The Broadband ISDN Source Simulator

Zhijie Tan, B.E.(Hons.)

Being a thesis submitted for the

Degree of Master of Engineering Science

in the Department of Electrical and Electronic Engineering

The University of Adelaide, South Australia.

February 1991.

Abstract

This thesis describes the development of a source simulator which generates parallel streams of packet traffic for a real time hardware testbed system which is used for experiments, studies, and evaluations relating to the Asynchronous Transfer Mode (ATM) networks. The ATM is a new form of switching and transmission for the broadband Integrated Services Digital Network (ISDN). The development includes both hardware design and software design to complete the source emulator board. This source board which operates under the control of a system control computer, is to generate parallel packet streams in several ways to simulate packet video sources.

In chapter 1, we give an introduction of the testbed project. Chapter 2 describes the design requirements of the testbed and the source generator. The hardware and software designs of the source generator are described in chapter 3 and chapter 4 respectively. Chapter 5 gives how to download a program to the microcontroller. In chapter 6 we give the test and debug of the source board. Conclusions on the use and future developments are given in chapter 7.

By testing the hardware and software of the source board, we give the performance evaluation of the source generator. A number of programs are developed for these tests.

Contents

Acknowledgements ix

Declaration x

1 Introduction 1

1.1 Why do we develop the testbed? 1

1.2 The testbed system 2

1.3 The source generator section 5

1.4 The software models 5

1.5 Elementary source generator 8

2 The Simulation Requirements 10

2.1 The experiment requirements 10

2.2 The source generator requirements 11

2.2.1 The packet requirements 12

2.2.2 The speed requirements 13

3 **Hardware Architecture** **14**

3.1 Overview of the design 14

3.2 The microcontroller 18

3.2.1 Features 19

3.2.2 Functional Aspects of the Microcontroller 19

3.3 Clock and the frame pulse 26

3.4 The random number generator 28

3.4.1 General 28

3.4.2 The mathematical model 31

3.4.3 Initialization 31

3.4.4 Normal operation 33

3.4.5 The hardware design 34

3.5 The traffic generation 36

3.5.1 Packet generation 36

3.5.2 The packet generator 37

3.5.3 The parameter generator. 44

3.6 Interfaces 48

3.6.1 The control computer interface. 48

3.6.2 The packet interface 52

3.7 The control route 62

3.8 The data transfer route 65

4 Software 68

4.1 Introduction 68

4.1.1 Packet video model 69

4.2 Programming 72

4.2.1 Gaussian random variables 73

4.2.2 The source model algorithm 74

5 Program Download 79

5.1 Introduction 79

5.2 Program download support circuit 81

5.2.1 Intersil ICL232 RS-232 to TTL Level Converter 81

5.2.2 The reset circuitry 82

5.2.3 The crystal driver 82

5.2.4 The MC68HC11A8 microcontroller 83

5.3 The software in the host computer 84

6 The Source Generator Debug and Test 86

6.1 General 86

6.2 The IV-1602 computer 87

6.3 IMON-68 debug monitor 92

6.3.1 Command line formats 92

6.3.2 IMON68 standard monitor command set 92

6.4 Instruction set summary 93

6.5 Software 93

6.6 Debugging and testing 98

6.6.1 Interfaces 98

6.6.2 Program download 98

6.6.3 Debug 100

6.7 Test 104

6.7.1 The software test 105

6.7.2 The packet generation test 109

7 Conclusions **116**

7.1 General 116

7.2 Future work 117

Appendices

3.1 Circuit Diagrams

4.1 Video Source Simulator 6811 Program Listing

5.1 MC68HC11A8 Test Board Circuit Diagram

6.1 Hardware Test Program Listings

6.2 Functional Test Program Listings

List of Figures

1.1 Structure of restriction queue and testbed 4

1.2 The source section 6

3.1 The source generator 15

3.2 Layout of the source generator 16

3.3 MC68HC11A8 microcontroller block diagram 20

3.4 The microcontroller connection 22

3.5 The microcontroller read/write timing 27

3.6 The clock and the frame pulse generators 29

3.7 The clock and the frame pulse timing 30

3.8 The random number generator 32

3.9 Random number latch timing 34

3.10 The packet header format 38

3.11 The packet generator 39

3.12 The packet distribution in the comparator and counter modes 40

3.13 The counter mode diagram 43

3.14 Timing of the packet generator 45

3.15 The parameter generator 47

3.16 The computer interface 50

3.17 The control register 51

3.18 The computer interface write/read timing 53

3.19 The VMEbus connector pin assignments. 54

3.20 The packet output interface 56

3.21 Standalone operation timing 57

3.22 Bus operation timing 60

3.23 Pin allocation for packet interface connector 61

3.24 The control route 63

3.25 The data transfer system 66

4.1 Coding bit rate 70

4.2 Bit rate histogram 71

4.3 Autocovariance function 72

5.1 The download block digram 80

5.2 ICL232 81

5.3 The reset circuitry 82

5.4 Crystal driver 83

6.1 The debug and test block diagram 88

6.2 The IV-1602 VMEbus single board computer 89

6.3 Block diagram 90

6.4 Connections 99

6.5 The bit rate histogram 109

6.6 The bit rate autocovariance function 110

6.7 The test diagram 111

6.8 The rate of packet generation in the comparator mode 114

6.9 The rate of packet generation in the counter mode 115

List of Tables

1.1 Service characteristics 7

3.1 Operating modes versus MODA and MODB 23

3.2 Bootstrap mode interrupt vectors 24

6.1 Generic monitor command descriptions 94

6.2 The IV-1602 memory map 96

6.3 Operating modes versus control number 103

Acknowledgements

I would like to thank my supervisor, Dr. K. W. Sarkies, for his help, advice and boundless enthusiasm, and guidance received during the course of this project. Also many thanks to a number of staff, engineers and technicians in our department for their assistance during the period of this work.

I would also like to thank Mr Linh Nguyen and some final year students who provide assistance for this project.

Declaration

This thesis has been submitted to the Faculty of Engineering at the University of Adelaide for examination in respect of the Degree of Master of Engineering Science (by research).

This thesis contains no material which has been accepted for the award of any other degree or diploma in any University and that, to the best of the author's knowledge and belief, the thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

The author hereby consents to the thesis being made available for photocopying and loan if accepted for the award of the degree.

Zhijie Tan

February 20, 1991

Chapter 1

Introduction

The source generator, as a major part of the testbed system, is developed in this project. In the first chapter we would like to introduce the background of the testbed system. At first, we will give a brief discussion of the genesis of the testbed project [1], an overview of the testbed system, then an outline of the source section and a brief description of the source models.

1.1 Why do we develop the testbed?

The Integrated Services Digital Network(ISDN) is a concept of worldwide interest, offering the prospect of new voice and data services, which are based on the availability of high speed digital transport [2]. Asynchronous Transfer Mode (ATM) is a new form of switching and transmission which has been proposed for the broadband ISDN telecommunications. ATM networks are expected to find wide use for their inherent flexibility, service independence, and high performance [3]. This concept involves digitization and packetization of the different services, and transmission and switching at very high throughputs and low delays. While the ATM has been selected by CCITT

standards committees for implementation of the broadband ISDN, a number of outstanding problems remain to be solved. These involve the selection of hardware structures for high speed switching and buffering of packets, and protocols and algorithms for control of congestion and routing within the network. The performance of the network is critically dependent on a correct selection of these aspects. Since many of the control structures will be implemented in hardware, an unsuitable selection may result in a poorly performing or excessively expensive network which will be very difficult to change. This results in a peculiar problem with regarding to the effective study of suitable hardware and protocol structures. Conventional simulation is likely to encounter problems due to the very high speed and large amount of packet traffic to be simulated. For example, performance of a proposed call acceptance algorithm for a network may need to be studied over a period of hours in real time, while the actual traffic in the network will have a throughput of million of packets per second. Also there are many issues such as packet loss rates, packet transmission delays, and other effects of congestion which need to be studied in order to find suitable algorithms for acceptance of new calls into the network.

Because of these problems, we are building a real time hardware testbed for the studies of the performance of suitable structures for the ATM networks, and of algorithms for congestion control at all levels.

1.2 The testbed system

The experiments to be performed on the testbed are characterized by the need for a high speed of simulation. The testbed provides common simulation elements such source generation, event capture and time measurement, and

some specialized components for the system under test. The testbed consists of a packet source generator section and a measurement section as shown in Figure 1.1 (from [1]). The source generator section consists of:

(1) A control computer. This is responsible for initialization and parameter modification of the elementary sources, and overall control of the experiment including startup, completion, and high level simulation of parts of the packet generation model. The Ironics IV-1602 single board computer based on the Motorola 68010 microprocessor is selected for this purpose.

(2) A source group. This consists of elementary source generators and possibly some real sources. Their outputs are mixed in the multiplexer to produce a final output stream.

(3) A shared bus to the statistical multiplexer. This allows the sources to transfer their generated packets to the multiplexer.

(4) A statistical multiplexer which combines the packets into a single combined stream. The output of the multiplexer is limited to the maximum rate of an elementary source. Thus it is possible to use a single elementary source, or a multiplexed stream of sources each operating at a lower rate, but in combination operating at a maximum rate.

(5) A serial output for certain network experiments.

The measurement section [4] is connected to the experiment to measure event time and to count events. It consists of:

(1) Event capture. This is used to capture the packet arrival time and the packet header from a packet stream. It may also include event counters. All the data is passed to the datalogger computer for processing and logging.

(2) A datalogger computer. This stores and processes the data from the event capture boards.

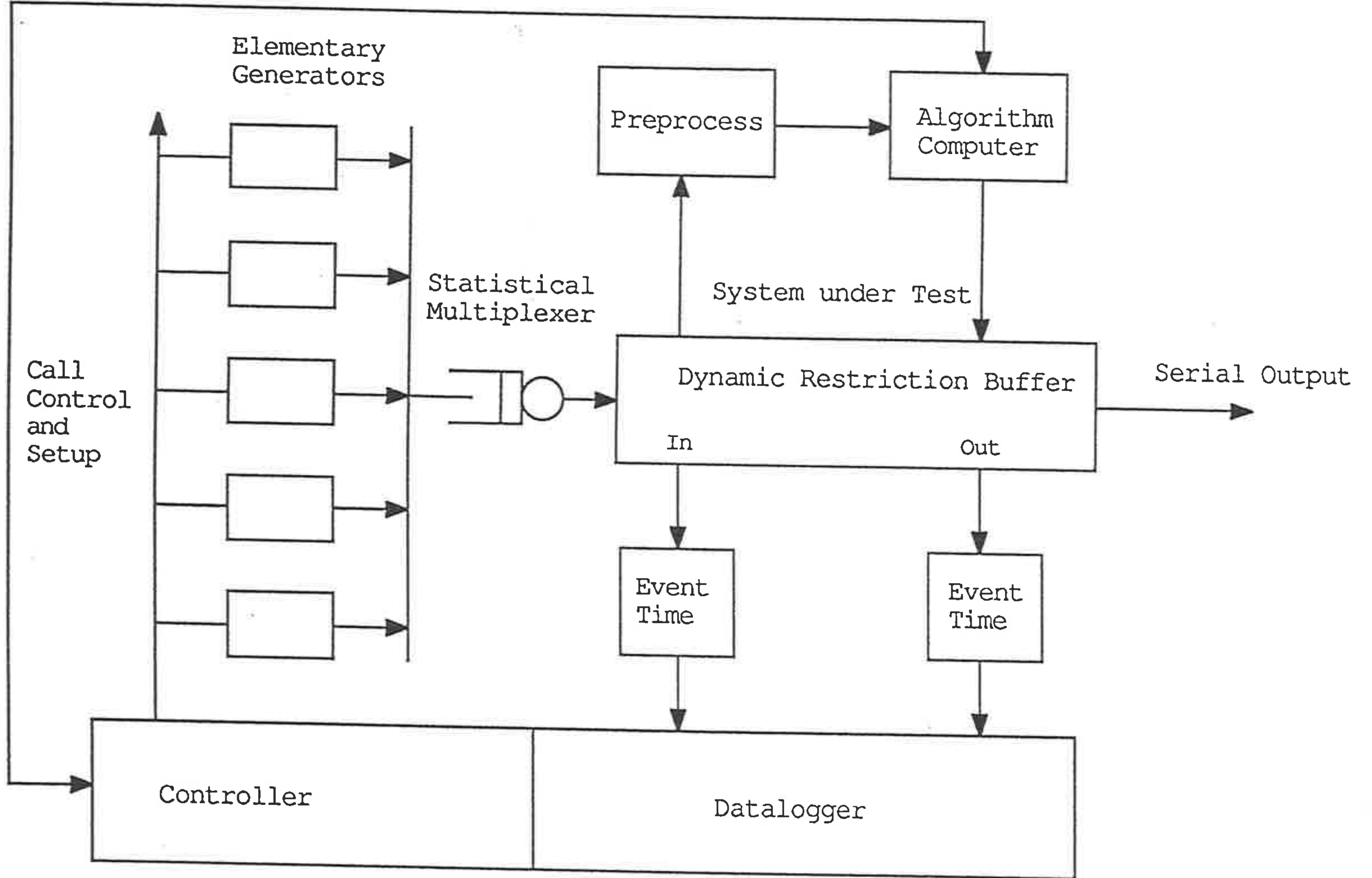


Figure 1.1: Structure of restriction queue and testbed

1.3 The source generator section

In an ATM network, the traffic arises from the combining of traffic from a large number of sources using different network services in different ways. These services include voice, video, image, interactive computer, database access, file transfer and other more specialized services. These services will be digitized and packetized. Each of these services has its own peculiar characteristics and requirements. In the case of interactive video, a service considered to dominate a future network, the packets are transferred in fixed period video frames using variable length encoding technologies. Each frame thus has a variable number of packets. As shown in Figure 1.2, the source section will generate traffic for simulating the ATM traffic. In order to generate traffic with realistic characteristics, we need a number of elementary source generators. Each of these elements is only required to generate one type of packet stream. The external control computer controls the source elements by switching sources on or off, and modifying the characteristics of each source element by changing some parameters. The generators feed their outputs to a statistical multiplexer which superposes the outputs to produce a final output stream.

1.4 The software models

Two important characteristics of the new services of the ATM networks are the natural bit rate and burstiness of the information [2]. The natural bit rate for each service is the rate at which the information would be processed if delays due to communication were not present. This is the rate that would be desired from an ideal communication system if cost were not an issue. In

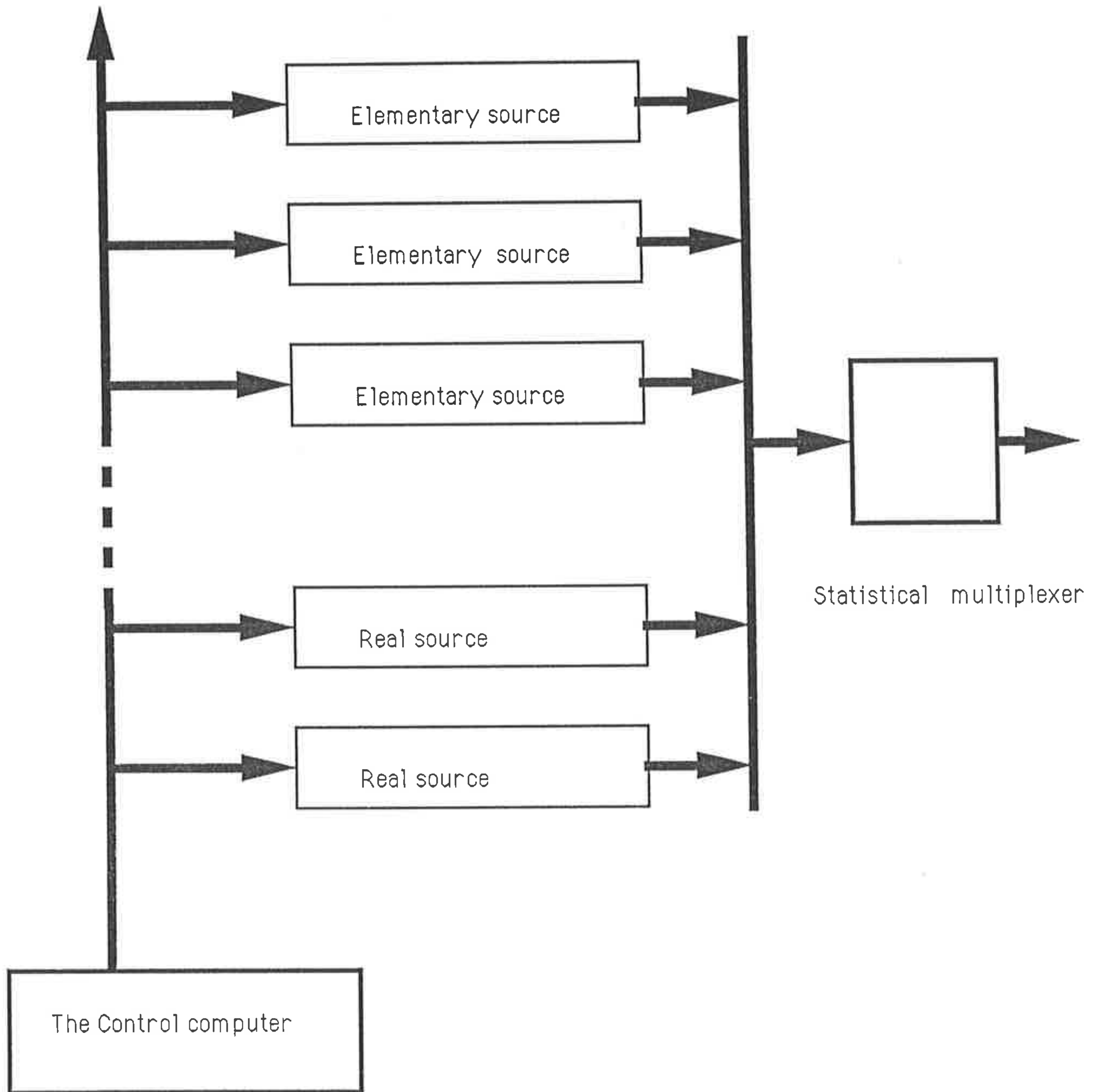


Figure 1.2: The source section

Service	Natural Rate	Burstiness
Voice	4Kb/s-64Kb/s	2-3
Interactive Data	1Kb/s-100Kb/s	>10
Bulk Data	1Mb/s+	1-10
Telemetry	<10Kb/s	>10
Image	10Kb/s-1Mb/s	1-10
Conference	1Mb/s	1-5
Vide0	>10Mb/s	1-2

Table 1.1: Service characteristics

determining the natural bit rate of an encoded signal such as voice or video, we must take into account advances in signal processing that allows economically feasible compression coding to lower the bit rate needed. Burstiness is the ratio of the peak bandwidth needed carry information at the natural rate, to the long term average bandwidth actually used. Burstiness is generally greater than 1 because the information source is not continuous, or does not maintain its peak bit rate over the duration of a call. Rough estimates of the values of these parameters for some current and future services are given in Table 1.1 (taken from [2]). Traffic models are developed based on the characteristics of service bit rate and burstiness. For example, voice will consist of packets transmitted at bit rates from 4 Kb/s to 64 Kb/s and burstiness at 2-3. Video will consist of variable length bursts at fixed frame intervals with peak bit rate greater than 10 Mb/s and burstiness 1-2. Other services may also be simulated based on their bit rate and burstiness characteristics.

Among these source elements, the packet video source is the most important and complicated one. The video traffic is expected to become a major component on the broadband ISDN. Applications such as video conference, video telephone, and entertainment video are expected to become more and

more popular on public and private networks. As such we have concentrated on modeling video sources, currently under investigation is a source generator for encoded packet video, based on the model proposed by Maglaris [5]. In his model, video frames are transmitted using variable length encoding techniques. A more detailed discussion about modeling of the video source is given in Chapter 4.

1.5 Elementary source generator

The elementary source generator may be designed in a number of ways depending on the type of traffic to be simulated. A model of a traffic source may consist of layers of models [1], each of which may modify the parameters of the model layer below. We consider here a three layer model consisting of a high speed packet generator at the lowest layer, an intermediate layer which modifies the rate of packet generation, and a high level layer which turns the sources on or off.

At the lowest layer, because of limitation of the hardware structure, the packet generation may be done only in two ways. For "soft" traffic, the frame model may be devised to specify a packet rate for generation of randomly distributed packets, which allows the packets to be spread over the whole frame interval. For bursty traffic, the frame model may specify a packet count for generation of packets in a single burst at the start of a frame. In here the frames are relevant to a video source. For other sources, the frames can be ignored. Otherwise the frames are a restriction on the type of traffic, for example, bursts can only be integral number of frames long.

The intermediate layer modifies the rate of packet generation. This is done by using a microcontroller which generates a rate parameter updated

once per frame period to modify the rate of packet generation. This allows the simulation to be quite flexible, and as a result quite comprehensive models may be incorporated.

The highest layer is implemented by a external control computer which performs call simulation by switching the sources on or off, and setting simple parameters to modify the characteristics of each elementary source.

This 'three level' source model is quite flexible, allowing a number of models to be programmed simply by changing the on-board software. Each model has its particular characteristics, for example, a voice call will consist of packets transmitted at fixed time intervals, and switched on and off by a high level call or a burst model. An encoded video will consist of variable length bursts at fixed frame intervals, with correlations over long time periods. Basic models such as pure random, Markov modulated, switched Poisson, interactive computer, file transfer are also possible to be simulated. The sources can be implemented using a single hardware platform, and a set of application software. The only restriction for source simulations is that the basic changes in the packet generation rates will be limited by the "frame" rate, which itself is limited by the program cycle time of the microcontroller. If the frame rate is made variable, then a larger class of traffic models may be simulated [1].

Chapter 2

The Simulation Requirements

2.1 The experiment requirements

The testbed system is being developed to perform experiments and evaluations relating to ATM networks. This requires the testbed to provide common simulation elements and specialized components to build up the network structures for each experiment [1]. Following is an outline of some common elements needed for the experiments.

(1) The first experiment involves the study of the nature of traffic in an ATM network to assist in the creation of suitable traffic models. The traffic arises from the mixing and combining of traffic from a large number of sources using different networks in different ways. In order to generate traffic with realistic characteristics, we need to understand the nature of the individual traffic streams; that is, how each network user would use the services in terms of access rate, holding time, and proportion of different services. This requires the development of models for the sources, simulation of sources, and combination into a mixed stream. The system will measure the charac-

teristics of the combined packet streams to establish suitable models for the combination of the individual streams. These experiments require a number of elementary source generators and a multiplexer.

(2) The second experiment aims to identify the behaviour of networks and switch components under different types of traffic, which may come from a combination of sources as described above. This requires simple traffic models which have the overall characteristics of real traffic. These traffic models should have specific parameters which identify the important characteristics of the traffic. The number of these parameters must be minimal so that the performance of networks and switches may be meaningfully characterized and compared. This type of experiment may require a number of parallel streams of traffic. Other studies include the study of burstiness of the network performance to parameters of the traffic, and the effect of traffic loading on real connections through a network.

(3) The third experiment involves the implementation and study of long term congestion control based on call acceptance decisions using measured packet loss rates in network components, notably buffers. This requires provision of hardware for analysis of measured packet loss rates or buffer lengths, and an algorithm processor for implementation of the actual algorithm.

2.2 The source generator requirements

As a section of the testbed, the source section design must be subject to the experiment requirements. One important requirement of the experiments is to study high speed traffic networks. This requires that the source generator can generate high speed traffic. But when the speed is over 100 Mbit/s, the difficulty of generation will dramatically increase when the speed increases.

The problem of speed may be tackled by generating "virtual packets". If we only use the packet header which can contain all information required for the simulation to represent the entire packet, this will dramatically reduce the speed of the simulation. To further reduce the speed of the source simulator, the header may be transmitted in parallel form instead serial form within the testbed or the system under test. This would reduce the entire packet transmission time to only one clock cycle per packet.

2.2.1 The packet requirements

The sources in the testbed would generate "virtual packets" to simulate traffic of the ATM networks. The simulation requires that packets are transmitted as fixed length entities in time slots called *cells*, which is the basic transmission mode proposed for the ATM networks.

The length of packet header which contains all control information for an individual packet, depends on the requirements of the experiments. We considered that a 16 bit packet header would be sufficient to provide all control for simulations. One 8 bit field is control information relating to characteristics of the experiments; namely a 5 bit destination address (for switch experiments) and a 3 bit class number. The other 8 bit field is the packet identifier which consists of a 4 bit source identifier and a 4 bit packet sequence number. This is needed for the measurement section to follow the progress of a packet through an experimental system. The full real packet would consist of 64 octets or 512 bits (CCITT recommends 53 octets or 424 bits), but in the simulation the source generator will only generate a 16 bit header to represent an entire real packet. This will reduce the speed of the simulation to allow simulation of very high speed traffic. If a serializer is used to generate real packets from these headers, then a dummy information

field would be added.

2.2.2 The speed requirements

In ATM networks, the maximum speed of packet transmission for a single user, is expected to be about 130 Mbit/second to about 160 Mbit/s. If a real packet contains 512 bits (CCITT recommends 424 bits) for example, transmitted serially at 160 Mbit/s, this is equivalent to transmitting about 312 thousand packets per second. The source generator only generates a 16 bit header to represent an entire packet, and the header is transferred in 16 bit parallel form within the testbed and the system under test. Thus the generator requires only one clock cycle time to transmit one equivalent packet. To simulate a source with an effective bit rate of 160 Mbit/s, the source generator requires only a 312 KHz clock rate.

Chapter 3

Hardware Architecture

3.1 Overview of the design

In this chapter, we will describe the hardware design of a general source generator. Before starting the description, we would like give a brief discussion about how the source generator will operate in the testbed. As illustrated in Figure 3.1, the source generator will interface with the control computer and the statistical multiplexer. The control computer via the VMEbus performs the highest layer control to the source generator. The source generator will output packets to the multiplexer via the packet interface which consists of 4 control lines and 16 packet parallel output lines. These four control lines will determine the packet interface operation. A detailed description will be given in a later section titled 'Interfaces'.

The proposed architecture for the elementary source generator is shown in Figure 3.2. It consists of five major blocks. Each block performs its own specialized functions. Following paragraph will give a short description of each block function.

