NAG 3-865

[illegible]

[illegible]

21423

Final Report

A PARALLEL-PIPELINED ARCHITECTURE FOR A MULTI CARRIER DEMODULATOR

Submitted to:

NASA Lewis Research Center
21000 Brookpark Road
Cleveland, Ohio 44135

Submitted by:

S. C. Kwatra Principal Investigator

M. M. Jamali Co-Investigator

L. P. Eugene Graduate Research Assistant

Department of Electrical Engineering
College of Engineering
The University of Toledo
Toledo, Ohio 43606

Report No. DTVI-26

March 1991

Final Report

A PARALLEL-PIPELINED ARCHITECTURE FOR A MULTI CARRIER DEMODULATOR

Submitted to:

NASA Lewis Research Center
21000 Brookpark Road
Cleveland, Ohio 44135

Submitted by:

S. C. Kwatra Principal Investigator

M. M. Jamali Co-Investigator

L. P. Eugene Graduate Research Assistant

Department of Electrical Engineering
College of Engineering
The University of Toledo
Toledo, Ohio 43606

Report No. DTVI-26

March 1991

This report contains part of the work performed under NASA grant NAG3-865 during the period January 1989 to August 1990. The research was performed as part of the Masters thesis requirement of Mr. Linus P. Eugene.

S. C. Kwatra

Principal Investigator

ABSTRACT

Analog devices have been used for processing the information on board the satellites. Presently, digital devices are being used because they are economical and flexible as compared to their analog counterparts. Several schemes of digital transmission can be used depending on the data rate requirement of the user. An economical scheme of transmission for small earth stations uses Single Channel Per Carrier/Frequency Division Multiple Access (SCPC/FDMA) on the uplink and Time Division Multiplexing (TDM) on the downlink. This is a typical communication service offered to low data rate users in commercial mass market. These channels usually pertain to either voice or data transmission.

An efficient digital demodulator architecture is provided for a large number of low data rate users. A demodulator primarily consists of carrier, clock and data recovery modules. This design uses principles of parallel processing, pipelining and time sharing schemes to process large number of voice or data channels. It maintains the optimum throughput which is derived from the designed architecture and from the use of high speed components. The design is optimized for reduced power and area requirements. This is essential for satellite applications. The design is also flexible in processing a group of varying number of channels. The algorithms used are verified by the use of a Computer Aided Software Engineering (CASE) tool called Block Oriented System Simulator. The data flow, control circuitry and interface of the hardware design is simulated in C language.

Also, a multiprocessor approach is provided to map, model and simulate the demodulation algorithms mainly from a speed view point. A hypercube based architecture implementation is provided for such a scheme of operation. The hypercube structure and the demodulation models on hypercubes are simulated in Ada.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter I

INTRODUCTION

For reasons of flexibility and economy, digital implementation of the signal processing hardware on board the communication satellites is the current trend. This is in contrast to the past when analog techniques and devices had been employed for the transmission of data. The analog devices were bulky, occupied large volume and consumed a large amount of power. There are several schemes for digital transmission of information through the satellites. The adoption of a particular scheme of transmission is based on the priority and importance given to some of the parameters involved in a transmission scheme. The advances in digital technology for the satellite communications aims at incorporating efficient schemes of transmission. It also aims at processing a large number of channels yet reducing power and weight requirements.

Depending on the bit rate requirement the users are classified into low, medium and high data rate users. Low and medium data rate users can use small earth stations with reduced costs and can ideally use the Single Channel Per Carrier/Frequency Division Multiple Access (SCPC/FDMA) scheme of transmission on the uplink and Time Division Multiplexing (TDM) on the downlink. The Time Division Multiple Access (TDMA) scheme appears to be more attractive for high data rate users. Because of a single carrier in TDMA the high power amplifier can operate in saturation. In addition, problems due to intermodulation distortion can also be eliminated. It is also possible to use several combinations of FDMA and TDMA schemes for transmitting the channel information via the satellite. One of the hybrid schemes that is gaining popularity is the Multi-Frequency Time Division Multiplexing Access (MF-TDMA).

One of the most economical schemes of transmission for low data rate users is SCPC/FDMA on the uplink and TDM on the down link. There are several advantages in using such a scheme for low data rate users, some of them being:

- Minimizes Effective Isotropic Radiated Power (EIRP) requirements.
- Eliminates ground network symbol synchronization.
- Makes full use of on board Traveling Wave Tube Amplifier (TWTA) power.
- Overcomes double hop which needs a hub station and introduces delay unacceptable for interactive voice and video communications.
- Reduces earth station complexity.

Despite these advantages, the major disadvantage involved in such a scheme of transmission is the task of regenerating the transmitted data on board the satellite. This needs extensive signal processing on board the satellite. In addition, the power and hardware requirements may be very large for the desired high throughput.

The FDMA/TDM system conventionally consists of a Transmultiplexer (TMUX), a bank of demodulators, baseband switch matrix, TDM multiplexer and a modulator as shown in Figure 1.1. The FDMA signal is first down converted from RF to IF. This wideband signal is downconverted and passed through an anti-aliasing filter, which is required before digitization. The signal is then sampled by an A/D converter. The digital signal at the output of the sampler contains information of all the SCPC carriers that need to be separated. The SCPC/FDMA channels are input to the Transmultiplexer (TMUX) which filters, separates and brings the channels to baseband. The bank of parallel demodulators extract the digital data for all the channels. This is followed by a baseband switch matrix which routes digital data to each of the channels. The channel data are then buffered into contiguous blocks, multiplexed, remodulated and transmitted back to the earth stations on a TDM down link.

The On Board Processing (OBP) system is called a Multi Carrier Demodulator (MCD). An MCD has two operations, namely, demultiplexing and demodulating.

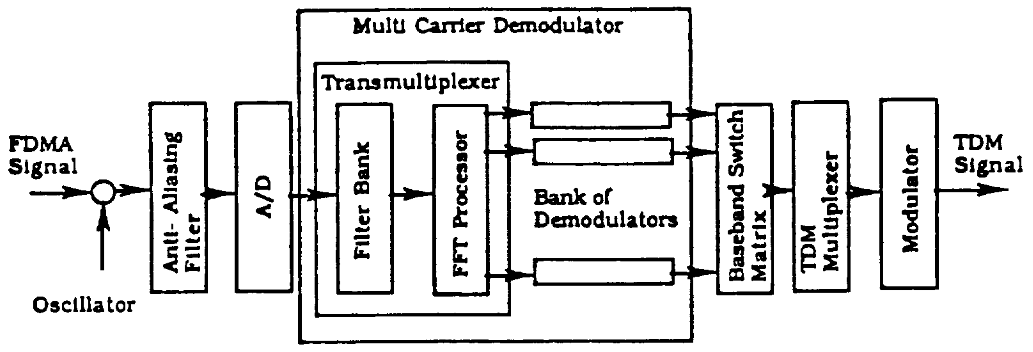

Figure 1.1: FDMA/TDM System

For the SCPC /FDMA scheme, much work has been done in the design of a Multi Carrier Demodulator (MCD) [1-11]. Several demodulators comprise the demodulating part of the MCD. Since a separate demodulator is suggested for each channel, the hardware requirements are severe. Hence, it is difficult to implement the MCD with reduced size and power requirements.

1.1 Proposed Research

A single hardware design is proposed for demodulating several channels simultaneously. The goal is to achieve high performance by designing a dedicated architecture. From a satellite point of view, reduced hardware and power requirements are ideal for On Board Processing. This research project presents an implementation of a single shared device to demodulate all the demultiplexed SCPC/FDMA channels. This concept is realized by incorporating the parallel, pipeline and time multiplexing techniques in the design. The time multiplexing saves on the hardware and the parallel-pipelined architecture provides the required speed. Because of the use of a single shared device, large savings in size and power are obtained while the architecture design and the availability of high speed VLSI chips allows the required throughput to be maintained. Flexibility is provided to process varying number of channels as long as it is not above the upper limit of the hardware design. Also a scheme for varying bit rate voice or data channels is given. It provides the flexibility to the user in terms of choosing between a higher number of low bit rate channels or a lower number of high bit rate channels. Hence this design is given the acronym PRODEM (PROgrammable DEModulator). The design is optimized for maintaining maximum bit rate. Also, if higher throughput is desired, several PRODEMs can be suitably used in parallel.

Simulations are carried out at different stages of this design. The demodulation algorithms are simulated as part of a MODEM. The input and the output bit streams of the MODEM are observed to check the accuracy of the operations. This simulation is done using a signal processing CASE (Computer Aided Software Engineering) tool called BOSS (Block Oriented System Simulator). Also, the hardware design is simulated with the data flow among the units and the modules. The control circuitry and interface of the modules are also simulated.

Also, speedup in the demodulation algorithm is easily achieved by using several processors of a multiprocessor system. One such multiprocessor sytem is a hypercube. A hypercube with n dimensions has 2^n processors. For example, the demodulation algorithm is divided into smaller units and is mapped onto a three dimensional hypercube consisting of eight processors. Each processor operates only on its assigned task. This enhances the performance of the system by providing considerable speedup. The creation and assignment of the tasks to the processors plays a crucial role in the performance of the hypercubes.

A hypercube architecture has fewer interprocessor communications and data transfer problems as compared to a time shared bus architecture. Also, in its class of multiprocessors, it is a tradeoff between its equivalent ring connected and completely connected topologies. Hence it is chosen for the simulation of our demodulation models for several channels. The algorithm models are simulated in Ada because it is easy to implement the parameters of a hypercube. Note that this approach is considered primarily for providing a speedup. Since the hypercubes have severe power and hardware requirements, it is not a solution for On Board Processing. However, this approach could be used for high speed demodulation needs of the earth stations. Using TDM on the down link will require all receiving earth stations to have relatively high speed demodulating requirements, hence this multiprocessor approach could be an attractive solution to the high speed requirement of the earth stations.

In Chapter 2, some of the existing satellite access techniques are reviewed and an overview of the parallel processing schemes is provided. Chapter 3 deals with the analysis of the carrier, timing and data recovery algorithms of the Multi Carrier Demodulator. Simulation of the algorithms is provided in this chapter. Chapter 4 deals with the hardware design from an on board processing stand point. The simulation of the hardware design is also discussed. The hypercube models and simulation of the demodulation algorithms for these models are provided in Chapter 5. Conclusions and future work are discussed in Chapter 6.

Chapter II

SYSTEM BACKGROUND AND PARALLEL PROCESSING OVERVIEW

2.1 Multiple Access Techniques

The users can access the satellite by multiplexing the data in frequency, time, or in code. Multiple access schemes have been used effectively and efficiently in the satellites. Some of the commonly used schemes are Frequency Division Multiple Access (FDMA), Time Division Multiple Access (TDMA), and Code Division Multiple Access (CDMA). In FDMA, each uplink RF carrier occupies a defined frequency slot and is assigned a specific bandwidth with a small guard band for separation of one carrier from another. The satellite receives all the carriers in its bandwidth, amplifies them and retransmits them back to the earth. The receiving station selects the desired carrier that contains its relevant message by appropriately choosing its allotted frequency. The main advantage of the FDMA access is that network synchronization is not required. It is used mostly by low data rate users.

The TDMA scheme uses a single carrier which is shared by all the users in time. It operates in burst mode such that the transmissions from all stations arrive at the satellite transponder successively. At any time each user has access to the entire transponder. The transmission timings of various bursts are carefully synchronized so that the bursts arriving at the satellite transponder from a group of users in the network are closely spaced in time but do not overlap. The satellite transponder receives one burst at a time, amplifies it and retransmits it back to the earth. Every earth station in the network will receive all the bursts of transmission from all the stations but selects only those that are relevant. A distinct advantage of TDMA over FDMA is that it uses a single carrier which avoids intermodulation distortion. Thus the satellite amplifier can operate in saturation to get the maximum output power. However, this scheme requires network synchronization.

The CDMA scheme has all uplink signals occupying the full frequency allocation at the same time. Each channel has its own pseudo-random code which distinguishes it from the other channels. The codes are chosen from an orthogonal set and are used to separate the desired signals. This scheme is primarily used in military applications for security purposes. The commercial applications of CDMA are beginning to emerge for low speed data communications. These schemes and several other hybrid schemes are discussed in detail in [15].

2.2 Satellite System Users

The various satellite users can be divided into three main categories depending on the bit rate they need for the transmission. They are the low, medium and high data rate users as mentioned in [9].

1. Low Data Rate (LDR) users require one telephone channel or less (16/32 Kbps). LDR can be arbitrarily classified as any traffic having bit rate in the range of 1 Kbps to 100 Kbps. Under this category, signals can be considered carrying "telematic" services, as Teletex, Videotex, Low/Medium Speed Facsimile, Slow-Scan Video, wideband PCM "toll quality" telephony (64 Kbps) or reduced rate "talk quality" PCM telephony (less than 10 Kbps).

2. Medium Data Rate (MDR) users require up to 10 Mbps (e.g., digital T1 signals at 1.544 Mbps or PCM hierrarchic levels as 2.048 Mbps or 8.448 Mbps. High speed Facsimile, Videophone, and Video conference are the main applications for this bit rate range.

3. High Data Rate (HDR) users requirement ranges from 10 Mbps to 150/200 Mbps (or more) which is considered today an upper practical limit to the per-transponder capacity. This range of data rate is essentially devoted to multiplexed voice circuitsdigital television or HDTV planned for broadband ISDN at 155 Mbps. Each high data rate user may require one or more transponders. A Time Division Multiple Access

(TDMA) system or a single access may be used for such HDR systems, needing no particular improvement of today's technology. On the other hand, LDR and MDR signals may be processed in several different ways to reduce the complexity of the user terminal or on board hardware. It could also be improved from a speed and power requirement view point. In this research effort, we shall be concerned with the implementation of LDR/MDR system design only.

2.2.1 LDR/MDR System Configurations

2.2.1.1 SCPC/FDMA Approach

The use of Single Channel Per Carrier (SCPC) technique using a Frequency Division Multiple Access (FDMA) on the uplink and the downlink offers the desirable feature to design the earth station EIRP for the user capacity. However, a major problem with the FDMA systems is the presence of intermodulation products in the composite signal bandwidth generated by the amplification of multiple carriers by a common Traveling Wave Tube Amplifier (TWTA). The TWTA in the satellite transponder exhibits both amplitude and phase nonlinearity. As the number of carriers increases, it is necessary to operate the TWTA near saturation in order to supply the required power per carrier to reduce the effect of downlink thermal noise. But near saturation, the input/output amplitude transfer characteristic of the TWTA is highly nonlinear. Consequently the level of intermodulation products is increased, which effects the overall performance. Thus the TWTA must be backed off from saturation and operated in the quasi-linear mode to obtain an acceptable value of the carrier-to-intermodulation product ratios.

At the optimum backoff (up to 6 dB at the TWTA output) the reduction of the satellite EIRP and the residual intermodulation products reduce the downlink carrier-to-noise ratio (C/N or Eb/No), typically 2-5 dB with respect to single carrier operation. It is possible to reduce the average output power by 50% or more to reduce the intermodulation products to an acceptable level with a high density of input signals. However this will cause problems at the receiver, because the downlink signal with reduced carrier-to-noise ratio cannot be received by earth stations with small antennae.

2.2.1.2 FDMA/TDMA Approach (Double Hop)

The best features of both the FDMA and the TDMA could be obtained using a modulation conversion on board a regenerative satellite or on ground. For modulation conversion on ground the small earth stations are connected using a double hop through a large Central Processing Station (CPS). In such a configuration, the first hop is from the small capacity stations to a large CPS via the satellite. It uses SCPC/FDMA scheme of transmission as shown in Figure 2.1. The CPS on the ground demodulates the received SCPC signals and remodulates on a single carrier and retransmits back to the satellite. This is the second hop from the CPS to the small capacity users on a TDMA format. Therefore the transmitting and receiving stations are linked by the satellite. The CPS has a forward link in FDMA and the return link in TDMA.

By using this scheme the users access the satellite freely by SCPC/FDMA uplink. The low data rate users are collectively retransmitted as high data rate using the TDMA scheme by the CPS. An advantage of this scheme is the reduction of cost for low data rate users. For these low bit rate users, the cost involved in using TDMA in the uplink is impractical. In this scheme, this is avoided by using FDMA on the uplink and by a FDMA/TDMA conversion using a CPS. Therefore, the cost involved in using TDMA can be distributed among all its users. Yet another advantage is that, due to the increased dimension of the CPS antenna, the satellite can transmit the SCPC signals at a lower carrier power. Due to this fact the HPA power on board and the satellite EIRP are reduced. Therefore the RF power usage on the satellite is minimized.

In the second hop, the TDMA transmission can achieve efficiencies in satellite power utilization of 90% or more compared to the 50% loss in the satellite average output power that is typical of FDMA operation. A disadvantage in this scheme is an introduction of excessive delays due to the double transmission. Note that for most applications of voice and interactive video transmission this delay is not acceptable. Also the net throughput of the satellite is reduced as it is accessed twice for the transmission of the same data.

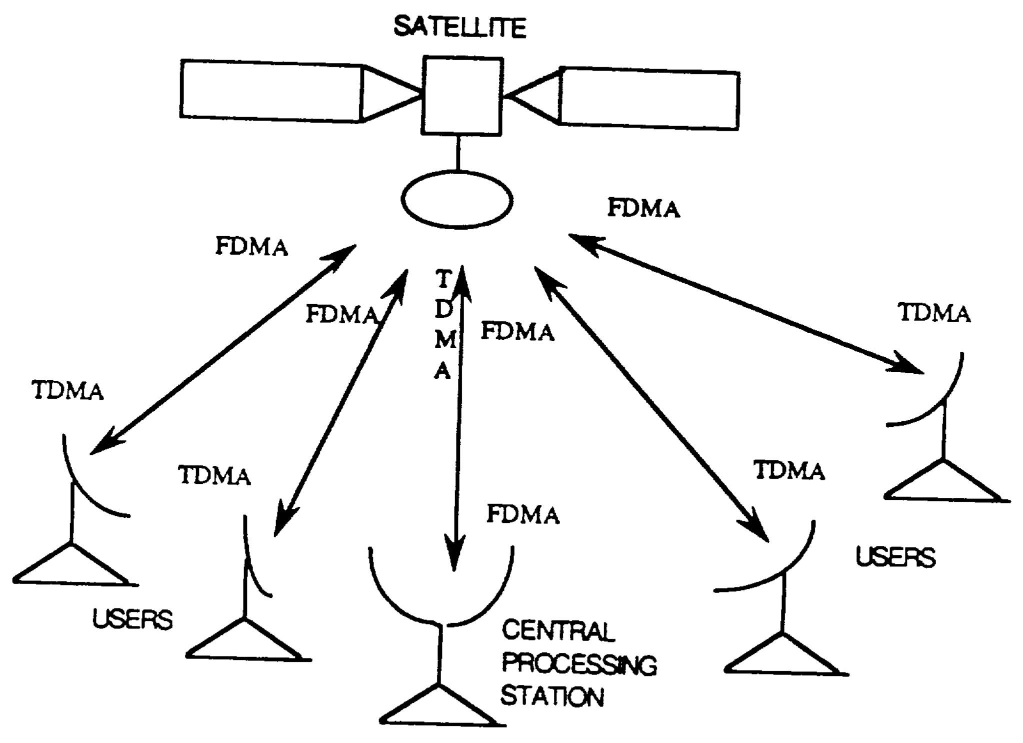

Figure 2.1: SCPC/FDMA Double Hop Scheme

2.2.1.3 FDMA/TDM Approach (Multi Carrier Demodulation)

The disadvantages of the double hop scheme can be overcome if the burden of the processing done by the CPS is transferred on board the satellite. The trend has been to use regenerative repeaters for digital transmission. In a regenerating transponder the digital signal is demodulated and remodulated within the transponder itself. This scheme separates the uplink and the downlink into independent paths. This will require baseband processing and frequency conversion of the modulation on board the satellite. Low power, narrow band, digitally modulated carriers operating on a frequency division basis on the uplink could be demodulated on board the satellite. The individual bit streams can then be combined on a down link using Time Division Multiplexing (TDM). Also the continuous wave transmission in the FDMA mode has its advantages. Due to the continuous transmission of the RF power, the demodulators on board do not have to acquire symbol timing and phase recovery every time data are transmitted. Instead the demodulators have to track the symbol timing and the carrier phase to keep it from drifting. Thus the bit energy requirement is reduced. Due to this reason the earth stations can transmit signals with a low C/N (yet it will be sufficient for the accurate operation of the demodulators). Therefore the earth station EIRP, the HPA power and the antenna size are reduced. At the receiving end, the signals received are in a TDM mode. The single carrier characteristic of the TDM mode allows satellite transponder operation in saturation. This results in an efficient utilization of the onboard RF transmit power.

Thus in satellite communication systems incorporating small earth stations, the use of SCPC/FDMA on the uplink, regeneration and remodulation of the user data on board the satellite and use of TDM on the downlink are significantly effective in improving the satellite transponder utilization and reducing the required EIRP in both the satellites and earth stations.

As discussed, it is observed that the system complexity on board using multicarrier demodulation makes such a regenerative system rather attractive and flexible. However it is not easy to implement on board MCD with low power and reduced hardware yet provide the system with a computational efficiency. Some investigations for on board processing have been carried out in various technologies such as Surface Acoustic Waves (SAW), acousto-optical techniques and baseband digital signal processing [9]. In this research, an efficient design is proposed using the baseband digital signal processing scheme. This design provides a low power and hardware complexity solution to the on board conversion of the FDMA/TDM system.

2.3 Parallel Processing

Advances in many fields had their impact on the design and development of advanced computing technologies. Some technologies need millions of instructions to be computed in a fraction of time. This relates to providing the user with a machine that supports high Million Instructions Per Second (MIPS) . There are some real-time applications that need this speed for an accurate operation of the system. Others need this speed for running the simulations that ordinarily take hours and days to provide results. This could be a waste of valuable research time. Yet others need this computational power simply because the computation time is more than the time allowed for computations. Parallel processing aims at providing maximum possible speedup to the end user. Some of the applications which use parallel processing are given in Table 2.1. A comprehensive study of these applications can be found in [26]. Any field that needs a high speed computation can find a solution in parallel processing.

Parallel processing is a means of computing using several processors or processing units. Its objective is centered around increasing the speed of computation which is achieved by using multiple units. The algorithm needing a speed up is appropriately partitioned and mapped onto these multiple units that operate either simultaneously or in a pipeline. This is achieved by observing the partitions of the algorithms which can be done simultaneously and those which can be done in overlapped time intervals. The former is called parallelism and the latter is called pipelining. In contrast to sequential processing, parallel processing demands concurrent execution of many events in the system. Special purpose dedicated hardware, advanced computer architectures, and supercomputers are based on the principles involved in parallel processing. The application of these architectures and algorithms [24-30] need the underlying principles involved in hardware and software structures and close interactions between algorithms and optimum allocation of the machine resources in solving large scale computing programs.

Table 2.1: Some Fields That Need Parallel Processing

- Modeling and Simulation
  - Weather forecasting
  - Oceanography and astrophysics
  - Socioeconomics and government use
- Engineering design and automation
  - Finite element analysis
  - Computational aerodynamics
  - Artificial Intelligence and automation
  - Remote sensing applications
  - Image and signal processing
  - Satellite communications
- Energy resources exploration
  - Seismic exploration
  - Reservoir modelling
  - Plasma fusion power
  - Nuclear reactor safety
- Medical, military and basic research
  - Computer Assisted Tomography (CAT scan)
  - Genetic engineering
  - Weapon research and defense
  - Space research
  - Basic research problems

2.3.1 Issues and Concerns in Parallel /Pipeline Processing

Parallel processing of the algorithms can be achieved in many cases where the algorithms can be divided in parallel. The problem is to identify the portions of the algorithm that can efficiently use more than one processor or a hardware unit. The effectiveness comes from the identification of the problem that lends itself to the parallelism, allocation of the algorithm, and mapping it on to a suitable architecture. The algorithms could be suitably mapped onto the available architectures by analyzing the characteristics of the algorithms and considering the parameters that effect the performance of the mapped algorithms on the architectures. One has to exploit the organization of the memory and architecture to attain a high speed as compared to a sequential solution of the same.

If the algorithm cannot be broken down into parallel units, it can be divided into units which are pipelined. This needs suitable interprocessor communication and synchronization techniques. Interprocessor communication pertains to an accurate transfer of the data from one processor to another. Based on the architectures either it is very simple to implement as in a hypercube or is complicated and affects the performance if the architecture is a time-shared common bus architecture. Synchronization is achieved mainly for processors to operate in conjunction with one another. The processors operate in such a manner that if any processor completes its job earlier than the others, it waits for others to come to its level so as to ensure accurate operation on the shared data. This ensures a synchronized performance of all the processors in the system. In a hardware architecture, synchronization will relate to transfer of data at the right instants of time. It may also relate to interfacing several interacting modules.

Chapter III

ALGORITHM ANALYSIS

Demodulation algorithms pertain to the recovery of carrier, data and timing. The algorithms are analyzed from a view point of providing implementations for a suitable hardware design. The analysis in terms of the interdependency of the data flow in the algorithms is noted. This study aims at developing a dedicated architecture. It is achieved by performing the data dependency analysis on the algorithm. The algorithm is examined for data dependencies and is appropriately partitioned. The underlying principle is to map the independent sections of the algorithm onto computational units operating in parallel and the dependent sections onto units operating in a pipeline [17-18]. This is implemented within the units of the modules and also among the modules.

A digital implementation of a demodulator consists of several modules which operate in cooperation with each other. A Quadrature Phase Shift Keying (QPSK) modulation scheme is used in this demodulator. A block diagram of a QPSK demodulator with its interfacing modules is shown in Figure 3.1. The interpolator provides samples of the incoming symbols at the precise rate of two samples per symbol, and at the precise positions of the symbol which are at its peak and crossover points. The interpolator is needed to properly sample the symbols needed at the input of the demodulator. This is because the outputs from the demultiplexer are not suitable for the demodulator where a fixed integer number of samples are required. The carrier recovery module estimates the carrier phase offset. This phase offset is introduced in the system due to the atmospheric disturbances, Doppler shift, etc. The clock recovery module extracts the correction for the timing information of the channels. This is required to ensure accurate sampling instances at the maximum eye opening positions. There are several algorithms for the recovery of carrier and clock. The non-linear estimation technique proposed by Viterbi and Viterbi [12] for the carrier recovery and the clock estimation method proposed by Gardner [13] have been chosen in our implementation. The carrier recovery algorithm provides a good estimate of the phase and is not very sensitive to the finite arithmetic implementation which has a bearing on the word-length. It requires a short acquisition time, and it does not have many feedback loops which are characteristic of certain other algorithms. These attractive features make it suitable for the joint recovery of all the channels. The clock recovery algorithm is independent of the carrier phase and also offers some attractive features for efficient implementation for multiple channels. Due to the atmospheric disturbances, propagation delays and the Doppler shift effects present in satellite communication, it is essential to continuously track the phase and timing estimates.

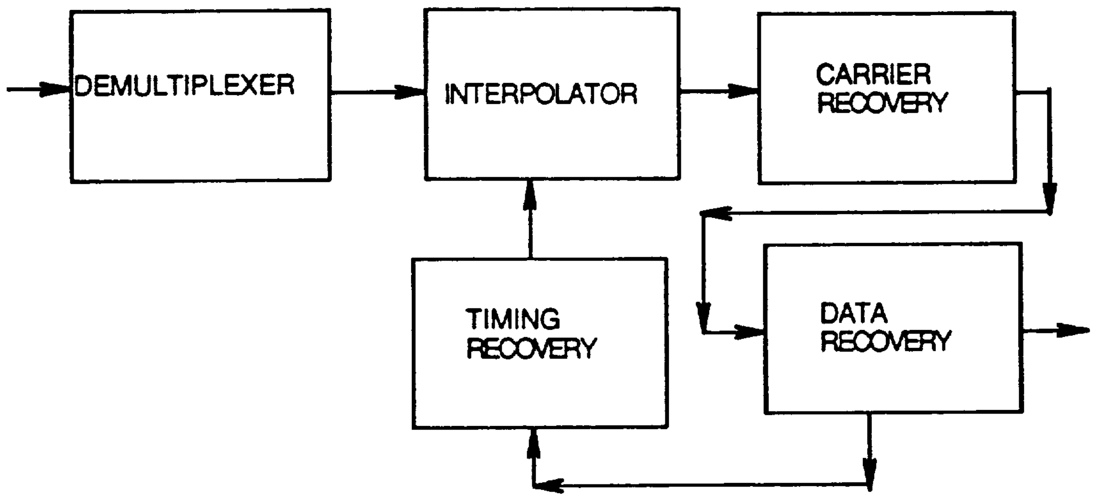

Figure 3.1: QPSK Demodulator

3.1 Carrier Recovery Algorithm

The carrier recovery is based on the implementation of the Viterbi algorithm in the proposed MCD structure by E. Del Re and R. Fantacci [6]. The carrier phase of the

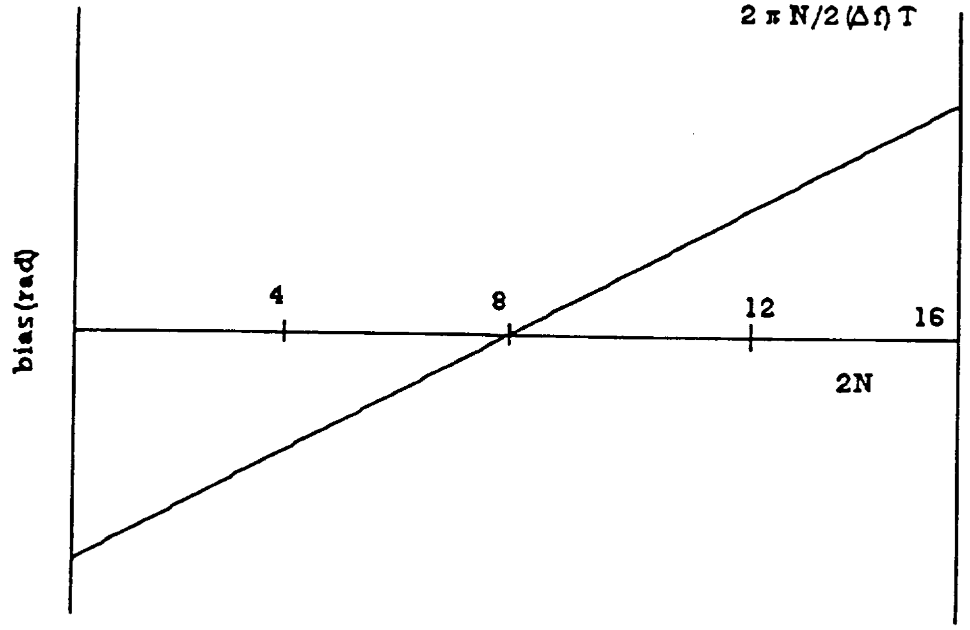

Figure 3.2: **Biased Carrier Phase Estimation**

incoming samples can be estimated continuously or periodically. The continuous estimate is obviously computation-intensive but is an unbiased estimate. Alternatively, an estimate can be made at the mid-interval symbol with N symbols preceding it and N symbols succeeding it where N is the number of contiguous symbols of the same user. The estimation interval is then 2N+1. This symmetric estimation of the carrier phase is a biased estimate. This bias depends on the position of the symbol from the mid-interval symbol as shown in Figure 3.2. The phase is unbiased for the mid-interval symbol and is a linear function of the position of the symbols away from it. This bias is maximum for the symbols at the end points of the estimation interval.

The operations that are performed in the carrier recovery algorithm are given in this section. The flowchart for the carrier recovery algorithm is as shown in Figure 3.3. Let T and Te be the duration of the symbol and the estimation intervals, respectively. By our earlier discussion, these parameters are related by the equation

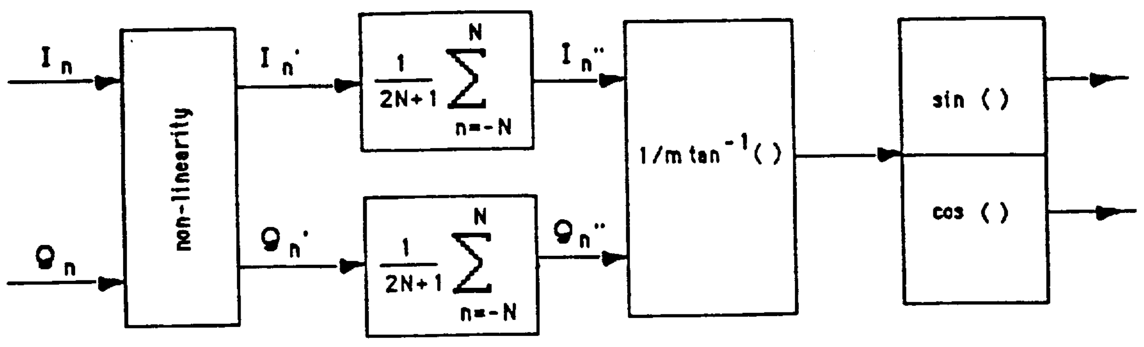

Figure 3.3: Carrier Recovery Algorithm Flowchart

(2N+1)T = Te. The inputs to the phase recovery are the in-phase and the quadrature-phase samples. Let the complex sample input be of the form:

$X(n) = I_n + j\,Q_n$ (3.1)

For PSK modulation,

$I_n = A_n \cos\phi_c - B_n \sin\phi_c$ (3.2)

$Q_n = A_n \sin\phi_c + B_n \cos\phi_c$ (3.3)

where ϕ_c is the carrier phase and A_n and B_n are the in-phase and quadrature-phase sampled data waveforms. These equations can also be written as:

$I_n = R_n \cos(\pi(2k+1)/m + \phi_c) = R_n \cos \phi_n$

$Q_n = R_n \sin(\pi(2k+1)/m + \phi_c) = R_n \sin \phi_n$

where $R_n = \sqrt{A_n{}^2 + B_n{}^2}$

$\phi_n = \pi(2k+1)/m + \phi_c$ (3.4)

m = number of phases

$k = 0, 1, 2, \ldots\ldots, m-1.$

Note that R_n is also equal to $\sqrt{I_n^2 + Q_n^2}$ and ϕ_n is $\tan^{-1}(Q_n/I_n)$. Equation (3.1) can also be written in the polar form as:

$$X(n) = R_n . e^{j\phi_n} \qquad (3.5)$$

A non-linear transformation is performed to eliminate the modulation information from ϕ_n. The new transformed vector is given by:

$$X'(n) = F(R_n).\exp(jm\phi_n).$$

$$= F(R_n).\exp(jm\phi_c). \qquad \text{from} \quad (3.4)$$

The function F is an arbitrary non-linear operator. The non-linear operation is used to remove the modulation information from the signal. The next step performed in the recovery of the carrier phase is to convert the polar form back to the rectangular form. This gives:

$$F(R_n).\exp(jm\phi_c) = I_n' + jQ_n' \qquad (3.6)$$

To obtain an estimate $\hat{\phi}_c$ of ϕ_c over the estimation interval Te, mean vector values are needed. These are obtained by:

$$I_n'' = (1/(2N+1))\Sigma I_n' \qquad n= -N \text{ to } n= N \qquad (3.7)$$

$$Q_n'' = (1/(2N+1))\Sigma Q_n' \qquad n= -N \text{ to } n= N \qquad (3.8)$$

The phase estimate is obtained as

$$\hat{\phi}_c = 1/m \tan^{-1}(Q_n''/I_n'') \qquad (3.9)$$

The operations performed for obtaining I_n' and Q_n' are rectangular to polar transformation, a phase multiplication by m, an arbitrary non-linear transformation on R_n and finally a polar to rectangular transformation. In the hardware implementation, the values of I_n' and Q_n' can be preprogrammed and stored in a look up table for any given I_n and Q_n values.

All the samples present in the estimation interval are summed and then averaged. The mean vector values I_n'' and Q_n'' are obtained. Corresponding to these values, a $1/m \tan^{-1}$ operation obtains the phase offset. The sine and cosine values of the phase output are then computed. These values are output from this block and are utilized appropriately by the data recovery module. Note that multiplying the phase by m and finally dividing the $\tan^{-1}$ function by m, gives rise to a m fold ambiguity in the phase estimate. However, in a practical situation this problem is solved by coding the data transitions rather than the data themselves (differential coding) and by performing differential decoding at the receiver. It could also be solved by considering all the positive and negative values of I_n'' and Q_n'' and creating a look up table for all these combinations.

