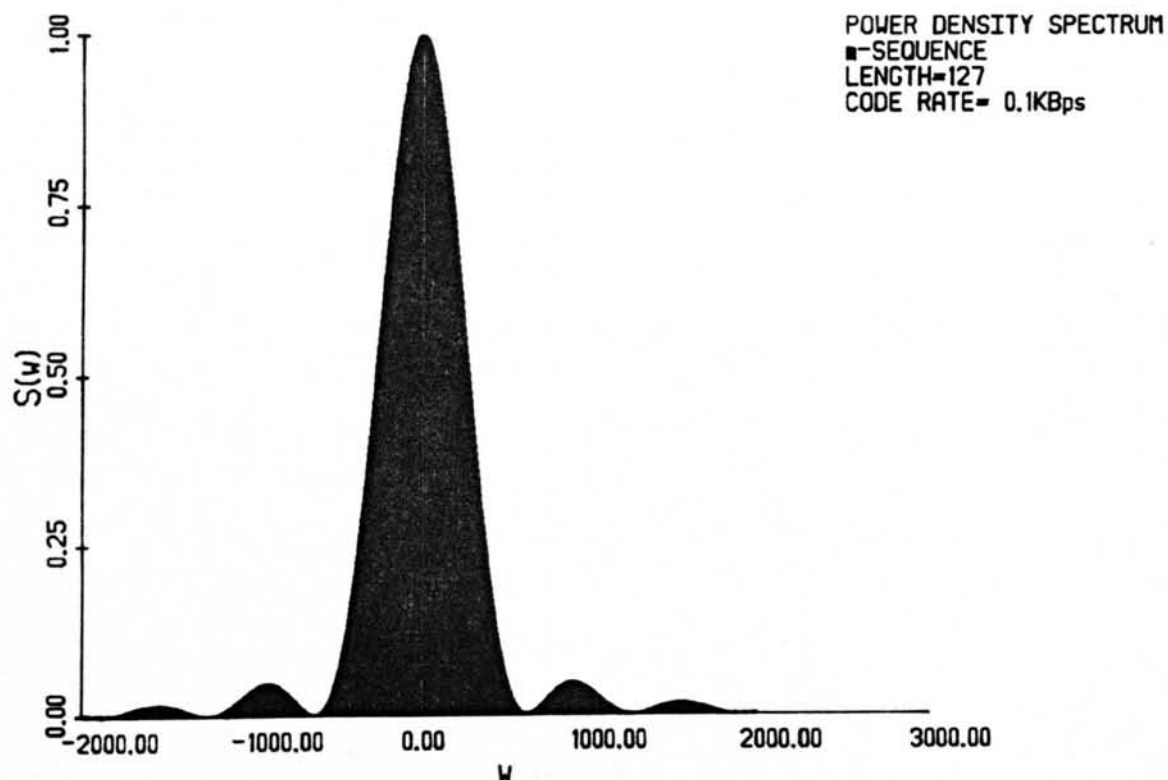
FIGURE(5.5a) m-SEQUENCE OF PERIOD 127 ONE-SIDED POWER DENSITY SPECTRUM.



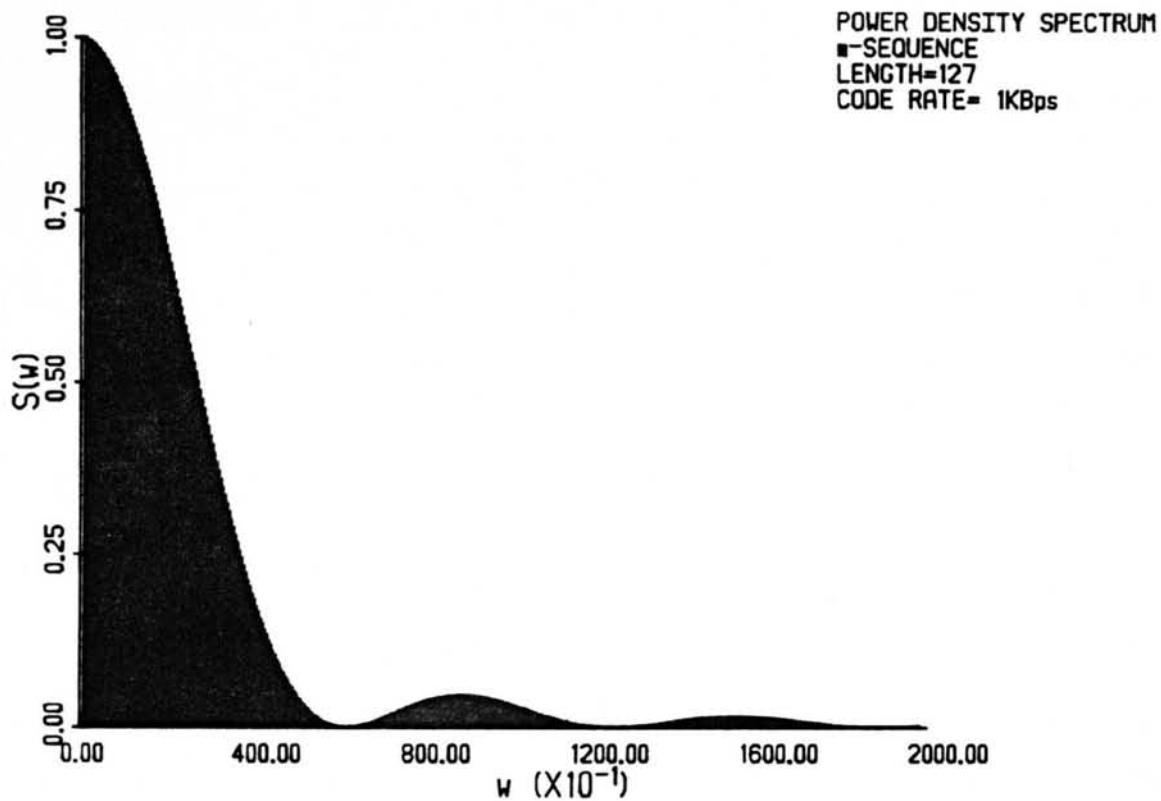
FIGURE(5.5b) m-SEQUENCE OF PERIOD 127 POWER DENSITY SPECTRUM



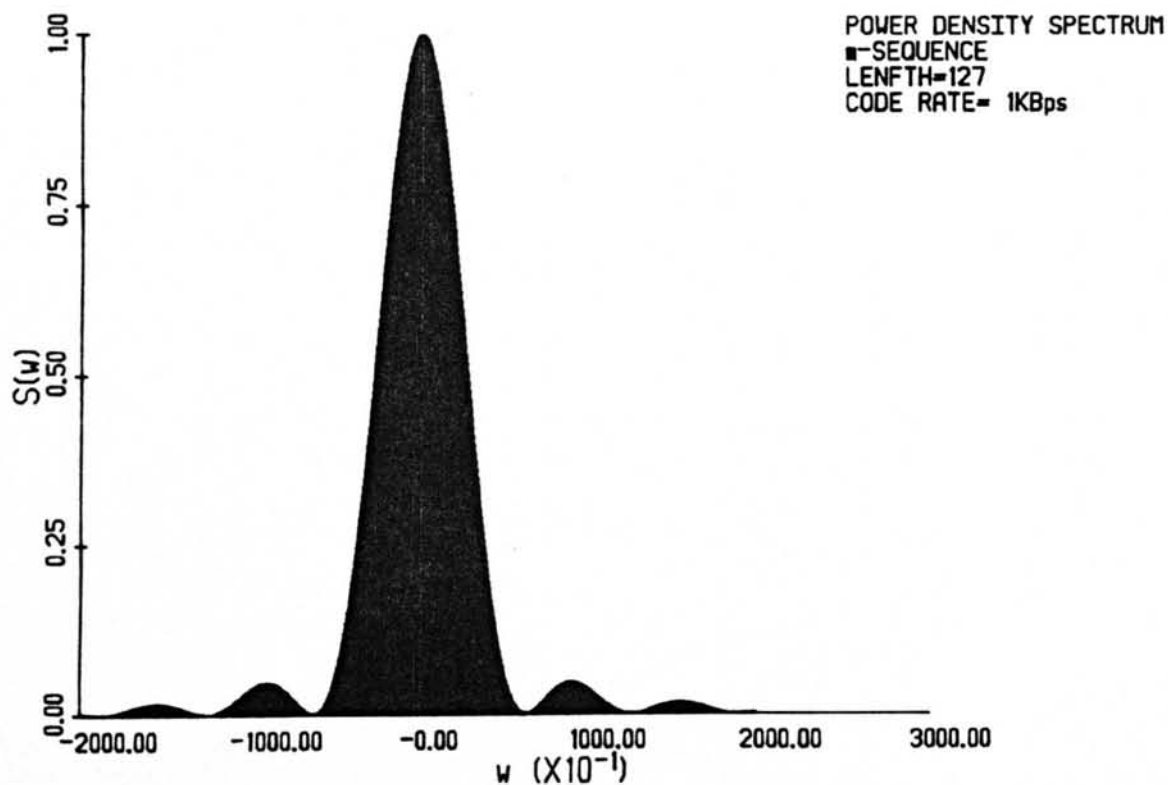
FIGURE(5.5a) m-SEQUENCE OF PERIOD 127 ONE-SIDED POWER DENSITY SPECTRUM.



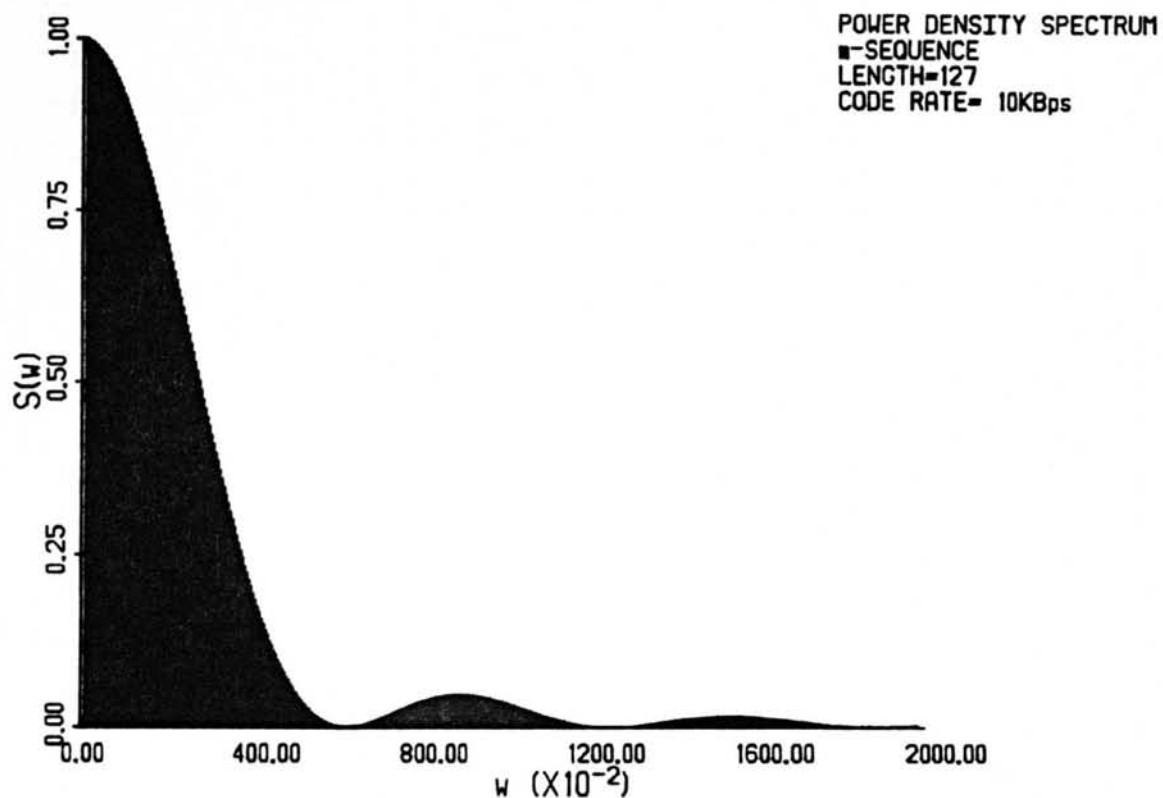
FIGURE(5.5b) m-SEQUENCE OF PERIOD 127 POWER DENSITY SPECTRUM.



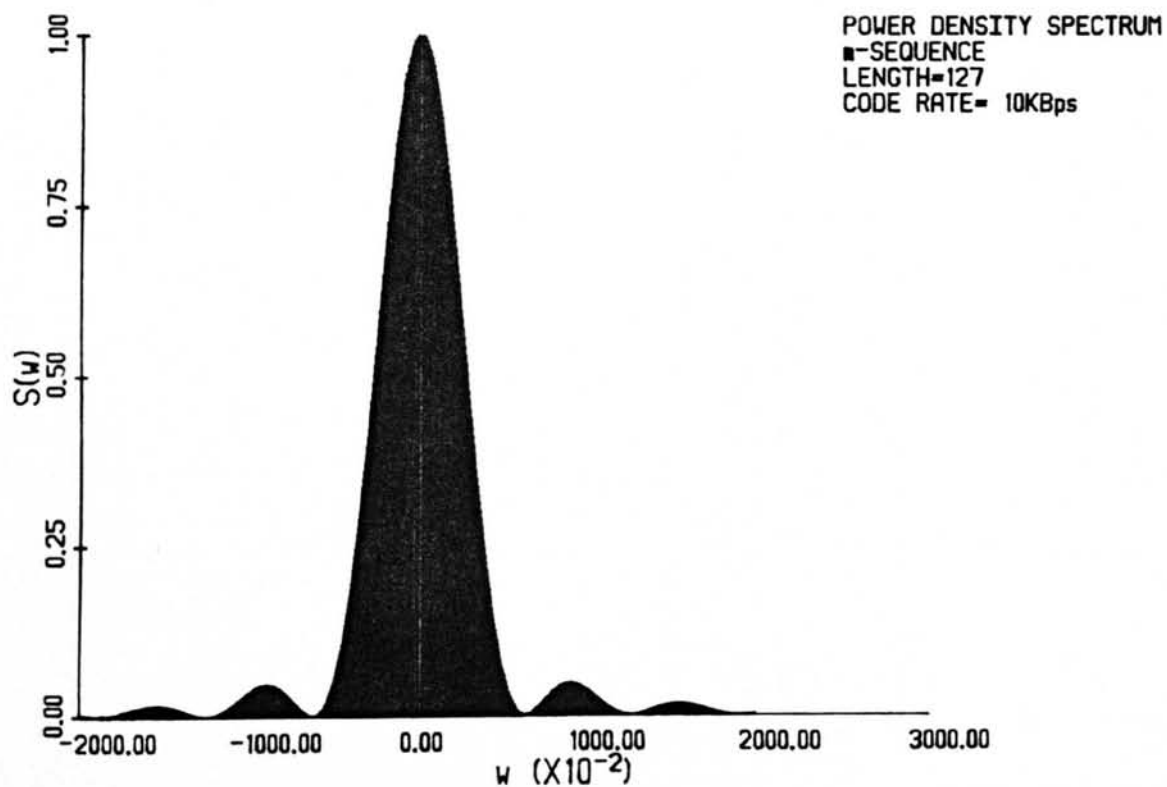
FIGURE(5.5a) m-SEQUENCE OF PERIOD 127 ONE-SIDED POWER DENSITY SPECTRUM.



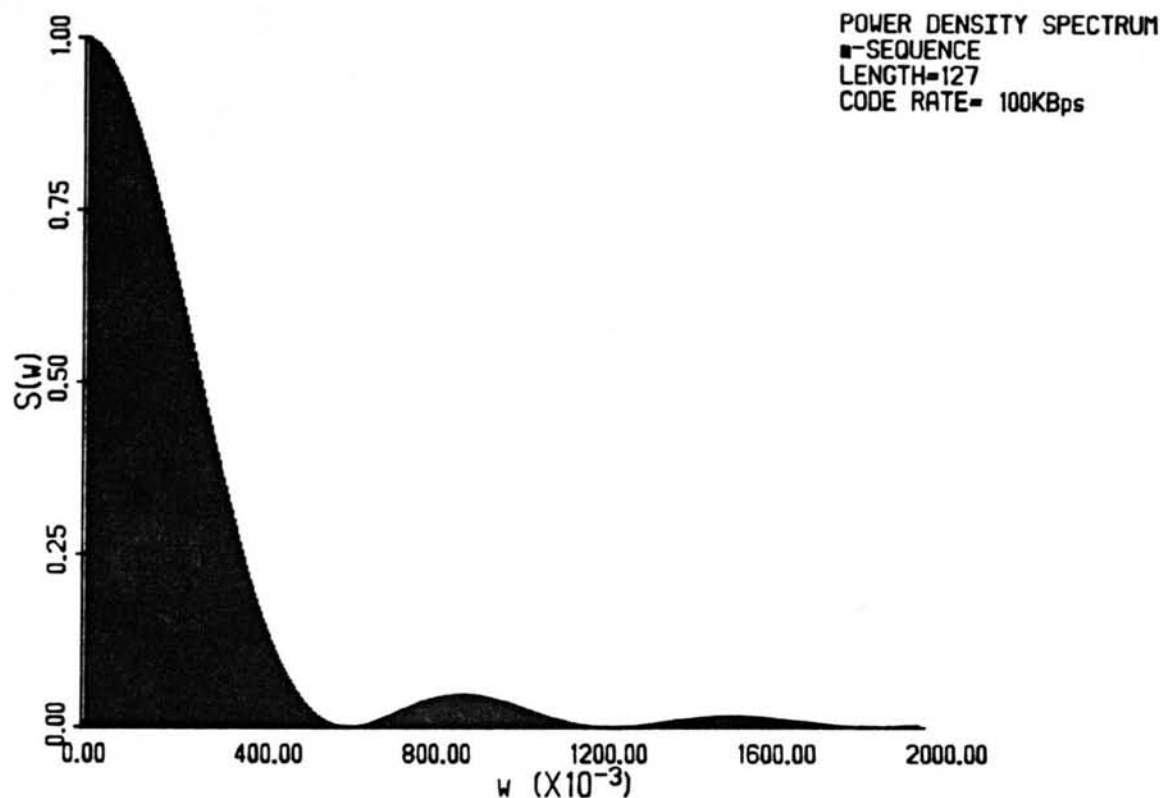
FIGURE(5.5b) m-SEQUENCE OF PERIOD 127 POWER DENSITY SPECTRUM.



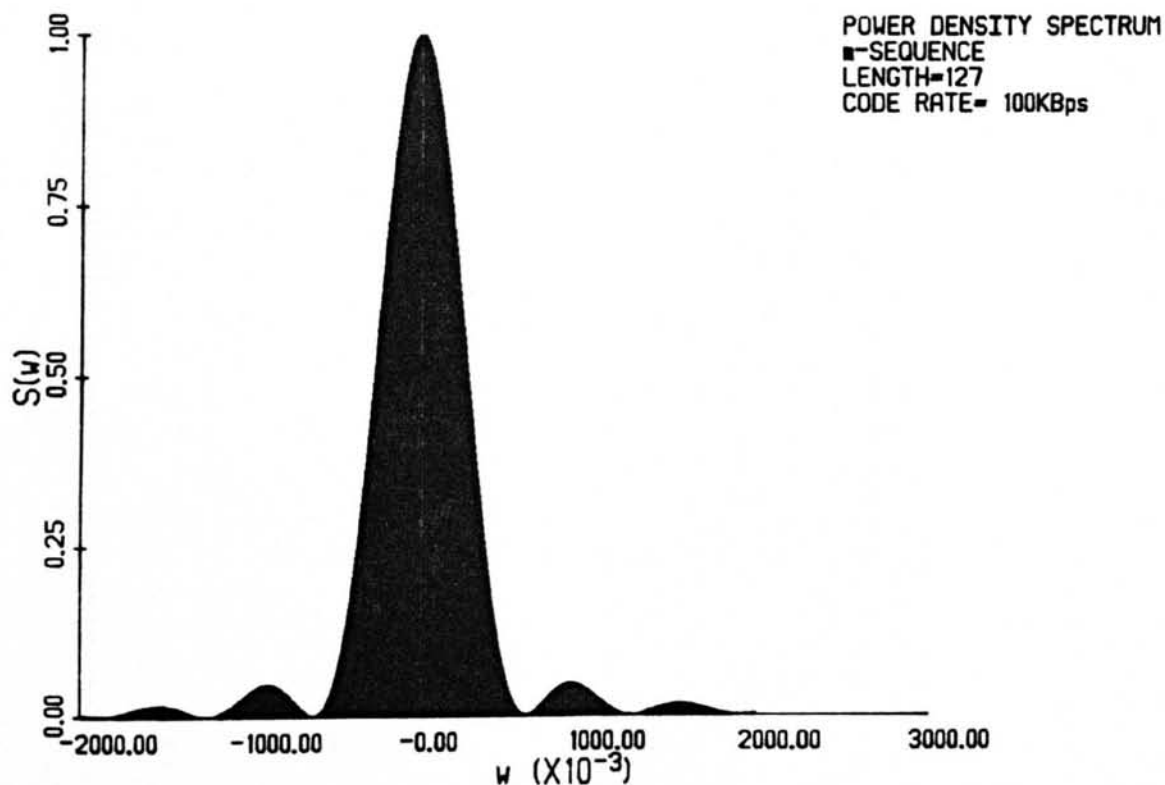
FIGURE(5.5a) m-SEQUENCE OF PERIOD 127 ONE-SIDED POWER DENSITY SPECTRUM.



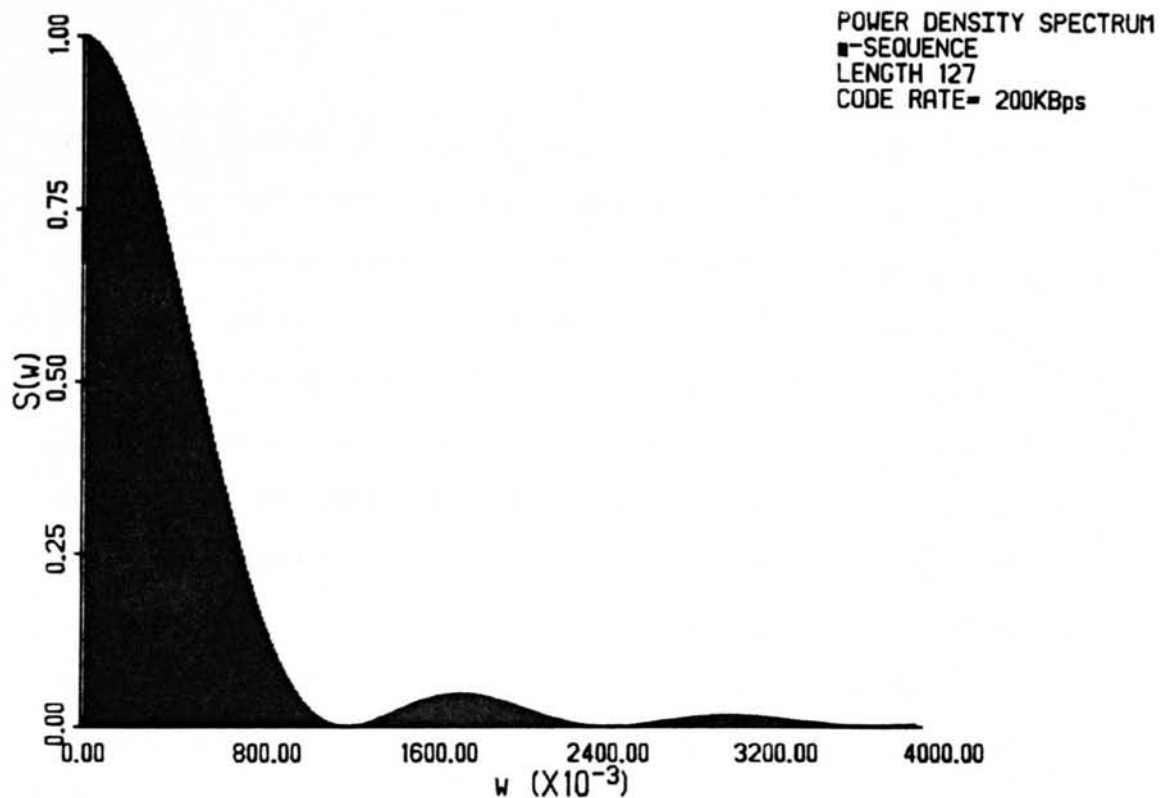
FIGURE(5.5b) m-SEQUENCE OF PERIOD 127 POWER DENSITY SPECTRUM.



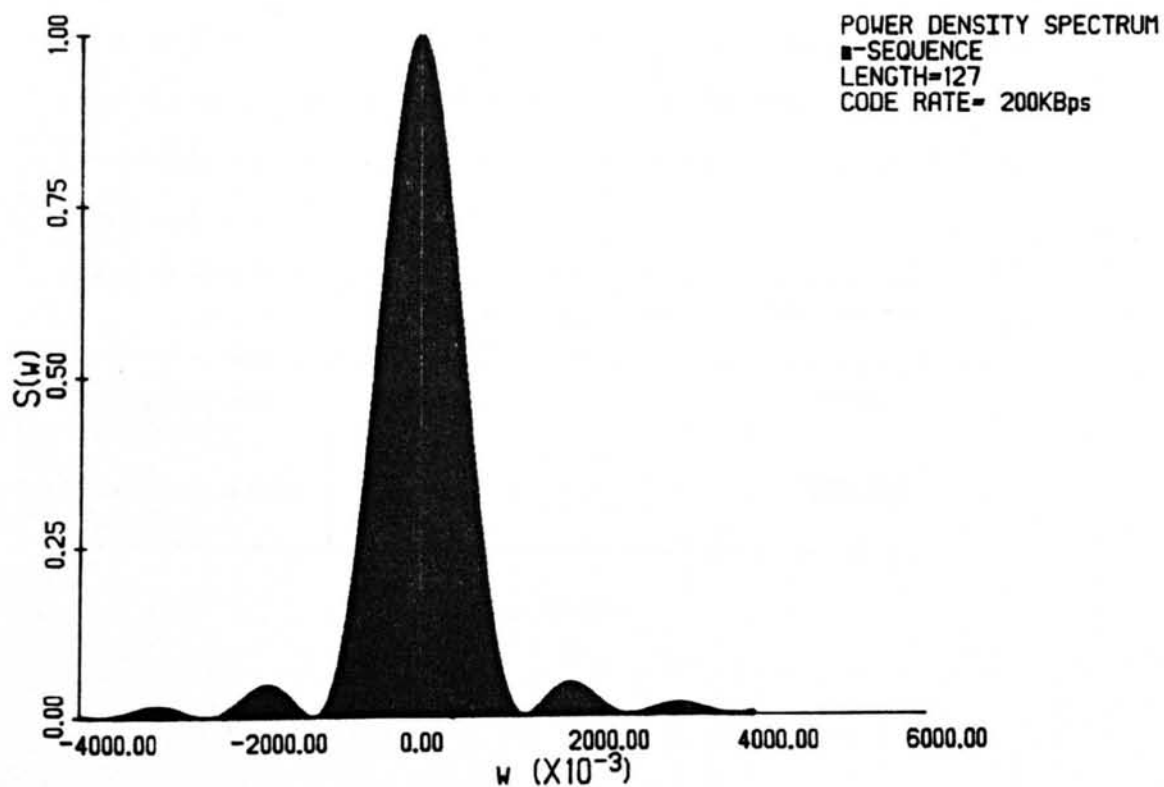
FIGURE(5.5a) m-SEQUENCE OF PERIOD 127 ONE-SIDED POWER DENSITY SPECTRUM.



FIGURE(5.5b) m-SEQUENCE OF PERIOD 127 POWER DENSITY SPECTRUM.



FIGURE(5.5a) m-SEQUENCE OF PERIOD 127 ONE-SIDED POWER DENSITY SPECTRUM.



FIGURE(5.5b) m-SEQUENCE OF PERIOD 127 POWER DENSITY SPECTRUM.

store the contents of the shift register in one or two work-space registers depending on the length of the sequence, or in successive memory words. When this is done, it is advantageous to assign a second register or memory word, containing the associated feedback coefficients, to each register storage location. When a certain register cell is connected to the EXCLUSIVE OR adder, the corresponding feedback bit in the feedback register is set to 1; otherwise, the feedback word bit contains 0. To calculate the feedback input to the shift register, feedback coefficients must be ANDed with the shift register content, and the number of binary ONEs in this result must be counted. For counting the number of ONEs, table look-up can be used. The feedback input will be 0, for an even number of 1s, otherwise 1. A successive bit isolation (masking) and EXCLUSIVE-OR operation may also be used to calculate the feedback input but it can be very time-consuming for multi-feedback tap organisations. The above approach can be compared as shown in table (5.6) by counting the number of microcycles per bit of the output sequence.

	number of microcycles per bit	code bit rate
successive bit isolation	15	200KBps
Hamming weight function	6	500KBps

Table (5.6)

A parallel implementation allows especially fast generation of m-sequences. Working in parallel, several bytes of fast PROM are loaded with a predefined and debugged code sequence and with a number of 1 shifted replicas of itself, where 1 is the word length of the processor, here, 8-bits. In general, $2^n - 1$ word locations of PROM are required to store all the possible phase patterns of a code sequence of length $2^n - 1$. Several factors make the parallel structure a good basis for effective software implementation. The PROM store can be arranged to match the word length of the processor and, therefore, every location produces 8-bits of the desired sequence in one clock cycle and shifting can be performed by incrementing a cyclic counter in a direct addressing mode. In this particular application, the parallel output structure is easy to manipulate in signal processing functions in which the code sequence takes part of it, such as, spreading, acquisition synchronisation, and despreading operations. It is important to note that with the parallel structure, increasing the processor word length leads to faster generators. Appendix B shows typical values (in Hex) for m-sequences of period 127, that were produced using the 2901 simulator, to be loaded in a PROM table look-up.

5.4 Sequence Inversion Keying (SIK) Modulation

Spread spectrum communications refers to a class of modulation methods by which the narrow-band information (data) is transmitted via a modulated signal having much greater bandwidth. Much of the literature refers to the direct sequence modulated signal that is produced by a multiphase phase shift keying (PSK)

modulation by a code sequence carrying the data (6), (7). The way in which the data is imbedded in a code sequence is called code modification or sequence inversion keying (SIK) modulation (77). This form of modulation means that we must change the code in such a way that the data is imbedded in it and can only be detected by an authorised terminal knowing the original code. In addition, the required code properties - good autocorrelation, low crosscorrelation, and the distinct spectrum - should be maintained. This process is achieved by digitising the data (principally voice) to be sent and modulo-2 adding it to the code sequence. This process has the effect of inverting the code each time a transition occurs in the data stream. The data transition rate could either be asynchronous with respect to the code sequence clock, or synchronous, in which case the number of data bits to be transmitted for each sequence period are restricted by the process gain and the system thresholding sources. One of these thresholds is the correlation detector threshold, that will be discussed later. For purposes of simplicity, in this project, we consider the transmission process as completely binary. SIK may be performed in the 2901 system based transmitter as follows; data bits are synchronised by recognising the all 1's state of the m-sequence generator and starting a data bit at that time. A cyclic counter is counted to determine the start of subsequent data bits. A typical example was implemented in which a 7-stage m-sequence generator produces a sequence whose length is 127 (Figure (5.3a)) bits. In this example, the data bit rate was chosen to be the repetition rate of the m-sequence. Because the length of the sequence is a prime number, the data bit is divided

into 16 data bytes where 15 bytes are of complete length (8-bits) and the 16th byte was masked in order to isolate the required number of bits. In this case the cyclic counter is reset to zero after each sequence period. Thus the transmission of alternate 0's and 1's of data is provided. Each bit of data, in this example, can take 48-64 microcycles in order to be transmitted.

The practical advantage of using asynchronous data is simplicity of the implemented interface between the data source, which may be a front end-processor, and the transmitter. Asynchronous data bits are random in length, this makes the synchronisation problem much more difficult.

5.5 Synchronisation

Despreading or demodulation of the spectrum-spreading modulation requires good synchronisation between the coded signal arriving at the receiver and the receiver's reference code. Synchronisation, here, means that the received signal and the reference code are accurately timed in both their code phase and their rate of bit generation, and should remain so. Changes in code phase and/or code rate are due to propagation path length changes, Doppler frequency shift, and nonaccurate frequency sources in both the transmitter and receiver. Good synchronisation requires that code phase alignment of the two codes within approximately one code bit (chip) must be achieved and maintained. The synchronisation process is usually regarded as consisting of two parts; coarse synchronisation (which is also called initial synchronisation or acquisition) and fine synchronisation (tracking). Initial acquisition may be defined

as the process of: firstly, adjusting the relative phase and rate of the reference code and the received signal to within the pull-in range of the tracking. Secondly, activating the tracking phase. Tracking is the process of maintaining a synchronised condition and initiating the acquisition process if tracking subsequently fails. Those parts of the receiver concerned with acquisition and with tracking can be mostly digital, excluding those systems which use direct transmission of very high rate code sequences (multiple Mbps) for which analogue correlation techniques (by using surface acoustic wave (SAW) (9) and charge coupled devices (CCD) filters (78)) are necessary.