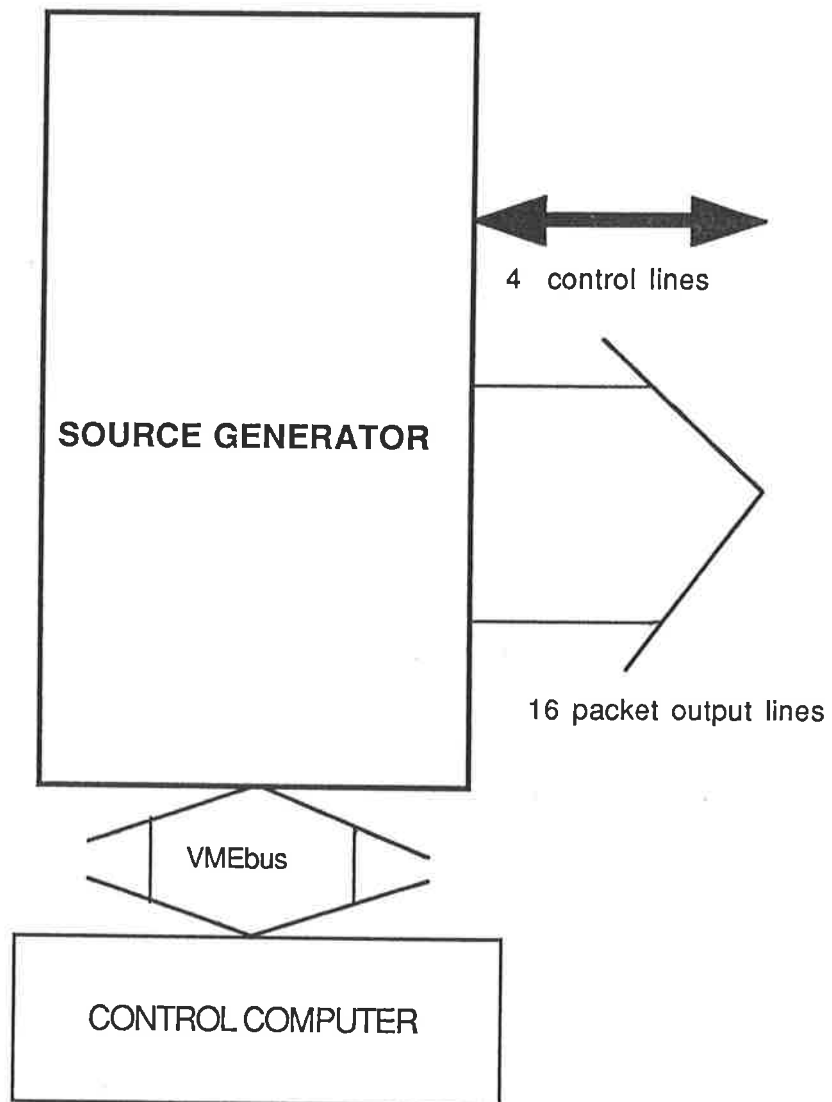


Figure 3.1: The source generator

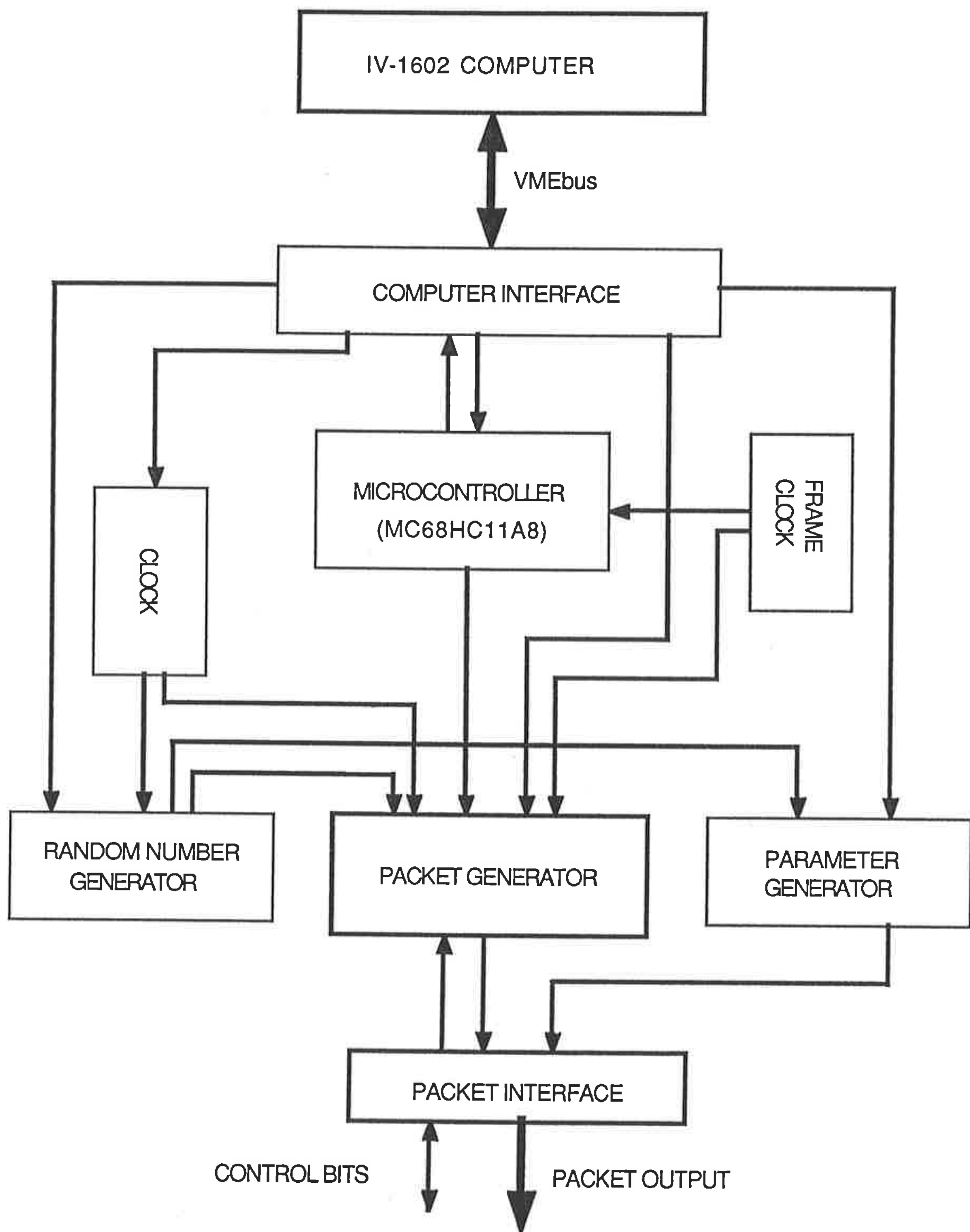


Figure 3.2: Layout of the source generator

(1). The microcontroller. This generates one parameter called the rate parameter which determines the rate of packet generation. It can communicate with the external control computer to allow the software in the microcontroller to be modified.

(2). The random number generator. This is a high speed pseudo random number generator. It outputs 16 consecutive bits to form two independent 8-bit random numbers in every 16 clock cycles. This consists of a feedback shift register sequence which takes the form of a maximal length sequence shift register. Maximal length sequences are known for a variety of shift register lengths. For our purpose, the sequence should have a cycle time of the order of a few months. Therefore, a 47-bit shift register sequence which gives a cycle time of 81 days with a 5 MHz clock is selected for this generator.

(3) The packet generator. This decides whether a packet is to be generated or not in a cell cycle, and does so at specified rates or probabilities. The rate of packet generation is determined by the rate parameter which is generated by the microcontroller. There are two generation modes which are comparator mode and counter mode which can be selected by the control bits. In the comparator mode, the "soft" traffic is generated, in which the packets are spread over the whole frame interval. In the counter mode, the bursty traffic is generated, in which the packet is generated in a single burst at the start of a frame.

(4). The parameter generator. This generates parameters for the 16 bit packet header parameter. These are 8 bit control information and 8 bit identification information. The control bits may determine whether the packet control information is provided by the external computer or generated internally. Typically these will be either fixed with a given class and destination, or according to a random choice in the destination or class or both. The iden-

tification information will only be generated internally. Each source board is allocated a 4 bit fixed source identification, and the packet identification consists of a 4 bit packet sequence number obtained by counting the generated packets.

(5). Interfaces. These are the computer interface and the packet interface. The control computer can access each source generator via a VMEbus. The address location of the board will be placed in the top 64K of the 68000 memory space, referred as the short VMEbus address space. Each board contains four short addresses so that the external control computer can communicate directly with the four registers on the board. The packet interface is a single output interface which consists of a 16 bit parallel packet output port and 4 control lines. It will be used to output packets to the statistical multiplexer.

The circuit diagram is given in APPENDIX 3.1. The following sections describe the detailed design of this source generator.

3.2 The microcontroller

In order to provide a flexible general purpose source simulator, a microcontroller that can be reprogrammed is used. Because the characteristics of the traffic depend on the rate of packet generation, this microcontroller is required to generate a corresponding rate parameter. For our purpose, we have selected the MC68HC11A8 microcontroller [6].

3.2.1 Features

The MC68HC11A8 is a high density CMOS microcontroller unit which contains highly sophisticated on-chip peripheral capabilities. This high speed and low power microcontroller has a nominal bus speed of 2 MHz, and the fully static design allows operation at frequencies down to dc. Refer to the block diagram in Figure 3.3 (from [6]) for the hardware features. The list below are some features which are relevant to the source generator.

1. 512 Bytes of EEPROM.
2. 256 Bytes of Static RAM.
3. Enhanced NRZ Serial Communications Interface(SCI). This port will be used to download the code to the EEPROM.
4. 8 Bits Output Port(port B). This is used as parallel output port. The rate parameter is output from this port.
5. 8 Bits Input/Output Port(PC). This is configured as parallel input port. The control computer transfers data via this port to modify the program operation.

This microcontroller has adequate capabilities to meet the design requirements for this source board.

3.2.2 Functional Aspects of the Microcontroller

The function of the microcontroller is to control the packet generation. In order to achieve this aim, the microcontroller will generate a rate parameter which is produced by the program contained in its EEPROM.

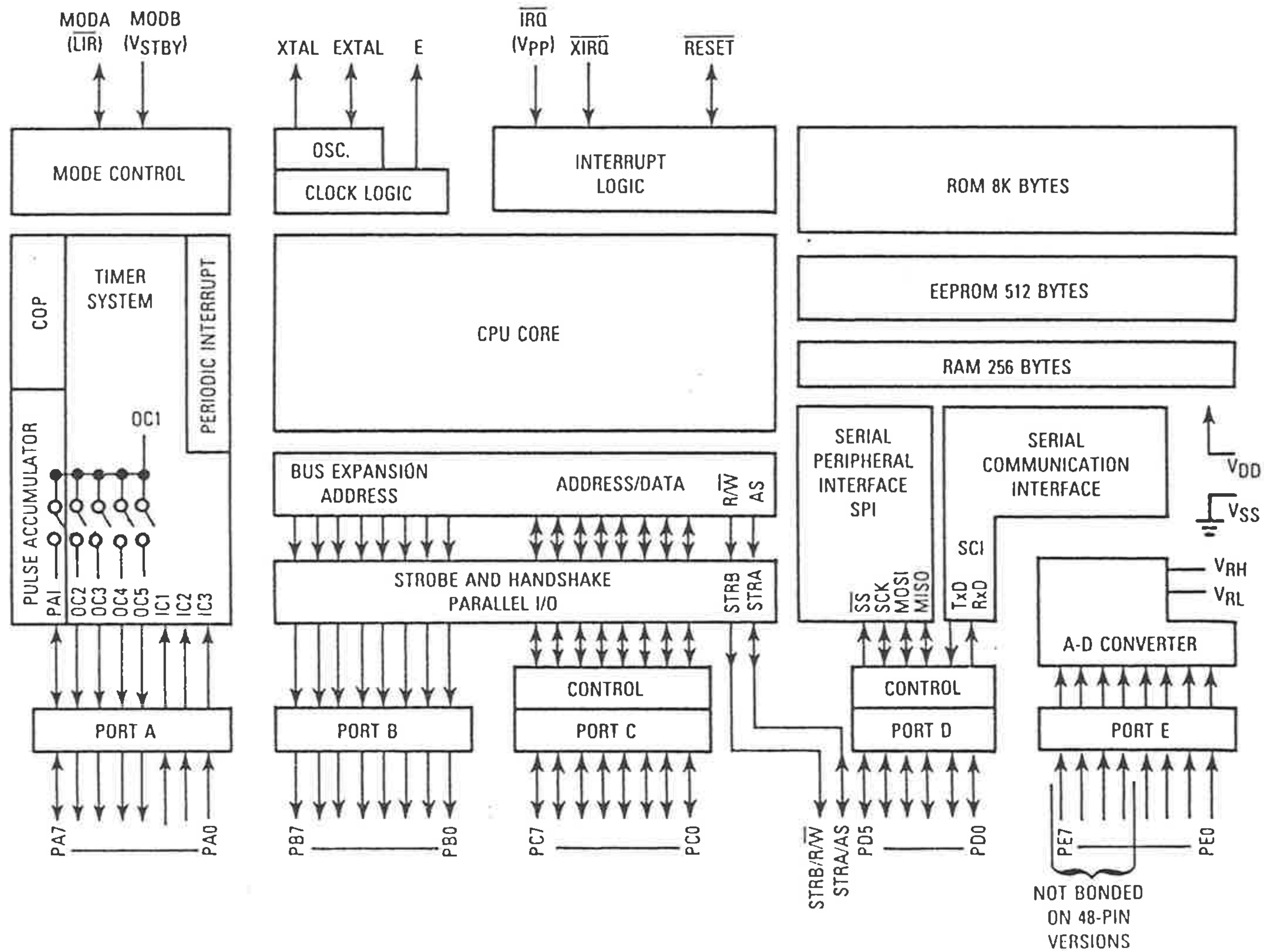


Figure 3.3: MC68HC11A8 microcontroller block diagram
(Motorola [6])

Port B is a strobed output port with the STRB line as the output strobe for the rate parameter. Port C which is configured as a parallel input port with the STRA line as the edge-detecting latch command input, is used to accept data from the control computer.

The block diagram is shown in Figure 3.4. The control computer communicates with the microcontroller via the VMEbus. It can write data to the port C to modify the program operation, and read the rate parameter from the port B.

The FRAME PULSE* which is from the frame clock section (to be described later) is passed to the XIRQ* pin to request an interrupt. The interrupt service program will then write the rate parameter to the port B. This will cause the rate parameter to be updated every frame cycle.

The operating mode

MC68HC11A8 microcontroller has four operating modes available: single-chip operating mode, expanded multiplexed operating mode, special bootstrap operating mode, and special test operating mode. Refer to Table 3.1 (from [6]), two dedicated pins (MODA and MODB) are used to select one of these modes. During reset, MODA and MODB are used to select one of the four operating modes.

The special bootstrap mode is the most suitable for our purpose. The single chip mode requires internal mask programmed ROM to be set, while the expanded mode requires external memory and I/O ports, also the test mode is primarily for main production at time of manufacture.

The special bootstrap mode is a very versatile operating mode since there are essentially no limitations on the program that can be loaded into the

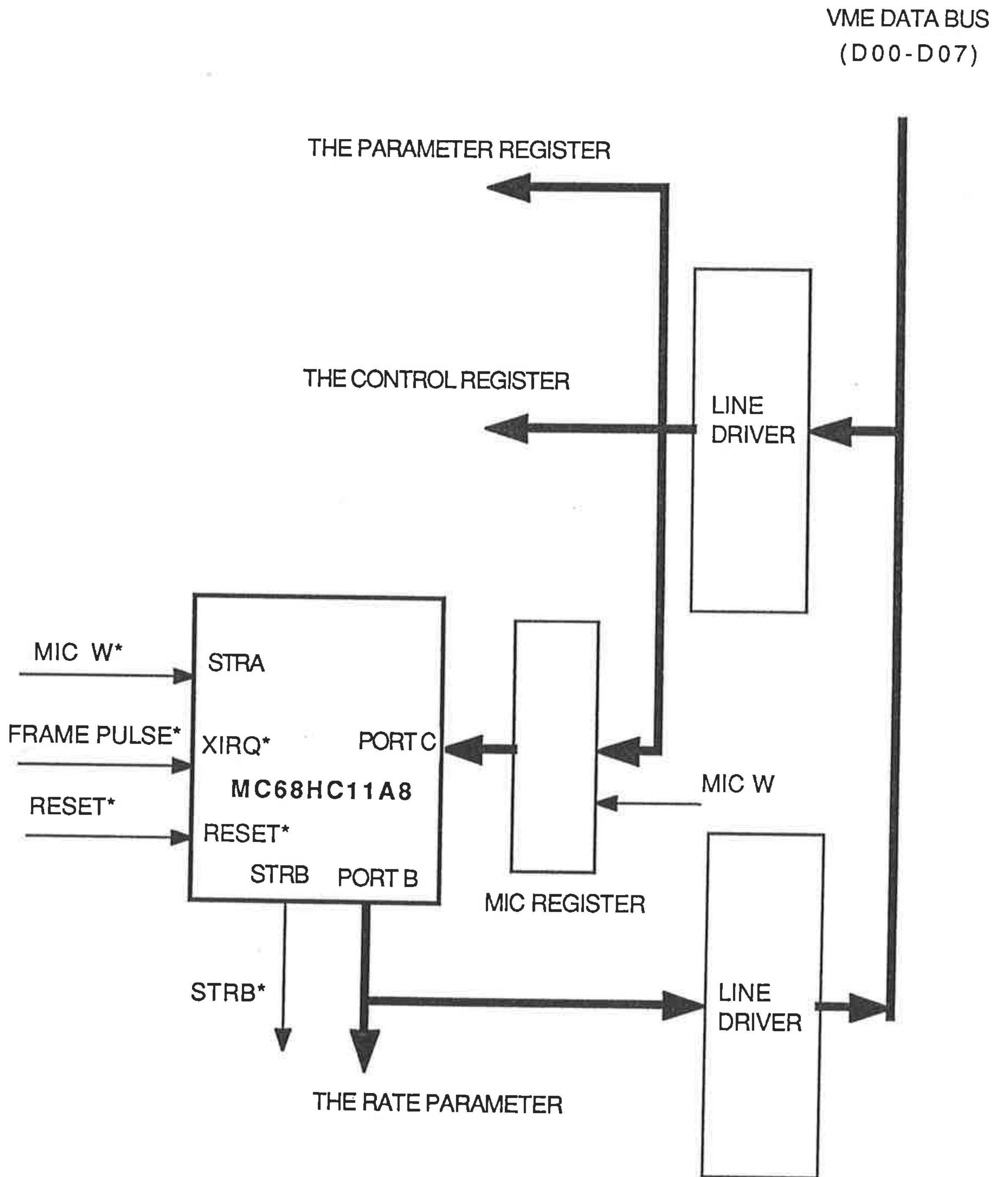


Figure 3.4: The microcontroller connection (App 3.1 Sh. 4)

MODB	MODA	Mode Selected
1	0	Single Chip
1	1	Expanded Multiplexed
0	0	Special Bootstrap
0	1	Special Test

Table 3.1: Operating modes versus MODA and MODB (from [6])

internal RAM or EEPROM. The boot loader program is contained in a 192-byte on-chip bootstrap ROM which appears as internal memory space at locations $\$BF40$-$\$BFFF$. The boot loader program will use the serial communication interface(SCI) to read a 256 byte program into on-chip RAM at locations \$0000-$\$00FF$. After the character for address $\$00FF$ is received, control is automatically passed to that program at location \$0000.

In special bootstrap operating mode the interrupt vectors are directed to RAM as shown in Table 3.2 (from [6]). This allows use of interrupts by way of a jump table. For example, to use the IRQ interrupt, a jump instruction would be placed in RAM at locations $\$00EE$, $\$00EF$, $\$00F0$. When an IRQ is encountered, the vector (which is in the boot loader ROM program) will direct program control to location $\$00EE$ in RAM which in turn contains a JUMP instruction to the interrupt service routine in the EEPROM.

If we tie the receiver to the transmitter (with an external pull-up resistor), then following a reset, the program will jump directly to the beginning of EEPROM rather than to the boot ROM. If the program has already been downloaded to EEPROM, then the microcontroller would begin executing that program.

Address	Vector
00C4	SCI
00C7	SPI
00CA	Pulse Accumulator Input Edge
00CD	Pulse Accumulator Overflow
00D0	Timer Overflow
00D3	Timer Output Compare 5
00D6	Timer Output Compare 4
00D9	Timer Output Compare 3
00DC	Timer Output Compare 2
00DF	Timer Output Compare 1
00E2	Timer Input Capture 3
00E5	Timer Input Capture 2
00E8	Timer Input Capture 1
00EB	Real Time Interrupt
00EE	IRQ
00F1	XIRQ
00F4	SWI
00F7	Illegal Opcode
00FA	COP Fail
00FD	Clock Monitor
BF40 (Boot)	Reset

Table 3.2: Bootstrap mode interrupt vectors (from [6])

Ports configuration

In the bootstrap operating mode, port C can be used as a general purpose input or output port under direct control of its data direction register, while port B is a fixed direction output port. In our design, port B and port C are configured as output port and input port, respectively. The parallel ports B, C have two operation modes available which can be selected by the parallel I/O control register. One is simple strobed mode, and the other is full handshake mode.

In our design, port C is configured as the simple strobed input protocol with the STRA line as an edge-sensitive latch command input which can be configured. The control computer can transfer data into the microprocessor via port C. When the external computer writes data to the microprocessor, it drives the STRA line. The active edge on the STRA line will request an IRQ interrupt. The IRQ interrupt program will be designed to read the data from the port C and store it. This process enables the control computer to modify the program in the EEPROM of the microcontroller. The timing for the control computer writing data to the port C is given in Figure 3.5. The STRA is configured active LOW going edge. The control computer drives the STRA* LOW when the data to be written to the microcontroller is valid in the port C. The STRA* falling edge will request a IRQ interrupt, and the IRQ interrupt program will cause the port C to be read.

Port B is also configured to operate in the simple strobed mode. In this mode, the STRB line is a strobe output which is pulsed for two microcontroller clock (E clock) periods each time there is a write to port B. This pulse is configured active LOW. The port B output timing is given in Figure 3.5. When the XIRQ* pin is activated, the XIRQ interrupt program will cause the calculated rate parameter to be output to the port B. The STRB*

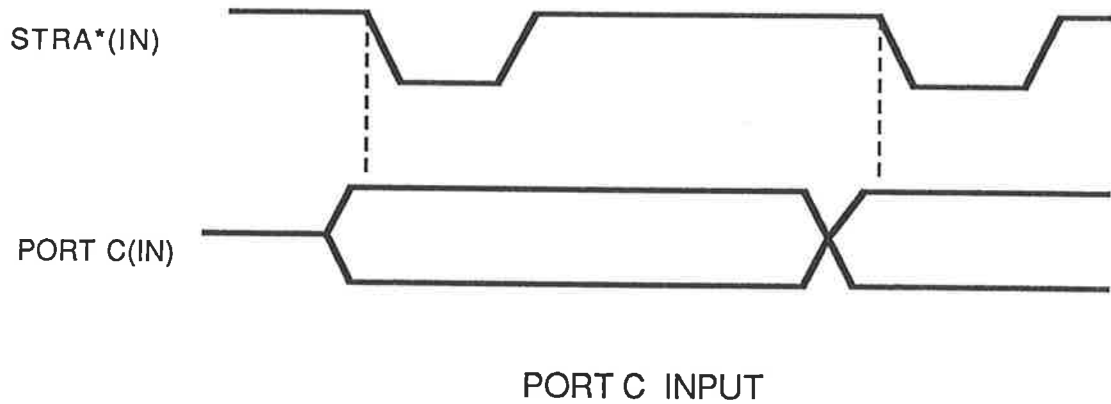
is pulsed LOW when the new data is available in the port B. The STRB* falling edge will drive the XIRQ* signal HIGH to remove the interrupt. The t_{DEB} is the delay time between new data valid to the STRB* falling edge. This is a maximum of 225 ns when the E clock is 2 MHz [6].

3.3 Clock and the frame pulse

The maximum rate of packet generation depends on the generator clock rate. In order to simulate high speed networks, this requires the clock frequency to be as high as possible. As a balance between the requirements of the simulation and the restriction of TTL ICs operation frequency, we select 5 MHz as the generator clock frequency. A 5MHz crystal oscillator is used to generate this clock.

One 74LS191 (u5) up counter is used to generated the RANDOM ENABLE* and COMPARATOR LATCH signals. The clock is passed to the CLK input pin of the counter. The Q_A and RCO pins of the counter will output COMPARATOR LATCH and RANDOM ENABLE* signals respectively. The COMPARATOR LATCH is passed to the packet generator as a latch signal. The RANDOM ENABLE* has one sixteenth clock frequency, which will be used as a random number latch signal. The block diagram is given in Figure 3.6, and timing is in Figure 3.7.

A frame signal is required for the simulated video frame generation. As shown in Figure 3.6, the CMOS MM5368 (u2) is employed to generate a 60 Hz square wave. A 74LS191 (u3) is used as divide-by-two divider, which gives a 30 Hz square wave. The 30 Hz square wave is provided to clock pulse input pin of a D Flip-Flop. At the rising edge of the 30 Hz square wave, the Q is set HIGH and the Q* output is set LOW. The Q* signal is passed to the XIRQ* pin



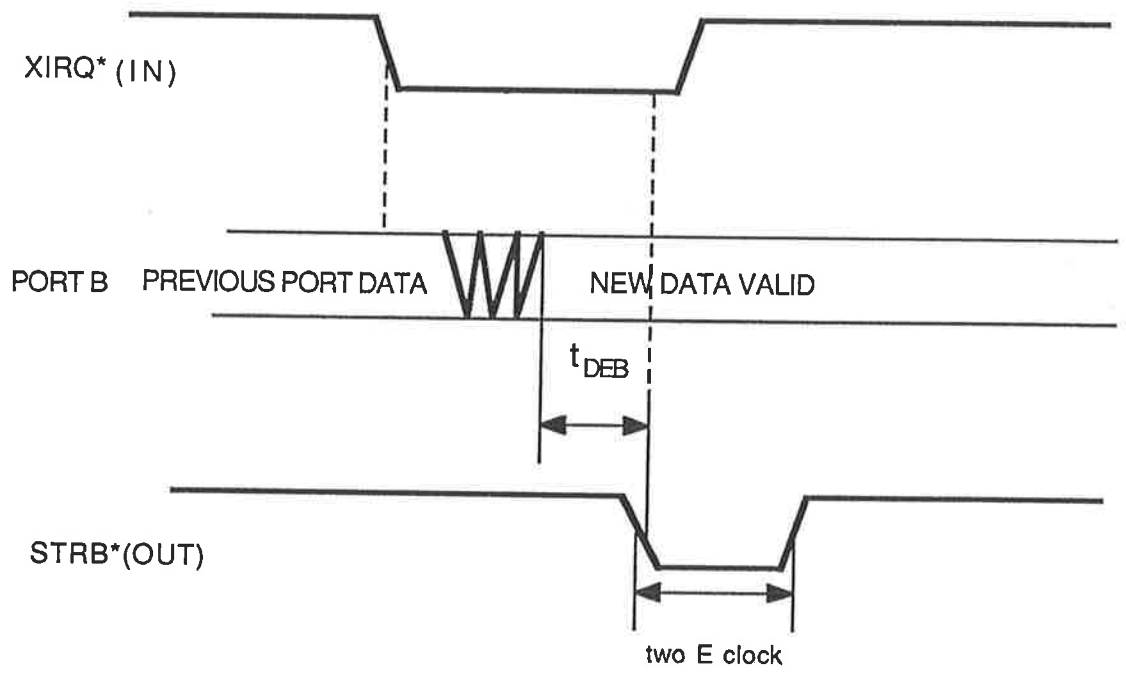


Figure 3.5: The microcontroller read/write timing

of the microcontroller to request an XIRQ interrupt. The interrupt service program will then write the rate parameter to the port B and will output a LOW active strobe pulse STRB* to the STRB pin. This pulse is used to clear the D Flip-Flop. Thus the Q is cleared to LOW, and it will remain LOW until the next rising edge of the 30 Hz square wave. The Q output is used as a frame pulse. The STRB* signal therefore has the same period as the frame pulse.

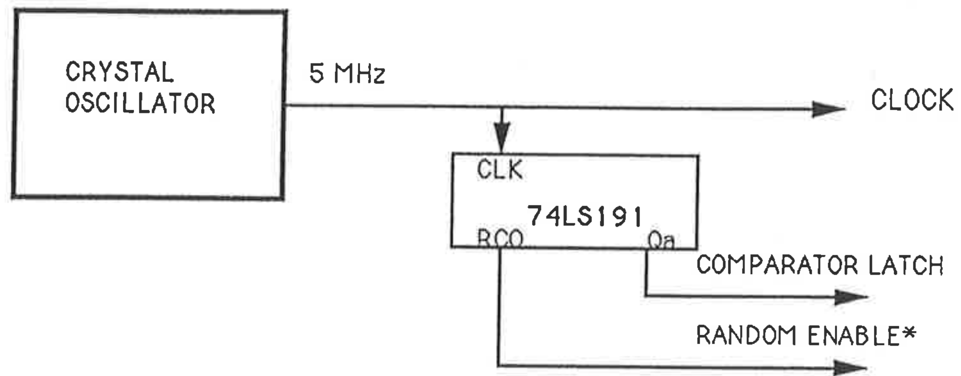
The timing is shown in Figure 3.7. The source generator clock frequency is 5MHz. The frame pulse frequency is 30 Hz.