3.2 Timing Recovery Algorithm

The clock recovery module extracts the timing information for each channel. A general structure of the timing recovery mechanism is shown in Figure 3.4. Samples from the I and Q channels are input to the timing error detector. The timing error information is used to update the timing instances by the timing error corrector. The output of the timing error corrector drives the interpolator. After this feedback input the interpolator outputs correctly sampled data. Figure 3.5 shows typical sample points used in the estimation of the timing error information. Samples I_n and I_{n-1} are at the peak points and $I_{n-1/2}$ is at the cross over point. A transition between the samples should have a zero mid-way sample. A non-zero value indicates an error which has to be corrected. For this purpose, the successive samples are compared. The difference between the peak values will provide the slope information. The product of the slope information and the midway sample provides the timing information. A timing correction is provided only when a transition exists. The timing error T_i for the I - channel is:

$$T_i = [I_{n-1/2} \{I_n - I_{n-1}\}] \qquad (3.10)$$

The flowchart of the algorithm is shown in Figure 3.6. Clearly the amplitude of the mid sample $I_{n-1/2}$ is proportional to the timing error T_i. The difference of the peak

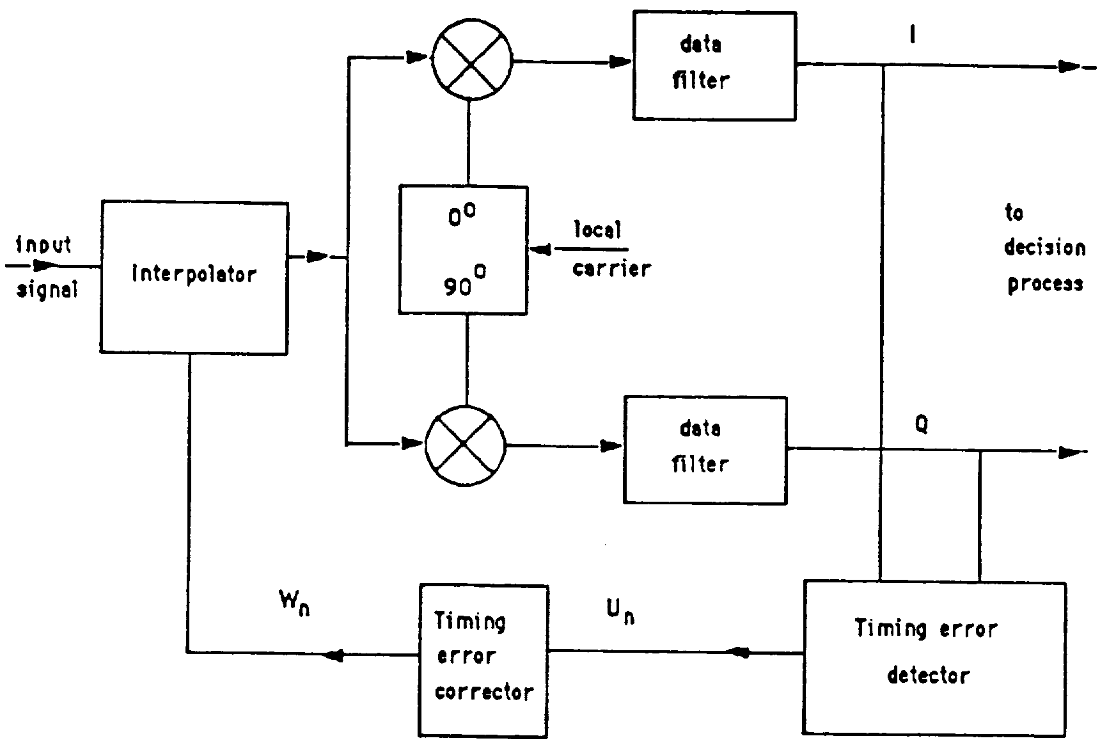

Figure 3.4: Timing Recovery Mechanism

sample values $\{I_n - I_{n-1}\}$ magnifies the amplitude of the sample at the crossover point. In fact, instead of using the actual values of the peak samples, their sign values can also be used. This will also simplify hardware implementation. Note that when there is no transition between the adjacent samples, the timing error is zero. Also the sign of the timing error is used to determine the direction of the timing correction. For a general PSK modulation, samples from both the I and Q channels are used to obtain the timing error which is given by

$$U_n = T_i + T_q$$
$$= I_{n-1/2} \cdot \{ I_n - I_{n-1} \} + Q_{n-1/2} . \{ Q_n - Q_{n-1} \} \qquad (3.11)$$

The cumulative timing error is given by

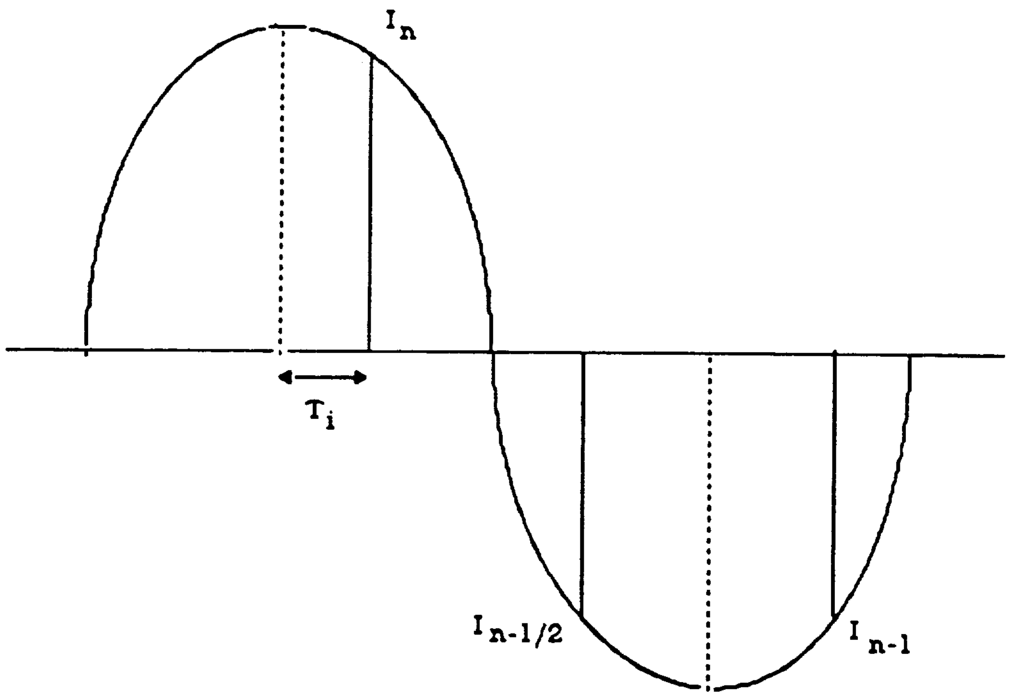

Figure 3.5: Timing Error Estimation

$$W_n = W_{n-1} + U_n \tag{3.12}$$

The error value is used by the timing error corrector to obtain the correct updated timing. This updated output is needed by the interpolator filter for the correction of the sampling instances. The correction keeps the timing instances free of any lead-lag erroneous sampling. The various possible combinations of sampling are shown in Figure 3.7. The ideal sampling would be to obtain the optimum peak and the zero-crossing values of the samples.

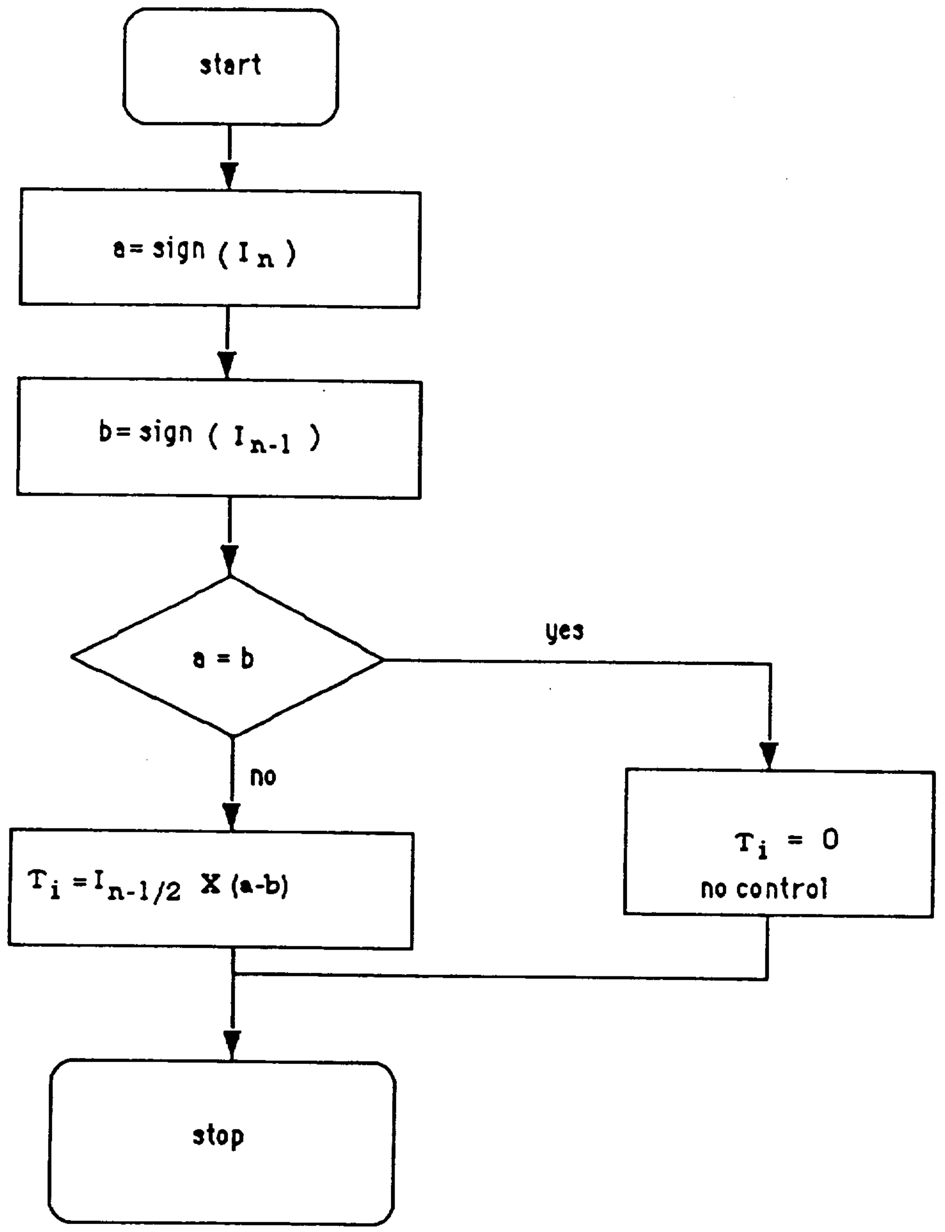

Figure 3.6: **Timing Recovery Algorithm Flowchart**

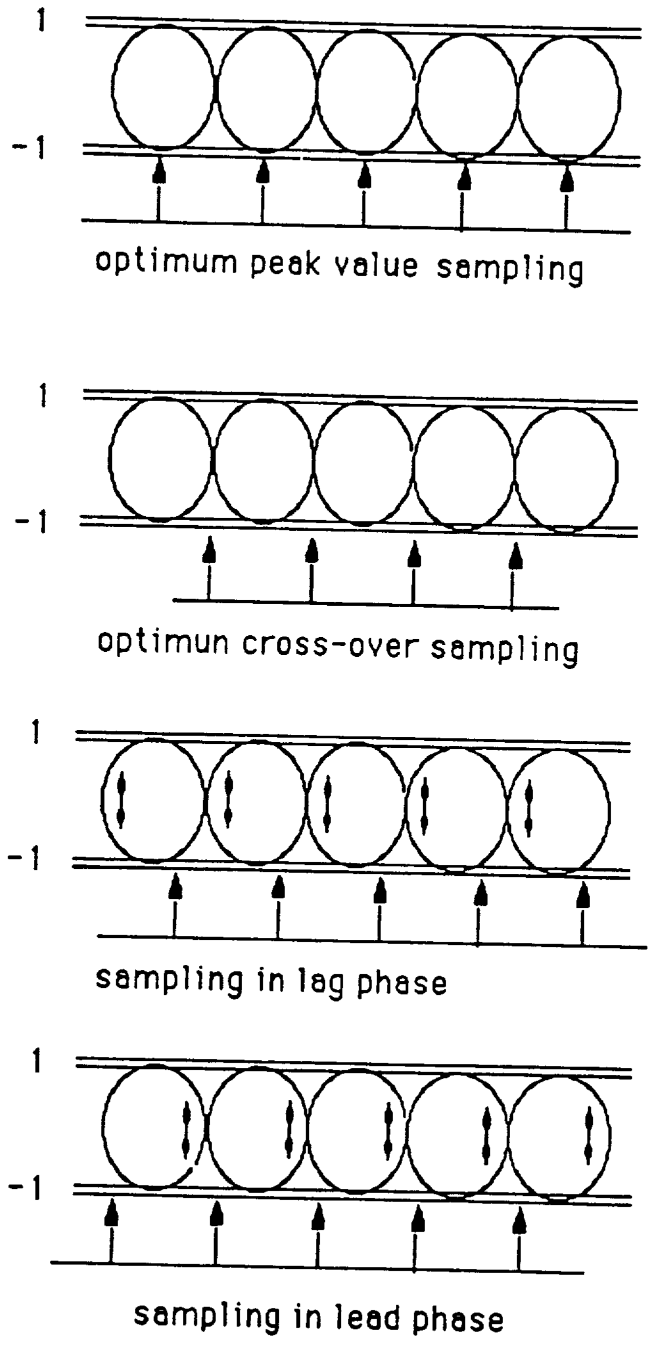

Figure 3.7: **Various Sampling Instances**

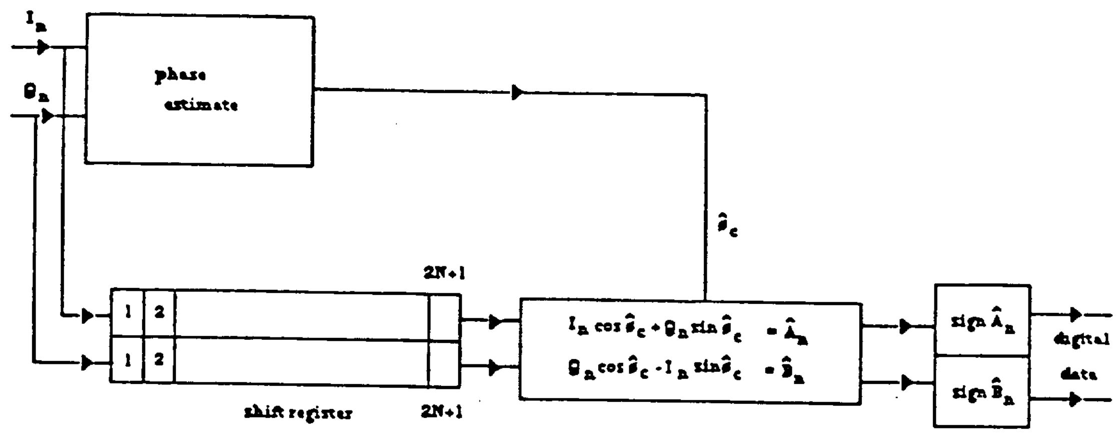
I_n
Q_n
phase estimate
$\hat{\phi}_c$
2N+1
1 2
1 2
shift register
2N+1
$I_n \cos\hat{\phi}_c + Q_n \sin\hat{\phi}_c = \hat{A}_n$
$Q_n \cos\hat{\phi}_c - I_n \sin\hat{\phi}_c = \hat{B}_n$
sign $\hat{A}_n$
sign $\hat{B}_n$
digital data

Figure 3.8: Data Demodulation Scheme

3.3 Data Recovery Algorithm

The demodulation process for the samples begins once the phase estimate is available. The data demodulation scheme is similar to [11] and is shown in Figure 3.8. A shift register is required to store 2N+1 samples. This is because 2N+1 samples are used to estimate the phase. During this time, demodulation cannot be carried out. So the sampled data are buffered. Demodulation can begin only after the phase estimate is available. This will result in an initial delay of 2N+1 symbols in the demodulation process. Using equations (3.2) and (3.3) to compute An and Bn shown in Figure 3.8, the following results are obtained.

$$\hat{A}_n = I_n . \cos \hat{\phi}_c + Q_n . \sin \hat{\phi}_c \quad (3.13)$$

$$= A_n . \cos \hat{\phi}_c \cos\phi_c - B_n . \sin\phi_c \cos \hat{\phi}_c + A_n . \sin \hat{\phi}_c \sin\phi_c + B_n . \cos\phi_c \sin \hat{\phi}_c$$

$$= A_n . \cos (\phi_c - \hat{\phi}_c) - B_n . \sin (\phi_c - \hat{\phi}_c) \quad (3.14)$$

$$\hat{B}_n = Q_n . \cos \hat{\phi}_c - I_n . \sin \hat{\phi}_c \quad (3.15)$$

$=A_n \cdot \sin\phi_c \cos \hat{\phi}_c + B_n \cdot \cos\phi_c \cos \hat{\phi}_c - A_n \cdot \cos\phi_c \sin \hat{\phi}_c + B_n \cdot \sin\phi_c \sin \hat{\phi}_c$

$$=A_n \cdot \sin(\phi_c - \hat{\phi}_c) + B_n \cdot \cos(\phi_c - \hat{\phi}_c) \tag{3.16}$$

when $\hat{\phi}_c$ is the estimate of ϕ_c. Equation (3.14) provides a close estimate of the A_n symbol and equation (3.16) provides a close estimate of the symbol B_n.

3.4 MODEM Simulation for Demodulation

A simulation of the demodulation algorithms is performed using a CASE tool called Block Oriented System Simulator (BOSS). The carrier and data algorithms are verified for a MODEM with a QPSK scheme of modulation as shown in Figure 3.9. The timing algorithm needed the interpolator algorithms for verification and hence was not simulated.

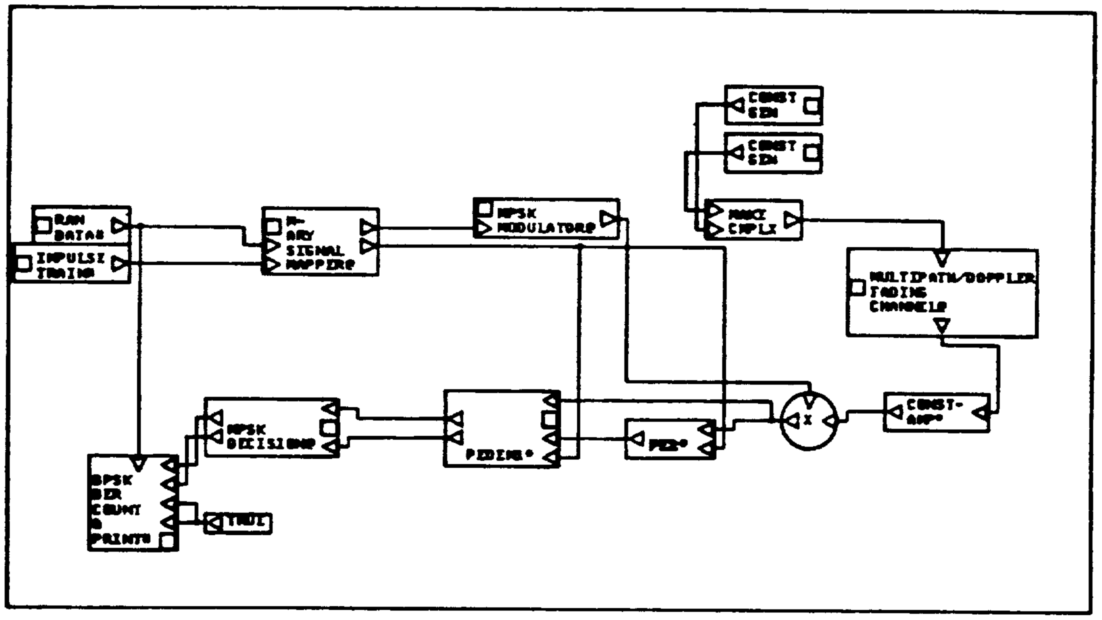

Figure 3.9: Software Model For The Simulation Of MODEM

Several MODEMs have been simulated with varying parameters in an extensive study [16]. From this library, a QPSK MODEM is selected for the purpose of verifying our algorithms. The MODEM has a random input datum (0 or 1) available at every clock cycle generated by the block RAN DATA as shown in Figure 3.9. This is mapped into the four different constellations of QPSK by the block SIGNAL MAPPER which uses two inputs to represent any of the four positions. The modulator uses this input to modulate the input information to the four different phases associated with these positions. The block PE2* gets an input which is a combination of the modulated phase and the phase jitter. Th multipath fading channel block is used to generate a phase disturbance and the const-amp block outputs this signal which has only a varying phase value. In reality, this is obtained from the disturbance and Doppler shift, etc. Therefore phase jitter along with the phase of the actual data from the modulator are input to the phase estimator block. This unit rotates out the modulated phase so that a phase which is purely related to the disturbance is extracted. It is used by the demodulator block PEDEM1* to extract the original modulated phase. A reverse process is done at the receiver to decide on the bits and BER count block checks for the bit error rate by comparing the input and the output bit streams. The programs written for the blocks CONST-AMP*, PE2* and PEDEM1* are listed in Appendix A.

Several parts of the MODEM are probed to examine the changes in the signal during this process. The signal at the output of the modulator block is shown in Figure 3.10. This corresponds to the modulated phase levels. The output of the phase jitter is shown in Figure 3.11 which is the simulation of the disturbance introduced onto the signal. The combination of these outputs results in the actual input signal to the demodulator. This input signal to the demodulator block PEDEM1* is shown in Figure 3.12. The carrier phase is recovered by the PE2* block. It rotates out the modulation and extracts the carrier phase as shown in Figure 3.13. The demodulator uses this extracted phase to determine the actual phase levels. These levels correspond to the original information which has been modulated . The output of the demodulator block PEDEM1* is shown in Figure 3.14. The input and the output bit streams are shown in Figure 3.15. It is seen that the two bit streams match very closely and thus the accurate

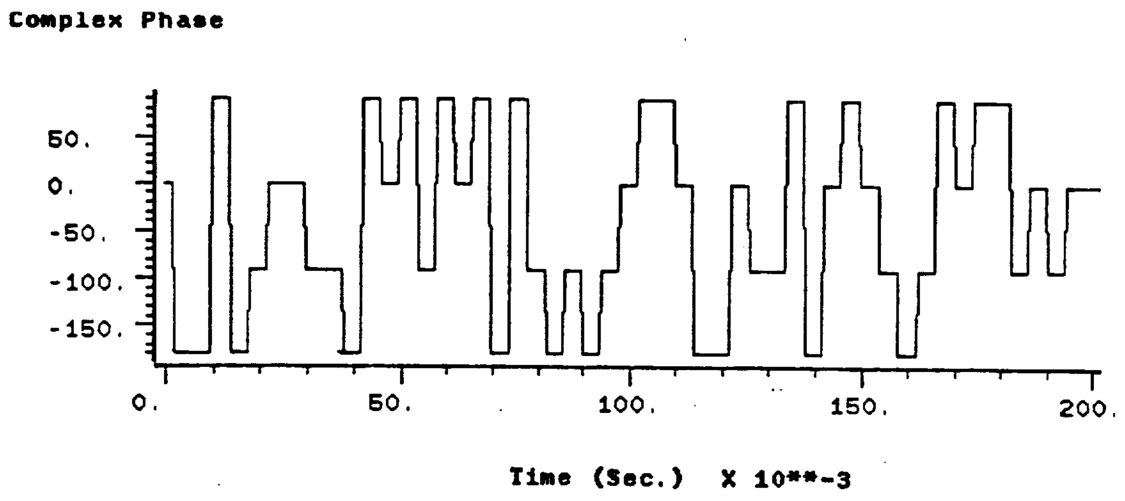

Figure 3.10: Output Signal From The Modulator

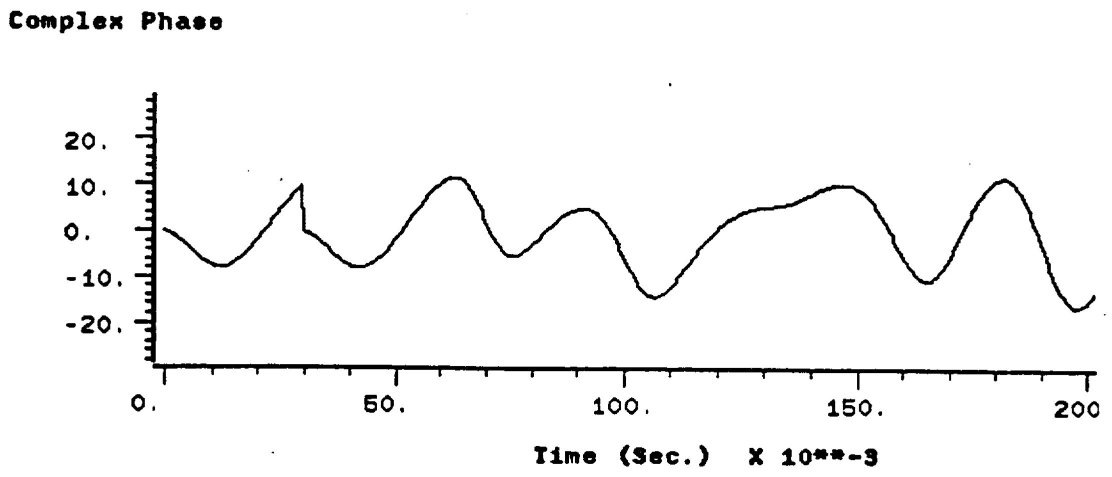

Figure 3.11: Phase Jitter Introduced By Atmospheric Disturbances

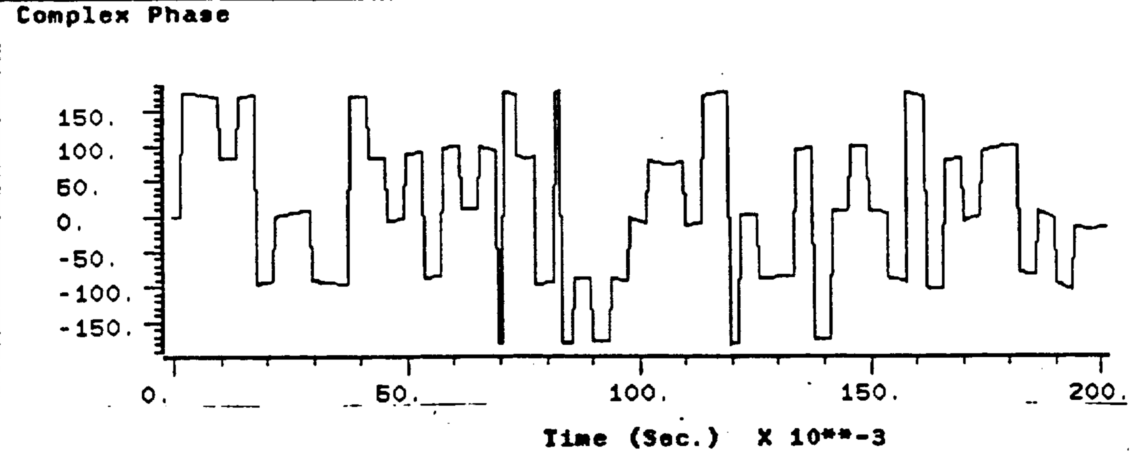

Figure 3.12: Phase Signal At The Input Of The Demodulator

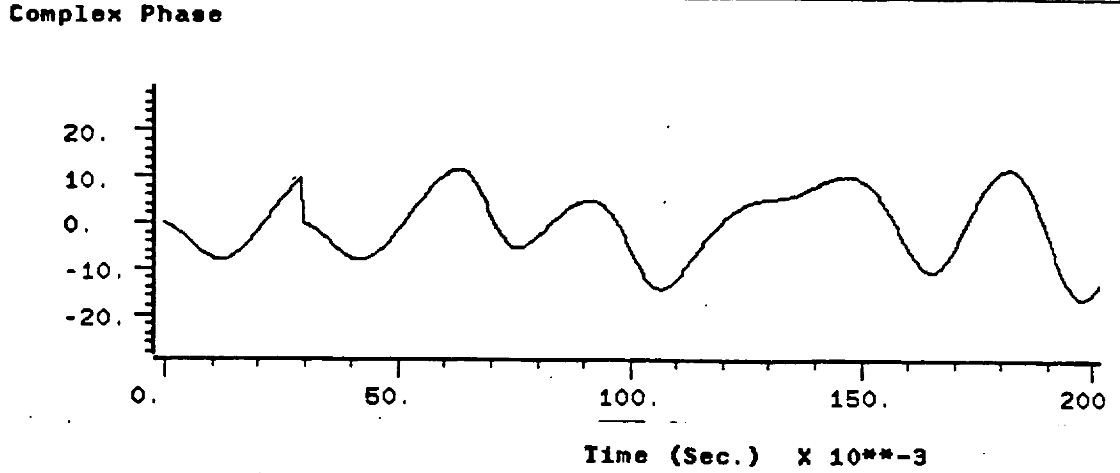

Figure 3.13: Phase Estimation For Extraction Of Phase Jitter

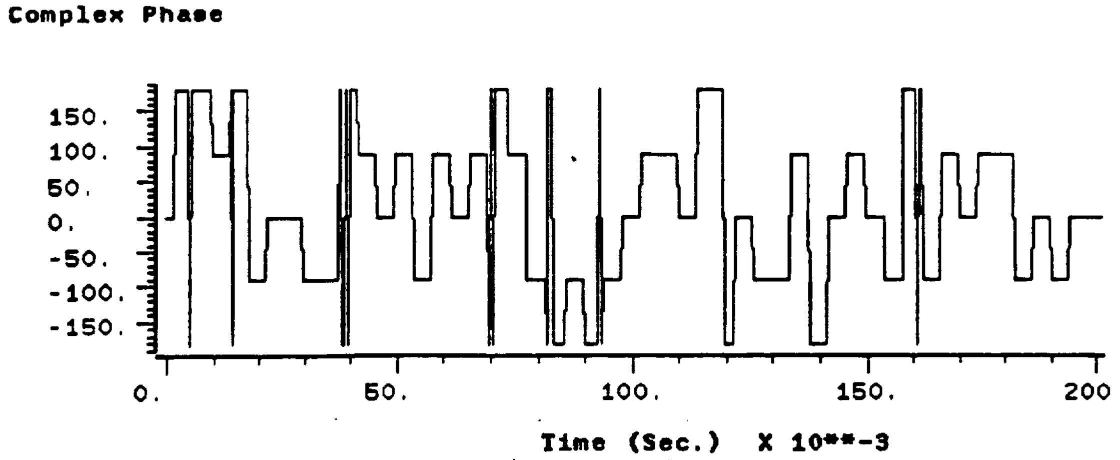

Figure 3.14: Output Of The Demodulator

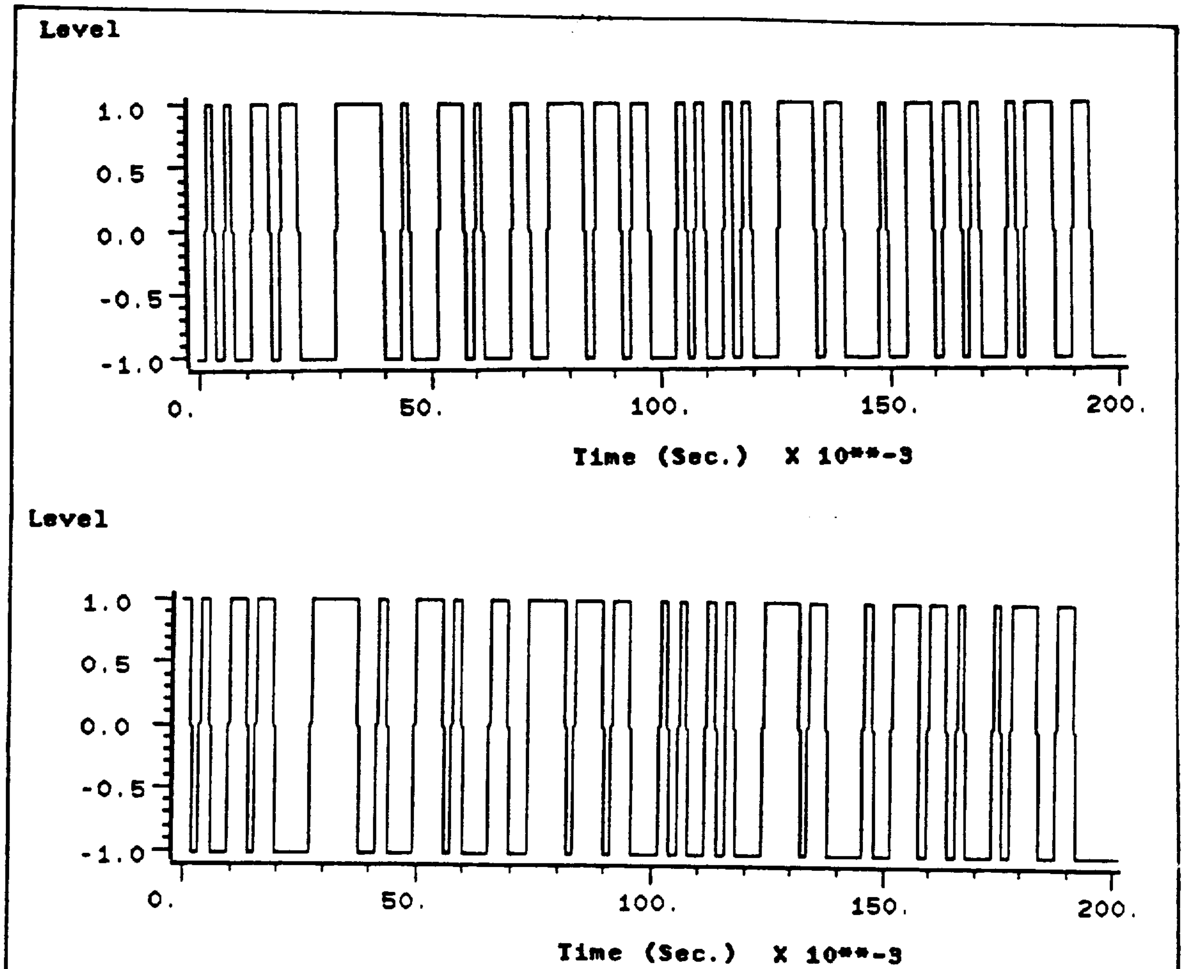

Figure 3.15: Comparison Of The Input And The Output Bit Streams

operation of the demodulation algorithms in a MODEM is verified. The simulation is test run for about 10,000 bits and the BER is noted. The listing of the programs with the Bit Error Rate (BER) are given in Appendix A.

Chapter IV

THE DEMODULATOR DESIGN METHODOLOGY AND SYSTEM SIMULATION

A hardware design is developed for demodulating several voice channels pertaining to the SCPC/FDMA system. This design incorporates pipelining and parallel processing techniques. The implementation of these techniques in the hardware design increases the performance of the system [22, 23]. The speedup achieved can then be utilized effectively to process a large number of channels which are multiplexed. The design attempts to provide a proof-of-concept for processing a large number of channels.

4.1 Preview To The Design Of The Demodulator

The A/D sampler in Figure 1.1 is used to sample the received FDMA signal. The signal at the output of the sampler contains information of the N SCPC channels. If each channel has a uniform spacing of Δf, then the combined bandwidth will be $F= NX.\Delta f$. The sampler will have to operate at least at the rate of (2F) to accurately retrieve the channel information. However, if complex sampling is performed, the sampler can operate at a reduced rate R=(4F). The filter bank and the FFT processor of the Transmultiplexer (TMUX) and the bank of demodulators need to operate at this rate for processing the data.

The bank of demodulators operate at the rate at which the samples are input to each demodulator. Since N demodulators are used corresponding to N channels, each demodulator operates at a reduced rate given by (NR) time units. This usually does not keep the hardware units of the demodulator operational at every clock cycle of the system. The process of keeping the units inactive for several clock cycles results in an ineffective use of the hardware resources. To overcome this drawback, a single hardware device is proposed to demodulate all the channels. The hardware units of this device will be operational at every clock cycle. The channels will be multiplexed and share the hardware resources of this device. Hence considerable savings in hardware and power are achieved by designing this single multiplexed demodulator. Once the channel information of this device is processed, it will be output at a high speed of R time units. This high speed output will take NR time units for processing N channels. This is equivalent to the bank of demodulators which also take NR time units to process N channels. This is because of the slower processing time of NR time units by each of the N demodulators.

4.2 Design Of Demodulation Modules

The demodulation process needs to recover the carrier and data from the input samples corresponding to various channels. Timing recovery is needed for tracking the positions of the samples and is used by the interpolator. Each of the modules for carrier, timing and data recovery is designed to operate according to the equations given in Chapter 3. In this section a multiplexed design for storage, carrier, data and timing recovery modules is presented. A case study of 800 channels is presented for an easier understanding of the design.

4.2.1 Multiplexed Carrier Recovery Module (MCRM)

A Multiplexed Carrier Recovery Module (MCRM) is designed to obtain the carrier phase for each of the channels. Samples of several channels are input serially to this module. At the same time these samples are also input to the Multiplexed RAM Buffer for Samples (MRBS). The MRBS stores these samples to be operated on later by the phase recovered information of the MCRM. The output of the MCRM module will be needed by the Multiplexed Data Recovery Module (MDRM).

4.2.1.1 MCRM Operations

The in-phase and quadrature-phase samples of the channels are input to the Multiplexed Carrier Recovery Module (MCRM) as shown in Figure 4.1. The input samples I_n and Q_n are transformed to a vector according to the equations (3.1) and (3.6) by using a look up table. The pre-programmed values of I_n' and Q_n' corresponding to the transformation of the input samples I_n and Q_n according to equation (3.4) are stored in the Input ROM (IR). The values of I_n' and Q_n' samples are accumulated in the Accumulation RAMs (AR). An Address Generator for Samples (AGS) supplies the addresses to the AR according to the channel numbers for storing the values of I_n' and Q_n'. The samples of various channels are stored in their allotted location of the RAMs. The successive samples over an interval length of 2N (where 2N is the number of samples considered for the accumulation of samples for each channel) are accumulated for each of the channels. The accumulated result needs to be divided by 2N to obtain an average vector (I_n'' and Q_n''). This is achieved by reading all the bits needed to represent I_n' and Q_n' except for the last $\text{Log}_2(2N)$ bits. I_n'' and Q_n'' are used as input to obtain the phase estimate by a $1/m \tan^{-1}$ operation. Sine and cosine values of this phase are obtained for each unique value of phase. I_n'' and Q_n'' are mapped to unique values of sine and cosine stored in the form of a single look up table in Output ROM (OR). The sine and cosine values are then input to a Storage RAM (SR). The SR stores the sine and cosine values corresponding to various channels in their unique locations. Sine and cosine values stored at the first location of the SR are used only by samples of the first channel. The second set of sine and cosine values stored in the second location of the SR are used only by the samples of the second channel. Similarly, the nth set of sine and cosine values are used only by the samples of the nth channel. The operations are carried out such that the nth sine and cosine values operate on the samples of the nth channel of the MRBS. This is followed by (n+1)th values operating on the samples of the (n+1)th channel until the first set of samples of all the channels are processed. Then,

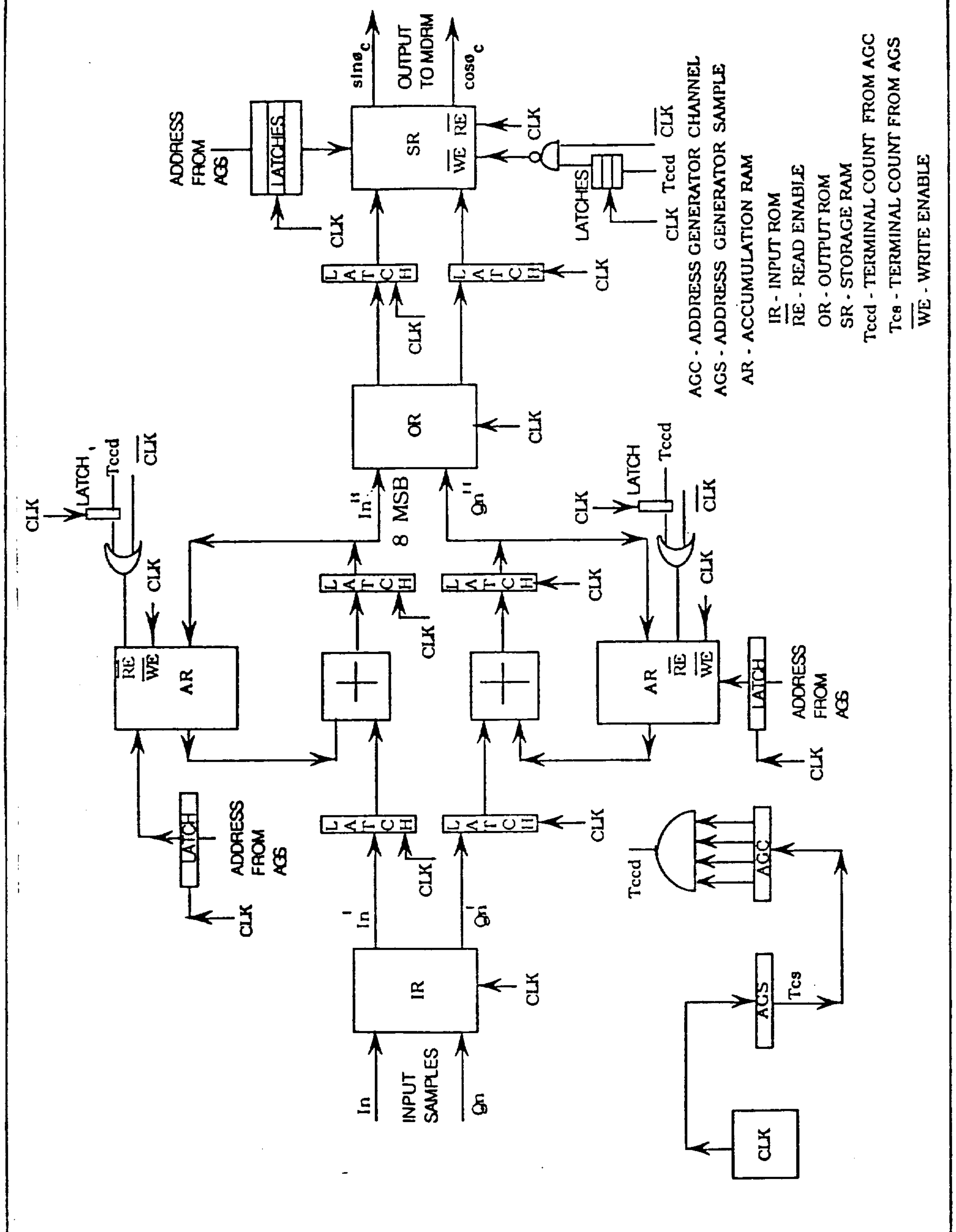

Figure 4.1: Multiplexed Carrier Recovery Module (MCRM)

the new set of samples from the MRBS use the same set of sine-cosine values. This process is repeated for 2N sets, after which time a new set of sine cosines overwrite the old set in the SR.