5.5.1 Initial Acquisition Techniques

When the communication link is first established or after a loss of contact, an acquisition process must take place. In this process, the receiver sequence generator is brought into synchronism with the incoming sequence and the tracking loop is locked in. A number of methods have been implemented for acquiring pseudonoise signals which can be used in the receivers of spread spectrum systems. The type of acquisition method to be chosen should make full use of the attractive correlation properties of the m-sequence in order to reduce the delay errors between local and received code sequences. Ward (16), and others have described a technique, known as sequential estimation, which depends on estimating n (the number of shift-register stages) sequential bits of the incoming signal and loading them into the receiver shift register in order to obtain an estimate of the present state of the input. Through correlating the received sequence with that generated locally for an examination period

T_e , a decision is made as whether the correct estimate has been loaded or not. In case of correct decision, acquisition occurs and a tracking mode is entered. In case of an erroneous estimate, a new estimate is made and the procedure is repeated. If a data ONE is being transmitted or if a data transition occurs during the n -bit estimate, an incorrect estimate will be obtained. These effects will be more detrimental when higher data-bit rates are being used. In the second method, which is usually referred to as the sequential detection method (79), a precalculated bias is added to the correlated code and integrated for a variable length of time until the threshold is exceeded, indicating only noise is present (local code out of phase). This method is especially appealing when a strict acquisition time requirement is imposed. These methods require a significant amount of hardware logic circuits and the data processing requirements for these methods exceed the capability of a 2901 microprocessor unit. Alternatively, 'sliding' correlation process (which is also called the serial search process (80)), in which the phase of the reference code is slipping while cross-correlating it against the received signal until the cross-correlation output rises to a value which exceeds a certain threshold and the sweep is halted, is almost always employed.

One of the most useful techniques, which makes use of the serial search process, employs a synchronisation preamble to be sent at the beginning of each transmission. This preamble is a special code sequence (72), which can be short (the only difference between them and the code sequences used after

acquisition is the length) to allow a search through all possible code positions in a reasonable time. The disadvantages of the preamble synchronisation method is that, its relatively short sequence length tends to be more vulnerable to false correlations.

The serial search method is very accurate but it is slow because it involves correlation over a complete period of the sequence, and the repetition of this for successive time-shifts until the peak is found. Fortunately, correlation over a partial-period (segment) of the sequence is adequate (81).

The receiving system, in searching for synchronisation, operates its code sequence generator at a rate slightly faster (or slower) than the transmitter's code generator such that the receiver code slips past the received signal. One way of achieving this using a microprocessor is by interfacing a microprocessor system to a voltage-controlled oscillator (VCO) and sequence generator in hardware. In that case a DAC is needed in order to send the chosen value of the voltage to the VCO which causes a frequency offset (search rate) between the local clock rate and the clock rate of the transmitter. This method is awkward and a compromise is needed between a small frequency offset which causes a long average-search time, and a large frequency offset, for which the cross-correlation peak is small and liable to be missed (false dismissal) unless a low threshold is set, and as will be seen shortly leads to a false acquisition (false alarm) in the presence of noise (80). The drawbacks are also that it depends on the machine to be used which, in some

cases, is too slow.

In this case, the whole system was implemented within the microprocessor system which was also used to drive the clock at both the transmitter and receiver, as will be seen in the next chapter.

5.5.2 Correlation Process

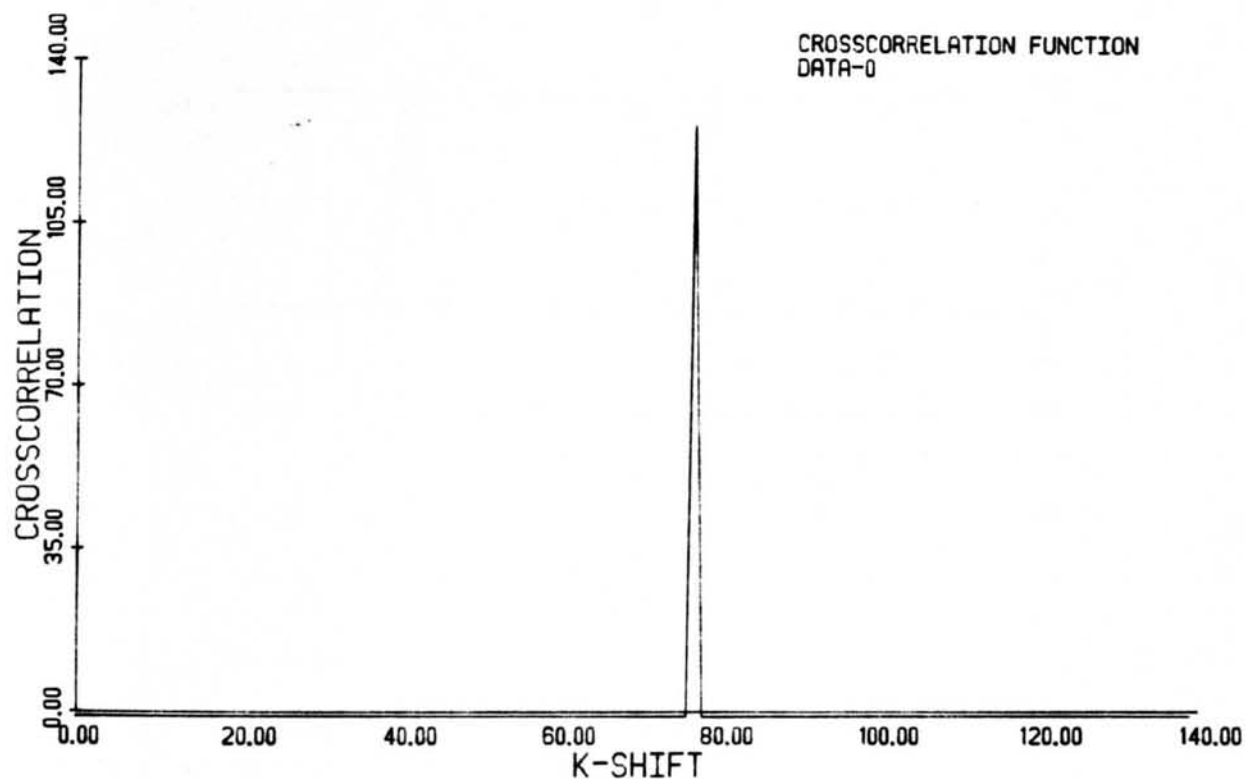
The performance of the 2901 system in achieving the acquisition process depends strongly on the partial crosscorrelation function

$$c_{as}^M(i, \tau) = \sum_{n=0}^{M-1} a_{i+n} s_{i+n+\tau} \quad (5.12)$$

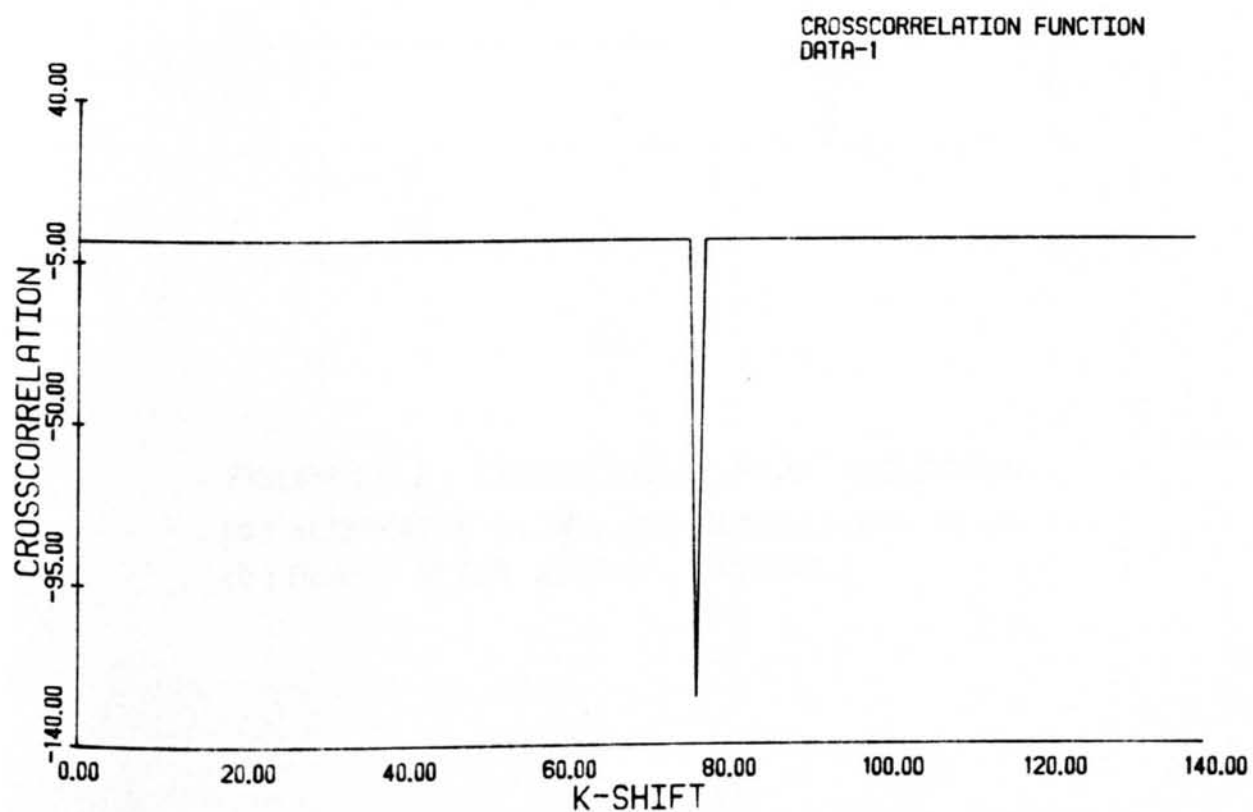
which represents the correlation of M symbols $a_i, a_{i+1}, \dots, a_{i+M-1}$ of the receiver's replica with the M symbols $s_{i+\tau}, s_{i+1+\tau}, \dots, s_{i+M-1+\tau}$ of the received sequence, such that $M \leq L$. In the case $M=L$ the quantity $c_{as}^M(i, \tau)$ is independent of the initial i , and correlation analysis becomes a relatively simple deterministic problem. A theoretical analysis of the relation between L and M as a function of i , and $c_{as}^M(i, \tau)$ is not available to our knowledge. Instead, many search trials were made using the 2901 simulator in order to choose a reasonable value for M . Notice that $c_{as}^M(i, \tau)$ is simply a comparison of the i -th binary weight bits of the received signals $s(t, \tau)$ and the replica $a(t)$. This comparison can be performed by computing the number of agreement bits between a_n and $s_{n+\tau}$. From that point of view the 2901 based correlator system is effective because it can be expanded to accommodate variations in the sequence length L . The 2901 system

computes the number of agreement bits between two 8-bit binary sequences in 3 microcycles, e.g., a 127-symbol crosscorrelation takes a total of 48 microcycles (16 u.sec) to be computed. The phase slipping process is achieved either by phase shifting the receiver's code periodically by 1 chip increments each time and carrying out the comparison with a stored sample of the received sequence, or by matching the incoming sequence with a fixed segment ($\leq L$) of the receiver's code until the proper point of synchronisation is reached. In this case, the time spent at peak correlation is minimum but the threshold, as will be seen shortly, is higher.

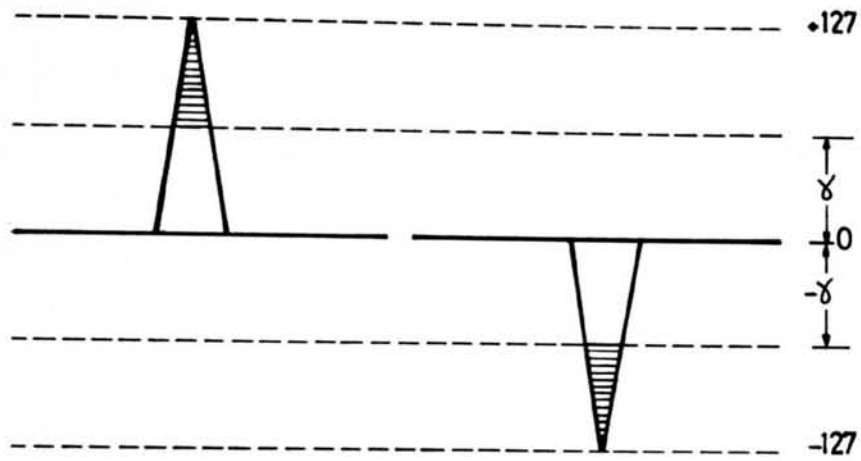
Recognising the peak in the correlation function is an inherent part of the acquisition process after which the receiver starts its local code tracking before the received and local codes drifts apart. Because the low rate binary data, which is modulating the subcarrier, is unknown, the peak in the correlation function may be maximum positive or maximum negative depending upon the phase of the received signal as shown in Figure (5.7). Therefore it is necessary to test whether the peak is above a positive threshold or below a negative threshold. An accurate but a very long computing time search method was implemented using the 2901 simulator in order to choose the optimal values required to be set as a threshold. Figure (5.8a) provides a description of the method used to detect the correlation peak in the microprogram design. The correlation values are arranged without truncation such that the peak magnitude lies in the range of the 8-bit two's complement integer



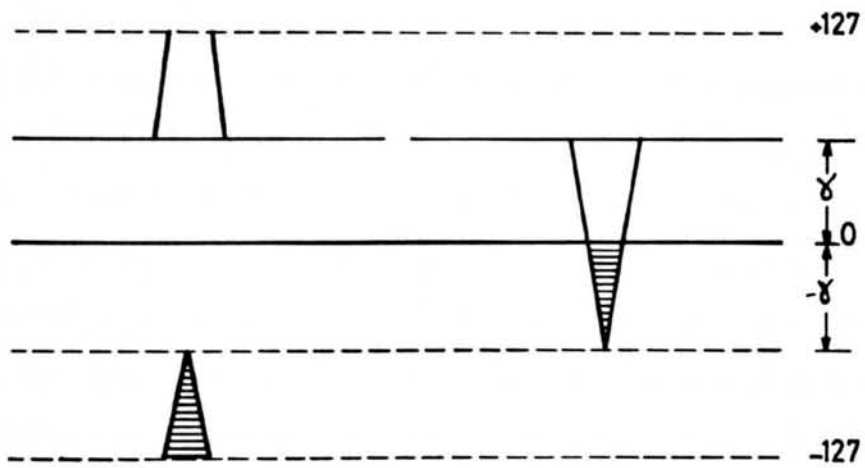
FIGURE(5.7a) EFFECT OF DATA POLARITY ON CORRELATION PEAK (DATA-0).



FIGURE(5.7b) EFFECT OF DATA POLARITY ON CORRELATION PEAK (DATA-1).



(a)



(b)

FIGURE (5.8) CORRELATION PEAK DETECTION
 (a) ALTERNATIVE VALUES FOR CORRELATION PEAK
 (b) PEAKS AFTER ADDING THRESHOLD.

and the threshold is added to the correlation values. The values will change as a result of the overflow properties of two's complement arithmetic as shown in Figure (5.8b). If the peak exceeds the threshold, in either direction, the result is negative. This requires only one microinstruction when it coded directly into 2901 system. Note that the lower the setting of the threshold, γ , the more assurance that a peak will be recognised. Note also, however, that lowering the threshold increases the noise lying above threshold, which increases the false alarm rate as shown in Figure (5.9).

5.6 Tracking

As a result of the acquisition process the receiver's code and the sequence embedded in the incoming signal are exactly matched in time. In order to maintain that synchronism, the receiver must cause its own code bit rate to match the incoming code bit rate (usually within one chip). In principle, a dithering loop (82), and delay-lock loop (83), (84) are often used for doing this, in which case, the output of the acquisition correlator will indicate the loss of synchronisation when the peak falls below the threshold. However, in an implementation such as this and because of the serial nature of the signal processing of the microprocessor implementation, it would be impractically slow to operate both the acquisition and tracking processes simultaneously.

The delay-lock loop uses an actuating error signal derived from received and local sequences to control the receiver's code bit rate. Operation of the delay-lock loop was first discussed

by Spilker (83). A survey of the basic properties and main parts of the delay-lock loop that are useful for the purpose of this thesis will be given in this section.

5.6.1 Delay-Lock Loop Correlator

Figure (5.10) shows the basic block diagram of the delay-lock loop correlator, which will be assumed to be in the locked-on situation. The backbone of the circuit is a local n -stage feedback shift register which generates time-displaced versions of a binary m -sequence $a(t)$. It is convenient to denote sequences with a delay and advance of kT by a_{-k} and a_{+k} respectively, e.g.

$$a_{-k} = a(t-kT), \quad a_{+k} = a(t+kT) \quad \text{and} \quad a_0 = a(t)$$

Each of the versions a_{-1} and a_{+1} which have amplitudes ± 1 is correlated with the input signal

$$r(t) = s(t+u) + n(t)$$

where u is the delay-error. This terminology stems from the use of the delay-lock loop in tracking and ranging systems where the delay between a transmitted and a received sequence is a measure of the distance between target and transmitter/receiver. The difference between the correlator output signals is obtained by a subtraction device which, together with the multiplier and summation forms the crosscorrelation network. Using the shift-and-multiply properties of m -sequences, it can easily be shown that the contribution of these sequences to the long time summation output of the crosscorrelation network is proportional to

$$D_{2T}(u) = R_a(u+T) - R_a(u-T) \quad (5.13)$$

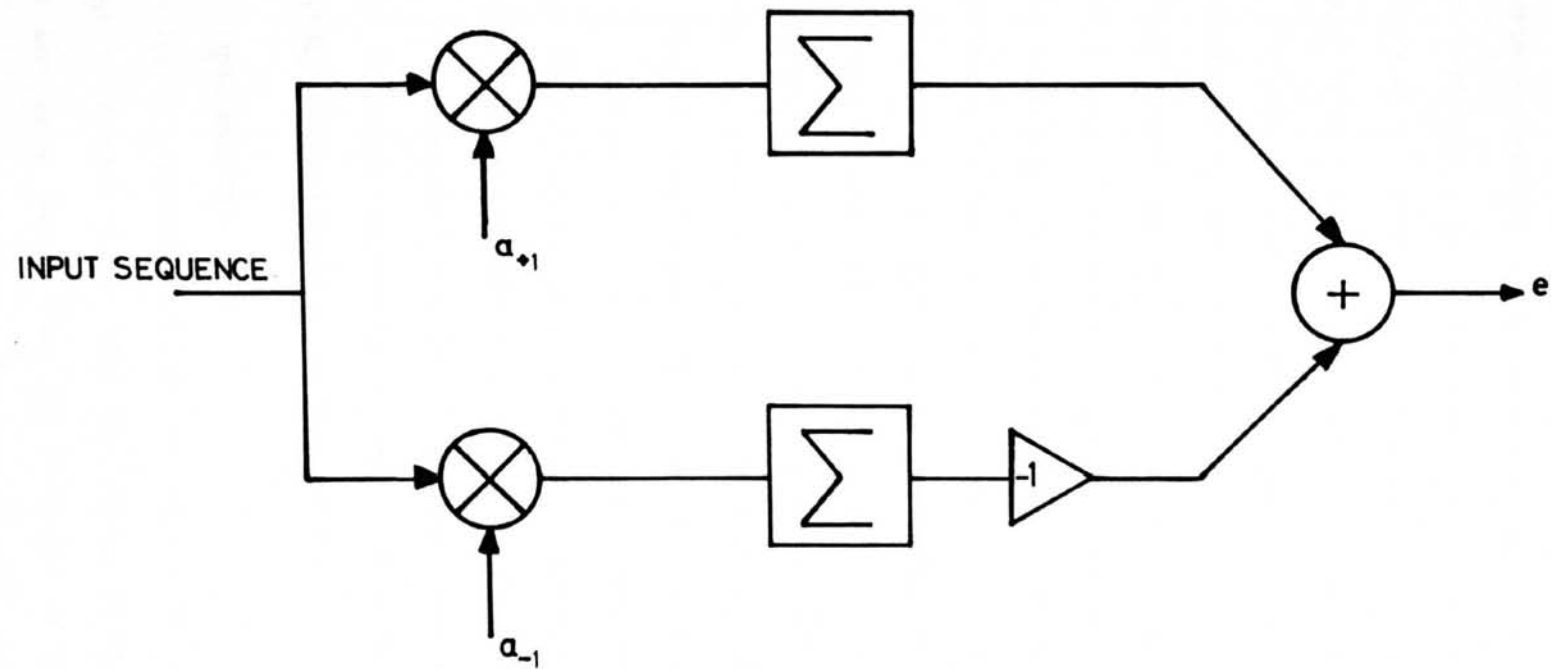


FIGURE (5.10) BASIC DIAGRAM OF DELAY-LOCK LOOP CORRELATOR.

where $R_a(u)$ is the autocorrelation function of the pseudonoise sequence in question which has a base width of $2T$, and $D_{2T}(u)$ is an N-shaped error signal made up of two copies of this autocorrelation function, with a relative shift of $2T$, one being inverted with respect to the other as shown in Figure (5.11). Thus for $|u| < 2T$, the function $D_{2T}(u)$ provides a discriminator characteristic or error-signal to control a VCO to maintain the alignment between the two sequences by minimising u . In this way a closed loop is obtained. If properly designed the loop tends to the locked-on state, for which $|u| \leq T$. If, at any time, the delay-error u exceeds one bit time ($|u| > T$) the loop loses lock, and the acquisition process has to be reinitiated. In practice, the period of sequences used is normally long (for example, Ward's ranging system (77) uses 15-stage shift register which produces a sequence whose length is 32767 tracking bits). The time elapsed before reacquisition may also be long, during which time a substantial block of transmitted data may be lost. A variety of methods by which the error curve can be widened to accommodate a displacement greater than $\pm T$ without losing lock have been discussed (85), (86). Although these methods have a higher threshold, permitting a larger offset to be used, the probability of false dismissal may be increased.

5.6.2 Implementation

Direct implementation of the delay-lock loop correlator circuit of Figure (5.10) with SIK having equally probable data zeros and ones means that the N-shaped error curve (Figure (5.11)) may or may not be inverted, depending upon whether the

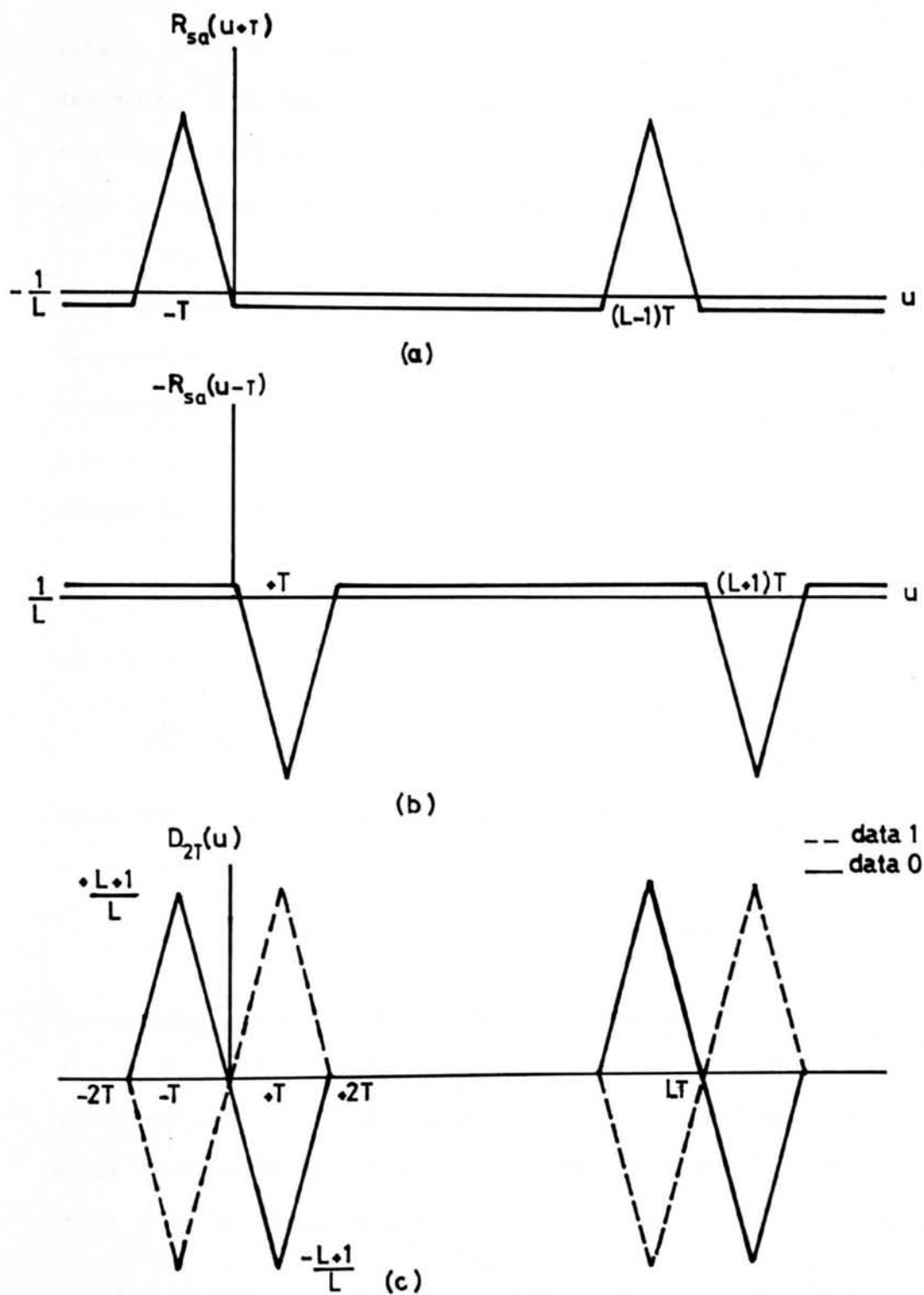


FIGURE (5.11) (a) AND (b) AUTOCORRELATION FUNCTION OF BINARY SEQUENCE (PN)
(c) DELAY-LOCK DISCRIMINATOR CHARACTERISTIC.

data is 1 or 0. One way to overcome this which has been described (77), (84) involves duplicating the delay-lock loop correlator, and rectifying and combining their outputs together with an appropriate bias such that either correlation or anticorrelation at a multiplier produces a negative feedback loop. In this case, by adding (modulo-2) the data obtained after despreading to the sequences s_{-1} and s_{+1} prior to the loop correlator, this dependence of the error curve on the data polarity is avoided, this may be recognised as follows:

Equation (5.13) can be rewritten as

$$s(t) = m(t+u) a(t+u) + n(t) \quad (5.14)$$

and this is correlated with

$$\hat{m}(t) a_{+1} - \hat{m}(t) a_{-1}$$

where $m(t)$ represents the estimated low-rate data. As a result of the term

$$m(t+u) \hat{m}(t) = 1$$

the correlator outputs are unaffected by the modulation. This method is a straightforward implementation using microprocessors; occasional errors in data despreading due to data-transmission errors and interference can be tolerated because of the smoothing effect of the correlation process.

The tracking correlator was implemented using the 2901 microprocessor system by generating the local sequences, a_{+1} and a_{-1} , using PROM and replacing the multipliers by EXCLUSIVE-OR operations.

5.7 Conclusion

In this chapter, direct sequence system configurations for data transmission have been described. One method to data modulate a pseudonoise code sequence is SIK. The only restriction on the data patterns transmitted, in case of short code lengths, is that the word length must be equal to the period of the m-sequence. The accurate performance of the acquisition and tracking system which has been discussed is due to the excellent properties of the m-sequence, based on the use of a delay-lock loop correlator. The use of a bipolar bit-slice microprocessor system to generate the pseudonoise sequence required as a subcarrier and to control the acquisition and tracking modes is effective because it can be expanded to accommodate variations in the sequence length. In addition, it offers the advantage of being able to define the parts of the spread spectrum receiver by software.

CHAPTER 6

Transmitter and Receiver Design

6.1 Introduction

The advances in digital electronics have come to the point of making circuitry and systems reasonably small, reliable, and inexpensive so as to enable practical implementations of spread spectrum techniques for the transmission of digital information. The suitability of microprocessor systems for the implementation of digital correlation and the synchronisation of spread spectrum receivers has been demonstrated in chapters 2 and 5. The extent to which microprocessors may be used in the implementation has been investigated in chapter 3 and 4, because of the flexibility that should follow from defining the procedures by software. Relatively high throughput requirements (processing of greater than 1 kbits/sec digital information) and several special function requirements dictated the selection of a bit-slice microprocessor for use in this case (other than general-purpose microprocessors).

This chapter describes how the 2901 microprocessor can be applied to perform the signal processing for the spreading, synchronising, and despreading of the transmitter and the receiver in real time. In order to achieve this it is necessary to adapt the way of executing the various operations to the computational capabilities of the microprocessor. This has been done by the derivation of suitable algorithms which specify the different functions that have to be performed. Both the

transmitter and receiver designs are based on the 2901 system.

6.2 Transmitter

The complete microprocessor configuration of the transmitter is represented in Figure (6.1). This hardware is generally divided into five parts, namely, a 2901 microprocessor system controlled by microprograms, external memories, programmable divide by n counter, buffer memory and a test box. The 2901 system which was described in chapter 2 and 3, is the core of this transmitter. The microprocessor includes eight 1/2 kbyte bipolar PROMs containing the transmitter software. The external memories is composed of PROMs. Two IC's PROM store the code sequence samples. These are addressed by the microprocessor through the data bus and the stored values are fed out to the same bus. The programmable divider uses the main clock of the microprocessor to generate the clock required to shift the data to be transmitted serially out from the buffer memory, FIFO (see Figure (3.10)). It provides square pulses at rates of $f_c/256 - f_c$ ($f_c = 3\text{MHz}$) as selected by the test box switches. Finally a few latches, two bus transceivers and a test box complete the system design.

The tasks performed by the transmitter fall into three broad categories:

- (1) data acquisition
- (2) spreading by SIK modulation
- (3) transmitting data.