3.4 The random number generator

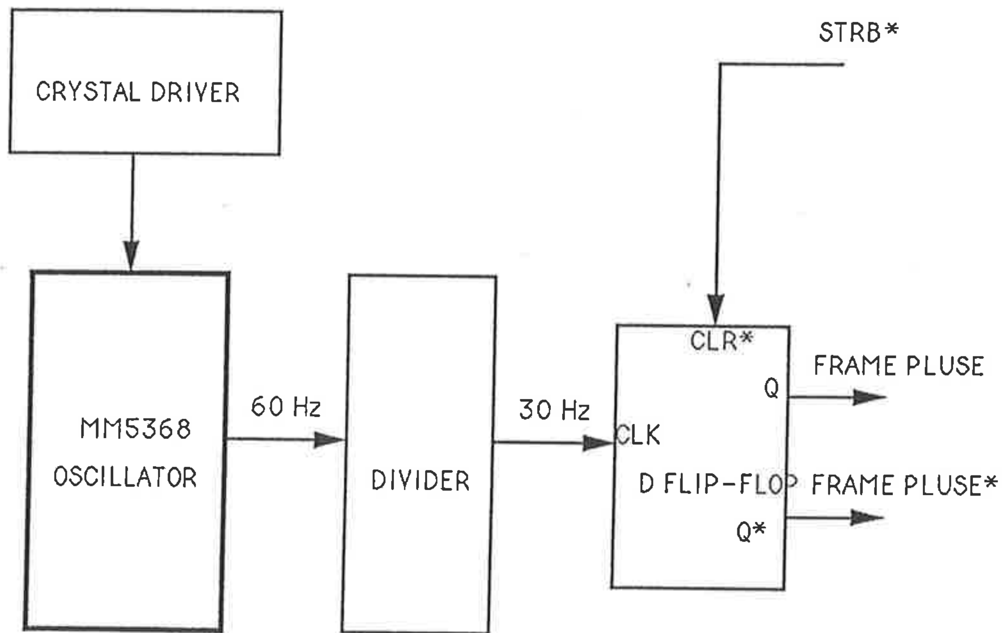
3.4.1 General

Two 8-bit random numbers are required for packet generation in every cell cycle. One is used as random packet header parameter, and the other is provided to the packet generator for the magnitude comparison. This gives rise to a need to develop a high speed, long period random number generator.

For this purpose, a 47 stage binary maximal length sequence pseudo random number generator [7] was developed. This has a period of $2^{47} - 1$ clock cycles. The generator outputs in parallel two 8-bit random numbers from 16 consecutive bits in the generator sequence. During operation, a clock cycle causes a single shift of the shift register. To avoid correlations between consecutive random numbers, the entire 16 bit number is shifted out before sampling the next. Therefore each new random number will require 16 clock cycles. Thus a cell cycle is equivalent to 16 clock cycles. If the generator is clocked at 5MHz, random numbers will be generated out every 3.2 microsec-

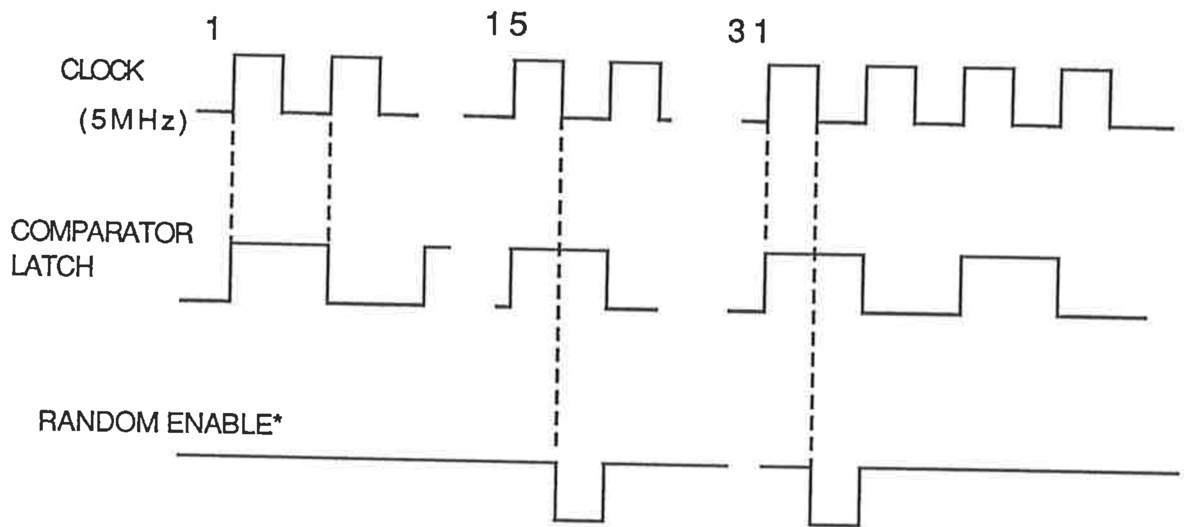


CLOCK



FRAME PULSE GENERATOR

Figure 3.6: The clock and the frame pulse generators (App 3.1 Sh. 1)



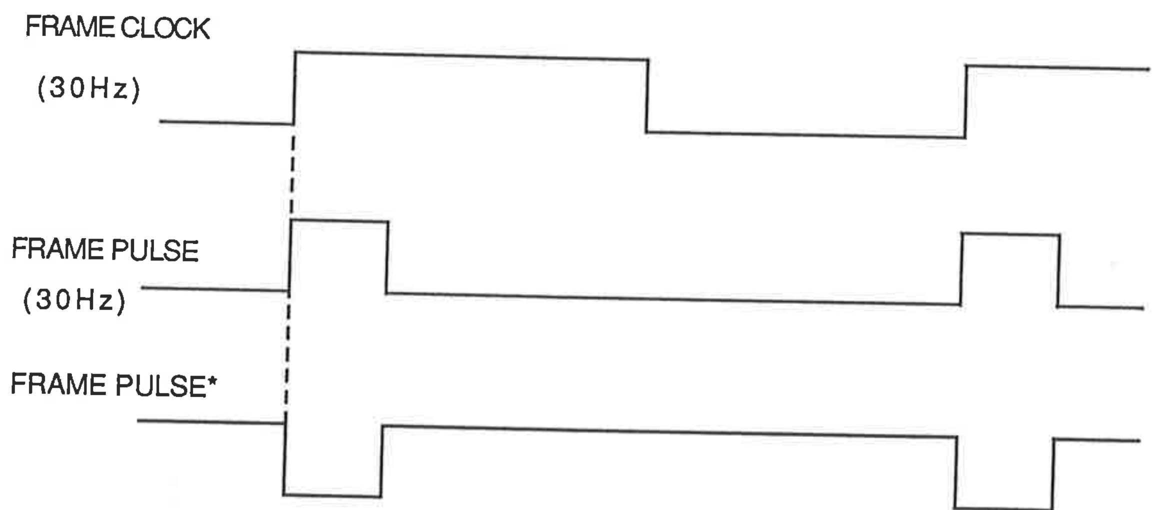


Figure 3.7: The clock and the frame pulse timing

onds. The generator will have a period of about 81 days, which is quite adequate for most simulations.

We use an initialization sequence to avoid correlations between different source boards. This will be described in detail later under a separate heading. This operation mode is called the initialization mode.

The random number generator block diagram is given in Figure 3.8. The generator consists of the shift register stages, two EOR gates, an inverter and a multiplexer. The multiplexer is used to select either the normal sequence or the initialization sequence as the generation sequence.

3.4.2 The mathematical model

This generator is built according to the sequence (from [7]),

$$X_n = (X_{n-14} + X_{n-47}) \bmod 2 \tag{3.1}$$

where $n \geq 47$. $X_0, ..., X_{47}$ are arbitrary bits not all zero.

The sequence is developed from the additive generator [7] defined by

$$X_n = (X_{n-l} + X_{n-k}) \bmod 2 \tag{3.2}$$

The subscript pairs (l,k) which give maximal length sequences (period $2^k - 1$) are listed for $k \leq 100$ in the reference [7]. For our generator, the pair (14,47) is used to generate random numbers. Therefore, this generator has a period of $2^{47} - 1$ clock cycles. The pair (13,31) is selected for the initialization.

3.4.3 Initialization

The multiplexer is used to select the random number generator operation modes. The operation mode is selected by the external computer via a control bit called the INITIALIZATION* bit. When this bit is low, initialization

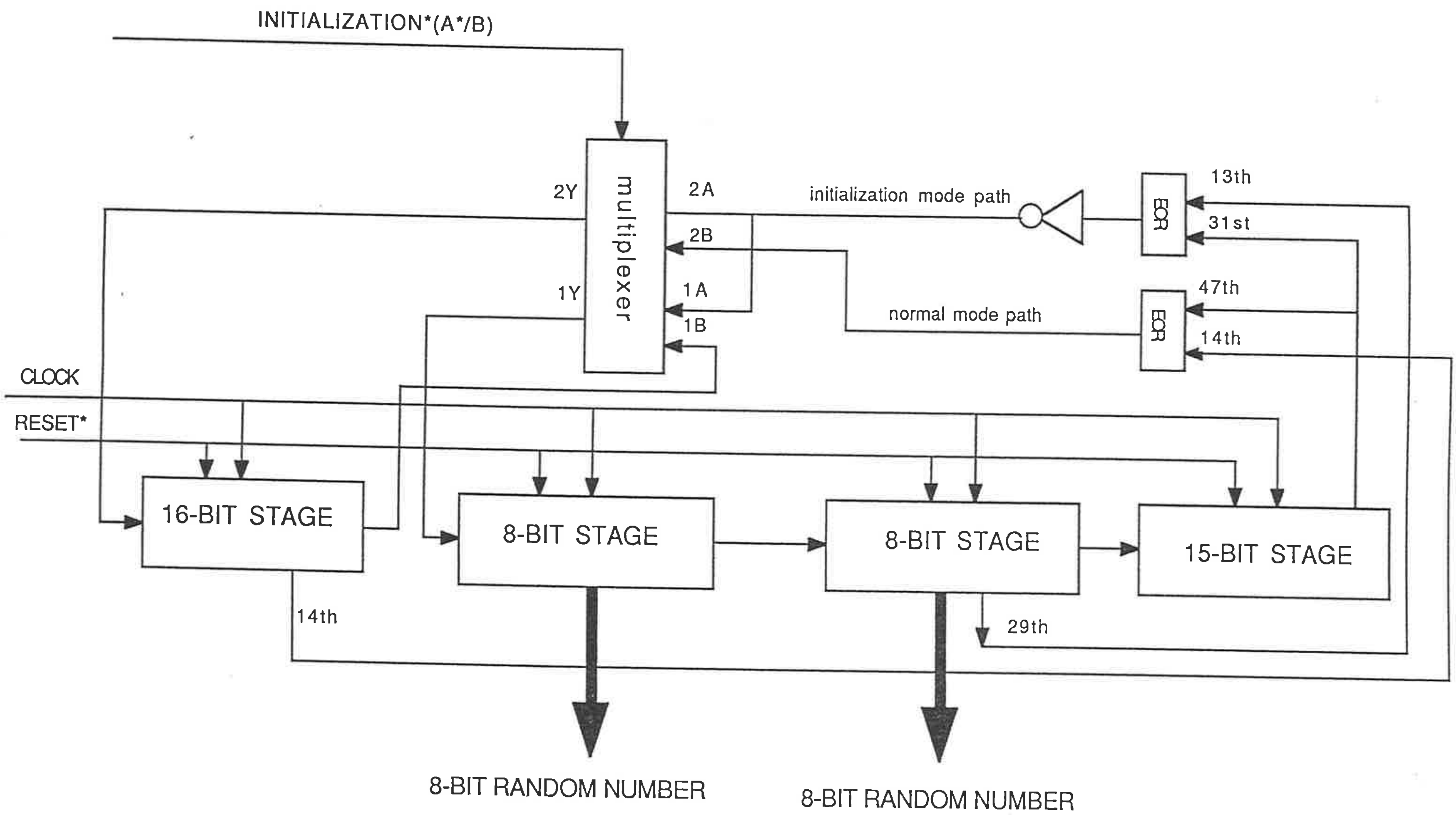


Figure 3.8: The random number generator (App 3.1 Sh. 1)

mode is selected. The multiplexer selects source A (1A,2A) as its output. This results in a shorter sequence with 31-bit length and the feedback path which consists of an inverter and an EOR gate are formed. This initialization generator will have a period length $2^{31} - 1$, which will have a period of about 7 minutes at 5 MHz clock. The initialization process is started and terminated by the external control computer, which selects a different (random) time to run the generator in each card. This ensures that there are no correlations between random numbers in different generator cards at the start of an experiment. An inverter in the initialization path ensures that zero state (all stages are zero) is not achieved at the end of the initialization process.

3.4.4 Normal operation

If the INITIALIZATION* bit is set high, the normal operation mode is selected. The multiplexer presents source B(1B,2B) to its output to form the 47-bit sequence. The 47th and 14th stages are provided to the EOR input, while the EOR output is passed to the 1st stage. This forms the normal mode feedback path. In normal operation mode, the generator is a 47 bit maximal length sequence pseudo random number generator.

In order to latch a stable output, the delay of transition must be taken into account. Figure 3.9 is the timing diagram concerning random number latch operation. The generator stages consist of six 8-bit shift registers with serial data entry and an output from each of the eight stages. For each LOW to HIGH transition on the clock, the stages transfer one bit to the right synchronously. This will have a maximum delay $t_s = 32ns$ [8]. Therefore after 32ns of the clock LOW to HIGH transition, the register stage outputs are stable. Supposing the clock frequency is 5MHz, t_{pl} will be 100ns. If the

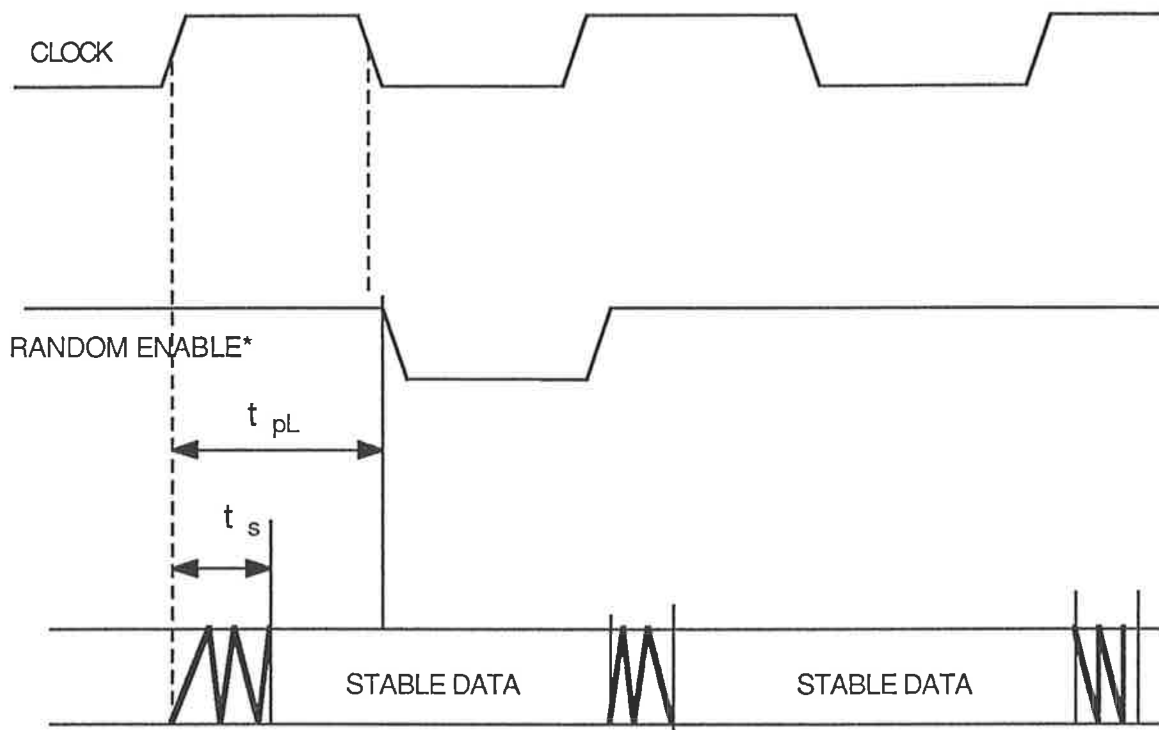


Figure 3.9: Random number latch timing

number is latched at the clock LOW going edge, the outputs will be stable. Because the RANDOM ENABLE* is used to latch the random number, thus the random numbers are latched in a stable state.

3.4.5 The hardware design

The full random number generator circuit diagram is given Appendix 3.1, sheet 1. The control signal INITIALIZATION* determines the operation mode of the generator which is controlled by the external computer. The signal has meaning "0" for the initialization operation mode "1" for the normal operation mode. During the normal operation the random number generator uses an EOR gate in the feedback path (see Fig 3.8) which combines the outputs of the 14th and 47th stages. The result is fedback into the serial input of

the shift register. When in the initialization mode, the feedback path and sequence are modified. An EOR gate and an INVERTER are used in the feedback path to combine the output of the 29th and 47th stages. The results is fedback into the serial input of the 17th stage. Because the 17th stage now acts as the first stage, the initialization generator is a 31 stage additive generator. The former 29th and 47th stages act as the 13th and 31th stage. The feedback result also is provided to the serial input of the first 16 stages to initialize these stages only.

Six 74LS164 shift registers (U7-U12) are used to form the 47 stage sequence. The first two registers are configured for 16-bit shift sequence. During the initialization, they only store bits from the 31 bit generator. The last two registers only 15 stages are used. The middle two registers are used as a 16 bit shift stages whose parallel outputs are latched as a 16 bit random number every 16 clock cycles. A 74LS86 (U13) and a 74LS04 (U14) are used as an EOR gate and an INVERTER respectively in the feedback path. During the normal operation, only the EOR is used. In the initialization mode, both the EOR gate and the INVERTER are used. A 74LS157 (U6) is used as a multiplexer to select the operation mode. The INITIALIZATION* signal is connected to its common select input to select either the initialization feedback path or the normal mode feedback path.

Because of the TTL IC propagation delay, the generator can only operate up to a limited clock frequency. The shift registers have a clock to output delay which is typically 21 ns and a maximum of 32 ns (taken from [8]). The propagation delay of the multiplexer and the EOR gate are, respectively, typically 9ns and 20 ns, and a maximum of 14 ns and 30 ns (from [8]). Therefore, the total delay is typically 44 ns and a maximum of 82 ns. This generator thus should have a minimum upper operation frequency of 12 MHz,

and a typical upper operation frequency of 22 MHz.

3.5 The traffic generation

As mentioned in chapter 2, the source generator is required to generate packet streams to simulate different types of traffic. In order to make simulation more flexible, the generator was developed to generate packets randomly either based on a probability parameter or on a count of packets in a frame. The generator block diagram (Figure 3.2) shows two generation machines called the packet generator and the parameter generator which determine the traffic generation. The packet generator will decide whether or not the source generator produces a packet in a time slot (cell), and the parameter generator will generate the packet header parameter for each packet.

3.5.1 Packet generation

It was decided that packets would be simulated as fixed length entities which are transmitted in time slots. This is basic to the asynchronous transfer mode.

Each generated packet is assumed to come from one of a number of specific sources. These packets may be related to each other in their final destination, and their specific requirements such as priority, packet loss tolerance or packet delay tolerance. The generator must provide these characteristics in each packet header.

For the simulation work which we envisage, it is considered that a 16-bit packet header will be able to contain all packet specific information which the simulation needs. Therefore, the source generator will only generate 16-bit

packet headers rather than whole packets. The lower 8-bit of the header is control information relating to particulars of experiments. This is split into a 5-bit field (called destination address) and 3-bit field (called class number). The upper 8 bits is an identifier to enable packets to be distinguished as separate entities for the measurement of delay. This identifier consists of a 4-bit source identifier and 4-bit packet sequence number. The packet header structure is shown in Figure 3.10. Wherever a packet is transmitted serially, it shall be transmitted least significant bit first, so the destination address always is transmitted first.

3.5.2 The packet generator

As mentioned earlier, the packet generator will decide whether a packet is to be generated or not, and does so at specified rates or probabilities. In the simulated traffic, each frame has a variable number of packets. As shown in Figure 3.11, the packet generation can be done in two ways by using the comparator and counter. When using the comparator, the frame model is devised to specify a packet probability for generation of randomly distributed packets. In this generation mode, the packets are spread over the whole frame interval. When using the counter, the frame model specifies a packet count for generation of packets in a single burst at start of each frame. The typical distribution of packets in a frame of these two types of generation is shown in Figure 3.12.

The packet generator is required to generate the same number of packets in a frame in the counter mode as in the comparator mode when using the same rate parameter value. In the counter operation mode, when a frame starting, the generator produces continuously the packets. One packet is generated every cell cycle, and the counter is decreased until the number held

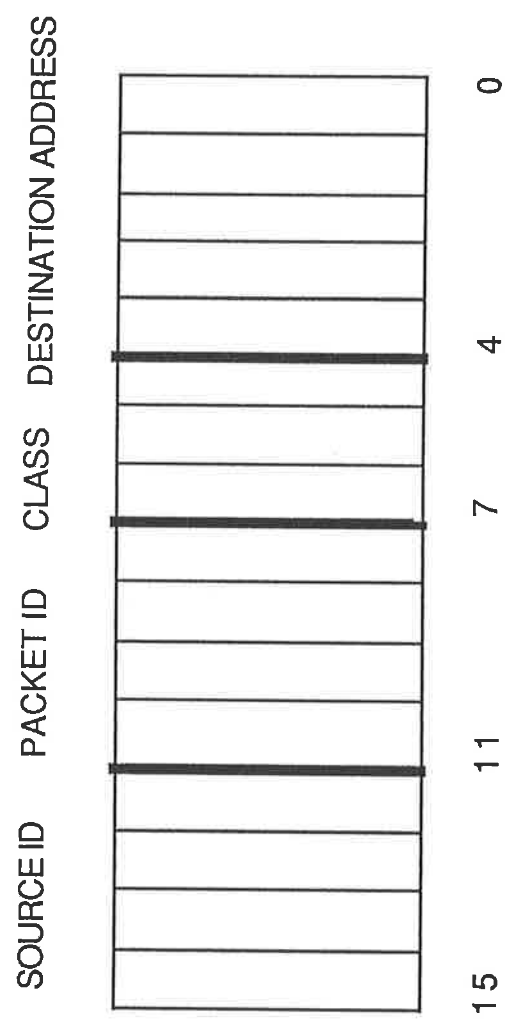


Figure 3.10: The packet header format

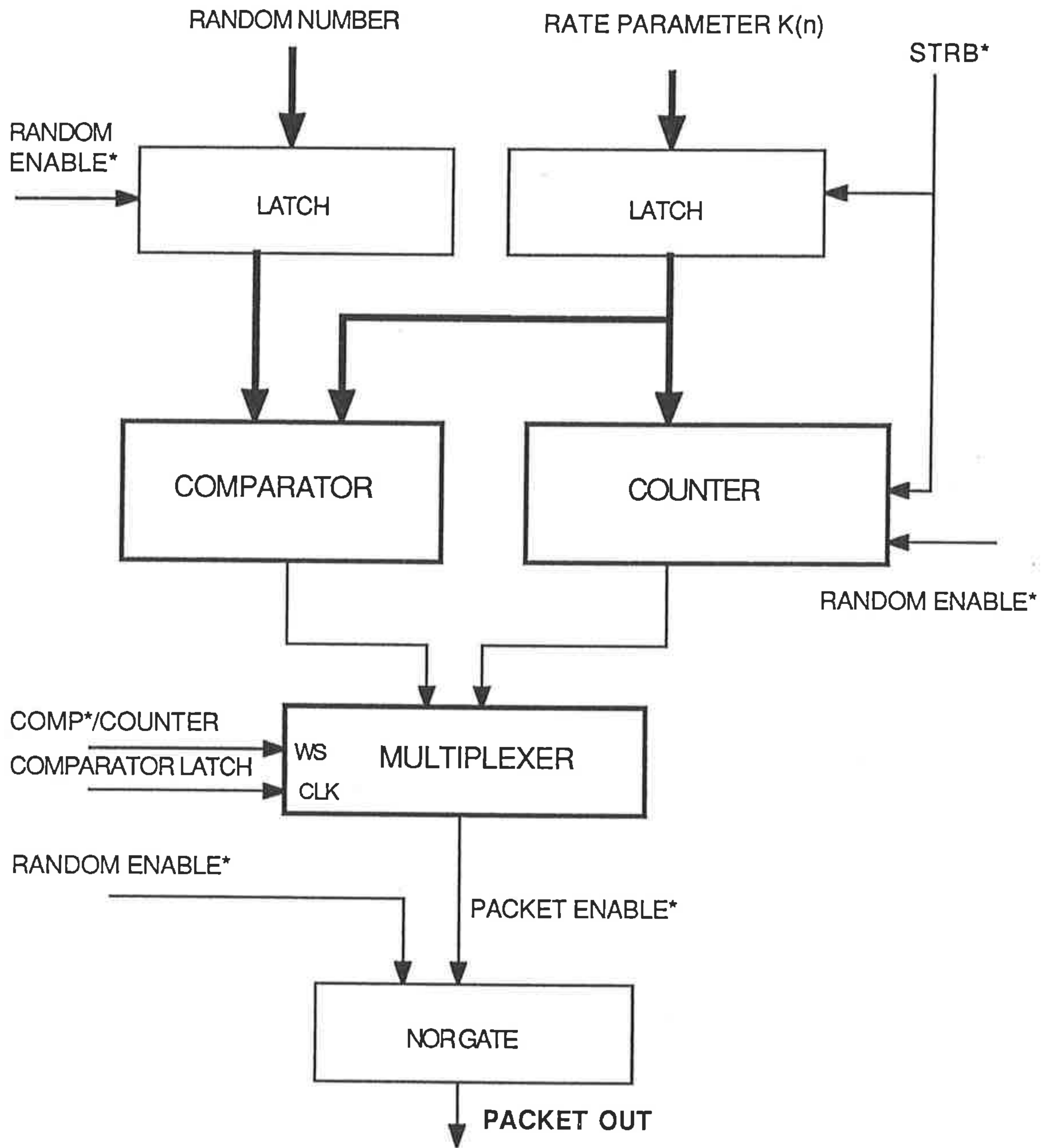
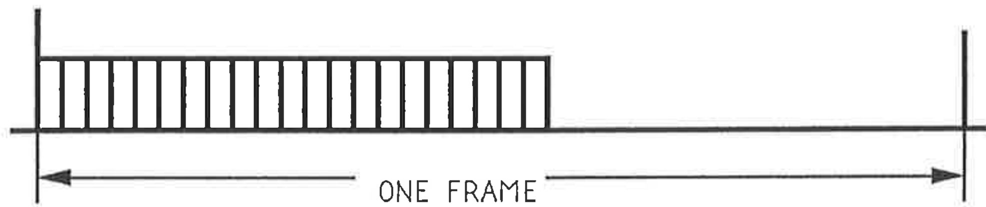
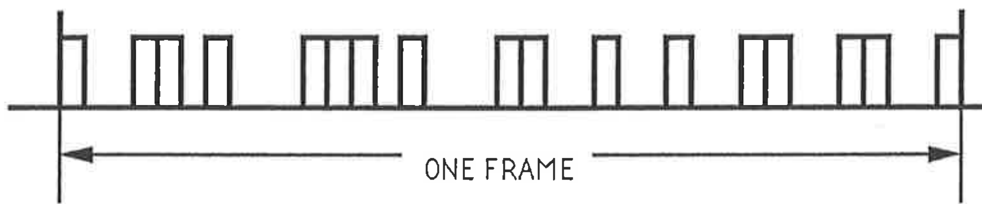


Figure 3.11: The packet generator (App 3.1 Sh. 2)



Packet distribution using the counter mode



Packet distribution using the comparator mode

Figure 3.12: The packet distribution in the comparator and counter modes

in the counter becomes zero. Then the generation is stopped. The number of packets in a frame is determined by the 8 bit rate parameter $K(n)$ loaded into the counter. In the comparator mode, the comparator compares the random number latched from the random number generator with the rate parameter $K(n)$. This determines whether to generate a packet in a cell. The user can select the operation mode by the external control computer.