4.2.1.2 Wordlength Of Quantized Samples

The input samples for the in-phase and the quadrature-phase are assumed to be eight bits wide. Together they form a 16 bit data input to the system. This 16 bit data will act as an address to the ROMs for a look up operation. This 16 bit input data (or address) will then require a 64KX16 ROM for the look up table. Also, the 16-bit data output from the ROM is viewed as a combination of two 8-bit data. The size of the ROM is proportional to the number of locations allowed for the input address. Therefore a 64KX16 IR and OR are necessary for a look up operation. The size of the RAMs physically limit the number of channels that can be processed. A 1KX12 AR will be sufficient to accomodate the accumulated 16 successive 8-bit samples for 1024 channels. A 64KX16 OR is used to store the look up table for the sine and cosine values which are 8 bits each. The OR reads the first 8 most significant bits out of the possible 12 bits at its input. This amounts to a division by 16. A 4-bit counter is used as an Address Generator for Channels (AGC). It counts to 16 which corresponds to 16 sets of samples with each set representing samples of all the channels. A 10 bit Address Generator for Samples (AGS) will be necessary to count up to 1024 unique channel locations. A 12-bit adder is used for the accumulation of 8-bit samples.

4.2.1.3 Control Circuitry And Data Flow In MCRM

Latches are used in the design to synchronize the data flow through various units. All the data latches are negative edge triggered. Therefore the data present at their input are latched at the negative edge of the clock. The AGS is used to appropriately address the desired locations of the RAMs. AGC is mainly used for control of data flow in the module. The terminal count of AGS is used as a clock to the AGC. AGS is a counter for the total number of channels in the system. AGC counts the number of complete sets of samples for all the channels. This is also the estimation period for the extraction of carrier phase (in this design, 2N=16). The control circuitry for the hardware design of MCRM is shown in Figure 4.1.

An input sample at the IR appears at the AR during the second clock cycle because of the delay due to two latches. For an accurate address to appear at the AR after a single clock delay, the address from the AGS has to be delayed by a latch. The AR accumulates samples over an estimation interval of 2N cycles. After this stage, the accumulated values should not be read for one whole cycle. To achieve this purpose, a Read Enable ($\overline{RE}$) is not provided to the AR during this cycle. However, a Write Enable ($\overline{WE}$) is provided which will overwrite the accumulated samples of the previous 2N cycles with a fresh set of samples. The logic requirements for AR are given in Table 4.1. Tccd denotes the terminal count from the AGC. CLK and $\overline{CLK}$ denote the clock and the clock bar used in the system. $\overline{WE}$ and $\overline{RE}$ denote the write enable and read enable of Accumulation RAM (AR).

Table 4.1 Logic for Accumulation RAM

Tccd	CLK	$\overline{CLK}$	$\overline{WE}$	$\overline{RE}$	COMMENTS
0	0	1	0	1	Read disable
0	1	0	1	0	Read enable
1	0	1	0	1	Read disable
1	1	0	1	1	Read disable

The logic needed to obtain a Read Enable ($\overline{RE}$) for the AR is achieved by an OR gate with Tccd and CLB as the inputs. This could have been achieved in a better way by using sequential logic as opposed to directly gating the clock onto combinational logic. The CLK is used for providing $\overline{WE}$ for the AR. In each clock cycle, a read operation is followed by a write operation. The $\overline{RE}$ to AR is not provided when Tccd is 1. The Tccd remains 1 only for one particular clock cycle of AGC. Therefore, the $\overline{RE}$ to AR is not provided during this cycle. Also, a latch delay is provided for the logic to appear at the AR. This is to synchronize with the data available for read operation at the AR. Hence the $\overline{RE}$ is denied for AR for one cycle when all the incoming new samples overwrite the existing accumulated sample values.

In each clock cycle, the SR does a write operation followed by a read operation. A $\overline{WE}$ to SR is provided only when the accurate sine-cosine values are available. Also, $\overline{WE}$ is available only for one particular AGC cycle during which time all the values are stored. Its Write Enable ($\overline{WE}$) is provided only when the output of OR corresponds to the sine-cosine of the accumulated samples. A $\overline{WE}$ is provided to the SR only after 15 AGC cycles. After this time, the Terminal Count of the AGC (Tccd) is output for one whole AGC cycle. A three latch delay is provided for Tccd to appear at $\overline{WE}$ of SR. This corresponds to the delay in the input sample which will take three AGS clock cycles to appear at the input of SR from the time it is input to MCRM. The address to SR is also delayed by three clocks to account for the same delay. This delay is needed to address the appropriate desired locations of the SR. The logic requirement for the SR is given in Table 4.2. The $\overline{WE}$ and $\overline{RE}$ correspond to the Write Enable and Read Enable of the SR.

Table 4.2: Logic for Storage RAM

Tccd	CLK	$\overline{CLK}$	$\overline{RE}$	$\overline{WE}$	COMMENTS
0	0	1	0	1	Write disable
0	1	0	1	1	Write disable
1	0	1	0	0	Write enable
1	1	0	1	1	Write disable

The logic needed to provide the Write Enable to the SR is designed by using Tccd and $\overline{CLK}$ as inputs to a NAND gate. The $\overline{WE}$ will be provided exactly for one particular AGC clock cycle (when Tccd=1) during which the sine-cosine values pertaining to all the channels are available. It is then disabled until the next set of values of sine-cosine are available. At this stage, the Tccd goes high again and the process is repeated.

4.2.2. Multiplexed RAM Buffer for Samples (MRBS)

The Multiplexed RAM Buffer for Samples (MRBS) is designed to store the incoming samples for the duration of an estimation interval. The MCRM operates on the samples obtaining the carrier phase for the channels. Also, at this time the input samples are buffered in the MRBS. The MDRM uses the output of the MCRM along with the stored values of MRBS to recover the digital data.

4.2.2.1 The Operations of MRBS

The Multiplexed RAM Buffer for Samples (MRBS) is designed basically to store the incoming samples. The hardware design for MRBS is shown in Figure 4.2. This design uses a single RAM-Latch combination at each stage to store samples of different channels corresponding to each AGC cycle. The latches are used to latch the incoming samples at the negative edge of the clock. When the clock is positive, the samples are read from the RAM. They are latched at the negative edge of the clock. And, when the clock is negative, the contents of the latch are written into the succeeding RAM. Throughout this single clock cycle, the AGS addresses an unique location of the RAMs. A 1KX8 RAMs and 8 bit latches are needed for this design.

A sample of the first channel is stored only in the first location of the RAMs. Similarly, any sample corresponding to the nth channel is stored only in the nth location of the RAMs. Therefore the number of locations in the RAMs (1K in our case) will be a physical limit on the number of channels that can be processed. In the specific

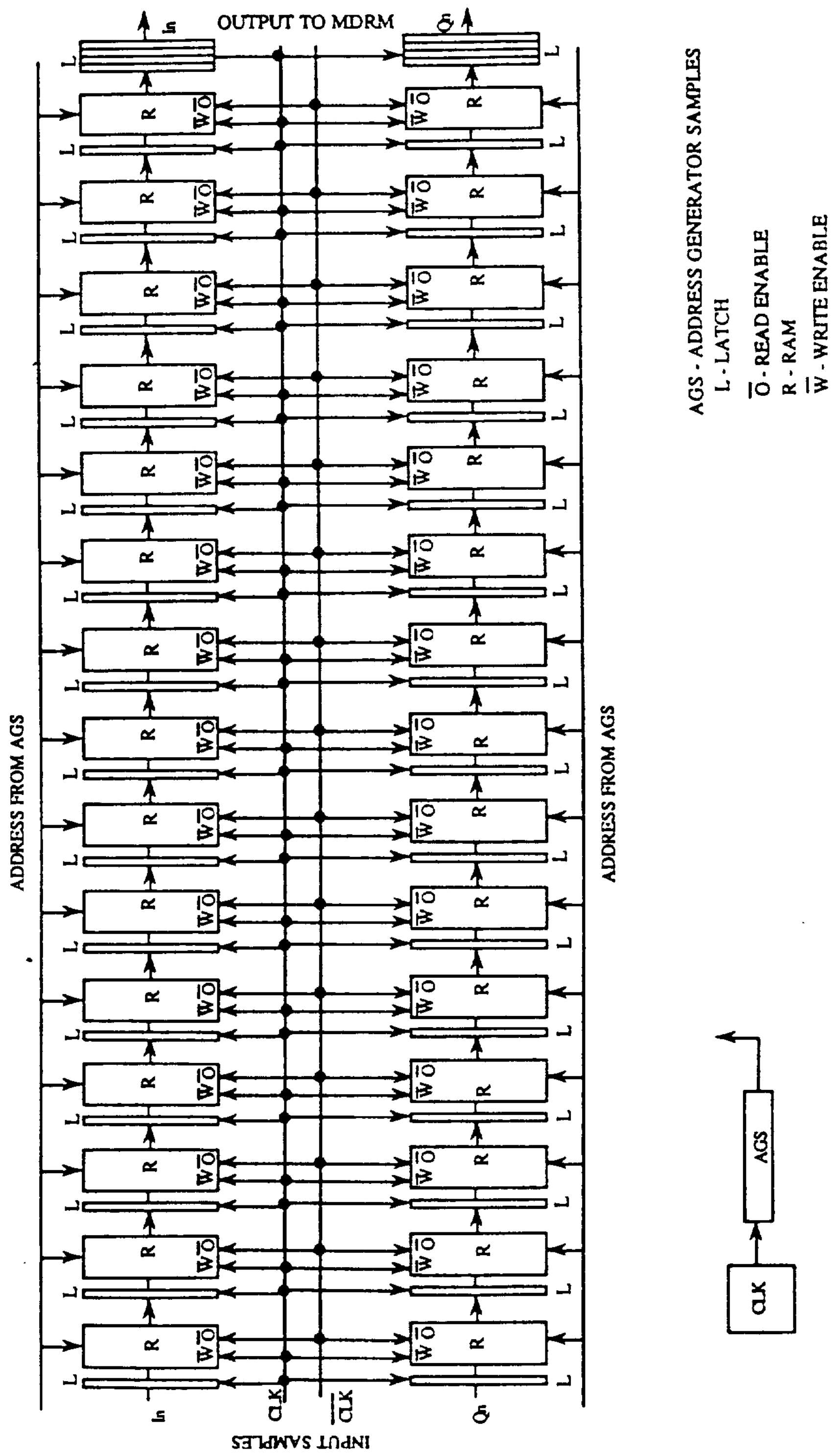

Figure 4.2: Multiplexed RAM Buffer For Samples (MRBS)

design for 800 channels the samples will be stored in 800 different locations of each RAM.

The samples of all the channels are input serially. The first sample is latched and moved to the first location of the first RAM with the AGS addressing the first location. The second sample is moved into the second location addressed by the AGS and so on until the 800th sample is in the 800th location addressed by the AGS. After the first cycle of the AGC, the first set of 800 samples for all the channels is received. These samples are stored in the 800 locations of the first RAM. The next set of operations begins with an incremented AGC and a reset AGS. This corresponds to the input of the second set of samples for all the channels. When the clock is positive, the first sample of the second set is read from the first RAM into the second latch and is latched at the negative edge; at the same time a new sample is latched at the first latch. When the clock is negative, the latched samples are written into their succeeding RAMs. Note again that these operations are done in one clock cycle with the AGS addressing the first location of the RAM. This process continues until the AGS points to the 800th location corresponding to the 800th channel. At this time, the first set of samples is transferred to the second RAM and the just arrived second set is stored in the first RAM. Movement of data and reading of new data continue for 15 AGC cycles. After 15 cycles of AGC, the first set of samples is in the 15th RAM and the 15th set of samples is in the first RAM. When the first sample of the 16th set is input to the MRBS, the very first sample received is output from the 15th RAM to its succeeding latch. This sample is delayed by three clock cycles by using three latches. This is done to provide the MDRM with synchronized samples from MCRM and MRBS.

4.2.3 Multiplexed Data Recovery Module (MDRM)

The Multiplexed Data Recovery Module (MDRM) is designed to extract the digital information. It operates on the samples processed by the MCRM and MRBS. Therefore,

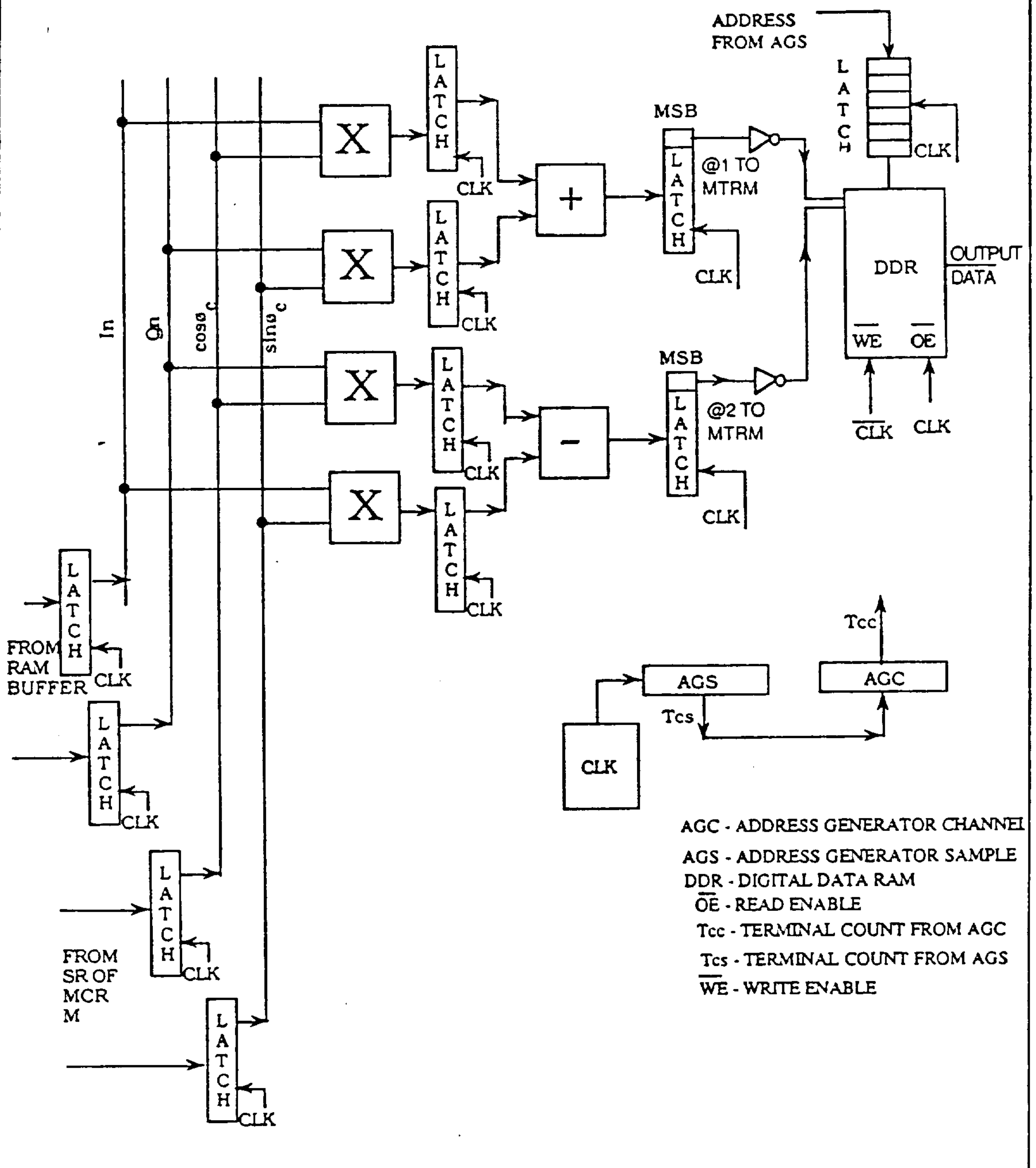

Figure 4.3: Multiplexed Data Recovery Module (MDRM)

it needs the input of these two modules for the start of its operations. This module recovers the digital data which was modulated and transmitted. The hardware design is shown in Figure 4.3.

4.2.3.1 MDRM Operations

The MDRM module utilizes the in-phase and quadrature-phase samples from the MRBS along with the sine and cosine values of the MCRM to extract the digital data for all the channels. At any time four values are input to this module. These values are operated according to equations (3.13) and (3.15). The output is computed and stored in a latch preceeding the Digital Data RAM (DDR) as shown in Figure 4.3. Also, these values are used as an input to the Multiplexed Timing Recovery Module (MTRM). After the necessary computations are performed the results are stored in unique locations of the Digital Data RAM (DDR). These locations are addressed by the Address Generator for Samples (AGS).

The result of the computations stored in the latch preceeding the DDR is either negative or positive. The sign of the values stored in the latches is determined by examining the Most Significant Bit (MSB) of the latch. For a positive value in the latch a '1' is stored in the DDR. A '0' is stored for a negative value. This will be a 2-bit data bus to the DDR corresponding to the input from the in-phase and the quadrature-phase channels. The AGS will provide the addresses to the DDR for storing the digital data of the various channels in their allotted locations.

From the time the samples are input to the MDRM, it will take three clock cycles for them to appear at the DDR. This is because of the three latches used preceeding the DDR. Hence the address from the AGS should be delayed by three clock cycles. However, a six clock delay is provided to address the DDR to account for a further three clock delay for the sample to traverse through the MCRM. This is due to considering the use of a single integrated AGS for all the modules.

4.2.4 Multiplexed Timing Recovery Module (MTRM)

A Multiplexed Timing Recovery Module (MTRM) is designed to extract the timing information needed for tracking the input samples. This timing information is used by the interpolator. Its input is available from the latches used preceeding the DDR of the MDRM. The output of these latches is used as an input to the MTRM.

4.2.4.1 Data Operations in MTRM

The Multiplexed Timing Recovery Module (MTRM) implementation for the multiplexed channels is shown in Figure 4.4. The input samples of all the channels are stored in the three Buffer RAMs (BRs). This RAM-latch sequence operates in a manner similar to the RAM-latch operation described earlier for the MRBS. These samples are input to the BR's at every clock cycle. The input samples stored in the first BR are successively moved from one BR to the next until the three sets of samples of all the channels are acquired. These samples are needed for the computations as in equation (3.10). The address from an integrated AGS is delayed by six latches to account for the delay in receiving the samples at the BRs as shown in Figure 4.4. The data sample is read from the BR and is latched at the negative edge of the clock into the succeeding latch. When the clock goes low, the preceeding latch contents are written into the BR. These operations are performed simultaneously in all six BRs. The data are transferred from one RAM to another using the intermediate latches. Once again, the AGS provides the addresses for storing the input samples in the unique locations allotted in the BRs. The computations are performed with the most recent peak sample in the first BR and the estimated cross-over and peak samples in the successive BR's.

The relevant computations will start once the samples are available in all of the BRs. The timing error in sampling each of the channels as in equation (3.11) is then computed, updated and stored in a Timing RAM (TR). This error is accumulated with the previous error values and is stored in the TR. A correction in timing is available only

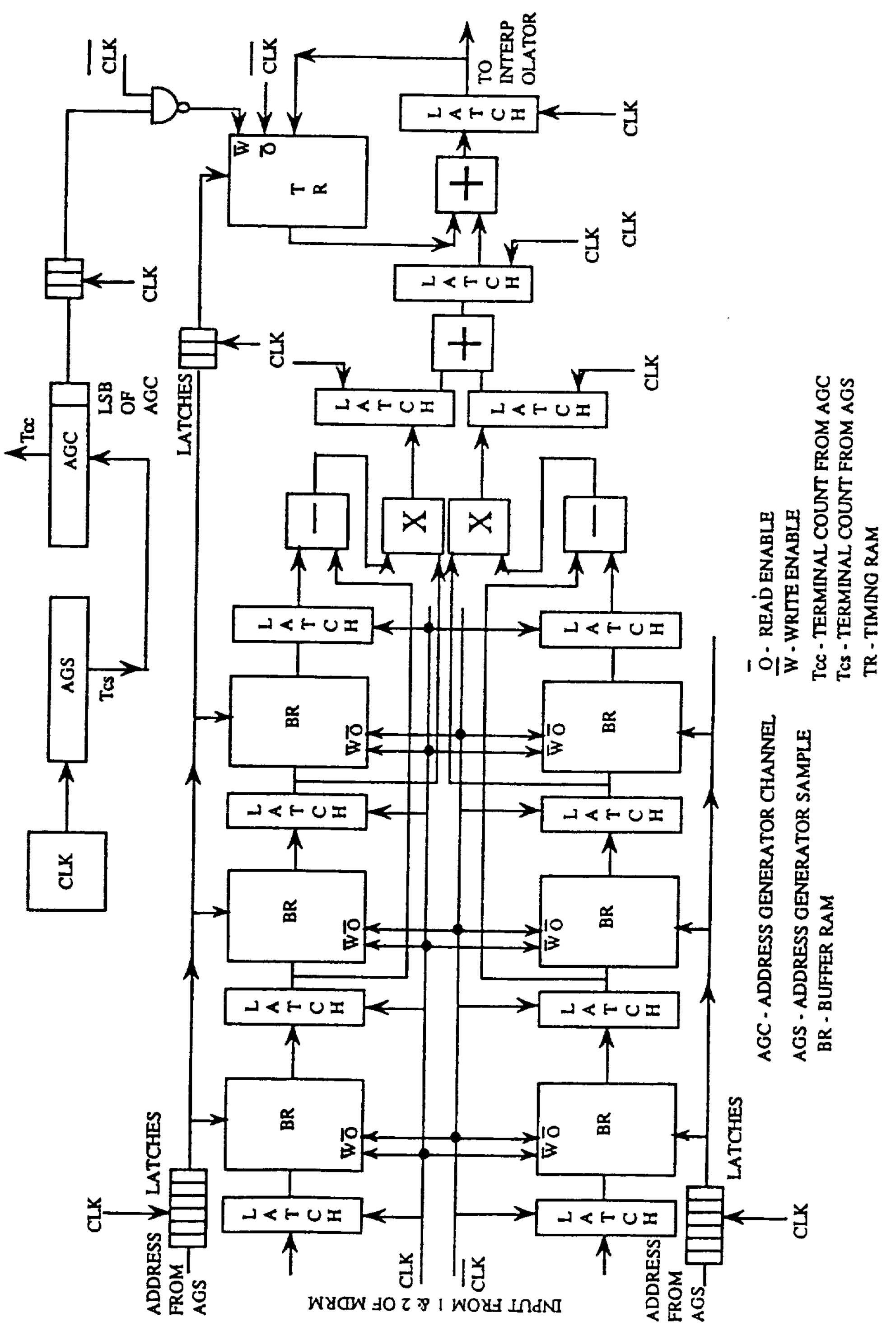

Figure 4.4: Multiplexed Timing Recovery Module (MTRM)

for successive symbols which have a transition. Also, the operations should be performed only at the next configuration with cross-over samples in the middle BR and the peak samples in the extreme BRs. This will be available only at every other AGC cycle. Each BR is a 1KX8 for accomodating 8 bit samples for a maximum of 1024 channel locations. The TR is a 1KX12 to accomodate for an overflow due to the accumulation of samples.

4.2.4.2 Control Circuitry

After the samples are input to the MTRM, it takes three AGC clock cycles and three AGS clock cycles for an input to be available at the Timing RAM (TR). A three latch delay is provided for the address of AGS to appear at the TR. As discussed earlier, the inputs to the TR should be read only at every other AGC cycle. Therefore, the TR is enabled appropriately by a logic using the Least Significant Bit (LSB) of the AGC. The TR needs the Write Enable ($\overline{WE}$) to be provided for one AGC cycle and denied for the next. This can be achieved by utilizing the LSB of AGC which changes from 0-1 and 1-0 every other AGC cycle. The logic for enabling the TR is provided in the Table 4.3. LAC denotes the LSB of AGC. $\overline{RE}$ and $\overline{WE}$ represent the Read Enable and Write Enable of the Timing RAM.

Table 4.3: Logic for Timing RAM

LAC	CLK	$\overline{CLK}$	$\overline{RE}$	$\overline{WE}$	COMMENTS
0	0	1	1	1	Write disable
0	1	0	0	1	Write disable
1	0	1	1	0	Write enable
1	1	0	0	1	Write disable

The logic needed to provide the $\overline{WE}$ for the TR at every other AGC cycle is achieved by a NAND realization of LAC and $\overline{CLK}$. This will provide $\overline{WE}$ of TR for one complete AGC cycle (amounting to 800 AGS cycles) and not provide it for the next AGC cycle. By this process only those operated values corresponding to the samples which are in the desired configuration in the BR's are written into the TR. A delay of three latches is provided for enabling the TR. This is needed to coincide with the data which appears at the TR after three clock cycles.

4.3 Design And Interface Of All The Modules

The combination of the four modules namely MCRM, MRBS, MDRM and MTRM is collectively called a PRODEM. These four modules need to be appropriately interfaced. The addressing scheme, control circuitry and the integration of addressing units for all the modules need special attention. A design for each of the modules with proper interfaces is shown in Figure 4.5.

As noticed from earlier discussions, the addressing for each of the modules is achieved by using a AGS. All the RAMs used in each of the modules need an AGS for addressing the locations pertaining to various channels. By interfacing these modules, a single integrated AGS is desired. Use of single AGS also requires additional latches to address the RAMs in each of the modules. Efforts were directed to use common control circuitry for all the modules. Use of system AGS and AGC also required glue logic to interface all of the modules. Also, buffers are needed to interface the outputs of MCRM and MRBS to the MDRM. This enables the MDRM to operate on the in-phase and quadrature-phase samples from the MRBS with the corresponding sine-cosine values from the MCRM. The MTRM uses the output of only one module (from the latches preceeding the DDR of MDRM) and hence does not need any buffers for its interface.

The design is flexible in terms of the total number of channels that can be processed. A single addressing scheme is used to keep the hardware flexible. By choosing larger AGS and RAMs, the total number of channels can be increased. The

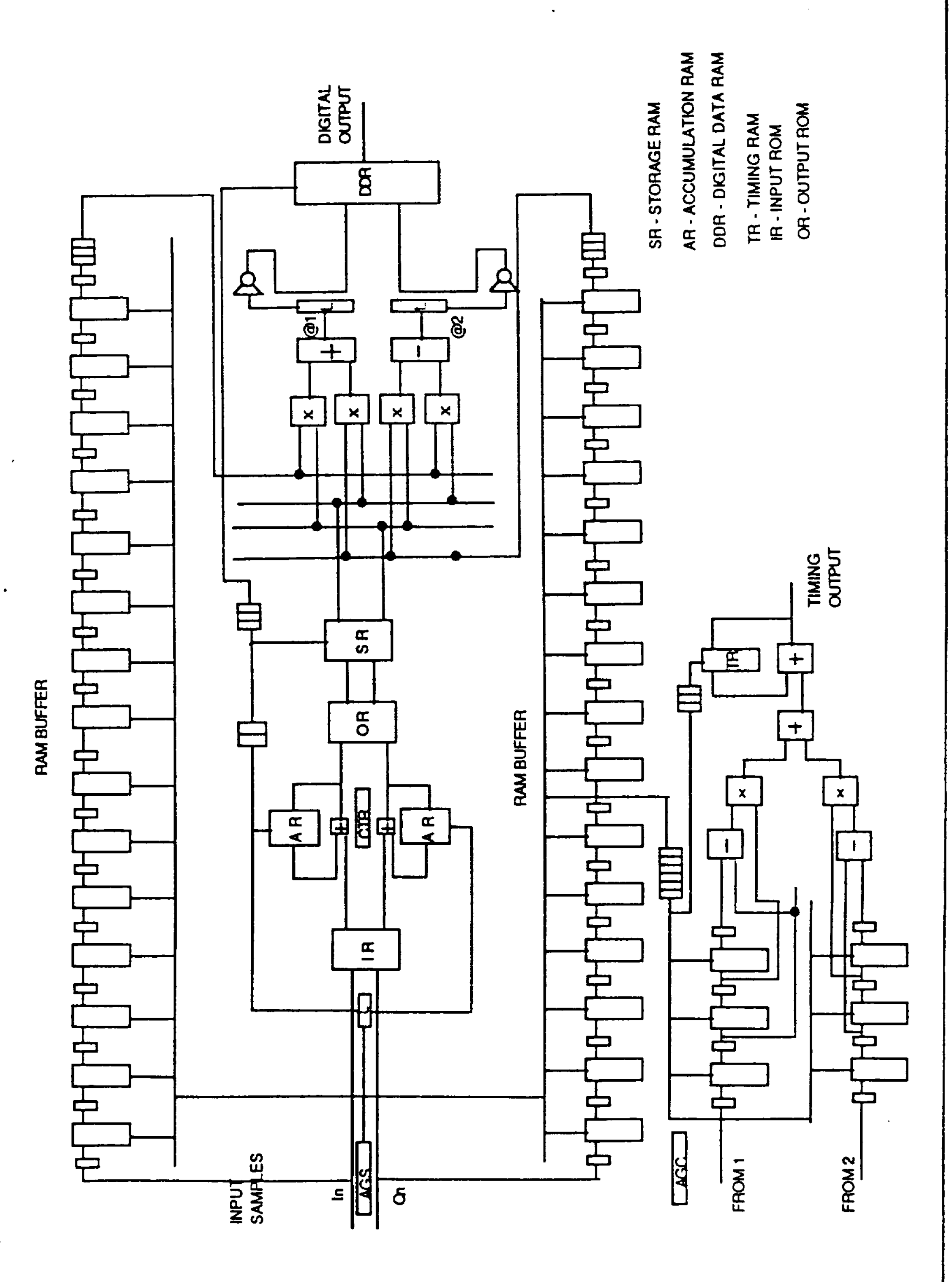

Figure 4.5: PRODEM

upper limit of the channels is restricted by the total number of locations in the RAMs. In this design up to 1K channels can be demodulated as 1K RAMs are selected. However, one should keep in mind that the clock speed of the units also restricts the number of channels that can be processed. By designing the AGS as a variable counter, it is very simple to accomodate a variable number of channels. The maximum count of the AGS will automatically correspond to the maximum number of channels for the same design. An algorithm is developed to process groups of varying bit rate channels and is described in the next section.

4.3.1 Issues in AGS for Reconfigurability

As discussed earlier, the address generator AGS plays a crucial role in addressing the appropriate locations of the RAMs. It is responsible in providing addresses for accurate storage and retrieval of the data. If all the channels had a uniform bit-rate, then the address generator AGS will count up to the maximum number of channels and start the count all over again. In case of groups of varying bit rate channels the algorithm will be different and is described as follows.

Let X and Y be the number of channels in two groups (Y<X). Then the address generator will not access the channels serially (X+Y) in the RAM, but will access them in such a way that the higher bit rate channels locations are accessed a greater number of times than the lower bit rate channels. Assume Y has a higher bit rate, then the algorithm will be as follows:

```
Do once
X channels
    Do (X/Y) times (rounded to an integer)
    Y channels
    end
end
```

AGS that will incorporate this algorithm in the design will be reconfigurable for groups of varying bit rate channels. The user will specify parameters related to the number of channels, number of groups and the bit rate of the channels. A microprocessor can be used to write a control word to control the logic of the address generator. The AGS will count to the number of channels in the case of channels with equal bit rates. If the channels are groups of different bit rate, then the addresses will be generated as described by the algorithm. The AGS counting scheme can be prestored in the form of a hardware look up table or it can be made available under the software control of a microprocessor. The address generator can be configured from the ground according to the user specifications. This is essential if the user needs to vary the number of channels or the groups of channels.

4.3.2 Certain Sytem Parameters

The data received by the PRODEM could relate to the modulated data pertaining to voice, data, high speed FAX or other applications. Based on the input, the bit rate can be classified in three different groups as considered in [1]. The three different cases considered are shown in Table 4.4.

Table 4.4: Several Bit Rate Applications

Bit rate	Number of channels	Total bit rate
1. 64 Kbps	800	51. 2 Mbps
2. 2.048 Mbps	24	51. 2 Mbps
3. 64 Kbps & 2.048 Mbps	400 & 12	51. 2 Mbps

It can be seen that the total bit rate for all the cases is 51.2 Mbps. The PRODEM has to maintain this rate of computation to process the information for all the channels. Some of the parameters involved for processing the 800 channel case are provided in Table 4.5.

Table 4.5: Some System Parameters

Modulation	QPSK
Uplink	SCPC/FDMA
Downlink	TDM
Number of channels	800
Bandwidth of each channel	45 Khz
Bit rate of each channel	64 Kbps
Symbol rate	32 Ksps
Total channel bandwidth	36 Mhz (45Khz X 800)
Time allowed for processing	27.7 ns
Time allowed for demodulation	27.7 ns (based on interpolation)
Overall bit rate required	51.2 Mbps (800 X 64 Kbps)
Word length	8 bits
Output of PRODEM	27.7 ns
Time for processing 800 channels	22.2 μs

The address generator AGS provides the addresses for appropriately storing and accessing the channel information. For 800 channels (64 Kbps each), it addresses 800 different locations of the 1K RAMs corresponding to 800 channels. Similarly, the data for each sample of the 24 channels (2.048 Mbps each) are stored in and accessed from 24 locations of the 1K RAM. In the third case which is a mix of varying bit rate channels, the address generator first stores the data of the 400 channels (64 Kbps each) in 400 locations of the 1K RAM. It then stores the data of the 12 channels (2.048 Mbps each) by counting 400/12 = 33.33 (actually 34) times. This is necessary to process higher bit rate channels.

4.4 Power Requirements

From a literature survey of the currently available memory and logic units [19-21] the power requirements of the MCRM, MBRS, MDRM and MTRM are estimated. The power estimates for the various units in the design are listed in Table 4.6. The components used in each of the modules are listed in Table 4.7. The power requirement of all the modules is listed in Table 4.8.

Table 4.6: Power Rating of the Units Used in the Design

RAMs	200 mW
ROMs	100mW
ADDER/SUB	20 mW
MULTIPLIER	250mW

Table 4.7: Total Number of Units Used in the Design

MODULE	RAMs	ROMs	ADD/SUB	MULT	LATCH
MCRM	3	2	2	-	16
MRBS	30	-	-	-	36
MDRM	1	-	2	4	16
MTRM	7	-	4	2	30

Table 4.8: Total Power Requirements of the Modules

MCRM	840 mW
MRBS	6000 mW

MDRM	1240 mW
MTRM	1980 mW
TOTAL	10.06 W

4.5 Hardware Simulation

The modules of the PRODEM are simulated in high level using C language. The software describes the operation of the hardware units for each of the module. The operation of RAMs, latches, adders, control circuitry, etc., of each of the modules will be described by the software. The software description pertains to the data flow in all the hardware units with respect to the system clock.

The software programs are written to describe the operations performed by the MCRM, MRBS, MTRM and MDRM. Also a program is written to describe the hardware interface of MCRM, MRBS and MDRM. Each of these programs uses an input file consisting of random numbers. These numbers are passed through the hardware described by the program. The contents of the hardware units are displayed after each clock cycle. Therefore the location of the data is easily examined for each clock cycle. After a certain number of clock cycles the data are output from the module. The random numbers are passed through the simulated hardware at least for this number of clock cycles. The control circuitry is also incorporated in the simulation program. It accounts for the $\overline{RE}$ and the $\overline{WE}$ of the RAMs. The data are therefore read or written into the RAMs based on whether or not certain control parameters are satisfied. The flow of control signals can be seen in the output display of the units for each of the clock cycle.

The simulation is carried out for eight SCPC/FDMA channels. The estimation period for these channels is assumed to be four samples. The simulation is therefore a scaled down version for 800 SCPC/FDMA channels which had a sixteen sample estimation period.

4.5.1. MCRM Simulation

A program is written to simulate the operations performed by the Multiplexed Carrier Recovery Module (MCRM). The samples are input from a file consisting of random numbers. These numbers are passed through the units of the MCRM. At each clock cycle the contents of the AR, IR, OR and SR are displayed. The AGS increments with every clock cycle. It counts up to eight unique values corresponding to eight channels. The terminal count of the AGS is a clock for the AGC. The AGC count represents the number of complete sets of samples pertaining to all the channels. In the simulation, AGC counts to four unique values corresponding to four sets of samples used in an estimation period. The simulation program, input and the output are listed in Appendix B.

The samples are input to the MCRM at every clock cycle. For each clock cycle various operations are performed to the IR, SR, AR and OR. The operation of these units with respect to each clock cycle is shown in Table 4.9. These operations are incorporated in the simulation program. Initially, the first set of data is available at the output of the IR after the first clock cycle. The in-phase and quadrature-phase samples in the IR are represented as IRI and IRQ, respectively, as shown in the output file. At the second clock cycle, these data are moved to the ARs. The ARs used to store the in-phase and quadrature-phase samples are represented as ARI and ARQ, respectively. The data are stored in the first location of the ARs as shown in the output file. After the second clock cycle, the data are stored in the next successive locations of the AR. Note that the AGS is incremented with each clock cycle and the AGC is incremented with every terminal count of the AGS. After the AGC is incremented the ARs will have the first set of data corresponding to eight channels. During the second AGC cycle, the input data from the IR are accumulated with the data in AR and stored back in the ARs. The sine and cosine parts of the OR are represented as ORS and ORC, respectively. The output data from the OR for each clock cycle are not written into the

SR until certain conditions are met. Data are written into the SR during the fourth AGC cycle (AGC=3 and AGS=4). This is because the sine-cosine values of the accumulated values are available only at this instant (as seen in the output file in Appendix B). This set of values is available for one AGC cycle during which time all the sine-cosine values for all the channels are received. SRS and SRC are used to represent the sine and cosine values of the SR.

Table 4.9: Operations of MCRM

Positive clock:	READ AR (only when Tccd = 0)
	READ IR
	READ OR
	WRITE SR (only when Tccd = 1)
Positive to negative transition:	LATCH data
Negative clock:	WRITE AR
	READ SR

Note that the instant when the first output is available from the MCRM, this happens when AGC=3 and AGS=4. Thereafter an output is available at every clock cycle. Also note that the same values of SR are output for four AGC cycles. Hence the SR values will not be updated until the AGC is reset and again reads the values that allow such an operation (AGC=3 and AGS=4). At this instant fresh values overwrite the existing values in the SR. Also, the AR will store accumulated values until AGC=0 and AGS=1. After this time, fresh samples from the IR will overwrite the accumulated samples in the AR as seen in the output listed in Appendix B.