It was found that a single 2901 system unit was not able to perform the RF carrier modulation and all of the required tasks

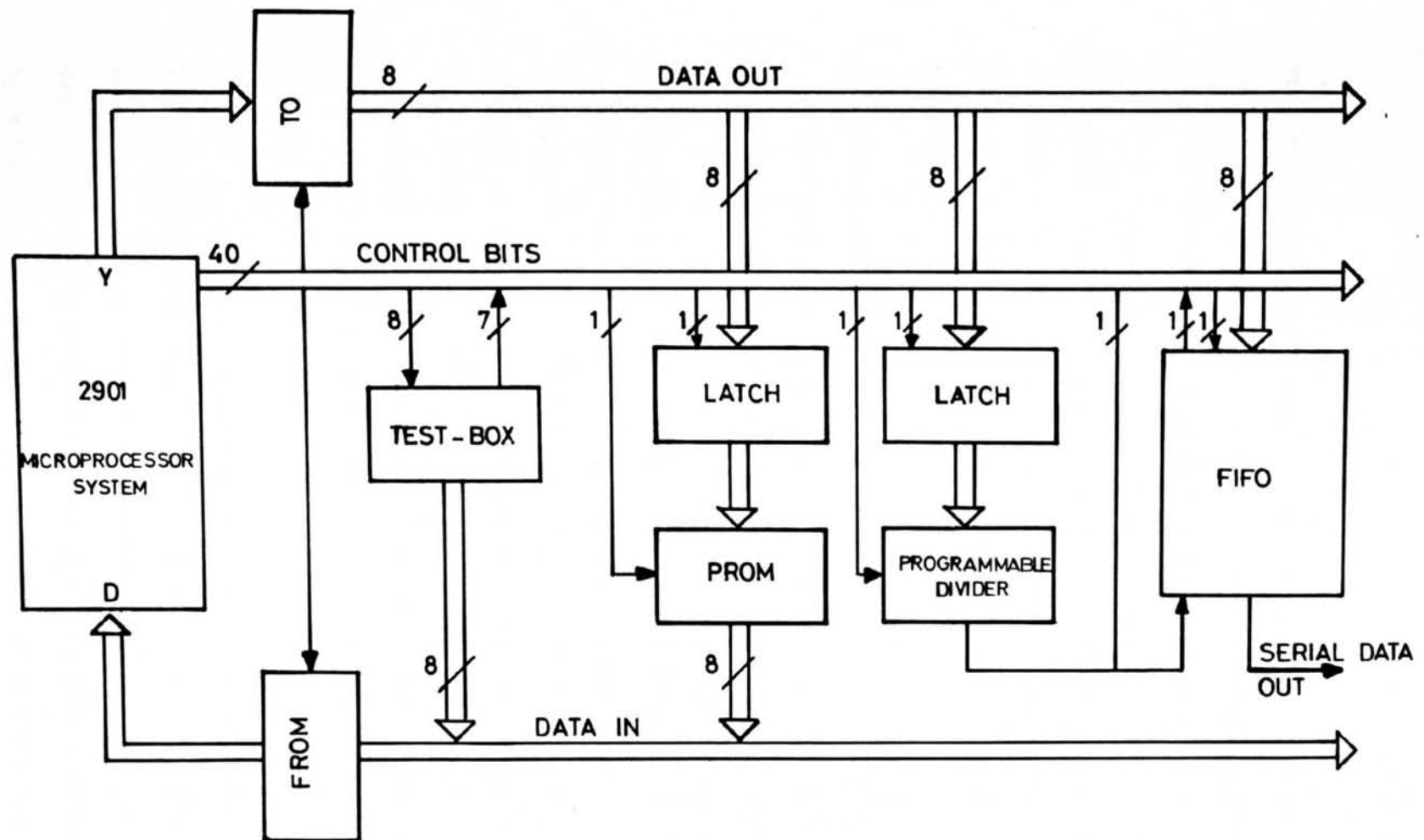


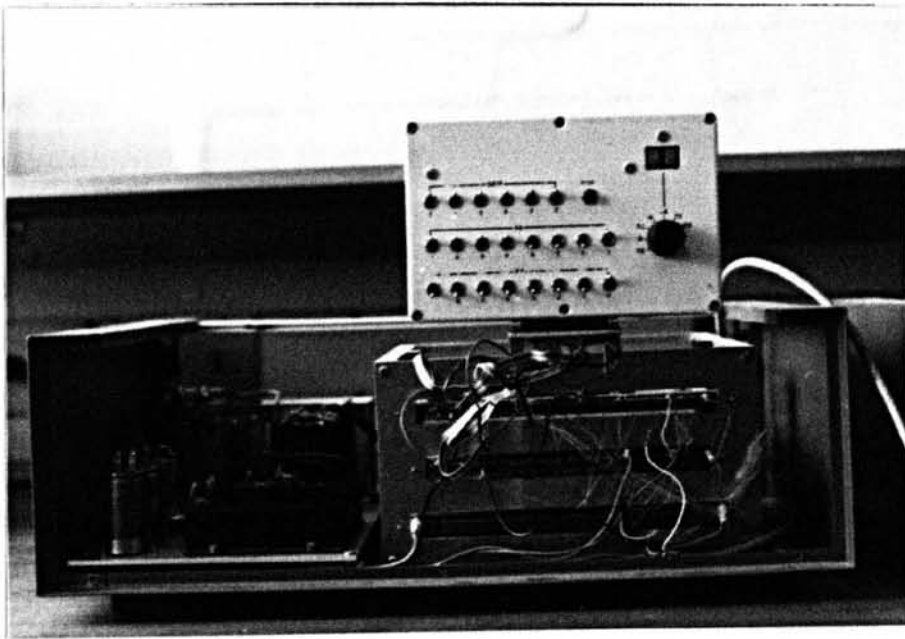
FIGURE (6.1) 2901 MICROPROCESSOR CONFIGURATION FOR THE TRANSMITTER.

mentioned above simultaneously in the available time. One of the main reasons for this was that the RF carrier modulation waveform must be generated continuously, while simultaneously spreading incoming data. In view of this, it was assumed that the RF modulation would be achieved by some other means and the remaining tasks were performed by this system without a separate hardware multiplier. Figure (6.2) shows a side view of the transmitter.

6.2.1 Data acquisition

The facility to interface the 2901 system with a host computer or an external data source was available, and consisted of a set of latches (plus circuits to actually carry out the transfers, control flip-flops, and interrupt handling circuitry) to enable the assembly of 16-bit data and address words under microprogram control. The address word latches may be counters so that the address can be incremented for each new data word. This facility was capable of accommodating both asynchronous and synchronous data (see section 5.4) by which the transmitted signal spectrum is made to be dependent only on the clock rate of the code generator.

In principle the baseband data to be communicated is digital, coming either from a data terminal or digitised voice or video. In order to measure the maximum rate of the error-free data to be transmitted, the binary data was, firstly, simulated by software before it was modulo-2 added with a code sequence just as any other binary data stream would be. The data bit length was chosen to be equal to a multiple integer of the length



FIGURE(6.2) TRANSMITTER HARDWARE (SIDE).

of the sequence in order to overcome the correlation loss problem. Therefore, it was felt, at this stage, that error control coding was not mandatory.

6.2.2 FIFO on transmit

The input/output buffer memory, FIFO, interconnection and operation has been described in chapter 3 which provides the means for the microprocessor system to interact with the communication link. The microprocessor loads the FIFO with a few bytes of modulated data which is clocked into the FIFO from the 8-bit bus by activating "data transfer" (T6) and checking the state of "data request" (SD) to ensure that the FIFO is not full. The transmit signal (C6) is then activated and latched in order to permit the clock to strobe data out of the FIFO serially into the link and it is only necessary the processor refills the FIFO from time to time.

6.3 Spreading

Consideration was given to the possibility of generating the code sequence using a purely software approach. However, the overhead required in adopting this approach represented a considerable proportion of the overall processing time. A compromise was needed between a pure software generation which results in a long processing time, and a hardware one using the available MSI packages (73). A look-up table stored in an external PROM (two of 825131 types) was used to generate samples of the code sequences, the code has been fully described in chapter 5 of this thesis. The table consists of L bytes of

samples each of which is stored as an unsigned 8-bit number. The table may be regarded as circular; the index of the current byte is always calculated modulo-L. The speed limitation on implementing this approach is due to the rate at which the data can be extracted from the FIFO in which case up to 1 MBps can be generated. Because the transmitter functions are entirely software controlled, it is possible to implement any sequence length by simple software modification.

Sequence inversion keying (SIK) or binary biphase phase shift keying (BPSK) in which the data to be transmitted is modulo-2 added to the code sequence, was adopted as the modulation scheme for the system. In theory, data and code sequences need not be synchronised, one problem of this is that it may be possible for anyone to read the data directly from a clean copy of the received signal. Systems which have coincident data and code sequence clocks are often said to have a data privacy feature (71). In this case, the data bits are completely hidden by the randomness of the code and this implies that the data bits cannot be extracted without first obtaining a detailed knowledge of the specific code sequence being used. In order to achieve a privacy feature in information transmission, and to simplify the transmitter and receiver structure, it is necessary for the spreading factor R defined as the ratio of the data and code symbol durations

$$R = T_m / T_c \quad (6.1)$$

to be integer, and for the data and code waveforms to change phase synchronously (87).

Therefore, we can write the transmitted signal equation as

$$s(t) = d(t)a(t) = \sum_{-\infty}^{\infty} d_{k/R} a_k \text{rect}_{T_c}(t - kT_c) \quad (6.2)$$

where $d(t)$ and $a(t)$ are the data and code waveforms given, respectively, by

$$d(t) = \sum_{-\infty}^{\infty} d_k \text{rect}_{T_m}(t - kT_m) \quad (6.3)$$

$$a(t) = \sum_{-\infty}^{\infty} a_k \text{rect}_{T_c}(t - kT_c) \quad (6.4)$$

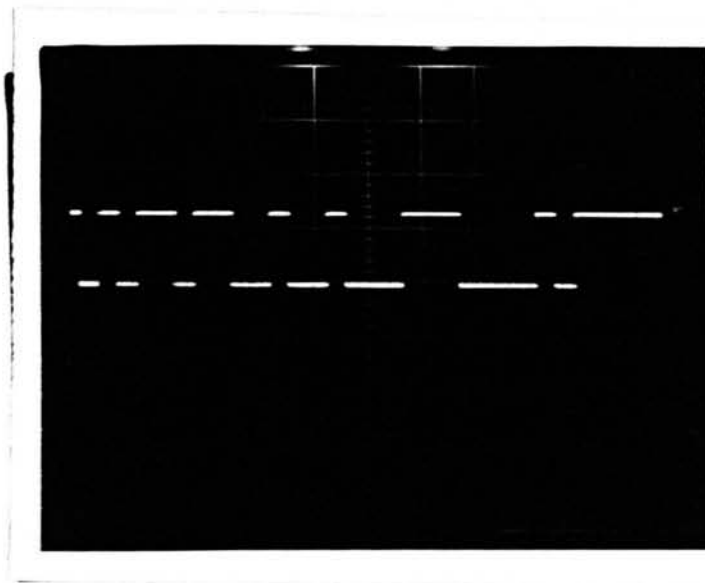
where $\{k/R\}$ denotes the integer part of k/R , d_k and a_k are the data and code sequences whose elements belong to the set $\{0,1\}$ and the notation $\text{rect}_{(T)}(t)$ is used here to denote a square T -second pulse of unit amplitude centred at the time origin. Hence the data clock rate is $1/R^{\text{th}}$ of the spreading code clock rate. This difference in clock rates (R is large) is necessary to produce spread spectrum effects. In this case R was chosen equal to L , i.e. one entire period of code sequence was contained in each data symbol.

6.4 Transmitter software

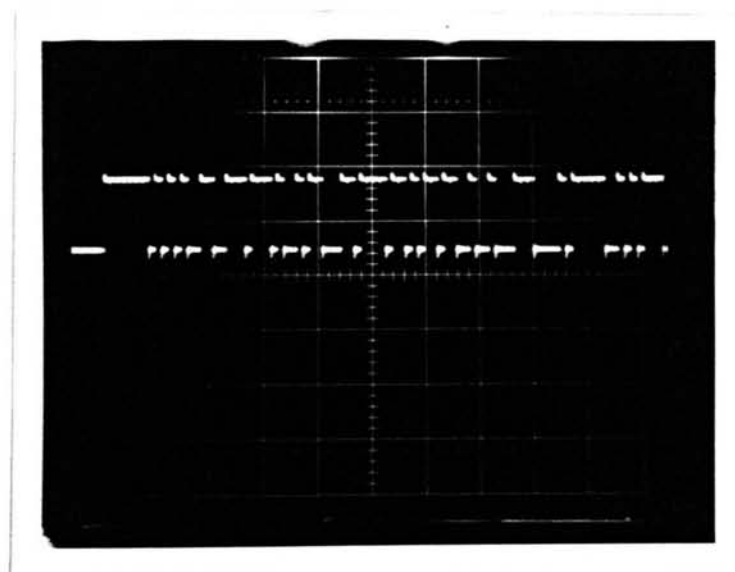
To implement the transmitter, 218 microinstructions were needed. The initialisation and display handling take another 85 instructions, so the total microprogram was 7Kbits. Utmost care was taken, to ensure that the part of the microprogram that deals with the actual transmitter (data synchronising, modulation, and data transmission), is carried out at the highest possible speed. As many functions as possible were performed in parallel. A listing of the transmitter microprogram is shown in the listing

in Appendix B.

To test the transmitter operation, the software was initially set up to generate a code sequence of length 127. The photographs of Figure (6.3) (a), and (b) show the transmitter output sequence in different cases of code clock rate. A charge coupled device (CCD) was available which, with its interface circuitry, allows the power spectrum of an analogue input waveform to be evaluated from a 512 point transform by the Chirp Z-transform algorithm (see chapter 2) in real time. This was used to display the power spectrum of this sequence, the result of which is shown in Figures (6.4) (a), and (b). Figure (6.5) shows the power spectra of the data information before spreading. The software was then configured to permit transmission of spreading signal with bandwidth approximately equals to twice the code rate. The power spectra obtained from this signal is shown in Figure (6.6). It has been emphasised in the illustrations that the spectral power envelope of a direct sequence signal follows a $((\sin x)/x)^2$ distribution. This is the expected result, for any good set of Fourier transform pairs (88) will show that the frequency function corresponding to a square pulse is $(\sin x)/x$ (which is a voltage distribution function) and code modulation produces a series of pulses. Because the power envelope is function of the voltage squared, the $((\sin x)/x)^2$ power spectrum results. Also due to the balanced property of the m-sequence, the spectral line at zero frequency of $P(w)$ is of reduced amplitude. It should be noted that spectrum analyser does not display phase information, only amplitude. Observation of any pseudonoise code sequence will show that it is made up of

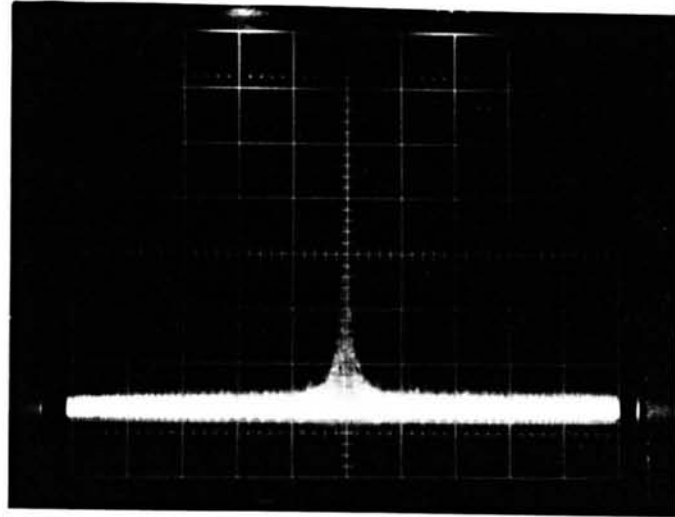


(a) $f_c = 14\text{KBps}$.

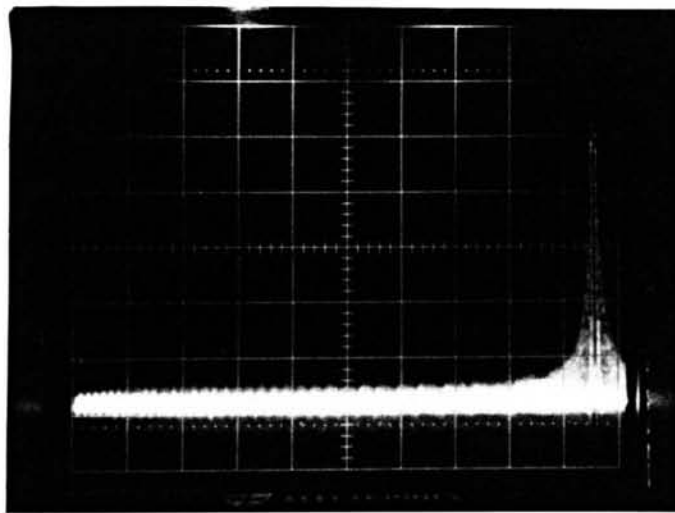


(b) $f_c = 1\text{MBps}$.

FIGURE(6.3) TRANSMITTER m-SEQUENCE OUTPUT.

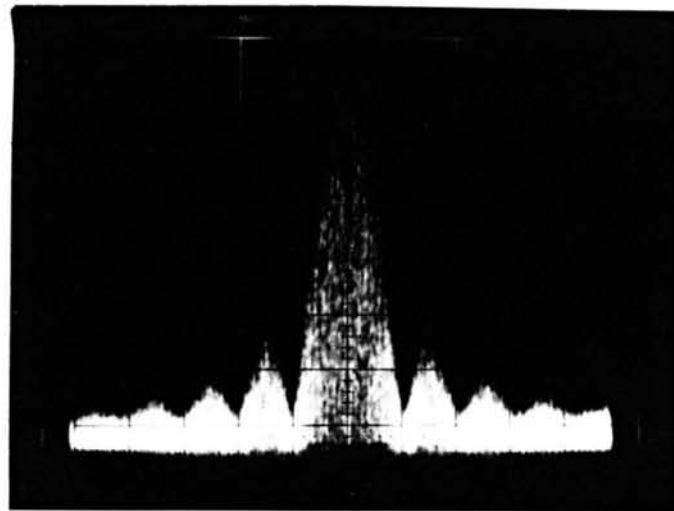


(a) One sided

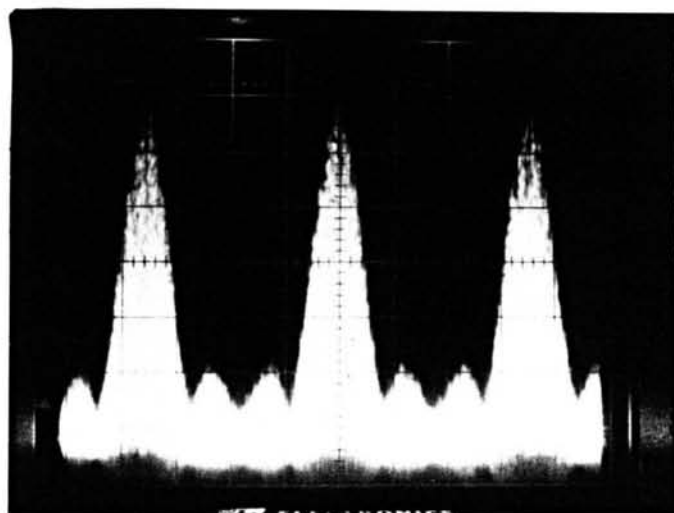


(b) Two sided

FIGURE(6.5) POWER SPECTRA FOR LOW DATA RATE (INFORMATON).



FIGURE(6.6) POWER SPECTRA FOR TRANSMITTED PN SIGNAL.



FIGURE(6.6) POWER SPECTRA FOR TRANSMITTED PN SIGNAL.

a series of variable-period pulses, which could be viewed as half-period sequence waves. These square wave half-periods vary in duration from one code clock bit to n bits for an m -sequence. Each of these half-periods has a $(\sin x)/x$ spectrum associated with it. As a code sequence modulates a direct sequence transmitter, the output spectrum is actually a composite made up of a series or group of spectra produced by the various half-wave components of the code. Because the pulse of shortest duration in the code sequence is equal in length to the code sequence clock period, T_c , it follows that the frequency spectrum for the composite modulation containing this code sequence must have a main lobe bandwidth such that its first nulls fall at the code clock rate. Reinforcing this expectation is the fact that the distribution of single ones and zeros in the code sequence is such that they outnumber all other pulse lengths. Chapter 5 has included a discussion of the distribution of the runs in an m -sequence. The number of frequency sets available is a function of the length of the code. For an n -bit sequence generator there are $n+1$ frequency sets, and the spacing of individual frequency components is as narrow as L/R_c .

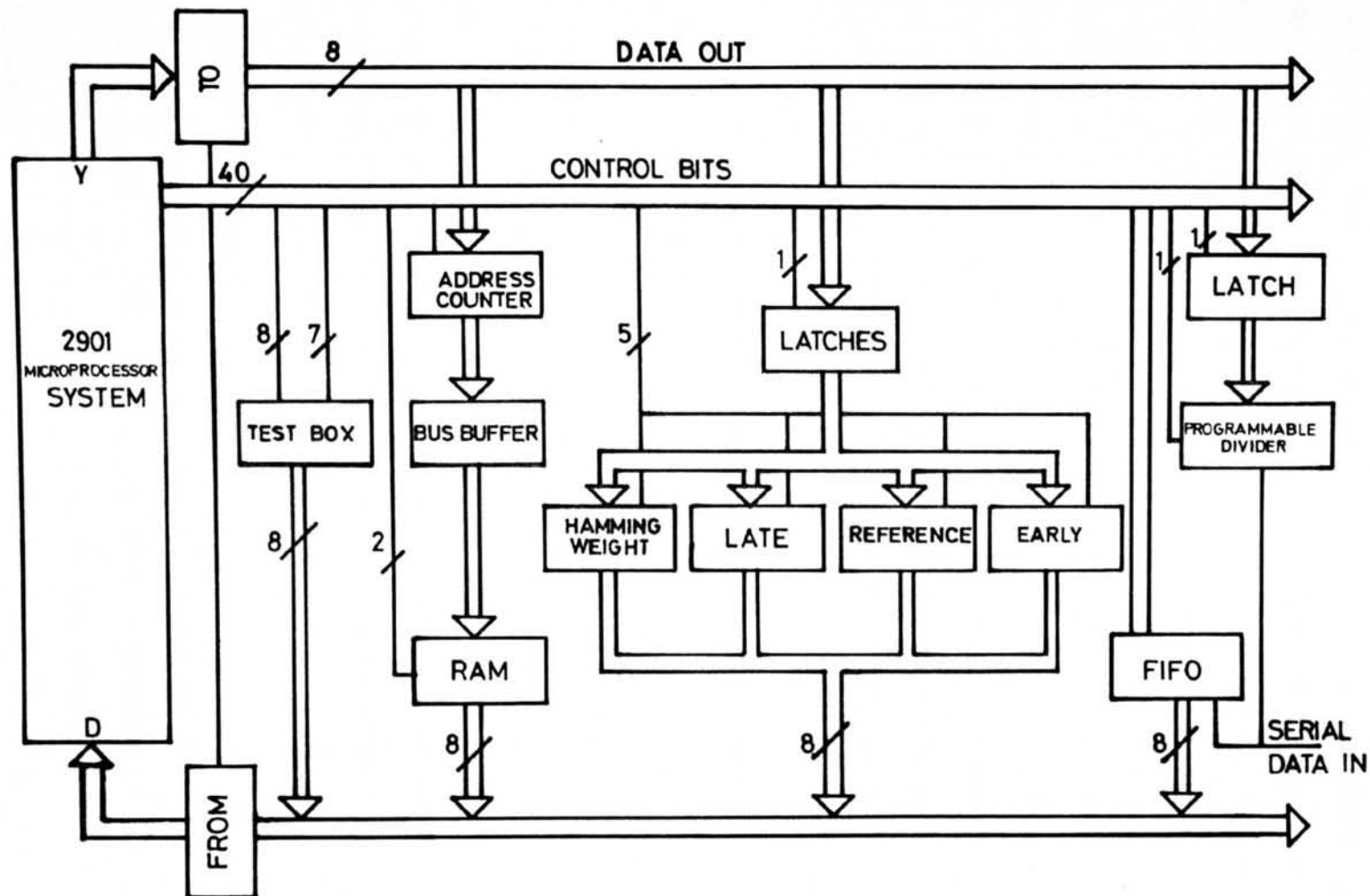
6.5 Receiver

The computational requirements in the receiving microprocessor for direct sequence modulation were considerably greater than for the transmitter. The major problem in the despreading was the synchronisation process for which it was necessary to compute the crosscorrelation function of a locally generated sequence which would enable the relative

time-displacement of the received and local sequences to be found, and hence, would permit initial acquisition. Practically, computing the whole crosscorrelation function would require a long processing time, however, it was envisaged that the use of a bit-slice microprocessor would permit the computation of the correlation function in the required time.

After acquisition, a tracking signal is introduced to maintain the codes in synchronism. Additional circuit may be activated at this time to check that the received signal has remained above threshold, which would not be the case if a noise signal had temporarily exceeded the threshold. If the confirmation fails, the acquisition phase would be resumed.

Figure (6.7) shows a block diagram of the receiving microprocessor for both acquisition and tracking processes. The 2901 microprocessor includes 1/2 kbyte of PROMs containing the receiving system software. The programmable divide by n counter, buffer memory, and the test box are similar to that were used at the transmitter. The external memory section consists of RAM and PROMs. One RAM IC (type TMM2016P), having a capacity of $2k \times 8$ bits and an access time of 100 n.sec was used to provide 2k of continuous memory capable of operating with clock frequencies in excess of 3 MHz. This RAM was used for data information in which the address word latch is a counter to enable the address to be incremented for each new access word. A look-up table stored in the external PROMs (8 of 82S131 type) were used to generate samples of a binary m-sequence a_0 (reference), a delay time-displaced version a_{-1} (late), an advance time-displaced



FIGURE(6.7) 2901 MICROPROCESSOR CONFIGURATION FOR THE RECEIVER.

version a_{+1} (early) (see chapter 5), and an 8 bit Hamming weight function. The Hamming weight function was implemented to provide a fast digital correlation capability in conjunction with the ALU exclusive-nor operation. The data bus interface is provided by two bi-directional tri-state buffers. The two control lines, transmit enable (TE) and receive enable (RE), allow three possible functions: data is passed from the processor to the external circuitry or from the RAM/PROMs to the processor, or both sides of the buffer enter the high impedance state. Figures (6.8) (a), (b), (c), and (d) show an interior views of the receiver.

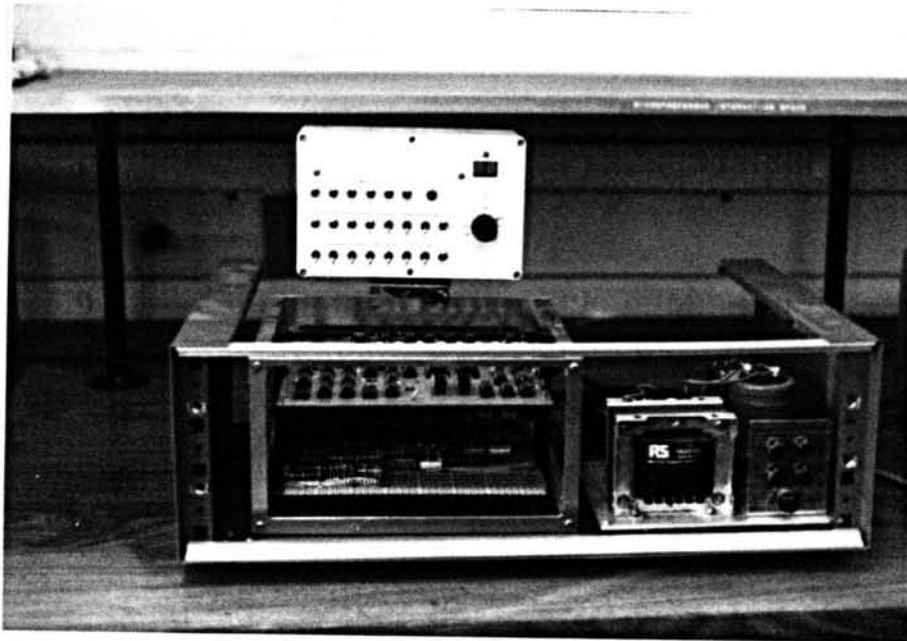
The tasks performed by the receiving microprocessor fall into three main categories:

- (1) initial acquisition
- (2) tracking
- (3) data recovery (readout).

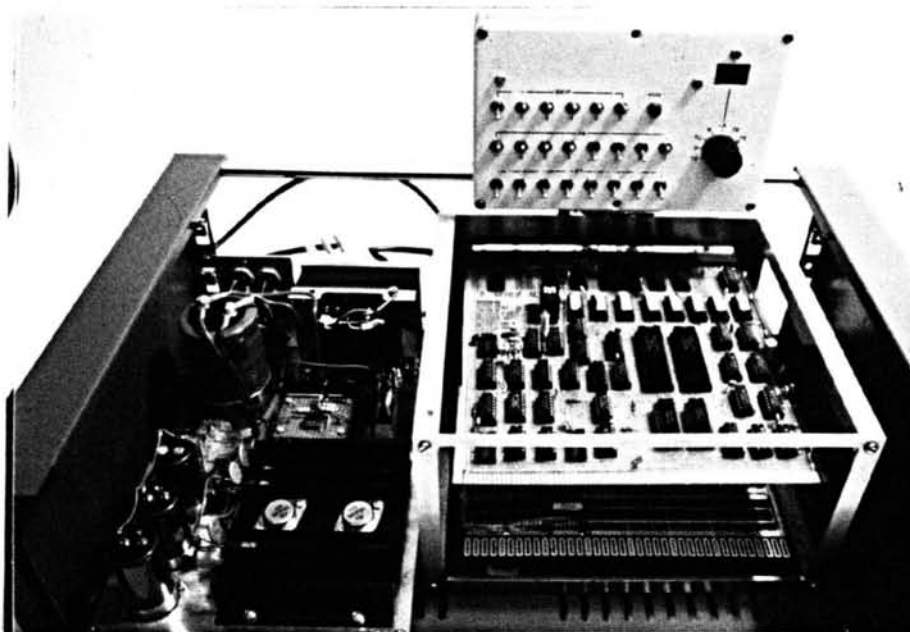
In this discussion, it is assumed that the input ~~is~~ consists of a pseudonoise sub-carrier which is BPSK modulated by the binary data. The RF carrier demodulation process may be achieved by other means.

6.5.1 FIFO on receive

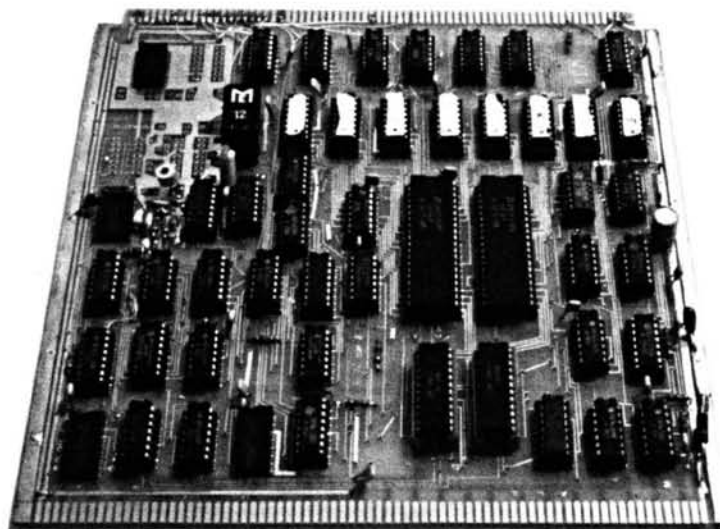
The input data sequence is serially clocked into the FIFO input register using the clock output which is generated by the programmable divider having a frequency ratio equals, $f_c/256 - f_c$ which is exactly equal to that ratio to be used at the transmitter. When the input register is full then inbuilt logic automatically requests that the word be dropped down the stack



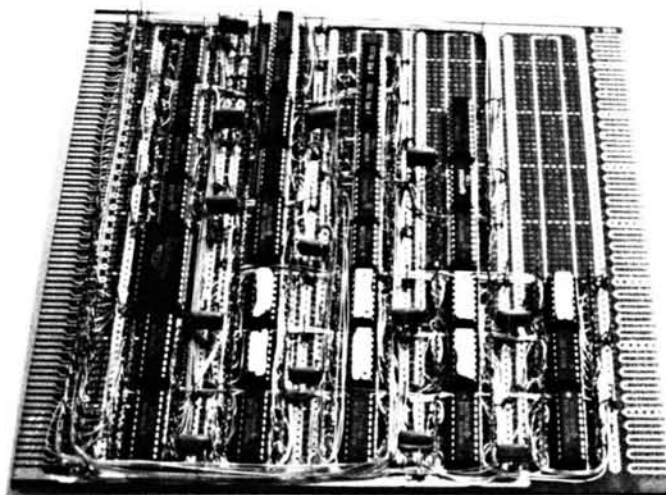
FIGURE(6.8a) RECEIVER HARDWARE (SIDE).



FIGURE(6.8b) RECEIVER HARDWARE (FRONT).



FIGURE(6.8c) 2901 MICROPROCESSOR BOARD.



FIGURE(6.8d) PROM/RAM BOARD.

towards the output register. "data request" (tied to input register full on the most significant FIFO (see Figure (3.10)) will go low briefly as this request is made and then honoured. If the line stays low then the FIFO is full and an error condition has been reached. Provided that the processor removes words from the FIFO faster than they are received then no problems should occur. The FIFO places a signal on "data accept" which loads the contents of the output register onto the 8-bit data bus. By clocking TOP (transfer out parallel) then the FIFO will be emptied onto the bus. "data available" (tied to output register empty 'S8') will indicate the state of the FIFO.

The advantages of using a FIFO buffer in this case was to ensure a good synchronisation between the incoming sequence clock rate and microprocessor system clock. In addition to this, one of the most powerful tools in enhancing the data processing capability, the pipeline processing technique, was inherent when utilising the FIFO. While the arithmetic operation on one set of data was performed, the read-in of the next set of data were executed concurrently.