The comparator mode

In this mode, the number of packets generated in a frame depends on the rate parameter $K(n)$. The $K(n)$ is generated by the microcontroller using the software model. Both the rate parameter $K(n)$ and the random number consist of unsigned 8-bit values. Refer to Figure 3.11, when the multiplexer selects the output of the comparator as its output, the packet generator is operating in the comparator mode. The comparator compares the $K(n)$ value with the current random number. If the $K(n)$ is greater than the random number, then a packet is generated in this cell. Otherwise there is no packet in this cell. When $K(n)$ is 255, every cell contains a packet except the cell in which the random number is 255, while if $K(n)$ is zero, no cell contains a packet.

The two 74LS374 Octal D-type Flip-Flops (U15, U16) are employed to latch the random number and the rate parameter $K(n)$ from the random number generator and the microcontroller. At the RANDOM ENABLE* falling edge, the 8-bit random number is latched and then output to the comparator. The $K(n)$ is latched into the comparator at the STRB* falling edge. Thus in each frame only one rate parameter value is available for the comparison. The comparator performs a magnitude comparison between the random number and the $K(n)$. If the $K(n)$ is greater than the random number, the comparator 74LS687 (U17) will output a low-active signal which is passed to an input pin of the multiplexer. On the rising edge of the COMPARATOR LATCH, this active signal is latched to the output of the multiplexer as the PACKET ENABLE* which will cause one packet to be generated. For a frame clock of 30 Hz, the average number of packets in the nth frame is

$$P(n) = (K(n)/256)(F/16)(1/30) \tag{3.3}$$

where F is the clock frequency.

The counter mode

When the multiplexer selects the output of the counter, the packet generator is operating in the counter mode. The counter diagram is shown in Figure 3.13. In order to generate the same number of packets as in the comparator mode, a divide-by-forty counter is required to modify the number of packets in a frame. Two ICs 74LS191 (U21) and 74LS90 (U22) which are used as divide-by-eight and divide-by-five counters, respectively, form this divide-by-forty divider (the reason for selecting a divide-by-forty divider will be given in Chapter 4). Two 74LS191 ICs (U18, U19) are serially connected to form an 8-bit down counter (refer to the circuit diagram Appendix 3.1). This counter loads the $K(n)$ every frame cycle. The cell cycle signal (RANDOM ENABLE*) is applied to the divider input. The divider output is applied to the clock pulse input of the counter.

While the number in the 8-bit counter is not zero, the PACKET ENABLE* signal is active. The source generator will therefore generate one packet every cell cycle. When the counter receives a pulse (active HIGH going edge) from the divider, the number in the 8-bit counter decreases by one until the number becoming zero, it turns the PACKET ENABLE* inactive. Then the generator stop generating packets, and at same time it makes the counter disable counting and the number in the counter remains at zero. Until the next STRB* active, it loads the number $K(n+1)$ and starts generating packets for next frame. The number of packets in nth frame is

$$P(n) = 40 * K(n) \tag{3.4}$$

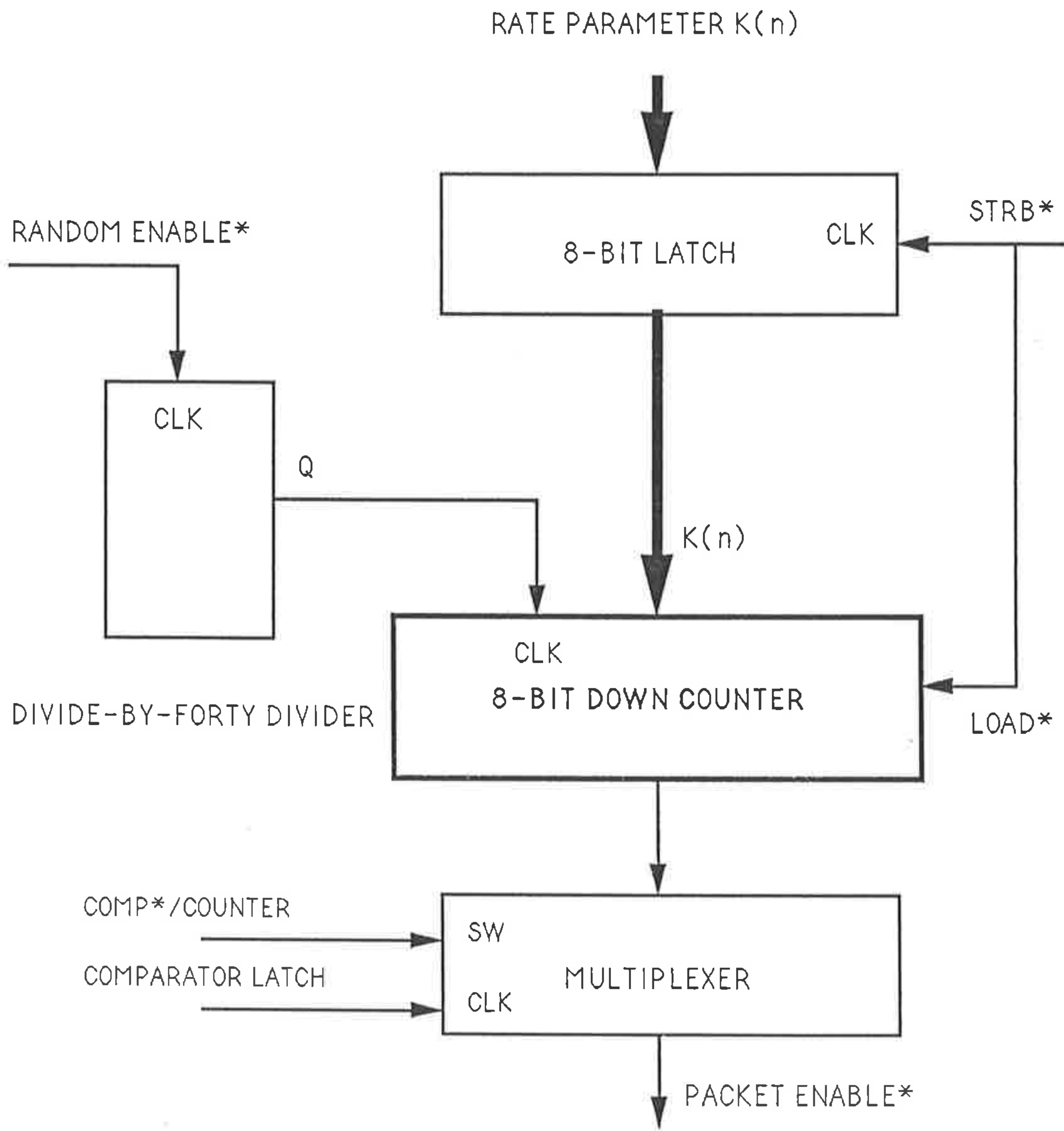


Figure 3.13: The counter mode diagram (App 3.1 Sh. 2)

Timing

In the comparator mode, the multiplexer selects the comparator output. The COMPARATOR LATCH signal, which is from the clock section, is provided to the clock pin of the multiplexer. The comparator compares the latched 8-bit random number with the rate parameter $K(n)$. If the $K(n)$ is greater than the random number, the comparator produces a LOW output which is provided to an input of the multiplexer. This LOW signal is latched to the multiplexer output (PACKET ENABLE*) on rising edge of the COMPARATOR LATCH. The NOR gate combines the PACKET ENABLE* and RANDOM ENABLE* to produce the active PACKET OUT signal. The upper in Figure 3.14 is the comparator mode timing diagram.

In the counter mode, the multiplexer selects the counter output. When a frame starts, the counter loads the rate parameter $K(n)$. The counter produces LOW output when the number in the counter register is not zero. This signal is passed to the PACKET ENABLE*. One active PACKET OUT therefore is produced every active RANDOM ENABLE* pulse. When the number in the counter register falls to zero, the counter drives the PACKET ENABLE* high inactive, and thus the packet generation is stopped. The counter mode timing is given in Figure 3.14.

3.5.3 The parameter generator.

The source parameter generator determines the packet header to be used for each generated packet. It may have different specifications for each different packet. This would allow the nature of the simulated traffic to be defined in a flexible way.

As shown in Figure 3.15, the parameter generator will generate a 16 bit

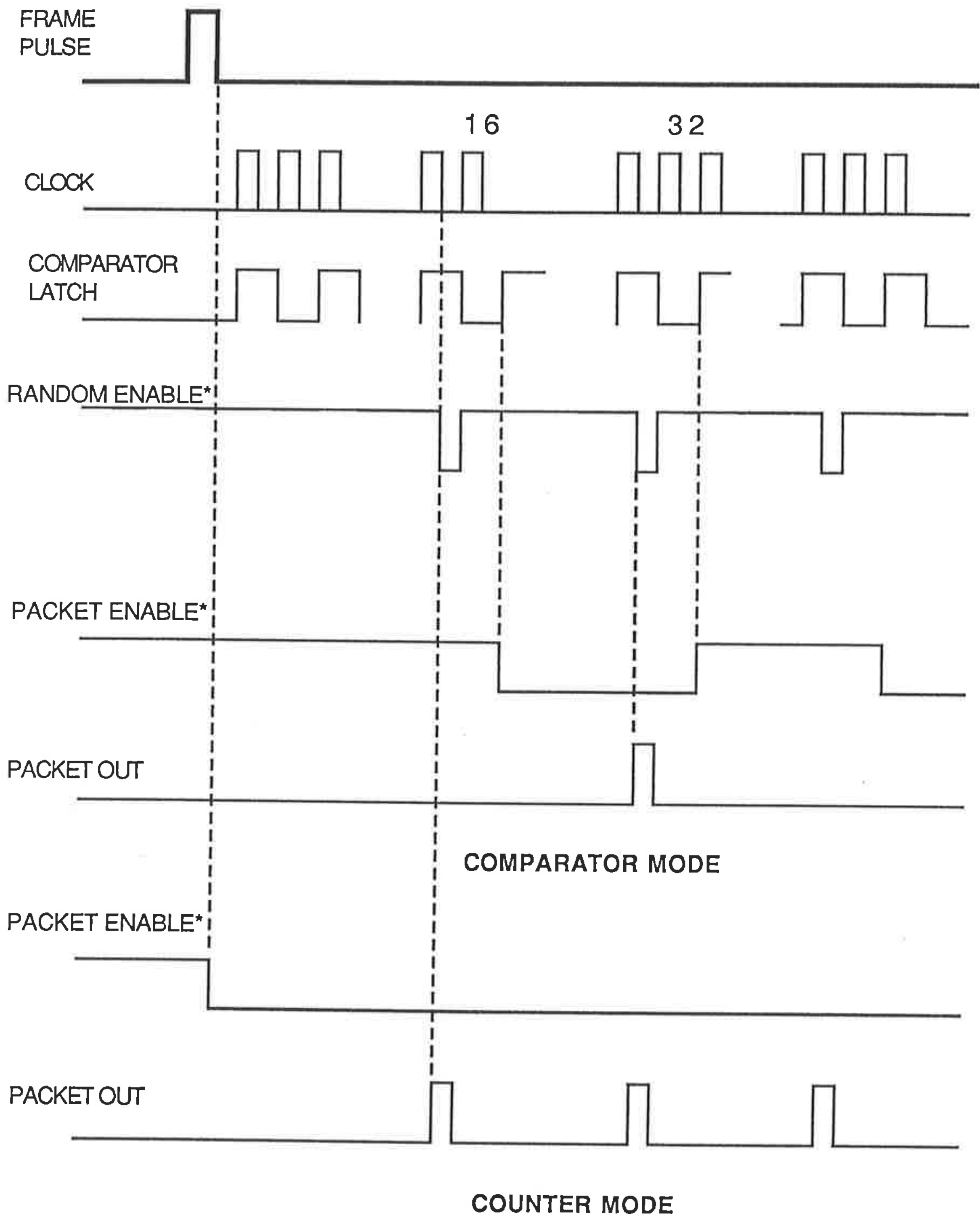


Figure 3.14: Timing of the packet generator

packet header which includes 8 bits of control information and 8 bits of identification information. The control information consists of a 5 bit destination address and a 3 bit class parameter. The identification information includes a 4 bit packet ID and a 4 bit source ID.

The generator has been designed to provide two types of control information. This can be either fixed with a given class and destination, or randomly allocated in the destination or class field or both. The fixed class and destination parameters will come via the VMEbus from the control computer, while the random class and destination parameters will use an 8-bit random number from the random number generator.

The 5-bit fixed address and 5-bit random address are passed to a multiplexer. The 3-bit fixed class and 3-bit random class are similarly passed to a different multiplexer. The external control computer via the VME bus determines the multiplexer selection of random or fixed address field, and random or fixed class field. The FIXED ADDRESS ENABLE* signal is provided to the multiplexer 74LS157 (U28, U29) common select input. When it is LOW, the output is the fixed address. If it is HIGH, the output is the random address. The FIXED CLASS ENABLE* is provided to the second multiplexer (U30) to select either fixed or random class. When this signal is LOW the multiplexer will output the fixed class. If it is HIGH, the multiplexer will output random class. The random class and destination are updated every cell cycle. The fixed address and class can only be changed by the control computer.

The switch DIP-4 is used as the source ID generator. Every source generator element will have its own source ID. Because DIP-4 is used as the source ID, there are 16 possible source IDs which can be selected. This would allow up to sixteen elementary source generators to operate at same time in any experiment which tracks individual cells.

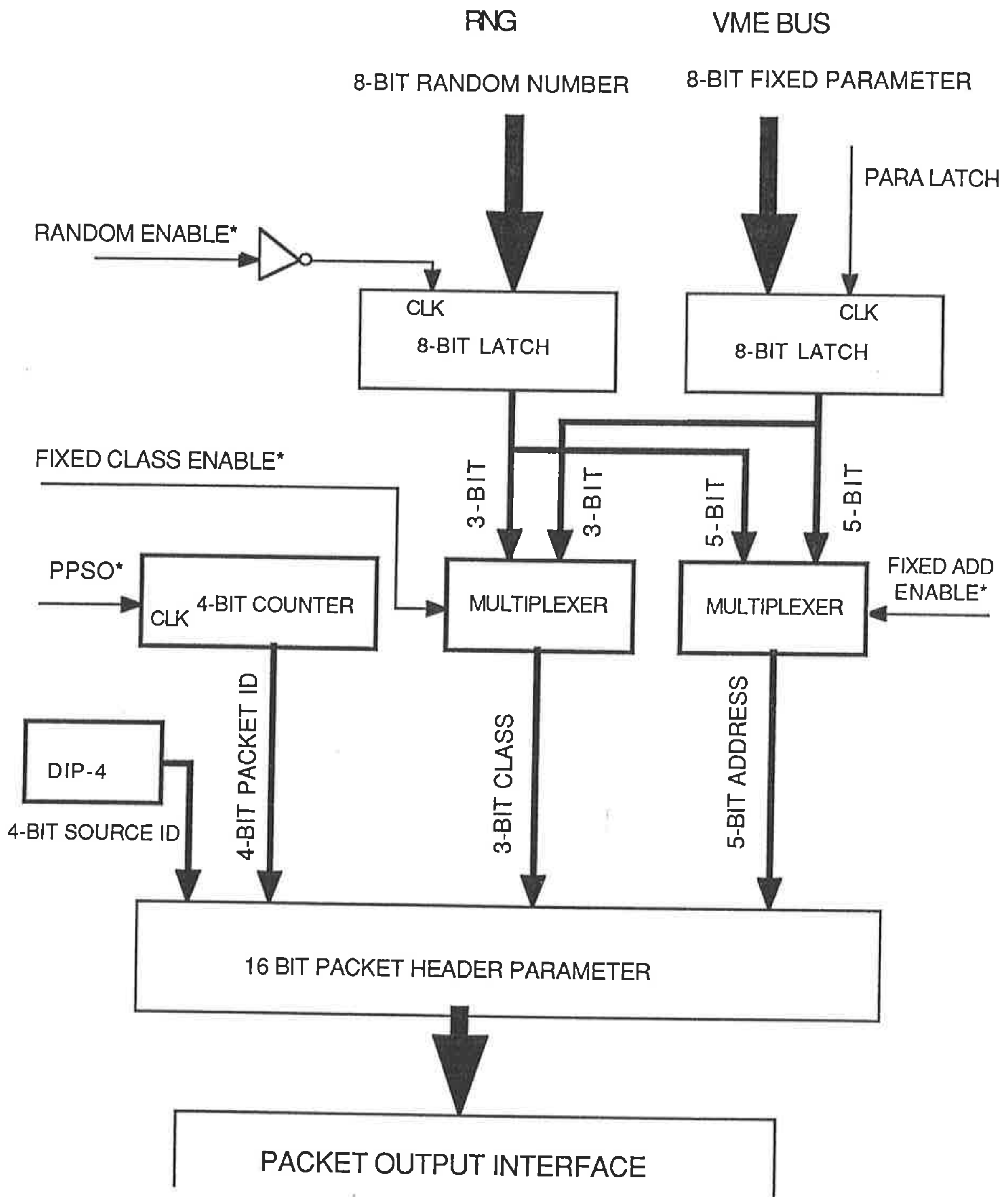


Figure 3.15: The parameter generator (App 3.1 Sh. 3)

The 4 bit binary counter 74LS191 (U27) is employed to generate the packet ID. The PPSO* signal is provided to its clock pulse input (PPSO* is the packet present strobe output signal from the packet output interface. Detailed discussion will be given a later chapter titled 'Interfaces'.). One active PPSO* is output every time the generator transmits a packet. Subsequent packets therefore have different packet ID modulo 16.

3.6 Interfaces

The elementary source generator contains two major interfaces. One is the control computer interface, the other is the packet interface. The control computer interfaces with the source generator to perform its control. The generator board interfaces with the multiplexer to output the packets.

3.6.1 The control computer interface.

Each of source generators will operate under the system control. The generator elements are addressed by the computer, and the address location of each card is placed in the top 64K of the 68000 memory space referred as a short address space with range $\$FF0000$–$\$FFFFFF$ of the VMEbus address. Each generator board is allocated four addresses so that the control computer can communicate directly with the four registers on the board. These four addresses are decoded using the lower 3 bits (A03,A02,A01) of address by the decoder. The address bits A07,A06,A05,A04 are determined by means of a 4DIL switch which also is used as the source identifier. This allows up to 16 generators to be addressed separately by the control computer.

DS0* and DS1* select the data to be transferred and, on a write transfer, strobe the transferred data. DS0* low means that the byte which would be addressed with A00 set high (the odd byte) is to be found on data lines D00 through D07. Likewise, DS1* low means that the byte which would be addressed with A00 set low (the even byte) is to be found on data lines D08 through D15 [9].

The computer interface diagram is given in Figure 3.16. the VMEbus option D8 [9] is selected, which specifies a SLAVE will read or write only 8 bits at a time on D00-D07. Option D8 also specifies that a MASTER will be capable of reading or writing via all 16 data lines D00-D15, but only eight in any one transfer. We select to use the date lines D00-D07, which will use DS0* only. Therefore the addresses for these locations are odd bytes in the short address space $\$FFFFXY$ of the VME bus where Y is the on-board register offset (odd byte only), and X is selectable by the 4DIL switch.

The control computer will transfer data to the four ~~blocks~~ registers of the generator board. They are:

1. The control register. The control computer transfers 5 control bits to this register. These bits are:

1. SOURCE ENABLE* bit for switching the source generator on or off.

2. INITIALIZATION* bit to select either initialization or normal operation mode of the random number generator.

3. FIXED CLASS ENABLE* bit to select a fixed class or random class as the packet class parameter.

4. FIXED ADD ENABLE* bit to select the packet address parameter from a fixed or random basis.

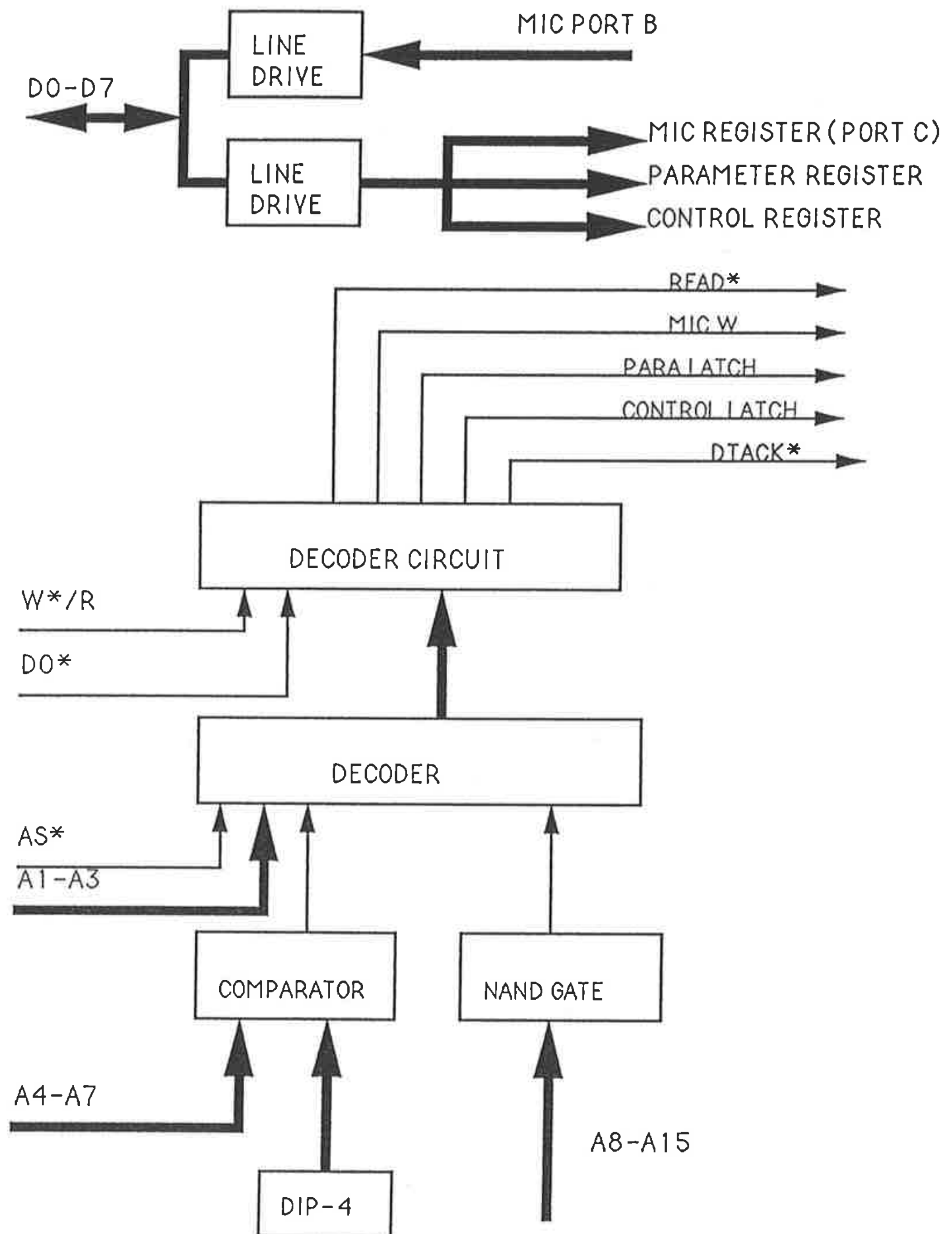


Figure 3.16: The computer interface (App 3.1 Sh. 6)

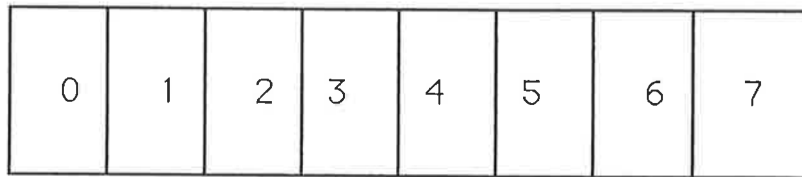


Figure 3.17: The control register

5. COMP*/COUNTER bit to select the comparator or counter mode of the packet generator.

The control bits in the control register are shown in Figure 3.17.

2. The parameter register. The control computer transfers 8 parameter bits to this register which consists of a 5 bit packet destination address parameter and a 3 bit class parameter.

3. The microcontroller port C. The control computer writes 8 bit data to the port C for communication with the program.

4. The microcontroller port B. The control computer can read asynchronously the 8 bit rate parameter $K(n)$ from port B.

Figure 3.18 gives the computer interface timing diagram. AS*, D0*, and W*/R are address strobe, data strobe and write/read control signal respectively from the VMEbus.

The byte write cycle is as follow:

Receive address, receive AS* driven to low. Data is presented on data lines D00–D07. Then receive WRITE* low, receive DS0* driven to low. If the address is valid for this board, latch data from lines D00–D07 and store it in the selected register. Then drive DTACK* to low to acknowledge data transfer. Then receive AS* and DS0* driven to high, release DTACK*. A write cycle is completed.

The byte read cycle is as follow:

Receive address, receive AS* driven to low. If the address is valid for this board then generate device selected. Then receive WRITE* high, receive DS0* driven to low, latch data and present it on lines D00-D07, drive DTACK* to low. After the master read data, the board receives AS* and DS0* driven to high, release D00-D07, release DTACK*, finish read cycle.

The connector for the interface on the back edge for each card is a 96-pin DIN connector. The Figure 3.19 identifies the VMEbus connector pin assignments (from [9]).