4.5.2 MRBS Simulation

The Multiplexed RAM Buffer for Samples (MRBS) is used to store the incoming samples. A program is written to describe the operations performed by the MRBS. RAMs are used to store samples of various different channels. Latches are used to transfer the data from one RAM to the next. As shown in Table 4.10, data are read from the RAMs when the clock is positive. The write operation is performed when the clock is negative. During the negative edge of the clock the data are latched. The simulation program incorporates these operations for each clock cycle. The simulation program, input and output are listed in Appendix C.

Table 4.10: Operation of MRBS

Positive clock:	READ RAM
Positive to negative transition:	LATCH DATA
Negative clock:	WRITE RAM

The samples are input to the first latch and first RAM of the MRBS. The first latch and RAM are represented in the output file as LA1 and RA1, respectively. At the end of the first clock cycle, the data are stored in LA1 and the first location of RA1 as shown in the output program listed in Appendix C. After this clock cycle the samples are stored in the successive locations of the RA1. The AGS is incremented with every clock cycle and is used to address the storage locations of the RAMs. Also the AGC is incremented by the terminal count of AGS. After one AGC cycle, the first set of samples of the eight channels is available in RA1. After this time, the next set of samples is input to the RA1 and its previous set is transferred to the second RAM (RA2) by using the second latch (LA2). This indicates a new AGC cycle. By the end of this cycle (AGC=1 and AGS=8), the first set of samples is in RA2 and the second set of samples is in RA1. This process is repeated until the samples are moved through the third RAM (RA3) and the fourth RAM (RA4) by using the third latch (LA3) and the fourth latch (LA4), respectively. As seen in the output file, the output of the module is available when AGS=1 and AGC=3. The data from LA4 are available as an output after this clock cycle.

4.5.3 MDRM Simulation

The Multiplexed Data Recovery Module (MDRM) needs inputs from the MCRM and the MRBS modules. Therefore the data output from these two modules are synchronized with input to this module. This is achieved by observing the data output from the two modules and designing latches to hold the data output from a faster module. Since the data from the MRBS is output three clocks earlier than that of the MCRM (see programs listed in Appendices B and C for MCRM and MRBS), three additional latches are used to interface the MRBS to the MDRM.

The AGS and the AGC operate for eight channels as described earlier. MU1, MU2, MU3 and MU4 are used in the output file to represent the data output from the multipliers of Figure 4.3. After the first clock cycle the data are available at the output of the multipliers. After the second clock cycle it is available at the output of the adder represented as ADD and the subtractor represented as SUB. The data are input to the latches LA1 and LA2 during the third clock cycle. DDI and DDQ are used to represent the in-phase and quadrature-phase storage locations of the DDR. The data values are input to the DDR during the fourth clock cycle. It takes three clock cycles for the data to appear at the input of the DDR. This is due to the three clock delays corresponding to the three latches used preceeding the DDR. The operation of the units of this module is shown in Table 4.11. The simulation program describes these operations. As can be seen in the output file listed in Appendix D, an output is available from this module when AGS=4 and AGC=0. After this instant, a sample is output from the DDR of this module at every clock cycle.

Table 4.11: Operations of MDRM

Positive clock:	WRITE DDR
Positive to negative transition:	LATCH data
Negative clock:	READ DDR

4.5.4 MTRM Simulation

The MTRM needs an input from the MDRM module for the start of its operations. The output from the latches preceding the DDR are used as an input to the MTRM. The data are moved from one BR to the next similar to the data movement in the RAMs of the MRBS module. It takes three AGC cycles for the data to traverse through the three BRs. The operation of the hardware units is shown in Table 4.11 and the simulation program listed in Appendix E incorporates these operations for each clock cycle.

Table 4.12: Operations of MTRM

Positive clock:	READ BR
	READ TR
Positive to negative transition:	LATCH data
Negative clock:	WRITE BR
	WRITE TR (only when LAC=1)

LQ1 and RQ1 used in the output file represent the first latch and the first BR of the quadrature channel. LI3 and RI3 represent the third latch and the third BR of the in-phase channel. Since a analysis of data flow for the MRBS module is already verified all the latches and RAMs of this part of the MTRM are not shown in the output listed in Appendix E. From an earlier discussion the data are available at the input of LI4 and LQ4 which represent the fourth latch of the in-phase and quadrature-phase when AGS=1 and AGC=3. ISB and QSB are the outputs of the subtractors for the in-phase and quadrature-phase respectively. IML and QML are the outputs of multipliers for the in-phase and quadrature-phase respectively. After the data are available at the output of the LI4 and LQ4 latches, it takes two clock cycles for it to appear at the input of TR. ADD and AER represent the data at the adders used preceeding the TR. The data are input to the TR when AGS=4 and AGC=3. After this instant, data are continuously input to different locations of TR corresponding to various channels till AGS=4 and AGC=0. For the next AGC cycle no samples are written into the TR. Following this the samples are once again written into the TR. The simulation program, input and the output files are listed in Appendix E.

4.5.5 System Simulation

The data flow for the modules is examined for the operation of the interface of the modules. The address generator AGS is integrated for all the modules in this simulation. Since the operation of the internal units of the MCRM, MDRM, MRBS and MTRM are already known, this program incorporates the operation of these modules and describes the hardware for the interface. The program is listed in Appendix F. It accounts for the interface of MCRM and MRBS for the MDRM module.

The nomenclature used in the output file is similar to the one used for the individual modules. In addition, SLC and SLS are the cosine and sine values in the latches at the input of the MDRM. BI4 and BQ4 are the interface latches used at the output of the MRBS. DIL and DQL are the latches of MDRM for storage of the digital data in the DDR. RID and RQD are the digital data present in the DDR which correspond to the in-phase and quadrature-phase respectively.

It is seen that when AGC=3 and AGS=1, the data are available at the output of the MRBS. It goes through three further latches after which it is available as an input to the MDRM. This is necessary because the MCRM has an output only when AGC=3 and AGS=4. At this instant the sine-cosine values are available from the SR of the MCRM module. It takes three more clock cycles for the data to be input to the DDR. Therefore the throughput of the system is easily observed by having a sample output from the

DDR for every clock cycle after the initial input to the DDR. When the outputs are available the counters are reset (AGS=1 and AGC=0). Incidentally, this is also the start of input of new set of samples.

Chapter V

A HYPERCUBE IMPLEMENTATION FOR SCPC/FDMA VOICE CHANNELS

In this chapter digital demodulation of Single Channel Per Carrier/ Frequency Division Multiple Access (SCPC/FDMA) voice channels is implemented using a hypercube. The demodulation algorithms are mapped onto several processors of a binary hypercube. The aim is to provide a mapping [24-31] of these algorithms on a hypercube architecture capable of recovering messages for several SCPC/FDMA voice channels. Two models are developed and their simulations are performed. The speedup results are also provided.

The discussion in this chapter is limited to exploring another scheme for high speed demodulation of SCPC/FDMA channels. Since the hardware and power requirements of an On Board Processing application are very crucial, this implementation may not suitable for such an application. However, it could possibly be used for certain other terrestrial applications. Also, the discussion is focussed on mapping of these algorithms, their assignment and the performance estimation in using such hypercubes.

A hypercube is a parallel computer with a fixed pattern of interconnection among its processors. A three dimensional binary hypercube has processors placed at the vertices of a cube with the edges being the interconnection between them as shown in Figure 5.1. It has 2^3 processors. A generalized hypercube of n dimensions has 2^n processors. Also, it has n disjoint paths between any pair of nodes. Each node of the hypercube has a processor and a memory unit and is called a Processor Element Module (PEM). To represent the eight processors three bits are needed. The processors are connected such that its neighbors differ only in a single bit position out of the three bits used to represent them. The versatility of the hypercube [32-35] comes from the various

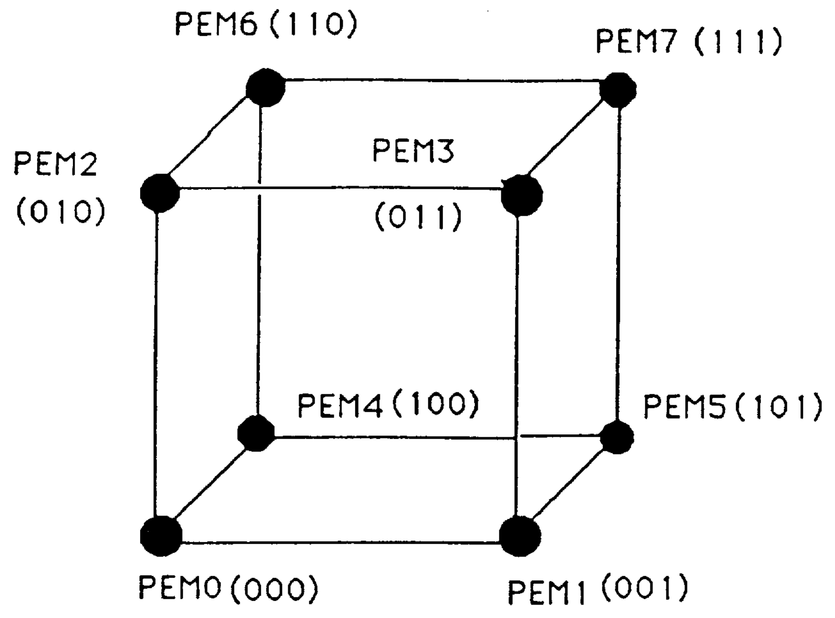

Figure 5.1: PEM Configurations For A Hypercube

dynamically configurable topologies and from a simple algorithm for node to node communications.

5.1 Mapping Strategy

5.1.1 Algorithm Division

In order to understand the principle behind the division of an algorithm, let us analyze a simple algorithm and its task units as shown in Figure 5.2. Let the three processes A, B & C be the tasks being pipelined with an assumption that B takes the maximum computational time. Therefore this task will govern the operating speed of the algorithm for a parallel computer. This computational time can be reduced by examining B and further splitting it into parallel parts if possible. Otherwise, it can be split into two further tasks in a pipeline. This process can be repeated till a lot of smaller tasks are obtained. But usually in an extreme case if many tasks are created, a lot of synchronizations and data transfers degrade the performance which we are trying to achieve. Note that the slowest module in the diagram will govern the eventual operation of the system. This principle is used to implement the demodulation algorithm.

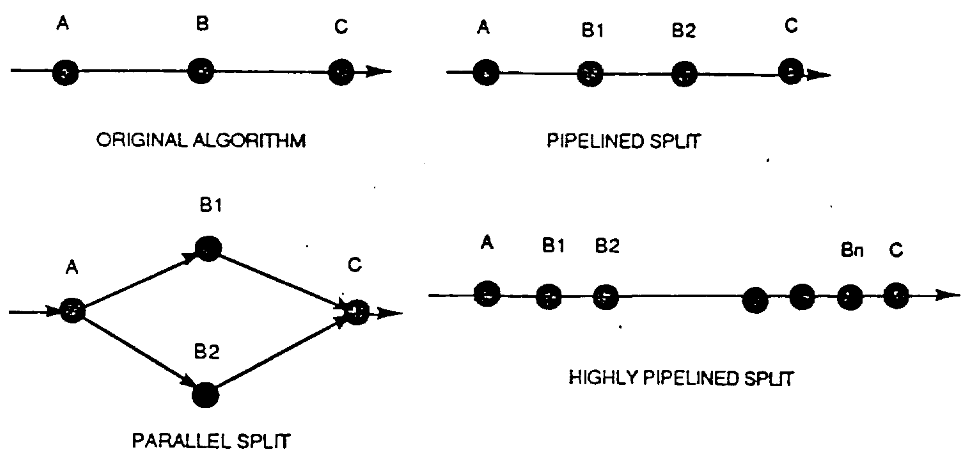

Figure 5.2: Parallel-Pipeline Splits Of An Algorithm

The demodulation algorithm is already studied in great depth in chapter 3. All the operations performed for the demodulation are listed in Table 5.1. The operations C-1 to C-5 pertain to the carrier recovery algorithm. T-1 to T-4 and D-1 to D-4 pertain to the timing and data recovery algorithms respectively. In a sequential implementation, all these operations will be assigned to a single processor. If several processors are available, portions of the algorithm will be assigned to each processor. This is necessary if a single processor cannot process the complete algorithm. From the list of computations needed for the demodulation, a simple flowgraph is drawn as shown in Figure 5.3. This algorithm is divided into four parts assuming four processors are used for all the operations of each channel. The other four processors of the binary hypercube can be used to operate on another channel in parallel. Hence, a single hypercube can be used to process two channels in parallel. A mapping of these models is performed in the next sections.

Table 5.1: List of operations for demodulation

$In + jQn$	$= Rn.\ e^{j\phi n}$.	(C-1)
$In' + jQn'$	$= (Rn)^4\ .\ \exp(j4\phi n)$	(C-2)
In''	$= 1/2N+1 \quad \Sigma\, In' \quad n=-N \text{ to } n=N$	(C-3)
Qn''	$= 1/2N+1 \quad \Sigma\, Qn' \quad n=-N \text{ to } n=N.$	(C-4)
ϕc	$= 1/m \tan^{-1} (Qn''/In'')$.	(C-5)
An	$= In.\ \cos\phi c + Qn.\ \sin\phi c$	(D-1)
Bn	$= Qn.\ \cos\phi c - In.\ \sin\phi c$	(D-2)
An	$= 1$ if $An < 0$	(D-3)
	$= 0$ otherwise	
Bn	$= 1$ if $Bn < 0$	(D-4)
	$= 0$ otherwise	
Ti	$= [A\,2n-1 \quad \{A2n - I2n-2\}]$	(T-1)
Tq	$= [B\,2n-1 \quad \{B2n - B2n-2\}]$	(T-2)
Un	$= Tin + Tqn$	(T-3)
Wn	$= Wn-1 + Un$	(T-4)

5.1.2 Model-I

A model is developed for the algorithm as shown in Figure 5.3. Four tasks are created for this algorithm. Each task is then assigned to a processor. The assignment is shown in Table 5.2. The configuration of the PEMs for such an assignment is shown in Figure 5.4. The numbers denote the PEMs of the binary hypercube. In this model four processors are used for operating on a single channel. The other four processors are used for operations on a second channel.

The mapping procedure is now discussed in detail. Let us assume that the operations C-1, C-2, C-3, C-4 are assigned to PEM0. This processor can communicate directly to PEM1, PEM2 or PEM4. If PEM1 is chosen to process C-5, D-1, D-2, its output can communicate directly only with PEM3 and PEM5 which are not assigned as yet.

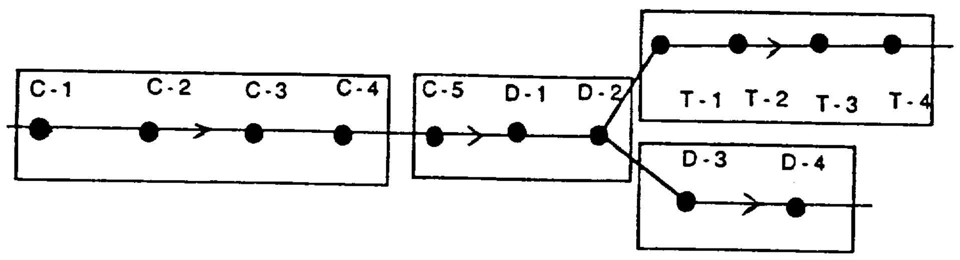

Figure 5.3: Mapping Of Model-I

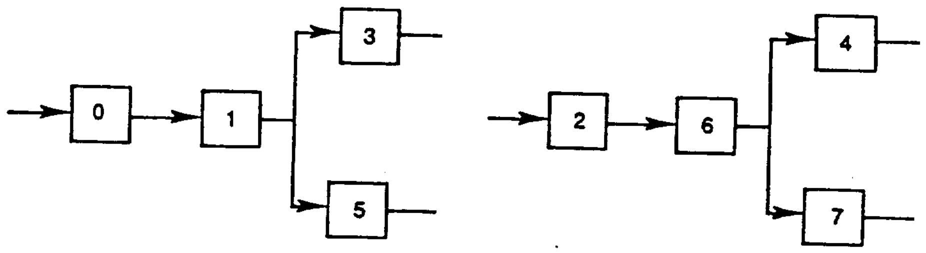

Figure 5.4: Hypercube Configuration For Model-I

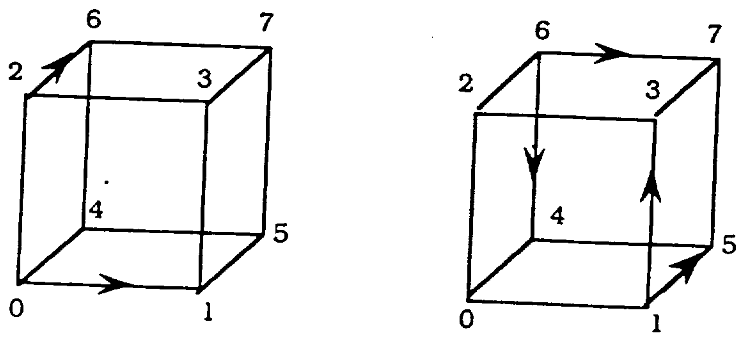

Figure 5.5: Data Flow For Model-I

This is because of the configuration of the processors of the hypercube (see Figure 5.1). Therefore, the processors PEM3 and PEM5 are designated T-1, T-2, T-3, T-4 and D-3, D-4 respectively. The other four processors of the hypercube are utilized for identical set of tasks of a second channel. Let PEM2 be the processor designated to operate on the operations C-1, C-2, C-3, C-4 processes. Now the option is limited to choosing PEM6 for the succeeding operations on C-5, D-1, D-2. This leaves PEM4 and PEM7 to perform the operations corresponding to T-1, T-2, T-3, T-4 and D-3, D-4 respectively. Note that we are striving to obtain those processor pairs which are directly connected to each other for data transfer. This will save time and avoid additional hops from one processor to another through an intermediate processor. Also, note that some of the additional available interconnections are not utilized for this application. The hypercube as assigned for this model, can operate simultaneously on 2 SCPC/FDMA channels in parallel. The direction of transfer of data for these two channels is shown in Figure 5.5. After the first task is computed by PEM0 and PEM2, their outputs are transferred to PEM6 and PEM1. In the second stage, PEM6 in turn computes its assigned task and broadcasts the data to PEM4 and PEM7. Similarly PEM1 broadcasts data to PEM3 and PEM5 after completing operations of its assigned task.

Table 5.2: Task assignment for Model-I

TASK NUMBER	PEM NUMBER	OPERATIONS
Task 1	0, 2	C-1, C-2, C-3, C-4
Task 2	1, 6	C-5, D-1, D-2
Task 3	3, 4	T-1, T-2, T-3, T-4
Task 4	5, 7	D-3, D-4

5.1.3 Model II

A second model is also developed for mapping the algorithm. In this model the algorithm is assigned differently as compared to the earlier model. However, as before four processors are used for processing four tasks of each channel. The tasks are assigned to the processors as shown in Figure 5.6. The mapping procedure is similar to the one described for Model-I. PEM3 and PEM5 are assigned the operations of C-1, C-2, C-3 and C-1, C-2, C-4 respectively. Since these two processing elements are connected directly only to PEM1 and PEM7, one of them is chosen to process the subsequent operations C-5, D-1, D-2, D-3, D-4 of the third task. Let us assume that PEM7 is chosen for achieving this process. It is in turn connected to the only other unassigned processor which is PEM6. Therefore the fourth task is assigned to PEM6 with the operations T-1, T-2, T-3, T-4. Therefore this model uses PEM3, PEM5, PEM6 and PEM7 for processing the operations of a single channel. Note that this model has duplicate operations in the first two tasks. Although this assignment appears to be redundant, as will be seen later, it was aimed at achieving a better load distribution among the processors.

Similarly, the other four processors are assigned the operations of a second channel. PEM4 and PEM1 are assigned the operations of the tasks with the operations C-1, C-2, C-3 and C-1, C-2, C-4 respectively. The only processor that is connected to these two processors and which is unassigned is PEM0. It is used to process the operations of the third task which are C-5, D-1, D-2, D-3, D-4. PEM2 is used to process the information output from PEM0. It is therefore assigned the operations T-1, T-2, T-3, T-4 of the fourth task. The list of operations and their mapping is shown in Table 5.3.

The data flow for this model will be with PEM3 and PEM5 operating on the first two tasks of the first channel. Simulataneously PEM4 and PEM1 operate similarly for the second channel as shown in Figure 5.7. In the second stage the PEM7 and PEM0 operate on their tasks and transfer the data to PEM6 and PEM2 respectively.

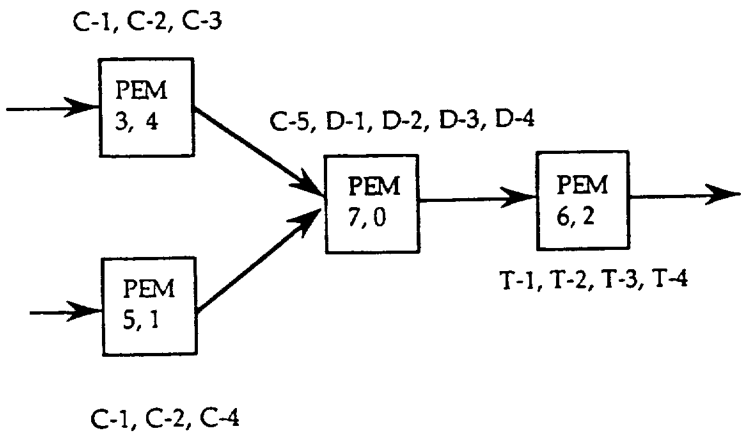

Figure 5.6: Mapping Of Model-II

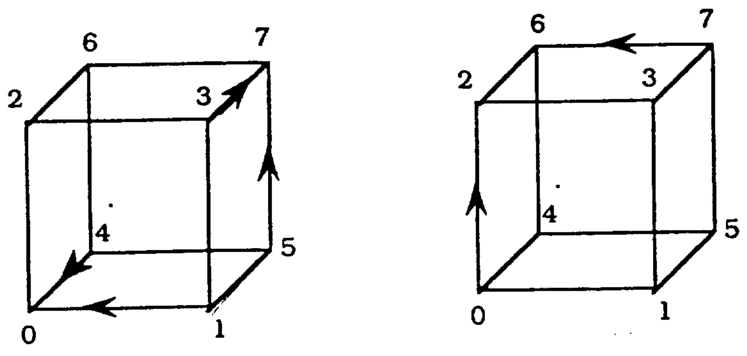

Figure 5.7: Data Flow For Model-II

Table 5.3: Task assignment for Model-II

TASK NUMBER	PEM NUMBER	OPERATIONS
Task 1	3, 4	C-1, C-2, C-3
Task 2	5, 1	C-1, C-2, C-4
Task 3	7, 0	C-5, D-1, D-2, D-3, D-4
Task 4	6, 2	T-1, T-2, T-3, T-4

5.2 Simulation Of Models

The channels are simulated in Ada programming language. Ada is chosen for this simulation as it is convenient to implement certain features of a multiprocessor such as a hypercube [36-41]. The important features of a multiprocessor are the interprocessor communication and synchronization techniques. The interprocessor communication deals with the actual transfer of data from one processor to its neighbour. The synchronization technique deals with making the processors operate at a given iteration level. This amounts to holding the processor that completes its assigned task earlier than its neighbouring processors. Ada supports the usage of these features which are incoporated in the predefined constructs called tasks. Hence it is chosen for the simulation of our hypercube.

The program for these models is developed for 32 channels as it was convenient to implement. The estimation period for these channels is sixteen samples. Therefore at any given time, a PEM processes its assigned task's operations for sixteen times. The flowchart of the models used in the simulation is shown in Figure 5.8. Each task is assigned the operations as discussed earlier. The operations of each task are repeated for sixteen times corresponding to the sixteen sets of input data. The tasks are allowed to communicate only as shown in the Figure 5.8. The simulation programs for Model-I and Model-II are listed in Appendix G and Appendix H respectively. In these programs.

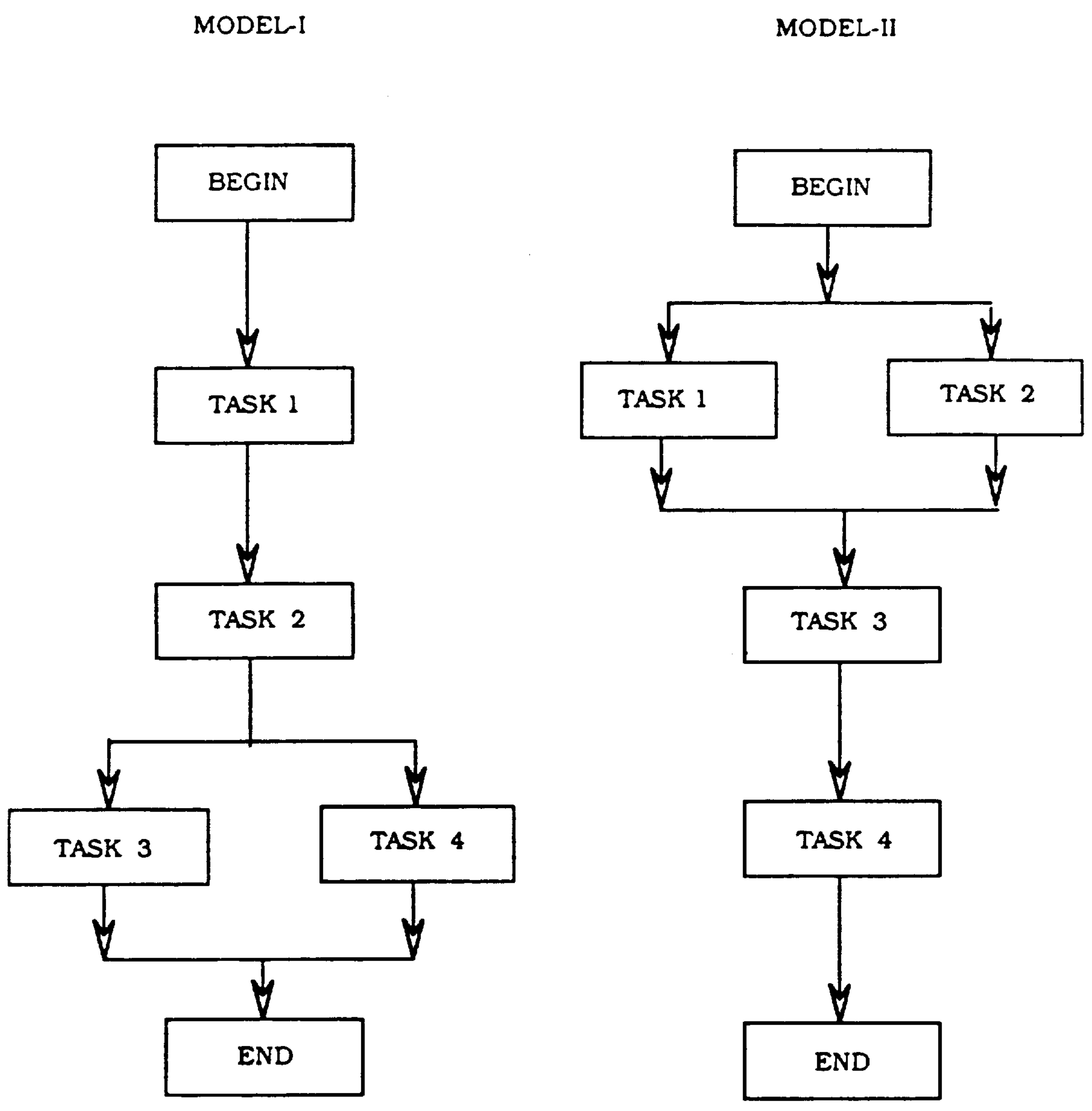

Figure 5.8: Simulation Flowchart For Model-I And Model-II

32 channels are simulated. However, PEMs of each channel operate independently and irrespective of the operations performed by other PEMs operating on other channels. Therefore the result of the simulation can be linearly extrapolated for any number of channels.

The number of floating point operations for each model is noted and is listed in Table 5.4. A communication penalty of 10 floating point operations is assumed as an overhead. The speedup is estimated based on the task which has the most number of floating point operations. This relates to an earlier discussion on the slowest computing processor governing the speed of the multiprocessor system. The speedup is compared to that of a single processor. It is observed that for our models, X hypercubes can process 2X channels. Also, X hypercubes (or 8X processors) can be used for achieving a speedup of 8X times over a single processor. However, in a practical situation, due to the unequal distribution of load among the processors and due to the communication penalty, a degradation in performance is expected. The achieved speedup for Model-I is shown in Figure 5.9. Since Model-II has a better distribution of its operations among the processors, it shows an improvement in its performance as shown in Figure 5.10.

Table 5.4: Floating point operations for Model-I and Model-II

MODEL-I (UNBALANCED)		MODEL-II (BALANCED)	
C-1, C-2, C-3, C-4	10X16 =160	C-1, C-2, C-3	8X16 =128
C-5, D-1, D-2	6X16+4=100	C-1, C-2, C-4	8X16 =128
D-3, D-4	2X16 =32	C-5, D-1, D-2, D-3, D-4	8X16+4=132
T-1, T-2, T-3, T-4	5X16 =80	T-1, T-2, T-3, T-4	5X16 =80

From earlier discussions in chapter 4, the data of all 800 channels is input every 22.2μs. Each task does its operations sixteen times corresponding to the sixteen samples of an estimation period. Therefore, the time available for completing each task is less than 355.2μs (22.2μsX16). This makes sure that the processing time is faster than

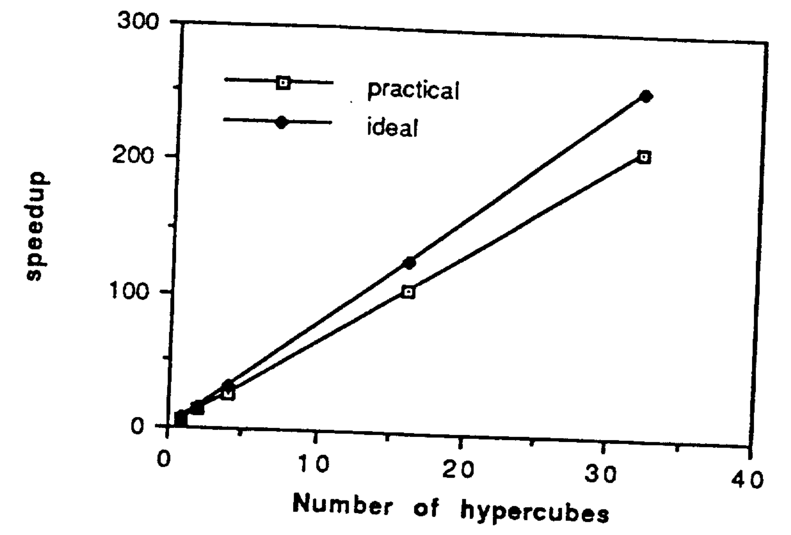

Figure 5.9: Performance Of Model-I

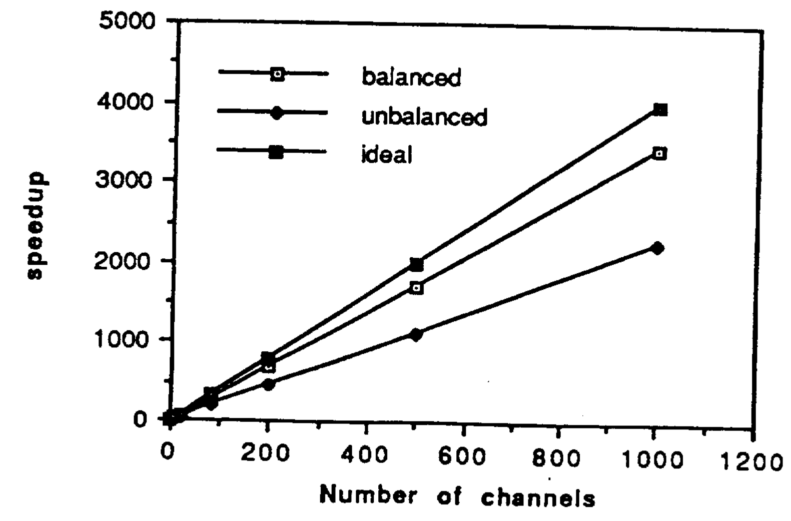

Figure 5.10: Performance Enhancement With Load Balancing

the input rate. As the slowest task has more number of floating point operations than the other tasks, an estimate of the time taken for its floating point operations is sufficient to determine the performance of the system. By knowing the number of floating point operations that need to be performed in a given time, the Mega Floating Point Operations Per Second (MFLOPS) rating of the hypercube can be predicted. Therefore each processor of the hypercube has to sustain atleast [(160+10)/355.2 = 0.48] MFLOPS for the operation of the Model-I. Ten floating point operations are assumed as an overhead for the communication of data between the processors. In case of Model-II, each processor needs to sustain lesser [(128+10)/355.2 = 0.39] MFLOPS. Therefore, the hypercube chosen for such an application is required to sustain (0.48X8 = 3.84) MFLOPS and (0.39X8= 3.12) MFLOPS for Model-I and Model-II respectively.

CHAPTER VI

CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this research project, an efficient demodulator architecture is developed for a large number of low data rate small earth station users in an SCPC/FDMA system. The development of the architecture is based on the parallel-pipelined design approach. This principle aims at mapping the algorithm in such a way that the independent sections are assigned to the parallel units and the dependent sections to the pipelined units. The speed of the computational units is governed by the clock rate of the module, hence high speed computational units are used. The speed of the demodulator depends on all the modules in operation. The input samples of the various channels are obtained from a pipelined FFT. These serially input samples support the multiplexed demodulator architecture as opposed to the bank of parallel demodulators. This device is capable of processing a large number of channels which may be variable in both number and bit rates.

Also, a multiprocessor approach is provided with an emphasis on a three dimensional binary hypercube. The development of this architecture is also based on the parallel processing approach. Two models are created for mapping the demodulation algorithms for such a scheme. The creation of an appropriate model has a bearing on the performance of the system. It is found that an improvement is achieved in providing load balanced models.

6.2 Future Research

Some of the future research could be directed in the following areas:

- The design of the hardware interface between the output of the transmultiplexer and the input to the PRODEM will be required. This will pertain to the design of an interpolator. The storage and the retrieval of the data for multiple channels for this device should also be investigated. The number, group size and the bit rate of the channels should be programmable. This is critical as time multiplexing is inversely related to the speed of the designed hardware.

-In the hardware design approach, the elimination of MRBS will improve the power and hardware requirements to a large extent. Future work should look at possibilities of eliminating the need for MRBS.

- The proposed design is for a QPSK modulation scheme. The design could be generalized to accomodate other bandwidth efficient modulation schemes with multiple data rates. To begin with, Offset-QPSK and 8-PSK could be considered.

-The design in this research maximizes the throughput rate for a given application. This is achieved by using parallel and pipeline techniques and may require several custom-VLSI chips. For low bit rate applications (1 to 10 Mbps) it will be desirable to design a single chip demodulator rather than distribute its functions over several chips. It should be recognized that a transmultiplexer is followed by a demodulator in our application (On-Board Processing) but the demodulator itself could also be used in other space and ground locations. In this regard some of the work done related to hypercubes and other multiprocessors could also be investigated.