6.5.2 Search/lock strategy

A functional diagram of the receiving system during synchronisation is shown in Figure (6.9). The diagram is divided into the three functions that were necessary for acquisition and tracking, i.e., a search/lock strategy to control all operations, a correlator detector to recognise when synchronisation has occurred, and an error-signal generation for a delay lock loop (DLL) correlator. The performance of

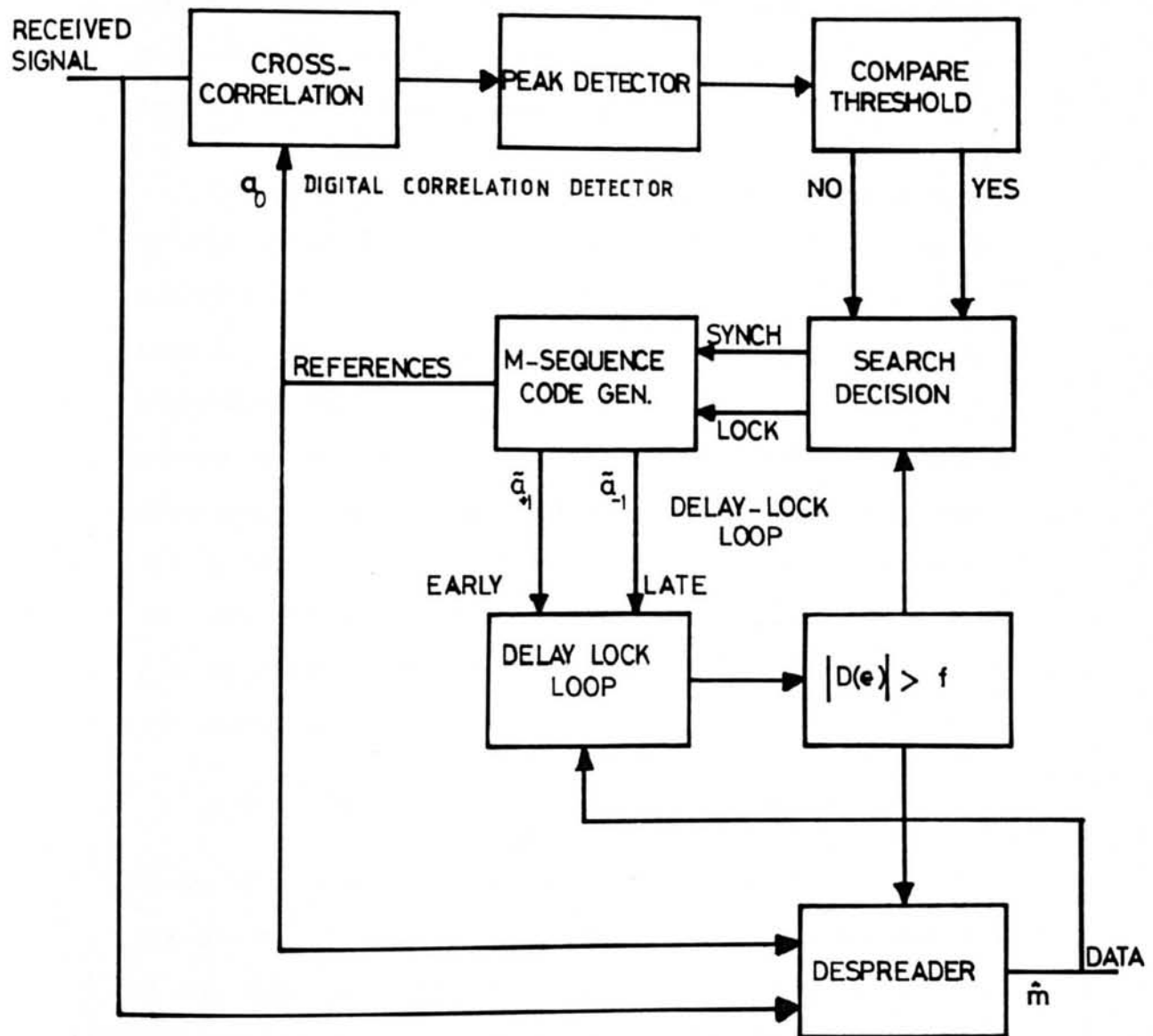


FIGURE (6.9) FUNCTIONAL DIAGRAM OF THE SYSTEM DURING SYNCHRONISATION.

correlator detector and the behaviour of the DLL have been discussed in chapter 5. In this section the intent is to realise the microprocessor implementation of synchronisation process including the interaction between the search and track modes.

Since rapid acquisition was desirable, the computation time of the crosscorrelation function should be as small as is practical. As mentioned previously, crosscorrelation over a subsequence will result in a fast search through the phase uncertainty region, but a low probability of detecting (P_D) the correct synchronisation when it occurs. Conversely, an accurate computation over the whole sequence length will result in a slow search but a high probability of detecting the correct phase. The scan rate in this case was chosen slow enough (equal to T_c) that the probability of detection P_D is equal to 1.

The received sequence is denoted by

$$r(t) = s(t+\tau) + n(t) \quad (6.5)$$

where $n(t)$ represents additive channel noise, $s(t)$ is simply a pseudonoise subcarrier which is SIK modulated by binary data as it was described by equation (6.2) (it will be assumed that $s(t)$ was used to modulate a radio frequency (RF) carrier usually by using multiphase phase shift keying (BPSK or QPSK), and then removed from the RF carrier prior to the crosscorrelation, this can be achieved by using a harmonic sampling technique (89) implemented with the aid of a pre-processor). The quantity ' τ ' represents the error (delay) in synchronising the received sequence with the replica. The received sequence initially may arrive having any phase with respect to the receiver replica.

The receiving system computes correlation according to equation (5.12), assuming the effect of $s(t)$, i.e.

$$c(\tau) = \sum_{n=0}^{L-1} a(nT_c) s(nT_c + \tau) \quad (6.6)$$

Single bit correlation, such as this, involves the multiplication of two functions followed by summation of the resulting products. The exclusive-nor multiplication logic which is the type of logic required for comparison, since it produces a '1' whenever two corresponding bits agree and a '0' when they don't, was used. That is, $c(\tau)$ is a binary correlation that was computed with very high speed ($3(L+1)/8$ microcycle) by an exclusive-nor operation with the aid of the Hamming weight function which was generated by using PROM. This operation is typically very difficult in FIS programmable processors. The receiving microprocessor performs a 127-bits correlation in 48 microcycle (16 u.sec). By this method, the receiver can identify a correlation during a period of time less than T_m , the reciprocal of the data rate. For a time uncertainty of $\tau = T_u$, there are T_u/T_c possible synchronisation points, each one of which must be tested for a time T_s . The maximum synchronisation time, T_{sync} , is therefore proportional to

$$T_{sync} = (T_s/T_c) T_u \quad (6.7)$$

The resultant of the correlation was then compared with a preset threshold γ ($\gamma \geq (L+1)/2$) before updating the phase position of the local sequence, synchronisation presence is then indicated when the correlation peak outputs exceed the threshold (in some

cases two consecutive peaks occurring would be required to indicate synchronisation in order to minimise the probability of false alarm P_{FA} (90)).

An error signal generation mode was then introduced in order to maintain synchronisation (tracking) to within the range $\pm T_c$. It was felt that fine tracking to within a fraction of $\pm T_c$ range would require the VCO function to be implemented in hardware outside the microprocessor and it was not possible to control the VCO without incurring a serious penalty in increased execution time. Instead, the three level error signal was obtained using a software technique by subtraction of two binary correlators which is given by

$$e = \sum_{n=0}^{L-1} (s_n \tilde{a}_{n+1} - s_n \tilde{a}_{n-1}) \quad (6.8)$$

where a was a priori modulated by the estimated data as was discussed in chapter 5, i.e.,

$$\tilde{a}_{n+1} = a_{n+1} \cdot \hat{m} \pmod{2} \quad (6.9)$$

For example, in the case of a 127-bit code sequence, every summation of the tracking correlator output 'e' takes 144 microcycles (as shown in the listing in Appendix B). The resultant 'e' was first compared with a predefined deviation range, $+d$ 0 $-d$, representing the error curve. For that specific example, these values are $+4$ 0 -4 if the levels of the binary sequence are 0 or 1, that may be estimated by running the transmitter-receiver microprograms on the 2901 simulator. If 'e' lies within this range, the current reference code sequence was

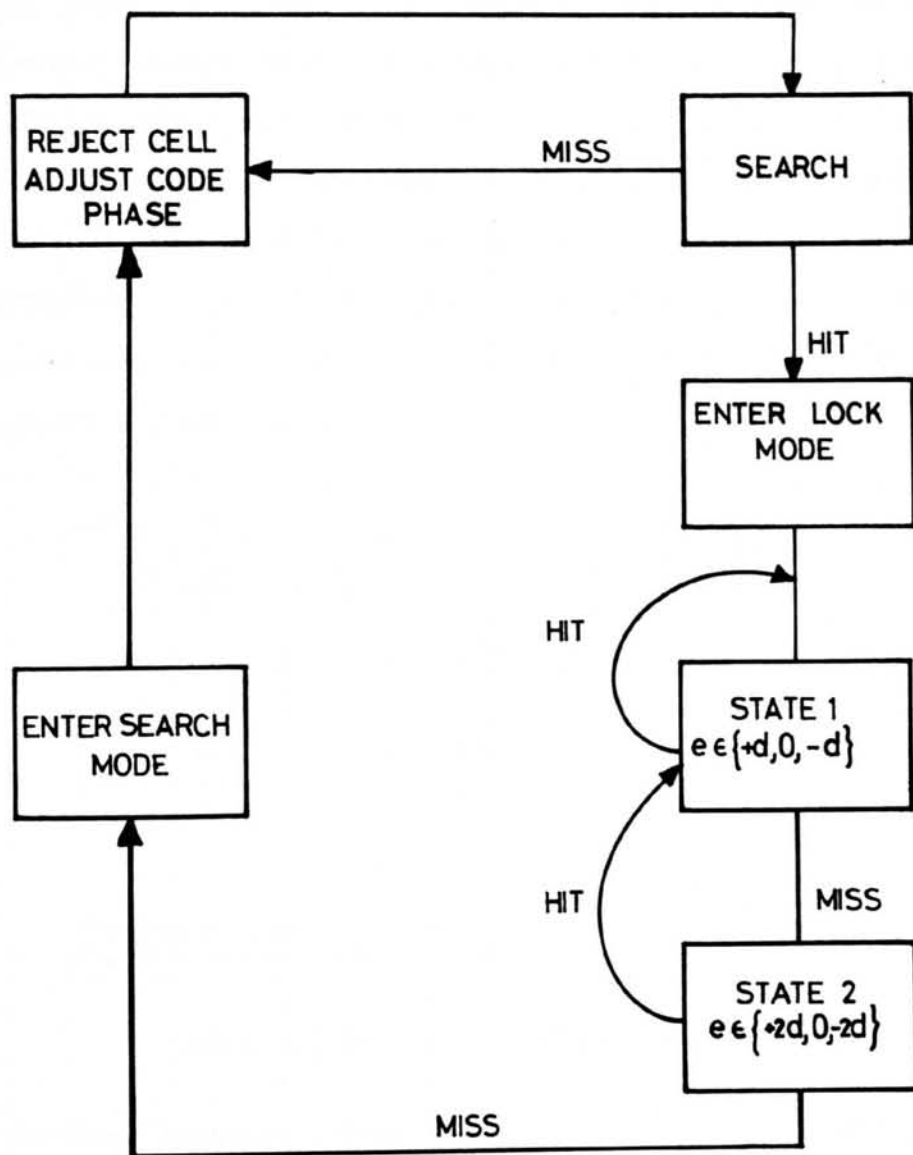
then used to despread the incoming signal. If 'e' exceeds this deviation and remains within the range, $+2d$ 0 $-2d$, the sign of 'e' was used to indicate the direction of which the reference code will slide, accelerating if 'e' is plus and retarding if 'e' is minus. Accordingly, the reference code address counter was modified in the form;

$$\text{new address} = \text{old address} + (L+1)/8 \pmod{L}$$

Finally, if 'e' was heavily out of range or the track modification failed in two consecutive trials which means that a synchronisation loss has arise, then the search mode may be reinitiated.

As described, this software technique cannot establish correct phasing between the incoming bit stream and the receiver. This is not a serious limitation for signals with short interpulse intervals such as this for which the receiver can clock in each data bit anytime during its valid state, i.e., at any phase within a fairly broad range.

The above strategy, which was the basis for the receiver microprogram, may be summarised as shown in state/transition form in Figure (6.10). Beginning with the first test of a phase position (cell), a miss will result in immediate rejection of the cell and a phase step to the next cell. A hit will cause the microprogram to enter the lock mode. A miss in state 2 of the track mode will cause a return to the search mode.



FIGURE(6.10) A SEARCH/LOCK STRATEGY.

6.5.3 Receiver Software

Table (6.1) presents a processor loading summary for the receiver implementation. This implementation was coded using the 2901 microcode (see Appendix B). The budget presented in Table (6.1) represents the worst-case loading of a 200 Kbps PN signal. These figures demonstrate the throughput capability and relative efficiency of the 2901 microprocessor system. The worst case processing load is approximately 70% of capacity, and 50% of program and data memory.

<u>222 instruction per data bit at 200 Kbps (BPSK)</u>			
<u>Acquisition mode</u>		<u>Tracking mode</u>	
initialisation	38	DLL correlator	60
correlation	64	despreading	16
peak detection	8		
data readout	36		
	<hr/>		<hr/>
	146		76
Acquisition mode = $146/222 = 70\%$			
Tracking mode = $76/222 = 35\%$			

TABLE (6.1) RECEIVER LOADING SUMMARY.

6.6 Clock Frequency Effects

One of the prime sources of uncertainty for synchronisation in the system was the clock rate error due to drift of the clock frequency generator outputs which were used to control the rate of code signal to be transmitted and the receiving sequence. The preceding discussion assumed that the transmitter's and the receiver's clocks were synchronised during the search mode, i.e., that each pulse of the receiver's clock would load in the

sequence bit immediately following the one loaded on the previous clock cycle. If the transmitter clock runs faster than the receiver clock, the receiver will periodically miss a sequence bit of the incoming signal. In contrast, a fast receiver clock occasionally loads a single bit twice in succession. These "missed-bit" and "doubled-bit" errors have a serious effect on the initial acquisition process. Any clock rate error is cumulative in code phase error; that is, a one-bit cumulative shift during the search time. These can be avoided only if the two clocks are synchronised. A typical clock and clock-adjustment hardware system comprises (91), (92) a VCO, a counter, a digital-to-analogue converter, and a simple voltage matching circuit. In that case, the number of bits between frame synchronisation pulses is counted and converted to a voltage, which is used to adjust the clock rate. If the number of receiver frame bits is smaller than the standard transmitted frame length, the clock is accelerated proportionately. Bit synchronisation systems such as this could not be used in this case, because they are applicable only to signals whose frame size is either fixed or varies according to a pattern known to the receiver, and must include periodic frame synchronisation patterns. One method that could overcome this clock drift accumulation problem during the search mode was implemented using a sampling technique. The system performance can be improved by setting the receiver clock rate equal to nR_c , where $n=2m+1$ is an odd integer. If the number of sampled 0's exceeds the number of sampled 1's in a block of n sampled digits, then the receiver announces a 0, and otherwise it announces a 1. Thus an error

occurs when $m+1$ or more samples out of $n=2m+1$ are incorrect. This method coincides with the Nyquist criterion (24) for which the received data must be sampled at twice (at least) the signal bandwidth ($2R_c$). Although in principle, the choice of n to be odd integer (rather than $n=8$) could be implemented with the microprocessor, the execution time would then have to be so long as to make this method useless. During tracking, the receiver clock requires only minor periodic adjustments to maintain synchronism with the transmitter clock which can be achieved using the track correlator.

6.7 Data recovery

The receiver signal was the data-modulated sequence to which noise had been added, as will be discussed in the next chapter. Since the delay-lock loop closely tracks the incoming sequence, the receiver's reference code and the received sequence were very nearly synchronous and either correlate or anticorrelate depending on whether a data 0 or 1 was being received. Two different ways exist for the form of the data on the sequence, these are unipolar and bipolar. A unipolar form has the values 0, 1 whereas a bipolar form has the values -1, +1. Practically, a bipolar sequence has no dc component so that all its power is modulated by data (77). In this case the input signal was limited to provide 0 and +4 volt levels (TTL levels). Since the reference sequence was also at these levels, a simple exclusive-or operation was used to recover the data information from the received sequence. If the received sequence and the reference were anticorrelated, the data output is a '1' level.

If the inputs are correlated the output is '0' level. This output data was displayed and analysed using the logic analyser as well as the test box (experimental results will be discussed in the next chapter) and shows complete agreement between the recovered data with that of the transmitter.

6.8 Conclusion

The real-time realisation of the baseband signal processing requirements at the transmitter and the receiver of a PN spread spectrum communication system using the 2901 bit-slice microprocessor system, together with an external digital circuitry has been investigated. The use of a memory buffer 'FIFO' in data manipulation has shown to be effective. The flexibility and high throughput in computing the correlation functions, and defining the error signal which is required to control the tracking mode by software have been described and experimental figures have been shown using a 127-bit sequence length. The effects on synchronisation uncertainty due to clock drifts has been described and are shown to be minimised by using sampling techniques. In addition, a data readout method was provided.

CHAPTER 7

System Performance & Experimental Results

7.1 Introduction

Direct sequence (PN) spread spectrum system performance in the presence of noise and interference has been analysed (93), (94), (95) and formulas have been developed for different forms of interfering signals (such as Additive White Gaussian Noise (AWGN), impulsive noise, and intersymbol interference). These formulas show the fact that the effect of the interfering signal is reduced by increasing the spreading ratio and hence by process gain (the ratio of clock rate of the PN code to the information data rate). That is, by increasing the clock rate of the PN code, the receiver will be able to demodulate the incoming signal in higher levels of interference. Various limitations exist with respect to increasing the code rate, and hence expanding the bandwidth ratio, arbitrarily so that process gain may be increased indefinitely. At present, integrated circuits are available which allow limited code generation at rates up to 300 Mbps. As code rates become higher, operating errors will become inversely lower, but on the other hand, high-speed logic circuits are sensitive to noise and are more liable to error. Also, high-speed digital circuits consume large amounts of current and their power dissipation is high. These reasons, in addition to the problems of spectrum occupancy, equipment implementation, and propagation constraints, tend to limit the code rates used for signal spreading. Data rate reduction to improve process gain is

limited by the tendency to speed up the information transfer and by the stability of the transmission link. However, there is a difference in high-speed codes used for spectrum spreading and in high-speed data. The difference in the two, results from the errors in the system; that is, an error (or errors) up to some correctable limit may be tolerated in a data-modulated system, whereas an error in a code sequence would completely disable the code-modulated system until resynchronisation occurs.

Any significant degree of crosscorrelation between the receiver's local reference and an interfering signal can detract from performance. An important problem in the design of PN systems lies in using codes that are too short for the rejected interference levels. The shorter codes when multiplied with interference tend to produce correlations that are not at all noiselike. Therefore a synchronisation detector in such a system is likely to give poor performance when it has been assumed that correlator output products due to interference are characteristically Gaussian. A good rule of thumb for selecting a minimum code length in a PN system is to choose the sequence used (7): $\text{code bit rate/code length} \geq f_{\min}$ where f_{\min} is the lowest frequency of interest in the information being modulated. That is, the code repetition rate should not fall in the information passband. Otherwise, code/interference crossproducts may fall into the demodulated signal band, so reducing receiver output SNR. Such crossproducts can also greatly increase the incidence of false synchronisation recognition.

This chapter discusses the performance of the receiving system in the presence of noise. It may be desirable to incorporate some filtering at the input of the receiver to minimise the effects of high level interference, however this may cause bandlimiting influences. It is shown that local code synchronisation errors become the dominant factor in the error rate when the system bandwidth increases.

7.2 System Performance

It has been mentioned that a major system performance measurement associated with the design of a PN spread spectrum system is the processing gain (G_p) which is the difference between output and input signal-to-noise ratio (SNR) in the receiver processor. This can be written, in the presence of AWGN, as (7)

$$G_p = B_{ss} / B_d \quad (7.1)$$

where B_{ss} is the bandwidth in hertz of the transmitted spread spectrum signal and B_d is the minimum bandwidth that would be required to send the information if we did not need to imbed it in the larger bandwidth for protection. This can be represented as the ratio of chip rate to the data rate, assuming that the code chip rate is much greater than that of the data. The process gain expression of equation (7.1) shows the ratio of SNR improvement, which is directly related to the bit error probability in this case, at the output of the despreading correlator for a specified input SNR to the correlator. This does not imply that in presence of AWGN the SNR at the output of the spread spectrum system, prior to data demodulation, is

greater by a factor G_p compared with a narrowband scheme. It can be observed that for a spread spectrum system since the front-end bandwidth is much higher than the data bandwidth, the noise power at the input of the correlator (despreading) processor, is much larger than the output. In other words, for a fixed signal power, the SNR before despreading is less than the output SNR by the process gain G_p . Thus there is no ambiguity in the meaning and existence of process gain in a spread spectrum system. However, in the presence of narrowband interference an improvement in signal to noise ratio at the output of a spread spectrum system will result. This is because the output signal to noise ratio is G_p times larger than for the narrowband scheme. Since the input noise power in this case is power limited, the spread spectrum scheme, in despreading the signal, spreads the noise power over a bandwidth comparable to the front-end. In general, it may not be possible to construct such an implementation.

The ability of the receiving microprocessor to efficiently despread the spread spectrum signal, as measured by acquisition time and bit error rate, is directly related to the performance of the algorithms under actual operational conditions. The quantisation of signals (which will not be discussed here) required by the finite register length and processor computations, tend to degrade the performance of the algorithms. As a first step in evaluating the performance, a microprogram was developed on the 2901 simulator. The simulator helped to determine the implementation of processor functions, as well as

to expose certain weak areas in the algorithms which could significantly degrade performance. For example, when the acquisition algorithm was simulated, it became apparent that a threshold could not be found which would give acceptable performance. If the threshold was low, the algorithm tended to detect too early. As the threshold was raised, the peak point in the correlation where synchronisation is declared had an increasing effect on performance. The results of the simulation led to refinements of the algorithms and aided the development of the receiver microprogram.

The performance of the receiving system is measured in terms of :

- (1) the speed of synchronisation; and
- (2) the bit error rate (BER) or the probability of error in the data at the output of the demodulator.

Missing synchronisation can be attributed to two major causes; the first is the probability of not detecting the correlation peak due to the channel noise. The second, is the large error in symbol timing due to clock drifts.

7.2.1 Acquisition Time Measurements

The local clock at the receiver samples the incoming sequence at regular intervals and loads a one or a zero, into the FIFO depending on whether the level of the signal is greater than or equal to zero at that instant. Note that the sampling clock and microprocessor instruction cycle were coherently related, i.e. $f_s = f_0/r$. In this discussion we will assume that T_s , the sampling period is regular and equals T_c (code

chip duration), i.e.

$$T_c = rT_0 \quad (7.2)$$

where T_0 is the microinstruction cycle (330 n.sec.), and r is a dividing ratio ($1 \leq r \leq 255$). For the present, the local clock f_c will be assumed to have the same frequency as the incoming sequence clock (the effects of clock instability, and the additive noise will be considered later). In this system, a single correlator based on the 2901 microprocessor system was used to achieve acquisition using the sliding correlation technique (see chapter 5). Initially the output phase k of the local PN generator is set to $k=0$ and a partial correlation was performed by examining a period of λ chips. The procedure steps through all possible states of the local PN sequence until the state is found which correlates with the input. The maximum synchronisation time is given by equation (6.7) and is repeated here for convenience;

$$T_{\text{sync}} = (T_u / T_c) \cdot T_s \quad (7.3)$$

where T_s is the search time at each code chip. Equation (7.3) means that the product of the number of cells and the expected search time per cell determines the maximum expected search time required to achieve synchronisation. It should be noted that the above equation ignores the effects of frequency uncertainty due to Doppler shift (5). The search time T_s can be expressed as:

$$T_s = T_e \cdot T_0 \quad (7.4)$$

where T_e is denoted by the examination time which is a function

of the partial correlation period λ (the number of chips examined during each search). Depending upon the statistics of the time uncertainty, the average uncertainty time T_u may be less than $\Delta \times T_c$ ($\Delta \leq L$);

$$T_{\text{sync}} = \Delta \cdot T_e \cdot T_0 \quad (7.5)$$

therefore the average acquisition time ($\Delta = L/2$), which is defined as the expected value of the time which elapsed between the initiation of the acquisition and its completion, can be estimated as

$$T_{\text{acq}} = (L/2) \cdot T_e \cdot T_0 = 2^{n-1} T_e \cdot T_0 \quad (7.6)$$

which is identical to that obtained by Holmes and Chen (96). On the other hand, the time required to load one data bit was

$$T_m = L \cdot T_c \quad (7.7)$$

where T_m is the data bit duration. The ratio of the average acquisition time to the time required to receive one data bit, therefore, can be written as;

$$T_{\text{acq}}/T_m = T_e/2r \quad (7.8)$$

Equation (7.8) is the key to the receiver performance and its limitation. Note that equation (7.5) has ignored the fact that the noise will occasionally cause a false in-lock indication (false alarm), and also occasionally will cause a true lock to go unnoticed (false dismissal). These parameters, the probability of false alarm P_{FA} and the probability of false dismissal P_{FD} that depend on the SNR at the receiver input and on the

acquisition threshold level, were very difficult to measure in this case. These effects can be minimised by choosing the acquisition threshold level to be high enough ($= L/2$).

It would appear from equation (7.5) that decreasing T_e , which means a short partial correlation process, would continue to decrease the acquisition time. However, a factor which becomes important when the correlation period ' λ ' is made short, is a form of self-noise at the correlator output. When the summation period is long, the result of cross-correlating the signal and the reference sequence is approximately zero when they are out of phase. However, as the summation period is made shorter, occasionally the period which was chosen, although the phase was incorrect, will correlate rather well with the incoming signal over a short time, this will cause the self-noise false alarm probability to increase. Note that there is no self-noise false dismissal because an ideal choice of ' λ ' produces no self-noise at the correlator output.

It was mentioned in section (6.6), that the effect of the clock frequency difference, due to instability of the transmitter and/or the receiver clock, is that occasionally the input sampling process misses an input bit or samples the same bit twice. The allowable difference frequency f_d before acquisition is

$$f_d < f_c/L \quad (7.9)$$

For larger difference frequencies, it is not possible to achieve acquisition no matter what the relative phases of the sequences

are during the sampling process. Since the initial phases of the sequences are uniformly distributed, the probability of achieving acquisition varies linearly with $f_d/(f_c/L)$ so that the average acquisition time when this effect is included becomes:

$$T_{acq} = 2^{n-1} T_e T_0 / (1 - L f_d / f_c) \quad (7.10)$$

A group of measurements was performed using different clock frequencies with 127-bit biphase modulated sequence. Acquisition times were measured using a test box and a Hewlett-Packard Model 1615A logic analyser. The examination period can be adjusted by suitable choice of λ . A reset circuit contained a manually operated switch which starts the receiver microprogram from any address location. The logic analyser was used to measure the relative time from the moment of resetting the switch (INITL) until the end of the acquisition phase. This time included that required for loading the FIFO. In order to measure acquisition times without degradation due to clock instability, the receiver clock bus also was taken from the transmitter clock. This effectively bypassed the tracking loop. For particular settings of the sequence length (127-bit) and the correlation period $\lambda = L$, the threshold setting which gave optimum probability of detection was found (63 (decimal)), and that threshold setting was generally used for all runs. Predicted acquisition time using equation (7.5) agrees closely with the results where the transmitter clock was used also for the receiver. When a delay-lock tracking loop was used so that the clocks were independent, acquisition times were expected to be about 70

percent longer.

7.2.2 Bit Error Rate Measurements

In a small area experiment such as this which is a single cell binary communication system, co-channel and adjacent channel interference can be ignored. Interference due to both other spread spectrum users and conventional users is not possible in this case. Also, channel distortion due to path attenuation, fading, multipath distortion, and Doppler shifts were neglected. In built-up areas man-made impulsive interference from machinery, fluorescent lights, power switching appliances, is common. It is not readily apparent how immune or not PN spread spectrum binary communication system may be to such noise. The shorter duration of those pulses suggests that, although they are wideband, the energy is limited and a correlation type despreading process may render it insignificant.

The bit error-rate (BER) or the probability of error (P_E) performance of the system depends primarily on the strength and nature of the noise (errors) which corrupts the received signal and on the effect of the clock frequency difference between the incoming sequence and the receiver (see chapter 6).

Experimentally the BER is measured and defined by equation (7.11)

$$BER = N_e/N_t = N_e/R_m t_m \quad (7.11)$$

where N_e is the number of bit error in time interval t_m , R_m is the data bit rate, and t_m is a measuring time interval,

i.e., error counting time. For a random, stationary error generation process and sufficiently long measurements interval t_m , the measured BER gives an estimate of the true probability of error P_E .

Evaluation of BER of non-operational (out of service) channels is a well-known measurement technique. A preliminary experiment was performed where the bit error rate was measured with a 127-bit code sequence which is transmitted through a wire communication link. The receiving microprocessor computes BER by comparing the recovered bits with a stored replica of the transmitted data bits. Error rates were counted for the case where the receiver clock line was open and using the transmitter clock for both the transmitter and receiver. The main problem associated with simple out of service BER measurements is that it is not feasible to evaluate the performance of operating in a service system carrying the unknown digital data stream. The measurement duration and the error rate count for a short time t_m might also cause serious difficulties. For example, to evaluate a $P_E=10^{-9}$ for a meaningful statistical estimate, at least 10 bit errors have to be counted, the measurement had to last for $t_m = 10^5$ sec (nearly 30 hrs), which was an impractical time in the case of dynamic operation. Many techniques have been reported in the literatures to evaluate the BER of in service or on-line monitoring channels such as;

- (1) test sequence interleaving
- (2) parity check coding
- (3) code violation detection

(4) pseudo-error detection

Those techniques require a feasible data readout equipment which exceeds the capability of the receiving microprocessor.

As it was mentioned earlier in chapter 6, data readout can be achieved by using a post-processor which is also may be used for counting the error rate. In this case, the test box and the logic analyser were used for data readout and error rate counting. The total numbers of errors accumulated at the end of the measurement period t_m was displayed on front panel LED's attached with the test box.

7.3 Noise Channel Simulation

The channel is the medium used to transmit the signal from the transmitting to the receiving point. It may be a wire link or a band of radio frequencies. During transmission, or at the receiving point, the signal may be perturbed by noise or distortion. In principle, distortion can be corrected by applying the inverse operation, while a perturbation due to noise cannot always be removed, since the change of the signal is not the same during transmission (1). In binary communication systems, the channel accepts 0's and 1's at its input and usually reproduces them at its output. Occasionally, however, because of noise and other channel impairments, the output digits do not agree with the input digits and errors have occurred.

In order to permit accurate tests on the system, a digital pseudonoise simulator that generates pseudonoise with good accuracy and impulsive noise with random pulses was required. The use of digital techniques to generate the noise makes it easy

to repeat the measurements. Since the generator can be controlled using software or hardware, it can be used in operational measurement and since a digital output of the noise may also be available the generator can act as a main or a peripheral unit that simulates the channel. For these reasons, the microprocessor was found to be the lower cost choice for achieving these requirements.

An experiment is described in this section, in which this technique was used to investigate the performance.

7.3.1 The microprocessor

The choice of a suitable microprocessor device was considered very carefully. A device was required having a comprehensive instruction set, fast execution speed, and for which support facilities were available. The Motorola MC6803 (97) satisfied these requirements and was chosen in view of the following merits:

(1) The MC6803 is object code compatible with the M6800 (98) instruction set and includes improved execution times of key instructions (80 basic instructions and 7 addressing modes). In addition, new instructions have been added; these include 16-bit operations and a hardware multiply.

(2) M6800 cross-assembler and several flexible development systems (a triple disc drive and interface board to facilitate high speed file access, and an EPROM programming card allowing machine code programs to be transferred from RAM to 2 Kbyte EPROMs under software control) were available on the department's M6800 computer which were used for developing the MC6803 software.

(3) Software packages, components, and good documentation were readily available for the M6800.

(4) Execution time is fast (normal clock frequency = 1MHz; average instruction time is approximately = 4 cycle).

The MC6803 is an 8-bit microcomputer having an 8-bit data bus and a 16-bit address bus. It has a 128 byte of RAM and seven internal registers: the A accumulator (8-bit), the B accumulator (8-bits), the D accumulator (16-bits), a program counter (16-bits), a stack pointer (16-bits) and a condition code (status) register (8-bits). The device provides an 8-bit port and a 5-bit port for interfacing to peripheral devices. It contains an on-chip 16-bits programmable timer which may be used to perform measurements on an input waveform while independently generating an output waveform. The MC6803 contains an asynchronous serial communications interface (SCI). The device requires only the addition of a ROM and an external crystal for microcomputer unit (MCU) normal operation.

7.3.2 Hardware Description

The constituent components of the 6803 microprocessor card are shown in the circuit diagram of figure (7.1). The 74LS373 transparent octal D-type latch was used in conjunction with Address Strobe (AS), to latch the least significant address byte. Address strobe signals the time to latch the address so the lines can output data during the 'E' pulse. This signal was also used to disable the address from the multiplexed bus allowing a deselect time before the data is enabled to the bus. The MC6803 software was contained in a 2716-type EPROM, occupying memory



FIGURE(7.1) THE 6803 MICROPROCESSOR CARD.

between F800-FFFF. Memory decoding and control was performed by 3-gates. The R/W line is ANDed with the E signal to provide an output enable (OE) which was used by the EPROM. The input/output port (P10-P17) was buffered by an octal tri-state buffer, type 74LS244. The reset line was not buffered. The 6803 would come out of reset when RESET is at a level above about 4 volts. The system clock was derived from a 4 MHz crystal, giving a 1 u.sec instruction cycle time. The 'E' signal was therefore 1 MHz.