3.6.2 The packet interface

The interface with the statistical multiplexer is a single output interface. The bus lines include 16 data lines to output the packet header and 4 control lines. There are two operation modes for the packet interface which are standalone operation mode and bus operation mode, which can be selected externally. The standalone operation mode is appreciated if only one source generator is used. If there are a number of source generators are used simultaneously, the bus operation mode will be used in conjunction with a multiplexer. The control bits have different functions in each mode. The packet output interface diagram is given in Figure 3.20. The following paragraph will give a

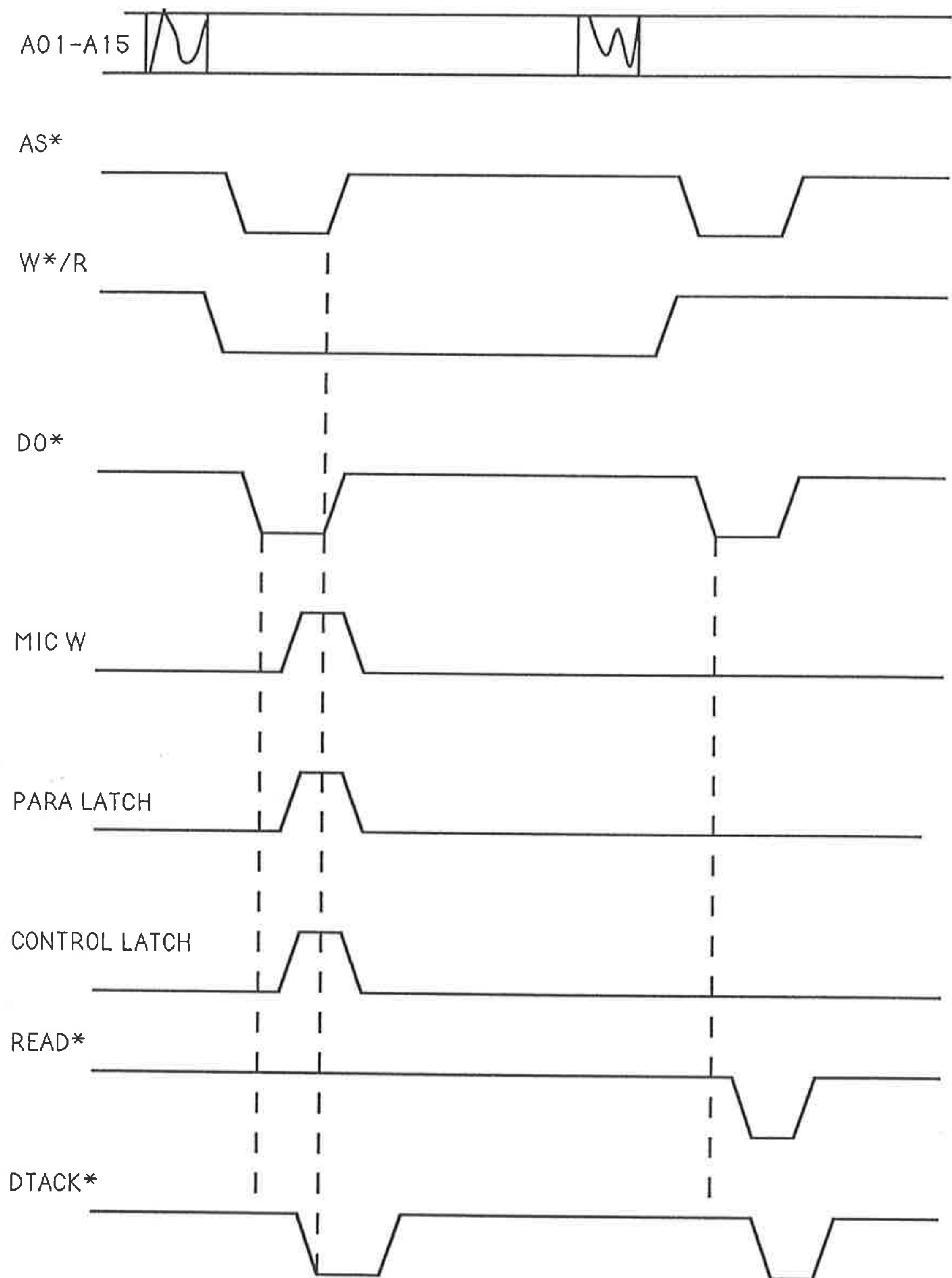


Figure 3.18: The computer interface write/read timing

PIN NUMBER	ROW A	ROW B	ROW C
1	D00		D08
2	D01		D09
3	D02		D10
4	D03		D11
5	D04		D12
6	D05		D13
7	D06		D14
8	D07		D15
9	GND		GND
10	SYSCLK		SYSFAIL*
11	GND		BERR*
12	DS1*		SYSRESET
13	DS0*		LWORD*
14	WRITE*		AM5*
15	GND		A23
16	DTACK*		A22
17	GND		A21
18	AS*		A20
19	GND		A19
20	IACK*		A18
21	IACKIN*		A17
22	IACKOUT*		A16
23	AM4	GND	A15
24	A07	IRQ7*	A14
25	A06	IRQ6*	A13
26	A05	IRQ5*	A12
27	A04	IRQ4*	A11
28	A03	IRQ3*	A10
29	A02	IRQ2*	A09
30	A01	IRQ1*	A08
31	-12V	+5V STDBY	+12V
32	+5V	+5V	+5V

Figure 3.19: The VMEbus connector pin assignments.

detailed description.

Standalone operation

There are three control lines in the standalone operation mode. They are: the MODE input line, the PACKET PRESENT STROBE OUTPUT* (PPSO*) line and the ACKNOWLEDGE INPUT* (AI*) line. The MODE line is used to select operation mode of the interface. When this line is HIGH, the interface is in the standalone operation mode.

The SOURCE ENABLE* signal which is from the control register is to switch the source generator on or off. When this signal is 1, it makes PPSO* inactive. Thus the packet output buffers are high impedance and no packets can be output. The source generator is in an off state. If the signal is 0, the generator is in an on state, and packets can be transferred.

When the packet generator is ready to generate a packet in a given time slot, an active PACKET OUT signal is generated, and passed to the clock input pin of the D Flip-Flop. On its rising edge, the PACKET OUT1* (Q* pin of D Flip-Flop) is set low. The PACKET OUT1* is gated by the OUTPUT ENABLE*. In standalone mode, MODE=1, the OUTPUT ENABLE* signal is active, enabling the PACKET OUT* to be transferred to the PACKET STROBE* line. An active PACKET STROBE* will enable the packet output buffer which presents a packet to Q00-Q15. After a short delay, the PACKET STROBE* is presented to the PPSO* line to inform the receiving system that a packet is ready to be taken. After receiving the packet, the external system drives ACKNOWLEDGE INPUT* low. This will set PACKET OUT1* high. This signal then drives PPSO* high impedance. A packet transfer cycle is completed, and one packet is output. The standalone operation timing diagram is given in Figure 3.21.

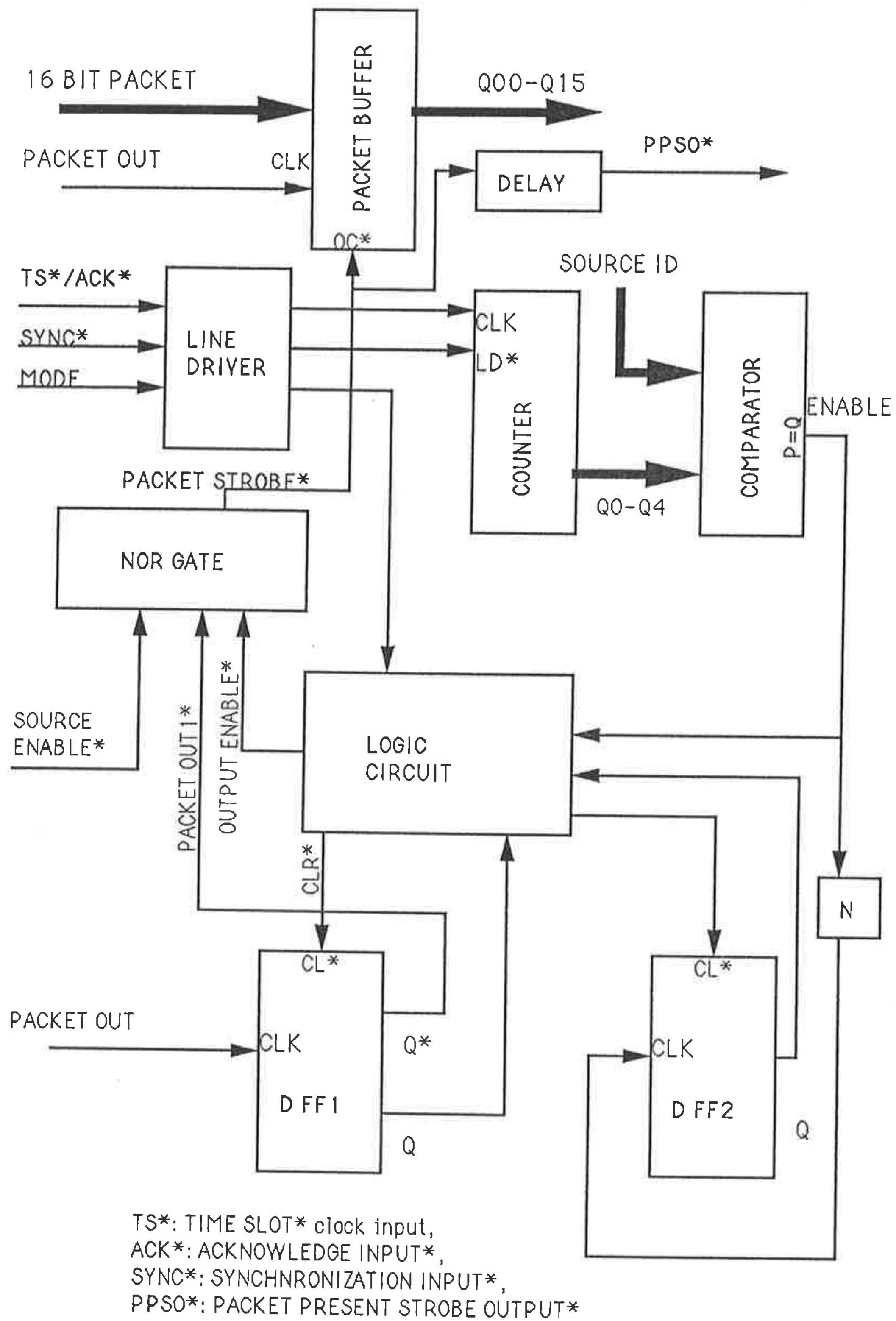


Figure 3.20: The packet output interface (App 3.1 Sh. 5)

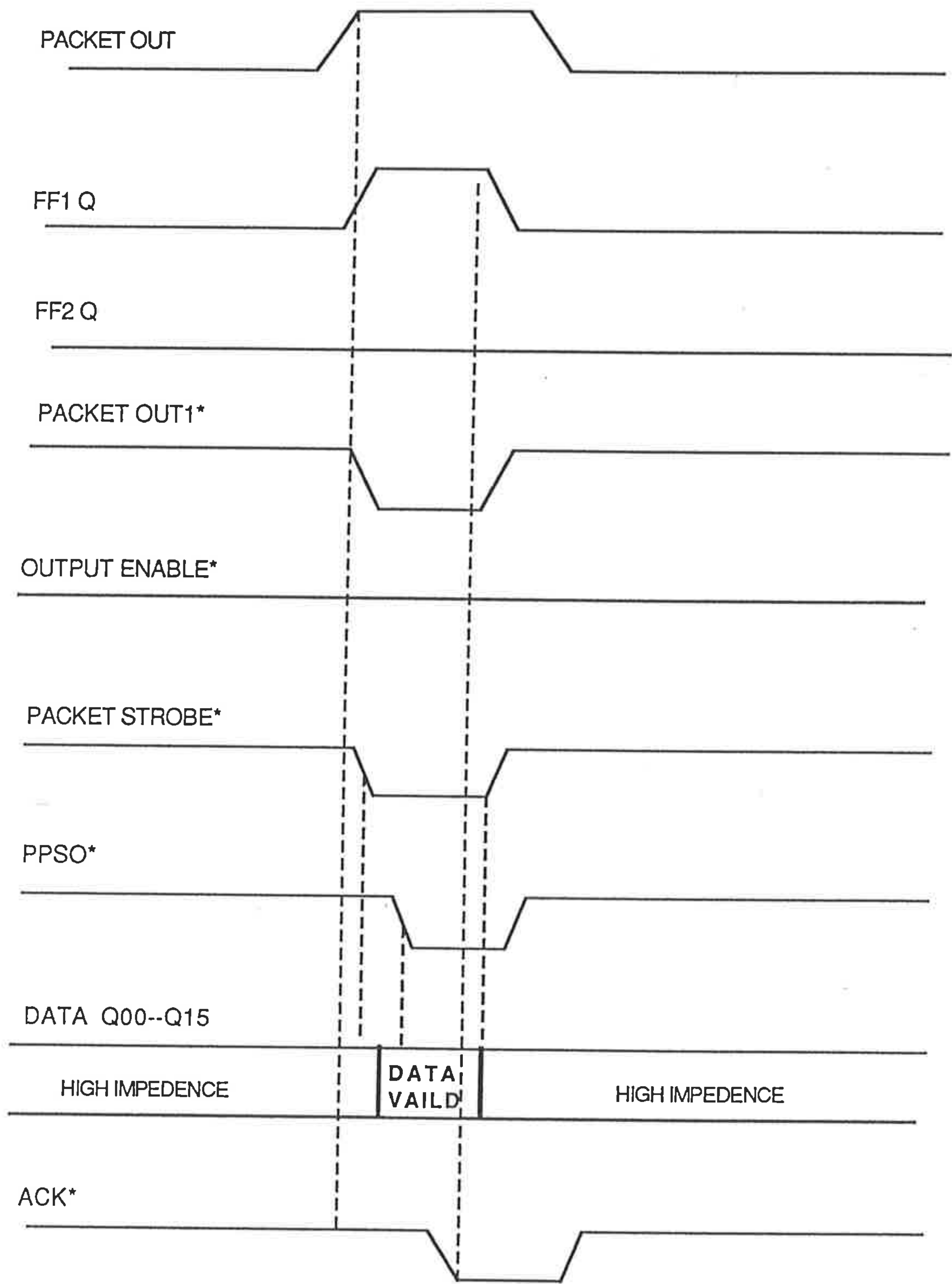


Figure 3.21: Standalone operation timing

Bus operation mode

If the MODE is "0", the interface is in the bus operation mode. In this mode, the testbed allows up to 16 elementary source generators to be used simultaneously. The source group is able to generate packet streams to simulate traffic in a flexible way. The other three control lines are SYNCHRONIZATION INPUT* (SI*) line, TIME SLOT CLOCK INPUT* (TS*) line and PACKET PRESENT STROBE OUTPUT* (PPSO*) line.

Referring to Figure 3.20, the SI* signal is to synchronize all source generators. The TS* is used to select each board time slot. Both SI* and TS* are from the external system. The PPSO* is used to inform the external system that a packet is ready to be received.

The TS* is provided to the clock input of a 4-bit counter. The output of the counter is provided to input pins A0-A3 of a 4 bit comparator. The 4-bit source identification is provide to the other input pins B0-B3 of this comparator. If two numbers are equal, the comparator will drive ENABLE high. The ENABLE then drives the OUTPUT ENABLE* low active. Thus the PACKET OUT1* is presented to the PACKET STROBE* line , and then to the PPSO* line.

At a minimum, one time slot should contain 16 clock cycles because up to 16 source generators may be used simultaneously. The source ID will be numbered from 0 to 15. We assume the first source board has ID 0, the second source board has ID 1, and so on, with the sixteenth source board having ID 15. The first TS* pulse will activate first source board, and the nth clock will activate the nth source board.

A generator can at most generate only one packet in a time slot. If a packet is generated, the packet generator will output an active pulse called

PACKET OUT. This is passed to the clock input of the D Flip-Flop, on its positive-going edge, setting Q* of the D Flip-Flop low (PACKET OUT1*) which is gated by the OUTPUT ENABLE* signal. When the counter number is equal to the source ID, the comparator drives ENABLE high. This then drives the OUTPUT ENABLE* low active. This causes the PACKET OUT* to be presented to the PACKET STROBE* line which enables the packet buffer to pass the packet header on Q00-Q15. Then the PACKET STROBE* is passed to the PPSO* line to inform the external system.

At same time, the ENABLE high also drives another D Flip-Flop. This makes its output Q drives the CLR* low. The CLR* low clears the first D Flip-Flop to drive the PACKET OUT1* high. Thus this makes the PACKET STROBE* high, and then PPSO* high inactive to terminate the packet output. A packet output cycle is completed.

If it is not a board time slot, the ENABLE is low. This will drive the OUTPUT ENABLE* high, making the PACKET STROBE* high inactive and the PPSO* high impedance.

The bus operation timing diagram is given Figure 3.22.

The connector

All interconnections between the source generator elements are required to be compatible. The card edge connector is 40-pin IDC connector in which the 20 signal lines are interspersed with ground lines. The assignment of pins is given in Figure 3.23 (from [1]). The 4 control bits are strobe and status bits which are all active LOW. The MODE input bit is used to signal variations in the operation mode of the interface. For the bus operation mode, allocations of the control lines are:

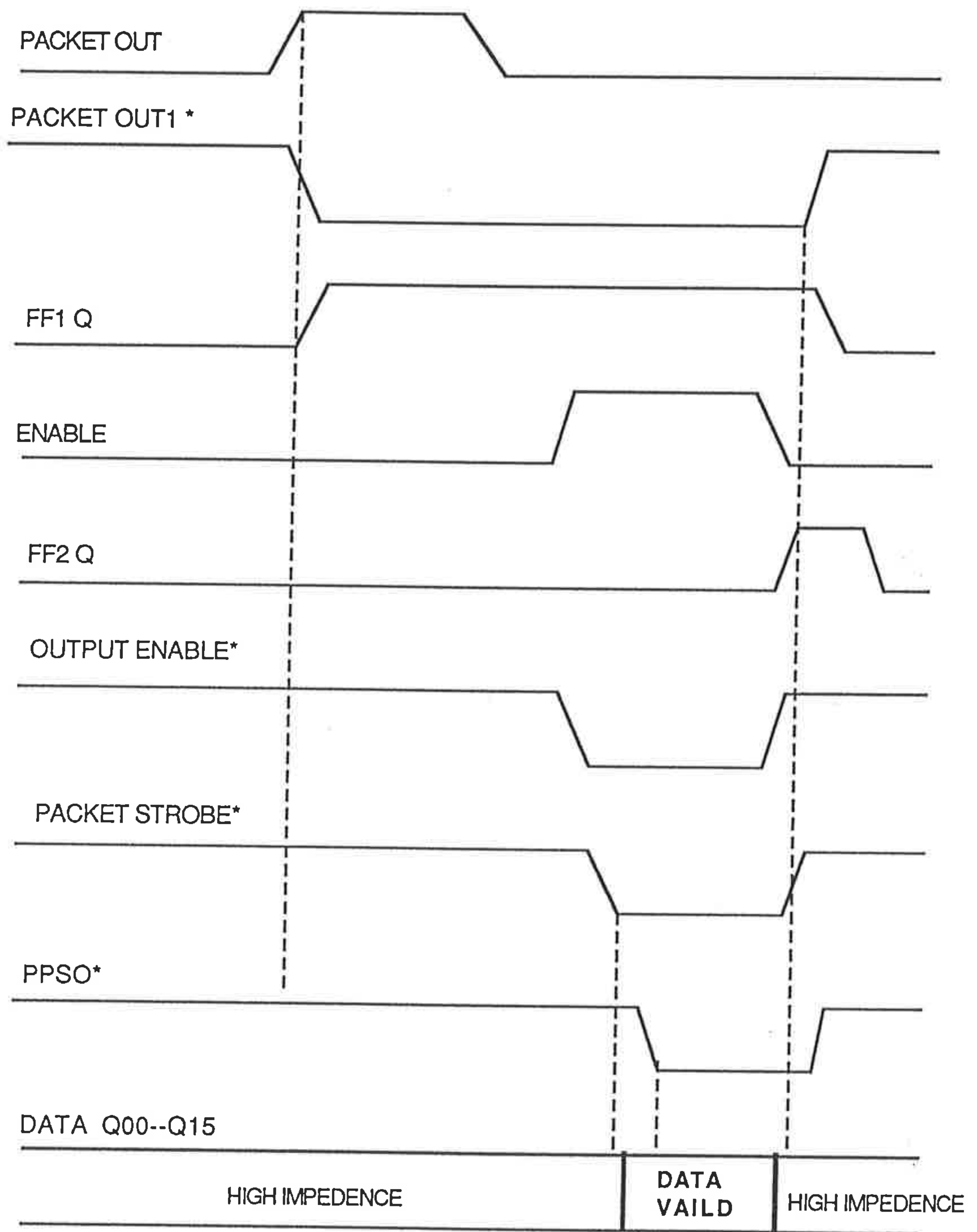


Figure 3.22: Bus operation timing

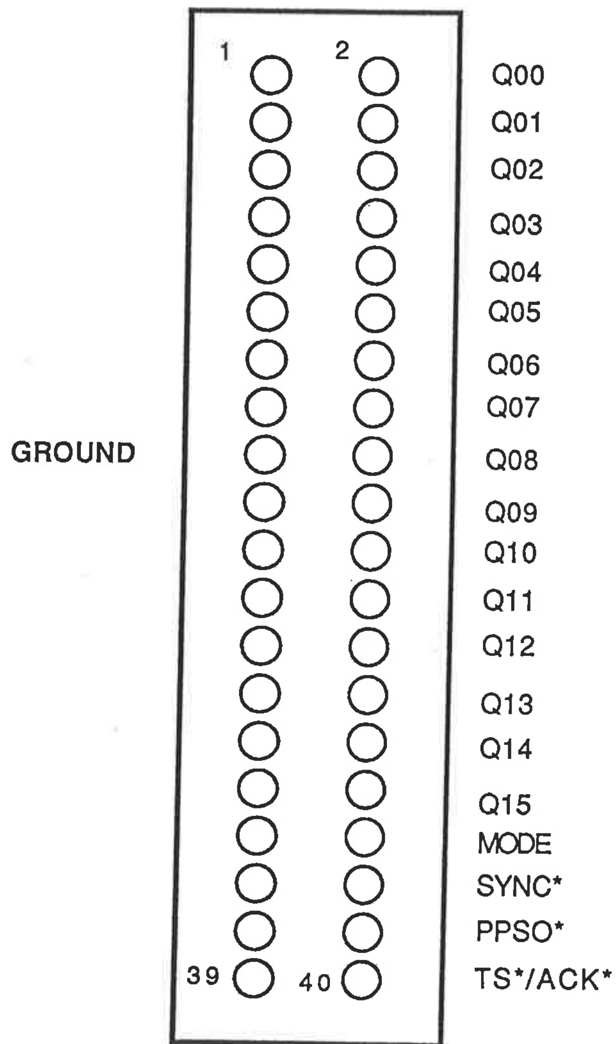


Figure 3.23: Pin allocation for packet interface connector

Pin 34: MODE input 0, bus operation.

Pin 36: SYNCHRONIZATION INPUT*.

Pin 38: PACKET PRESENT STROBE OUTPUT*.

Pin 40: TIME SLOT* clock input.

For standalone operation, the assignments of the control lines are compatible with those the output of the multiplexer:

Pin 34: MODE input 1, standalone operation.

Pin 36:

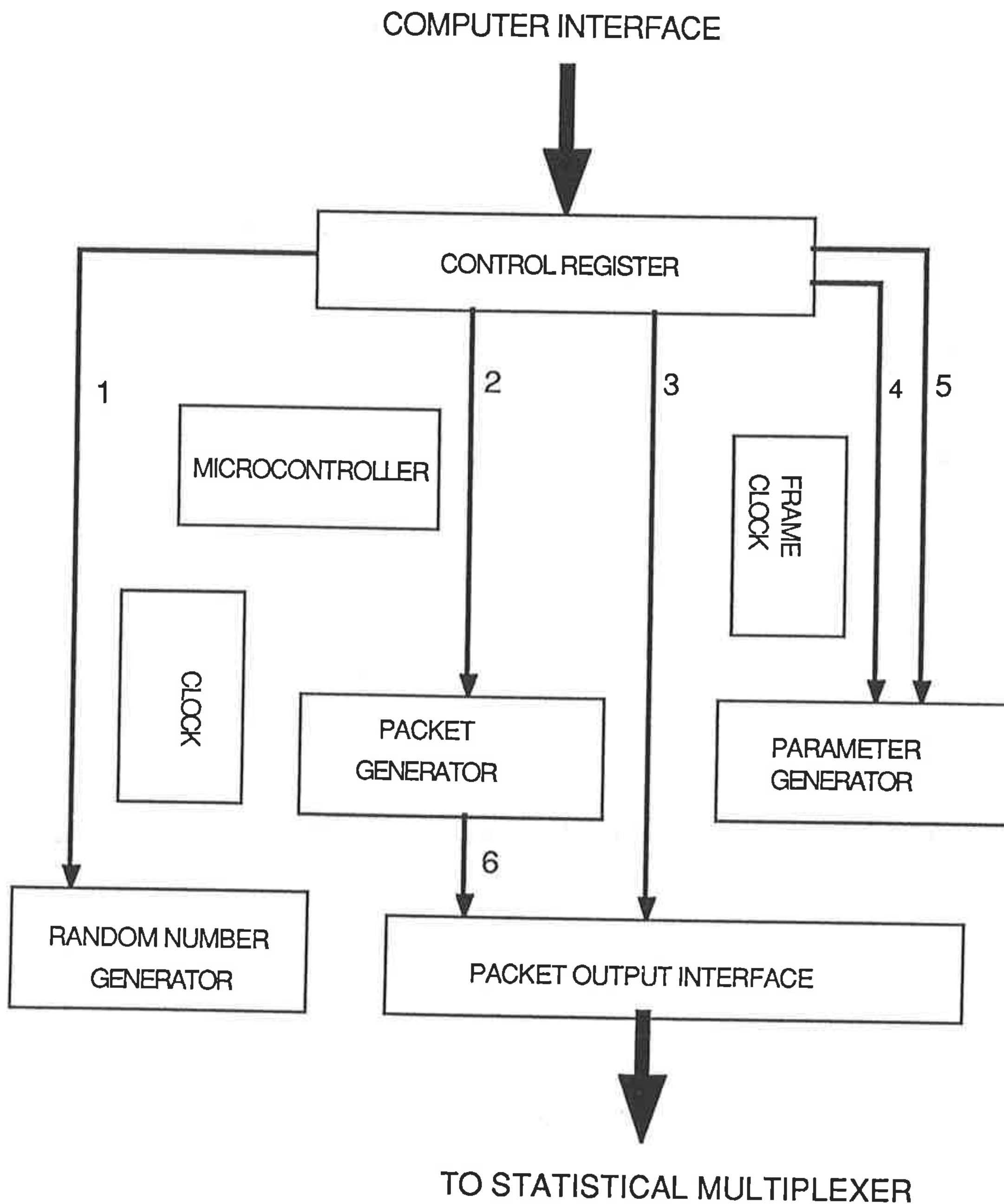
Pin 38: PACKET PRESENT STROBE OUTPUT*.

Pin 40: ACKNOWLEDGE INPUT*.

3.7 The control route

The source generator elements will operate under supervision of the control computer. This will control the call level processes, operation mode changes, and possibly some parameter changes during calls. In order to describe how the control computer controls each source generator operation, a control route is outlined in Figure 3.24. The external control computer writes the control bits to the control register via the VMEbus. These bits are SOURCE ENABLE* bit, INITIALIZATION* bit, FIXED CLASS ENABLE* bit, FIXED ADD ENABLE* bit and COMP*/COUNTER bit.

The SOURCE ENABLE* is provided to the packet output interface for enabling the PPSO* signal. If the SOURCE ENABLE* is inactive, the PPSO* will be high impedance. This source generator is in the off state.



1. INITIALIZATION*, 2. COMP*/COUNTER
3. SOURCE ENABLE*, 4. PARA CLASS ENABLE*
5. PARA ADD ENABLE* 6. PACKET OUT

Figure 3.24: The control route

The control computer uses this bit to switch on or off each of the generator boards.

The INITIALIZATION* bit is provided to the random number generator to select the operation mode of the random number generator. When this bit is LOW, the generator will operate in initialization mode. The shift register feedback path is modified so that the generator register sequences are randomised. When this bit is HIGH, the generator will operate in normal operation mode. Two 8-bit random numbers are brought out every cell cycle.

The FIXED CLASS ENABLE* and the FIXED ADD ENABLE* bits are used to select the packet header class and address parameter. These bits are provided to the parameter generator which contains packet specific (fixed or random) destination address as well as packet class parameter. When the FIXED CLASS ENABLE* is LOW, the parameter generator selects the three bit class parameter given by the control computer. If this bit is HIGH, the parameter generator outputs the random class which is a 3 bit random number from the random number generator. If the FIXED ADD ENABLE* is LOW, the parameter generator selects the five bit address parameter from the control computer. When the FIXED ADD ENABLE* is HIGH, the random address is selected as the packet header address parameter.

The COMP*/COUNTER bit is provided to the packet generator to select the comparator or counter operation mode. When this bit is LOW, the comparator operation mode is selected. If this bit is HIGH,the counter mode is selected.

When the packet generator decides to output a packet in a cell, it will produce an active PACKET OUTPUT signal. The PACKET OUT is provided to the packet output interface to cause one packet to be output in this cell.

3.8 The data transfer route

This section shall describe how the source generator transfers internal data. The data transfer route diagram is given in Figure 3.25.