Appendix A

SIMULATION OF THE DEMODULATION ALGORITHMS IN A QPSK MODEM

```
      SUBROUTINE phaseest(insig,strobe,outsig)

c     AUTHOR: LINUS P. EUGENE
c             ADVANCED COMMUNICATION RESEARCH LABORATORY
c             UNIVERSITY OF TOLEDO
c             TOLEDO, OHIO

c     Carrier phase estimator

c    This subroutine extracts the carrier phase. The modulation
c    information is killed and the carrier phase obtained.
c

c     ARGUMENTS:
c        insig   - A complex input signal with phase disturbance
c        outsig  - An output signal with the phase estimate of the
c                  carrier

      IMPLICIT NONE
      INCLUDE 'BOSS$SYSTEM:[SYSTEM]BOSSFORTRAN.INC'

      REAL tphase,q,in,newphase,y,quad,realpart,imagpart,r,i,phase,mag
      INTEGER count
      COMPLEX newsig, insig,outsig,phasesig
      LOGICAL *1 strobe

c     1) Rotate out the demodulation

      in=REAL(insig)
      quad=AIMAG(insig)
      phase=ATAN2(quad,in)
      phase=4*phase

c     2) Obtain the phase of the disturbance

      mag=in*in+quad*quad
      newsig=CMPLX(COS(phase),SIN(phase))
      i=REAL(newsig)
      q=AIMAG(newsig)
      newphase=ATAN2(q,i)
      newphase=0.25*newphase
      outsig=CMPLX(COS(newphase),SIN(newphase))

      RETURN
      END
      SUBROUTINE conphase(insig,outsig)

c     LINUS P. EUGENE
c     ADVANCED COMMUNICATION RESEARCH LABORATORY
c     UNIVERSITY OF TOLEDO
c     TOLEDO, OHIO

c     Constant magnitude and transparent phase  unit.

c     This subroutine outputs the same phase of the input signals.
c     It normalizes the magnitude of the input signal

c     ARGUMENTS:
c     insig - input complex signal
c     outsig- output complex signal

      IMPLICIT NONE
      INCLUDE 'BOSS$SYSTEM:[SYSTEM]BOSSFORTRAN.INC'

      REAL inphase,newphase,y,quad,realpart,imagpart
      COMPLEX insig,outsig


      inphase=REAL(insig)
      quad=AIMAG(insig)
      y=(quad/inphase)
      newphase=ATAN(y)
      realpart=COS(newphase)
      imagpart=SIN(newphase)
      outsig=CMPLX(realpart,imagpart)

      RETURN
      END
      SUBROUTINE pedemod(insig, outsig, strobe, outstrobe,
     &                    carrier,phasesig,sym_rate)

c        AUTHOR : MARK J. VANDERAAR
c        REVISED AUTHOR : LINUS P.EUGENE
c        ADVANCED COMMUNICATION RESEARCH LABORATORY
c        UNIVERSITY OF TOLEDO
c        TOLEDO, OHIO

c   QPSK Coherent Demodulator

c        This subroutine demodulates a signal in complex envelope form.
c        It obtains the phase estimate from the carrier phase estimator
c        and effectively uses it with the complex signal.

c   ARGUMENTS :
c      insig     - A complex number that represents the input to
c                  the demodulator
c      outsig    - The demodulated data (complex). Note that it is
c                  not the output symbol, but the vector that
c                  represents the output symbol.
c      strobe    - A logical input impulse train at the rate of
c                  the symbol rate.
c      outstrobe - A logical output impulse train at the rate of
C                  the symbol rate.
c      carrier   - The carrier frequency for simulation purposes,
c                  this will most often be zero.
c      phasesig  - The phase estimate from the carrier phase estimator
c                  module.

      IMPLICIT NONE
      INCLUDE 'BOSS$SYSTEM:[SYSTEM]BOSSFORTRAN.INC'

c     Define the variables
      LOGICAL*1 strobe, outstrobe
      REAL      inphase, quad,inpart,qapart, carrier, phi1, phi2,radian
      REAL      ininp , tphase, inquad, sym_rate
      COMPLEX   insig,   outsig, area, phasesig,   phi

c     Set up the local memory structure
      STRUCTURE / LOCMEM /
          LOGICAL*1 inited,    started, strobeflg
          REAL      phase,     oldin,   oldquad
          COMPLEX   fintegral, integral
      END STRUCTURE
      RECORD /LOCMEM/ MEM

c     If this is the first call to the subroutine,
c     initialize variables.
      IF(.NOT.mem.inited) THEN
         mem.started   = .FALSE.
         mem.strobeflg = .FALSE.
         mem.inited    = .TRUE.
         mem.integral  = (0.0,0.0)
         mem.fintegral = (0.0,0.0)
         mem.oldin     = 0.0
         mem.oldquad   = 0.0
      ENDIF
```

```
c     At the beginning of the first symbol, calculate
c     the phase delay
      IF (strobe) THEN
          IF (.NOT.mem.started) THEN
              mem.phase   = carrier*curtime
              mem.started = .TRUE.
          ENDIF
      ENDIF

c     Perform a step of the demodulation
c     1) Split the signal into the inphase and quadrature components
        inphase       = REAL(insig)
        quad          = AIMAG(insig)

c     2) Calculate the real and imaginary part of the input phase
c        estimate.

        phi1          = REAL(phasesig)
        phi2          = AIMAG(phasesig)

c     3) Multiply the components by the basis function

        inpart        = inphase*phi1+quad*phi2
        qapart        = quad*phi1-inphase*phi2
        outstrobe     = strobe
        outsig        = CMPLX(inpart,qapart)

        RETURN
        END
```

OUTPUT FILE OF MODEM

```
  Time =   0.2028000E+00
#Errors      #Bits       BER
     0         100      0.0000E+00

  Time =   0.4028000E+00
#Errors      #Bits       BER
     0         200      0.0000E+00

  Time =   0.6028000E+00
#Errors      #Bits       BER
     0         300      0.0000E+00

  Time =   0.8028000E+00
#Errors      #Bits       BER
     0         400      0.0000E+00

  Time =   0.1002800E+01
#Errors      #Bits       BER
     0         500      0.0000E+00

  Time =   0.1202800E+01
#Errors      #Bits       BER
     0         600      0.0000E+00

  Time =   0.1402800E+01
#Errors      #Bits       BER
     0         700      0.0000E+00

  Time =   0.1602800E+01
#Errors      #Bits       BER
     0         800      0.0000E+00

  Time =   0.1802800E+01
#Errors      #Bits       BER
     0         900      0.0000E+00

  Time =   0.2002800E+01
#Errors      #Bits       BER
     0        1000      0.0000E+00

  Time =   0.2202800E+01
#Errors      #Bits       BER
     0        1100      0.0000E+00

  Time =   0.2402800E+01
#Errors      #Bits       BER
     0        1200      0.0000E+00

  Time =   0.2602800E+01
#Errors      #Bits       BER
     0        1300      0.0000E+00

  Time =   0.2802800E+01
#Errors      #Bits       BER
     0        1400      0.0000E+00
```

```
   Time =   0.3002800E+01
 #Errors      #Bits        BER
      0        1500     0.0000E+00

   Time =   0.3202800E+01
 #Errors      #Bits        BER
      0        1600     0.0000E+00

   Time =   0.3402800E+01
 #Errors      #Bits        BER
      0        1700     0.0000E+00

   Time =   0.3602800E+01
 #Errors      #Bits        BER
      0        1800     0.0000E+00

   Time =   0.3802800E+01
 #Errors      #Bits        BER
      0        1900     0.0000E+00

   Time =   0.4002800E+01
 #Errors      #Bits        BER
      0        2000     0.0000E+00

   Time =   0.4202800E+01
 #Errors      #Bits        BER
      0        2100     0.0000E+00

   Time =   0.4402800E+01
 #Errors      #Bits        BER
      0        2200     0.0000E+00

   Time =   0.4602800E+01
 #Errors      #Bits        BER
      0        2300     0.0000E+00

   Time =   0.4802800E+01
 #Errors      #Bits        BER
      0        2400     0.0000E+00

   Time =   0.5002800E+01
 #Errors      #Bits        BER
      0        2500     0.0000E+00

   Time =   0.5202800E+01
 #Errors      #Bits        BER
      0        2600     0.0000E+00

   Time =   0.5402800E+01
 #Errors      #Bits        BER
      0        2700     0.0000E+00

   Time =   0.5602800E+01
 #Errors      #Bits        BER
      0        2800     0.0000E+00

   Time =   0.5802800E+01
 #Errors      #Bits        BER
      0        2900     0.0000E+00
   Time =   0.6002800E+01
#Errors       #Bits         BER
     0         3000     0.0000E+00

   Time =   0.6202800E+01
#Errors       #Bits         BER
     0         3100     0.0000E+00

   Time =   0.6402800E+01
#Errors       #Bits         BER
     0         3200     0.0000E+00

   Time =   0.6602800E+01
#Errors       #Bits         BER
     0         3300     0.0000E+00

   Time =   0.6802800E+01
#Errors       #Bits         BER
     0         3400     0.0000E+00

   Time =   0.7002800E+01
#Errors       #Bits         BER
     0         3500     0.0000E+00

   Time =   0.7202800E+01
#Errors       #Bits         BER
     0         3600     0.0000E+00

   Time =   0.7402800E+01
#Errors       #Bits         BER
     0         3700     0.0000E+00

   Time =   0.7602800E+01
#Errors       #Bits         BER
     0         3800     0.0000E+00

   Time =   0.7802800E+01
#Errors       #Bits         BER
     0         3900     0.0000E+00

   Time =   0.8002800E+01
#Errors       #Bits         BER
     0         4000     0.0000E+00

   Time =   0.8202800E+01
#Errors       #Bits         BER
     0         4100     0.0000E+00

   Time =   0.8402800E+01
#Errors       #Bits         BER
     0         4200     0.0000E+00

   Time =   0.8602799E+01
#Errors       #Bits         BER
     0         4300     0.0000E+00

   Time =   0.8802800E+01
#Errors       #Bits         BER
     0         4400     0.0000E+00
```

```
  Time =  0.9002800E+01
#Errors      #Bits        BER
     0        4500     0.0000E+00

  Time =  0.9202800E+01
#Errors      #Bits        BER
     0        4600     0.0000E+00

  Time =  0.9402800E+01
#Errors      #Bits        BER
     0        4700     0.0000E+00

  Time =  0.9602799E+01
#Errors      #Bits        BER
     0        4800     0.0000E+00

  Time =  0.9802800E+01
#Errors      #Bits        BER
     0        4900     0.0000E+00

  Time =  0.1000280E+02
#Errors      #Bits        BER
     0        5000     0.0000E+00

  Time =  0.1020280E+02
#Errors      #Bits        BER
     0        5100     0.0000E+00

  Time =  0.1040280E+02
#Errors      #Bits        BER
     0        5200     0.0000E+00

  Time =  0.1060280E+02
#Errors      #Bits        BER
     0        5300     0.0000E+00

  Time =  0.1080280E+02
#Errors      #Bits        BER
     0        5400     0.0000E+00

  Time =  0.1100280E+02
#Errors      #Bits        BER
     0        5500     0.0000E+00

  Time =  0.1120280E+02
#Errors      #Bits        BER
     0        5600     0.0000E+00

  Time =  0.1140280E+02
#Errors      #Bits        BER
     0        5700     0.0000E+00

  Time =  0.1160280E+02
#Errors      #Bits        BER
     0        5800     0.0000E+00

  Time =  0.1180280E+02
#Errors      #Bits        BER
     0        5900     0.0000E+00
  Time =   0.1200280E+02
#Errors       #Bits        BER
    0          6000     0.0000E+00

  Time =   0.1220280E+02
#Errors       #Bits        BER
    0          6100     0.0000E+00

  Time =   0.1240280E+02
#Errors       #Bits        BER
    0          6200     0.0000E+00

  Time =   0.1260280E+02
#Errors       #Bits        BER
    0          6300     0.0000E+00

  Time =   0.1280280E+02
#Errors       #Bits        BER
    0          6400     0.0000E+00

  Time =   0.1300280E+02
#Errors       #Bits        BER
    0          6500     0.0000E+00

  Time =   0.1320280E+02
#Errors       #Bits        BER
    0          6600     0.0000E+00

  Time =   0.1340280E+02
#Errors       #Bits        BER
    0          6700     0.0000E+00

  Time =   0.1360280E+02
#Errors       #Bits        BER
    0          6800     0.0000E+00

  Time =   0.1380280E+02
#Errors       #Bits        BER
    0          6900     0.0000E+00

  Time =   0.1400280E+02
#Errors       #Bits        BER
    0          7000     0.0000E+00

  Time =   0.1420280E+02
#Errors       #Bits        BER
    0          7100     0.0000E+00

  Time =   0.1440280E+02
#Errors       #Bits        BER
    0          7200     0.0000E+00

  Time =   0.1460280E+02
#Errors       #Bits        BER
    0          7300     0.0000E+00

  Time =   0.1480280E+02
#Errors       #Bits        BER
    0          7400     0.0000E+00
  Time =  0.1500280E+02
#Errors     #Bits        BER
     0       7500     0.0000E+00

  Time =  0.1520280E+02
#Errors     #Bits        BER
     0       7600     0.0000E+00

  Time =  0.1540280E+02
#Errors     #Bits        BER
     0       7700     0.0000E+00

  Time =  0.1560280E+02
#Errors     #Bits        BER
     0       7800     0.0000E+00

  Time =  0.1580280E+02
#Errors     #Bits        BER
     0       7900     0.0000E+00

  Time =  0.1600280E+02
#Errors     #Bits        BER
     0       8000     0.0000E+00

  Time =  0.1620280E+02
#Errors     #Bits        BER
     0       8100     0.0000E+00

  Time =  0.1640280E+02
#Errors     #Bits        BER
     0       8200     0.0000E+00

  Time =  0.1660280E+02
#Errors     #Bits        BER
     0       8300     0.0000E+00

  Time =  0.1680280E+02
#Errors     #Bits        BER
     0       8400     0.0000E+00

  Time =  0.1700280E+02
#Errors     #Bits        BER
     0       8500     0.0000E+00

  Time =  0.1720280E+02
#Errors     #Bits        BER
     0       8600     0.0000E+00

  Time =  0.1740280E+02
#Errors     #Bits        BER
     0       8700     0.0000E+00

  Time =  0.1760280E+02
#Errors     #Bits        BER
     0       8800     0.0000E+00

  Time =  0.1780280E+02
#Errors     #Bits        BER
     0       8900     0.0000E+00
  Time =   0.1800280E+02
#Errors      #Bits         BER
    0         9000      0.0000E+00

  Time =   0.1820280E+02
#Errors      #Bits         BER
    0         9100      0.0000E+00

  Time =   0.1840280E+02
#Errors      #Bits         BER
    0         9200      0.0000E+00

  Time =   0.1860280E+02
#Errors      #Bits         BER
    0         9300      0.0000E+00

  Time =   0.1880280E+02
#Errors      #Bits         BER
    0         9400      0.0000E+00

  Time =   0.1900280E+02
#Errors      #Bits         BER
    0         9500      0.0000E+00

  Time =   0.1920280E+02
#Errors      #Bits         BER
    0         9600      0.0000E+00

  Time =   0.1940280E+02
#Errors      #Bits         BER
    0         9700      0.0000E+00

  Time =   0.1960280E+02
#Errors      #Bits         BER
    0         9800      0.0000E+00

  Time =   0.1980280E+02
#Errors      #Bits         BER
    0         9900      0.0000E+00
```

Appendix B

PROGRAMS IN C AND SIMULATION RESULTS FOR DATA FLOW OF SAMPLES IN THE HARDWARE DESIGN OF MCRM

```
/* Hardware behavioural simulation for a Multiplexed
Carrier Recovery Module (MCRM) */

/* This program shows the data flow of samples for 8 channels
for an estimation period of 4 samples for each channel */

#include <stdio.h>

/* l   is Address Generator for Samples
   agc is Address Generator for Channels
   ARE is Enable for Accumulation RAM
   SRE is Enable for Storage RAM */

main()
{
int adl=1;
int sdl=3;
int read_disable=0;
int l=1,en=0,aen=0,sen=0;
int count,m,k,i,q,j,adder_I,adder_Q;
int agc=0;
int h=1;
int I_temp,Q_temp;
int d=1,s=1,x=0,ARE=0,ctr=0,SRE=0;
FILE *fp,*fp1;

/* Allocate memory for RAMs AR & SR and ROMs OR & IR */

struct ram
        {
        int location[10];
        };
struct ram Accumulation_RAM_I,Accumulation_RAM_Q;

struct doubleram
        {
        int location_S[9];
        int location_C[9];
        };
struct doubleram Storage_RAM;

struct rom
        {
        int location_S;
        int location_C;
        };
struct rom Output_ROM;

struct irom
        {
        int location_I;
        int location_Q;
        };
struct irom Input_ROM;
```

```
                                /* Initialize the memory locations */

   for (j=1;j<=9;j++)
      {
      Storage_RAM.location_S[j]=0;
      Storage_RAM.location_C[j]=0;
      Accumulation_RAM_I.location[j]=0;
      Accumulation_RAM_Q.location[j]=0;
      Input_ROM.location_I=0;
      Input_ROM.location_Q=0;
      Output_ROM.location_S=0;
      Output_ROM.location_C=0;
      }

                                       /* Start the main program */

printf("Enter the count of Address Generator for Samples Please");
printf("\n");
scanf("%d",&count);
fp=fopen("sam.dat","r");
fp1=fopen("outcar.dat","w");
   fprintf(fp1,"Enter the count of Address Generator for Samples Please");
   fprintf(fp1, "\n");
   fprintf(fp1, "count");
   fprintf(fp1,"%2d", count);
   fprintf(fp1,"\n");

for (k=1;k<=count;k++)
   {
    fscanf(fp,"%d",&i);
    fscanf(fp,"%d",&q);
                                       /* When the clock is negative  */

    Output_ROM.location_C=adder_I;
    Output_ROM.location_S=adder_Q;
                                       /* Read of AR enabled */
    if (read_disable==1)
       {
       adder_I=Accumulation_RAM_I.location[d]+Input_ROM.location_I;
       adder_Q=Accumulation_RAM_Q.location[d]+Input_ROM.location_Q;
       }

                                       /* Read of AR disabled */

    if (read_disable==0)
      {
      adder_I=Input_ROM.location_I;
      adder_Q=Input_ROM.location_Q;
      h=h+1;
      if (h>8)
         {
         h=h-8;
         read_disable=1;
```

C-2

```
      }
    }
                                   /* Read input samples */
 Input_ROM.location_I=i;
 Input_ROM.location_Q=q;
 if (SRE==1)
 {                                 /* The clock is positive */
 Storage_RAM.location_C[s]=I_temp;
 Storage_RAM.location_S[s]=Q_temp;
 s=s+1;
 if (s>8)
    {
    s=s-8;
    SRE=0;
    }
 }

 I_temp=Output_ROM.location_C;
 Q_temp=Output_ROM.location_S;
                                   /* Write enable AR */
 if (ARE==1)
    {
    Accumulation_RAM_I.location[d]=adder_I;
    Accumulation_RAM_Q.location[d]=adder_Q;
    d=d+1;
    if (d>8)
       {
       d=d-8;

       }
    }
                             /* Print the data state for all units */

fprintf(fp1,"%3s %3s %3s %3s %3s %3s %3s %3s %3s %3s\n",
"AGS","AGC","IRI","IRQ","ARI","ARQ","ORC","ORS","SRC","SRS");
for (j=1;j<9;j++)
   {
   fprintf(fp1,"%2d",l);
   fprintf(fp1,"%4d",agc);
   fprintf(fp1,"%4d",Input_ROM.location_I);
   fprintf(fp1,"%4d",Input_ROM.location_Q);
   fprintf(fp1,"%4d",Accumulation_RAM_I.location[j]);
   fprintf(fp1,"%4d",Accumulation_RAM_Q.location[j]);
   fprintf(fp1,"%4d",Output_ROM.location_C);
   fprintf(fp1,"%4d",Output_ROM.location_S);
   fprintf(fp1,"%4d",Storage_RAM.location_C[j]);
   fprintf(fp1,"%4d",Storage_RAM.location_S[j]);
   fprintf(fp1,"\n");
   }
l=l+1;
                                   /* Define the control circuitry */


if (en==1)
```

```
              {
              if (l==4) SRE=1;
              }
          if (l==2) ARE=1;
          if (l>8)
              {
              l=(l-8);
              agc=agc+1;
              if (agc==3) en=1;
              if (agc>3) agc=agc-4;
              if (agc==0) en=0;
              printf("\n");
              }
           if (agc==0)
              {
              if (l==2) read_disable=0;
              }
      }
      fclose(fp);
      fclose(fp1);
      }
```

INPUT FILE FOR MCRM

```
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
```

OUTPUT FILE OF MCRM

```
Enter the count of Address Generator for Samples Please
count65
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
 1   0   1   2   0   0   0   0   0   0
 1   0   1   2   0   0   0   0   0   0
 1   0   1   2   0   0   0   0   0   0
 1   0   1   2   0   0   0   0   0   0
 1   0   1   2   0   0   0   0   0   0
 1   0   1   2   0   0   0   0   0   0
 1   0   1   2   0   0   0   0   0   0
 1   0   1   2   0   0   0   0   0   0
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
 2   0   1   2   1   2   0   0   0   0
 2   0   1   2   0   0   0   0   0   0
 2   0   1   2   0   0   0   0   0   0
 2   0   1   2   0   0   0   0   0   0
 2   0   1   2   0   0   0   0   0   0
 2   0   1   2   0   0   0   0   0   0
 2   0   1   2   0   0   0   0   0   0
 2   0   1   2   0   0   0   0   0   0
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
 3   0   1   2   1   2   1   2   0   0
 3   0   1   2   1   2   1   2   0   0
 3   0   1   2   0   0   1   2   0   0
 3   0   1   2   0   0   1   2   0   0
 3   0   1   2   0   0   1   2   0   0
 3   0   1   2   0   0   1   2   0   0
 3   0   1   2   0   0   1   2   0   0
 3   0   1   2   0   0   1   2   0   0
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
 4   0   1   2   1   2   1   2   0   0
 4   0   1   2   1   2   1   2   0   0
 4   0   1   2   1   2   1   2   0   0
 4   0   1   2   0   0   1   2   0   0
 4   0   1   2   0   0   1   2   0   0
 4   0   1   2   0   0   1   2   0   0
 4   0   1   2   0   0   1   2   0   0
 4   0   1   2   0   0   1   2   0   0
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
 5   0   1   2   1   2   1   2   0   0
 5   0   1   2   1   2   1   2   0   0
 5   0   1   2   1   2   1   2   0   0
 5   0   1   2   1   2   1   2   0   0
 5   0   1   2   0   0   1   2   0   0
 5   0   1   2   0   0   1   2   0   0
 5   0   1   2   0   0   1   2   0   0
 5   0   1   2   0   0   1   2   0   0
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
 6   0   1   2   1   2   1   2   0   0
 6   0   1   2   1   2   1   2   0   0
 6   0   1   2   1   2   1   2   0   0
 6   0   1   2   1   2   1   2   0   0
 6   0   1   2   1   2   1   2   0   0
 6   0   1   2   0   0   1   2   0   0
 6   0   1   2   0   0   1   2   0   0
 6   0   1   2   0   0   1   2   0   0
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
 7   0   1   2   1   2   1   2   0   0
 7   0   1   2   1   2   1   2   0   0
 7   0   1   2   1   2   1   2   0   0
 7   0   1   2   1   2   1   2   0   0
 7   0   1   2   1   2   1   2   0   0
 7   0   1   2   1   2   1   2   0   0
 7   0   1   2   0   0   1   2   0   0
 7   0   1   2   0   0   1   2   0   0
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
 8   0   1   2   1   2   1   2   0   0
 8   0   1   2   1   2   1   2   0   0
 8   0   1   2   1   2   1   2   0   0
 8   0   1   2   1   2   1   2   0   0
 8   0   1   2   1   2   1   2   0   0
 8   0   1   2   1   2   1   2   0   0
 8   0   1   2   1   2   1   2   0   0
 8   0   1   2   0   0   1   2   0   0
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
 1   1   1   2   1   2   1   2   0   0
 1   1   1   2   1   2   1   2   0   0
 1   1   1   2   1   2   1   2   0   0
 1   1   1   2   1   2   1   2   0   0
 1   1   1   2   1   2   1   2   0   0
 1   1   1   2   1   2   1   2   0   0
 1   1   1   2   1   2   1   2   0   0
 1   1   1   2   1   2   1   2   0   0
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
 2   1   1   2   2   4   1   2   0   0
 2   1   1   2   1   2   1   2   0   0
 2   1   1   2   1   2   1   2   0   0
 2   1   1   2   1   2   1   2   0   0
 2   1   1   2   1   2   1   2   0   0
 2   1   1   2   1   2   1   2   0   0
 2   1   1   2   1   2   1   2   0   0
 2   1   1   2   1   2   1   2   0   0
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
 3   1   1   2   2   4   2   4   0   0
 3   1   1   2   2   4   2   4   0   0
 3   1   1   2   1   2   2   4   0   0
 3   1   1   2   1   2   2   4   0   0
 3   1   1   2   1   2   2   4   0   0
 3   1   1   2   1   2   2   4   0   0
 3   1   1   2   1   2   2   4   0   0
 3   1   1   2   1   2   2   4   0   0
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
 4   1   1   2   2   4   2   4   0   0
 4   1   1   2   2   4   2   4   0   0
 4   1   1   2   2   4   2   4   0   0
 4   1   1   2   1   2   2   4   0   0
 4   1   1   2   1   2   2   4   0   0
 4   1   1   2   1   2   2   4   0   0
```

4	1	1	2	1	2	2	4	0	0
4	1	1	2	1	2	2	4	0	0
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
5	1	1	2	2	4	2	4	0	0
5	1	1	2	2	4	2	4	0	0
5	1	1	2	2	4	2	4	0	0
5	1	1	2	2	4	2	4	0	0
5	1	1	2	1	2	2	4	0	0
5	1	1	2	1	2	2	4	0	0
5	1	1	2	1	2	2	4	0	0
5	1	1	2	1	2	2	4	0	0
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
6	1	1	2	2	4	2	4	0	0
6	1	1	2	2	4	2	4	0	0
6	1	1	2	2	4	2	4	0	0
6	1	1	2	2	4	2	4	0	0
6	1	1	2	2	4	2	4	0	0
6	1	1	2	1	2	2	4	0	0
6	1	1	2	1	2	2	4	0	0
6	1	1	2	1	2	2	4	0	0
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
7	1	1	2	2	4	2	4	0	0
7	1	1	2	2	4	2	4	0	0
7	1	1	2	2	4	2	4	0	0
7	1	1	2	2	4	2	4	0	0
7	1	1	2	2	4	2	4	0	0
7	1	1	2	2	4	2	4	0	0
7	1	1	2	1	2	2	4	0	0
7	1	1	2	1	2	2	4	0	0
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
8	1	1	2	2	4	2	4	0	0
8	1	1	2	2	4	2	4	0	0
8	1	1	2	2	4	2	4	0	0
8	1	1	2	2	4	2	4	0	0
8	1	1	2	2	4	2	4	0	0
8	1	1	2	2	4	2	4	0	0
8	1	1	2	2	4	2	4	0	0
8	1	1	2	1	2	2	4	0	0
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
1	2	1	2	2	4	2	4	0	0
1	2	1	2	2	4	2	4	0	0
1	2	1	2	2	4	2	4	0	0
1	2	1	2	2	4	2	4	0	0
1	2	1	2	2	4	2	4	0	0
1	2	1	2	2	4	2	4	0	0
1	2	1	2	2	4	2	4	0	0
1	2	1	2	2	4	2	4	0	0
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
2	2	1	2	3	6	2	4	0	0
2	2	1	2	2	4	2	4	0	0
2	2	1	2	2	4	2	4	0	0
2	2	1	2	2	4	2	4	0	0
2	2	1	2	2	4	2	4	0	0
2	2	1	2	2	4	2	4	0	0

```
 2   2   1   2   2   4   2   4   0   0
 2   2   1   2   2   4   2   4   0   0
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
 3   2   1   2   3   6   3   6   0   0
 3   2   1   2   3   6   3   6   0   0
 3   2   1   2   2   4   3   6   0   0
 3   2   1   2   2   4   3   6   0   0
 3   2   1   2   2   4   3   6   0   0
 3   2   1   2   2   4   3   6   0   0
 3   2   1   2   2   4   3   6   0   0
 3   2   1   2   2   4   3   6   0   0
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
 4   2   1   2   3   6   3   6   0   0
 4   2   1   2   3   6   3   6   0   0
 4   2   1   2   3   6   3   6   0   0
 4   2   1   2   2   4   3   6   0   0
 4   2   1   2   2   4   3   6   0   0
 4   2   1   2   2   4   3   6   0   0
 4   2   1   2   2   4   3   6   0   0
 4   2   1   2   2   4   3   6   0   0
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
 5   2   1   2   3   6   3   6   0   0
 5   2   1   2   3   6   3   6   0   0
 5   2   1   2   3   6   3   6   0   0
 5   2   1   2   3   6   3   6   0   0
 5   2   1   2   2   4   3   6   0   0
 5   2   1   2   2   4   3   6   0   0
 5   2   1   2   2   4   3   6   0   0
 5   2   1   2   2   4   3   6   0   0
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
 6   2   1   2   3   6   3   6   0   0
 6   2   1   2   3   6   3   6   0   0
 6   2   1   2   3   6   3   6   0   0
 6   2   1   2   3   6   3   6   0   0
 6   2   1   2   3   6   3   6   0   0
 6   2   1   2   2   4   3   6   0   0
 6   2   1   2   2   4   3   6   0   0
 6   2   1   2   2   4   3   6   0   0
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
 7   2   1   2   3   6   3   6   0   0
 7   2   1   2   3   6   3   6   0   0
 7   2   1   2   3   6   3   6   0   0
 7   2   1   2   3   6   3   6   0   0
 7   2   1   2   3   6   3   6   0   0
 7   2   1   2   3   6   3   6   0   0
 7   2   1   2   2   4   3   6   0   0
 7   2   1   2   2   4   3   6   0   0
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
 8   2   1   2   3   6   3   6   0   0
 8   2   1   2   3   6   3   6   0   0
 8   2   1   2   3   6   3   6   0   0
 8   2   1   2   3   6   3   6   0   0
 8   2   1   2   3   6   3   6   0   0
 8   2   1   2   3   6   3   6   0   0
```

8	2	1	2	3	6	3	6	0	0
8	2	1	2	2	4	3	6	0	0
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
1	3	1	2	3	6	3	6	0	0
1	3	1	2	3	6	3	6	0	0
1	3	1	2	3	6	3	6	0	0
1	3	1	2	3	6	3	6	0	0
1	3	1	2	3	6	3	6	0	0
1	3	1	2	3	6	3	6	0	0
1	3	1	2	3	6	3	6	0	0
1	3	1	2	3	6	3	6	0	0
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
2	3	1	2	4	8	3	6	0	0
2	3	1	2	3	6	3	6	0	0
2	3	1	2	3	6	3	6	0	0
2	3	1	2	3	6	3	6	0	0
2	3	1	2	3	6	3	6	0	0
2	3	1	2	3	6	3	6	0	0
2	3	1	2	3	6	3	6	0	0
2	3	1	2	3	6	3	6	0	0
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
3	3	1	2	4	8	4	8	0	0
3	3	1	2	4	8	4	8	0	0
3	3	1	2	3	6	4	8	0	0
3	3	1	2	3	6	4	8	0	0
3	3	1	2	3	6	4	8	0	0
3	3	1	2	3	6	4	8	0	0
3	3	1	2	3	6	4	8	0	0
3	3	1	2	3	6	4	8	0	0
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
4	3	1	2	4	8	4	8	4	8
4	3	1	2	4	8	4	8	0	0
4	3	1	2	4	8	4	8	0	0
4	3	1	2	3	6	4	8	0	0
4	3	1	2	3	6	4	8	0	0
4	3	1	2	3	6	4	8	0	0
4	3	1	2	3	6	4	8	0	0
4	3	1	2	3	6	4	8	0	0
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
5	3	1	2	4	8	4	8	4	8
5	3	1	2	4	8	4	8	4	8
5	3	1	2	4	8	4	8	0	0
5	3	1	2	4	8	4	8	0	0
5	3	1	2	3	6	4	8	0	0
5	3	1	2	3	6	4	8	0	0
5	3	1	2	3	6	4	8	0	0
5	3	1	2	3	6	4	8	0	0
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
6	3	1	2	4	8	4	8	4	8
6	3	1	2	4	8	4	8	4	8
6	3	1	2	4	8	4	8	4	8
6	3	1	2	4	8	4	8	0	0
6	3	1	2	4	8	4	8	0	0
6	3	1	2	3	6	4	8	0	0

6	3	1	2	3	6	4	8	0	0
6	3	1	2	3	6	4	8	0	0
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
7	3	1	2	4	8	4	8	4	8
7	3	1	2	4	8	4	8	4	8
7	3	1	2	4	8	4	8	4	8
7	3	1	2	4	8	4	8	4	8
7	3	1	2	4	8	4	8	0	0
7	3	1	2	4	8	4	8	0	0
7	3	1	2	3	6	4	8	0	0
7	3	1	2	3	6	4	8	0	0
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
8	3	1	2	4	8	4	8	4	8
8	3	1	2	4	8	4	8	4	8
8	3	1	2	4	8	4	8	4	8
8	3	1	2	4	8	4	8	4	8
8	3	1	2	4	8	4	8	4	8
8	3	1	2	4	8	4	8	0	0
8	3	1	2	4	8	4	8	0	0
8	3	1	2	3	6	4	8	0	0
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
1	0	1	2	4	8	4	8	4	8
1	0	1	2	4	8	4	8	4	8
1	0	1	2	4	8	4	8	4	8
1	0	1	2	4	8	4	8	4	8
1	0	1	2	4	8	4	8	4	8
1	0	1	2	4	8	4	8	4	8
1	0	1	2	4	8	4	8	0	0
1	0	1	2	4	8	4	8	0	0
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
2	0	1	2	1	2	4	8	4	8
2	0	1	2	4	8	4	8	4	8
2	0	1	2	4	8	4	8	4	8
2	0	1	2	4	8	4	8	4	8
2	0	1	2	4	8	4	8	4	8
2	0	1	2	4	8	4	8	4	8
2	0	1	2	4	8	4	8	4	8
2	0	1	2	4	8	4	8	0	0
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
3	0	1	2	1	2	1	2	4	8
3	0	1	2	1	2	1	2	4	8
3	0	1	2	4	8	1	2	4	8
3	0	1	2	4	8	1	2	4	8
3	0	1	2	4	8	1	2	4	8
3	0	1	2	4	8	1	2	4	8
3	0	1	2	4	8	1	2	4	8
3	0	1	2	4	8	1	2	4	8
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
4	0	1	2	1	2	1	2	4	8
4	0	1	2	1	2	1	2	4	8
4	0	1	2	1	2	1	2	4	8
4	0	1	2	4	8	1	2	4	8
4	0	1	2	4	8	1	2	4	8
4	0	1	2	4	8	1	2	4	8

4	0	1	2	4	8	1	2	4	8
4	0	1	2	4	8	1	2	4	8

AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
5	0	1	2	1	2	1	2	4	8
5	0	1	2	1	2	1	2	4	8
5	0	1	2	1	2	1	2	4	8
5	0	1	2	1	2	1	2	4	8
5	0	1	2	4	8	1	2	4	8
5	0	1	2	4	8	1	2	4	8
5	0	1	2	4	8	1	2	4	8
5	0	1	2	4	8	1	2	4	8

AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
6	0	1	2	1	2	1	2	4	8
6	0	1	2	1	2	1	2	4	8
6	0	1	2	1	2	1	2	4	8
6	0	1	2	1	2	1	2	4	8
6	0	1	2	1	2	1	2	4	8
6	0	1	2	4	8	1	2	4	8
6	0	1	2	4	8	1	2	4	8
6	0	1	2	4	8	1	2	4	8

AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
7	0	1	2	1	2	1	2	4	8
7	0	1	2	1	2	1	2	4	8
7	0	1	2	1	2	1	2	4	8
7	0	1	2	1	2	1	2	4	8
7	0	1	2	1	2	1	2	4	8
7	0	1	2	1	2	1	2	4	8
7	0	1	2	4	8	1	2	4	8
7	0	1	2	4	8	1	2	4	8

AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
8	0	1	2	1	2	1	2	4	8
8	0	1	2	1	2	1	2	4	8
8	0	1	2	1	2	1	2	4	8
8	0	1	2	1	2	1	2	4	8
8	0	1	2	1	2	1	2	4	8
8	0	1	2	1	2	1	2	4	8
8	0	1	2	1	2	1	2	4	8
8	0	1	2	4	8	1	2	4	8

AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
1	1	1	2	1	2	1	2	4	8
1	1	1	2	1	2	1	2	4	8
1	1	1	2	1	2	1	2	4	8
1	1	1	2	1	2	1	2	4	8
1	1	1	2	1	2	1	2	4	8
1	1	1	2	1	2	1	2	4	8
1	1	1	2	1	2	1	2	4	8
1	1	1	2	1	2	1	2	4	8

AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
2	1	1	2	2	4	1	2	4	8
2	1	1	2	1	2	1	2	4	8
2	1	1	2	1	2	1	2	4	8
2	1	1	2	1	2	1	2	4	8
2	1	1	2	1	2	1	2	4	8
2	1	1	2	1	2	1	2	4	8

2	1	1	2	1	2	1	2	4	8
2	1	1	2	1	2	1	2	4	8
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
3	1	1	2	2	4	2	4	4	8
3	1	1	2	2	4	2	4	4	8
3	1	1	2	1	2	2	4	4	8
3	1	1	2	1	2	2	4	4	8
3	1	1	2	1	2	2	4	4	8
3	1	1	2	1	2	2	4	4	8
3	1	1	2	1	2	2	4	4	8
3	1	1	2	1	2	2	4	4	8
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
4	1	1	2	2	4	2	4	4	8
4	1	1	2	2	4	2	4	4	8
4	1	1	2	2	4	2	4	4	8
4	1	1	2	1	2	2	4	4	8
4	1	1	2	1	2	2	4	4	8
4	1	1	2	1	2	2	4	4	8
4	1	1	2	1	2	2	4	4	8
4	1	1	2	1	2	2	4	4	8
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
5	1	1	2	2	4	2	4	4	8
5	1	1	2	2	4	2	4	4	8
5	1	1	2	2	4	2	4	4	8
5	1	1	2	2	4	2	4	4	8
5	1	1	2	1	2	2	4	4	8
5	1	1	2	1	2	2	4	4	8
5	1	1	2	1	2	2	4	4	8
5	1	1	2	1	2	2	4	4	8
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
6	1	1	1	2	4	2	4	4	8
6	1	1	1	2	4	2	4	4	8
6	1	1	1	2	4	2	4	4	8
6	1	1	1	2	4	2	4	4	8
6	1	1	1	2	4	2	4	4	8
6	1	1	1	1	2	2	4	4	8
6	1	1	1	1	2	2	4	4	8
6	1	1	1	1	2	2	4	4	8
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
7	1	1	1	2	4	2	4	4	8
7	1	1	1	2	4	2	4	4	8
7	1	1	1	2	4	2	4	4	8
7	1	1	1	2	4	2	4	4	8
7	1	1	1	2	4	2	4	4	8
7	1	1	1	2	3	2	4	4	8
7	1	1	1	1	2	2	4	4	8
7	1	1	1	1	2	2	4	4	8
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
8	1	1	1	2	4	2	3	4	8
8	1	1	1	2	4	2	3	4	8
8	1	1	1	2	4	2	3	4	8
8	1	1	1	2	4	2	3	4	8
8	1	1	1	2	4	2	3	4	8
8	1	1	1	2	3	2	3	4	8

```
  8   1   1   1   2   3   2   3   4   8
  8   1   1   1   1   2   2   3   4   8
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
  1   2   1   1   2   4   2   3   4   8
  1   2   1   1   2   4   2   3   4   8
  1   2   1   1   2   4   2   3   4   8
  1   2   1   1   2   4   2   3   4   8
  1   2   1   1   2   4   2   3   4   8
  1   2   1   1   2   3   2   3   4   8
  1   2   1   1   2   3   2   3   4   8
  1   2   1   1   2   3   2   3   4   8
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
  2   2   1   1   3   5   2   3   4   8
  2   2   1   1   2   4   2   3   4   8
  2   2   1   1   2   4   2   3   4   8
  2   2   1   1   2   4   2   3   4   8
  2   2   1   1   2   4   2   3   4   8
  2   2   1   1   2   3   2   3   4   8
  2   2   1   1   2   3   2   3   4   8
  2   2   1   1   2   3   2   3   4   8
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
  3   2   1   1   3   5   3   5   4   8
  3   2   1   1   3   5   3   5   4   8
  3   2   1   1   2   4   3   5   4   8
  3   2   1   1   2   4   3   5   4   8
  3   2   1   1   2   4   3   5   4   8
  3   2   1   1   2   3   3   5   4   8
  3   2   1   1   2   3   3   5   4   8
  3   2   1   1   2   3   3   5   4   8
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
  4   2   1   1   3   5   3   5   4   8
  4   2   1   1   3   5   3   5   4   8
  4   2   1   1   3   5   3   5   4   8
  4   2   1   1   2   4   3   5   4   8
  4   2   1   1   2   4   3   5   4   8
  4   2   1   1   2   3   3   5   4   8
  4   2   1   1   2   3   3   5   4   8
  4   2   1   1   2   3   3   5   4   8
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
  5   2   1   1   3   5   3   5   4   8
  5   2   1   1   3   5   3   5   4   8
  5   2   1   1   3   5   3   5   4   8
  5   2   1   1   3   5   3   5   4   8
  5   2   1   1   2   4   3   5   4   8
  5   2   1   1   2   3   3   5   4   8
  5   2   1   1   2   3   3   5   4   8
  5   2   1   1   2   3   3   5   4   8
AGS AGC IRI IRQ ARI ARQ ORC ORS SRC SRS
  6   2   1   1   3   5   3   5   4   8
  6   2   1   1   3   5   3   5   4   8
  6   2   1   1   3   5   3   5   4   8
  6   2   1   1   3   5   3   5   4   8
  6   2   1   1   3   5   3   5   4   8
  6   2   1   1   2   3   3   5   4   8
```

6	2	1	1	2	3	3	5	4	8
6	2	1	1	2	3	3	5	4	8
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
7	2	1	1	3	5	3	5	4	8
7	2	1	1	3	5	3	5	4	8
7	2	1	1	3	5	3	5	4	8
7	2	1	1	3	5	3	5	4	8
7	2	1	1	3	5	3	5	4	8
7	2	1	1	3	4	3	5	4	8
7	2	1	1	2	3	3	5	4	8
7	2	1	1	2	3	3	5	4	8
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
8	2	1	1	3	5	3	4	4	8
8	2	1	1	3	5	3	4	4	8
8	2	1	1	3	5	3	4	4	8
8	2	1	1	3	5	3	4	4	8
8	2	1	1	3	5	3	4	4	8
8	2	1	1	3	4	3	4	4	8
8	2	1	1	3	4	3	4	4	8
8	2	1	1	2	3	3	4	4	8
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
1	3	1	1	3	5	3	4	4	8
1	3	1	1	3	5	3	4	4	8
1	3	1	1	3	5	3	4	4	8
1	3	1	1	3	5	3	4	4	8
1	3	1	1	3	5	3	4	4	8
1	3	1	1	3	4	3	4	4	8
1	3	1	1	3	4	3	4	4	8
1	3	1	1	3	4	3	4	4	8
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
2	3	1	1	4	6	3	4	4	8
2	3	1	1	3	5	3	4	4	8
2	3	1	1	3	5	3	4	4	8
2	3	1	1	3	5	3	4	4	8
2	3	1	1	3	5	3	4	4	8
2	3	1	1	3	4	3	4	4	8
2	3	1	1	3	4	3	4	4	8
2	3	1	1	3	4	3	4	4	8
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
3	3	1	1	4	6	4	6	4	8
3	3	1	1	4	6	4	6	4	8
3	3	1	1	3	5	4	6	4	8
3	3	1	1	3	5	4	6	4	8
3	3	1	1	3	5	4	6	4	8
3	3	1	1	3	4	4	6	4	8
3	3	1	1	3	4	4	6	4	8
3	3	1	1	3	4	4	6	4	8
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
4	3	1	1	4	6	4	6	4	6
4	3	1	1	4	6	4	6	4	8
4	3	1	1	4	6	4	6	4	8
4	3	1	1	3	5	4	6	4	8
4	3	1	1	3	5	4	6	4	8
4	3	1	1	3	4	4	6	4	8