7.3.3 Implementation

The MC6803 microprocessor software was required to (a) read the serial output data from the transmitting microprocessor, (b) generate the noise or interference signal and mix it with the incoming signal, and (c) send the perturbed signal to the receiver. After system initialisation, the transmitted signal which was represented by equation (6.2) (assuming a normalised unit power) was read using one bit of the 8-bit I/O port. Two other bits were used for clock input and perturbed data output.

Two main types of noise were implemented; pseudonoise binary sequence and impulsive noise. The use of a pseudonoise binary sequence, which was discussed in chapter 5, as a noise source results in a simple hardware realisation of the additive noise with parameters that can be modified using software. Since we were using a one-bit word length as an I/O, this saves the use of A/D and D/A converters.

Impulsive noise, on the other hand, has been used frequently for testing the performance of the communication systems (99). This can be reasonably modelled as a time series of impulses at

the receiver input;

$$n(t) = \sum_{i=1}^N b_i \delta(t-t_i) \quad (7.12)$$

where the strengths (amplitude spectral densities) b_i , N , and time of occurrence t_i of these impulses are random variables. In this case, the occupation time of the impulse noise, when it occurs, was assumed to be 1 percent. This implied that with the noise bit of the same duration as the signal bit (in practice this sort of noise, when it occurs, may last to up to few msec. so that errors in data transmission appear in bursts separated by relatively long intervals), the average number of noise pulses to the number of signal bits is 0.01. However, this ratio was considered over a range 0.001 to 0.1, i.e. a factor of ten either way. For the sake of simplicity, and the speed requirements, two assumptions were considered in the software design. First, the noise pulse duration was considered to be an integer number of the code chip (≥ 1). Second, the transmitter clock was used by the 6803 in order to read the data samples under software control. By this method, the maximum data rate was dependent mainly upon the speed of the 6803.

7.4 Experimental Results

The equipment designed for experimental verification of the preceding sections is shown in block diagram form in Figure (7.2). The designs were made on the basis of simplicity or convenience and of the maximum efficiency of using the 2901 system in developing the digital signal processing requirements, as was discussed in the preceding section and in the previous

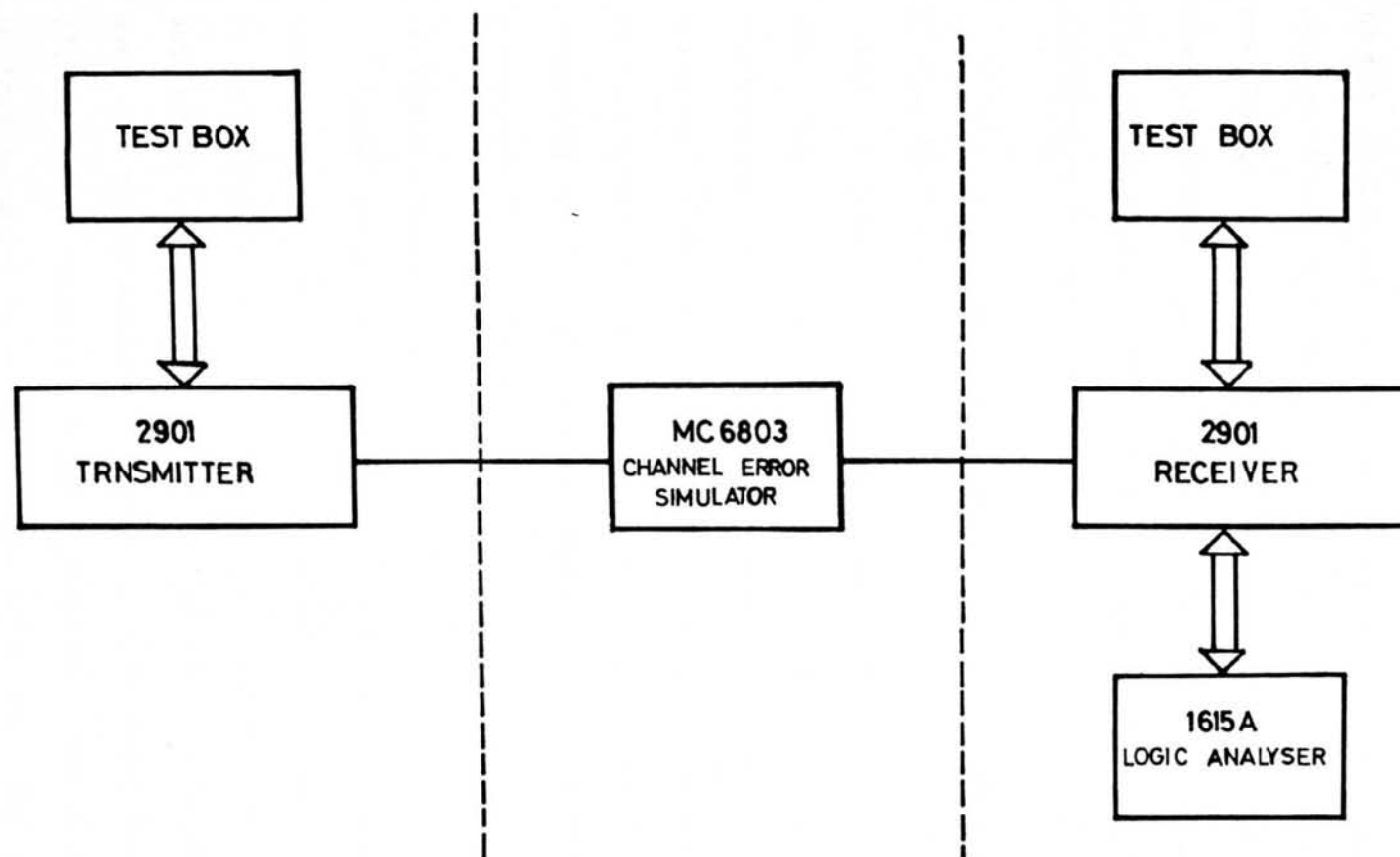


FIGURE (7.2) ERROR MEASUREMENTS EQUIPMENT.

chapters. The transmitter and receiver microprograms were tested under real-time conditions using a test box and the Hewlett-Packard model 1615A logic analyser. The trace specification of the 1615A logic analyser was set up to present a two-dimensional view of the microinstruction address vs. the 2901 data output. This display allowed verification of the algorithms to be made at a glance.

Throughout all the measurements which were performed, the 127-bit ($n=7$) sequences was used, this length being chosen to provide a useful processing gain whilst retaining a respectable data rate. It should be noted that the system is capable of implementing longer sequences by slight modification of the software. It was assumed that, for timing computation, the receiver presupposes random arrival time of the binary PN signal. Acquisition of reference code and received signal is confirmed in a time equivalent to one data bit.

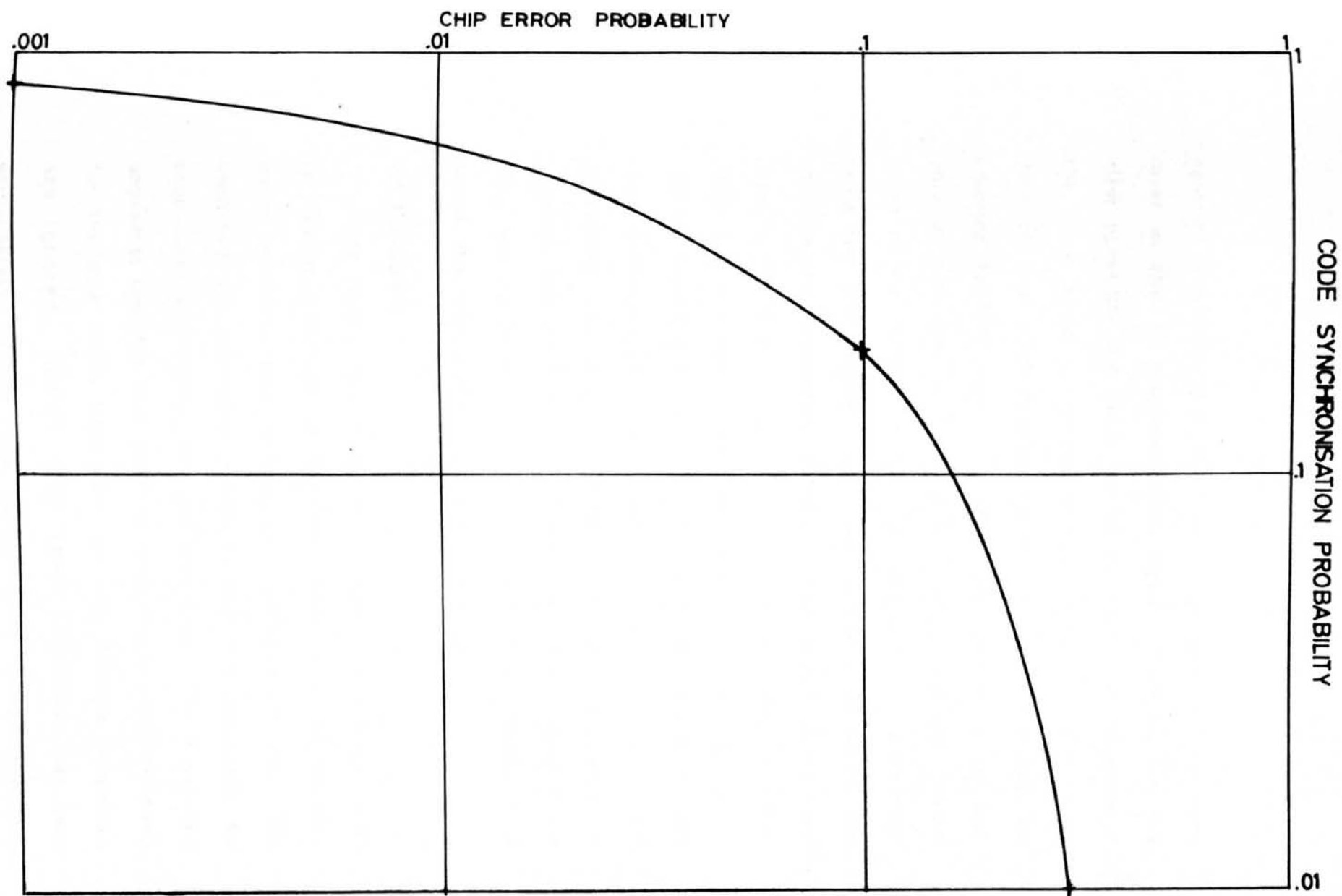
(a) Signal without errors

Under ideal operation conditions (without data perturbation) the worst case ($=127$) average synchronisation time was estimated to be 1.032 msec., this employs a sequence clock during acquisition of more than 100 Kchips/sec result in corresponding data rate of approximately 1K baud, i.e. a processing gain of 20 dB. It can be expected that system performance will be identical to that predicted in the 2901 system simulation. The minimum value that λ can have was around $L/3$, below which the probability of correct detection was very low. For λ equal to $L/2$ it can be expected that a performance improvement ratio of about 2 to 1

will result. After acquisition, and for the case where the transmitter and receiver clocks were derived from the same generator, the receiver was capable of despreading a PN signal of data rate 3.9K baud corresponding to a code rate of 500 Kchips/sec which is the maximum transmitting clock rate. For the case where the delay-lock tracking loop was operating, this rate reduces to 200 Kchips/sec result in corresponding data rate of 1.5K baud.

(b) Signal with errors

Data synchronisation was described above with the assumption that there would be no errors in the transmitted signal as received. Now the transmitter was supplying the receiver with a binary PN signal to which errors (noise) had been added by using the MC6803 noise simulator. The performance of the receiver when the signal was subjected to impulsive errors can be represented in the graph shown in Figure (7.3). The graph shows that if the error rate were as poor as 1×10^{-3} (one error every 1000 chips), the probability of achieving synchronisation was above 0.9. This probability did not drop to 0.5 until the error rate rises to 0.1 (one error every 10 chips). Synchronisation had failed completely when the error rate reached 0.3. Note that the peak detection threshold setting was at 63 (decimal), which corresponds to a very low false alarm probability, and the spreading ratio was 127. This type of noise can either be designed as a single pulse every few code chips or a group of error pulses in succession forming a burst of errors every data interval. Note also, that the pulses were considered to have



FIGURE(7.3) CODE SYNCHRONISATION VS. BIT(CHIP) ERROR PROBABILITY.

constant amplitude (TTL level) and in all cases have the same level as that of the transmitted signal. Therefore, the data error probability was mainly due to the duration of occurrence of the noise pulses. However, it was noticed that errors in the data bit and synchronisation loss were related (depend upon the tracking threshold level), if we assume that an error in one code chip or more does not mean an error in the data bit. Several cases of the interfering PN code were performed experimentally using the 2901 simulator, e.g. with different code lengths, and with maximal and nonmaximal codes. It was noticed that the worst case (synchronisation fail) can occur provided that the actual code and the interference are correlated, i.e., they are of exactly synchronised clocks. As a result of the relatively long time overhead which was spent for the generation of the interfering code using the MC6803, the data rate had to be reduced. This was a serious limitation (the nonequal speed of the 2901 and the MC6803) to examining the receiver performance at normal data rates subject to PN code interference.

7.5 Conclusion

This chapter has discussed the receiver performance, which is measured in terms of the synchronisation time and the BER, under real-time tests using a wire communication link. The important link parameters include the code rate and length, the code clock uncertainty, and the data rate. The theoretical acquisition time has been shown to agree closely with experiments for the case where false alarm and false dismissal probabilities are ignored. Indeed, quite rapid acquisition has been demonstrated in this case. Experimental results have been

presented using a 127-bit PN signal. The system is capable of acquiring synchronisation during an average time equal to 516 u.sec. This enables the system to use a spreading code with a processing gain of approximately 20 dB during synchronisation, and considerably larger than that during data transfer. A noise channel simulator, based on the MC6803 microprocessor, has been described and implemented to examine the receiver behaviour in the presence of an erroneous data environment.

CHAPTER 8

Conclusion

The emphasis of this work has been on the applications of bit-slice microprocessors to the design and implementation of the correlation process and other signal processing requirements in spread spectrum and other related communication applications. It is shown that those parts of the receiver which previously required large amounts of expensive analogue or discrete equipment can be realised at lower cost and with increased flexibility using all digital techniques.

Spectrum spreading is one of the most important tools that we have to prevent communications jamming. It can also be used for several other purposes: rejecting unintentional interference, lowering the probability of a transmission being intercepted by an unintended receiver, combating multipath problems, and providing multiple access to a communications system shared by a number of users.

Previously, spread spectrum systems were expensive and were therefore employed to a very limited extent only in areas where communications must be maintained in difficult environments, particularly in the presence of intentional interference. Because of the high cost of communication satellites links and susceptibility of military communications to jamming, spread spectrum techniques have been employed extensively in military satellite communications. Examples of these systems are notably NAVSTAR GPS (100), SKYNET (71), and others. Most of the above

uses have been handicapped by the difficulty and expense of implementation and the problems of synchronisation. During that time most spread spectrum systems were implemented using digital integrated circuits. In the analogue domain SAW and CCD devices have been introduced with the major advantage of these two technologies being the considerable speed that can be obtained compared to IC implementation.

The recent developments in VLSI devices and, in particular, the microprocessors have enabled substantial reductions to be made in both the size and the cost of new digital signal processing techniques. This may allow spread spectrum systems to be designed and implemented in small, powerful, functional blocks, which may alleviate many of the problems associated with present system applications. Recently microprocessors has been found to be efficient to implement the post-correlation signal processing- demodulation, detection and tracking, especially for low rate signals (101). Relatively little work has been published on the direct applications of microprocessors to the correlation process, this is because of their restricted bandwidths. However, little use is made of bit-slice microprocessors when compared to fixed-wordlength, fixed-instruction-set microprocessors, because their application is more complex and requires longer development periods. Spread spectrum bandwidth must be large to obtain a significant performance improvement. This means that the sequence rate must be fast and so very fast microprocessors will be required when they are used to perform spread spectrum correlations (code

acquisition). This problem has been exemplified in this thesis which also has described some of the methods used to overcome this problem.

The implementation described in this thesis demonstrates some of the advantages obtained by the use of bit-slice devices instead of fixed-wordlength, fixed-instruction-set microprocessors. These advantages include flexibility -wordlength is easily increased without loss of speed- and high-speed operation owing to the use of the bipolar technology and pipelining techniques.

A real-time binary communication system has been described in which bit-slice microprocessor may be assessed as to its suitability for implementing direct sequence spread spectrum techniques. Some considerable attention has been paid to the signal processing requirements of PN spread spectrum systems, for spectral analysis, code modulation, and demodulation. This has included investigations into fast transformation using microprocessor techniques, in addition to a study of the chirp-Z transformation using a charge coupled device. The flexibility and high throughput in computing the correlation functions, and defining the error signal, which was required to control the tracking mode, by software have been demonstrated. The experimental results which have presented using a 127-bit sequence length show that the 2901 based correlator system is efficient because it can be expanded to accomodate variations in the sequence length. The effects on synchronisation uncertainty due to clock drifts has been described and shown to be minimised

by using sampling techniques. The acquisition time has been analysed and a formula has been obtained which coincides with that obtained by Holmes and Chen (96). This has been shown to agree closely with experiments in the case where the false alarm and false dismissal probabilities are ignored. The receiver system is capable of achieving synchronisation during an average time equal to 512 u.sec which enables the system to use a spreading code with a processing gain of approximately 20 dB.

Although analogue devices using SAW and CCD technologies are finding new uses in spread spectrum communications, still digital signal processing has many advantages over alternative techniques. These advantages include higher reliability, insensitivity to temperature changes and component tolerances, greater accuracy and repeatability, and a higher level of flexibility because they are programmable.

We hope that this thesis has illustrated the potential of applications of VLSI technology to the implementation of effective, low-cost systems in the field of spread spectrum communications. Future work in the field of spread spectrum communications will take advantage of advances in VLSI technology and the large number of signal processing algorithms which have been developed in the last two years. The architectures of the latest microprocessors, ALU/register chips, and signal processing components are implementing more digital signal processing operations on the chip. Furthermore, these are allocating more chip area to interface buses for greater programming flexibility. An examples of these components are the recent Advanced Micro

Devices and TRW families products (46), (17). The availability of such digital devices at relatively low cost will undoubtedly increase the interest in developing a new microprogrammable processors which can be derived by wider horizontal microcodes word and employ more parallel ALUs. This will offer higher throughput in processors, but at the price of software complexity. The other alternatives which can be used are the parallel and pipelined procesor techniques which may offer speed advantages, but they are limited in flexibility.

APPENDIX A

REFERENCES

- (1) Shannon, C.E., "Communication in the Presence of Noise", Proceedings IRE, Vol.37, Jan.1949, pp.10-21.
- (2) Costas, J.P., "Poisson, Shannon, and Radio Amateur", Proceedings IRE, Vol.47, December 1959, pp.2058-2068.
- (3) Golomb, S.W., Shift Register Sequences, Holden-Day, Inc., 1967.
- (4) Lange, F.H., Correlation Techniques, Princeton, NJ:Van Nostrand, 1966.
- (5) Chan, C.R., Spread Spectrum Communications, "Applications and State-of-the-Art Equipments", AGARD-NATO Lecture Series NO.58, 1973, pp.(5-1)-(5-110).
- (6) Dixon, R.C., Spread Spectrum Techniques, IEEE Press, New York, 1976.
- (7) Dixon, R.C., Spread Spectrum Systems, John Wiley & Sons, Inc., 1976.
- (8) Scholtz, R.A., "The Origins of Spread Spectrum Communications", IEEE Transactions on Communications, Vol.COM-30, No.5, May 1982, pp.822-854.
- (9) Grant, P.M., "The Potential Application of Analogue Matched and Adaptive Filters in Spread Spectrum Communications", The Radio and Electronic Engineer, Vol.52, No.5, May 1982, pp.246-258.
- (10) MacWilliams, F.J., and N.J.Sloane, "Pseudo-Random Sequences and Arrays", Proceedings of the IEEE, Vol.64, No.12, December 1976, pp.1715-1730.
- (11) Sarwate, D.V., and M.B.Pursley, "Crosscorrelation Properties of Pseudorandom and Related Sequences", Proceedings of the IEEE, Vol.68, No.5, May 1980, pp.593-619.

- (12) IEEE Transactions on Communications, Special Issue on Spread Spectrum Systems, August 1977.
- (13) Rappaport, S.S., "On Practical Setting of Detection Thresholds", Proceedings of the IEEE, Vol.57, August 1969, pp.1420-1421.
- (14) Bair, W.P., K.Dostert, and M.Pandit, "A novel, Spread Spectrum Receiver Synchronisation Scheme Using a SAW-Tapped Delay Line", IEEE Transactions on Communications, Vol.COM-30, No.5, May 1982, pp.1037-1047.
- (15) Bair, W.P., M.Pandit, and H.Grammuller, "Combined Acquisition and Fine Synchronisation System For Spread Spectrum Receivers Using A Tapped Delay Line Correlator", AGARD Conference Proceedings No.230, June 1978, pp.5.9.1-5.9.12.
- (16) Ward, R.B., and K.P.Yiu, "Acquisition of Pseudonoise Signals by Recursion-Aided Sequential Estimation", IEEE Transactions on Communications, Vol.COM-25, August 1977, pp.784-794.
- (17) "TDC1023J Monolithic Digital Correlator" Preliminary Information, TRW LSI Products, TRW Inc., 1980.
- (18) Cooley, J.W., and J.W.Tukey, "An Algorithm for Machine Calculation of Complex Fourier Series", Mathematics of Computation, Vol.19, No.9, 1965, pp.297-301.
- (19) Cooley, J.W., P.A.W.Lews, and P.D.Welsh, "Application of the Fast Fourier Transform to Computation of Fourier Integrals, Fourier series, and Convolution Integrals", IEEE Transactions on Audio and Electroacoustics, Vol.AU-15, June 1967, pp.79-84.
- (20) Winograd, S., "Some Bilinear Forms Whose Multiplicative Complexity Depends on the Field of Constants", IBM T.J., Watson Res. Ctr., IBM Res. Rep., NY, RC 5669, Oct.1975.

- (21) Winograd, S., "On computing the Discrete Fourier Transform", Proc.Nat.Acad.Sci., U.S.A, Vol.73, April 1976, pp.1005-1006.
- (22) Agarwal, R.C., and C.S.Burrus, "Number Theoretic Transforms to Implement Fast Digital Convolution" Proceedings of the IEEE, Vol.63, No.4, April 1975, pp.550-560.
- (23) Agarwal, R.C., and J.W.Cooley, "New Algorithms for Digital Convolution", IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol.ASSP-25, No.5, October 1977, pp392-410.
- (24) Brigham, E.O., The Fast Fourier Transform, Prentice-Hall, Inc., New Jersey, 1974.
- (25) Gold, B., and C.M.Rader, Digital Processing of Signals, McGraw-Hill, New York, 1969.
- (26) Silverman, H.F., "An Introduction to Programming the Winograd Fourier Transform Algorithm (WFTA)", IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol.ASSP-25, No.2, April 1977, pp.152-165.
- (27) Rabiner, L.R., and B.Gold, Theory and Applications of Digital Signal Processing, Printice-Hall, 1975, pp.419-433.
- (28) Agarwal, R.C., and C.S.Burrus, "Fast Convolution Using Fermat Number Transforms with Application to Digital Filtering", IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol.ASSP-22, No.2, April 1974, pp.87-97.
- (29) Martin, S.C., and B.J.Stanier, "Microprocessor Implementation of Number Theoretic Transforms", Electronic Circuits and Systems, Vol.3, No.1, January 1979, pp.21-26.
- (30) Moore, C.H., "FORTH: A New Way to Program A Mini-computer", Astron. Astrophys. Suppl.15, 1974, pp.497-511.
- (31) "microFORTH PRIMER", FORTH, Inc., Manhattan Beach, CA,

August 1978.

- (32) Leventhal, L.A., 8080/8085 Assembly Language Programming, Adam Osborne & Associates, Inc., California 1978.
- (33) Burns, R., and D.Sauitt,"Microprogramming and Stack Architecture Ease the Minicomputer Programmer's Burden", Electronics, Vol.46, 15 February 1973.
- (34) Texas Instruments, TMS-9900 System Development Manual, Texas Instruments Incorporated, 1976.
- (35) Mick, J., and J.Brick, Bit-Slice Microprocessor Design, McGraw-Hill Book Company, 1980.
- (36) Broderson, R.W., C.R.Hewes, and D.D.Bass,"A 500-stage CCD Transversal Filter for Spectral Analysis", IEEE Journal of Solid-State Circuits, Vol.SC-11, No.1, February 1976, pp.75-84.
- (37) Rabiner, L.R., R.W.Schafer, and C.M.Rader,"The Chirp Z-Transform Algorithm", IEEE Transactions On Audio & Electroacoustic, Vol.AU-17, June 1969, pp.86-92.
- (38) Benjamin, R.,"Real-time Spectrum Analysis Using Hardware Fourier and Chirp-Z Transasformation", The Radio & Electronic Engineer, Vol.49, No.2, February 1979, pp.101-107.
- (39) Buss, D.D., R.L.Veenkant, R.W.Broderson, and C.R.Hewes,"Comparison Between the CCD CZT and the Digital FFT", in Proc.Int.Conf.Applications of Charge-Coupled Devices, San Diago, CA, Oct.1975, pp.267-281.
- (40) Kosonocky, W.F., and J.Saver,"The ABCs of CCDs", Electronic Devices, Vol.23, April 1975, pp.58-63.
- (41) Hewes, C.R., R.W.Broderson, and D.D.Buss,"Applications of CCD and Switched Capacitor Filter Technology ", Proceedings of the IEEE, Vol.67, No.10, October 1979, pp.1403-1415.

- (42) White, D.E., Bit-Slice Design: Controllers and ALUs. New York: Garland STPM Press, 1981.
- (43) The Am2900 Family Data Book. Advanced Micro Devices Inc., Sunnyvale, California, 1979.
- (44) Myers, G.J., Digital System Design with LSI Bit-Slice Logic. New York: John Wiley & Sons, Inc., 1980.
- (45) Kraft, George D., and Wing N.Toy, Mini/Microcomputer Hardware Design: Bell Telephone Laboratories Inc., 1979.
- (46) Bipolar Microprocessor Logic and Interface Data Book. Advanced Micro Devices Inc., Sunnyvale, California, 1981.
- (47) Gibson, Glenn A., and Yu-Cheng Liu, Microcomputers for Engineers and Scientists, Prentice-Hall International, Inc., Englewood Cliffs, N.J., 1980.
- (48) MOS and Bipolar ROM/PROM. Signetics Corporation, Croydon, Surrey, 1975.
- (49) Schottky and Low-Power Schottky Data Book, Advanced Micro Devices, Inc., Sunnyvale, California, 1977.
- (50) The TTL Data Book for Design Engineers, Texas Instruments, U.S, 1980.
- (51) Peatman, John B., Microcomputer-Based Design, McGraw-Hill, Inc., Tokyo, 1977.
- (52) Artwick, Bruce A., Microcomputer Interfacing. New Jersey: Prentice-Hall, Inc., 1980.
- (53) Bipolar Memory Data Book, Fairchild Camera & Instrument Corporation, Mountain View, California, 1979.
- (54) Kraft, George D., and Wing N.Toy, Microprogrammed Control and Reliable Design of Small Computers, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1981.

- (55) Agrawala, A.K., and T.G.Rauscher, Foundations of Microprogramming: Architecture, Software and Applications. Acadimec Press, New York, 1976.
- (56) Rauscher, T.G., and P.M.Adams, "Microprogramming: A Tutorial and Survey of Recent Developments", IEEE Transactions on Computers, Vol.C-29, No.1, January 1980, pp.2-20.
- (57) Andrews, M., "A Bit-Slice Architecture for Microprogrammable Machines", SIGMICRO Newsletters, Vol.7, September 1976, pp.5-8.
- (58) Lau, S.Y., "Bit-Slice Microprogramming Saves Software Compatibility", EDN, Vol-23, Mar 5, 1978, pp.42-46.
- (59) idem, "Bit-Slice Microprogramming Saves Software Compatibility," EDN, Vol-25, Mar 20, 1978, pp.68-74.
- (60) Opler, A., "Fourth-Generation Software", Datamation, Vol.13, January 1967, pp.22-24.
- (61) DesRochers, G., "Microprogramming Helps Squeeze More from your Equipment Dollar", EDN, Sept 20, 1976, pp.102-105.
- (62) Agrawala, A.K., and T.G.Rauscher, "Microprogramming: Perspective and Status ", IEEE Transactions on Computers, Vol.C-23, No.8, August 1974, pp817-837.
- (63) Redfield, S.R., "A Study in Microprogrammed Processors: A Medium Sized Microprogrammed Processor", IEEE Transactions on Computers, Vol.C-20, No.7, July 1971, pp.743-750.
- (64) Webb, J.T., Coral 66 Programming, Manchester, NCC Publications, 1978.
- (65) Richie, D.M., and K.Thompson, "The UNIX Time-Sharing System", The Bell System Technical Journal, Vol.57, No.6, July-August 1978, pp.1905-1929.
- (66) Powers, V.M., and J.H.Hernandez, "Microprogram Assemblers for

- Bit-Slice Microprocessors", Computer, Vol.11, No.7, 1978, pp.108-120.
- (67) Spracklen, C.T., ASWE Serial Highway Simulator, Report, Durham University, 1978.
- (68) PRO LOG M900/920 PROM Programmer, Computer Interface Handbook, PRO-LOG Corporation, Monterey, California, 1978.
- (69) Leventhal, L.A., 6809 Assembly Language Programming, Osborne/McGraw-Hill, California, 1981.
- (70) Texas Instruments, The Optoelectronics Data for Design Engineers, Texas, 1976.
- (71) "Spread Spectrum Communications," AGARD Lecture Series no.58, National Technical Information Services AD-766-914, July 1973.
- (72) Golomb, S.A., Digital Communications: with Space Applications, Prentice-Hall, Inc., Englewood Cliffs, N.J, 1964.
- (73) National Semiconductor, Television /Radio, Santa Clara, California, 1978.
- (74) Cumming, I.G., "Autocorrelation Function and Spectrum of a Filtered Pseudorandom Binary Sequences", IEEE Transactions on Computers, Vol.C-20, No.3, March 1971, pp.270-281.
- (75) Knuth, D.E., The Art of Computer Programming, vol.2, Addison-Wesley, Reading, Mass, 1980.
- (76) Camp, Warren V., and T.G.Lewis, "Implementing a Pseudorandom Number Generator on a Minicomputer", IEEE Transactions on Software Engineering, Vol.SE-3, No.3, May 1977, pp.259-262.
- (77) Ward, R.B., "Digital Communications on a Pseudonoise Tracking Link Using Sequence Inversion Modulation", IEEE Transactions on Communication Technology, Vol.Com-15, No.1, February 1967,

pp.69-78.

(78) Grieco, D.M., "The Application of CCD's to Spread Spectrum Systems", IEEE Transactions on Communications, Vol.COM-28, No.9, September, 1980.

(79) Alem, W.K., and C.L.Weber, "Acquisition Techniques of PN Sequences", NTC'77 Conference Record, pp.35:2-1 - 35:2-4.

(80) Sage, G.F., "Serial Synchronisation of Pseudonoise Systems", IEEE Transactions on Communication Technology, 1964, pp.123-127.

(81) Lindholm, J.H., "An Analysis of Pseudo-Randomness Properties of Subsequences of Long m-sequences", IEEE Transactions on Information Theory, Vol.IT-14, No.4, July 1968. pp.569-576.

(82) Hartmann, H.P., "Analysis of Dithering Loop for PN Code Tracking", IEEE Transactions on Aerospace and Electronic Systems, Vol. AES-10, January 1974, pp.2-9.

(83) Spilker, J.J., "Delay-lock Tracking of Binary Signals", IEEE Transactions on Space Electronics and Telemetry, March 1963, pp.1-8.

(84) Gill, W.J., "A Comparison of Binary Delay-lock Tracking Loop Implementations", IEEE Transactions on Aerospace and Electronic Systems, Vol.AES-2, No.4, July 1966, pp.415-424.

(85) Davies, A.C., and Al-Rawas, "Error-Signal Generation for Pseudonoise Tracking Loop", Electronic Circuit and Systems, Vol.2, No.6, November 1978, pp.189-192.

(86)idem, "Synchronisation of a Spread Spectrum Receiver by a Microprocessor control system", The Radio and Electronic Engineer, Vol.49, No.6, June 1979, pp.306-310.

(87) Holmes, J.K., Coherent Spread Spectrum Systems, John Wiley & Sons, Inc., New York, 1982.