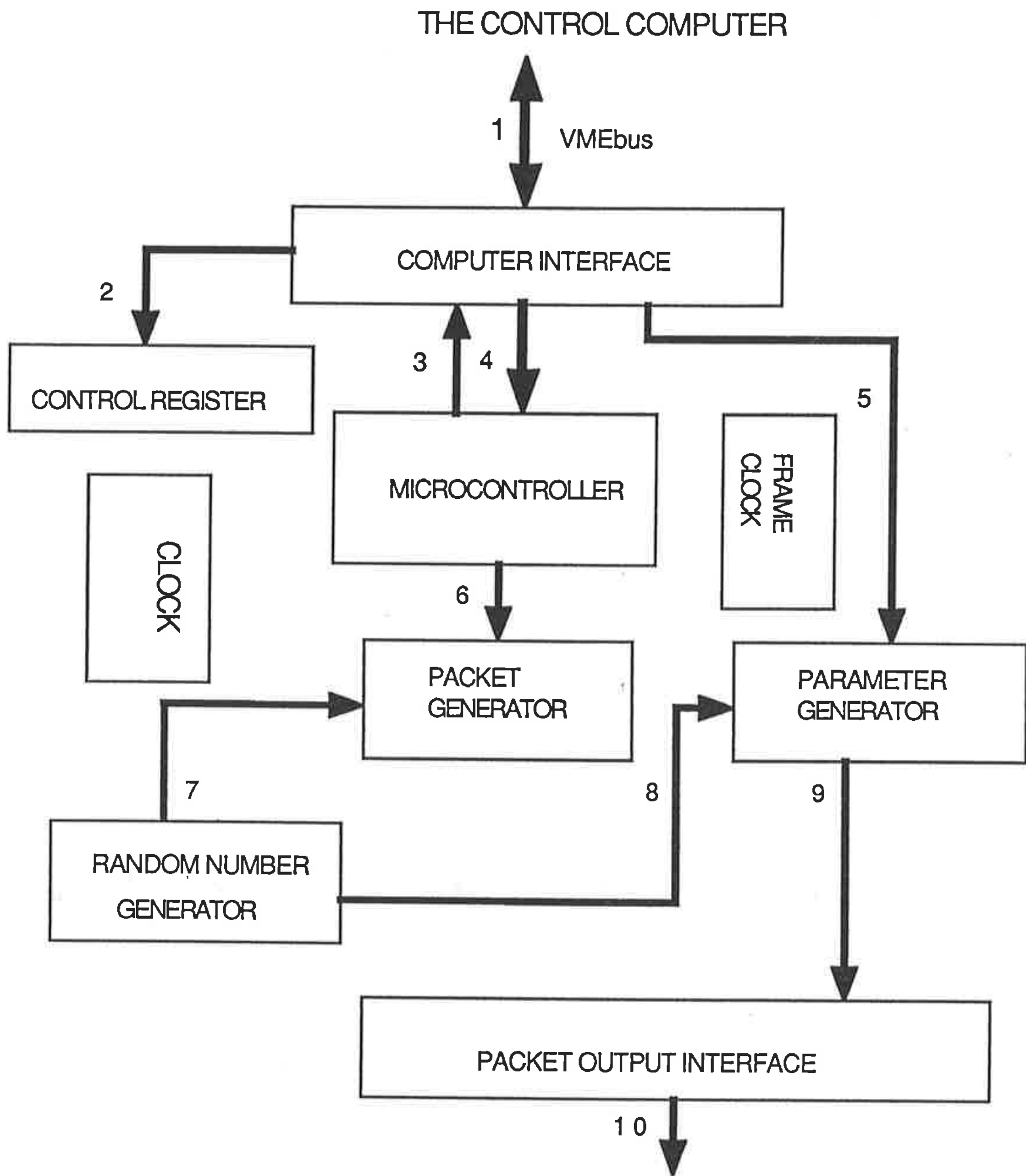
The external control computer transfers data to the generator board via the VMEbus. The control computer can directly transfer data to four registers on the board. These blocks are the control register, the parameter generator, port C and port B of the microcontroller.

The control register stores the control bits which control the generator operation. The fixed address and class parameter are stored in the parameter register. The control computer may write a 8-bit data to the port C of the microcontroller to modify the program stored in its EEPROM. The control computer also may read the rate parameter from the port B.

The random number generator brings out two 8-bit random numbers in every cell cycle. One is provided to the packet generator for the magnitude comparison. The other is provided to the parameter generator to be used as the random class and address of a packet header.

The microcontroller generates the rate parameter by executing the program in its EEPROM. The 8 bit rate parameter is output to port B and is updated every frame cycle. This parameter is provided to the packet generator to determine the rate of the packet generation.

The parameter generator generates a 16-bit packet header parameter. These are 8 bit packet control information and 8 bit packet identification. The control information includes 5 bit destination address and 3 bit packet priority class parameter. The 8 bit fixed class and address parameter is stored in the parameter register, and 8 bit random class and address parameter is stored in another register. We can choose to use fixed or random parameters



1. 8-bit VME bata bus,
2. Control bits,
3. 8-bit K(n),
4. 8-bit parameter,
5. 8-bit packet header parameter,
6. 8-bit K(n),
7. 8-bit random number,
8. 8-bit random number,
9. 16-bit packet header,
10. 16-bit packet header.

Figure 3.25: The data transfer system

as a packet header address and/or class parameter. The packet identification information is a 4-bit source ID and a 4-bit packet ID which will be generated internally by the parameter generator. The parameter generator provides the 16 bit packet header to the packet output buffer. On the PACKET OUT rising edge, the packet header parameter is latched into the output buffer.

Chapter 4

Software

4.1 Introduction

To study the nature of traffic in an ATM network, the testbed requires a number of source generation machines to generate a mixture of different traffic types. In order to generate traffic with realistic characteristics, we need to understand the nature of the individual traffic streams; that is, how each network user is using the services in terms of access rate, holding time, and proportion of different services. This requires source models to be developed for individual sources.

In this chapter, the model for encoded packet video based on model proposed by Maglaris et al [5], is developed. The program for this model also is developed using fixed point arithmetic. The packet video source may be simulated using the source generator hardware and this program. The source generator hardware may also allow a wide range of other models to be used.

4.1.1 Packet video model

In the next few decades, interactive video will become an increasingly important component of media communications because of increasing user demand for this service, and rapid advances in compression coding algorithms and VLSI technology [10]. An ATM environment can provide a high degree of flexibility in video communication, taking advantages of the inherent burstiness of video information. Variable rate transmission can be achieved in ATM networks. The following advantages are expected from variable rate transmission video signals: (1) reduction of delay; (2) constant video quality; (3) user control of video quality [10].

There are a large number of recent publications on variable rate video models. Among these models, Maglaris' model which was described in his published paper [5] is the best one for the hardware simulation. In his model, the information bit rate $\lambda(t)$ depends on the compression algorithm and the nature of the video scene. For a scene without abrupt movement such as the head of a person in a videophone, the bit rate was found experimentally to have a bell-shaped probability density, and to exhibit significant correlations over an interval of several frames. In Maglaris' model, the instantaneous bit rate is given in bits/pixel. There are 250,000 pixels per frame and 30 frames per second for the particular experimental setup used.

The bit rate for Maglaris' model in bits/pixel (averaged for each frame) is shown in figure 4.1 (from [5]). It shows 300 frames (10 seconds) of the test sequence. We assume it to be a continuous-time function since the frame period is very small compared to our time scale. The average value u over all 300 frames and the standard deviation σ were found to be $u = 0.52$ bits/pixel and $\sigma = 0.23$ bits/pixel. The maximum value of the bit rate was 1.41 bits/pixel and minimum 0.08 bits/pixel. Figure 4.2 (from [5]) shows a

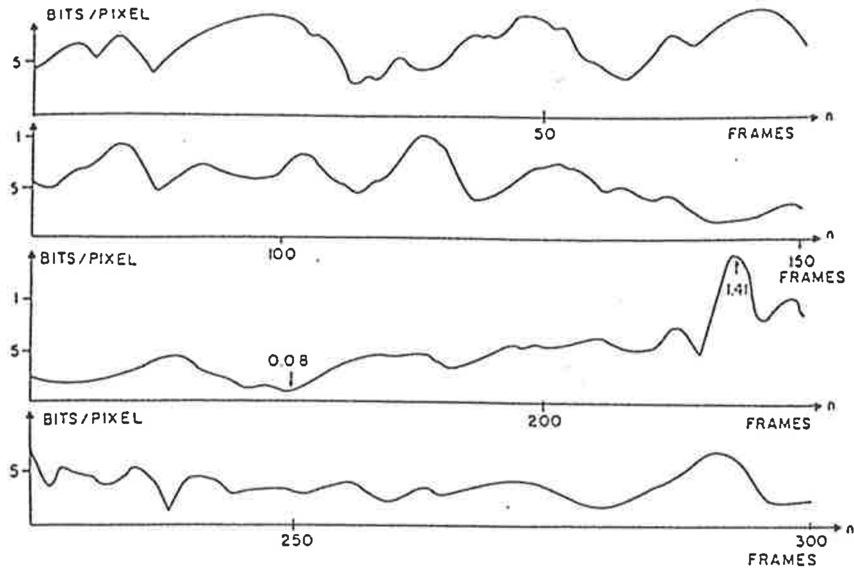


Figure 4.1: Coding bit rate

histogram of the values for the bit rate, an indication of the bell-like shape of its probability density function. The bit rate must be positive.

The autocovariance $C(\tau) = E\{\lambda(t)\lambda(t+\tau)\} - u^2$ of the sequence was evaluated and is shown in Figure 4.3 (from [5]) as a function of either time (τ) or frame (n) difference. The dotted line corresponds to an exponential fit of the form $\hat{C}(\tau) = \sigma^2 e^{-a\tau}$, with $a = 3.9s^{-1}$. This is accurate up to $n = 10$, except that $C(1)$ is slightly larger than $\hat{C}(1)$. The mismatch for $n > 10$ is less meaningful.

Supposing the bit rate is a continuous-state, discrete time stochastic process. Let $\lambda(n)$ represent the bit rate of signal source during the nth frame. A first order for $\lambda(n)$ has an exponential $C(\tau)$. It is autoregressive Markov process generated by the recursion relation [5]:

$$\lambda(n) = a\lambda(n-1) + bw(n) \tag{4.1}$$

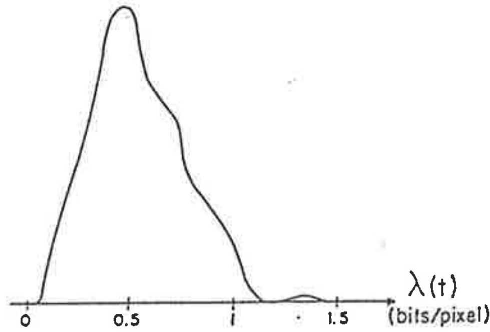


Figure 4.2: Bit rate histogram

where $w(n)$ is a sequence of independent Gaussian random variables, and a and b are constants. Assume that $w(n)$ has a mean η and variance 1. Further, assume that $|a| < 1$; thus, the process achieves steady state with large n. We can derivate expressions for the steady state average $E(k)$ and discrete autocovariance $C(n)$:

$$E(k) = br/(1-a) \tag{4.2}$$

$$C(n) = \frac{b^2}{1-a^2}a^n \qquad n \geq 0 \tag{4.3}$$

There is a reasonable match between experimental data and the autoregressive model. Maglaris obtains:

$$E(k) = 0.52 bits/pix \tag{4.4}$$

$$C(n) = 0.0536(e^{-0.13})^n (bits/pixel)^2 \tag{4.5}$$

The discrete autocovariance $C(n)$ is obtained from the experimental fit $C[\tau] = 0.0536e^{-3.9\tau}$ by sampling at $n/\tau = 30$ frames/s. Compare equations with measured data gives:

$a = 0.8781$; $b = 0.1108$; $\eta = 0.572$

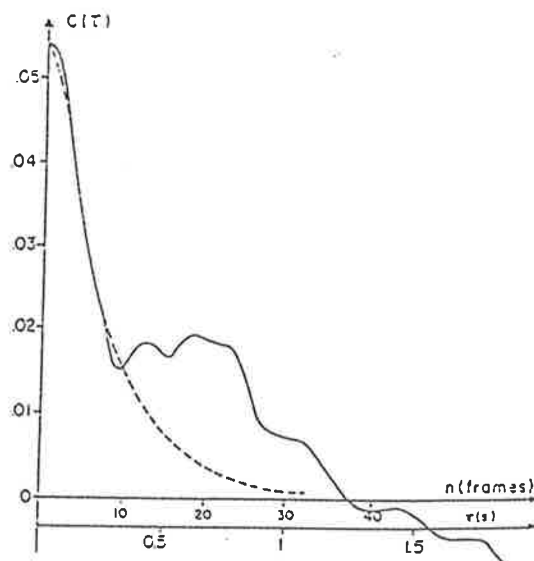


Figure 4.3: Autocovariance function

We use the equation $\lambda(n) = a\lambda(n-1)+bw(n)$ with parameters $a = 0.8781$, $b = 0.1108$, $\eta = 0.572$ to generate the bit rate for the video source simulator.

4.2 Programming

The MC68HC11A8 is an advanced 8-bit microcontroller. A program can be loaded into its internal EEPROM to generate the rate parameter $\lambda(t)$ according to the video source model given earlier. Because the microcontroller only has 8-bit parallel output ports, the maximum range of the output number is limited to a range from 0 to 255. Because of these requirements, we have to develop the model equation 4.1 to a more suitable form.

4.2.1 Gaussian random variables

For generating Gaussian random variables, uniform random variables must be generated first. Currently there are a lot of models to generate random variables. As the video model requires a long period and high speed random number, we select an additive generator as the random variables generator which is defined by [7]

$$X_n = (X_{n-24} + X_{n-55}) \bmod m, n \geq 55 \tag{4.6}$$

where m is even, and X_0,...,X_{55} are arbitrary integers not all even. The constants 24 and 55 in this definition were chosen to have the property that the least significant bits will have a period of length $2^f(2^{55} - 1)$ for some f, $0 \leq f < e$, when $m = 2^e$. The sequence X_n thus has a period of at least $2^{55} - 1$. At first glance this equation may not seem to be well suited to machine implementation, but in fact there is a very efficient way to generate the sequence using a cyclic list. Since it does not require any multiplication, the generator is very fast. The algorithm is given as follows [7]:

> Memory cells $Y[1], Y[2], Y[3], ..., Y[55]$ are initially set to values X_{54},X_{53},...X_0, respectively. j is initially equal to 24 and k is 55. Successive performances of this algorithm will produce the numbers X_{55},X_{56},... as output.
>
> 1. Set $Y[k] = (Y[k] + Y[j]) \bmod m$, and output $Y[k]$.
>
> 2. Decrease j and k by 1. If $j = 0$, set $j = 55$. If $k = 0$, set $k = 55$.

If we select $m = 256$, then in 6811 assembler, $Y[k] = (Y[k] + Y[j]) \bmod m$ becomes $Y[k] + Y[j]$ (clear overflow) $\rightarrow Y[k]$. This means that the algorithm does not require any multiplications.

Now we can obtain a reliable random sequence. The standard Gaussian variables can be generated using this random sequence and the algorithm is given as [11]

$$Z = \sum_{i=1}^{12} R_i - 6 \tag{4.7}$$

where sequence $[R_i]$ are random number between 0 to 1. If we use the random sequence $[X_i]$ of integer 0 to 255, then $R_i \rightarrow 1/255 * X_i$, and the algorithm becomes:

$$Z = (1/255 * \sum_{i=1}^{12} X_i) - 6 \tag{4.8}$$

If we set $w = Z + 0.572$, we should obtain Gaussian random variables w with mean 0.572 and variance 1 [11]. The resulting equation is:

$$w(n) = 1/255 * \sum_{i=1}^{12} X_i - 5.428 \tag{4.9}$$

4.2.2 The source model algorithm

By combining (4.1) and (4.9), the algorithm for generating bit rate is obtained:

$$\lambda(n) = a\lambda(n-1) + b(1/255 * \sum_{i=1}^{12} X_i - 5.428) \tag{4.10}$$

where λ is between 0 and 1.41, a is about 0.8781, b is about 0.1108. If each packet contains 512 bits and the frame rate is 30 frames per second, the source simulator will generate Y packets in one second. Y is given by

$$Y = 250000 * 30 * \lambda/512 \tag{4.11}$$

assuming 250000 pixels (bits) per frame.

The comparator mode

Each elementary source generator has two operation modes to generate the packets. One is the comparator mode which generates packets at random,

and the other is the counter mode which counts the number held in a counter register to determine packet generation. In the comparator mode, the packet generation is based on a fixed probability, which is the rate parameter. This rate parameter is generated by the microcontroller, and is latched into the probability threshold register. An 8-bit comparator compares the rate parameter value with the current random number and then decides whether a packet should be generated during a cell cycle. If the rate parameter value is greater than the random number, a packet will be generated, otherwise, no packet is generated in this cycle. Define K to be the rate parameter: a binary integer in the range of 0 to 255. F is defined as the source generator clock frequency. The packet generation rate Y is:

$$Y = (K/256) * (F/16) \tag{4.12}$$

Its unit is packets per second. Using (4.11) and (4.12) we get

$$(K/256) * (F/16) = 250000 * 30 * \lambda/512 \tag{4.13}$$

$$\lambda = \frac{512 * K * F}{250000 * 30 * 256 * 16} \tag{4.14}$$

Thus,

$$\lambda = \frac{K * F}{234375 * 256} \tag{4.15}$$

Then, combining (4.10) and (4.15) we obtain

$$\frac{K(n) * F}{234375 * 256} = a\frac{K(n-1) * F}{234375 * 256} + b(1/256 * \sum_{i=1}^{12} X_i - 5.428) \tag{4.16}$$

Multiplying both sides of (4.15) by $(234375 * 256)/F$ gives

$$K(n) = aK(n-1) + b\frac{234375}{F} * \sum_{i=1}^{12} X_i - b\frac{234375 * 256 * 5.428}{F} \tag{4.17}$$

Where, a is about 0.8781, b is about 0.1108. If F is 5MHz, the (4.17) becomes:

$$K(n) = 0.8781 * K(n-1) + 0.00519375 * \sum_{i=1}^{12} X_i - 7.2170688 \tag{4.18}$$

To ensure that the microcontroller works in real time, we choose to use fixed point arithmetic. If we multiply (4.18) by 200 then the number 0.00519375 will become integer. Thus the equation becomes:

$$200K(n) = 175.62 * K(n-1) + 1.03875 * \sum_{i=1}^{12} X_i - 1443.41376 \quad (4.19)$$

The fractional parts of the numbers here still have a significance in the final result. We get:

$$200K(n) = (176-0.38)*K(n-1)+(1+0.03875)*\sum_{i=1}^{12} X_i - 1443.41376 \quad (4.20)$$

$$200K(n) = 176*K(n-1)+1*\sum_{i=1}^{12} X_i - [1443.41376+0.38*K(n)-0.03875*\sum_{i=1}^{12} X_i] \quad (4.21)$$

Because the $[X_i]$ is a random sequence of integer 0 to 255, the average $\sum_{i=1}^{12} X_i$ is $12 * (255/2) = 1530$. Using (4.15), when F is 5 MHz the average K is $0.52 * 234375 * 256/F = 6.24$. Thus $1443.41375 + 0.38 * K(n) - 0.03875 * \sum_{i=1}^{12} X_i$ is about $1443.41375 + 0.38 * 6.24 - 0.03875 * 1530 = 1386$. Define $A = 176$, $B = 1$, $C = 1386$, the algorithm in integer form is:

$$200K(n) = AK(n-1) + B * \sum_{i=1}^{12} X_i - C \quad (4.22)$$

If $K(n) > 255$, set $K(n)$ =255; if $K(n) < 0$, set $K(n)$ =0. By modifying parameters A, B, C, the rate parameter $K(n)$ may be changed to allow simulating different video sources. The A, B, may be single precision in range of 0 to 255, and C may be double precision between 0 to 32768. The external computer can write A, B, C to the microcontroller to modify the rate parameter generation.

Using (4.12) we obtain the number of packets in the nth frame as

$$P(n) = [K(n) * F]/(256 * 16 * 30) \quad (4.23)$$

The program for the rate parameter generation is given in Appendix 4.1, which is written in Motorola 6811 assembler language. The procedure is:

1. Set the interrupt vectors; the XIRQ vector is directed to $K(n)$ output interrupt service routine ,and the IRQ vector is directed to the interrupt service routine which reads A, B and C from port C.

2. Generate a random initial value for 55 memory cells which will store 55 random variables. Set the initial value for some parameters.

3. Generate uniform random variables X_i. Multiplying random variables with parameter B, and get $B * \sum_{i=1}^{12} X_i$.

4. We define the memory RATE to hold $K(n)$. Calculate $AK(n-1) + B * \sum_{i=1}^{12} X_i - C$, then divide this value by 200, if the result is less than 0, set RATE is 0; if the result is greater than 255, set RATE is 255; if the result is between 0 and 255, test the remainder. If the remainder is less than 100, save the result to the RATE. If the remainder is greater or equal to 100, add 1 to the result then save to RATE.

5. Interrupt service routines. If an XIRQ interrupt occurs, write the RATE to port B. One rate parameter $K(n)$ is output. If an IRQ interrupt occurs, read data from the port C and save to the memory holding the parameter A, this will replace the parameter A, then change IRQ vector to make next IRQ jump to the routine which is to change the parameter B. Then change IRQ vector again to make next IRQ interrupt to replace the parameter C. After changing C, recovers the IRQ vector.

The counter mode

The source generator also may operate in the counter mode. In this mode, the rate parameter $K(n)$ is latched in the counter register at start of each frame cycle. The source generator generates one packet in every cell cycle while the number in the counter register is not zero. For every N packets which are generated, the number in the register is decreased by one. When the number in the register becomes zero, the source stops generating the packets in this frame. When next frame starts, the new rate parameter is latched into the register for next frame packet generation. Thus the number of packets in a frame is:

$$P(n) = (K(n) * N) \tag{4.24}$$

where $K(n)$ is the rate parameter.

The counter mode and comparator mode are two different modes, which are developed to allow the packet generation more flexible. For the same value of $K(n)$, the same number of packets should be generated on average in a frame. Using (4.23) and (4.24), we obtain:

$$[K(n) * F]/(256 * 16 * 30) = [K(n) * N] \tag{4.25}$$

Hence,

$$N = F/(256 * 16 * 30) \tag{4.26}$$

where F is the generator clock frequency, and the frame rate is 30 frames per second. In view of (4.26), N is determined by F and the frame rate. For a clock rate $F = 5$ MHz and the frame rate of 30 Hz, N is about 40.

Chapter 5

Program Download

5.1 Introduction

In chapter 4 the video source model was developed, and the program, written in 6811 assembler language was described. In this chapter we will describe how the program can be downloaded to the microcontroller. This involves development of a circuit and use of available software for download. The circuit is used to put the program into the MC68HC11A8. The download block diagram is given in Figure 5.1. This consists of five parts:

1. The microcontroller; the program will be downloaded to its EEPROM.

2. An IBM compatible PC; this uses available software to assemble 6811 assembler program to Motorola S1-record code, and download these code to the microcontroller.

3. ICL232 level translator [12]; this links the external RS-232 communication port and the TTL microcontroller interface.

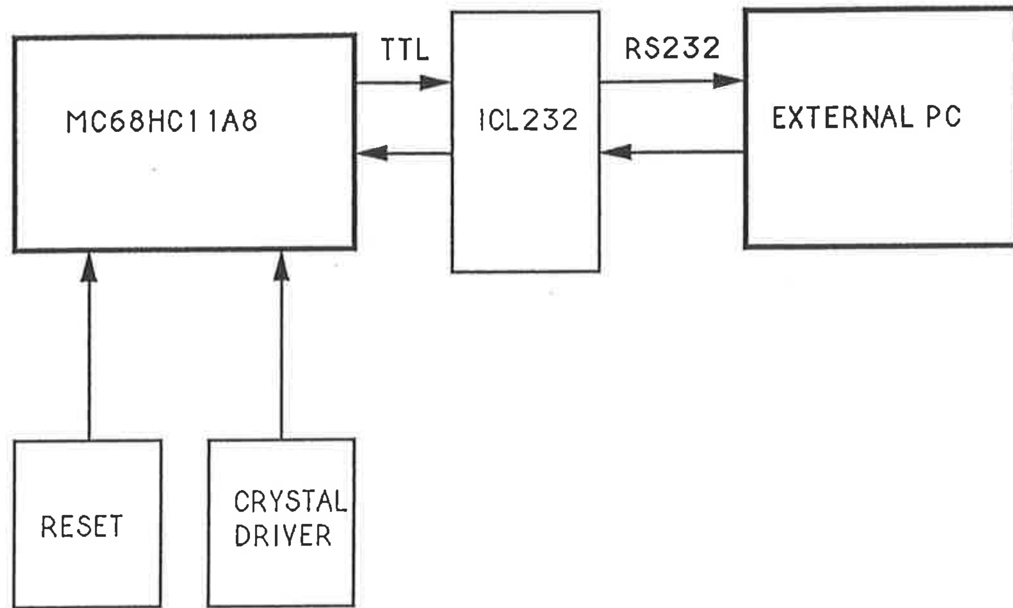


Figure 5.1: The download block digram

4. Reset circuit; this is a low voltage inhibit circuit which is required to protect against EEPROM corruption when circuit power is removed and applied.

5. Crystal Driver; this circuit controls the microcontroller internal clock generator circuitry.

The circuit diagram is shown in APPENDIX 5.1. The following paragraphs provide a further description of this circuitry.

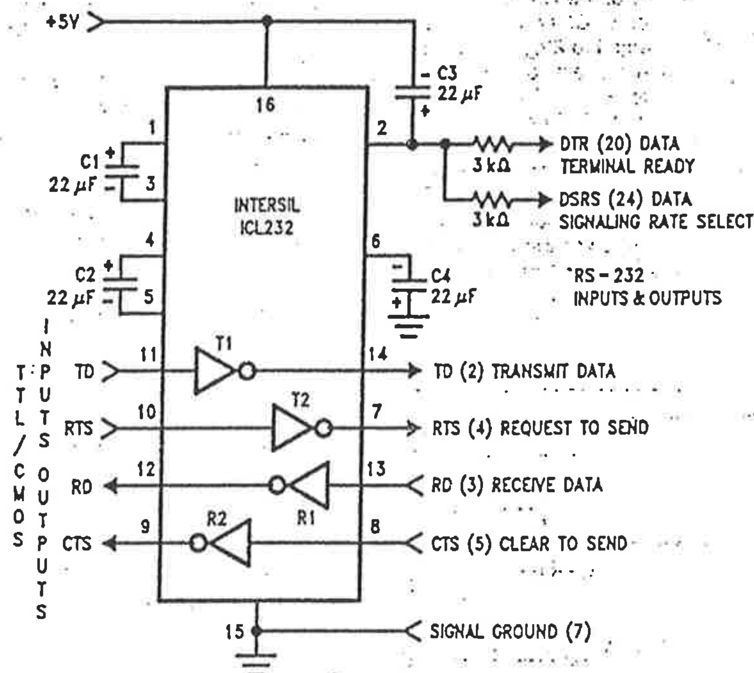


Figure 5.2: ICL232 (from [12])

5.2 Program download support circuit

5.2.1 Intersil ICL232 RS-232 to TTL Level Converter

A PC is used to download the 'S1' formatted object file to the microcontroller. Because there are different signal voltage levels between the PC RS-232 communication port and the microcontroller TTL serial communication port, we use the level translator device ICL232 for interfacing. The circuit diagram is shown in figure 5.2 (from [12]). The ICL232 has three sections, a dual transmitter, a dual receiver and a +5 v and ± 10 v dual charge pump voltage converter. The transmitter converts an TTL input level into a ± 9 v RS-232 output. The receiver converts an RS-232 input to a +5 v TTL output level.