4	3	1	1	3	4	4	6	4	8
4	3	1	1	3	4	4	6	4	8
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
5	3	1	1	4	6	4	6	4	6
5	3	1	1	4	6	4	6	4	6
5	3	1	1	4	6	4	6	4	8
5	3	1	1	4	6	4	6	4	8
5	3	1	1	3	5	4	6	4	8
5	3	1	1	3	4	4	6	4	8
5	3	1	1	3	4	4	6	4	8
5	3	1	1	3	4	4	6	4	8
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
6	3	1	1	4	6	4	6	4	6
6	3	1	1	4	6	4	6	4	6
6	3	1	1	4	6	4	6	4	6
6	3	1	1	4	6	4	6	4	8
6	3	1	1	4	6	4	6	4	8
6	3	1	1	3	4	4	6	4	8
6	3	1	1	3	4	4	6	4	8
6	3	1	1	3	4	4	6	4	8
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
7	3	1	1	4	6	4	6	4	6
7	3	1	1	4	6	4	6	4	6
7	3	1	1	4	6	4	6	4	6
7	3	1	1	4	6	4	6	4	6
7	3	1	1	4	6	4	6	4	8
7	3	1	1	4	5	4	6	4	8
7	3	1	1	3	4	4	6	4	8
7	3	1	1	3	4	4	6	4	8
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
8	3	1	1	4	6	4	5	4	6
8	3	1	1	4	6	4	5	4	6
8	3	1	1	4	6	4	5	4	6
8	3	1	1	4	6	4	5	4	6
8	3	1	1	4	6	4	5	4	6
8	3	1	1	[illegible]	5	4	5	4	8
8	3	1	1	[illegible]	5	4	5	4	8
8	3	1	1	[illegible]	4	4	5	4	8
AGS	AGC	IRI	IRQ	ARI	ARQ	ORC	ORS	SRC	SRS
1	0	1	1		6	4	5	4	6
1	0	1	1		6	4	5	4	6
1	0	1	1		6	4	5	4	6
1	0	1	1		6	4	5	4	6
1	0	1	1		6	4	5	4	6
1	0	1	1		5	4	5	4	5
1	0	1	1		5	4	5	4	8
1	0	1	1		5	4	5	4	8

Appendix C

PROGRAMS IN C AND SIMULATION RESULTS FOR DATA FLOW OF SAMPLES IN THE HARDWARE DESIGN OF MRBS

```
/* Hardware behavioural simulation for Multiplexed RAM
Buffer for Samples (MRBS) */

/* This program describes the data flow for storage of
input samples for 8 channels for an estimation period of
4 samples for each channel */

/* agc is Address Generator for Channels
   l   is Address Generator for Samples */

#include <stdio.h>
main()
{
int l=1;
int agc=0;
int count,m,n,k,i,j;
FILE *fp,*fp1;


                                        /* Allocate memory for RAMs */
struct rambuffer
        {
        int latch;
        int location[9];
        };
struct rambuffer ram[5];
                                    /* Initialization of memory */
for (i=1;i<=4;i++)
   {
   for (j=1;j<=8;j++)
      {
      ram[i].location[j]=0;
      }
   ram[i].latch=0;
   }
                                        /* Start Main program */

printf("Enter the count of Address Generator for Samples Please");
printf("\n");
scanf("%d",&count);
fp1=fopen("outram.c","w");
fp=fopen("sampl.c","r");
fprintf(fp1,"Enter the count of Address Generator for Samples Please");
fprintf(fp1,"\n");
fprintf(fp1, "count");
fprintf(fp1, "%2d", count);
fprintf(fp1, "\n");

for (i=1;i<=count;i++)
   {
   fscanf(fp,"%d",&k);
   ram[1].latch=k;                      /* Memory read */
   for (m=2;m<5;m++)
      {
      ram[m].latch=ram[m-1].location[l];
```

```
         }
      for (m=1;m<5;m++)                         /* Memory write */
         {
         ram[m].location[l]=ram[m].latch;
         }

                                                /* Display MRBS structure */

      fprintf(fp1,"%3s %3s %3s %3s %3s %3s %3s %3s %3s %3s\n",
      "AGS", "AGC","LA1","RA1","LA2","RA2","LA3","RA3","LA4","RA4");
      for (j=1;j<9;j++)
         {
         fprintf(fp1,"%2d",l);
         fprintf(fp1,"%4d",agc);
         for (n=1;n<5;n++)
            {
            fprintf(fp1,"%4d",ram[n].latch);
            fprintf(fp1,"%4d",ram[n].location[j]);
            }
         fprintf(fp1,"\n");
         }
                                               /* Increment AGC and AGS */
      l=l+1;
      if (l>8)
         {
         l=l-8;
         agc=agc+1;
         if (agc>3) agc=agc-4;
         }
      }
   fclose(fp);
   fclose(fp1);
   }
```

INPUT FILE FOR MRBS

```
01 02 03 04 05 06 07 08 11 12 13 14 15
16 17 18 21 22 23 24 25 26 27 28 31 32
33 34 35 36 37 38 41 42 43 44 45 46 47
48 51 52 53 54 56 57 58 61 62 63 64 65
66 67 68
```

OUTPUT FILE OF MRBS

```
Enter the count of Address Generator for Samples Please
count30
AGS AGC LA1 RA1 LA2 RA2 LA3 RA3 LA4 RA4
 1   0   1   1   0   0   0   0   0   0
 1   0   1   0   0   0   0   0   0   0
 1   0   1   0   0   0   0   0   0   0
 1   0   1   0   0   0   0   0   0   0
 1   0   1   0   0   0   0   0   0   0
 1   0   1   0   0   0   0   0   0   0
 1   0   1   0   0   0   0   0   0   0
 1   0   1   0   0   0   0   0   0   0
AGS AGC LA1 RA1 LA2 RA2 LA3 RA3 LA4 RA4
 2   0   2   1   0   0   0   0   0   0
 2   0   2   2   0   0   0   0   0   0
 2   0   2   0   0   0   0   0   0   0
 2   0   2   0   0   0   0   0   0   0
 2   0   2   0   0   0   0   0   0   0
 2   0   2   0   0   0   0   0   0   0
 2   0   2   0   0   0   0   0   0   0
 2   0   2   0   0   0   0   0   0   0
AGS AGC LA1 RA1 LA2 RA2 LA3 RA3 LA4 RA4
 3   0   3   1   0   0   0   0   0   0
 3   0   3   2   0   0   0   0   0   0
 3   0   3   3   0   0   0   0   0   0
 3   0   3   0   0   0   0   0   0   0
 3   0   3   0   0   0   0   0   0   0
 3   0   3   0   0   0   0   0   0   0
 3   0   3   0   0   0   0   0   0   0
 3   0   3   0   0   0   0   0   0   0
AGS AGC LA1 RA1 LA2 RA2 LA3 RA3 LA4 RA4
 4   0   4   1   0   0   0   0   0   0
 4   0   4   2   0   0   0   0   0   0
 4   0   4   3   0   0   0   0   0   0
 4   0   4   4   0   0   0   0   0   0
 4   0   4   0   0   0   0   0   0   0
 4   0   4   0   0   0   0   0   0   0
 4   0   4   0   0   0   0   0   0   0
 4   0   4   0   0   0   0   0   0   0
AGS AGC LA1 RA1 LA2 RA2 LA3 RA3 LA4 RA4
 5   0   5   1   0   0   0   0   0   0
 5   0   5   2   0   0   0   0   0   0
 5   0   5   3   0   0   0   0   0   0
 5   0   5   4   0   0   0   0   0   0
 5   0   5   5   0   0   0   0   0   0
 5   0   5   0   0   0   0   0   0   0
 5   0   5   0   0   0   0   0   0   0
 5   0   5   0   0   0   0   0   0   0
AGS AGC LA1 RA1 LA2 RA2 LA3 RA3 LA4 RA4
 6   0   6   1   0   0   0   0   0   0
 6   0   6   2   0   0   0   0   0   0
 6   0   6   3   0   0   0   0   0   0
 6   0   6   4   0   0   0   0   0   0
 6   0   6   5   0   0   0   0   0   0
 6   0   6   6   0   0   0   0   0   0
 6   0   6   0   0   0   0   0   0   0
 6   0   6   0   0   0   0   0   0   0
AGS AGC LA1 RA1 LA2 RA2 LA3 RA3 LA4 RA4
 7   0   7   1   0   0   0   0   0   0
 7   0   7   2   0   0   0   0   0   0
 7   0   7   3   0   0   0   0   0   0
 7   0   7   4   0   0   0   0   0   0
 7   0   7   5   0   0   0   0   0   0
 7   0   7   6   0   0   0   0   0   0
 7   0   7   7   0   0   0   0   0   0
 7   0   7   0   0   0   0   0   0   0
AGS AGC LA1 RA1 LA2 RA2 LA3 RA3 LA4 RA4
 8   0   8   1   0   0   0   0   0   0
 8   0   8   2   0   0   0   0   0   0
 8   0   8   3   0   0   0   0   0   0
 8   0   8   4   0   0   0   0   0   0
 8   0   8   5   0   0   0   0   0   0
 8   0   8   6   0   0   0   0   0   0
 8   0   8   7   0   0   0   0   0   0
 8   0   8   8   0   0   0   0   0   0
AGS AGC LA1 RA1 LA2 RA2 LA3 RA3 LA4 RA4
 1   1  11  11   1   1   0   0   0   0
 1   1  11   2   1   0   0   0   0   0
 1   1  11   3   1   0   0   0   0   0
 1   1  11   4   1   0   0   0   0   0
 1   1  11   5   1   0   0   0   0   0
 1   1  11   6   1   0   0   0   0   0
 1   1  11   7   1   0   0   0   0   0
 1   1  11   8   1   0   0   0   0   0
AGS AGC LA1 RA1 LA2 RA2 LA3 RA3 LA4 RA4
 2   1  12  11   2   1   0   0   0   0
 2   1  12  12   2   2   0   0   0   0
 2   1  12   3   2   0   0   0   0   0
 2   1  12   4   2   0   0   0   0   0
 2   1  12   5   2   0   0   0   0   0
 2   1  12   6   2   0   0   0   0   0
 2   1  12   7   2   0   0   0   0   0
 2   1  12   8   2   0   0   0   0   0
AGS AGC LA1 RA1 LA2 RA2 LA3 RA3 LA4 RA4
 3   1  13  11   3   1   0   0   0   0
 3   1  13  12   3   2   0   0   0   0
 3   1  13  13   3   3   0   0   0   0
 3   1  13   4   3   0   0   0   0   0
 3   1  13   5   3   0   0   0   0   0
 3   1  13   6   3   0   0   0   0   0
 3   1  13   7   3   0   0   0   0   0
 3   1  13   8   3   0   0   0   0   0
AGS AGC LA1 RA1 LA2 RA2 LA3 RA3 LA4 RA4
 4   1  14  11   4   1   0   0   0   0
 4   1  14  12   4   2   0   0   0   0
 4   1  14  13   4   3   0   0   0   0
 4   1  14  14   4   4   0   0   0   0
 4   1  14   5   4   0   0   0   0   0
 4   1  14   6   4   0   0   0   0   0
```

AGS	AGC	LA1	RA1	LA2	RA2	LA3	RA3	LA4	RA4
4	1	14	7	4	0	0	0	0	0
4	1	14	8	4	0	0	0	0	0
AGS	AGC	LA1	RA1	LA2	RA2	LA3	RA3	LA4	RA4
5	1	15	11	5	1	0	0	0	0
5	1	15	12	5	2	0	0	0	0
5	1	15	13	5	3	0	0	0	0
5	1	15	14	5	4	0	0	0	0
5	1	15	15	5	5	0	0	0	0
5	1	15	6	5	0	0	0	0	0
5	1	15	7	5	0	0	0	0	0
5	1	15	8	5	0	0	0	0	0
AGS	AGC	LA1	RA1	LA2	RA2	LA3	RA3	LA4	RA4
6	1	16	11	6	1	0	0	0	0
6	1	16	12	6	2	0	0	0	0
6	1	16	13	6	3	0	0	0	0
6	1	16	14	6	4	0	0	0	0
6	1	16	15	6	5	0	0	0	0
6	1	16	16	6	6	0	0	0	0
6	1	16	7	6	0	0	0	0	0
6	1	16	8	6	0	0	0	0	0
AGS	AGC	LA1	RA1	LA2	RA2	LA3	RA3	LA4	RA4
7	1	17	11	7	1	0	0	0	0
7	1	17	12	7	2	0	0	0	0
7	1	17	13	7	3	0	0	0	0
7	1	17	14	7	4	0	0	0	0
7	1	17	15	7	5	0	0	0	0
7	1	17	16	7	6	0	0	0	0
7	1	17	17	7	7	0	0	0	0
7	1	17	8	7	0	0	0	0	0
AGS	AGC	LA1	RA1	LA2	RA2	LA3	RA3	LA4	RA4
8	1	18	11	8	1	0	0	0	0
8	1	18	12	8	2	0	0	0	0
8	1	18	13	8	3	0	0	0	0
8	1	18	14	8	4	0	0	0	0
8	1	18	15	8	5	0	0	0	0
8	1	18	16	8	6	0	0	0	0
8	1	18	17	8	7	0	0	0	0
8	1	18	18	8	8	0	0	0	0
AGS	AGC	LA1	RA1	LA2	RA2	LA3	RA3	LA4	RA4
1	2	21	21	11	11	1	1	0	0
1	2	21	12	11	2	1	0	0	0
1	2	21	13	11	3	1	0	0	0
1	2	21	14	11	4	1	0	0	0
1	2	21	15	11	5	1	0	0	0
1	2	21	16	11	6	1	0	0	0
1	2	21	17	11	7	1	0	0	0
1	2	21	18	11	8	1	0	0	0
AGS	AGC	LA1	RA1	LA2	RA2	LA3	RA3	LA4	RA4
2	2	22	21	12	11	2	1	0	0
2	2	22	22	12	12	2	2	0	0
2	2	22	13	12	3	2	0	0	0
2	2	22	14	12	4	2	0	0	0
2	2	22	15	12	5	2	0	0	0
2	2	22	16	12	6	2	0	0	0

```
  2    2   22   17   12    7    2    0    0    0
  2    2   22   18   12    8    2    0    0    0
AGS  AGC  LA1  RA1  LA2  RA2  LA3  RA3  LA4  RA4
  3    2   23   21   13   11    3    1    0    0
  3    2   23   22   13   12    3    2    0    0
  3    2   23   23   13   13    3    3    0    0
  3    2   23   14   13    4    3    0    0    0
  3    2   23   15   13    5    3    0    0    0
  3    2   23   16   13    6    3    0    0    0
  3    2   23   17   13    7    3    0    0    0
  3    2   23   18   13    8    3    0    0    0
AGS  AGC  LA1  RA1  LA2  RA2  LA3  RA3  LA4  RA4
  4    2   24   21   14   11    4    1    0    0
  4    2   24   22   14   12    4    2    0    0
  4    2   24   23   14   13    4    3    0    0
  4    2   24   24   14   14    4    4    0    0
  4    2   24   15   14    5    4    0    0    0
  4    2   24   16   14    6    4    0    0    0
  4    2   24   17   14    7    4    0    0    0
  4    2   24   18   14    8    4    0    0    0
AGS  AGC  LA1  RA1  LA2  RA2  LA3  RA3  LA4  RA4
  5    2   25   21   15   11    5    1    0    0
  5    2   25   22   15   12    5    2    0    0
  5    2   25   23   15   13    5    3    0    0
  5    2   25   24   15   14    5    4    0    0
  5    2   25   25   15   15    5    5    0    0
  5    2   25   16   15    6    5    0    0    0
  5    2   25   17   15    7    5    0    0    0
  5    2   25   18   15    8    5    0    0    0
AGS  AGC  LA1  RA1  LA2  RA2  LA3  RA3  LA4  RA4
  6    2   26   21   16   11    6    1    0    0
  6    2   26   22   16   12    6    2    0    0
  6    2   26   23   16   13    6    3    0    0
  6    2   26   24   16   14    6    4    0    0
  6    2   26   25   16   15    6    5    0    0
  6    2   26   26   16   16    6    6    0    0
  6    2   26   17   16    7    6    0    0    0
  6    2   26   18   16    8    6    0    0    0
AGS  AGC  LA1  RA1  LA2  RA2  LA3  RA3  LA4  RA4
  7    2   27   21   17   11    7    1    0    0
  7    2   27   22   17   12    7    2    0    0
  7    2   27   23   17   13    7    3    0    0
  7    2   27   24   17   14    7    4    0    0
  7    2   27   25   17   15    7    5    0    0
  7    2   27   26   17   16    7    6    0    0
  7    2   27   27   17   17    7    7    0    0
  7    2   27   18   17    8    7    0    0    0
AGS  AGC  LA1  RA1  LA2  RA2  LA3  RA3  LA4  RA4
  8    2   28   21   18   11    8    1    0    0
  8    2   28   22   18   12    8    2    0    0
  8    2   28   23   18   13    8    3    0    0
  8    2   28   24   18   14    8    4    0    0
  8    2   28   25   18   15    8    5    0    0
  8    2   28   26   18   16    8    6    0    0
```

8	2	28	27	18	17	8	7	0	0
8	2	28	28	18	18	8	8	0	0
AGS	AGC	LA1	RA1	LA2	RA2	LA3	RA3	LA4	RA4
1	3	31	31	21	21	11	11	1	1
1	3	31	22	21	12	11	2	1	0
1	3	31	23	21	13	11	3	1	0
1	3	31	24	21	14	11	4	1	0
1	3	31	25	21	15	11	5	1	0
1	3	31	26	21	16	11	6	1	0
1	3	31	27	21	17	11	7	1	0
1	3	31	28	21	18	11	8	1	0
AGS	AGC	LA1	RA1	LA2	RA2	LA3	RA3	LA4	RA4
2	3	32	31	22	21	12	11	2	1
2	3	32	32	22	22	12	12	2	2
2	3	32	23	22	13	12	3	2	0
2	3	32	24	22	14	12	4	2	0
2	3	32	25	22	15	12	5	2	0
2	3	32	26	22	16	12	6	2	0
2	3	32	27	22	17	12	7	2	0
2	3	32	28	22	18	12	8	2	0
AGS	AGC	LA1	RA1	LA2	RA2	LA3	RA3	LA4	RA4
3	3	33	31	23	21	13	11	3	1
3	3	33	32	23	22	13	12	3	2
3	3	33	33	23	23	13	13	3	3
3	3	33	24	23	14	13	4	3	0
3	3	33	25	23	15	13	5	3	0
3	3	33	26	23	16	13	6	3	0
3	3	33	27	23	17	13	7	3	0
3	3	33	28	23	18	13	8	3	0
AGS	AGC	LA1	RA1	LA2	RA2	LA3	RA3	LA4	RA4
4	3	34	31	24	21	14	11	4	1
4	3	34	32	24	22	14	12	4	2
4	3	34	33	24	23	14	13	4	3
4	3	34	34	24	24	14	14	4	4
4	3	34	25	24	15	14	5	4	0
4	3	34	26	24	16	14	6	4	0
4	3	34	27	24	17	14	7	4	0
4	3	34	28	24	18	14	8	4	0
AGS	AGC	LA1	RA1	LA2	RA2	LA3	RA3	LA4	RA4
5	3	35	31	25	21	15	11	5	1
5	3	35	32	25	22	15	12	5	2
5	3	35	33	25	23	15	13	5	3
5	3	35	34	25	24	15	14	5	4
5	3	35	35	25	25	15	15	5	5
5	3	35	26	25	16	15	6	5	0
5	3	35	27	25	17	15	7	5	0
5	3	35	28	25	18	15	8	5	0
AGS	AGC	LA1	RA1	LA2	RA2	LA3	RA3	LA4	RA4
6	3	36	31	26	21	16	11	6	1
6	3	36	32	26	22	16	12	6	2
6	3	36	33	26	23	16	13	6	3
6	3	36	34	26	24	16	14	6	4
6	3	36	35	26	25	16	15	6	5
6	3	36	36	26	26	16	16	6	6

6	3	36	27	26	17	16	7	6	0
6	3	36	28	26	18	16	8	6	0

Appendix D

PROGRAMS IN C AND SIMULATION RESULTS FOR DATA FLOW OF SAMPLES IN THE HARDWARE DESIGN OF MDRM

```
/* Hardware behavioural simulation for Multiplexed
 Data Recovery Module (MDRM) */

/* This program describes the data flow in the MDRM
for 8 channels and 4 sample estimation period */

#include <stdio.h>

/* agc is Address Generator for Channels
   l   is Address Generator for Samples */

main()
{                        /* Initialize variables */
int l=1,d=1;
int we=0,agc=0;
int count,m,k,i,j,q,s,c,I_inv,Q_inv;
FILE *fp, *fp1;
int mult[5];
int add,sub,latch[3];
                         /* Define DDR memory structure */
struct ram
        {
        int location_I[9];
        int location_Q[9];
        };
struct ram DDR;
                         /* Iniatialization */

   for (j=1;j<9;j++)
      {
      add=0;
      sub=0;
      latch[1]=0;
      latch[2]=0;
      DDR.location_I[j]=0;
      DDR.location_Q[j]=0;
      }

printf("Enter the count of Address Generator for Samples Please");
printf("\n");
scanf("%d",&count);
fp=fopen("sadat.c","r");
fp1=fopen("outdat.c","w");
                         /* Start main program */

fprintf(fp1,"Enter the count of Address Generator for Samples Please");
fprintf(fp1,"\n");
fprintf(fp1, "count");
fprintf(fp1, "%2d", count);
fprintf(fp1,"\n");


                         /* Describe data flow */
for (k=1;k<=count;k++)
```

```
      {
      fscanf(fp,"%d",&i);
      fscanf(fp,"%d",&q);
      fscanf(fp,"%d",&s);
      fscanf(fp,"%d",&c);  /* clock is negative */
      latch[1]=add;
      latch[2]=sub;
      add=mult[1]+mult[2];
      sub=mult[3]-mult[4];
      mult[1]=i*c;
      mult[2]=q*s;
      mult[3]=q*c;
      mult[4]=i*s;
                           /* clock is positive */
    if (we==1)
      {
      DDR.location_I[d]=I_inv;
      DDR.location_Q[d]=Q_inv;
      d=d+1;
      if (d>8) d=d-8;
      }
      I_inv=latch[1];
      Q_inv=latch[2];
                           /* Print output of MDRM units */

    fprintf(fp1,"%3s %3s %3s %3s %3s %3s %3s %3s %3s %3s %3s %3s\n",
    "AGS", "AGC","MU1","MU2","MU3","MU4","ADD","SUB","LA1","LA2","DDI","DDQ");
    for (j=1;j<9;j++)
        {
        fprintf(fp1,"%2d",l);
        fprintf(fp1,"%4d",agc);
        fprintf(fp1,"%4d",mult[1]);
        fprintf(fp1,"%4d",mult[2]);
        fprintf(fp1,"%4d",mult[3]);
        fprintf(fp1,"%4d",mult[4]);
        fprintf(fp1,"%4d",add);
        fprintf(fp1,"%4d",sub);
        fprintf(fp1,"%4d",latch[1]);
        fprintf(fp1,"%4d",latch[2]);
        fprintf(fp1,"%4d",DDR.location_I[j]);
        fprintf(fp1,"%4d",DDR.location_Q[j]);
        fprintf(fp1,"\n");
        }
      fprintf(fp1,"\n");
      l=l+1;
      if (l==4) we=1;
      if (l>8)
      {
        l=l-8;
        agc=agc+1;
        if (agc>3) agc=agc-4;
      printf("\n");
      }
      }
```

```
fclose(fp);
fclose(fp1);
}
```

INPUT FILE FOR MDRM

```
1 2 3 4 5 6 7 8 9 10 11 12
1 2 3 4 5 6 7 8 9 10 11 12
1 2 3 4 5 6 7 8 9 10 11 12
1 2 3 4 5 6 7 8 9 10 11 12
1 2 3 4 5 6 7 8 9 10 11 12
1 2 3 4 5 6 7 8 9 10 11 12
1 2 3 4 5 6 7 8 9 10 11 12
1 2 3 4 5 6 7 8 9 10 11 12
1 2 3 4 5 6 7 8 9 10 11 12
```

OUTPUT FILE OF MDRM

```
Enter the count of Address Generator for Samples Please
count20
AGS AGC MU1 MU2 MU3 MU4 ADD SUB LA1 LA2 DDI DDQ
 1   0   4   6   8   3   0   0   0   0   0   0
 1   0   4   6   8   3   0   0   0   0   0   0
 1   0   4   6   8   3   0   0   0   0   0   0
 1   0   4   6   8   3   0   0   0   0   0   0
 1   0   4   6   8   3   0   0   0   0   0   0
 1   0   4   6   8   3   0   0   0   0   0   0
 1   0   4   6   8   3   0   0   0   0   0   0
 1   0   4   6   8   3   0   0   0   0   0   0

AGS AGC MU1 MU2 MU3 MU4 ADD SUB LA1 LA2 DDI DDQ
 2   0  40  42  48  35  10   5   0   0   0   0
 2   0  40  42  48  35  10   5   0   0   0   0
 2   0  40  42  48  35  10   5   0   0   0   0
 2   0  40  42  48  35  10   5   0   0   0   0
 2   0  40  42  48  35  10   5   0   0   0   0
 2   0  40  42  48  35  10   5   0   0   0   0
 2   0  40  42  48  35  10   5   0   0   0   0
 2   0  40  42  48  35  10   5   0   0   0   0

AGS AGC MU1 MU2 MU3 MU4 ADD SUB LA1 LA2 DDI DDQ
 3   0 108 110 120  99  82  13  10   5   0   0
 3   0 108 110 120  99  82  13  10   5   0   0
 3   0 108 110 120  99  82  13  10   5   0   0
 3   0 108 110 120  99  82  13  10   5   0   0
 3   0 108 110 120  99  82  13  10   5   0   0
 3   0 108 110 120  99  82  13  10   5   0   0
 3   0 108 110 120  99  82  13  10   5   0   0
 3   0 108 110 120  99  82  13  10   5   0   0

AGS AGC MU1 MU2 MU3 MU4 ADD SUB LA1 LA2 DDI DDQ
 4   0   4   6   8   3 218  21  82  13  10   5
 4   0   4   6   8   3 218  21  82  13   0   0
 4   0   4   6   8   3 218  21  82  13   0   0
 4   0   4   6   8   3 218  21  82  13   0   0
 4   0   4   6   8   3 218  21  82  13   0   0
 4   0   4   6   8   3 218  21  82  13   0   0
 4   0   4   6   8   3 218  21  82  13   0   0
 4   0   4   6   8   3 218  21  82  13   0   0

AGS AGC MU1 MU2 MU3 MU4 ADD SUB LA1 LA2 DDI DDQ
 5   0  40  42  48  35  10   5 218  21  10   5
 5   0  40  42  48  35  10   5 218  21  82  13
 5   0  40  42  48  35  10   5 218  21   0   0
 5   0  40  42  48  35  10   5 218  21   0   0
 5   0  40  42  48  35  10   5 218  21   0   0
 5   0  40  42  48  35  10   5 218  21   0   0
 5   0  40  42  48  35  10   5 218  21   0   0
 5   0  40  42  48  35  10   5 218  21   0   0

AGS AGC MU1 MU2 MU3 MU4 ADD SUB LA1 LA2 DDI DDQ
 6   0 108 110 120  99  82  13  10   5  10   5
```

6	0	108	110	120	99	82	13	10	5	82	13
6	0	108	110	120	99	82	13	10	5	218	21
6	0	108	110	120	99	82	13	10	5	0	0
6	0	108	110	120	99	82	13	10	5	0	0
6	0	108	110	120	99	82	13	10	5	0	0
6	0	108	110	120	99	82	13	10	5	0	0
6	0	108	110	120	99	82	13	10	5	0	0

AGS	AGC	MU1	MU2	MU3	MU4	ADD	SUB	LA1	LA2	DDI	DDQ
7	0	4	6	8	3	218	21	82	13	10	5
7	0	4	6	8	3	218	21	82	13	82	13
7	0	4	6	8	3	218	21	82	13	218	21
7	0	4	6	8	3	218	21	82	13	10	5
7	0	4	6	8	3	218	21	82	13	0	0
7	0	4	6	8	3	218	21	82	13	0	0
7	0	4	6	8	3	218	21	82	13	0	0
7	0	4	6	8	3	218	21	82	13	0	0

AGS	AGC	MU1	MU2	MU3	MU4	ADD	SUB	LA1	LA2	DDI	DDQ
8	0	40	42	48	35	10	5	218	21	10	5
8	0	40	42	48	35	10	5	218	21	82	13
8	0	40	42	48	35	10	5	218	21	218	21
8	0	40	42	48	35	10	5	218	21	10	5
8	0	40	42	48	35	10	5	218	21	82	13
8	0	40	42	48	35	10	5	218	21	0	0
8	0	40	42	48	35	10	5	218	21	0	0
8	0	40	42	48	35	10	5	218	21	0	0

AGS	AGC	MU1	MU2	MU3	MU4	ADD	SUB	LA1	LA2	DDI	DDQ
1	1	108	110	120	99	82	13	10	5	10	5
1	1	108	110	120	99	82	13	10	5	82	13
1	1	108	110	120	99	82	13	10	5	218	21
1	1	108	110	120	99	82	13	10	5	10	5
1	1	108	110	120	99	82	13	10	5	82	13
1	1	108	110	120	99	82	13	10	5	218	21
1	1	108	110	120	99	82	13	10	5	0	0
1	1	108	110	120	99	82	13	10	5	0	0

AGS	AGC	MU1	MU2	MU3	MU4	ADD	SUB	LA1	LA2	DDI	DDQ
2	1	4	6	8	3	218	21	82	13	10	5
2	1	4	6	8	3	218	21	82	13	82	13
2	1	4	6	8	3	218	21	82	13	218	21
2	1	4	6	8	3	218	21	82	13	10	5
2	1	4	6	8	3	218	21	82	13	82	13
2	1	4	6	8	3	218	21	82	13	218	21
2	1	4	6	8	3	218	21	82	13	10	5
2	1	4	6	8	3	218	21	82	13	0	0

AGS	AGC	MU1	MU2	MU3	MU4	ADD	SUB	LA1	LA2	DDI	DDQ
3	1	40	42	48	35	10	5	218	21	10	5
3	1	40	42	48	35	10	5	218	21	82	13
3	1	40	42	48	35	10	5	218	21	218	21
3	1	40	42	48	35	10	5	218	21	10	5
3	1	40	42	48	35	10	5	218	21	82	13

3	1	40	42	48	35	10	5	218	21	218	21
3	1	40	42	48	35	10	5	218	21	10	5
3	1	40	42	48	35	10	5	218	21	82	13

AGS	AGC	MU1	MU2	MU3	MU4	ADD	SUB	LA1	LA2	DDI	DDQ
4	1	108	110	120	99	82	13	10	5	218	21
4	1	108	110	120	99	82	13	10	5	82	13
4	1	108	110	120	99	82	13	10	5	218	21
4	1	108	110	120	99	82	13	10	5	10	5
4	1	108	110	120	99	82	13	10	5	82	13
4	1	108	110	120	99	82	13	10	5	218	21
4	1	108	110	120	99	82	13	10	5	10	5
4	1	108	110	120	99	82	13	10	5	82	13

AGS	AGC	MU1	MU2	MU3	MU4	ADD	SUB	LA1	LA2	DDI	DDQ
5	1	4	6	8	3	218	21	82	13	218	21
5	1	4	6	8	3	218	21	82	13	10	5
5	1	4	6	8	3	218	21	82	13	218	21
5	1	4	6	8	3	218	21	82	13	10	5
5	1	4	6	8	3	218	21	82	13	82	13
5	1	4	6	8	3	218	21	82	13	218	21
5	1	4	6	8	3	218	21	82	13	10	5
5	1	4	6	8	3	218	21	82	13	82	13

AGS	AGC	MU1	MU2	MU3	MU4	ADD	SUB	LA1	LA2	DDI	DDQ
6	1	40	42	48	35	10	5	218	21	218	21
6	1	40	42	48	35	10	5	218	21	10	5
6	1	40	42	48	35	10	5	218	21	82	13
6	1	40	42	48	35	10	5	218	21	10	5
6	1	40	42	48	35	10	5	218	21	82	13
6	1	40	42	48	35	10	5	218	21	218	21
6	1	40	42	48	35	10	5	218	21	10	5
6	1	40	42	48	35	10	5	218	21	82	13

AGS	AGC	MU1	MU2	MU3	MU4	ADD	SUB	LA1	LA2	DDI	DDQ
7	1	108	110	120	99	82	13	10	5	218	21
7	1	108	110	120	99	82	13	10	5	10	5
7	1	108	110	120	99	82	13	10	5	82	13
7	1	108	110	120	99	82	13	10	5	218	21
7	1	108	110	120	99	82	13	10	5	82	13
7	1	108	110	120	99	82	13	10	5	218	21
7	1	108	110	120	99	82	13	10	5	10	5
7	1	108	110	120	99	82	13	10	5	82	13

AGS	AGC	MU1	MU2	MU3	MU4	ADD	SUB	LA1	LA2	DDI	DDQ
8	1	4	6	8	3	218	21	82	13	218	21
8	1	4	6	8	3	218	21	82	13	10	5
8	1	4	6	8	3	218	21	82	13	82	13
8	1	4	6	8	3	218	21	82	13	218	21
8	1	4	6	8	3	218	21	82	13	10	5
8	1	4	6	8	3	218	21	82	13	218	21
8	1	4	6	8	3	218	21	82	13	10	5
8	1	4	6	8	3	218	21	82	13	82	13

AGS	AGC	MU1	MU2	MU3	MU4	ADD	SUB	LA1	LA2	DDI	DDQ
1	2	40	42	48	35	10	5	218	21	218	21
1	2	40	42	48	35	10	5	218	21	10	5
1	2	40	42	48	35	10	5	218	21	82	13
1	2	40	42	48	35	10	5	218	21	218	21
1	2	40	42	48	35	10	5	218	21	10	5
1	2	40	42	48	35	10	5	218	21	82	13
1	2	40	42	48	35	10	5	218	21	10	5
1	2	40	42	48	35	10	5	218	21	82	13

AGS	AGC	MU1	MU2	MU3	MU4	ADD	SUB	LA1	LA2	DDI	DDQ
2	2	108	110	120	99	82	13	10	5	218	21
2	2	108	110	120	99	82	13	10	5	10	5
2	2	108	110	120	99	82	13	10	5	82	13
2	2	108	110	120	99	82	13	10	5	218	21
2	2	108	110	120	99	82	13	10	5	10	5
2	2	108	110	120	99	82	13	10	5	82	13
2	2	108	110	120	99	82	13	10	5	218	21
2	2	108	110	120	99	82	13	10	5	82	13

AGS	AGC	MU1	MU2	MU3	MU4	ADD	SUB	LA1	LA2	DDI	DDQ
3	2	4	6	8	3	218	21	82	13	218	21
3	2	4	6	8	3	218	21	82	13	10	5
3	2	4	6	8	3	218	21	82	13	82	13
3	2	4	6	8	3	218	21	82	13	218	21
3	2	4	6	8	3	218	21	82	13	10	5
3	2	4	6	8	3	218	21	82	13	82	13
3	2	4	6	8	3	218	21	82	13	218	21
3	2	4	6	8	3	218	21	82	13	10	5

AGS	AGC	MU1	MU2	MU3	MU4	ADD	SUB	LA1	LA2	DDI	DDQ
4	2	40	42	48	35	10	5	218	21	82	13
4	2	40	42	48	35	10	5	218	21	10	5
4	2	40	42	48	35	10	5	218	21	82	13
4	2	40	42	48	35	10	5	218	21	218	21
4	2	40	42	48	35	10	5	218	21	10	5
4	2	40	42	48	35	10	5	218	21	82	13
4	2	40	42	48	35	10	5	218	21	218	21
4	2	40	42	48	35	10	5	218	21	10	5