- (88) "Spectrum Analysis" Hewlett-Packard Journal, July 1964.
- (89) Cheung, R.P., J.Hovey, L.N.Ma, and T.J.Stephens,"LSI Digital Correlation Detector", NAECON'74 Record, 1974, pp.617-622.
- (90) Hopkins, P.M., "A Unified Analysis of Pseudonoise Synchronisation by Envelope Correlation", IEEE Transactions on Communications, Vol.COM-25, No.8, August 1977, pp.770-778.
- (91) Stremler, F.G., Introduction to Communication Systems, Addison-Wesley, Reading MA, 1977.
- (92) Lindsey, W.C., and M.K.Simon, Telecommunication Systems Engineering, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973.
- (93) Davies, N.G., "Performance and Synchronisation Considerations", 'Spread Spectrum Communications', AGARD Lecture series No.58, National Information Services AD-766-914, July 1973, PP.(4-1)-(4-24).
- (94) Hopkins, P.M., and R.S.Simpson, "Probability of Error in Pseudonoise (PN)- Modulated Spread Spectrum Binary Communication Systems", IEEE Transactions on Communications, Vol.COM-23, No.4, April 1975, pp.467-472.
- (95) German, E.H., "A comment on: Probability of Error in PN-Modulated Spread Spectrum Binary Communications Systems", IEEE Transactions on Communications, Vol.COM-26, No.6, June 1978, pp.932-934.
- (96) Holmes, J.K., and C.C.Chen, "Acquisition Time Performance of PN Spread Spectrum Systems", IEEE Transactions on Communications, Vol.COM-25, No.8, August 1977, pp.778-784.
- (97) 'Motorola MC6803 Advance Information', Motorola Publications, 1979.

- (98) 'Motorola M6800 Microprocessor Programming Manual', Motorola Publications, 1978.
- (99) Jain, V.K., and S.N.Gupta,"Digital Communication Systems in Impulsive Atmospheric Radio Noise", IEEE Transactions on Aerospace and Electronic Systems, Vol.AES-15, No-2, March 1979, pp.228-236.
- (100) Blair, P.K.,"Receivers for the NAVSTAR Global Positioning System", IEE Proc., Vol.127, Part-F, No.2, April 1980, pp.163-167.
- (101) Cahn,C.R., D.K.Leimer, C.L.Marsh, F.J.Huntowski, and G.D.Larve,"Software Implementation of Spread Spectrum Receiver to Accommodate Dynamics", IEEE Transactions on Communications, Vol.COM-25, No.8, August 1977, pp.832-840.

APPENDIX B

PROGRAM LISTINGS

Listings for chapter 2

- (1) FORTRAN optimal short Rectangular Transforms
- (2) FORTH digital correlation using Intel 8080 system
- (3) FORTH digital correlation using TMS9900

APPENDIX B

OPTIMAL SHORT CORRELATION ..N=2
USING RECTANGULAR TRANSFORM ALGORITHM

```
INTEGER A(2), B(2), M(2), Y(2), X(2), F(2)
DATA X, H /0, 2, 6, 2/
M16(IVAL) = MOD(IVAL,65536)
```

A.H TRANSFORMATION TRANSFORMS H-SEQUENCES TO RECT.ARRAY

```
A(1) = M16((H(1) + H(2))/2)
A(2) = M16((H(1) - H(2))/2)
```

B.X TRANSFORMATION TRANSFORMS X-SEQUENCES TO RECT.ARRAY

```
B(1) = M16(X(1) + X(2))
B(2) = M16(X(1) - X(2))
```

CORRELATION OPERATION IN THE ORIGINAL DOMAIN BECOMES
ELEMENT-BY-ELEMENT MULTIPLICATION IN THE TRANSFORM DOMAIN

```
DO 10 K = 1, 2
  M(K) = M16(A(K)*B(K))
10 CONTINUE
OPER. C DENOTES THE INVERSE RECT. TRANSFORM THIS IS
REPRESENTED BY C (A.H 'X'B.X)
Y(1) = M16(M(1) + M(2))
Y(2) = M16(M(1) - M(2))
WRITE (6,20) Y
20 FORMAT ('Y(1)=', I3, 20X, 'Y(2)=', I3)
STOP
END
```

OPTIMAL SHORT CORRELATION ..N=3
USING RECTANGULAR TRANSFORM ALGORITHM

```
INTEGER X(3), H(3), A(4), B(4), Y(3), M(4), HF, HF1
M16(IVAL) = MOD(IVAL,65536)
```

```
DATA X, H /1, 2, 3, 5, 0, 3/
```

```
HF = M16(H(1) + H(2) + H(3))
```

```
INV3 = 43691
```

* * A-ELEMENTS * *

```
A(1) = M16(HF*INV3)
```

```
A(2) = M16(H(1) - H(2))
```

```
A(3) = M16(H(3) - H(2))
```

```
HF1 = M16((H(1) - H(2)) + (H(3) - H(2)))
```

```
A(4) = M16(HF1*INV3)
```

* * B-ELEMENTS * *

```
B(1) = M16(X(1) + X(2) + X(3))
```

```
B(2) = M16(X(1) - X(3))
```

```
B(3) = M16(X(2) - X(3))
```

```
B(4) = M16((X(1) - X(3)) + (X(2) - X(3)))
```

```
DO 10 K = 1, 4
```

```
  M(K) = M16(A(K)*B(K))
```

```
10 CONTINUE
```

APPENDIX B

```

Y(1) = M16(M(1) + (M(2) - M(4)))
Y(2) = M16(M(1) - (M(2) - M(4)) - (M(3) - M(4)))
Y(3) = M16(M(1) + (M(3) - M(4)))
WRITE (6,20) (K,Y(K),K=1,3)
20 FORMAT (T5, 3('Y(',I1,'')=',I7,2X))
STOP
END
*****
OPTIMAL SHORT CORRELATION .. N=4
USING RECTANGULAR TRANSFORM ALGORITHM

DIMENSION X(4), H(4), A(5), B(5), Y(4)
REAL M(5)
DATA X, H /2., 4., 1., 3., 4., 5., 0., 1./
A-ELEMENTS
A(1) = ((H(1) + H(3)) + (H(4) + H(2))) / 4.
A(2) = ((H(1) + H(3)) - (H(4) + H(2))) / 4.
A(3) = (H(1) - H(3)) / 2.
A(4) = ((H(1) - H(3)) - (H(4) - H(2))) / 2.
A(5) = ((H(1) - H(3)) + (H(4) - H(2))) / 2.
B-ELEMENTS
B(1) = (X(1) + X(3)) + (X(2) + X(4))
B(2) = (X(1) + X(3)) - (X(2) + X(4))
B(3) = (X(1) - X(3)) + (X(2) - X(4))
B(4) = X(1) - X(3)
B(5) = X(2) - X(4)

DO 10 K = 1, 5
  M(K) = A(K) * B(K)
10 CONTINUE

Y(1) = (M(1) + M(2)) + (M(3) - M(5))
Y(2) = (M(1) - M(2)) + (M(3) - M(4))
Y(3) = (M(1) + M(2)) - (M(3) - M(5))
Y(4) = (M(1) - M(2)) - (M(3) - M(4))
20 WRITE (6,20) (K,Y(K),K=1,4)
20 FORMAT (T5, 4('Y(',I1,'')=',F4.0,2X))
STOP
END
*****
OPTIMAL CORRELATION ALGORITHM . .N=5
USING RECTANGULAR TRANSFORM

INTEGER A(10), B(10), M(10), X(5), H(5), Y(5), HF, HF1
M16(IVAL) = MOD(IVAL,65536)
DATA X, H /1, 2, 3, 4, 5, 5, 0, 3, 1, 4/
HF = M16(H(1) + H(2) + H(3) + H(4) + H(5))
INV5 = 52429

COMPUTE THE A-ELEMENTS

A(1) = M16(HF*INV5)
A(2) = M16(H(1) - H(2))

```

APPENDIX B

```

A(3) = M16(H(5) - H(2))
A(4) = M16(H(4) - H(2))
A(5) = M16(H(3) - H(2))
A(6) = M16((H(1) - H(2)) + (H(5) - H(2)))
A(7) = M16((H(4) - H(2)) + (H(3) - H(2)))
A(8) = M16((H(1) - H(2)) + (H(4) - H(2)))
A(9) = M16((H(5) - H(2)) + (H(3) - H(2)))
HF1 = M16((H(1) - H(2)) + (H(5) - H(2)) + ((H(4) - H(2)) + (H(3)
1 - H(2))))
A(10) = M16(HF1*INV5)

```

COMPUTE THE B-ELEMENTS

```

B(1) = M16(X(1) + X(2) + X(3) + X(4) + X(5))
B(2) = M16(X(1) - X(5))
B(3) = M16(X(2) - X(5))
B(4) = M16(X(3) - X(5))
B(5) = M16(X(4) - X(5))
B(6) = M16((X(1) - X(5)) + (X(2) - X(5)))
B(7) = M16((X(3) - X(5)) + (X(4) - X(5)))
B(8) = M16((X(1) - X(5)) + (X(3) - X(5)))
B(9) = M16((X(2) - X(5)) + (X(4) - X(5)))
B(10) = M16((X(1) - X(5)) + (X(2) - X(5)) + ((X(3) - X(5)) + (X(
14) - X(5))))

```

DO 10 K = 1, 10

M(K) = M16(A(K)*B(K))

10 CONTINUE

```

Y(1) = M16((M(1) - M(10)) + (M(2) - M(5)) - M(4) + M(7))
Y(2) = M16((M(1) - M(10)) - (M(2) - M(5)) - M(3) + M(6))
Y(3) = M16((M(1) - M(10)) + (M(3) - M(4)) - M(2) + M(9))
Y(5) = M16((M(1) - M(10)) - (M(3) - M(4)) - M(5) + M(9))
Y(4) = M16((M(1) + M(1)) + 2*M(1) + M(1) - Y(1) - Y(2) - Y(3) - Y(
15))

```

WRITE (6,20) (K,Y(K),K=1,5)

20 FORMAT (T5, 5('Y(',I1,'')=',I7,2X))

STOP

END

OPTIMAL SHORT CORRELATION ALGORITHM USING RECT. TRANS.

FOR REAL DATA SEQUENCE ..N=6

DIMENSION H(6), X(6), Y(6), A(8), B(8)

REAL M(8)

DATA X, H /3., 0., 6., 3., 0., 0., 6., 9., 3., 9., 0., 3./

A(1) = ((H(1) - H(5)) + (H(4) - H(2))) / 6.

A(2) = ((H(6) - H(5)) + (H(3) - H(2))) / 6.

A(3) = A(2) - A(1)

A(4) = ((H(1) - H(5)) - (H(4) - H(2))) / 6.

A(5) = ((H(6) + H(5)) - (H(3) + H(2))) / 6.

A(6) = A(4) + A(5)

A(7) = ((H(1) + H(5)) - (H(6) + H(4)) + (H(3) - H(2))) / 6.

APPENDIX B

```
A(8) = ((H(1) + H(5)) + (H(6) + H(4)) + (H(3) + H(2))) / 6.
```

```
B(1) = (X(1) - X(3)) + (X(4) - X(6))
```

```
B(2) = (X(2) - X(3)) + (X(5) - X(6))
```

```
B(3) = B(1) - B(2)
```

```
B(4) = (X(1) - X(3)) - (X(4) - X(6))
```

```
B(5) = (X(2) + X(3)) - (X(5) + X(6))
```

```
B(6) = B(4) + B(5)
```

```
B(7) = (X(1) + X(3)) - (X(2) + X(4)) + (X(5) - X(6))
```

```
B(8) = (X(1) + X(3)) + (X(2) + X(4)) + (X(5) + X(6))
```

```
DO 10 K = 1, 8
```

```
    M(K) = A(K) * B(K)
```

```
10 CONTINUE
```

```
Y(1) = ((M(1) - M(2)) - (M(2) + M(3))) + ((M(4) - M(5)) - (M(5) -
```

```
1 M(6))) + (M(7) + M(8))
```

```
Y(2) = ((M(1) + M(3)) + (M(2) + M(3))) - ((M(4) - M(6)) + (M(5) -
```

```
1 M(6))) - (M(7) - M(8))
```

```
Y(3) = -((M(1) - M(2)) + (M(1) + M(3))) - ((M(4) - M(5)) + (M(4) -
```

```
1 M(6))) + (M(7) + M(8))
```

```
Y(4) = ((M(1) - M(2)) - (M(2) + M(3))) - ((M(4) - M(5)) - (M(5) -
```

```
1 M(6))) - (M(7) - M(8))
```

```
Y(5) = ((M(1) + M(3)) + (M(2) + M(3))) + ((M(4) - M(6)) + (M(5) -
```

```
1 M(6))) + (M(7) + M(8))
```

```
Y(6) = -((M(1) - M(2)) + (M(1) + M(3))) + ((M(4) - M(5)) + (M(4) -
```

```
1 M(6))) - (M(7) - M(8))
```

```
WRITE (6,20) (K,Y(K),K=1,6)
```

```
20 FORMAT (T5, 6('Y(',I1,',')= ',F4.0,2X))
```

```
STOP
```

```
END
```

```
*****
```

```
OPTIMAL SHORT CORRELATION ..N=7
```

```
SING RECTANGULAR TRANSFORM ALGORITHM
```

```
INTEGER H(7), X(7), A(19), B(19), U(8), Y(7), M(19), HA
```

```
M16(IVAL) = MOD(IVAL,65536)
```

```
DATA X, H /4, 5, 2, 0, 6, 9, 0, 6, 0, 8, 4, 3, 0, 1/
```

```
4-ELEMENTS
```

```
A(2) = M16(H(1) - H(2))
```

```
A(3) = M16(H(7) - H(2))
```

```
A(4) = M16(H(6) - H(2))
```

```
A(5) = M16(H(5) - H(2))
```

```
A(6) = M16(H(4) - H(2))
```

```
A(7) = M16(H(3) - H(2))
```

```
A(8) = M16(A(2) + A(5))
```

```
A(9) = M16(A(3) + A(6))
```

```
A(10) = M16(A(4) + A(7))
```

```
A(11) = M16(A(2) + A(3))
```

```
A(12) = M16(A(3) + A(4))
```

```
A(13) = M16(A(2) + A(4))
```

```
A(14) = M16(A(5) + A(6))
```

```
A(15) = M16(A(6) + A(7))
```

APPENDIX B

```

A(16) = M16(A(5) + A(7))
A(17) = M16(A(11) + A(14))
A(18) = M16(A(12) + A(15))
HA = M16(A(8) + A(19))
INV7 = 28087
A(19) = MOD(HA*INV7,65536)
A(1) = M16(A(19) + H(2))

```

ELEMENTS

```

B(2) = M16(X(1) - X(7))
B(3) = M16(X(2) - X(7))
B(4) = M16(X(3) - X(7))
B(5) = M16(X(4) - X(7))
B(6) = M16(X(5) - X(7))
B(7) = M16(X(6) - X(7))
B(8) = M16(B(2) + B(5))
B(9) = M16(B(3) + B(6))
B(10) = M16(B(4) + B(7))
B(11) = M16(B(2) + B(3))
B(12) = M16(B(3) + B(4))
B(13) = M16(B(2) + B(4))
B(14) = M16(B(5) + B(6))
B(15) = M16(B(6) + B(7))
B(16) = M16(B(5) + B(7))
B(17) = M16(B(11) + B(14))
B(18) = M16(B(12) + B(15))
B(19) = M16(B(8) + B(18))
B(1) = M16(B(19) + X(7) + ((X(7) + X(7)) + 2*X(7)) + 2*X(7))

```

ELEMENT-BY-ELEMENT MULT.

```

DO 10 K = 1, 19
  M(K) = M16(A(K)*B(K))
10 CONTINUE

U(1) = M16(M(1) - M(19))
U(2) = M16(M(2) - M(6))
U(3) = M16(M(5) + M(7))
U(4) = M16(M(2) + M(4))
U(5) = M16(M(3) - M(7))
U(6) = M16(M(3) + M(4) + M(5) + M(6) - M(9))
U(7) = M16(U(1) - U(4))
U(8) = M16(U(1) + U(6))
Y(1) = M16(U(1) + U(2) - U(3) - M(4) + M(10) + M(14))
Y(2) = M16(U(1) - U(2) - U(3) - M(3) + M(11) + M(16))
Y(3) = M16(U(7) + U(5) - M(6) + M(13) + M(15))
Y(4) = M16(U(7) - U(5) - M(5) + M(8) + M(12))
Y(5) = M16(U(8) + M(2) - M(8) - M(11) - M(14) + M(17))
Y(7) = M16(U(8) + M(7) - M(10) - M(12) - M(15) + M(18))
Y(6) = M16((M(1) + M(1)) + (2*M(1) + 2*M(1)) + M(1) - Y(1) - Y(2))
1- Y(3) - Y(4) - Y(5) - Y(7))
WRITE (6,20) (K,Y(K),K=1,7)
20 FORMAT (T5, 7('Y(',I1,')=',I7,2X))
STOP

```

APPENDIX B

END

 ONE-TO-ONE MAPPING USING CHINESE REMAINDER THEOREM (CRT)

```

INTEGER X(15), Y(15)
INTEGER XX(5,3)
DATA X /1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15/
DO 20 I1 = 1, 3
  DO 10 I2 = 1, 5
    II = 10 * I1 + 6 * I2 - 1
    XX(I2,I1) = X(MOD(II,15) + 1)
  
```

INVERSE CRT

```

      Y(MOD(II,15) + 1) = XX(I2,I1)
10  CONTINUE
20  CONTINUE

WRITE (6,30) ((XX(I,J),J=1,3),I=1,5)
WRITE (6,30) Y
30  FORMAT (T4, 3(10X,I4))
STOP
END

```

 FAST CORRELATION USING RECTANGULAR TRANSFORM
 TWO- FACTOR ALGORITHM . . N=3*5

```

INTEGER X(15), H(15), Y(15), XX(5,3), HH(5,3), YY(10,3)
INTEGER A1(10,3), B1(10,3), A2(10,4), B2(10,4), A3(10,4)
INTEGER A1F, A1F1, A2F, A2F1, Z1(5), V1(5), Z2(3), V2(3)
INTEGER W(4), WW(10), Y1(5,3)
M16(IVAL) = MOD(IVAL,65536)
DATA X /4, 3, 2, 1, 2, 3, 4, 3, 2, 1, 2, 3, 4, 3, 2/
DATA H /1, 2, 3, 2, 1, 1, 2, 3, 2, 1, 1, 2, 3, 2, 1/

```

ONE-TO-ONE MAPPING USING C.R.T

```

DO 20 I1 = 1, 3
  DO 10 I2 = 1, 5
    II = 10 * I1 + 6 * I2 - 1
    XX(I2,I1) = X(MOD(II,15) + 1)
    HH(I2,I1) = H(MOD(II,15) + 1)
  
```

10 CONTINUE
 20 CONTINUE

CORRELATION OF COLUMNS . . APPLICATION OF RECT. TRANSF.
 ALGORITHM TO THE 5-POINT COLUMN CORRELATION
 THIS GIVES A1 * H
 INV5 = 52429
 DO 40 JJ = 1, 3
 DO 30 II = 1, 5
 Z1(II) = HH(II,JJ)
 V1(II) = XX(II,JJ)

30 CONTINUE

APPENDIX 3

FIRST A-ARRAY . . (A1 * H)

```

A1F = M16(Z1(1) + Z1(2) + Z1(3) + Z1(4) + Z1(5))
A1(1,JJ) = M16(A1F*INV5)
JT1 = M16(Z1(1) - Z1(2))
JT2 = M16(Z1(5) - Z1(2))
JT3 = M16(Z1(4) - Z1(2))
JT4 = M16(Z1(3) - Z1(2))
A1(2,JJ) = JT1
A1(3,JJ) = JT2
A1(4,JJ) = JT3
A1(5,JJ) = JT4
A1(6,JJ) = M16(JT1 + JT2)
A1(7,JJ) = M16(JT3 + JT4)
A1(8,JJ) = M16(JT1 + JT3)
A1(9,JJ) = M16(JT2 + JT4)
A1F1 = M16((JT1 + JT2) + (JT3 + JT4))
A1(10,JJ) = M16(A1F1*INV5)

```

FIRST B-ARRAY . . (B1 * X)

```

B1(1,JJ) = M16(V1(1) + V1(2) + V1(3) + V1(4) + V1(5))
NS1 = M16(V1(1) - V1(5))
NS2 = M16(V1(2) - V1(5))
NS3 = M16(V1(3) - V1(5))
NS4 = M16(V1(4) - V1(5))
B1(2,JJ) = NS1
B1(3,JJ) = NS2
B1(4,JJ) = NS3
B1(5,JJ) = NS4
B1(6,JJ) = M16(NS1 + NS2)
B1(7,JJ) = M16(NS3 + NS4)
B1(8,JJ) = M16(NS1 + NS3)
B1(9,JJ) = M16(NS2 + NS4)
B1(10,JJ) = M16((NS1 + NS2) + (NS3 + NS4))

```

CONTINUE

CORRELATION OF ROWS ..APPLICATION OF RECT. TRANSF.
 ALGORITHM TO THE 3-POINT ROW CORRELATION
 THIS GIVES .. A2(A1 * H) . .

```

DO 70 J = 1, 10
  DO 50 I = 1, 3
    Z2(I) = A1(J,I)
    V2(I) = B1(J,I)
  CONTINUE

```

```

INV3 = 43691
A2F = M16(Z2(1) + Z2(2) + Z2(3))
A2(J,1) = M16(A2F*INV3)
NT1 = M16(Z2(1) - Z2(2))
NT2 = M16(Z2(3) - Z2(2))
A2(J,2) = NT1
A2(J,3) = NT2
A2F1 = M16(NT1 + NT2)

```


APPENDIX B

```
A2(J,4) = M16(A2F1*INV3)
```

```
B2-ELEMENT. .B2(B1 * X)
```

```
B2(J,1) = M16(V2(1) + V2(2) + V2(3))
NSS1 = M16(V2(1) - V2(3))
NSS2 = M16(V2(2) - V2(3))
B2(J,2) = NSS1
B2(J,3) = NSS2
B2(J,4) = M16(NSS1 + NSS2)
```

```
ELEMENT-BY-ELEMENT MULT. (A2.A1.H X B2.B1.X)
```

```
DO 60 K = 1, 4
  W(K) = A2(J,K) * B2(J,K)
```

```
60 CONTINUE
```

```
OPER. C2 REDUCES THE DIMENSIONALITY OF (A2.A1.H X B2.B1.X)
```

```
MQ1 = M16(W(2) - W(4))
MQ2 = M16(W(3) - W(4))
YY(J,1) = M16(W(1) + MQ1)
YY(J,2) = M16(W(1) - MQ1 - MQ2)
YY(J,3) = M16(W(1) + MQ2)
```

```
70 CONTINUE
```

```
OPER. C1(C2.(A2 A1 H X B2 B1 X) REDUCES THE DIMENSIONALITY
OF THE COLUMNS
```

```
DO 90 KK = 1, 3
  DO 80 LL = 1, 10
    WW(LL) = YY(LL, KK)
```

```
80 CONTINUE
```

```
M1 = M16(WW(1) - WW(10))
M2 = M16(WW(2) - WW(5))
M3 = M16(WW(3) - WW(4))
Y1(1, KK) = M16(M1 + M2 - WW(4) + WW(7))
Y1(2, KK) = M16(M1 - M2 - WW(3) + WW(6))
Y1(3, KK) = M16(M1 + M3 - WW(2) + WW(8))
Y1(5, KK) = M16(M1 - M3 - WW(5) + WW(9))
NN = M16(2*WW(1))
Y1(4, KK) = M16(WW(1) + WW(1) + NN + WW(1) - Y1(1, KK) - Y1(2, KK)
1 - Y1(3, KK) - Y1(5, KK))
```

```
90 CONTINUE
```

```
ONE-TO-ONE MAPPING USING INVERSE C.R.T.
```

```
DO 110 J1 = 1, 3
  DO 100 J2 = 1, 5
    LLL = 10 * J1 + 6 * J2 - 1
    Y(MOD(LL, 15) + 1) = Y1(J2, J1)
```

```
100 CONTINUE
```

```
110 CONTINUE
```

```

( 100 POINTS DIGITAL CORRELATION USING DIRECT METHOD
  FOR INTEL 8080 WRITTEN IN FORTH PROGRAMMING LANGUAGE )
( */ IS USED TO MULTIPLY 16-BIT BY 16-BIT AND DIVIDE
  THE RESULT 32-BIT BY 16-BIT NUMBER )
OCTAL CODE HLDE*2 XCHG H DAD XCHG FC IF H DAD H INX
RET THEN H DAD RET
CODE DV 20 A MVI BEGIN PSW PUSH A XRA $ HLDE*2 CALL RAL
H PUSH B DAD 0 ACI 2 CPI $ QUESTION JNC RAR FC IF SP INX
SP INX 0 INX ELSE H POP THEN PSW POP A DCR FZ END RET
CODE */ $ PPD CALL 0 PUSH A 0 MOV A CRA FM IF TCD THEN
XCHG $ PPD CALL 0 PUSH A 0 MOV A CRA FM IF TCD THEN B 0
MOV C E MOV 0 0 LXI 20 A MVI BEGIN $ HLDE*2 CALL FC IF
XCHG B DAD XCHG FC IF H INX THEN THEN A DCR FZ END 0 PUSH
$ PPD CALL 0 PUSH A 0 MOV A CRA FM IF TCD THEN B 0 MOV
C E MOV 0 POP XCHG XTHL XCHG $ DV CALL H PUSH B DAD H POP
$ QUESTION JC A 0 MOV A CRA $ QUESTION JM B POP PSW POP
H POP H XRA FM IF 0 INX THEN B XRA FM IF TCD THEN
$ PSD JMP
( DEFINE THE VARIABLES AND/OR ARRAY USED . . . )
OCTAL 4 ARRAY ANSWER 7 CONSTANT STATUS 0 INTEGER DELAY
6 CONSTANT CONVERT 0 INTEGER MAXNB 0 INTEGER MINNB
200 CONSTANT N 400 CONSTANT NN NN ARRAY 1DATA N ARRAY 2DATA
NN ARRAY CORREL N ARRAY SINK 176 CONSTANT OPTION
( CLEAR TEMP STORE *ANSWER* . . )
: ZERO ANSWER EMPTY ;
( /* MULTIPLY AND STORE THE RESULT IN ANSWER 32 BIT )
OCTAL CODE /* $ PPD CALL $ PPH CALL A L MOV $ *ML CALL
XCHG ANSWER H LXI A M MOV E ADD M A MOV F INX A M MOV 0 ADD
M A MOV H INX 0 A MVI M ADD M A MOV A XRA H INX M A MOV RET
( ROUTINE INPUT GET DATA INTO ARRAY 1DATA - 2DATA . . )
CODE INPUT A XRA CONVERT CUT 14 A MVI CONVERT OUT XTHL XTHL
A XRA CONVERT OUT BEGIN STATUS IN 14 ANI 14 CPI FZ END 6 IN
CMA E A MOV 0 0 MVI $ PSD CALL 2 IN CMA E A MOV 0 0 MVI
$ PSD JMP
( STORE DATA IN 1DATA ARRAY & 2DATA ARRAY . . . )
: GETINPUT NN 0 DO INPUT I 1DATA + ? I DUP OPTION > IF 2DROP
ELSE 2DATA + ? THEN DELAY @ MSEC 2 +LOOP ;
( CORRELATION PART . . DIRECT METHOD )
: SHIFT + 1DATA + @ ;
: COEFFICIENT N 0 DO DUP I SHIFT I 2DATA + @ /* 2 +LOOP
DROP ;
: TRANSFER 4 0 DO DUP I ANSWER + @B SWAP ?B 1+ LOOP DROP ;
: XCORREL N 0 DO ZERO I COEFFICIENT I 2 * CORREL + TRANSFER
2 +LOOP ; : COMAX 0 NN 0 DO I 1+ CORREL + @ 2OVER < IF SWAP I
4 / LOCATION ? THEN DROP 4 +LOOP MAXNB ? ;
( ROUTINE COFACTOR IS SCALING ROUTINE . . )
: COFACTOR NN 0 DO MAXNB @ I 1+ CORREL + @ 377 /* I 2 /
SINK + ? 4 +LOOP ;
( DISPLAY CORRELATION FUNCTION . . )
CODE OUTPUT $ PPD CALL A E MOV CMA 7 OUT RET
: 2TEST 377 0 DO I OUTPUT 12 MSEC LOOP ;
: SCOPE BEGIN DUP LIMITS DO I @B OUTPUT 2 +LOOP 0 OUTPUT ?VDU
END 2DROP ;

```

APPENDIX 9

```

( FORMAT TO WRITE THE RESULT. .)
: TEST LOCATION @B 4 * CORREL + ;
: 1TEST 4 0 DO DLP I TEST + @B SWAP ?B 1+ LOOP DROP ;
: V STRING # CORREL IN BYTES # SAY CRLF CORREL PRINTB CRLF
STRING " MAXIMUM NUMBER = " SAY ANSWER 1TEST ANSWER PRINTB
CRLF STRING A LOCATION IS A SAY LOCATION ? CRLF
STRING " SINK IN BYTES " SAY CRLF SINK PRINT CRLF ;
( THIS VRSION USING SCALED INPUT DATA MAX. NO. =32 . .)
( ROUTINE *? IS NO LONGER USED IN THIS VERSION . .*/ )
DECIMAL 0 INTEGER ANSWER 38 CONSTANT OPTION 7 CONSTANT STATUS
6 CONSTANT CONVERT 0 INTEGER 10MAXN 0 INTEGER 20MAXN 0 INTEGER
COMAXN 40 CONSTANT N 80 CONSTANT NN 0 INTEGER COMIN
N ARRAY SINK N ARRAY CORREL NN ARRAY 10DATA N ARRAY 20DATA
CODE 1IN A XRA CONVERT OUT 1 A MVI CONVERT OUT XTHL XTHL A XRA
CONVERT OUT BEGIN STATUS IN 1 ANI FNZ END 6 IN CMA E A MOV 0 0
MVI $ PSD JMP CODE 2IN A XRA CONVERT OUT 1 A MVI CONVERT OUT
XTHL XTHL A XRA CONVERT OUT BEGIN STATUS IN 1 ANI FNZ END 6 IN
CMA E A MOV 0 0 MVI $ PSD JMP : 10GET NN 0 DC 1IN I 10DATA + ? 4
MSEC 2 +LOOP ; : 20GET N 0 DO 2IN I 20DATA + ? 4 MSEC 2 +LOOP ;
: GETDATA NN 0 DO 1IN I 10DATA + ? 2IN I DUP OPTION > IF 2DROP
ELSE 20DATA + ? THEN 4 MSEC 2 +LOOP ;
: MAXN 0 SWAP LIMITS DO I @ MAX 2 +LOOP ;
: 10LOC 10DATA MAXN 10MAXN ? ; : 20LOC 20DATA MAXN 20MAXN ? ;
( MAX. ELEMENT OF DATA SEQUENCE 32)
: 10FACTOR LIMITS DC 10MAXN @ I @ 32 */ I ? 2 +LOOP ;
: 20FACTOR LIMITS DC 10MAXN @ I @ 32 */ I ? 2 +LOOP ;
: T OPTION @ > ; : 10 + DUP T IF DROP 0 ELSE 20DATA + @ THEN ;
: POINT N 0 DO DUF I 10 I 20DATA + @ * ANSWER +? 2 +LOOP DROP ;
: XCORREL N 0 DO C ANSWER ? I POINT ANSWER @ N I - / I CORREL +
? 2 +LOOP ; : COLOC CORREL MAXN COMAXN ? ;
: TEMPOR N 0 DO I CORREL + @ I SINK + ? 2 +LOOP ;
: COFACTOR LIMITS DO COMAXN @ I @ 255 */ I ? 2 +LOOP ;
: MINNB 255 SWAP LIMITS DO I @ MIN 2 +LOOP ;
: FINE CORREL MINNB COMIN ? ;
: SUBTRACT LIMITS DO I @ COMIN @ - I ? 2 +LOOP ;
CODE OUTPUT $ PPC CALL A E MOV CMA 7 OUT RET
: TEST 256 0 DO I OUTPUT 10 MSEC LOOP ;
: SCOPE BEGIN DUP LIMITS DO I @B OUTPUT 2 +LOOP C OUTPUT ?VDU
END 2DROP ;