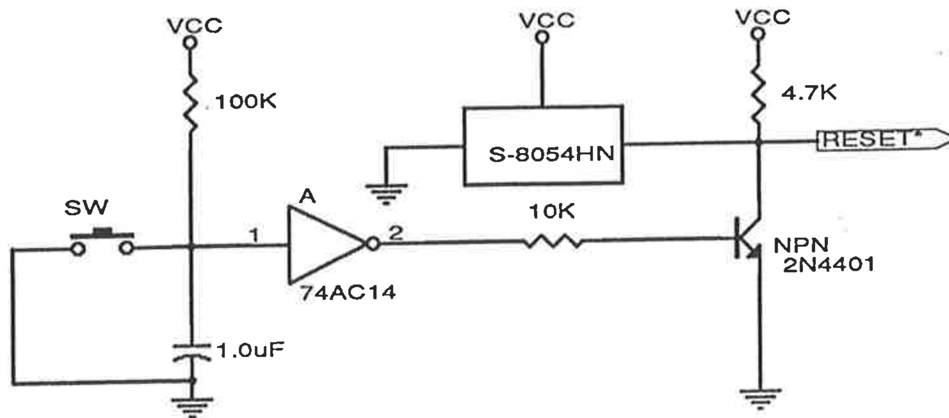


Figure 5.3: The reset circuitry

5.2.2 The reset circuitry

The figure 5.3 shows the reset circuitry (from [6]), which is a low voltage inhibit circuit. Since there is the EEPROM on chip, it is very important to control reset during power transitions. If the reset line is not held low while V_{CC} is below its minimum operating level, the EEPROM contents could be corrupted. Corruption occurs due to improper instruction execution when there is insufficient voltage to execute instructions correctly. Both the EEPROM memories and the EEPROM based CONFIG register are subject to this potential problem. This reset circuit which holds reset low whenever V_{CC} is below its minimum operating level is required to protect against the EEPROM corruption.

5.2.3 The crystal driver

The crystal driver circuit is given in Figure 5.4 (taken from [6]). The pins, XTAL and EXTAL, provide the interface for a crystal to control the micro-

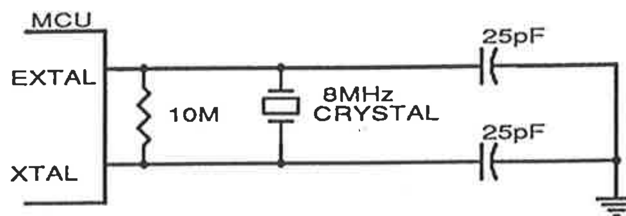


Figure 5.4: Crystal driver

processor internal clock generator circuitry. The frequency applied to these pins is four times higher than the desired the microcontroller operation clock E. We select the crystal frequency to be 8 MHz giving an E clock frequency of 2 MHz.

5.2.4 The MC68HC11A8 microcontroller

We have given a detailed description of the microcontroller in chapter 3. In the following paragraphs, we will describe some features of the microcontroller relating to the program download.

The program to generate the rate parameter is downloaded to the EEPROM of the microcontroller. In the MC68HC11A8 there are 512 bytes of EEPROM located at $\$B600$ through $\$B7FF$. The erased state of an EEPROM byte is $\$FF$. Programming changes ones to zeros. If any byte in a location needs to be changed from a zero to a one, the byte must be erased in a separate operation before it is reprogrammed. An 8-bit register PPROG is used to control the programming and erasure of the EEPROM. Another register BPROT prevents inadvertent writes to the CONFIG register and to the EEPROM.

For download in the special bootstrap operating mode, a boot loader

program is contained in a 192 byte bootstrap ROM. This ROM is enabled only if the microcontroller is reset in the special bootstrap operating mode. The boot loader program will use the SCI port to read a 256 byte program into RAM.

5.3 The software in the host computer

The program download software are available in the host computer which contains two sections:
(1) The IBM PC 6800/01/04/05/09/11 cross assemblers [13] (public domain). The cross assemblers are used to assemble the source program file and create an 'S1' formatted object file.

The assembler is named **as11.exe**. Command line arguments specify the filename to the assembler.

The 'S1' formatted object file is placed on the file 'm.out', the listing and error messages are written to the standard output. If you wish to save the listing, redirect it to another file using the DOS '>' sign followed by the file name. For example:

as11 source > listing

(2) ld6811.
The 'S1' formatted object file can be downloaded to the EEPROM of the microcontroller using the public domain program **ld6811.exe** which was written by Dr. K.W Sarkies. After connecting the RS-232C cable between the host computer and the download circuit board, we load and run ld6811. The monitor will display the message:

The MC68HC11 EEPROM Program Loader and Con-

figuration Program. By K.W Sarkies, 17/12/1990.

The MC68HC11 must be in special bootstrap mode, and the serial port connected to the PC port 1.

1. Load a bootstrap program to RAM and execute.

2. Load a program to EEPROM.

3. Read significant register.

4. Put into terminal mode.

C. Toggle to alternate Comms. port(1 or 2).

Q. Quit.

Enter Selection:

We enter selection "2", then the monitor prompts:

Ensure that the MC68HC11 is now reset, and in bootstrap mode, press any key to continue.

At this stage a 6811 program is written to the microcontroller. If the circuit connection is correct, the monitor will show:

Successful download, Enter filename:

Now we can enter the 'S1' formatted object file 'm.out'. The code will be downloaded to the EEPROM while it is echoed to the screen. For further information, please refer the document file **ld6811.doc** written by Dr. K.W Sarkies.

In summary, there are two stages to download the program to the EEPROM of the MC68HC11A8 microcontroller.

Stage one: use **as11.exe** to assembler the program. This will create an 'S1' formatted object file **m.out**.

Stage two: use **ld6811.exe** to download the 'S1' formatted object file to the EEPROM of the microcontroller.

Chapter 6

The Source Generator Debug and Test

6.1 General

As a component of the testbed system, the source generator will operate under supervision of a system controller. This control machine will allow each source to be controlled in a limited way to simulate the higher level switching processes which occur in multiservice networks. Because of the possible heavy load on such a controller, all but the highest levels of the source model are controlled within the elementary source. This controller therefore is only required to control the call level processes, and some parameter changes during calls.

Ironics IV-1602 single board VMEbus computer[14] based on the 68010 was selected to act as this controller. The IV-1602 has sufficient resources to allow it to be a system controller. It also can interface via RS-232 with a foreign host and a terminal. The terminal is the basic tool for hardware and

software debugging and testing.

Figure 6.1 shows the connection diagram for the source generator debugging and testing. Here we will describe a outline of IV-1602, the on-board IMON68 monitor, and the 68010 instruction set. Then we give a detailed description for the test of the source generator.

6.2 The IV-1602 computer

The Ironics IV-1602 is an MC68010 based microprocessor board designed specifically to function as a high-performance, single-board computer and VMEbus system controller. The available VMEbus system controller functions include a single level arbiter, the capability to drive the system reset and system clock lines, and the ability to handle the interrupt acknowledge daisy-chain. The IV-1602 can be configured to run in slot one of the system as the system controller, or it can be configured to run in any slot of a system which already has a system controller. In the testbed, it is configured to run as a system controller.

Figure 6.2 is the IV-1602 VMEbus single board computer (from [14]). Its block diagram is given in Figure 6.3 (from [14]).

The following are some major features of IV-1602:

- User input/output hardware; the IV-1602 provides a full range of input/output hardware. A Small Computer System Interface(SCSI) port is operated via the NCR 5380, allowing high-speed connection to floppy, windchest, and tape drives. Two channels of asynchronous serial communications are driven by the SCN68681 Dual Asynchronous Receiver/Transmitter(DUART). The DUART signals can drive data

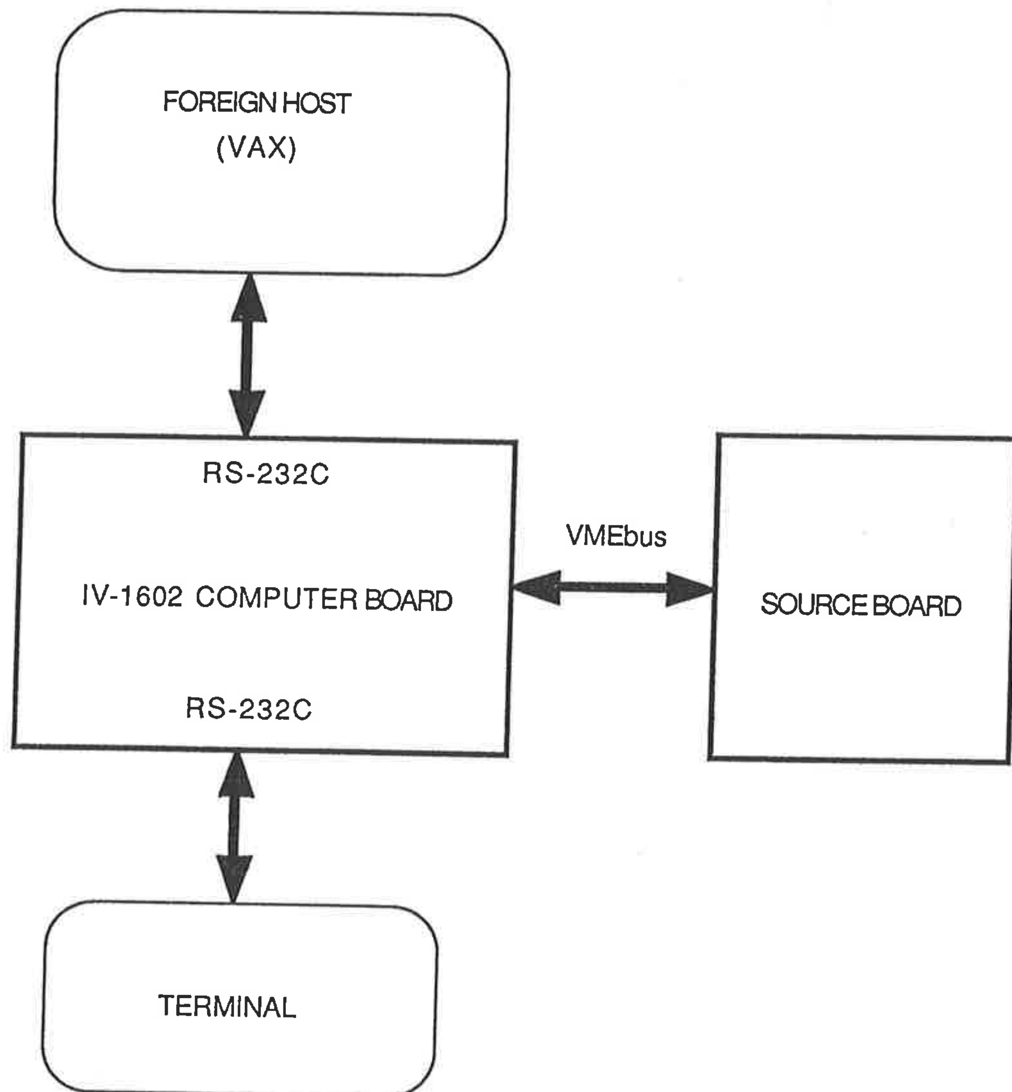


Figure 6.1: The debug and test block diagram

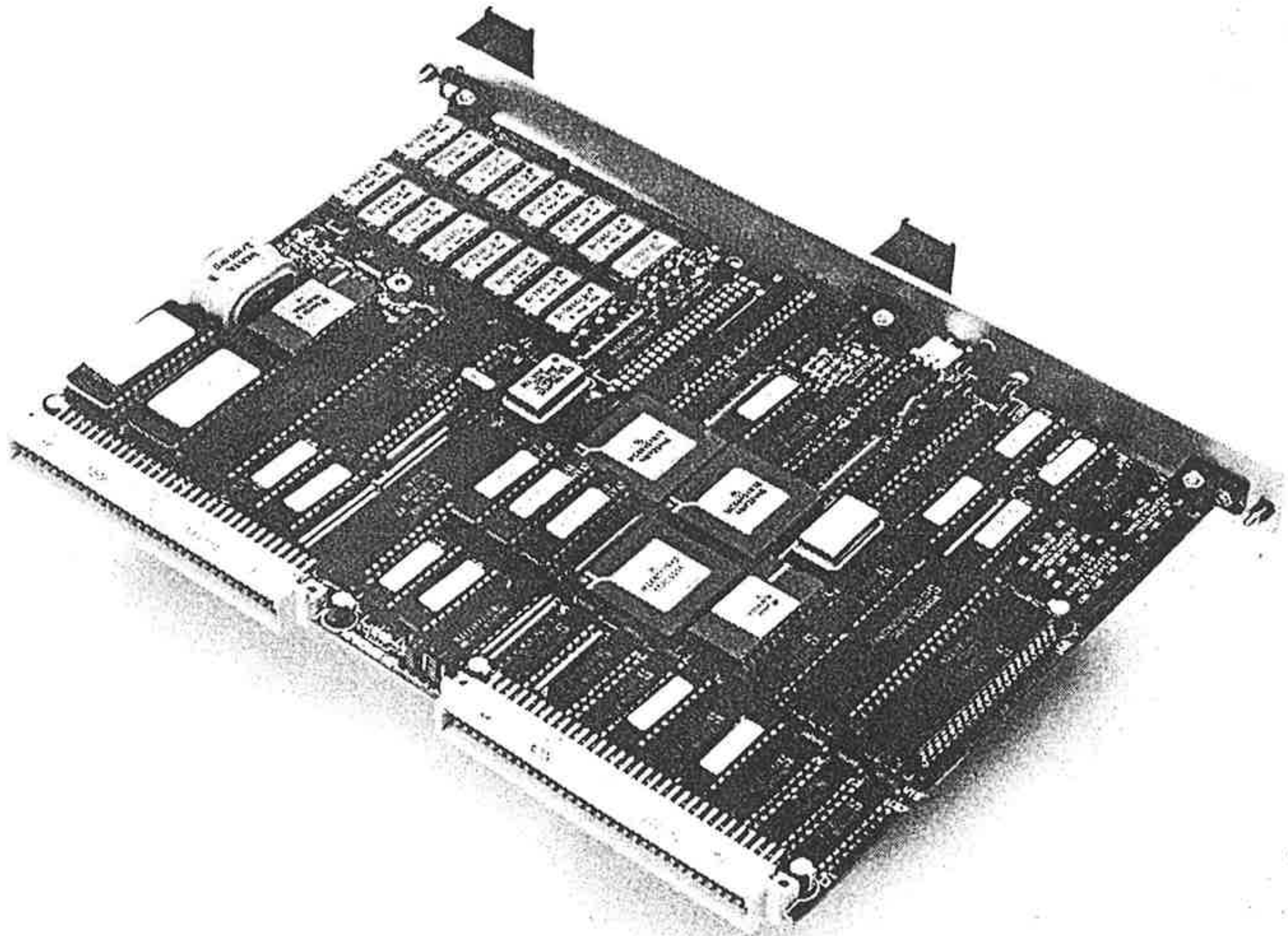


Figure 6.2: The IV-1602 VMEbus single board computer

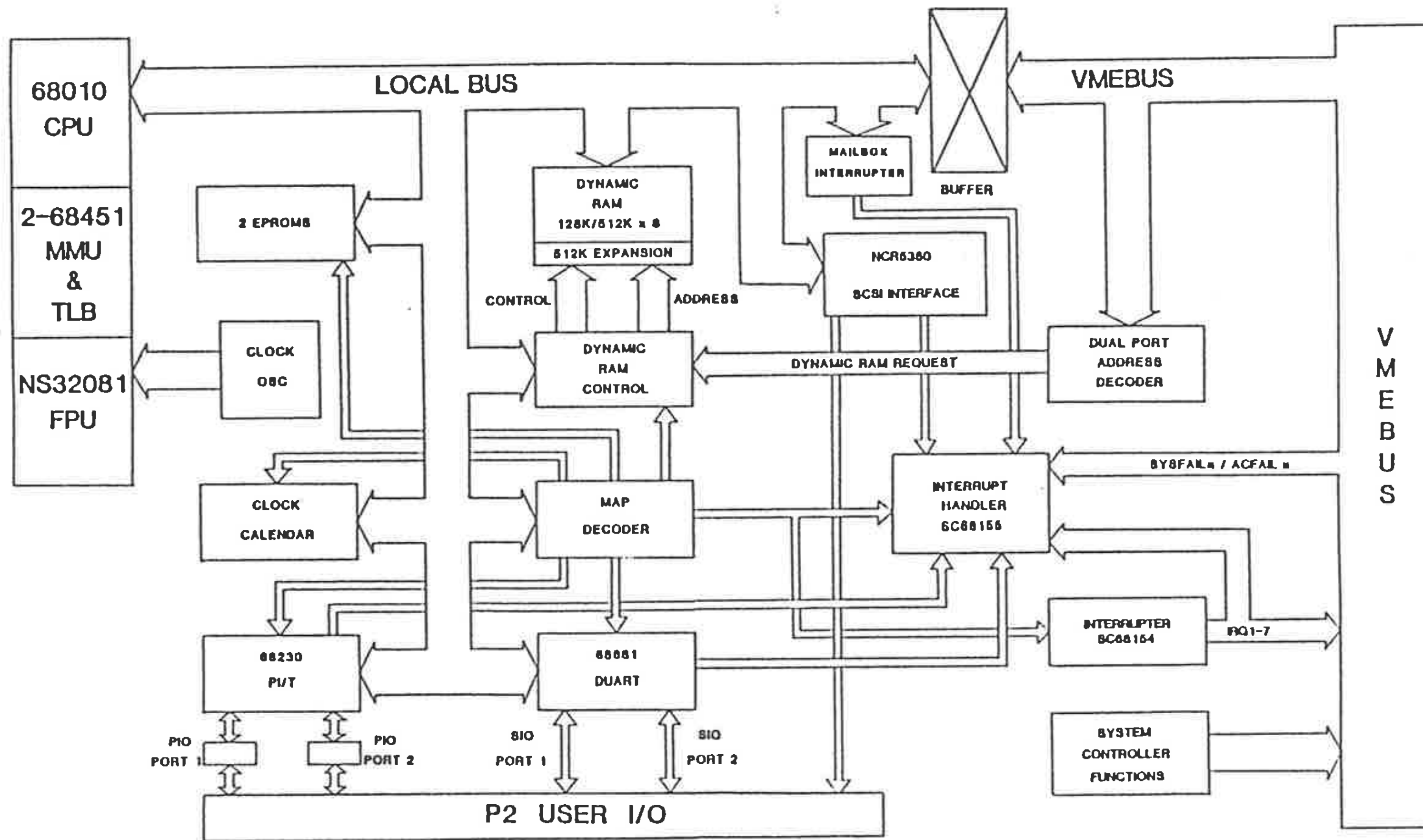


Figure 6.3: Block diagram

communication equipment (DCE) and data terminal equipment(DTE) versions of RS-232, RS-422, MIL-188, or current loop line protocols.

- VMEbus interface; the bus interface and interrupt processing circuitry of the IV-1602 is comprised of the Ironics optimized implementation of the standard Signetics VMEbus chip set. This includes a Bus Controller, an Interrupt Generator, an Interrupt Handler, and control and decode logic contained in programmable logic devices.

- Microprocessor; the processor on the IV-1602 is the MC68010; a 16/32-bit microprocessor that is fully compatible with the MC68000 family. It provides virtual memory support and have enhanced instruction execution timing. It is available at either 10 or 12.5 MHz. It provides 57 instruction types.

- Optional Floating Point Unit and Memory Management Units;

- Local memory, dual-ported; the IV-1602 provides 512K bytes of dynamic random access memory (DRAM), all of which is accessible to the local processor or from the VMEbus.

- Large EPROM space; the IV-1602 contains up to 256K bytes of erasable programmable read-only memory (EPROM).

- System addressing via Global Memory Map; the IV-1602 is capable of addressing any A16 or A24 board in a VMEbus system.

- On-board debug monitor;

- Rapid Strobe Deassertion and Mailbox Interrupts;

For further information of IV-1602, see *IV-1602 VMEbus Single Board Computer User's Manual.*

6.3 IMON-68 debug monitor

The IV-1602 contains on-board ROM which includes the IMON-68 monitor program[15]. The monitor uses the memory below $002000 for its workspace. All program will therefore only use memory above $002000.

6.3.1 Command line formats

The following is the standard input form for the IMON debug monitor command lines:

IMON(1602) v4.0 > <command> [<parameters>] [;<options>]

The first field is the prompt generated by the monitor. The second field is the command mnemonic. The third field contains parameters separated by spaces and can be either an expression or an address. The fourth field contains an option field.

6.3.2 IMON68 standard monitor command set

The monitor program provides a self-contained programming and environment. It interacts with the user through pre-defined commands that entered via the terminal. The commands fall into four general categories:

1. Commands which allow the user to display and modify memory.
2. Commands which allow the user to display or modify the various internal registers of the MC68010.
3. Commands which allow the user to execute a program under various levels of control.

4. Commands which control access to the various input/output resources on the board.

Table 6-1 (from [15]) is a summary of the monitor commands.

6.4 Instruction set summary

The IV-1602 single board computer is a 68010 microprocessor based microcomputer. The MC68010 is fully compatible with the M68000 family. It provides 57 instruction types[16], which form a set of tools that include all the machine functions to perform the operations. The complete range of instruction capabilities combined with the flexible addressing modes provide a very flexible base for program development. For detailed information about instruction set, see the reference [16].

6.5 Software

The control computer programs are developed to perform some basic tasks, such as to control the source board operation, modify the characteristics of the generated streams, and obtain feedback information for modifying the packet generation. These programs are all written in Motorola 68000 assembler language. They are:

1. **wrabc**: write A,B,C into the microprocessor;
2. **wrcontrol**: write the control parameter to the control register of the source board;

Command	Description
BF <address1> <address2> <word>	Memory block fill
BI <address1> <address2>	Memory block initialize
BM <address1> <address2> <address3>	Memory block move
BR <address>	Breakpoint set
BS <addrress1> <address2> <data> [mask] [;<option>]	Memory block search
BT <address1> <address2>	Memory block test
BU	Executes bootstrap
DC <expression>	Data conversion
DF	Display formatted registers
DP <address1> [<count>]	Dual-port memory test
DU <address1> <address2> [<text..>]	Dump memory**
GD [<address>]	Go direct
G [<address>]	Go
GO [<address>]	Go
GT <breakpoint address>	Go until breakpoint
HE	Help!
LO [;<options>] [=text]	Load**
MD <address1> [<count>] [;<options>]	Memory display
M <address1> [;<options>]	Memory modify
MM <address1> [;<options>]	Memory modify
NOBR [<address> <address>...]	Breakpoint remove
NOPA	Reset printer attach
PA	Printer attach
IO[n]	I/O configuration
OF	Display offset registers
OS	RS + SW combination
RS	Remote console reconfiguration
SG	Remote go request
SW	Remote/local console switch
TM[E] [<exit character>]	Transparent mode
T [<count>]	Trace
TR[C] [<count>]	Trace
TT [<breakpoint address>]	Temporary breakpoint trace
VE [=text]	Verify **
.Rn	Display register (1)
.Rn <data>	Set register (1)

Table 6.1: Generic monitor command descriptions

3. **wrparameter**: write the packet header parameter to the parameter register of the source board;

4. **rdrate**: read the rate parameter from the source board;

A complete listing of these programs is given in Appendix 6.1. The IV-1602 computer memory map is given in Table 6.2 (from [15]), which is used often when programming.

The program **wrabc** will write A,B,C (Maglaris' video model parameters, see Chapter 4 for detail) to the EEPROM of the microcontroller to change the video model parameters. This will modify the characteristics of the source model to allow the source board to generate traffic in a flexible way. The procedure is as follows:

- Step 1. Display the information and instructions.
- Step 2. Read the decimal data 'A' from user.
- Step 3. Convert the ASCII encoded decimal data to the hex.
- Step 4. Write to the microcontroller.
- Step 5. Read the decimal data 'B' from user, repeat Step 3, 4 to write 'B'.
- Step 6. Read the decimal data 'C' from user, repeat Step 3, 4 to write 'C', then end.

The program **wrcontrol** will write the decimal control number (single precision from 0 to 255) to the control register (a byte) of the source board. The following is the write procedure:

TABLE 1. IV-1602 Standard memory map and I/O base addresses

Address Range & Size		Addressed Space
end: FFFFFF base: FF0000	size: 64Kb	bus: VME devices: A16 D16
end: FEFFFF base: FA0000	size: 320Kb	bus: VME devices: A24 D16
end: F7FFFF base: F70000	size: 64Kb	bus: IV-1602 local devices: Local I/O
end: F6FFFF base: F60000	size: 64Kb	bus: IV-1602 local devices: MMU 1 (option)
end: F5FFFF base: F50000	size: 64Kb	bus: IV-1602 local devices: MMU 0 (option)
end: F4FFFF base: F20000	size: 192Kb	bus: VME devices: A24 D16
end: F1FFFF base: F00000	size: 128Kb	bus: IV-1602 local devices: onboard EPROM
end: EFFFFF base: 080000	size: 14.8Mb	bus: VME devices: A24 D16
end: 0FFFFF end: 07FFFF base: 000000	size: 1Mb (option) size: 512Kb (std)	bus: IV-1602 local bus: IV-1602 local devices: onboard DRAM

Table 6.2: The IV-1602 memory map

- Step 1. Display the information and instructions.
- Step 2. Read the decimal control number from user.
- Step 3. Convert the ASCII encoded decimal to the hex.
- Step 4. Write to the control register.

The program **wrparameter** will write the decimal parameter number to the parameter register (a byte). The number will be used as the packet header parameter. The program is same as **wrcontrol** except Step 4. Here, Step 4 is: write the packet header byte to the parameter register.

The program **rdrate** will read the rate parameter values which appear at port B of the microcontroller, then save and display these values. This data may be used to analyse packet generation for purpose of modifying the characteristic of the generated streams or for testing. As the reads are asynchronous, a value is displayed only when a change is detected. The read procedure is:

- Step 1. Display the program title information.
- Step 2. Read the 8-bit rate parameter from the source board.
- Step 3. Display the hex rate parameter on the monitor.
- Step 4. Save the hex rate parameter to the memory at address $(006000)+.
- Step 5. Read the rate parameter again.
- Step 6. Compare the current rate parameter with the earlier one.
- Step 6. If equal, go to step 5; otherwise, go to step 2.
- Step 7. Continue until 20,000 data read, then end.

6.6 Debugging and testing

6.6.1 Interfaces

As illustrated in Figure 6.1, the IV-1602 interfaces with the source generator, the terminal and the foreign host. The interface to the source board is the VMEbus interface. This uses the VMEbus Backplane Connector J1/P1. The interfaces to the terminal and the VAX computer are the RS-232-C interfaces, which are available on the Backplane Connector J2/P2. Two 25-pin D-connectors are required for connections. Two MC1488 drivers and two MC1489 receivers [17] are used to drive signals between devices. Figure 6.4 shows the interconnections made between the devices.

6.6.2 Program download

The 68000 assembler called **asm68k** is available on the VAX computer. The **asm68k** assembler uses standard mnemonics and generates assembled code in the form of S-records. S-records are the standard form for transferring data to the IV-1602 computer. The procedure to assemble and download a program to the IV-1602 is therefore as follows:

augean% asm68k filename
augean % < CTRL A >
IMON (1602) v4.0 > LO;X=cat filename.hex

The X flag used on the download command enables the transfer to be visible on the screen.