Appendix E

PROGRAMS IN C AND SIMULATION RESULTS FOR DATA FLOW OF SAMPLES IN THE HARDWARE DESIGN OF MTRM

```
/* Hardware behavioural simulation for Multiplexed
Timing Recovery Module (MTRM) */

/* This program describes the data flow in the MTRM for
8 channel & 4 samples used for periodic estimation */

#include <stdio.h>

/* l    is Address Generator for Samples
   agc  is Address Generator for Channels
   te   is enable for Timing RAM */

                                /* Define memory for the RAMs */

struct rambuffer
        {
        int latch;
        int location[9];
        };
struct rambuffer ram_i[4],ram_q[4];

struct storage
     {
     int location[9];
     };
struct storage Timing_RAM;

main()
{
                                /* Start main program */

int LI4=0,LQ4=0,isub=0,qsub=0,add=0,adder=0,imult=0,qmult=0;
int temp=0,idumLatch=0,qdumLatch=0;
int l=1,d=1,dd=0;
int h,p,count,m,j,agc=0;
int te=1,le=0;
int dummy=0;
int dum1=0;
int del1,del2,del3;
FILE *fp,*fp1;
int I_s,Q_s,i,y;

printf("Enter the AGS please");
printf("\n");
scanf("%d",&count);
fp=fopen("tim1.dat","r");
fp1=fopen("outtim.dat","w");
for(y=1;y<9;y++)
   {
   ram_i[3].location[y]=0;
   ram_i[3].latch=0;
   ram_q[1].location[l]=0;
   ram_q[1].latch=0;
   }
fprintf(fp1,"Enter the AGS please");
fprintf(fp1,"\n");
fprintf(fp1, "Count");
fprintf(fp1,"\t");
fprintf(fp1, "%2d", count);
fprintf(fp1,"\n");



for (i=1;i<=count;i++)
   {                              /* Input from MDRM */
   fscanf(fp,"%d",&I_s);
   fscanf(fp,"%d",&Q_s);

                        /* Data flow in the RAM latch design */
   ram_i[1].latch=I_s;
   ram_q[1].latch=Q_s;
   for (m=2;m<4;m++)
      {
      ram_i[m].latch=ram_i[m-1].location[l];
      }
      LI4=ram_i[3].location[l];
      for (m=2;m<4;m++)
         {
         ram_q[m].latch=ram_q[m-1].location[l];
         }
      LQ4=ram_q[3].location[l];
      adder=add + Timing_RAM.location[d];
                                  /* Store data in TR */
      add=imult+qmult;
      imult=isub*idumLatch;
      qmult=qsub*qdumLatch;

      for (m=1;m<4;m++)           /* Clock is positive */
         {
         ram_i[m].location[l]=ram_i[m].latch;
         }
      for (m=1;m<4;m++)
         {
         ram_q[m].location[l]=ram_q[m].latch;
         }
                                  /* Data flow in RAM latch */

      idumLatch=ram_i[3].latch;
      qdumLatch=ram_q[3].latch;
      isub=ram_i[2].latch-LI4;
      qsub=ram_q[2].latch-LQ4;
                                  /* Control circuitry */
       if (dd==1)
         {
         if(l==4) le=1;
         }
      if (le==1)
      {
      if (te==1)
         {
         Timing_RAM.location[d]=adder;
         d=d+1;
         if (d>8)
            {
            d=(d-8);
            te=2;
            }
         }

      if (te==2)
         {
         d=d+1;
         if (d>8)
            {
            d=(d-8);
            te=1;
            }
         }
      }
                                        /* Print data flow in the MTRM */

  for (h=1;h<2;h++)
    {
    fprintf(fp1,
    "%3s %3s %3s %3s %3s %3s %3s %3s %3s %3s %3s %3s %3s %3s %3s\n",
    "AGS", "AGC","LI3","RI3", "LI4","ISB","IML",
    "LQ1","RQ1","LQ4","QSB","QML","ADD","AER","TIR");

       for (j=1;j<9;j++)
          {
          fprintf(fp1,"%2d",l);
          fprintf(fp1,"%4d",agc);
          fprintf(fp1,"%4d",ram_i[3].latch);
          fprintf(fp1,"%4d",ram_i[3].location[j]);
          fprintf(fp1, "%4d",LI4);
          fprintf(fp1, "%4d",isub);
          fprintf(fp1,"%4d",imult);
          fprintf(fp1,"%4d",ram_q[1].latch);
          fprintf(fp1,"%4d",ram_q[1].location[j]);
          fprintf(fp1,"%4d",LQ4);
          fprintf(fp1,"%4d",qsub);
          fprintf(fp1,"%4d",qmult);
          fprintf(fp1,"%4d",add);
          fprintf(fp1,"%4d",adder);
          fprintf(fp1,"%4d",Timing_RAM.location[j]);
          fprintf(fp1,"\n");
          }
                                        /* Define Control Logic */

       l=l+1;
```

```
        if (l>8)
           {
           l=l-8;
           agc=agc+1;
           if (agc==3) dd=1;
           if (agc>3) agc=agc-4;
           }
           printf("\n");
        }

    }
fclose(fp);
fclose(fp1);
}
```

INPUT FILE FOR MTRM

```
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
```

OUTPUT FILE OF MTRM

```
Enter the AGS please
Count   45
AGS AGC LI3 RI3 LI4 ISB IML LQ1 RQ1 LQ4 QSB QML ADD AER TIR
 1   0   0   0   0   0   0   2   2   0   0   0   0   0   0
 1   0   0   0   0   0   0   2   0   0   0   0   0   0   0
 1   0   0   0   0   0   0   2   0   0   0   0   0   0   0
 1   0   0   0   0   0   0   2   0   0   0   0   0   0   0
 1   0   0   0   0   0   0   2   0   0   0   0   0   0   0
 1   0   0   0   0   0   0   2   0   0   0   0   0   0   0
 1   0   0   0   0   0   0   2   0   0   0   0   0   0   0
 1   0   0   0   0   0   0   2   0   0   0   0   0   0   0
AGS AGC LI3 RI3 LI4 ISB IML LQ1 RQ1 LQ4 QSB QML ADD AER TIR
 2   0   0   0   0   0   0   1   2   0   0   0   0   0   0
 2   0   0   0   0   0   0   1   1   0   0   0   0   0   0
 2   0   0   0   0   0   0   1   0   0   0   0   0   0   0
 2   0   0   0   0   0   0   1   0   0   0   0   0   0   0
 2   0   0   0   0   0   0   1   0   0   0   0   0   0   0
 2   0   0   0   0   0   0   1   0   0   0   0   0   0   0
 2   0   0   0   0   0   0   1   0   0   0   0   0   0   0
 2   0   0   0   0   0   0   1   0   0   0   0   0   0   0
AGS AGC LI3 RI3 LI4 ISB IML LQ1 RQ1 LQ4 QSB QML ADD AER TIR
 3   0   0   0   0   0   0   3   2   0   0   0   0   0   0
 3   0   0   0   0   0   0   3   1   0   0   0   0   0   0
 3   0   0   0   0   0   0   3   3   0   0   0   0   0   0
 3   0   0   0   0   0   0   3   0   0   0   0   0   0   0
 3   0   0   0   0   0   0   3   0   0   0   0   0   0   0
 3   0   0   0   0   0   0   3   0   0   0   0   0   0   0
 3   0   0   0   0   0   0   3   0   0   0   0   0   0   0
 3   0   0   0   0   0   0   3   0   0   0   0   0   0   0
AGS AGC LI3 RI3 LI4 ISB IML LQ1 RQ1 LQ4 QSB QML ADD AER TIR
 4   0   0   0   0   0   0   2   2   0   0   0   0   0   0
 4   0   0   0   0   0   0   2   1   0   0   0   0   0   0
 4   0   0   0   0   0   0   2   3   0   0   0   0   0   0
 4   0   0   0   0   0   0   2   2   0   0   0   0   0   0
 4   0   0   0   0   0   0   2   0   0   0   0   0   0   0
 4   0   0   0   0   0   0   2   0   0   0   0   0   0   0
 4   0   0   0   0   0   0   2   0   0   0   0   0   0   0
 4   0   0   0   0   0   0   2   0   0   0   0   0   0   0
AGS AGC LI3 RI3 LI4 ISB IML LQ1 RQ1 LQ4 QSB QML ADD AER TIR
 5   0   0   0   0   0   0   1   2   0   0   0   0   0   0
 5   0   0   0   0   0   0   1   1   0   0   0   0   0   0
 5   0   0   0   0   0   0   1   3   0   0   0   0   0   0
 5   0   0   0   0   0   0   1   2   0   0   0   0   0   0
 5   0   0   0   0   0   0   1   1   0   0   0   0   0   0
 5   0   0   0   0   0   0   1   0   0   0   0   0   0   0
 5   0   0   0   0   0   0   1   0   0   0   0   0   0   0
 5   0   0   0   0   0   0   1   0   0   0   0   0   0   0
AGS AGC LI3 RI3 LI4 ISB IML LQ1 RQ1 LQ4 QSB QML ADD AER TIR
 6   0   0   0   0   0   0   3   2   0   0   0   0   0   0
 6   0   0   0   0   0   0   3   1   0   0   0   0   0   0
 6   0   0   0   0   0   0   3   3   0   0   0   0   0   0
 6   0   0   0   0   0   0   3   2   0   0   0   0   0   0
 6   0   0   0   0   0   0   3   1   0   0   0   0   0   0
 6   0   0   0   0   0   0   3   3   0   0   0   0   0   0
```

6	0	0	0	0	0	0	3	0	0	0	0	0	0	0
6	0	0	0	0	0	0	3	0	0	0	0	0	0	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
7	0	0	0	0	0	0	2	2	0	0	0	0	0	0
7	0	0	0	0	0	0	2	1	0	0	0	0	0	0
7	0	0	0	0	0	0	2	3	0	0	0	0	0	0
7	0	0	0	0	0	0	2	2	0	0	0	0	0	0
7	0	0	0	0	0	0	2	1	0	0	0	0	0	0
7	0	0	0	0	0	0	2	3	0	0	0	0	0	0
7	0	0	0	0	0	0	2	2	0	0	0	0	0	0
7	0	0	0	0	0	0	2	0	0	0	0	0	0	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
8	0	0	0	0	0	0	1	2	0	0	0	0	0	0
8	0	0	0	0	0	0	1	1	0	0	0	0	0	0
8	0	0	0	0	0	0	1	3	0	0	0	0	0	0
8	0	0	0	0	0	0	1	2	0	0	0	0	0	0
8	0	0	0	0	0	0	1	1	0	0	0	0	0	0
8	0	0	0	0	0	0	1	3	0	0	0	0	0	0
8	0	0	0	0	0	0	1	2	0	0	0	0	0	0
8	0	0	0	0	0	0	1	1	0	0	0	0	0	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
1	1	0	0	0	1	0	3	3	0	2	0	0	0	0
1	1	0	0	0	1	0	3	1	0	2	0	0	0	0
1	1	0	0	0	1	0	3	3	0	2	0	0	0	0
1	1	0	0	0	1	0	3	2	0	2	0	0	0	0
1	1	0	0	0	1	0	3	1	0	2	0	0	0	0
1	1	0	0	0	1	0	3	3	0	2	0	0	0	0
1	1	0	0	0	1	0	3	2	0	2	0	0	0	0
1	1	0	0	0	1	0	3	1	0	2	0	0	0	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
2	1	0	0	0	3	0	2	3	0	1	0	0	0	0
2	1	0	0	0	3	0	2	2	0	1	0	0	0	0
2	1	0	0	0	3	0	2	3	0	1	0	0	0	0
2	1	0	0	0	3	0	2	2	0	1	0	0	0	0
2	1	0	0	0	3	0	2	1	0	1	0	0	0	0
2	1	0	0	0	3	0	2	3	0	1	0	0	0	0
2	1	0	0	0	3	0	2	2	0	1	0	0	0	0
2	1	0	0	0	3	0	2	1	0	1	0	0	0	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
3	1	0	0	0	2	0	1	3	0	3	0	0	0	0
3	1	0	0	0	2	0	1	2	0	3	0	0	0	0
3	1	0	0	0	2	0	1	1	0	3	0	0	0	0
3	1	0	0	0	2	0	1	2	0	3	0	0	0	0
3	1	0	0	0	2	0	1	1	0	3	0	0	0	0
3	1	0	0	0	2	0	1	3	0	3	0	0	0	0
3	1	0	0	0	2	0	1	2	0	3	0	0	0	0
3	1	0	0	0	2	0	1	1	0	3	0	0	0	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
4	1	0	0	0	1	0	3	3	0	2	0	0	0	0
4	1	0	0	0	1	0	3	2	0	2	0	0	0	0
4	1	0	0	0	1	0	3	1	0	2	0	0	0	0
4	1	0	0	0	1	0	3	3	0	2	0	0	0	0
4	1	0	0	0	1	0	3	1	0	2	0	0	0	0
4	1	0	0	0	1	0	3	3	0	2	0	0	0	0

4	1	0	0	0	1	0	3	2	0	2	0	0	0	0
4	1	0	0	0	1	0	3	1	0	2	0	0	0	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
5	1	0	0	0	3	0	2	3	0	1	0	0	0	0
5	1	0	0	0	3	0	2	2	0	1	0	0	0	0
5	1	0	0	0	3	0	2	1	0	1	0	0	0	0
5	1	0	0	0	3	0	2	3	0	1	0	0	0	0
5	1	0	0	0	3	0	2	2	0	1	0	0	0	0
5	1	0	0	0	3	0	2	3	0	1	0	0	0	0
5	1	0	0	0	3	0	2	2	0	1	0	0	0	0
5	1	0	0	0	3	0	2	1	0	1	0	0	0	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
6	1	0	0	0	2	0	1	3	0	3	0	0	0	0
6	1	0	0	0	2	0	1	2	0	3	0	0	0	0
6	1	0	0	0	2	0	1	1	0	3	0	0	0	0
6	1	0	0	0	2	0	1	3	0	3	0	0	0	0
6	1	0	0	0	2	0	1	2	0	3	0	0	0	0
6	1	0	0	0	2	0	1	1	0	3	0	0	0	0
6	1	0	0	0	2	0	1	2	0	3	0	0	0	0
6	1	0	0	0	2	0	1	1	0	3	0	0	0	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
7	1	0	0	0	1	0	3	3	0	2	0	0	0	0
7	1	0	0	0	1	0	3	2	0	2	0	0	0	0
7	1	0	0	0	1	0	3	1	0	2	0	0	0	0
7	1	0	0	0	1	0	3	3	0	2	0	0	0	0
7	1	0	0	0	1	0	3	2	0	2	0	0	0	0
7	1	0	0	0	1	0	3	1	0	2	0	0	0	0
7	1	0	0	0	1	0	3	3	0	2	0	0	0	0
7	1	0	0	0	1	0	3	1	0	2	0	0	0	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
8	1	0	0	0	3	0	2	3	0	1	0	0	0	0
8	1	0	0	0	3	0	2	2	0	1	0	0	0	0
8	1	0	0	0	3	0	2	1	0	1	0	0	0	0
8	1	0	0	0	3	0	2	3	0	1	0	0	0	0
8	1	0	0	0	3	0	2	2	0	1	0	0	0	0
8	1	0	0	0	3	0	2	1	0	1	0	0	0	0
8	1	0	0	0	3	0	2	3	0	1	0	0	0	0
8	1	0	0	0	3	0	2	2	0	1	0	0	0	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
1	2	1	1	0	2	0	1	1	0	3	0	0	0	0
1	2	1	0	0	2	0	1	2	0	3	0	0	0	0
1	2	1	0	0	2	0	1	1	0	3	0	0	0	0
1	2	1	0	0	2	0	1	3	0	3	0	0	0	0
1	2	1	0	0	2	0	1	2	0	3	0	0	0	0
1	2	1	0	0	2	0	1	1	0	3	0	0	0	0
1	2	1	0	0	2	0	1	3	0	3	0	0	0	0
1	2	1	0	0	2	0	1	2	0	3	0	0	0	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
2	2	3	1	0	1	2	3	1	0	2	6	0	0	0
2	2	3	3	0	1	2	3	3	0	2	6	0	0	0
2	2	3	0	0	1	2	3	1	0	2	6	0	0	0
2	2	3	0	0	1	2	3	3	0	2	6	0	0	0
2	2	3	0	0	1	2	3	2	0	2	6	0	0	0
2	2	3	0	0	1	2	3	1	0	2	6	0	0	0

2	2	3	0	0	1	2	3	3	0	2	6	0	0	0
2	2	3	0	0	1	2	3	2	0	2	6	0	0	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
3	2	2	1	0	3	3	2	1	0	1	2	8	0	0
3	2	2	3	0	3	3	2	3	0	1	2	8	0	0
3	2	2	2	0	3	3	2	2	0	1	2	8	0	0
3	2	2	0	0	3	3	2	3	0	1	2	8	0	0
3	2	2	0	0	3	3	2	2	0	1	2	8	0	0
3	2	2	0	0	3	3	2	1	0	1	2	8	0	0
3	2	2	0	0	3	3	2	3	0	1	2	8	0	0
3	2	2	0	0	3	3	2	2	0	1	2	8	0	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
4	2	1	1	0	2	6	1	1	0	3	3	5	8	0
4	2	1	3	0	2	6	1	3	0	3	3	5	8	0
4	2	1	2	0	2	6	1	2	0	3	3	5	8	0
4	2	1	1	0	2	6	1	1	0	3	3	5	8	0
4	2	1	0	0	2	6	1	2	0	3	3	5	8	0
4	2	1	0	0	2	6	1	1	0	3	3	5	8	0
4	2	1	0	0	2	6	1	3	0	3	3	5	8	0
4	2	1	0	0	2	6	1	2	0	3	3	5	8	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
5	2	3	1	0	1	2	3	1	0	2	6	9	5	0
5	2	3	3	0	1	2	3	3	0	2	6	9	5	0
5	2	3	2	0	1	2	3	2	0	2	6	9	5	0
5	2	3	1	0	1	2	3	1	0	2	6	9	5	0
5	2	3	3	0	1	2	3	3	0	2	6	9	5	0
5	2	3	0	0	1	2	3	1	0	2	6	9	5	0
5	2	3	0	0	1	2	3	3	0	2	6	9	5	0
5	2	3	0	0	1	2	3	2	0	2	6	9	5	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
6	2	2	1	0	3	3	2	1	0	1	2	8	9	0
6	2	2	3	0	3	3	2	3	0	1	2	8	9	0
6	2	2	2	0	3	3	2	2	0	1	2	8	9	0
6	2	2	1	0	3	3	2	1	0	1	2	8	9	0
6	2	2	3	0	3	3	2	3	0	1	2	8	9	0
6	2	2	2	0	3	3	2	2	0	1	2	8	9	0
6	2	2	0	0	3	3	2	3	0	1	2	8	9	0
6	2	2	0	0	3	3	2	2	0	1	2	8	9	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
7	2	1	1	0	2	6	1	1	0	3	3	5	8	0
7	2	1	3	0	2	6	1	3	0	3	3	5	8	0
7	2	1	2	0	2	6	1	2	0	3	3	5	8	0
7	2	1	1	0	2	6	1	1	0	3	3	5	8	0
7	2	1	3	0	2	6	1	3	0	3	3	5	8	0
7	2	1	2	0	2	6	1	2	0	3	3	5	8	0
7	2	1	1	0	2	6	1	1	0	3	3	5	8	0
7	2	1	0	0	2	6	1	2	0	3	3	5	8	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
8	2	3	1	0	1	2	3	1	0	2	6	9	5	0
8	2	3	3	0	1	2	3	3	0	2	6	9	5	0
8	2	3	2	0	1	2	3	2	0	2	6	9	5	0
8	2	3	1	0	1	2	3	1	0	2	6	9	5	0
8	2	3	3	0	1	2	3	3	0	2	6	9	5	0
8	2	3	2	0	1	2	3	2	0	2	6	9	5	0

```
  8   2   3   1   0   1   2   3   1   0   2   6   9   5   0
  8   2   3   3   0   1   2   3   3   0   2   6   9   5   0
AGS AGC LI3 RI3 LI4 ISB IML LQ1 RQ1 LQ4 QSB QML ADD AER TIR
  1   3   2   2   1   2   3   2   2   2  -1   2   8   9   0
  1   3   2   3   1   2   3   2   3   2  -1   2   8   9   0
  1   3   2   2   1   2   3   2   2   2  -1   2   8   9   0
  1   3   2   1   1   2   3   2   1   2  -1   2   8   9   0
  1   3   2   3   1   2   3   2   3   2  -1   2   8   9   0
  1   3   2   2   1   2   3   2   2   2  -1   2   8   9   0
  1   3   2   1   1   2   3   2   1   2  -1   2   8   9   0
  1   3   2   3   1   2   3   2   3   2  -1   2   8   9   0
AGS AGC LI3 RI3 LI4 ISB IML LQ1 RQ1 LQ4 QSB QML ADD AER TIR
  2   3   1   2   3  -1   4   1   2   1   2  -3   5   8   0
  2   3   1   1   3  -1   4   1   1   1   2  -3   5   8   0
  2   3   1   2   3  -1   4   1   2   1   2  -3   5   8   0
  2   3   1   1   3  -1   4   1   1   1   2  -3   5   8   0
  2   3   1   3   3  -1   4   1   3   1   2  -3   5   8   0
  2   3   1   2   3  -1   4   1   2   1   2  -3   5   8   0
  2   3   1   1   3  -1   4   1   1   1   2  -3   5   8   0
  2   3   1   3   3  -1   4   1   3   1   2  -3   5   8   0
AGS AGC LI3 RI3 LI4 ISB IML LQ1 RQ1 LQ4 QSB QML ADD AER TIR
  3   3   3   2   2  -1  -1   3   2   3  -1   4   1   5   0
  3   3   3   1   2  -1  -1   3   1   3  -1   4   1   5   0
  3   3   3   3   2  -1  -1   3   3   3  -1   4   1   5   0
  3   3   3   1   2  -1  -1   3   1   3  -1   4   1   5   0
  3   3   3   3   2  -1  -1   3   3   3  -1   4   1   5   0
  3   3   3   2   2  -1  -1   3   2   3  -1   4   1   5   0
  3   3   3   1   2  -1  -1   3   1   3  -1   4   1   5   0
  3   3   3   3   2  -1  -1   3   3   3  -1   4   1   5   0
AGS AGC LI3 RI3 LI4 ISB IML LQ1 RQ1 LQ4 QSB QML ADD AER TIR
  4   3   2   2   1   2  -3   2   2   2  -1  -1   3   1   1
  4   3   2   1   1   2  -3   2   1   2  -1  -1   3   1   0
  4   3   2   3   1   2  -3   2   3   2  -1  -1   3   1   0
  4   3   2   2   1   2  -3   2   2   2  -1  -1   3   1   0
  4   3   2   3   1   2  -3   2   3   2  -1  -1   3   1   0
  4   3   2   2   1   2  -3   2   2   2  -1  -1   3   1   0
  4   3   2   1   1   2  -3   2   1   2  -1  -1   3   1   0
  4   3   2   3   1   2  -3   2   3   2  -1  -1   3   1   0
AGS AGC LI3 RI3 LI4 ISB IML LQ1 RQ1 LQ4 QSB QML ADD AER TIR
  5   3   1   2   3  -1   4   1   2   1   2  -3  -4   3   1
  5   3   1   1   3  -1   4   1   1   1   2  -3  -4   3   3
  5   3   1   3   3  -1   4   1   3   1   2  -3  -4   3   0
  5   3   1   2   3  -1   4   1   2   1   2  -3  -4   3   0
  5   3   1   1   3  -1   4   1   1   1   2  -3  -4   3   0
  5   3   1   2   3  -1   4   1   2   1   2  -3  -4   3   0
  5   3   1   1   3  -1   4   1   1   1   2  -3  -4   3   0
  5   3   1   3   3  -1   4   1   3   1   2  -3  -4   3   0
AGS AGC LI3 RI3 LI4 ISB IML LQ1 RQ1 LQ4 QSB QML ADD AER TIR
  6   3   3   2   2  -1  -1   3   2   3  -1   4   1  -4   1
  6   3   3   1   2  -1  -1   3   1   3  -1   4   1  -4   3
  6   3   3   3   2  -1  -1   3   3   3  -1   4   1  -4  -4
  6   3   3   2   2  -1  -1   3   2   3  -1   4   1  -4   0
  6   3   3   1   2  -1  -1   3   1   3  -1   4   1  -4   0
  6   3   3   3   2  -1  -1   3   3   3  -1   4   1  -4   0
```

6	3	3	1	2	-1	-1	3	1	3	-1	4	1	-4	0
6	3	3	3	2	-1	-1	3	3	3	-1	4	1	-4	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
7	3	2	2	1	2	-3	2	2	2	-1	-1	3	1	1
7	3	2	1	1	2	-3	2	1	2	-1	-1	3	1	3
7	3	2	3	1	2	-3	2	3	2	-1	-1	3	1	-4
7	3	2	2	1	2	-3	2	2	2	-1	-1	3	1	1
7	3	2	1	1	2	-3	2	1	2	-1	-1	3	1	0
7	3	2	3	1	2	-3	2	3	2	-1	-1	3	1	0
7	3	2	2	1	2	-3	2	2	2	-1	-1	3	1	0
7	3	2	3	1	2	-3	2	3	2	-1	-1	3	1	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
8	3	1	2	3	-1	4	1	2	1	2	-3	-4	3	1
8	3	1	1	3	-1	4	1	1	1	2	-3	-4	3	3
8	3	1	3	3	-1	4	1	3	1	2	-3	-4	3	-4
8	3	1	2	3	-1	4	1	2	1	2	-3	-4	3	1
8	3	1	1	3	-1	4	1	1	1	2	-3	-4	3	3
8	3	1	3	3	-1	4	1	3	1	2	-3	-4	3	0
8	3	1	2	3	-1	4	1	2	1	2	-3	-4	3	0
8	3	1	1	3	-1	4	1	1	1	2	-3	-4	3	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
1	0	3	3	2	-1	-1	3	3	3	-1	4	1	-4	1
1	0	3	1	2	-1	-1	3	1	3	-1	4	1	-4	3
1	0	3	3	2	-1	-1	3	3	3	-1	4	1	-4	-4
1	0	3	2	2	-1	-1	3	2	3	-1	4	1	-4	1
1	0	3	1	2	-1	-1	3	1	3	-1	4	1	-4	3
1	0	3	3	2	-1	-1	3	3	3	-1	4	1	-4	-4
1	0	3	2	2	-1	-1	3	2	3	-1	4	1	-4	0
1	0	3	1	2	-1	-1	3	1	3	-1	4	1	-4	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
2	0	2	3	1	2	-3	2	3	2	-1	-1	3	1	1
2	0	2	2	1	2	-3	2	2	2	-1	-1	3	1	3
2	0	2	3	1	2	-3	2	3	2	-1	-1	3	1	-4
2	0	2	2	1	2	-3	2	2	2	-1	-1	3	1	1
2	0	2	1	1	2	-3	2	1	2	-1	-1	3	1	3
2	0	2	3	1	2	-3	2	3	2	-1	-1	3	1	-4
2	0	2	2	1	2	-3	2	2	2	-1	-1	3	1	1
2	0	2	1	1	2	-3	2	1	2	-1	-1	3	1	0
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
3	0	1	3	3	-1	4	1	3	1	2	-3	-4	3	1
3	0	1	2	3	-1	4	1	2	1	2	-3	-4	3	3
3	0	1	1	3	-1	4	1	1	1	2	-3	-4	3	-4
3	0	1	2	3	-1	4	1	2	1	2	-3	-4	3	1
3	0	1	1	3	-1	4	1	1	1	2	-3	-4	3	3
3	0	1	3	3	-1	4	1	3	1	2	-3	-4	3	-4
3	0	1	2	3	-1	4	1	2	1	2	-3	-4	3	1
3	0	1	1	3	-1	4	1	1	1	2	-3	-4	3	3
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
4	0	3	3	2	-1	-1	3	3	3	-1	4	1	-1	1
4	0	3	2	2	-1	-1	3	2	3	-1	4	1	-1	3
4	0	3	1	2	-1	-1	3	1	3	-1	4	1	-1	-4
4	0	3	3	2	-1	-1	3	3	3	-1	4	1	-1	1
4	0	3	1	2	-1	-1	3	1	3	-1	4	1	-1	3
4	0	3	3	2	-1	-1	3	3	3	-1	4	1	-1	-4

4	0	3	2	2	-1	-1	3	2	3	-1	4	1	-1	1
4	0	3	1	2	-1	-1	3	1	3	-1	4	1	-1	3
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
5	0	2	3	1	2	-3	2	3	2	-1	-1	3	-3	1
5	0	2	2	1	2	-3	2	2	2	-1	-1	3	-3	3
5	0	2	1	1	2	-3	2	1	2	-1	-1	3	-3	-4
5	0	2	3	1	2	-3	2	3	2	-1	-1	3	-3	1
5	0	2	2	1	2	-3	2	2	2	-1	-1	3	-3	3
5	0	2	3	1	2	-3	2	3	2	-1	-1	3	-3	-4
5	0	2	2	1	2	-3	2	2	2	-1	-1	3	-3	1
5	0	2	1	1	2	-3	2	1	2	-1	-1	3	-3	3
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
6	0	1	3	3	-1	4	1	3	1	2	-3	-4	4	1
6	0	1	2	3	-1	4	1	2	1	2	-3	-4	4	3
6	0	1	1	3	-1	4	1	1	1	2	-3	-4	4	-4
6	0	1	3	3	-1	4	1	3	1	2	-3	-4	4	1
6	0	1	2	3	-1	4	1	2	1	2	-3	-4	4	3
6	0	1	1	3	-1	4	1	1	1	2	-3	-4	4	-4
6	0	1	2	3	-1	4	1	2	1	2	-3	-4	4	1
6	0	1	1	3	-1	4	1	1	1	2	-3	-4	4	3
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
7	0	3	3	2	-1	-1	3	3	3	-1	4	1	-1	1
7	0	3	2	2	-1	-1	3	2	3	-1	4	1	-1	3
7	0	3	1	2	-1	-1	3	1	3	-1	4	1	-1	-4
7	0	3	3	2	-1	-1	3	3	3	-1	4	1	-1	1
7	0	3	2	2	-1	-1	3	2	3	-1	4	1	-1	3
7	0	3	1	2	-1	-1	3	1	3	-1	4	1	-1	-4
7	0	3	3	2	-1	-1	3	3	3	-1	4	1	-1	1
7	0	3	1	2	-1	-1	3	1	3	-1	4	1	-1	3
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
8	0	2	3	1	2	-3	2	3	2	-1	-1	3	-3	1
8	0	2	2	1	2	-3	2	2	2	-1	-1	3	-3	3
8	0	2	1	1	2	-3	2	1	2	-1	-1	3	-3	-4
8	0	2	3	1	2	-3	2	3	2	-1	-1	3	-3	1
8	0	2	2	1	2	-3	2	2	2	-1	-1	3	-3	3
8	0	2	1	1	2	-3	2	1	2	-1	-1	3	-3	-4
8	0	2	3	1	2	-3	2	3	2	-1	-1	3	-3	1
8	0	2	2	1	2	-3	2	2	2	-1	-1	3	-3	3
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
1	1	1	1	3	-1	4	1	1	1	2	-3	-4	4	1
1	1	1	2	3	-1	4	1	2	1	2	-3	-4	4	3
1	1	1	1	3	-1	4	1	1	1	2	-3	-4	4	-4
1	1	1	3	3	-1	4	1	3	1	2	-3	-4	4	1
1	1	1	2	3	-1	4	1	2	1	2	-3	-4	4	3
1	1	1	1	3	-1	4	1	1	1	2	-3	-4	4	-4
1	1	1	3	3	-1	4	1	3	1	2	-3	-4	4	1
1	1	1	2	3	-1	4	1	2	1	2	-3	-4	4	3
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
2	1	3	1	2	-1	-1	3	1	3	-1	4	1	-1	1
2	1	3	3	2	-1	-1	3	3	3	-1	4	1	-1	3
2	1	3	1	2	-1	-1	3	1	3	-1	4	1	-1	-4
2	1	3	3	2	-1	-1	3	3	3	-1	4	1	-1	1
2	1	3	2	2	-1	-1	3	2	3	-1	4	1	-1	3
2	1	3	1	2	-1	-1	3	1	3	-1	4	1	-1	-4

2	1	3	3	2	-1	-1	3	3	3	-1	4	1	-1	1
2	1	3	2	2	-1	-1	3	2	3	-1	4	1	-1	3
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
3	1	2	1	1	2	-3	2	1	2	-1	-1	3	2	2
3	1	2	3	1	2	-3	2	3	2	-1	-1	3	2	3
3	1	2	2	1	2	-3	2	2	2	-1	-1	3	2	-4
3	1	2	3	1	2	-3	2	3	2	-1	-1	3	2	1
3	1	2	2	1	2	-3	2	2	2	-1	-1	3	2	3
3	1	2	1	1	2	-3	2	1	2	-1	-1	3	2	-4
3	1	2	3	1	2	-3	2	3	2	-1	-1	3	2	1
3	1	2	2	1	2	-3	2	2	2	-1	-1	3	2	3
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
4	1	1	1	3	-1	4	1	1	1	2	-3	-4	6	2
4	1	1	3	3	-1	4	1	3	1	2	-3	-4	6	6
4	1	1	2	3	-1	4	1	2	1	2	-3	-4	6	-4
4	1	1	1	3	-1	4	1	1	1	2	-3	-4	6	1
4	1	1	2	3	-1	4	1	2	1	2	-3	-4	6	3
4	1	1	1	3	-1	4	1	1	1	2	-3	-4	6	-4
4	1	1	3	3	-1	4	1	3	1	2	-3	-4	6	1
4	1	1	2	3	-1	4	1	2	1	2	-3	-4	6	3
AGS	AGC	LI3	RI3	LI4	ISB	IML	LQ1	RQ1	LQ4	QSB	QML	ADD	AER	TIR
5	1	3	1	2	-1	-1	3	1	3	-1	4	1	-8	2
5	1	3	3	2	-1	-1	3	3	3	-1	4	1	-8	6
5	1	3	2	2	-1	-1	3	2	3	-1	4	1	-8	-8
5	1	3	1	2	-1	-1	3	1	3	-1	4	1	-8	1
5	1	3	3	2	-1	-1	3	3	3	-1	4	1	-8	3
5	1	3	1	2	-1	-1	3	1	3	-1	4	1	-8	-4
5	1	3	3	2	-1	-1	3	3	3	-1	4	1	-8	1
5	1	3	2	2	-1	-1	3	2	3	-1	4	1	-8	3

PAGE 137 INTENTIONALLY BLANK

PRECEDING PAGE BLANK NOT FILMED

Appendix F

PROGRAMS IN C AND SIMULATION RESULTS FOR DATA FLOW OF SAMPLES IN THE HARDWARE INTERFACE

```
/* Hardware behavioural model for data flow in the carrier
data and storage modules */

/* This program aims at governing the data flow among the
three modules come into play by interfacing them. The data
flow, control circuitry and interfacing is easily simulated
for the structure. */

#include <stdio.h>

main()

{
                        /* Initializations */

int read_disable=0;
int l=1,en=0;
int count,m,k,i,q,j,adder_I,adder_Q;
int I_buf,Q_buf,I_inv,Q_inv;
int agc=0;
int h=1;
int sine,cosine,i_sample,q_sample,I_temp,Q_temp;
int d=1,s=1,x=0,ARE=0,ctr=0,SRE=0;
int i_bufferlatch[5],q_bufferlatch[5];
int mult[5];
int add,sub,latch[3];
int srlatch_s,srlatch_c;
FILE *fp,*fp1;
                        /* Define structures for memory */
struct ram_d
        {
        int location_I[9];
        int location_Q[9];
        };
struct ram_d DDR;

struct rambuffer
        {
        int latch;
        int location[9];
        };
struct rambuffer ram_i[5],ram_q[5];

struct ram
        {
        int location[10];
        };
struct ram Accumulation_RAM_I,Accumulation_RAM_Q;

struct doubleram
        {
        int location_S[9];
        int location_C[9];
        };
struct doubleram Storage_RAM;

struct rom
        {
        int location_S;
        int location_C;
        };
struct rom Output_ROM;

struct irom
        {
        int location_I;
        int location_Q;
        };
struct irom Input_ROM;
                                        /* Initializations for
                                        memory structure */

   for (j=1;j<=9;j++)
      {
      Storage_RAM.location_S[j]=0;
      Storage_RAM.location_C[j]=0;
      Accumulation_RAM_I.location[j]=0;
      Accumulation_RAM_Q.location[j]=0;
      Input_ROM.location_I=0;
      Input_ROM.location_Q=0;
      Output_ROM.location_S=0;
      Output_ROM.location_C=0;
      }

for (i=1;i<=4;i++)
   {
   for (j=1;j<=8;j++)
      {
      ram_i[i].location[j]=0;
      }
   ram_i[i].latch=0;
   }


for (i=1;i<=4;i++)
   {
   for (j=1;j<=8;j++)
      {
      ram_q[i].location[j]=0;
      }
   ram_q[i].latch=0;
   }

   for (j=1;j<9;j++)
      {
      DDR.location_I[j]=0;
      DDR.location_Q[j]=0;
```

```
       }
       add=0;
       sub=0;
       latch[1]=0;
       latch[2]=0;
                                   /* Start main program */

printf("Enter the count of Address Generator for Samples Please");
printf("\n");
scanf("%d",&count);
fp=fopen("sam.dat","r");
fp1=fopen("outint.dat","w");
fprintf(fp1,"Enter the count of Address Generator for Samples Please");
fprintf(fp1,"\n");
fprintf(fp1,"count");
fprintf(fp1, "\t");
fprintf(fp1,"%d", count);
fprintf(fp1,"\n");


for (k=1;k<=count;k++)
   {
    fscanf(fp,"%d",&i);
    fscanf(fp,"%d",&q);          /* clock is negative */
                                  /* Data flow in MCRM */
    if (SRE==1)
    {
    Storage_RAM.location_C[s]=I_temp;
    Storage_RAM.location_S[s]=Q_temp;
    s=s+1;
    if (s>8)
       {
       s=s-8;
       SRE=0;
       }
    }
    Output_ROM.location_C=adder_I;
    Output_ROM.location_S=adder_Q;
    if (read_disable==1)
       {
       adder_I=Accumulation_RAM_I.location[d]+Input_ROM.location_I;
       adder_Q=Accumulation_RAM_Q.location[d]+Input_ROM.location_Q;
       }
    if (read_disable==0)
       {
       adder_I=Input_ROM.location_I;
       adder_Q=Input_ROM.location_Q;
       h=h+1;
       if (h>8)
          {
          h=h-8;
          read_disable=1;
          }
       }
```

```
Input_ROM.location_I=i;
Input_ROM.location_Q=q;
ram_i[1].latch=i;                       /* clock is negative */
ram_q[1].latch=q;
for (m=2;m<4;m++)
   {
   ram_q[m].latch=ram_q[m-1].location[l];
   }
for (m=2;m<4;m++)
   {
   ram_i[m].latch=ram_i[m-1].location[l];
   }
                                        /* Additional buffer
                            interface needed for the modules */

i_bufferlatch[4]=i_bufferlatch[3];
i_bufferlatch[3]=i_bufferlatch[2];
i_bufferlatch[2]=i_bufferlatch[1];
i_bufferlatch[1]=ram_i[3].location[l];
q_bufferlatch[4]=q_bufferlatch[3];
q_bufferlatch[3]=q_bufferlatch[2];
q_bufferlatch[2]=q_bufferlatch[1];
q_bufferlatch[1]=ram_q[3].location[l];
srlatch_s=Storage_RAM.location_S[s-1];
srlatch_c=Storage_RAM.location_C[s-1];

                            /* Data in MDRM */

latch[1]=add;
latch[2]=sub;
add=mult[1]+mult[2];
sub=mult[3]-mult[4];
mult[1]=i_sample*cosine;
mult[2]=q_sample*sine;
mult[3]=q_sample*cosine;
mult[4]=i_sample*sine;
q_sample=q_bufferlatch[4];
i_sample=i_bufferlatch[4];
sine=srlatch_s;
cosine=srlatch_c;                       /* clock is positive */

DDR.location_I[l]=I_buf;
DDR.location_Q[l]=Q_buf;
I_buf=I_inv;
Q_buf=Q_inv;
I_inv=latch[1];
Q_inv=latch[2];
I_temp=Output_ROM.location_C;
Q_temp=Output_ROM.location_S;

                                        /* Control circuitry */
if (ARE==1)
   {
   Accumulation_RAM_I.location[d]=adder_I;
```

```
    Accumulation_RAM_Q.location[d]=adder_Q;
    d=d+1;
    if (d>8)
       {
       d=d-8;
       }
    }

                                                   /* MRBS data flow */
for (m=1;m<4;m++)
   {
   ram_i[m].location[1]=ram_i[m].latch;
   ram_q[m].location[1]=ram_q[m].latch;
   }
                                                   /* Print the data flow */

fprintf(fp1,
"%3s %3s %3s %3s %3s %3s %3s %3s %3s %3s %3s %3s %3s %3s %3s %3s %3s %3s\n",
"AGS","AGC","ARI","ARQ","SRC","SRS","LI3","RI3","LQ3","RQ3",
"BI4","BQ4","SLC","SLS","DIL","DQL","RID","RQD");
for (j=1;j<9;j++)
   {
   fprintf(fp1,"%2d",l);
   fprintf(fp1,"%4d",agc);
   fprintf(fp1,"%4d",Accumulation_RAM_I.location[j]);
   fprintf(fp1,"%4d",Accumulation_RAM_Q.location[j]);
   fprintf(fp1,"%4d",Storage_RAM.location_C[j]);
   fprintf(fp1,"%4d",Storage_RAM.location_S[j]);
   fprintf(fp1,"%4d",ram_i[3].latch);
   fprintf(fp1,"%4d",ram_i[3].location[j]);
   fprintf(fp1,"%4d",ram_q[3].latch);
   fprintf(fp1,"%4d",ram_q[3].location[j]);
   fprintf(fp1,"%4d",i_bufferlatch[4]);
   fprintf(fp1,"%4d",q_bufferlatch[4]);
   fprintf(fp1,"%4d",srlatch_c);
   fprintf(fp1,"%4d",srlatch_s);
   fprintf(fp1,"%4d",latch[1]);
   fprintf(fp1,"%4d",latch[2]);
   fprintf(fp1,"%4d",DDR.location_I[j]);
   fprintf(fp1,"%4d",DDR.location_Q[j]);
   fprintf(fp1,"\n");
   }
                                           /* Control circuitry */
l=l+1;
if (en==1)
    {
    if (l==4) SRE=1;
    }
if (l==2) ARE=1;
if (l>8)
    {
     l=(l-8);
     agc=agc+1;
     if (agc==3) en=1;
```

```
                 if (agc>3) agc=agc-4;
                 if (agc==0) en=0;
                 printf("\n");
                }
             if (agc==0)
                {
                if (1==2) read_disable=0;
                }
        }
        fclose(fp);
        fclose(fp1);
        }
```