```

APPENDIX B

```

( 100 POINTS CORRELATION USING DIRECT METHOD FOR TMS9900
  SYSTEM WRITTEN IN FORTH PROGRAMMING LANGUAGE )
( DEFINITIONS OF VARIABLES AND ARRAYES )
HEX 64 CONSTANT N      C8 CONSTANT NN 62 CONSTANT OPTION
0 INTEGER MAXNB      0 INTEGER LOCATION 0 INTEGER DELAY
4 ARRAY ANSWER      4 ARRAY TEMPOR      NN ARRAY 1INPUT
N ARRAY 2INPUT      N ARRAY SINK      NN ARRAY CORREL
( ROUTINES 1IN, 2IN, 1GET AND 2GET CONVERT & STORE DATA)
CODE 1IN 280 0C LI 2 SBZ 2 SBD 220 0C LI BEGIN 2 TB FNE
END 320 0C LI 0 4 CLR 8 0 4 STCR 0 4 SWPB 4 PUSH RETURN
CODE 2IN 280 0C LI 3 SBZ 3 SBD 220 0C LI BEGIN 3 TB FNE
END 330 0C LI 0 4 CLR 8 0 4 STCR 0 4 SWPB 4 PUSH RETURN
: 1GET NN 0 DO 1IN I 1+ 1INPUT + ? DELAY 2 MSEC 2 +LOOP ;
: 2GET N 0 DO 2IN I 1+ 2INPUT + ? DELAY @ MSEC 2 +LOOP ;
( I/O & GETI/O STORE DATA IN ARRAY POINT BY POINT)
CODE I/O 230 0C LI 2 SBZ 3 SBZ 2 SBD 3 SBD
220 0C LI BEGIN 2 TB FNE IF 3 TB THEN FNE END 330 0C LI
0 4 CLR 8 0 4 STCR 0 4 SWPB 4 PUSH 320 0C LI
0 4 CLR 8 0 4 STCR 0 4 SWPB 4 PUSH RETURN
: GETI/O NN 0 DO I/O I 1+ 1INPUT + ? I DUP OPTION > IF 2CDROP
ELSE 1+ 2INPUT + ? THEN DELAY @ MSEC 2 +LOOP ;
( *M MULTIPLY 2X16-BIT NC.)
( */M MULTIPLY 2X16-BIT NO. & DIVIDE BY 16-BIT NO.)
( */ COMPLETE */M)
( ** MOD 16 MULTIPLY)
HEX CODE *M 0 9 CLR 0 1 1 0E MOV 0 5 0 1 MOV 8000 1 ANDI
FNE IF 0 5 NEG THEN 2 0 1 2 0E MOV 0 6 0 1 MOV 8000 1 ANDI
FNE IF 0 6 NEG 0 9 DEC THEN 5 0 6 MPY 0 9 0 9 MOV FNE IF
0 5 NEG THEN 1 0E 0 6 MOV 2 2 0E 0 5 MOV RETURN
CODE */M 0 9 CLR 0 1 1 0E MOV 0 2 0 1 MOV 8000 1 ANDI FNE IF
0 2 NEG 0 9 INC THEN 2 0 1 2 0E MOV 0 3 0 1 MOV 8000 1 ANDI
FNE IF 0 3 NEG 0 9 DEC THEN 2 0 3 MPY 0 9 0 9 MOV FNE IF
0 2 NEG THEN 4 0 1 2 0E MOV 0 4 0 1 MOV 8000 1
ANDI FNE IF 0 4 NEG
0 9 DEC THEN 2 0 4 DIV 0 9 0 9 MOV FNE IF 0 2 NEG THEN
1 0E 0 4 MOV 2 2 0E 0 3 MOV 4 2 0E 0 2 MOV RETURN
: */ */M 2DROP ; : ** *M SWAP DROP ;
( *? MULTIPLY AND SUMMING THE RESULTS IN ANSWER)
CODE *? 0 1 3 0E MOV 0 2 3 0E MOV 2 0 1 MPY
ANSWER 5 LI 3 5 0 3 A FOC IF 1 5 INC THEN 1 5 0 2 A RETURN
( THIS VERSION COMPUTE CORRELATION FUNCTION USING DIRECT
  METHOD )
: PRINTB LIMITS DO I @B , LOOP CRLF ;
: TEST DUP ANSWER + @B SWAP 1+ ANSWER + @B ;
: 1TEST DUP TEMPOR + ROT SWAP ?B 1+ TEMPOR + ?B ;
: TRY 4 0 DO I TEST I 1TEST 2 +LOOP ;
: SHIFT + 1INPUT + @ ; : ZERO ANSWER EMPTY ;
: COEFFICIENT N 0 DO DUP I SHIFT I 2INPUT + @ *? 2 +LOOP
DROP ;
: TRANSFER 4 0 DO DUP I TEMPOR + @B SWAP ?B 1+ LOOP DROP ;
: XCORREL N 0 DO ZERO I COEFFICIENT TRY I DUP + CORREL +
TRANSFER 2 +LOOP ;
( THIS VERSION TRANSFORMS THE WORD LENTH TO 8-BIT)

```

APPENDIX 9

```

IN ORDER TO FIT D/A CONVERTER )
: 1T 2DUP + DUP 1+ CORREL + 3B ROT SINK + ?B 2+ CORREL + 3B ;
: 2T 1+ SINK + ?B ;
: STORE N 0 DO I 1T I 2T 2 +LOOP ;
: FIT N 0 DO I SINK + DUP 3B I 1+ SINK + DUP 3B 2ROT ?B
SWAP ?B 2 +LOOP ;
: COMAX 0 N 0 DO I SINK + 3 2OVER < IF SWAP I 2 / LOCATION ?
THEN DROP 2 +LOOP MAXNB ? ;
: COFACTOR N 0 DO MAXNB 2 I SINK + 3 FF */ I SINK + ?
2 +LOOP ;
CODE OUTPUT 2A0 0C LI 4 PCP 0 4 SWPB 9 0 4 LDCR RETURN
: SCOPE BEGIN DUP LIMITS DO I 1+ 3B OUTPUT 2 +LOOP 0 OUTFLT
?VDU END 2DROP ;
( FORMAT TO WRITE THE OUTPUT ON VDU )
: FMT LOCATION 1+ 3B 4 * CORREL + ;
: 1FMT 4 0 DO DUP I FMT + 3B SWAP ?B 1+ LOOP DROP ;
: V STRING # CORREL IN BYTES # SAY CRLF CORREL PRINTB
CRLF STRING " MAXIMUM NUMBER = " SAY ANSWER 1FMT ANSWER
PRINTB CRLF STRING " A LOCATION IS A SAY LOCATION ? CRLF
STRING 8 SINK IN BYTES 8 SAY CRLF SINK PRINT CRLF ;
( CYCLIC CORRELATION ROUTINE )
HEX 8 CONSTANT N 0 INTEGER ANSWER
N ARRAY Z N ARRAY X N ARRAY Y
: ZERO 0 ANSWER ? ; : SHIFT X + 3 ;
: COEF N 0 DO DUP I + N MOD SHIFT I Y + 3 *
ANSWER +? 2 +LOOP DROP ;
: CORREL N 0 DO ZERO I COEF ANSWER 3 I Z + ? 2 +LOOP ;
( DEFINE MOD I + K )
CODE +INDEX 0 2 1 0E MOV 2 0 2 2 0E A 3 2 CI FGT IF
-8 2 AI THEN 1 0E 0 2 MOV RETURN
: INDEX +INDEX SWAP DROP ;
( N=2 OPTIMAL SHORT CORRELATION USING RECT. TRANSFORM )
HEX 4 ARRAY HH 4 ARRAY XX
6 ARRAY AA 6 ARRAY MM 6 ARRAY WW
: 1STEP 2 HH + 3 2DUP 2 AA + ? HH 3 + 2 /
4 AA + ? HH 3 - 2 / AA ? ;
: 2STEP XX 3 2DUP 2 AA + 3 * 2 MM + ? 2 XX + 3 + 4 AA +
3 * 4 MM + ? 2 XX + 3 - AA 3 * MM ? ;
: 3STEP 2 MM + 3 4 WW + ? 4 MM + 3 MM 3 - 2 WW + ? MM 3
4 MM + 3 + 2 MM + 3 - WW ? ;
: CORL 1STEP 2STEP 3STEP ;
( N=3 OPTIMAL SHORT CORRELATION USING RECT. TRANSFORM )
HEX 6 ARRAY HH 6 ARRAY XX 6 ARRAY YY
8 ARRAY MM 8 ARRAY AA 8 ARRAY BB
: 1STEP 2 HH + 3 DUP HH 3 ROT - DUP 2 AA + ? 4 HH + 3 ROT
- DUP 4 AA + ? + 43691 *M* 6 AA + ? HH 3 4 HH + 3 2 HH +
3 + 43691 *M* AA ? ;
: 2STEP 4 XX + 3 DUP XX 3 ROT - DUP 2 BB + ? 2 XX + 3 ROT -
DUP 4 BB + ? + 6 BB + ? XX 3 2 XX + 3 4 XX + 3 + 3B ? ;
: 3STEP 8 0 DO I AA + 3 I BB + 3 *M* I MM + ? 2 +LOOP ;
: 4STEP 6 MM + 3 DUP 2 MM + 3 ROT - DUP MM 3 + YY ? 4 MM + 3
ROT - DUP MM 3 + 4 YY + ? + MM 3 SWAP - 2 YY + ? ;
: CORL 1STEP 2STEP 3STEP 4STEP ;
( N=5 OPTIMAL SHORT CORRELATION USING RECT. TRANSFORM )

```

APPENDIX B

```

DECIMAL 10 ARRAY FH      10 ARRAY XX      10 ARRAY YY
          20 ARRAY BB      20 ARRAY MM      20 ARRAY AA
: 1SS 2 HH + @ 2DLP 2DUP HH @ DUP ROT - 2 AA + ? + ROT 8 HH +
@ DUP ROT - 4 AA + ? + ROT 6 HH + @ DUP ROT - 6 AA + ? + SWAP
4 HH + @ DUP ROT - 8 AA + ? + 52429 ** AA ? ;
: 1STEP 1SS 2 AA + @ DUP 4 AA + @ DUP ROT + DUP 10 AA + ? SWAP
8 AA + @ DUP ROT + 16 AA + ? 6 AA + @ DUP ROT + DUP 12 AA + ?
ROT + 52429 ** 18 AA + ? + 14 AA + ? ;
: 2SS 8 XX + @ 2DLP 2DUP XX @ DUP ROT - 2 BB + ? + ROT 2 XX +
@ DUP ROT - 4 BB + ? + ROT 4 XX + @ DUP ROT - 6 BB + ?
+ SWAP 6 XX + @ DUP ROT - 8 BB + ? + BB ? ;
: 2STEP 2SS 2 BB + @ DUP 4 BB + @ DUP ROT + DUP 10 BB + ? SWAP 8
BB + @ DUP ROT + 16 BB + ? 6 BB + @ DUP ROT + DUP 12 BB + ? ROT
+ 18 BB + ? + 14 BB + ? ;
: 3STEP 20 0 DO I AA + @ I BB + @ ** I MM + ? 2 +LOOP ;
: 4STEP MM @ 18 MM + @ - 2DUP DUP 2 MM + @ 8 MM + @ - DUP ROT
+ 6 MM + @ - 12 MM + @ + YY ? - 4 MM + @ - 10 MM + @ + 2 YY + ?
4 MM + @ 6 MM + @ - DUP ROT + 2 MM + @ - 14 MM + @ + 4 YY + ?
- 8 MM + @ - 16 MM + @ + 8 YY + ? MM @ 2DUP + SWAP DUP 2 ** + +
YY @ - 2 YY + @ - 4 YY + @ - 8 YY + @ - 6 YY + ? ;
: CORL 1STEP 2STEP 3STEP 4STEP ;

```

Listings for chapter 5

(1) PN sequence generator microprogram

registers are :- t9----lsa

ta----msa
tb----lsdi
tc----msdi
r0----fsr
r3----lenth
r7----res
rb----line
s1----skip
s3----sdnac
sf----szero

labels are :-strt at location 000

out1 at location 030

shift at location 050

out2 at location 070

```
/ *****  
/  
/ title noise: pseudo random sequence generator  
/  
/ *****  
/ assignments  
%lsa = t9  
%msa = ta  
%lsdi = tb  
%msdi = tc  
%sdnac = s3  
%line = rb  
%fsr = r0  
%lenth = r3  
%szero = sf  
%skip = s1  
%res = r7  
/ *****  
/ this algorithm, masking bit-7 and bit-1, by  
/ ex-or and loading bit-one.  
/ output = the output sequence  
/ lsa = l. s. address  
/ msa = m. s. address of store ram simulator  
/ lsdi = l. s. data input  
/ msdi = m. s. data input  
/ sdnac = s3 skip if data not accepted  
/ skip = skip always
```



```

/ length = the sequence length
/ szero = skip if zero
/ res = counter initially(07)
/ r5 = the present 8-bits of the sequence
/ skip = skip always
/ ****
*begin
0 000 000A137010 strt: msa=#00
1 001 0009137010 lsa = #00
2 010 550C137010 msdi=#55
3 011 FE02337110 r2=#fe
4 020 A003337110 lenh=#a0
5 021 7F00337111 fsr=#7f , skip /initialization
6 030 005B104030 out1: *even
7 031 0005337110 lsdi=0+r5 /output to ram
8 040 0807337110 r5=#00
9 041 0000037110 res=#08
/ .. initiate the counter
a=#00
10 050 000B304130 shift: *even
11 051 002B352110 line=0+fsr
12 060 00B0030110 line=r2 msk line
13 061 0005433110 a=line ior a
r5,a=0 ior r5 , down
14 070 0700137013 out2: *even
15 071 3F0015611F branch out2 , sdnac
16 080 8000366110 0=#3f msk fsr , szero
17 081 002015211F fsr=#80 xor fsr
18 090 8000366110 0=r2 msk fsr , szero
19 091 0000533110 fsr=#80 xor fsr
20 0A0 000331313F fsr=0 ior fsr , down
21 0A1 0000001131 lenh=0_lenh-1 , szero
22 0B0 0000137011 a=0+a , skip
23 0B1 000731313F branch strt
24 0C0 0500137011 res=0_res-1 , szero
25 0C1 0300137010 branch shift
branch out1
*end

```

Listings for chapter 6

- (1) Transmitter microprogram
- (2) Receiver microprogram

registers are :- t6----fifo

tf----ctrl
r0----temp
r1----count
r5----byte
r7----cps0
rb----data
rc----add
re----pnsn
s1----skip
s8----skfe
sd----siffl
sf----szero

labels are :-start at location 000

testx0 at location 001

testx1 at location 071

tst0 at location 090

testx2 at location 0E1

tst1 at location 110

datx at location 141

datx0 at location 1A1

data0 at location 1C0

dat00 at location 200

ctx at location 220

data1 at location 240

dd0 at location 270

sntx at location 291

all1 at location 311

ctx00 at location 360

rstrt at location 3F0

inv11 at location 460

ctx01 at location 4B0

ctx10 at location 510

inv01 at location 580

ch02 at location 5E0

ctx02 at location 640

inv12 at location 6B0

ch12 at location 700

ctx12 at location 760

inv03 at location 7D0

ch03 at location 830

ctx03 at location 890

inv13 at location 900

ch13 at location 950
 ctx13 at location 9B0
 inv04 at location A20
 ch04 at location A70
 ctx04 at location AD0
 inv14 at location B40
 ch14 at location B90
 ctx14 at location BF0
 inv00 at location C60
 error0 at location CD0
 loop0 at location CE0
 error1 at location CF0
 loop1 at location D00

```

/ *****
/
/      direct sequence spread spectrum microProgram
/
/ *****
/ assignments
%skip = s1
%zero = sf
%ctrl = tf
%fifo = t6
%skfe = s8
%siffl = sd
%add = rc
%data = rb
%temp = r0
%cpso = r7
%count = r1
%byte = r5
%psgn = re
/ *****
/ data: either 0 or 1 one bit per sequence period
/ rnsn: the p.r.s.s and the modulated data transmitted
/ temp : 8-bit of sequence
/ te : display the modulated data
/ td : the p.r.s.s byte
/ tb : the low rate data information
/ input data are loaded to the fifo (0,1), 1-bit every one
/ sequence period
/ count: counter to load the data
/ cpso : clock pulse serial input
  
```

```

/ skip : s1 skip always
/ szero : sf skip if zero
/ siffl : sd skip if fifo full
/ skfe : s8 skip if fifo empty
/ fifo : fifo input register
/ contrl : control flass (tf)
/ ***** start *****
/
/the program starts with an instruction which by loading
/the program counter (t0) and skipping to give a 0 l.s.b
/not skipping to give 1 l.s.b allows a branch to any location
/in the prom.
*begin
0 000 0040107037 start: t0=f4+0 , s7
/..the external register (f7) is used to load the
/programmable divide by n counter
/ to test the transmitter operation the microprogram is initially
/ set up to generate a code sequence of length 127
/
1 001 0007337110 testx0: cps0=#00
2 010 010F13701F contrl=#01 , szero /should be no skip
3 011 0000103131 0=0+temp , skip
4 020 CD00137011 branch error0
5 021 0077307130 cps0=f7+0
6 030 0007103030 t7=0+cps0 / load programmable counter
7 031 0000337110 temp=#00
8 040 7F01337110 count=#7f /no. of bytes in the p.r.s.d from
9 041 0605337110 byte=#06
10 050 000C337110 add=#00
11 051 000E337110 pnsn=#00
12 060 070F137010 contrl=#07 / fifo master reset
13 061 0000103138 0=0+temp , skfe /should skip
14 070 CD00137011 branch error0
15 071 000C103030 testx1: tc=0+add
16 080 002E307130 pnsn=f2+0 / read m-sequence of length 127
17 081 00ED104030 td=0+pnsn / display m-sequence
*even
18 090 090013701D tst0: branch tst0 , siffl / check fifo full
19 091 00E6104030 fifo=0+pnsn / load fifo
/ address modulo 127
20 0A0 000C303100 add=0+add+1
21 0A1 001C12210F 0=count-add , szero
22 0B0 0000103131 0=0+temp , skip
23 0B1 000C337110 add=#00

```

24	0C0	000531313F		byte=0_byte-1 , szero
25	0C1	0700137011		branch testx1
				/ start transmitting
26	0D0	060F13701F		contrl=#06 , szero /should be no skip
27	0D1	0000103131		0=0+temp , skip
28	0E0	CD00137011		branch error0
				/ variable m-sequence clock rate
29	0E1	0077307130	testx2:	cpso=f7+0
30	0F0	0007103030		t7=0+cpso
31	0F1	000C103030		tc=0+add
32	100	002E307130		pnsgn=f2+0
33	101	00ED104030		td=0+pnsgn
				*even
34	110	110013701D	tst1:	branch tst1 , siffl
35	111	00E6104030		fifo=0+pnsgn
36	120	000C303100		add=0+add+1
37	121	001C12210F		0=count-add , szero
38	130	0000103131		0=0+temp , skip
39	131	000C337110		add=#00
40	140	0E00137010		branch testx2
				/
				/*****
				/..data transmit microprogram
				/*****
				/
				/ the microprogram transmits an alternative zero and one data bits
				/
41	141	0007337110	dattx:	cpso=#00
42	150	010F13701F		contrl=#01 , szero /should be no skip
43	151	0000103131		0=0+temp , skip
44	160	CD00137011		branch error0
45	161	0077307130		cpso=f7+0
46	170	0007103030		t7=0+cpso
47	171	0000337110		temp=#00
48	180	070F137010		contrl=#07 /fifo master reset
49	181	0000103138		0=0+temp , skfe /should skip
50	190	CD00137011		branch error0
51	191	0500337110		temp=#05
52	1A0	1005337110		byte=#10
				/ display data zero
53	1A1	000B337010	datx0:	data,tb=#00
54	1B0	0077307130		cpso=f7+0
55	1B1	0007103030		t7=0+cpso
				*even

```

56 1C0 1C0013701D data0: branch data0 , siffl
57 1C1 00B6104030 fifo=0+data
58 1D0 0005313130 byte=0_byte-1 , szero
59 1D1 000031313F temp=0_temp-1 , szero
60 1E0 1C00137011 branch data0
61 1E1 060F13701F contrl=#06 , szero
62 1F0 0000103131 0=0+temp , skip
63 1F1 CD00137010 branch error0
    *even
64 200 200013701F dat00: branch dat00 , szero
65 201 00B6104030 fifo=0+data
66 210 000531313F byte=0_byte-1 , szero
67 211 2000137010 branch dat00
68 220 0077307130 ctx: cps0=f7+0
69 221 0007103030 t7=0+cps0
70 230 1005337110 byte=#10
    / display data one
71 231 FF0B337010 data,tb=#ff
    *even
72 240 240013701D data1: branch data1 , siffl
73 241 00B6104030 fifo=0+data
74 250 000531313F byte=0_byte-1 , szero
75 251 2400137010 branch data1
76 260 1005337110 byte=#10
77 261 000B337010 data,tb=#00
    *even
78 270 270013701D dd0: branch dd0 , siffl
79 271 00B6104030 fifo=0+data
80 280 000531313F byte=0_byte-1 , szero
81 281 2700137010 branch dd0
82 290 2200137011 branch ctx
    /
    /*****
    /..general transmit microprogram
    /..the modulation type used is sequence inversion kyng(sik)
    /..the data bit period is equal 127 times the pn-chip duration
    /(i.e. the spreading ratio is 127)
    /*****
    / the microprogram is then configured to permit transmission
    / of spreading signal
    /
    / adjust the transmitter clock rate output
83 291 0007337110 sntx: cps0=#00
84 2A0 010F13701F contrl=#01 , szero /should be no skip

```

85	2A1	0000103131		0=0+temp , skip
86	2B0	CD00137011		branch error0
87	2B1	0077307130		cpso=f7+0
88	2C0	0007103030		t7=0+cpso
				/ initialisation
89	2C1	0000337110		temp=#00
90	2D0	7F01337110		count=#7f
91	2D1	000C337110		add=#00
92	2E0	000E337110		pnsn=#00
93	2E1	070F137010		contrl=#07 /fifo master reset
94	2F0	0000103138		0=0+temp , skfe / should skip
95	2F1	CD00137010		branch error0
				/
				/ data bits are 'synchronised' by recognising the all one's
				/ state of the m-sequence generator and starting a data bit
				/ at that time
				/.. check the all 1's state ..(7f)
96	300	0F05337111		byte=#0f , skip
97	301	0000103130		0=0+temp
98	310	7F00337110		temp=#7f
99	311	000C103030	all1:	tc=0+add
100	320	002E307130		pnsn=f2+0
				/
101	321	000E15211F		0=temp msk pnsn , szero
102	330	000C303101		add=0+add+1 , skip
103	331	0500337111		temp=#05 , skip
104	340	3100137010		branch all1
105	341	000B337010		data,tb=#00
106	350	00ED104030		td=0+pnsn
				/ display the transmitted signal
107	351	00BE362010		pnsn,te=data xor pnsn
				*even
108	360	360013701D	ctx00:	branch ctx00 , siffl
109	361	00E6104030		fifo=0+pnsn
				/ a cyclic counter is counted to determine the start of subsequent
				/ data bits
110	370	0005313130		byte=0_byte-1
111	371	000C303100		add=0+add+1
112	380	001C12210F		0=count-add , szero
113	381	0000103131		0=0+temp , skip
114	390	000C337110		add=#00
115	391	000C103030		tc=0+add
116	3A0	002E307130		pnsn=f2+0
117	3A1	00ED104030		td=0+pnsn

			/ sik or bask in which the data to be transmitted is modulo-2 / (exclusive-or) added to the code sequence is the modulation / scheme for this system pnsn,te=data xor pnsn temp=0-temp-1 , szero branch ctx00 / start transmission contrl=#06 , szero /should be no skip 0=0+temp , skip branch error1 cpso=f7+0 t7=0+cpso / / because the length ofthe sequence is a prime number (127) / the data bit is divided into 16 data bytes where 15 bytes are / of complete length (8-bits) and the 16th byte masked in order / to isolate the required number of bits *even
118	3B0	00BE362010	
119	3B1	000031313F	
120	3C0	3600137011	
121	3C1	060F13701F	
122	3D0	0000103131	
123	3D1	CF00137010	
124	3E0	0077307130	
125	3E1	0007103030	
126	3F0	3F0013701D	rstrt:
127	3F1	00E6104030	
128	400	000531313F	
129	401	0000103131	
130	410	4600137011	
131	411	000C303100	
132	420	001C12210F	
133	421	0000103131	
134	430	000C337110	
135	431	000C103030	
136	440	002E307130	
137	441	00ED104030	
138	450	00BE362010	
139	451	3F00137010	
140	460	000C303100	inv11:
141	461	001C12210F	
142	470	0000103131	
143	471	000C337110	
144	480	000C103030	
145	481	002E307130	
146	490	00ED104030	
147	491	800B337010	
148	4A0	00BE362011	
149	4A1	0000103130	

150	4B0	4B0013701D	ctx01:	branch ctx01 , siff1
151	4B1	00E6104030		fifo=0+pnsgn
152	4C0	0F05337110		byte=#0f
153	4C1	FF0B337010		data,tb=#ff
154	4D0	000C303100		add=0+add+1
155	4D1	001C12210F		0=count-add , szero
156	4E0	0000103131		0=0+temp , skip
157	4E1	000C337110		add=#00
158	4F0	000C103030		tc=0+add
159	4F1	002E307130		pnsgn=f2+0
160	500	00ED104030		td=0+pnsgn
161	501	00BE362010		pnsgn,te=data xor pnsgn
				*even
162	510	510013701D	ctx10:	branch ctx10 , siff1
163	511	00E6104030		fifo=0+pnsgn
164	520	000531313F		byte=0_byte-1 , szero
165	521	0000103131		0=0+temp , skip
166	530	5800137011		branch inv01
167	531	000C303100		add=0+add+1
168	540	001C12210F		0=count-add , szero
169	541	0000103131		0=0+temp , skip
170	550	000C337110		add=#00
171	551	000C103030		tc=0+add
172	560	002E307130		pnsgn=f2+0
173	561	00ED104030		td=0+pnsgn
174	570	00BE362010		pnsgn,te=data xor pnsgn
175	571	5100137010		branch ctx10
				/ last byte = 00111111
176	580	3F0B337010	inv01:	data,tb=#3f
177	581	000C303100		add=0+add+1
178	590	001C12210F		0=count-add , szero
179	591	0000103131		0=0+temp , skip
180	5A0	000C337110		add=#00
181	5A1	000C103030		tc=0+add
182	5B0	002E307130		pnsgn=f2+0
183	5B1	00ED104030		td=0+pnsgn
184	5C0	00BE362011		pnsgn,te=data xor pnsgn , skip
185	5C1	0000103130		0=0+temp
186	5D0	0077307130		cpso=f7+0
187	5D1	0007103030		t7=0+cpso
				*even
188	5E0	5E0013701D	ch02:	branch ch02 , siff1
189	5E1	00E6104030		fifo=0+pnsgn
190	5F0	0F05337110		byte=#0f

191	5F1	000B337010		data, tb=#00
192	600	000C303100		add=0+add+1
193	601	001C12210F		0=count-add , szero
194	610	0000103131		0=0+temp , skip
195	611	000C337110		add=#00
196	620	000C103030		tc=0+add
197	621	002E307130		pnsgn=f2+0
198	630	00ED104030		td=0+pnsgn
199	631	00BE362010		pnsgn, te=data xor pnsgn
200	640	640013701D	ctx02:	*even
201	641	00E6104030		branch ctx02 , siffl
202	650	000531313F		fifo=0+pnsgn
203	651	0000103131		byte=0_byte-1 , szero
204	660	6B00137011		0=0+temp , skip
205	661	000C303100		branch inv12
206	670	001C12210F		add=0+add+1
207	671	0000103131		0=count-add , szero
208	680	000C337110		0=0+temp , skip
209	681	000C103030		add=#00
210	690	002E307130		tc=0+add
211	691	00ED104030		pnsgn=f2+0
212	6A0	00BE362010		td=0+pnsgn
213	6A1	6400137010		pnsgn, te=data xor pnsgn
214	6B0	E00B337010	inv12:	branch ctx02
215	6B1	000C303100		/ last byte = 11100000
216	6C0	001C12210F		data, tb=#e0
217	6C1	0000103131		add=0+add+1
218	6D0	000C337110		0=count-add , szero
219	6D1	000C103030		0=0+temp , skip
220	6E0	002E307130		add=#00
221	6E1	00ED104030		tc=0+add
222	6F0	00BE362011		pnsgn=f2+0
223	6F1	0000103130		td=0+pnsgn
224	700	700013701D	ch12:	pnsgn, te=data xor pnsgn , skip
225	701	00E6104030		0=0+temp
226	710	0F05337110		*even
227	711	FF0B337010		branch ch12 , siffl
228	720	000C303100		fifo=0+pnsgn
229	721	001C12210F		byte=#0f
230	730	0000103131		data, tb=fff
231	731	000C337110		add=0+add+1
				0=count-add , szero
				0=0+temp , skip
				add=#00

232	740	000C103030		tc=0+add
233	741	002E307130		fnsgn=f2+0
234	750	00ED104030		td=0+fnsgn
235	751	00BE362010		fnsgn,te=data xor fnsgn
236	760	760013701D	ctx12:	*even
237	761	00E6104030		branch ctx12 , siffl
238	770	000531313F		fifo=0+fnsgn
239	771	0000103131		byte=0_byte-1 , szero
240	780	7D00137011		0=0+temp , skip
241	781	000C303100		branch inv03
242	790	001C12210F		add=0+add+1
243	791	0000103131		0=count-add , szero
244	7A0	000C337110		0=0+temp , skip
245	7A1	000C103030		add=#00
246	7B0	002E307130		tc=0+add
247	7B1	00ED104030		fnsgn=f2+0
248	7C0	00BE362010		td=0+fnsgn
249	7C1	7600137010		fnsgn,te=data xor fnsgn
250	7D0	0F0B337010	inv03:	branch ctx12
251	7D1	000C303100		/ last byte= 00001111
252	7E0	001C12210F		data,te=#0f
253	7E1	0000103131		add=0+add+1
254	7F0	000C337110		0=count-add , szero
255	7F1	000C103030		0=0+temp , skip
256	800	002E307130		add=#00
257	801	00ED104030		tc=0+add
258	810	00BE362011		fnsgn=f2+0
259	811	0000103130		td=0+fnsgn
260	820	0077307130		fnsgn,te=data xor fnsgn , skip
261	821	0007103030		0=0+temp
262	830	830013701D	ch03:	crso=f7+0
263	831	00E6104030		t7=0+crso
264	840	0F05337110		*even
265	841	000B337010		branch ch03 , siffl
266	850	000C303100		fifo=0+fnsgn
267	851	001C12210F		byte=#0f
268	860	0000103131		data,te=#00
269	861	000C337110		add=0+add+1
270	870	000C103030		0=count-add , szero
271	871	002E307130		0=0+temp , skip
272	880	00ED104030		add=#00
				tc=0+add
				fnsgn=f2+0
				td=0+fnsgn

273	881	00BE362010		pnsn,te=data xor pnsn
				*even
274	890	890013701D	ctx03:	branch ctx03 , siff1
275	891	00E6104030		fifo=0+pnsn
276	8A0	000531313F		byte=0_byte-1 , szero
277	8A1	0000103131		0=0+temp , skip
278	8B0	9000137011		branch inv13
279	8B1	000C303100		add=0+add+1
280	8C0	001C12210F		0=count-add , szero
281	8C1	0000103131		0=0+temp , skip
282	8D0	000C337110		add=#00
283	8D1	000C103030		tc=0+add
284	8E0	002E307130		pnsn=f2+0
285	8E1	00ED104030		td=0+pnsn
286	8F0	00BE362010		pnsn,te=data xor pnsn
287	8F1	8900137010		branch ctx03
				*even
				/ last byte = 11111000
288	900	F80B337010	inv13:	data,tb=#f8
289	901	000C303100		add=0+add+1
290	910	001C12210F		0=count-add , szero
291	911	0000103131		0=0+temp , skip
292	920	000C337110		add=#00
293	921	000C103030		tc=0+add
294	930	002E307130		pnsn=f2+0
295	931	00ED104030		td=0+pnsn
296	940	00BE362011		pnsn,te=data xor pnsn , skip
297	941	0000103130		0=0+temp
				*even
298	950	950013701D	ch13:	branch ch13 , siff1
299	951	00E6104030		fifo=0+pnsn
300	960	0F05337110		byte=#0f
301	961	FF0B337010		data,tb=#ff
302	970	000C303100		add=0+add+1
303	971	001C12210F		0=count-add , szero
304	980	0000103131		0=0+temp , skip
305	981	000C337110		add=#00
306	990	000C103030		tc=0+add
307	991	002E307130		pnsn=f2+0
308	9A0	00ED104030		td=0+pnsn
309	9A1	00BE362010		pnsn,te=data xor pnsn
				*even
310	9B0	9B0013701D	ctx13:	branch ctx13 , siff1
311	9B1	00E6104030		fifo=0+pnsn

```

312 9C0 000531313F
313 9C1 0000103131
314 9D0 A200137011
315 9D1 000C303100
316 9E0 001C12210F
317 9E1 0000103131
318 9F0 000C337110
319 9F1 000C103030
320 A00 002E307130
321 A01 00ED104030
322 A10 00BE362010
323 A11 9B00137010

```

```

324 A20 030B337010 inv04:
325 A21 000C303100
326 A30 001C12210F
327 A31 0000103131
328 A40 000C337110
329 A41 000C103030
330 A50 002E307130
331 A51 00ED104030
332 A60 00BE362011
333 A61 0000103130

```

```

334 A70 A70013701D ch04:
335 A71 00E6104030
336 A80 0F05337110
337 A81 000B337010
338 A90 000C303100
339 A91 001C12210F
340 AA0 0000103131
341 AA1 000C337110
342 AB0 000C103030
343 AB1 002E307130
344 AC0 00ED104030
345 AC1 00BE362010

```

```

346 AD0 AD0013701D ctx04:
347 AD1 00E6104030
348 AE0 000531313F
349 AE1 0000103131
350 AF0 B400137011
351 AF1 000C303100

```

```

byte=0_byte-1 , szero
0=0+temp , skip
branch inv04
add=0+add+1
0=count-add , szero
0=0+temp , skip
add=#00
tc=0+add
Pnsn=f2+0
td=0+Pnsn
Pnsn,te=data xor Pnsn
branch ctx13
*even
/ last byte = 00000011
data,tb=#03
add=0+add+1
0=count-add , szero
0=0+temp , skip
add=#00
tc=0+add
Pnsn=f2+0
td=0+Pnsn
Pnsn,te=data xor Pnsn , skip
0=0+temp
*even
branch ch04 , siff1
fifo=0+Pnsn
byte=#0f
data,tb=#00
add=0+add+1
0=count-add , szero
0=0+temp , skip
add=#00
tc=0+add
Pnsn=f2+0
td=0+Pnsn
Pnsn,te=data xor Pnsn
*even
branch ctx04 , siff1
fifo=0+Pnsn
byte=0_byte-1 , szero
0=0+temp , skip
branch inv14
add=0+add+1

```

```

352 B00 001C12210F
353 B01 0000103131
354 B10 000C337110
355 B11 000C103030
356 B20 002E307130
357 B21 00ED104030
358 B30 00BE362010
359 B31 AD00137010

```

```

360 B40 FE0B337010 inv14:
361 B41 000C303100
362 B50 001C12210F
363 B51 0000103131
364 B60 000C337110
365 B61 000C103030
366 B70 002E307130
367 B71 00ED104030
368 B80 00BE362011
369 B81 0000103130

```

```

370 B90 B90013701D ch14:
371 B91 00E6104030
372 BA0 0F05337110
373 BA1 FF0B337010
374 BB0 000C303100
375 BB1 001C12210F
376 BC0 0000103131
377 BC1 000C337110
378 BD0 000C103030
379 BD1 002E307130
380 BE0 00ED104030
381 BE1 00BE362010

```

```

382 BF0 BF0013701D ctx14:
383 BF1 00E6104030
384 C00 000531313F
385 C01 0000103131
386 C10 C600137011
387 C11 000C303100
388 C20 001C12210F
389 C21 0000103131
390 C30 000C337110
391 C31 000C103030

```

```

0=count-add , szero
0=0+temp , skip
add=#00
tc=0+add
Pnsn=f2+0
td=0+Pnsn
Pnsn,te=data xor Pnsn
branch ctx04
*even
/ last byte = 11111110
data,tb=#fe
add=0+add+1
0=count-add , szero
0=0+temp , skip
add=#00
tc=0+add
Pnsn=f2+0
td=0+Pnsn
Pnsn,te=data xor Pnsn , skip
0=0+temp
*even
branch ch14 , siff1
fifo=0+Pnsn
byte=#0f
data,tb=#ff
add=0+add+1
0=count-add , szero
0=0+temp , skip
add=#00
tc=0+add
Pnsn=f2+0
td=0+Pnsn
Pnsn,te=data xor Pnsn
*even
branch ctx14 , siff1
fifo=0+Pnsn
byte=0_byte-1 , szero
0=0+temp , skip
branch inv00
add=0+add+1
0=count-add , szero
0=0+temp , skip
add=#00
tc=0+add

```

392	C40	002E307130		Pnsn=f2+0
393	C41	00ED104030		td=0+Pnsn
394	C50	00BE362010		Pnsn,te=data xor Pnsn
395	C51	BF00137010		branch ctx14
				/ last byte = 00000000
396	C60	000B337010	inv00:	data,tb=#00
397	C61	0F05337111		byte=#0f , skip
398	C70	0000103130		0=0+temp
399	C71	000C303100		add=0+add+1
400	C80	001C12210F		0=count-add , szero
401	C81	0000103131		0=0+temp , skip
402	C90	000C337110		add=#00
403	C91	000C103030		tc=0+add
404	CA0	002E307130		Pnsn=f2+0
405	CA1	00ED104030		td=0+Pnsn
406	CB0	00BE362010		Pnsn,te=data xor Pnsn
407	CB1	0077307130		cpso=f7+0
408	CC0	0007103030		t7=0+cpso
409	CC1	3F00137010		branch rstrt
				/ error routines
				*even
410	CD0	CE00137011	error0:	branch loop0
411	CD1	0000103130		0=0+temp
				*even
412	CE0	CD00137011	loop0:	branch error0
413	CE1	0000103130		0=0+temp
				*even
414	CF0	D000137011	error1:	branch loop1
415	CF1	0000103130		0=0+temp
				*even
416	D00	CF00137011	loop1:	branch error1
				*end

registers are :- f1----ramrd

f2----refrd
f3----errd
f5----promrd
f6----fifrd
f8----ltrd
t5----promad
t9----lsa
tb----ramwt
tc----refad
td----erad
te----ltad
tf----ctrl
r0----temp
r1----count
r2----refce
r3----erly
r4----asmnt
r5----const
r6----thred
r7----cpsi
r8----late
r9----ladd
rb----data
rd----dsmnt
re----byte
rf----ones
s1----skip
s8----skfe
sb----siplus
sd----siffl
sf----szero

labels are :-start at location 000

ssrx at location 070
tdat at location 080
rxff at location 0B0
error0 at location 0D0
strt at location 190
pnx0 at location 1E1
tff0 at location 210
Phi0 at location 2B0
asn at location 310
Phi1 at location 3A0

cal at location 3E1
 true at location 441
 xcal at location 491
 ntve at location 4E1
 corrl at location 510
 dsprd at location 571
 len at location 5B0
 chfe at location 600
 track at location 6B0
 eror1 at location 7C0
 dsrx0 at location 871
 Prsrx at location 8C1
 tfifrx at location 8F0
 seq0 at location 991
 rfram at location 9F1
 seq1 at location A90
 Procs0 at location AD1
 chthr at location B31
 modthrd at location B81
 minus at location BD1
 colpt at location C00
 demod at location C61
 Proms at location CA0
 drff at location CF0
 cornet at location DA0
 eror2 at location EB0

```

/ ****
/
/   spread spectrum correlator receiver microprogram
/
/ ****
/
/assignments
/ .....
%zero = sf
%skip = s1
%skfe = s8
%siffl = sd
%siplus = sb
%lsa = t9
%promad = t5
%ramwt = tb
%contrl = tf
  
```

```

%ramrd = f1
%promrd = f5
%fifrd = f6
%refad = tc
%refrd = f2
%temp = r0
%count = r1
%data = rb
%epsi = r7
%byte = re
%refce = r2
%erly = r3
%late = r8
%errd = f3
%erad = td
%ltad = te
%ltrd = f8
%asmnt = r4
%ladd = r9
%dsmnt = rd
%thred = r6
%const = r5
%ones = rf
/
/ *****
/ the program starts with an instruction which by loading
/ the program counter (t0) and skipping to give a 0 l.s.b
/ not skipping to give 1 l.s.b allows a branch to any
/ location in the prom.
/ szero- skip if zero
/ skip - skip always
/ skfe - skip if fifo empty
/ siffl - skip if fifo full
/ siplus - skip if output positive
/ ramwt - write into ram
/ ramrd - read from ram
/ fifrd - read from fifo
/ epsi - fifo clock pulse serial input
/ contrl - control flags (tf)
/ promad - hamming weight function address
/ promrd - hamming weight function output
/ refad - reference p.n.s address
/ refrd - references p.n.s data
/ erad - erly address

```

```

/ errd - early data
/ ltad - late address
/ ltrd - late data
/
/
*begin
0 000 0040107037 start: t0=f4+0 , s7
/ the external register (f7) is used to load the
/ programmable divide by n counter
/
/ *****
/
/ ..data test microprogram.
/
/ *****
/
/ the program tests the correct reception of the data.
/ t8 displays the correct reception of (ff), t9 displays
/ the correct reception of (00).
/ the input data sequence is serially clocked into the
/ fifo input register using the clock output into the
/ generated by the programmable counter.
/ initialisation
1 001 0008337110 r8=#00
2 010 0009337110 r9=#00
3 011 0000037110 a=#00
4 020 FF03337110 r3=#ff
5 021 070F137010 contrl=#07 /fifo master reset
6 030 0000103138 0=0+temp , skfe / should skip
7 031 0D00137010 branch error0
8 040 0007337110 cpsi=#00 /initialise programmable divider
9 041 010F13701F contrl=#01 , szero /should be no skip
10 050 0000103131 0=0+temp , skip
11 051 0D00137010 branch error0
12 060 0008103030 t8=0+r8
13 061 0009103030 t9=0+r9
/ load the divider factor
/ the fifo places a signal on the (data accept) which loads
/ the contents of the output register onto 8-bit data bus.
*even
14 070 0077307130 ssrx: cpsi=f7+0
15 071 0007103030 t7=0+cpsi
/ test fifo empty
*even

```

```

16. 080 0800137018 tdat: branch tdat , skfe
17 081 0066307130 r6=f6+0
18 090 006312210F 0=r6-r3 , szero
19 091 0800137010 branch rxff
20 0A0 0008303000 r8,t8=0+r8+1
21 0A1 0700137010 branch ssrx
22 0B0 006012010F rxff: 0=r6-g , szero
23 0B1 0700137010 branch ssrx
24 0C0 0009303000 r9,t9=0+r9+1
25 0C1 0700137010 branch ssrx
                                *even
26 0D0 0000103130 error0: 0=0+temp
27 0D1 0D00137010 branch error0
                                /
                                / *****
                                /
                                / general fn spread spectrum receiver microprogram
                                /
                                / *****
                                / the program contains an instruction which by loading
                                / a programmable counter (f7), an skip control flag (s6)
                                / and skipping to test the modulated data, not skipping
                                / to test the sequence without modulation
                                / (f4) is used to load the value (thred).
                                / t8: displays the modulating data
                                / td: should display (80)
                                / te: tumplins display (00,7f)
28 0E0 000E337010 re,te=#00
29 0E1 000D337010 rd,td=#00
30 0F0 000F337110 ones=#00
31 0F1 0000337110 temp=#00
32 100 0008337010 r8,t8=#00
33 101 0805337110 const=#08
34 110 0003337110 erly=#00
35 111 7F01337110 count=#7f /no.of bytes in the p.r.s.d from
36 120 070F137010 contrl=#07
37 121 0000103138 0=0+temp , skfe /should skip
38 130 7C00137011 branch error1
39 131 0007337110 cpsi=#00
40 140 010F13701F contrl=#01 , szero /should be no skip
41 141 0000103131 0=0+temp , skip
42 150 7C00137011 branch error1
43 151 000C337110 rc=#00
44 160 0077307130 cpsi=f7+0

```

```

45 161 0007103030          t7=0+cpsi
46 170 0004337110          asmnt=#00
47 171 7F08337110          late=#7f
48 180 4106337110          thred=#41
49 181 1E00137011          branch pnx0
50 190 0077307130  strt:   cpsi=f7+0
51 191 0007103030          t7=0+cpsi
52 1A0 4106337110          thred=#41
53 1A1 7F08337110          late=#7f
54 1B0 0004337110          asmnt=#00
55 1B1 000D337110          dsmnt=#00
56 1C0 000E137010          te=#00
57 1C1 000C303100          rc=0+rc+1
58 1D0 001C12210F          0=count-rc , szero
59 1D1 0000103131          0=0+temp , skip
60 1E0 000C337110          rc=#00
                               / *****
                               /
                               /          acquisition phase
                               /
                               / *****
                               / initialize the address counter (t9).
61 1E1 0009337010  pnx0:   ladd,lsa=#00
62 1F0 000C103030          refad=0+rc
63 1F1 0022307130          refce=refrd+0
64 200 1000037110          a=#10
65 201 000B337110          data=#00
                               /
                               / .... correlation process....
                               /
                               / this program computes the number of agreement bits while
                               / shifting the receiver's code periodically by 1 chip
                               / increment each time
                               / even
66 210 2100137018  tff0:   branch tff0 , skfe
67 211 006B307130          data=fifrd+0
68 220 000B103030          ramwt=0+data
69 221 002B372110          data=refce xnr data
70 230 000001113F          a=0_a-1 , szero
71 231 0000103131          0=0+temp , skip
72 240 2B00137011          branch phi0
73 241 00B5104030          promad=0+data
74 250 005F307130          ones=promrd+0
75 251 00FE304130          byte=0+ones

```

76	260	00F4302130		asmnt=ones+asmnt
77	261	005E322100		byte=const-byte
78	270	00ED302130		dsmnt=byte+dsmnt
79	271	000C303100		rc=0+rc+1
80	280	001C12210F		0=count-rc , szero
81	281	0000103131		0=0+temp , skip
82	290	000C337110		rc=#00
83	291	000C103030		refad=0+rc
84	2A0	0022307130		refce=refrd+0
85	2A1	2100137010		branch tff0
86	2B0	8000337110	phi0:	temp=#80
87	2B1	000B332110		data=temp ior data
88	2C0	00B5104030		promad=0+data
89	2C1	005F307130		ones=promrd+0
90	2D0	00FE304130		byte=0+ones
91	2D1	000F313130		ones=0_ones-1
92	2E0	00F4302130		asmnt=ones+asmnt
93	2E1	005E322100		byte=const-byte
94	2F0	00ED302130		dsmnt=byte+dsmnt
95	2F1	1000037110		a=#10
96	300	0009337010		ladd,lsa=#00
97	301	3E00137011		branch cal
98	310	001B307130	asn:	data=ramrd+0
99	311	002B372110		data=refce xnr data
100	320	000001113F		a=0_a-1 , szero
101	321	0000103131		0=0+temp , skip
102	330	3A00137011		branch phi1
103	331	00B5104030		promad=0+data
104	340	005F307130		ones=promrd+0
105	341	00FE304130		byte=0+ones
106	350	00F4302130		asmnt=ones+asmnt
107	351	005E322100		byte=const-byte
108	360	00ED302130		dsmnt=byte+dsmnt
109	361	000C303100		rc=0+rc+1
110	370	001C12210F		0=count-rc , szero
111	371	0000103131		0=0+temp , skip
112	380	000C337110		rc=#00
113	381	000C103030		refad=0+rc
114	390	0022307130		refce=refrd+0
115	391	3100137010		branch asn
116	3A0	8000337110	phi1:	temp=#80
117	3A1	000B332110		data=temp ior data
118	3B0	00B5104030		promad=0+data
119	3B1	005F307130		ones=promrd+0

120	3C0	00FE304130		byte=0+ones
121	3C1	000F313130		ones=0_ones-1
122	3D0	00F4302130		asmnt=ones+asmnt
123	3D1	005E322100		byte=const-byte
124	3E0	00ED302130		dsmnt=byte+dsmnt
125	3E1	004D322100	cal:	dsmnt=asmnt-dsmnt
126	3F0	00DE304036		byte,te=0+dsmnt , s6
				/
				/.search for correlation peak
				/
				/ recognising the correlation peak is an inherent part of
				/ the acquisition process, the peak may be maximum positive
				/ or maximum negative, the microprogram tests whether the peak
				/ is above a positive threshold or below a negative threshold.
				/
127	3F1	0106337111		thred=#01 , skip
128	400	4400137010		branch true
129	401	006D30203B		dsmnt,td=thred+dsmnt , siplus
130	410	7F00337111		temp=#7f , skip
131	411	5100137010		branch corrl
132	420	000E12210F		0=temp-byte , szero
133	421	0000103131		0=0+temp , skip
134	430	5700137010		branch dsprd
				/
				/.ta displays the number of errors in detecting the corr.
				/ peak.
				/
135	431	000A303000		ra,ta=0+ra+1
136	440	1900137011		branch strt
137	441	0000103132	true:	0=0+temp , s2
138	450	4900137010		branch xcal
				/ the correlation values are arranged such that the peak
				/ amplitude lies in the range of 8-bits two's complement
				/ the values will change as a result of the overflow
				/ properties of two's complement.
				/
139	451	006D30203B		dsmnt,td=thred+dsmnt , siplus
140	460	5700137010		branch dsprd
141	461	00D010413F		0=0+dsmnt , szero
142	470	0000103131		0=0+temp , skip
143	471	5700137011		branch dsprd
144	480	000831313F		late=0_late-1 , szero
145	481	5100137010		branch corrl
146	490	1900137011		branch strt


```

147 491 00D030413B xcal: temp=0+dsmt , siplus
148 4A0 4E00137010 branch ntve
149 4A1 510030613B temp=#51+temp , siplus
150 4B0 5700137010 branch dsprd
151 4B1 000010313F 0=0+temp , szero
152 4C0 0000337111 temp=#00 , skip
153 4C1 5700137011 branch dsprd
154 4D0 000E31313F byte=0_byte-1 , szero
155 4D1 5100137010 branch corrl
156 4E0 1900137011 branch strt
157 4E1 300030613B ntve: temp=#30+temp , siplus
158 4F0 5700137010 branch dsprd
159 4F1 000E31313F byte=0_byte-1 , szero
160 500 5100137011 branch corrl
161 501 1900137010 branch strt
162 510 0004337110 corrl: asmnt=#00
163 511 000D337110 dsmt=#00
164 520 1000037110 a=#10
165 521 0009337010 ladd,lsa=#00
166 530 000C303100 rc=0+rc+1
167 531 001C12210F 0=count-rc , szero
168 540 0000103131 0=0+temp , skip
169 541 000C337110 rc=#00
170 550 000C103030 refad=0+rc
171 551 0022307130 refce=refrd+0
172 560 0077307130 cpsi=f7+0
173 561 0007103030 t7=0+cpsi
174 570 3100137011 branch asn
/ *****
/ ..despread and tracking phase
/ *****
/ an error signal generated mode is introduced in the same
/ phase with despreading the tracking correlator is
/ implemented by generating the local sequences (early and
/ late), using prom's and replacing the multiplier by exclusive
/ or operation.

175 571 1000037110 dsprd: a=#10
176 580 0077307130 cpsi=f7+0
177 581 0007103030 t7=0+cpsi
178 590 0009337110 r9=#00
179 591 000A337110 ra=#00
180 5A0 0004337110 r4=#00
181 5A1 000D337110 rd=#00
182 5B0 000C303100 lfn: rc=0+rc+1

```

183	5B1	001C12210F		0=count-rc , szero
184	5C0	0000103131		0=0+temp , skip
185	5C1	000C337110		rc=#00
186	5D0	000C103030		refad=0+rc
187	5D1	0022307130		refce=refrd+0
				/ early generator
188	5E0	00CD104030		erad=0+rc
189	5E1	0033307130		erly=errd+0
				/ late generator
190	5F0	00CE104030		ltad=0+rc
191	5F1	0088307130		late=ltrd+0
				/
				/.retreave data from fifo
				/
				*even
192	600	6000137018	chfe:	branch chfe , skfe
193	601	006B307130		data=fifrd+0
				/
				/. start despread
				/
194	610	00B2362110		refce=data xor refce
195	611	0028104030		t8=0+refce
196	620	000001113F		a=0-a-1 , szero
197	621	0000103131		0=0+temp , skip
198	630	6B00137011		branch track
199	631	00B8372110		late=data xnr late
200	640	0085104030		promad=0+late
201	641	005F307130		ones=promrd+0
202	650	00FE304130		byte=0+ones
203	651	00F4302130		r4=onest+r4
204	660	005E322100		byte=const-byte
205	661	00ED302130		rd=byte+rd
206	670	00B3372110		erly=data xnr erly
207	671	0035104030		promad=0+erly
208	680	005F307130		ones=promrd+0
209	681	00FE304130		byte=0+ones
210	690	00F9302130		r9=onest+r9
211	691	005E322100		byte=const-byte
212	6A0	00EA302130		ra=byte+ra
213	6A1	5B00137010		branch lfn
				/
				/ early-late correlation process
214	6B0	8000337110	track:	temp=#80
215	6B1	00B8372110		late=data xnr late

216	6C0	0008332110		late=temp ior late
217	6C1	0085104030		promad=0+late
218	6D0	005F307130		ones=promrd+0
219	6D1	00FE304130		byte=0+ones
220	6E0	000F313130		ones=0-ones-1
221	6E1	00F4302130		r4=ones+r4
222	6F0	005E322100		byte=const-byte
223	6F1	00ED302130		rd=byte+rd
224	700	00B3372110		erly=data xnr erly
225	701	0003332110		erly=temp ior erly
226	710	0035104030		promad=0+erly
227	711	005F307130		ones=promrd+0
228	720	00FE304130		byte=0+ones
229	721	000F313130		ones=0-ones-1
230	730	00F9302130		r9=ones+r9
231	731	005E322100		byte=const-byte
232	740	00EA302130		ra=byte+ra
233	741	004D322100		rd=r4-rd
234	750	009A322100		ra=r9-ra
				/
				/ error signal
				/
235	751	00AD32210F		rd=ra-rd , szero
236	760	0400337111		temp=#04 , skip
237	761	5700137011		branch dsard
238	770	0000103136		0=0+temp , s6
239	771	1900137010		branch strt
240	780	00D030213F		temp=rd+temp , szero
241	781	0000103131		0=0+temp , skip
242	790	5700137010		branch dsard
243	791	080032610F		temp=#08-temp , szero
244	7A0	0000337111		temp=#00 , skip
245	7A1	5700137011		branch dsard
				/.false alarm
246	7B0	1900137011		branch strt
247	7B1	0000103130		0=0+temp
				*even
248	7C0	0000103130	error1:	0=0+temp
249	7C1	7C00137010		branch error1
				/
				/
				/
				/
				/

```

/
/ *****
/      pn-reciever microProgram.,version-11
/ *****
/ in this versio the case of outo-increment ram address
/ is eliminated. to test the hardware method
/

250 7D0 000E337010      re,te=#00
251 7D1 000D337010      rd,td=#00
252 7E0 000F337110      ones=#00
253 7E1 0000337110      temp=#00
254 7F0 0008337010      r8,t8=#00
255 7F1 0805337110      const=#08
256 800 0003337110      erly=#00
257 801 7F01337110      count=#7f
258 810 070F137010      contrl=#07
259 811 0000103138      0=0+temp , skfe /should be skip
260 820 EB00137011      branch error2
261 821 0007337110      cpsi=#00
262 830 010F13701F      contrl=#01 , szero /should be no skip
263 831 0004337111      asmnt=#00 , skip
264 840 EB00137011      branch error2
265 841 000C337110      rc=#00
266 850 0077307130      cpsi=f7+0
267 851 0007103030      t7=0+cpai
268 860 7F08337110      late=#7f
269 861 4106337110      thred=#41
270 870 8C00137010      branch prsrx
271 871 0077307130      cpsi=f7+0
272 880 0007103030      t7=0+cpai
273 881 4106337110      thred=#41
274 890 7F08337110      late=#7f
275 891 0004337110      asmnt=#00
276 8A0 000D337110      dsmnt=#00
277 8A1 000C303100      rc=0+rc+1
278 8B0 001C12210F      0=count-rc , szero
279 8B1 0000103131      0=0+temp , skip
280 8C0 000C337110      rc=#00
/initialize the address counter (t9).
281 8C1 0009337010      ladd,lsa=#00
282 8D0 000C103030      refad=0+rc
283 8D1 0022307130      refce=refrd+0
284 8E0 1000037110      a=#10
285 8E1 000B337110      data=#00

```

dsrx0:

prsrx:

```

      *even
286 8F0 8F00137018 tfifrx: branch tfifrx , skfe
287 8F1 006B307130 data=fifrd+0
288 900 000B103030 ramwt=0+data
289 901 002B372110 data=refce xnr data
290 910 0009303000 ladd,lsa=0+ladd+1
291 911 000001113F a=0_a-1 , szero
292 920 0000103131 0=0+temp , skip
293 921 9900137011 branch seq0
294 930 00B5104030 fromad=0+data
295 931 005F307130 ones=fromrd+0
296 940 00FE304130 byte=0+ones
297 941 00F4302130 asmnt=ones+asmnt
298 950 005E322100 byte=const-byte
299 951 00ED302130 dsmnt=byte+dsmnt
300 960 000C303100 rc=0+rc+1
301 961 001C12210F 0=count-rc , szero
302 970 0000103131 0=0+temp , skip
303 971 000C337110 rc=#00
304 980 000C103030 refad=0+rc
305 981 0022307130 refce=refrd+0
306 990 8F00137011 branch tfifrx
307 991 8000337110 seq0: temp=#80
308 9A0 000B332110 data=temp ior data
309 9A1 00B5104030 fromad=0+data
310 9B0 005F307130 ones=fromrd+0
311 9B1 00FE304130 byte=0+ones
312 9C0 000F313130 ones=0_ones-1
313 9C1 00F4302130 asmnt=ones+asmnt
314 9D0 005E322100 byte=const-byte
315 9D1 00ED302130 dsmnt=byte+dsmnt
316 9E0 1000037110 a=#10
317 9E1 0009337010 ladd,lsa=#00
318 9F0 AD00137010 branch froms0
319 9F1 001B307130 rfram: data=ramrd+0
320 A00 002B372110 data=refce xnr data
321 A01 0009303000 ladd,lsa=0+ladd+1
322 A10 000001113F a=0_a-1 , szero
323 A11 0000103131 0=0+temp , skip
324 A20 A900137011 branch seq1
325 A21 00B5104030 fromad=0+data
326 A30 005F307130 ones=fromrd+0
327 A31 00FE304130 byte=0+ones
328 A40 00F4302130 asmnt=ones+asmnt

```

329	A41	005E322100		byte=const-byte
330	A50	00ED302130		dsmnt=byte+dsmnt
331	A51	000C303100		rc=0+rc+1
332	A60	001C12210F		0=count-rc , szero
333	A61	0000103131		0=0+temp , skip
334	A70	000C337110		rc=#00
335	A71	000C103030		refad=0+rc
336	A80	0022307130		refce=refrd+0
337	A81	9F00137011		branch rfram
338	A90	8000337110	seq1:	temp=#80
339	A91	000B332110		data=temp ior data
340	AA0	00B5104030		fromad=0+data
341	AA1	005F307130		ones=fromrd+0
342	AB0	00FE304130		byte=0+ones
343	AB1	000F313130		ones=0_ones-1
344	AC0	00F4302130		asmnt=ones+asmnt
345	AC1	005E322100		byte=const-byte
346	AD0	00ED302130		dsmnt=byte+dsmnt
347	AD1	004D322100	procs0:	dsmnt=asmnt-dsmnt
348	AE0	00DE304036		byte,te=0+dsmnt , s6
349	AE1	0106337111		/. detect correlation peak
350	AF0	B300137010		thred=#01 , skip
351	AF1	006D30203B		branch chthr
352	B00	7F00337111		dsmnt,td=thred+dsmnt , siplus
353	B01	C000137010		temp=#7f , skip
354	B10	000E12210F		branch colst
355	B11	0000103131		0=temp-byte , szero
356	B20	C600137010		0=0+temp , skip
				branch demod
				/.ta displays the no. of errors in detecting the corr.
				/ Peak.
357	B21	000A303000		ra,ta=0+ra+1
358	B30	8700137010		branch dsrx0
359	B31	0000103132	chthr:	0=0+temp , s2
360	B40	B800137010		branch modthrd
361	B41	006D30203B		dsmnt,td=thred+dsmnt , siplus
362	B50	C600137010		branch demod
363	B51	00D010413F		0=0+dsmnt , szero
364	B60	0000103131		0=0+temp , skip
365	B61	C600137011		branch demod
366	B70	000831313F		late=0_late-1 , szero
367	B71	C000137010		branch colst
368	B80	8700137010		branch dsrx0
369	B81	00D030413B	modthrd:	temp=0+dsmnt , siplus

370	B90	BD00137010		branch minus
371	B91	510030613B		temp=#51+temp , siplus
372	BA0	C600137010		branch demod
373	BA1	000010313F		0=0+temp , szero
374	BB0	0000337111		temp=#00 , skip
375	BB1	C600137011		branch demod
376	BC0	000831313F		late=0_late-1 , szero
377	BC1	C000137010		branch colft
378	BD0	8700137010		branch dsrx0
379	BD1	300030613B	minus:	temp=#30+temp , siplus
380	BE0	C600137010		branch demod
381	BE1	000E31313F		byte=0_byte-1 , szero
382	BF0	0000103131		0=0+temp , skip
383	BF1	8700137011		branch dsrx0
384	C00	0004337110	colft:	asmnt=#00
385	C01	000D337110		dsmnt=#00
386	C10	1000037110		a=#10
387	C11	0009337010		ladd,lsa=#00
388	C20	000C303100		rc=0+rc+1
389	C21	001C12210F		0=count-rc , szero
390	C30	0000103131		0=0+temp , skip
391	C31	000C337110		rc=#00
392	C40	000C103030		refad=0+rc
393	C41	0022307130		refce=refrd+0
394	C50	0077307130		cpsi=f7+0
395	C51	0007103030		t7=0+cpsi
396	C60	9F00137010		branch rfram
				/.. track phase
				/
397	C61	1000037110	demod:	a=#10
398	C70	0077307130		cpsi=f7+0
399	C71	0007103030		t7=0+cpsi
400	C80	0009337110		r9=#00
401	C81	000A337110		ra=#00
402	C90	0004337110		r4=#00
403	C91	000D337110		rd=#00
404	CA0	000C303100	proms:	rc=0+rc+1
405	CA1	001C12210F		0=count-rc , szero
406	CB0	0000103131		0=0+temp , skip
407	CB1	000C337110		rc=#00
408	CC0	000C103030		refad=0+rc
409	CC1	0022307130		refce=refrd+0
410	CD0	00CD104030		erad=0+rc
411	CD1	0033307130		erly=errrd+0

```

412 CE0 00CE104030
413 CE1 0088307130

414 CF0 CF00137018
415 CF1 006B307130

416 D00 00B2362110
417 D01 0028104030
418 D10 000001113F
419 D11 0000103131
420 D20 DA00137011
421 D21 00B8372110
422 D30 0085104030
423 D31 005F307130
424 D40 00FE304130
425 D41 00F4302130
426 D50 005E322100
427 D51 00ED302130
428 D60 00B3372110
429 D61 0035104030
430 D70 005F307130
431 D71 00FE304130
432 D80 00F9302130
433 D81 005E322100
434 D90 00EA302130
435 D91 CA00137010
436 DA0 8000337110
437 DA1 00B8372110
438 DB0 0008332110
439 DB1 0085104030
440 DC0 005F307130
441 DC1 00FE304130
442 DD0 000F313130
443 DD1 00F4302130
444 DE0 005E322100
445 DE1 00ED302130
446 DF0 00B3372110
447 DF1 0003332110
448 E00 0035104030
449 E01 005F307130
450 E10 00FE304130
451 E11 000F313130
452 E20 00F9302130

```

drff:

cornet:

```

ltad=0+rc
late=ltad+0
/*.. retrieve data from fifo
*even
branch drff , skfe
data=fifrd+0
/*.. start desreading
refce=data xor refce
t8=0+refce
a=0-a-1 , szero
0=0+temp , skip
branch cornet
late=data xnr late
promad=0+late
ones=promrd+0
byte=0+ones
r4=ones+r4
byte=const-byte
rd=byte+rd
erly=data xnr erly
promad=0+erly
ones=promrd+0
byte=0+ones
r9=ones+r9
byte=const-byte
ra=byte+ra
branch proms
temp=#80
late=data xnr late
late=temp ior late
promad=0+late
ones=promrd+0
byte=0+ones
ones=0-ones-1
r4=ones+r4
byte=const-byte
rd=byte+rd
erly=data xnr erly
erly=temp ior erly
promad=0+erly
ones=promrd+0
byte=0+ones
ones=0-ones-1
r9=ones+r9

```


453	E21	005E322100		byte=const-byte
454	E30	00EA302130		ra=byte+ra
455	E31	004D322100		rd=r4-rd
456	E40	009A322100		ra=r9-ra
457	E41	00AD32210F		rd=ra-rd , szero
458	E50	0400337111		temp=#04 , skip
459	E51	C600137011		branch demod
460	E60	0000103136		0=0+temp , s6
461	E61	8700137011		branch dsrx0
462	E70	00D030213F		temp=rd+temp , szero
463	E71	0000103131		0=0+temp , skip
464	E80	C600137010		branch demod
465	E81	080032610F		temp=#08-temp , szero
466	E90	0000337111		temp=#00 , skip
467	E91	C600137011		branch demod
				/.false alarm
468	EA0	8700137010		branch dsrx0
469	EA1	0000103130		0=0+temp
				*even
470	EB0	0000103130	error2:	0=0+temp
471	EB1	EB00137010		branch error2
				*end