INPUT FILE FOR HARDWARE INTERFACE

```
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
```

OUTPUT FILE OF INTERFACE

```
Enter the count of Address Generator for Samples Please
count   50
AGS AGC ARI ARQ SRC SRS LI3 RI3 LQ3 RQ3 BI4 BQ4 SLC SLS DIL DQL RID RQD
 1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
AGS AGC ARI ARQ SRC SRS LI3 RI3 LQ3 RQ3 BI4 BQ4 SLC SLS DIL DQL RID RQD
 2   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
AGS AGC ARI ARQ SRC SRS LI3 RI3 LQ3 RQ3 BI4 BQ4 SLC SLS DIL DQL RID RQD
 3   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 3   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 3   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 3   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 3   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 3   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 3   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 3   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
AGS AGC ARI ARQ SRC SRS LI3 RI3 LQ3 RQ3 BI4 BQ4 SLC SLS DIL DQL RID RQD
 4   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 4   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 4   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 4   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 4   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 4   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 4   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 4   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
AGS AGC ARI ARQ SRC SRS LI3 RI3 LQ3 RQ3 BI4 BQ4 SLC SLS DIL DQL RID RQD
 5   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 5   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 5   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 5   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 5   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 5   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 5   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 5   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
AGS AGC ARI ARQ SRC SRS LI3 RI3 LQ3 RQ3 BI4 BQ4 SLC SLS DIL DQL RID RQD
 6   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 6   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 6   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 6   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 6   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 6   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

```
  6   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  6   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
AGS AGC ARI ARQ SRC SRS LI3 RI3 LQ3 RQ3 BI4 BQ4 SLC SLS DIL DQL RID RQD
  7   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  7   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  7   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  7   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  7   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  7   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  7   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  7   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
AGS AGC ARI ARQ SRC SRS LI3 RI3 LQ3 RQ3 BI4 BQ4 SLC SLS DIL DQL RID RQD
  8   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  8   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  8   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  8   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  8   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  8   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  8   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  8   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
AGS AGC ARI ARQ SRC SRS LI3 RI3 LQ3 RQ3 BI4 BQ4 SLC SLS DIL DQL RID RQD
  1   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  1   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  1   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  1   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  1   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  1   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  1   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  1   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
AGS AGC ARI ARQ SRC SRS LI3 RI3 LQ3 RQ3 BI4 BQ4 SLC SLS DIL DQL RID RQD
  2   1   2   4   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  2   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  2   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  2   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  2   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  2   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  2   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  2   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
AGS AGC ARI ARQ SRC SRS LI3 RI3 LQ3 RQ3 BI4 BQ4 SLC SLS DIL DQL RID RQD
  3   1   2   4   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  3   1   2   4   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  3   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  3   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  3   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  3   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  3   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  3   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
AGS AGC ARI ARQ SRC SRS LI3 RI3 LQ3 RQ3 BI4 BQ4 SLC SLS DIL DQL RID RQD
  4   1   2   4   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  4   1   2   4   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  4   1   2   4   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  4   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  4   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  4   1   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

4	1	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
5	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
6	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
7	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
8	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
1	2	2	4	0	0	1	1	2	2	0	0	0	0	0	0	0	0
1	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
1	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
1	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
1	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
1	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
1	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
1	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
2	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
2	2	2	4	0	0	1	1	2	2	0	0	0	0	0	0	0	0
2	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
2	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
2	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
2	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0

2	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
2	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
3	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
3	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
3	2	2	4	0	0	1	1	2	2	0	0	0	0	0	0	0	0
3	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
3	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
3	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
3	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
3	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
4	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
4	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
4	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
4	2	2	4	0	0	1	1	2	2	0	0	0	0	0	0	0	0
4	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
4	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
4	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
4	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
5	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
5	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
5	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
5	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
5	2	2	4	0	0	1	1	2	2	0	0	0	0	0	0	0	0
5	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
5	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
5	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
6	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
6	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
6	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
6	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
6	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
6	2	2	4	0	0	1	1	2	2	0	0	0	0	0	0	0	0
6	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
6	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
7	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
7	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
7	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
7	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
7	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
7	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
7	2	2	4	0	0	1	1	2	2	0	0	0	0	0	0	0	0
7	2	2	4	0	0	1	0	2	0	0	0	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
8	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
8	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
8	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
8	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
8	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0
8	2	3	6	0	0	1	1	2	2	0	0	0	0	0	0	0	0

```
  8   2   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  8   2   2   4   0   0   1   1   2   2   0   0   0   0   0   0   0   0
AGS AGC ARI ARQ SRC SRS LI3 RI3 LQ3 RQ3 BI4 BQ4 SLC SLS DIL DQL RID RQD
  1   3   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  1   3   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  1   3   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  1   3   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  1   3   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  1   3   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  1   3   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  1   3   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
AGS AGC ARI ARQ SRC SRS LI3 RI3 LQ3 RQ3 BI4 BQ4 SLC SLS DIL DQL RID RQD
  2   3   4   8   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  2   3   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  2   3   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  2   3   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  2   3   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  2   3   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  2   3   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  2   3   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
AGS AGC ARI ARQ SRC SRS LI3 RI3 LQ3 RQ3 BI4 BQ4 SLC SLS DIL DQL RID RQD
  3   3   4   8   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  3   3   4   8   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  3   3   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  3   3   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  3   3   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  3   3   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  3   3   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
  3   3   3   6   0   0   1   1   2   2   0   0   0   0   0   0   0   0
AGS AGC ARI ARQ SRC SRS LI3 RI3 LQ3 RQ3 BI4 BQ4 SLC SLS DIL DQL RID RQD
  4   3   4   8   4   8   1   1   2   2   1   2   4   8   0   0   0   0
  4   3   4   8   0   0   1   1   2   2   1   2   4   8   0   0   0   0
  4   3   4   8   0   0   1   1   2   2   1   2   4   8   0   0   0   0
  4   3   3   6   0   0   1   1   2   2   1   2   4   8   0   0   0   0
  4   3   3   6   0   0   1   1   2   2   1   2   4   8   0   0   0   0
  4   3   3   6   0   0   1   1   2   2   1   2   4   8   0   0   0   0
  4   3   3   6   0   0   1   1   2   2   1   2   4   8   0   0   0   0
  4   3   3   6   0   0   1   1   2   2   1   2   4   8   0   0   0   0
AGS AGC ARI ARQ SRC SRS LI3 RI3 LQ3 RQ3 BI4 BQ4 SLC SLS DIL DQL RID RQD
  5   3   4   8   4   8   1   1   2   2   1   2   4   8   0   0   0   0
  5   3   4   8   4   8   1   1   2   2   1   2   4   8   0   0   0   0
  5   3   4   8   0   0   1   1   2   2   1   2   4   8   0   0   0   0
  5   3   4   8   0   0   1   1   2   2   1   2   4   8   0   0   0   0
  5   3   3   6   0   0   1   1   2   2   1   2   4   8   0   0   0   0
  5   3   3   6   0   0   1   1   2   2   1   2   4   8   0   0   0   0
  5   3   3   6   0   0   1   1   2   2   1   2   4   8   0   0   0   0
  5   3   3   6   0   0   1   1   2   2   1   2   4   8   0   0   0   0
AGS AGC ARI ARQ SRC SRS LI3 RI3 LQ3 RQ3 BI4 BQ4 SLC SLS DIL DQL RID RQD
  6   3   4   8   4   8   1   1   2   2   1   2   4   8   0   0   0   0
  6   3   4   8   4   8   1   1   2   2   1   2   4   8   0   0   0   0
  6   3   4   8   4   8   1   1   2   2   1   2   4   8   0   0   0   0
  6   3   4   8   0   0   1   1   2   2   1   2   4   8   0   0   0   0
  6   3   4   8   0   0   1   1   2   2   1   2   4   8   0   0   0   0
  6   3   3   6   0   0   1   1   2   2   1   2   4   8   0   0   0   0
```

6	3	3	6	0	0	1	1	2	2	1	2	4	8	0	0	0	0
6	3	3	6	0	0	1	1	2	2	1	2	4	8	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
7	3	4	8	4	8	1	1	2	2	1	2	4	8	20	0	0	0
7	3	4	8	4	8	1	1	2	2	1	2	4	8	20	0	0	0
7	3	4	8	4	8	1	1	2	2	1	2	4	8	20	0	0	0
7	3	4	8	4	8	1	1	2	2	1	2	4	8	20	0	0	0
7	3	4	8	0	0	1	1	2	2	1	2	4	8	20	0	0	0
7	3	4	8	0	0	1	1	2	2	1	2	4	8	20	0	0	0
7	3	3	6	0	0	1	1	2	2	1	2	4	8	20	0	0	0
7	3	3	6	0	0	1	1	2	2	1	2	4	8	20	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
8	3	4	8	4	8	1	1	2	2	1	2	4	8	20	0	0	0
8	3	4	8	4	8	1	1	2	2	1	2	4	8	20	0	0	0
8	3	4	8	4	8	1	1	2	2	1	2	4	8	20	0	0	0
8	3	4	8	4	8	1	1	2	2	1	2	4	8	20	0	0	0
8	3	4	8	4	8	1	1	2	2	1	2	4	8	20	0	0	0
8	3	4	8	0	0	1	1	2	2	1	2	4	8	20	0	0	0
8	3	4	8	0	0	1	1	2	2	1	2	4	8	20	0	0	0
8	3	3	6	0	0	1	1	2	2	1	2	4	8	20	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
1	0	4	8	4	8	1	1	2	2	1	2	4	8	20	0	20	0
1	0	4	8	4	8	1	1	2	2	1	2	4	8	20	0	0	0
1	0	4	8	4	8	1	1	2	2	1	2	4	8	20	0	0	0
1	0	4	8	4	8	1	1	2	2	1	2	4	8	20	0	0	0
1	0	4	8	4	8	1	1	2	2	1	2	4	8	20	0	0	0
1	0	4	8	4	8	1	1	2	2	1	2	4	8	20	0	0	0
1	0	4	8	0	0	1	1	2	2	1	2	4	8	20	0	0	0
1	0	4	8	0	0	1	1	2	2	1	2	4	8	20	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
2	0	1	2	4	8	1	1	2	2	1	2	4	8	20	0	20	0
2	0	4	8	4	8	1	1	2	2	1	2	4	8	20	0	20	0
2	0	4	8	4	8	1	1	2	2	1	2	4	8	20	0	0	0
2	0	4	8	4	8	1	1	2	2	1	2	4	8	20	0	0	0
2	0	4	8	4	8	1	1	2	2	1	2	4	8	20	0	0	0
2	0	4	8	4	8	1	1	2	2	1	2	4	8	20	0	0	0
2	0	4	8	4	8	1	1	2	2	1	2	4	8	20	0	0	0
2	0	4	8	0	0	1	1	2	2	1	2	4	8	20	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
3	0	1	2	4	8	1	1	2	2	1	2	0	0	20	0	20	0
3	0	1	2	4	8	1	1	2	2	1	2	0	0	20	0	20	0
3	0	4	8	4	8	1	1	2	2	1	2	0	0	20	0	20	0
3	0	4	8	4	8	1	1	2	2	1	2	0	0	20	0	0	0
3	0	4	8	4	8	1	1	2	2	1	2	0	0	20	0	0	0
3	0	4	8	4	8	1	1	2	2	1	2	0	0	20	0	0	0
3	0	4	8	4	8	1	1	2	2	1	2	0	0	20	0	0	0
3	0	4	8	4	8	1	1	2	2	1	2	0	0	20	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
4	0	1	2	4	8	1	1	2	2	1	2	0	0	20	0	20	0
4	0	1	2	4	8	1	1	2	2	1	2	0	0	20	0	20	0
4	0	1	2	4	8	1	1	2	2	1	2	0	0	20	0	20	0
4	0	4	8	4	8	1	1	2	2	1	2	0	0	20	0	20	0
4	0	4	8	4	8	1	1	2	2	1	2	0	0	20	0	0	0
4	0	4	8	4	8	1	1	2	2	1	2	0	0	20	0	0	0

4	0	4	8	4	8	1	1	2	2	1	2	0	0	20	0	0	0
4	0	4	8	4	8	1	1	2	2	1	2	0	0	20	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
5	0	1	2	4	8	1	1	2	2	1	2	0	0	20	0	20	0
5	0	1	2	4	8	1	1	2	2	1	2	0	0	20	0	20	0
5	0	1	2	4	8	1	1	2	2	1	2	0	0	20	0	20	0
5	0	1	2	4	8	1	1	2	2	1	2	0	0	20	0	20	0
5	0	4	8	4	8	1	1	2	2	1	2	0	0	20	0	20	0
5	0	4	8	4	8	1	1	2	2	1	2	0	0	20	0	0	0
5	0	4	8	4	8	1	1	2	2	1	2	0	0	20	0	0	0
5	0	4	8	4	8	1	1	2	2	1	2	0	0	20	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
6	0	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
6	0	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
6	0	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
6	0	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
6	0	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
6	0	4	8	4	8	1	1	2	2	1	2	0	0	0	0	20	0
6	0	4	8	4	8	1	1	2	2	1	2	0	0	0	0	0	0
6	0	4	8	4	8	1	1	2	2	1	2	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
7	0	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
7	0	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
7	0	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
7	0	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
7	0	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
7	0	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
7	0	4	8	4	8	1	1	2	2	1	2	0	0	0	0	20	0
7	0	4	8	4	8	1	1	2	2	1	2	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
8	0	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
8	0	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
8	0	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
8	0	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
8	0	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
8	0	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
8	0	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
8	0	4	8	4	8	1	1	2	2	1	2	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
1	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	0	0
1	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
1	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
1	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
1	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
1	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
1	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
1	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
2	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
2	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	0	0
2	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
2	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
2	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
2	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0

2	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
2	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
3	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
3	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
3	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	0	0
3	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
3	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
3	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
3	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
3	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
4	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
4	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
4	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
4	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	0	0
4	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
4	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
4	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
4	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
5	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
5	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
5	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
5	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
5	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	0	0
5	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
5	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
5	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
6	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
6	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
6	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
6	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
6	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
6	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	0	0
6	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	20	0
6	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
7	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
7	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
7	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
7	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
7	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
7	1	2	3	4	8	1	1	2	2	1	2	0	0	0	0	0	0
7	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	0	0
7	1	1	2	4	8	1	1	2	2	1	2	0	0	0	0	0	0
AGS	AGC	ARI	ARQ	SRC	SRS	LI3	RI3	LQ3	RQ3	BI4	BQ4	SLC	SLS	DIL	DQL	RID	RQD
8	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
8	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
8	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
8	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
8	1	2	4	4	8	1	1	2	2	1	2	0	0	0	0	0	0
8	1	2	3	4	8	1	1	2	2	1	2	0	0	0	0	0	0

```
 8   1   2   3   4   8   1   1   2   2   1   2   0   0   0   0   0   0
 8   1   1   2   4   8   1   1   2   2   1   2   0   0   0   0   0   0
AGS AGC ARI ARQ SRC SRS LI3 RI3 LQ3 RQ3 BI4 BQ4 SLC SLS DIL DQL RID RQD
 1   2   2   4   4   8   1   1   2   2   1   2   0   0   0   0   0   0
 1   2   2   4   4   8   1   1   2   2   1   2   0   0   0   0   0   0
 1   2   2   4   4   8   1   1   2   2   1   2   0   0   0   0   0   0
 1   2   2   4   4   8   1   1   2   2   1   2   0   0   0   0   0   0
 1   2   2   4   4   8   1   1   2   2   1   2   0   0   0   0   0   0
 1   2   2   3   4   8   1   1   2   2   1   2   0   0   0   0   0   0
 1   2   2   3   4   8   1   1   2   2   1   2   0   0   0   0   0   0
 1   2   2   3   4   8   1   1   2   2   1   2   0   0   0   0   0   0
AGS AGC ARI ARQ SRC SRS LI3 RI3 LQ3 RQ3 BI4 BQ4 SLC SLS DIL DQL RID RQD
 2   2   3   5   4   8   1   1   2   2   1   2   0   0   0   0   0   0
 2   2   2   4   4   8   1   1   2   2   1   2   0   0   0   0   0   0
 2   2   2   4   4   8   1   1   2   2   1   2   0   0   0   0   0   0
 2   2   2   4   4   8   1   1   2   2   1   2   0   0   0   0   0   0
 2   2   2   4   4   8   1   1   2   2   1   2   0   0   0   0   0   0
 2   2   2   3   4   8   1   1   2   2   1   2   0   0   0   0   0   0
 2   2   2   3   4   8   1   1   2   2   1   2   0   0   0   0   0   0
 2   2   2   3   4   8   1   1   2   2   1   2   0   0   0   0   0   0
```

Appendix G

A PROGRAM FOR THE MAPPING OF 2 CHANNELS ON A BINARY HYPERCUBE FOR MODEL-I

```
-- This is a program for simulation of 32 simultaneous
-- voice channels. Each channel has 4 tasks operating in parallel
-- or pipeline as in Model-I.  In this program each task is assumed
-- to be assigned to a processor. Therefore each hypercube has
-- 2 channels assigned to it. We need 16 hypercubes for this purpose.

with math_lib;
with text_io;
procedure demod_hypercube is
use text_io;
type real is new float;
package my_real is new float_io(real);
use my_real;
package math1 is new math_lib(real);
use math1;
package int1_io is new integer_io(integer);
use int1_io;

count: integer;
input_buffer  : file_type;
output_buffer : file_type;
I_sample, Q_sample : real;

-- Define the tasks in parallel or pipeline

task type each_carrier_1 is
   entry sample (In_s, Qn_s : in real; Channel_no : in  integer);
end;

task type each_carrier_2 is
   entry sample_data(In_data,Qn_data: in real;channel: in integer);
   entry new_sample(new_I,new_Q : in real; ch: in integer);
end;

task type each_timing is
   entry samples (I_sample, Q_sample : in real; channel : in integer);
end;

task type each_data_recovery is
   entry phase(i_part,q_part: in real; channel : in integer);
end;

-- Define an array of parallel tasks

multi_carrier_2: array(1..32) of each_carrier_2;
multi_data_recovery: array(1..32) of each_data_recovery;
multi_carrier_1: array(1..32) of each_carrier_1;
multi_timing: array(1..32) of each_timing;

-- Code alloted to task Each_Carrier-1
--
task body each_carrier_1 is
   new_In, new_Qn : real;
```

```
      updated_In, updated_Qn: real:=0.0;
      new_magnitude, new_phase : real;
      m : real := 4.0;
      type period is array(1..16) of real;
      temp_In, temp_Qn : period;
      channel : integer;
   begin
      for i in 1..16 loop
         accept sample(In_s,Qn_s : in real; Channel_no: in integer) do
            temp_In(i) := In_s;
            temp_Qn(i) := Qn_s;
            channel := channel_no;
         end;
      end loop;
      for i in 1..16 loop
         new_magnitude := (temp_In(i) * temp_In(i) + temp_Qn(i) * temp_Qn(i));
         new_magnitude := new_magnitude * new_magnitude;
         new_phase     := m * atan(temp_Qn(i)/temp_In(i));
         new_In        := new_magnitude * cos (new_phase);
         new_Qn        := new_magnitude * sin (new_phase);
         updated_In    := new_In + updated_In;
         updated_Qn    := new_Qn + updated_Qn;
      end loop;
      multi_carrier_2(channel).new_sample(updated_In,updated_Qn,channel);
   end;

   -- Code assigned to task Each_Carrier_2

   task body each_carrier_2 is
      updated_In,updated_Qn: real;
      output_In,output_Qn,carrier_phase: real;
      sine_output, cosine_output:real;
      channel: integer;
      type period is array(1..16) of real;
      my_In,my_Qn, digital_I : period;
      ch: integer;

   begin
      for i in 1..16 loop
         accept sample_data (In_data, Qn_data : in real; channel: in integer) do
            my_In(i) := In_data;
            my_Qn(i) := Qn_data;
            ch := channel;
         end;
      end loop;
      accept new_sample( new_I, new_Q : in real; ch: in integer) do
         updated_In:= new_I;
         updated_Qn:= new_Q;
         channel := ch;
      end;

      output_In        := updated_In/16.0;
      output_Qn        := updated_Qn/16.0;
      carrier_phase    := (1.0/4.0)*atan(output_Qn/output_In);
   sine_output      := sin (carrier_phase);
   cosine_output    := cos (carrier_phase);
   for i in 1..16 loop
      my_In(i) := my_In(i)*cosine_output + my_Qn(i)*sine_output;
      my_Qn(i) := my_Qn(i)*cosine_output - my_In(i)*sine_output;
   end loop;
   for i in 1..16 loop
      multi_data_recovery(channel).phase(my_In(i),my_Qn(i),channel);
      multi_timing(channel).samples(my_In(i),my_Qn(i),channel);
   end loop;
 end;

-- Code assigned to Each_timing

task body each_timing is
   Un, I_Un, Q_Un : real;
   Wn : real := 0.0;
   type estimate is array(1..16) of real;
   in_between_I, in_between_Q : estimate;
   ch:integer;
begin
   for i in 1..16 loop
      accept samples(I_sample, Q_sample : in real; channel: in integer) do
         in_between_I(i) := I_sample;
         in_between_Q(i) := Q_sample;
         ch := channel;
      end;
   end loop;
   for i in 1..16 loop
      if (i mod 2)= 1  and i < 15 then
         I_Un := (in_between_I(i)- in_between_I(i+2))*in_between_I(i+1);
         Q_Un := (in_between_Q(i)- in_between_Q(i+2))*in_between_Q(i+1);
         Un   := I_Un + Q_Un;
         Wn   := Wn + Un;
         new_line;
         put("This is timing of ");
         put(ch);
         put(" channel");
         put(wn);
         new_line;
      end if;
   end loop;
end;

-- Code assigned to Each_data_recovery

task body each_data_recovery is
   type period is array(1..16) of real;
   my_In,my_Qn, digital_I, digital_Q : period;
   ch: integer;
begin
   for i in 1..16 loop
      accept phase(i_part, q_part: in real; channel: in integer) do
         digital_I(i) :=i_part;
```

```
            digital_Q(i) :=q_part;
            ch := channel;
         end;
      end loop;
      for i in 1..16 loop
         new_line(2);
         put("This is I  data of channel");
         put(ch);
         if digital_I(i) > 0.0 then
            digital_I(i) := 1.0;
         else
            digital_I(i) := 0.0;
         end if;
         put(digital_I(i));
         new_line;
         put("This is Q  data of channel");
         put(ch);
         if digital_Q(i) > 0.0 then
            digital_Q(i) := 1.0;
         else
            digital_Q(i) := 0.0;
         end if;
         put(digital_Q(i));
      end loop;
      new_line;
   end;

   -- Main program sets off the tasks of several channels to
   -- operate in parallel or pipeline.
   -- It calls the tasks for 32 channels. It gets the input
   -- from a buffer called "samples.ada" and transfers data to the
   -- tasks which operate on this data.

   begin
      open(input_buffer,in_file,"samples.ada");
      for k in 1..16 loop
         count:=1;
         while not(end_of_file(input_buffer))and count <= 32 loop
            get(input_buffer,I_sample);
            get(input_buffer,Q_sample);
            multi_carrier_1(count).sample(I_sample,Q_sample,count);
            multi_carrier_2(count).sample_data(I_sample,Q_sample,count);
            count:=count+1;
         end loop;
      end loop;
      close(input_buffer);
   end;
```

Appendix H

A PROGRAM FOR LOAD BALANCED PROCESSING OF 2 CHANNELS ON A BINARY HYPERCUBE FOR MODEL-II.

```
-- This is a program for simulation of 32  channels. It has
-- 4 tasks operate in parallel or pipeline for each channel.
-- The tasks are assigned such that they have nearly equal amount
-- of operations (load balancing).


with math_lib;
with text_io;
procedure sigma_demod is
use text_io;
type real is new float;
package my_real is new float_io(real);
use my_real;
package math1 is new math_lib(real);
use math1;
package int1_io is new integer_io(integer);
use int1_io;

count: integer;
input_buffer  : file_type;
output_buffer : file_type;
I_sample, Q_sample : real;

-- Define the tasks for channel

task type each_carrier_1i is
   entry sample_i (In_s, Qn_s : in real; Channel_no : in  integer);
end;

task type each_carrier_1q is
   entry sample_q (In_s, Qn_s : in real; Channel_no : in  integer);
end;

task type each_carrier_2 is
   entry sample_data(In_data,Qn_data: in real;channel: in integer);
   entry new_sample_i(new_I : in real; ch: in integer);
   entry new_sample_q(new_Q : in real; ch: in integer);
end;

task type each_timing is
   entry samples (I_sample, Q_sample : in real; channel : in integer);
end;

-- Define the maximum channel array in the system

multi_carrier_2: array(1..32) of each_carrier_2;
multi_carrier_1i : array(1..32) of each_carrier_1i;
multi_carrier_1q : array(1..32) of each_carrier_1q;
multi_timing: array(1..32) of each_timing;

-- Define code for each_carrier_1i

task body each_carrier_1i is
   new_In, new_Qn : real;
   updated_In, updated_Qn: real:=0.0;
   new_magnitude, new_phase : real;
   m : real := 4.0;
   type period is array(1..16) of real;
   temp_In, temp_Qn : period;
   channel : integer;
begin
   for i in 1..16 loop
      accept sample_i(In_s,Qn_s : in real; Channel_no: in integer) do
         temp_In(i) := In_s;
         temp_Qn(i) := Qn_s;
         channel := channel_no;
      end;
   end loop;
   for i in 1..16 loop
      new_magnitude := (temp_In(i) * temp_In(i) + temp_Qn(i) * temp_Qn(i));
      new_magnitude := new_magnitude * new_magnitude;
      new_phase     := m * atan(temp_Qn(i)/temp_In(i));
      new_In        := new_magnitude * cos (new_phase);
      updated_In    := new_In + updated_In;
   end loop;
   multi_carrier_2(channel).new_sample_i(updated_In,channel);
end;

-- Define code for each_carrier_1q

task body each_carrier_1q is
   new_In, new_Qn : real;
   updated_In, updated_Qn: real:=0.0;
   new_magnitude, new_phase : real;
   m : real := 4.0;
   type period is array(1..16) of real;
   temp_In, temp_Qn : period;
   channel : integer;
begin
   for i in 1..16 loop
      accept sample_q(In_s,Qn_s : in real; Channel_no: in integer) do
         temp_In(i) := In_s;
         temp_Qn(i) := Qn_s;
         channel := channel_no;
      end;
   end loop;
   for i in 1..16 loop
      new_magnitude := (temp_In(i) * temp_In(i) + temp_Qn(i) * temp_Qn(i));
      new_magnitude := new_magnitude * new_magnitude;
      new_phase     := m * atan(temp_Qn(i)/temp_In(i));
      new_Qn        := new_magnitude * sin (new_phase);
      updated_Qn    := new_Qn + updated_Qn;
   end loop;
   multi_carrier_2(channel).new_sample_q(updated_Qn,channel);

--The tasks each_carrier_1i and each_carrier_1q operate in parallel.
-- Define code for each_carrier_2. It needs input from both the
-- previous tasks.
```

```
task body each_carrier_2 is
   updated_In,updated_Qn: real;
   output_In,output_Qn,carrier_phase: real;
   sine_output, cosine_output:real;
   channel: integer;
   type period is array(1..16) of real;
   my_In,my_Qn, digital_I,digital_Q : period;
   ch: integer;

begin
   for i in 1..16 loop
      accept sample_data (In_data, Qn_data : in real; channel: in integer) do
         my_In(i) := In_data;
         my_Qn(i) := Qn_data;
         ch := channel;
      end;
   end loop;
   accept new_sample_i( new_I : in real; ch: in integer) do
      updated_In:= new_I;
      channel := ch;
   end;
   accept new_sample_q( new_Q : in real; ch: in integer) do
      updated_Qn:= new_Q;
      channel := ch;
   end;

   output_In        := updated_In/16.0;
   output_Qn        := updated_Qn/16.0;
   carrier_phase    := (1.0/4.0)*atan(output_Qn/output_In);
   sine_output      := sin (carrier_phase);
   cosine_output    := cos (carrier_phase);

   for i in 1..16 loop
      digital_I(i) := my_In(i)*cosine_output + my_Qn(i)*sine_output;
      digital_Q(i) := my_Qn(i)*cosine_output - my_In(i)*sine_output;
      if digital_I(i) > 0.0 then
         my_In(i) := 1.0;
      else
         my_In(i) := 0.0;
      end if;
      put("channel");
      put(ch);
      new_line;
      if digital_Q(i) > 0.0 then
         my_Qn(i) := 1.0;
      else
         my_Qn(i) := 0.0;
      end if;
      put(my_In(i));
      put(my_Qn(i));
   end loop;
   new_line;
```

```
    for i in 1..16 loop
       multi_timing(channel).samples(digital_I(i), digital_Q(i),channel);
    end loop;
  end;

 -- Define code for each_timing. It needs input from each_carrier_2.
 -- These four tasks define the division of work among the processors
 -- of a hypercube for each of the channel.

 task body each_timing is
    Un, I_Un, Q_Un : real;
    Wn : real := 0.0;
    type estimate is array(1..16) of real;
    in_between_I, in_between_Q : estimate;
    ch:integer;
 begin
    for i in 1..16 loop
       accept samples(I_sample, Q_sample : in real; channel: in integer) do
          in_between_I(i) := I_sample;
          in_between_Q(i) := Q_sample;
          ch := channel;
       end;
    end loop;
    for i in 1..16 loop
       if (i mod 2)= 1  and i < 15 then
          I_Un := (in_between_I(i)- in_between_I(i+2))*in_between_I(i+1);
          Q_Un := (in_between_Q(i)- in_between_Q(i+2))*in_between_Q(i+1);
          Un   := I_Un + Q_Un;
          Wn   := Wn + Un;
          new_line;
          put("This is timing of ");
          put(ch);
          put(" channel");
          put(wn);
          new_line;
       end if;
    end loop;
 end;

 -- The main program is the front end system. It picks up the data
 -- from a buffer for each channel and starts tasks which operate
 -- in parallel or pipeline. The tasks each_carrier_1i and
 -- each_carrier_1q operate in parallel. Their input is needed by
 -- task each_carrier_2. The output of this task is used by the
 -- task each_timing.


 begin
    open(input_buffer,in_file,"samples.ada");
    for k in 1..16 loop -- samples
       count:=1;
       while not(end_of_file(input_buffer))and count <= 32 loop --channels
          get(input_buffer,I_sample);
          get(input_buffer,Q_sample);
```

```
         multi_carrier_1i(count).sample_i(I_sample,Q_sample,count);
         multi_carrier_1q(count).sample_q(I_sample,Q_sample,count);
         multi_carrier_2(count).sample_data(I_sample,Q_sample,count);
         count:=count+1;
      end loop;
   end loop;
   close(input_buffer);
end;
```

REFERENCES

[1] S. J. Campanella and S. Sayegh, "Onboard multichannel demultiplexer/ demodulator," NASA contract no. cr1801827, 1987.

[2] S. C. Kwatra and R. Bexten, "Analysis and design of a burst mode digital demodulator implemented using a digital signal processor," Report DTVI-22, Electrical Engineering Department, The University of Toledo, Dec. 1988.

[3] S. C. Kwatra and A. A. Thanawala, "FDMA/TDM conversion for non-contiguous carriers," Report DTVI-23, Electrical Engineering Department, The University of Toledo, March 1989.

[4] K. Ohtani and S. Kato, "An onboard digital demodulator for regenerative SCPC satellite communication systems," IEEE International Conference on Communications, pp. 1803-1808, 1986.

[5] K. Betaharon, K. Kinuhata, P. P. Nuspl and R. Peters, "On-board processing for communication satellites: technologies and implementations," International Journal of Satellite Communications, Vol. 5, pp. 139-145, 1987.

[6] E. Del Re and R. Fantacci, "Alternatives for onboard digital multicarrier demodulation," International Journal of Satellite Communications, Vol. 6, pp. 267-281, 1988.

[7] W. Yim, C. C. D. Kwan, F. P. Coakley and B. G. Evans, "Multicarrier demodulators for onboard processing satellites," International Journal of Satellite Communications, Vol. 6, pp. 243-251, 1988.

[8] F. Ananasso and E. Del Re, "Clock and carrier synchronization in FDMA/TDM user-oriented satellite systems," IEEE International Conference on Communications, Seattle, Washington, pp. 1473-1477, June 7-10, 1987.

[9] F. Ananasso and E. Saggese, "User-oriented satellite systems for the 1990"s," 11th AIAA Communication Satellite Systems Conference, San Diego, California, pp. 1-11, March 16-20, 1986.

[10] T. Ohsawa and J. Namiki, "Digital group demodulation system for multiple PSK carriers," 11th AIAA Communication Satellite Systems Conference, San Diego, California, pp. 313-320, March 16-20, 1986.

[11] G. Perrotta, G. Losquadro and R. Giubilei, "Satellite communication systems for domestic/business services," International Journal of Satellite Communications, Vol. 5, pp. 85-103, 1987.

[12] A. J. Viterbi and A. M. Viterbi, "Non-linear estimator of PSK modulated carrier phase with application to burst digital transmission," IEEE Transactions on Information Theory, IT-29, pp. 543- 551, 1983.

[13] F. M. Gardner, "A BPSK /QPSK timing error detector for sampled receivers," IEEE Transactions on Communications, COM-34, pp. 423-429, 1986.

[14] P.J. Fernandes, L.P. Eugene, S.C. Kwatra, M.M. Jamali and J. Budinger, "A parallel pipelined architecture for a digital multicarrier demodulator," 13th AIAA International Communications Satellite Systems Conference,

[14] P.J. Fernandes, L.P. Eugene, S.C. Kwatra, M.M. Jamali and J. Budinger, "A parallel pipelined architecture for a digital multicarrier demodulator," 13th AIAA International Communications Satellite Systems Conference, pp. 285-294, March 11-15, 1990.

[15] J. G. Proakis and D. G. Manolakis, *Intoduction to digital signal processing*, MacMillan Publishing Company, New York, 1988.

[16] S. C. Kwatra and M. J. Vanderaar, "Trellis coded modulation for satellite-based mobile communications," Report DTVI-25, Electrical Engineering Department, The University of Toledo, August1989.

[17] L. Kronsjo, *Algorithms: Their complexity and efficiency*, John Wiley and Sons, Inc., New York, 1987.

[18] Selim G. Akl, *The design and analysis of parallel algorithms*. Prentice-Hall Inc., Englewood Cliffs, New Jersey 1989.

[19] K. Yano, T. Yamanaka, T. Nishida, M. Saito, K. Shimohigashi and A. Shimizu, "A 3.8-ns CMOS 16X16-b multiplier using complementary pass-transistor logic," IEEE Journal of Solid-State Circuits, pp. 388-394, April 1990.

[20] B. Gabillard, T. Ducourant, C. Rocher, M. Prost and J. Maluenda, "A 200-mW GaAs 1K SRAM with 2-ns cycle time," IEEE Journal of Solid-State Circuits, pp. 693-698. Oct. 1987.

[21] S. T. Chu, J. Dikken, C. D. Hartgring, F.J. List, J. G. Raemakers, S. A. Bell, B. Walsh and R. H. W. Salters, "A 25-ns low power full-CMOS 1-Mbit (128K X 8) SRAM," IEEE Journal of Solid-State Circuits, pp.1078-1084, Oct. 1988.

[22] F. Guibaly and B. McKinney, "A flexible pipeline architecture for digital signal processors," IEEE Conference on Communications, Computers & Signal Processing, pp. 370-373, 1987.

[23] B. W. Smith and H. J. Siegel, "Models for use in the design of macro-pipelined parallel processors," IEEE Journal on Computer Architecture, pp. 116-123, 1985.

[24] Ewing Lusk, *Portable programs for parallel processors*. Holt, Rinehart & Wintson, Inc, New York, 1987.

[25] D. J. Kuck, *The structure of computers and computations Vol-I*. John Wiley & Sons, Inc., New York, 1978.

[26] K. Hwang and F. A. Briggs, *Computer architecture and parallel processing*, McGraw Hill Inc, New York, 1984.

[27] J. P. Hayes, *Computer architecture and organization*, McGraw Hill Inc., New York , 1988.

[28] Steven Brawer, *Introduction to parallel programming*. Academic Press Inc., San Diego, California, 1989.

[29] G. J. Lipovski and M. Malek, *Parallel computing*. John Wiley & Sons Inc., New York, 1987.

[30] S. Dasgupta, *The design and description of computer architectures*. John Wiley & Sons, Inc., New York, 1984.

[31] L.P. Eugene, "Parallel port interprocessor communications for 80386 based systems," internal report National Aeronautical Laboratory, Bangalore, India, June 1988.

[32] L.N. Bhuyan and D.P. Agrawal, "Generalized hypercube and hyperbus structures for a computer network," IEEE Transactions on Computers, Vol C-33, pp. 323-333, April 1984.

[33] S. Abraham and K. Padmanabhan, "Performance of the direct binary n-cube network for multiprocessors," Proceedings of Conference on Parallel Processing, pp. 636-639, August1986.

[34] J.P. Hayes, T.N. Mudge, Q.I. Stout, S. Colley and J. Palmer, "Architecture of a hypercube supercomputer", Proceedings of Conference on Parallel Processing, pp. 653-660, Aug 1986.

[35] L.P.Eugene, D.Kaur and S.C.Kwatra, "Performance of SCPC/FDMA models on hypercubes," Proceedings of the 21st Annual Modeling and Simulation Conference, May, 1990.

[36] Whiddett, *Concurrent programming for software engineers.* John Wiley & Sons, Inc., New York, 1987.

[37] H. Schildt, *C:The complete reference,* Osborne McGraw Hill, New York, 1987.

[38] *Reference Manual for the Ada programming Language,* ANSI/MIL-STD-1815A-1983, Springer Verlag, 1983.

[39] Henry F. Ledgard, *Professional Software: Programming Practice.* vol-2, Addison Wesley, 1987.

[40] J.G.P. Barnes, *Programming in Ada.* Addison Wesley, 1989.

[41] Henry F. Ledgard, *Ada: An introduction.* 2nd edition, Springer Verlag, 1983.