Error diagnostics from the assembler appears in the listing file just before the line containing the error. Error messages are meant to be self-

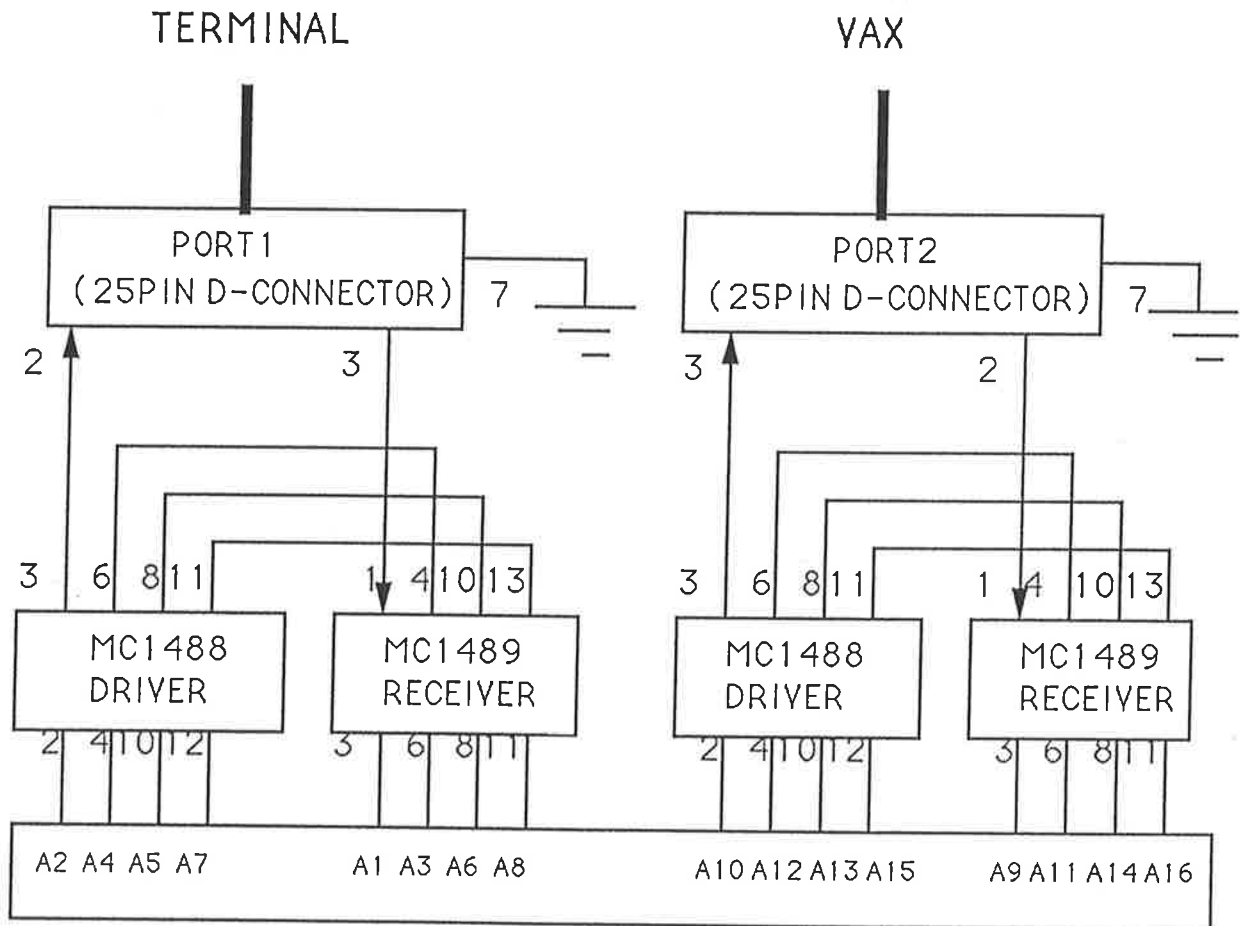


Figure 6.4: Connections

explanatory. Only if no errors occur, the output code is passed to **file-name.hex**. The procedure mentioned above, will cause the program to be downloaded to the IV-1602. For example, to download **wrabc**, the procedure is as follows:

augean % asm68k wrabc
augean % < CTRL A >
Imon (1602) v4.0 > LO;X=cat wrabc.hex

After this process, the program **wrabc** has been downloaded to the IV-1602.

6.6.3 Debug

The program which have been downloaded to IV-1602 are **wrabc** with start address $004000, **wrcontrol** with start address $004300, **wrparameter** with start address $004600, and **rdrate** with start address $005000. The address 006000 to 00*AE*1*F* are used to store the rate parameter values. The GO command causes the target program to execute. Run the **wrabc**, the monitor appears message as follows:

Imon (1602) v4.0 > GO $004000
This program is to write A, B, C to the microcontroller
of the source board, please input a decimal number:
Input A (0–255):
176
Write to the microcontroller is completed.
Input B (0–255):
1
Write to the microcontroller is completed.

Input C (0–32768):

1386

Write to the microcontroller is completed – write C low byte is completed.

Write C high byte is completed.

Write A, B, and C to the microcontroller is completed.

where 176, 1, 1386 are entered by user. The program in the EEPROM of the microcontroller will read these numbers to replace its existing A, B, C.

How can we test whether the program **wrabc** correctly writes A, B, C to the microprocessor? One solution is to make a small change to the 6811 program to write the rate parameter to the output port B. To do this, we change the XIRQ interrupt routine to write parameters A, B, and C consecutively to the port B. First run the program **wrabc** to write A, B, C to the microcontroller, then run the program **rdrate**. The monitor will display A, B, C which had been written to the microcontroller.

Run the program **wrcontrol**, the monitor will display message as follows:

Imon (1602) v4.0 > GO $004300

This program is to write the control number to the control register of the source board, please input decimal number.

Control number (0–255):

40

Write to the control register is completed.

The program will convert the decimal number to an 8-bit binary number, then write to the control register. 40 is equal to binary number 00101000.

By testing the logic level of outputs of the control register, we can determine whether the control number has been transferred correctly to the control register. The control number is used to select the operating modes. Table 6.3 provides the control numbers to be used to control the source generator board.

Run the program **wrparameter**, the monitor will display message as follows:

Imon (1602) v4.0 > GO $004600
This program is to write the packet header parameter
to the parameter register of the source board, please
input decimal number.
Packet header parameter (0–255):
255
Write to the parameter register is completed.

The program will convert this decimal number to an 8-bit binary number, then write to the parameter register. By testing the output pins of the parameter register, We can determine whether the packet header parameter has been written correctly to the parameter register. 255 is equal to binary number 11111111. So the parameter written is 11111111. The lower 5 bits are the fixed packet destination address parameter, and the upper 3 bits are the fixed packet class parameter.

Run the program **rdrate**, the monitor will display 20,000 rate parameter numbers.

Imon (1602) v4.0 > GO $005000
This program is to read the rate parameter from the
source board, 20,000 numbers are:

CONTROL NUMBER	OPERATING MODES
0	INITIALIZATION
16	SOURCE DISABLE
8---15	NORMAL OPERATIONS (Below)
8	FIXED ADDRESS, FIXED CLASS, COMPARATOR MODE
9	FIXED ADDRESS, FIXED CLASS, COUNTER MODE
10	RANDOM ADDRESS, FIXED CLASS, COMPARATOR MODE
11	RANDOM ADDRESS, FIXED CLASS, COUNTER MODE
12	FIXED ADDRESS, RANDOM CLASS, COMPARATOR MODE
13	FIXED ADDRESS, RANDOM CLASS, COUNTER MODE
14	RANDOM ADDRESS, RANDOM CLASS, COMPARATOR MODE
15	RANDOM ADDRESS, RANDOM CLASS, COUNTER MODE

Table 6.3: Operating modes versus control number

(numbers appear on the monitor)

Read is completed.

This program will read a set of consecutive rate parameter values from the port B of the microcontroller, while saving them in memory cells at start address $006000. Using these data, we can analyze the performance of the source, and then select suitable model parameter A, B, C.

By doing the tests, the programs **wrabc**, **wrcontrol**, **wrparameter** and **rdrate** all have very well performance.

The following commands are often used when debugging: DC: data conversion; DF: display formatted registers; GO: execute program; HE: help; LO; load; MD: memory display; MM: memory modify. For detail, see *IMON68 debug monitor user's guide.*

6.7 Test

To evaluate the performance of the packet video source software, we need obtain the mean and standard deviation of the packet generation bit rate, as well as the bit rate histogram and autocovariance function.

We also need test the correlation between the rate of packet generation and the rate parameter to evaluate the performance of the source generator hardware.

6.7.1 The software test

The programs for the test

After the 6811 program **RATE** given in Chapter 4 for the rate parameter generation has been downloaded to the MC68HC11A8 microcontroller, we can read the rate parameter values by running the program **rdrate**. The program **rdrate** will display the 20,000 rate parameter values on the monitor while saving to the memory locations from \$006000 to \$00$AE1F$. The command DU2 can be used to output S records of memory from \$006000 to \$00$AE1F$ to the VAX computer in a file called **fread**.

We develop the program **hexdata.p**, **convert.p**, **autocov.p** which are all written in Pascal (written by Linh Nguyen, modified by Z. Tan) to process the S-record data file **fread** and then to obtain the bit rate histogram and autocovariance function.

These programs are given in Appendix 6.2. The program **hexdata.p** will convert S-record data in file **fread** to the hex data, and then save to file **hexdata**. The program procedure is:

- Step 1. Display the information.
- Step 2. Reset input file **fread** and output file **hexdata**.
- Step 3. Get the S-record data from **fread**.
- Step 4. Convert the S-record data to the hex data, then put to the file **hexdata** while displaying the hex data.

The program **convert.p** will convert the hex date in the file **hexdata** to the decimal data, and then save to the file **data**. The program procedure is:

- Step 1. Display the information.
- Step 2. Get the hex data from the file **hexdata**. If the data is a hex number, put the data to an array, otherwise ignore.
- Step 3. Convert the hex number in the array to the decimal number.
- Step 4. Display the converted numbers and write them to the file **data**.

The program **autocov.p** will use the decimal rate parameter in the file **data** to obtain the bit rate histogram and autocovariance function. At first we should convert the rate parameter to the bit rate parameter, then draw the bit rate histogram frequency curve and the autocovariance function curve. When the generator clock frequency is 5 MHz, we get by using equation 4.15:

$$\lambda = (\frac{5000000}{234375 * 256}) * K\lambda = \frac{1}{12} * K \quad (6.1)$$

where λ is the bit rate, and K is the rate parameter. The program procedure is :

- Step 1. Display the information.
- Step 2. Get the rate parameter from the file **data**.
- Step 3. Display the data number obtained from the **data**.
- Step 4. Call the function **average** to obtain the MEAN of the bit rate.
- Step 5. Call the function **standdev** to obtain the STANDARD DEVIATION of the bit rate.
- Step 6. Call the procedure **histogram** to obtain the bit rate histogram.
- Step 7. Call the procedure **autocovar** to obtain the autocovariance function of the bit rate.

The function **average** is to calculate the mean of the rate parameter, and then convert to the mean of the bit rate. The function **standdev** is to calculate the standard deviation of the rate parameter, and then convert to the mean of the bit rate.

The procedure **histogram** is at first to calculate the frequency distribution of the rate parameter, and then change the scale to get the bit rate histogram. The procedure **autocovar** is to calculate the autocovariance function of the rate parameter, and the change scale to get the autocovariance function curve of the bit rate.

test procedure

The test diagram is given in Figure 6.1. At first we need log in the VAX computer, and make file **hexdata** and **data**. The test operation procedure is:

augean % cat > fread

ctrl A

Imon (1602): GO $005000 (execute the program rdrate)

Imon (1602): DU2 6000 AE1F

Imon (1602): TM

ctrl C

augean % pix hexdata.p

augean % pix convert.p

augean % pc autocov.p

augean % a.out > result

The simulation results will be saved in the file **result.** In the test, 20,000 rate parameter values are sampled, and the MEAN and STANDARD DEVIATION of the bit rate were found to be 0.51 and 0.22 for the values of A, B, C which matched those the Maglaris' model. Compare with the Maglaris' model where the MEAN and STANDARD DEVIATION are 0.52 and 0.23 respectively, the simulation results are quite well.

The bit rate histogram is shown in Figure 6.5, where the dotted line is the test result and the solid line is Maglaris' model bit rate histogram (from [5]). The bit rate density function in the simulation is also not exactly symmetric around its average u like in the Maglaris' bit rate histogram. Some slight deviation of the test is partly due to the conversion of Maglaris' model equation 4.1 into the program equation 4.22 where fixed point arithmetic is used. The truncation results in a slight change of a and b.

The test autocovariance function is given in Figure 6.6. The solid line is the autocovariance function of Maglaris' model (from [5]). The dotted line is the simulation autocovariance function, which is quite well matched to Maglaris' model, except that the measured value $\hat{C}(\tau)$ is a little smaller than Maglaris' model $C(\tau)$, for example, where the simulation $\hat{C}(0) = 0.049$ and the model $C(0) = 0.0536$ (8% different). It is due to we make the change of a and b when the model equation 4.1 was converted into the program equation 4.22. There $B = 1.03875$ was changed to $B = 1$, and $A = 175.62$ was changed to $A = 176$. Because Maglaris proposed his model using $C(n) = \frac{b^2}{1-a^2}a^n$ (from [5]), the smaller b will cause $C(n)$ to be smaller. Roughly, we made B about 4% smaller and A about 0.2% bigger. The change of B therefore effects $C(n)$ more than A. This causes the simulation autocovariance function $\hat{C}(n)$ to be smaller than Maglaris' model autocovariance function $C(n)$.

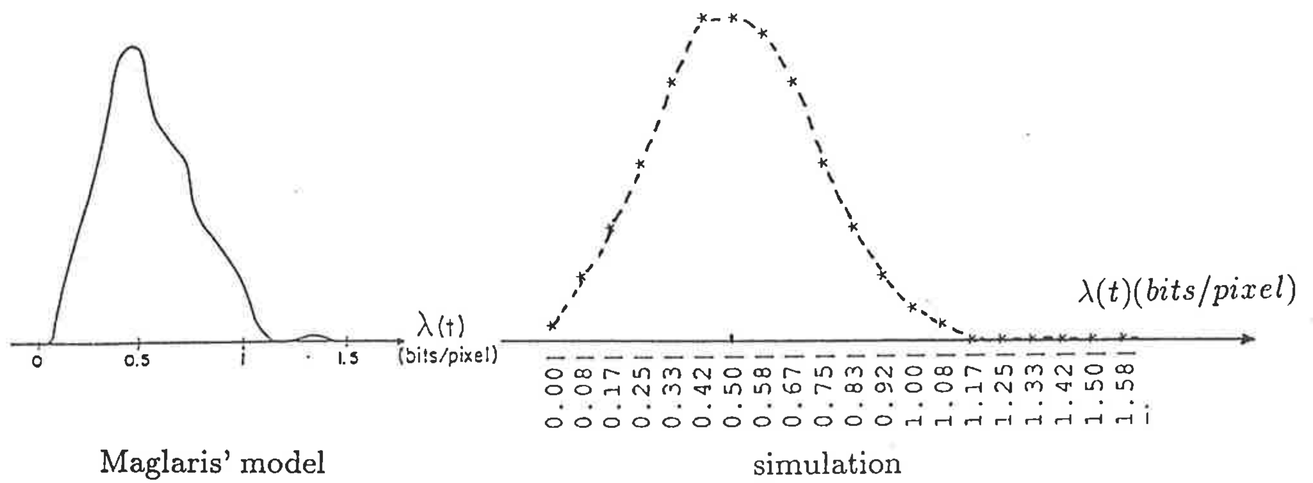


Figure 6.5: The bit rate histogram

6.7.2 The packet generation test

To evaluate the performance of the generator hardware, we need to test the line relation between the rate of packet generation and the rate parameter. According to the equations 4.12 and 3.4, the rate of packet generation in theory is:

$Y = K * F/(256 * 16)$ in the comparator mode.

$Y = 40 * K * R$ in the counter mode.

where Y is the rate of packet generation in packets/second, F is the source generator clock frequency, K is the rate parameter, and R is the frame clock frequency. In the test, we choose $F = 5$ MHz and $R = 30$ Hz. Thus the rate of packet generation is $Y = 1220.7 * K$ packets/second in the comparator mode, or $Y = 1200 * K$ packets/second in the counter mode.

In the source board, the PACKET OUT signal is used to drive packet output. One PACKET OUT will cause one packet to be generated. Therefore, in the test we use a counter to count PACKET OUT to obtain the

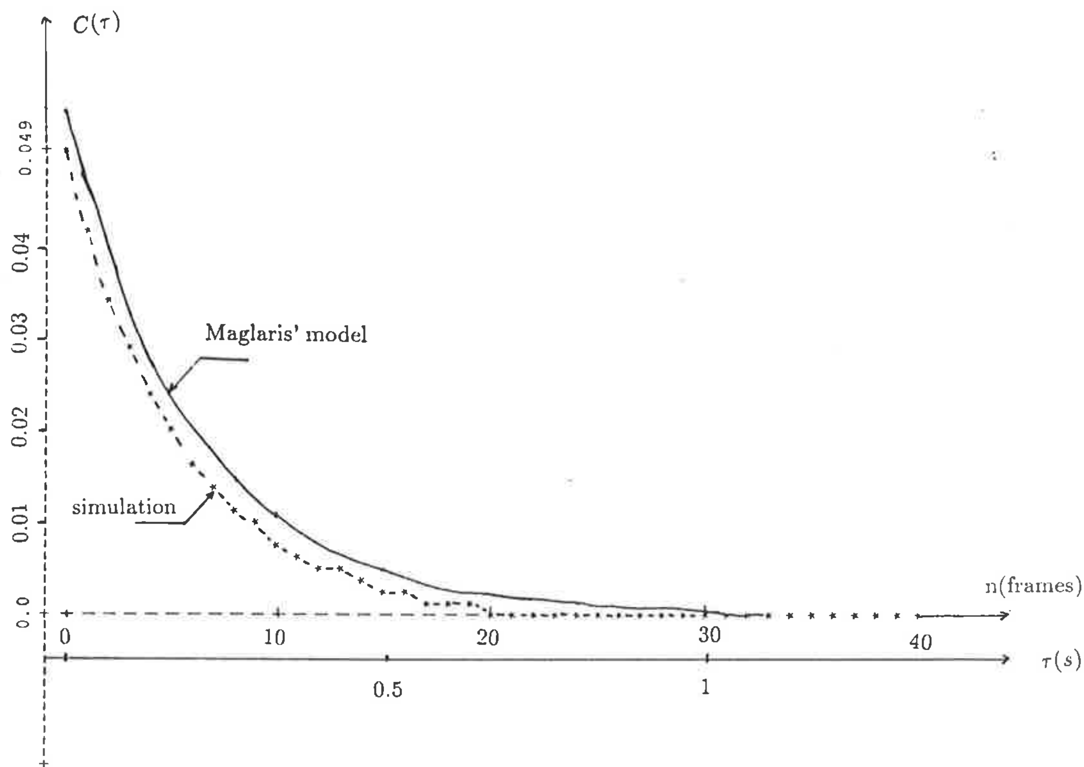


Figure 6.6: The bit rate autocovariance function

rate of packet generation. The test instrument layout is given in Figure 6.7. The counter is HEWLETT.PACKARD 5245L Electronic Counter, and the generator is WAVETEK 50 MHz Pulse Generator. We set the generator to output 5 MHz TTL pulses, and the counter to operate in frequency mode with time base in 10s. The display number on the counter will be average rate of packet generation in 10 second sampling period. Because the frame rate is 30 frames per second and the rate of packet generation may be up to 312 thousand packets per second, 10 second period is long enough to obtain an accurate average value.

If K is variable, it is very difficult to test the rate Y. We therefore modified the 6811 program **RATE** to output a constant K rather than variable

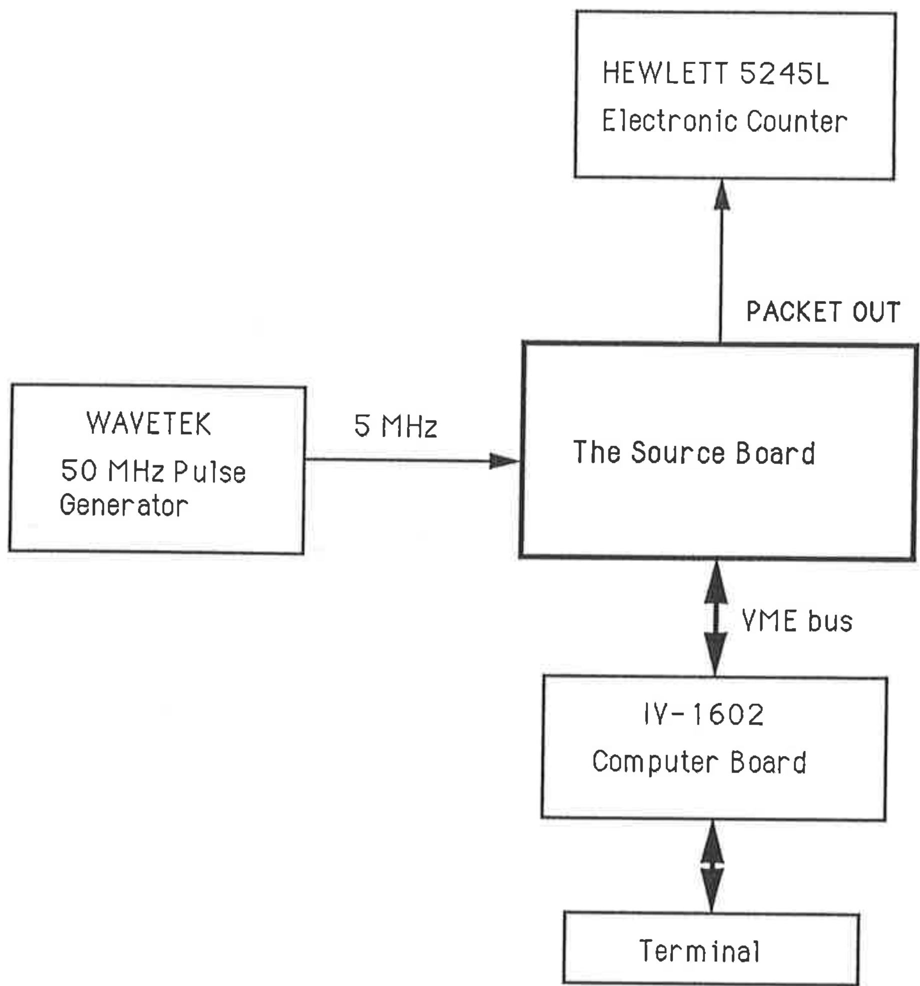


Figure 6.7: The test diagram

K. To do this, we change the XIRQ interrupt routine to write the constant parameter A to the port B. The A now is regarded as a constant K. The program **wrabc** was run to change the A value, and the program **wrcontrol** was run to change the generator operation mode to either the comparator mode or the counter mode. The test procedure is:

1. Set up the test instruments as in Figure 6.7, and assure the program which is to output parameter 'A' has been downloaded to the micro-controller.

2. Set the counter to operate in the frequency mode with time base in 10s, and set the pulse generator to output 5 MHz square waves.

3. Run the program **wrcontrol** to set the board to operate in the comparator mode.

4. Run the program **wrabc** to change the parameter A. If we write $A = K$, the display number in the counter will be the average rate of packet generation in 10 seconds for the given K.

5. After changing the value of K, and when the number in the counter was stable, then three consecutive values were read from the counter. The average value is the measured rate of packet generation. We tested the rate of packet generation for K equal to 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 20, 30, 50, 100, 150, 200, 255.

6. The program **wrcontrol** was run to set the board to operate in the counter mode. The steps 4 and 5 were repeated to obtain the average rate of packet generation in the counter mode.

In Maglaris' packet video model, most of the K values fell in the range of 0 to 15. We therefore took a large number of samples with K between 0 to 15.

The rate of packet generation in the comparator mode is shown in Figure 6.8 as a function of the rate parameter K. The dots are the simulation test values, and the line corresponds to the $Y = 1220.7 * K$. The error bars are too small to show in the figure. For instance, when $K = 10$, the test result is 12234 packets/s and the theoretical result is $1220.7 * 10 = 12207$ packets/s. The error bar is about 0.3%.

The rate of packet generation in the counter mode is given in Figure 6.9. The dots are our test values, and the line corresponds to the $P = 1200 * K$. The simulation result is again agree well with the theoretical result. When $K = 10$, for example, the test rate is 12045 packets/s and the theoretical result is $1200 * 10 = 12000$ packets/s, which has an error bar about 0.4%. Thus the error bars are too small to show in the figure.

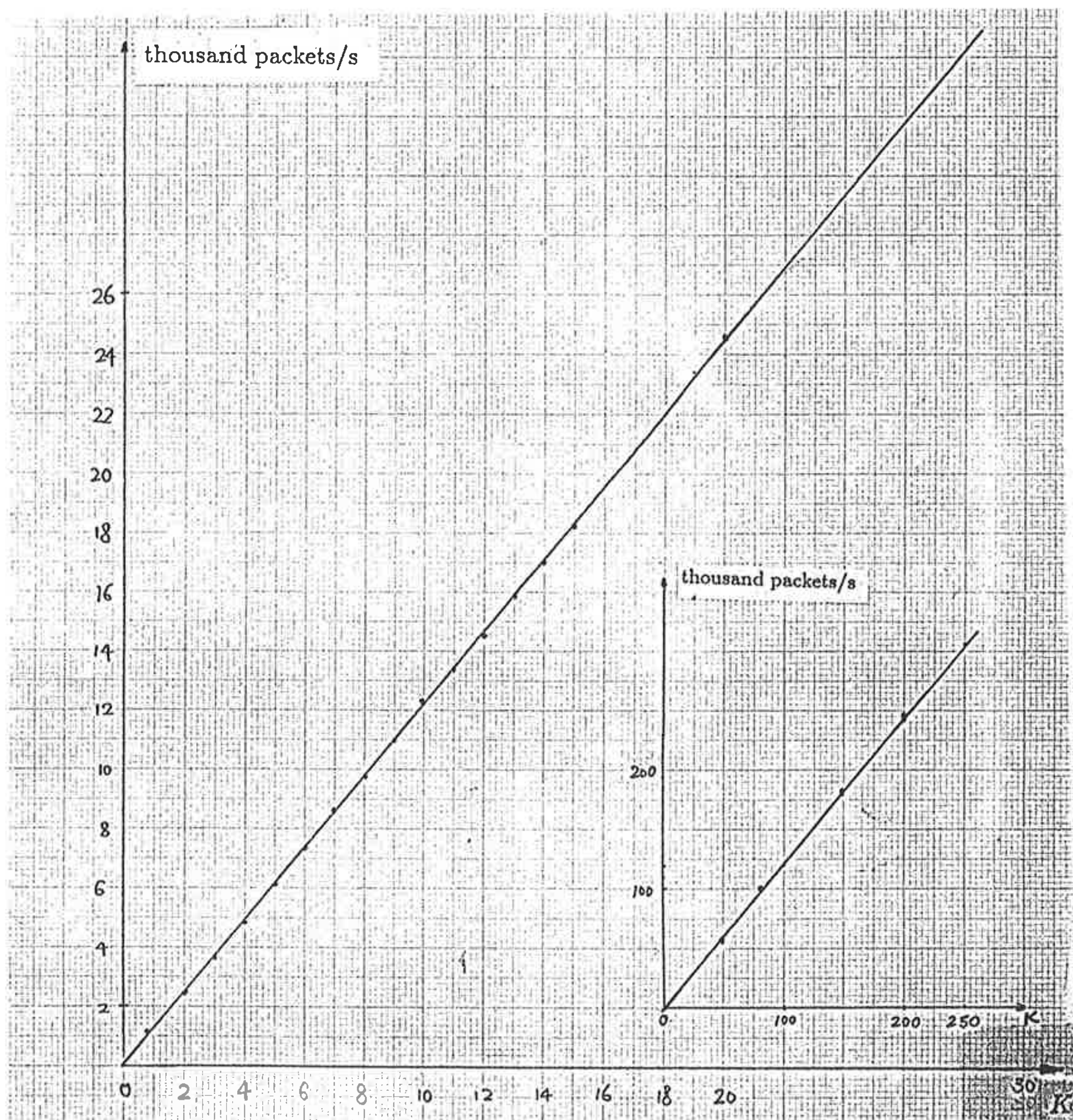


Figure 6.8: The rate of packet generation in the comparator mode

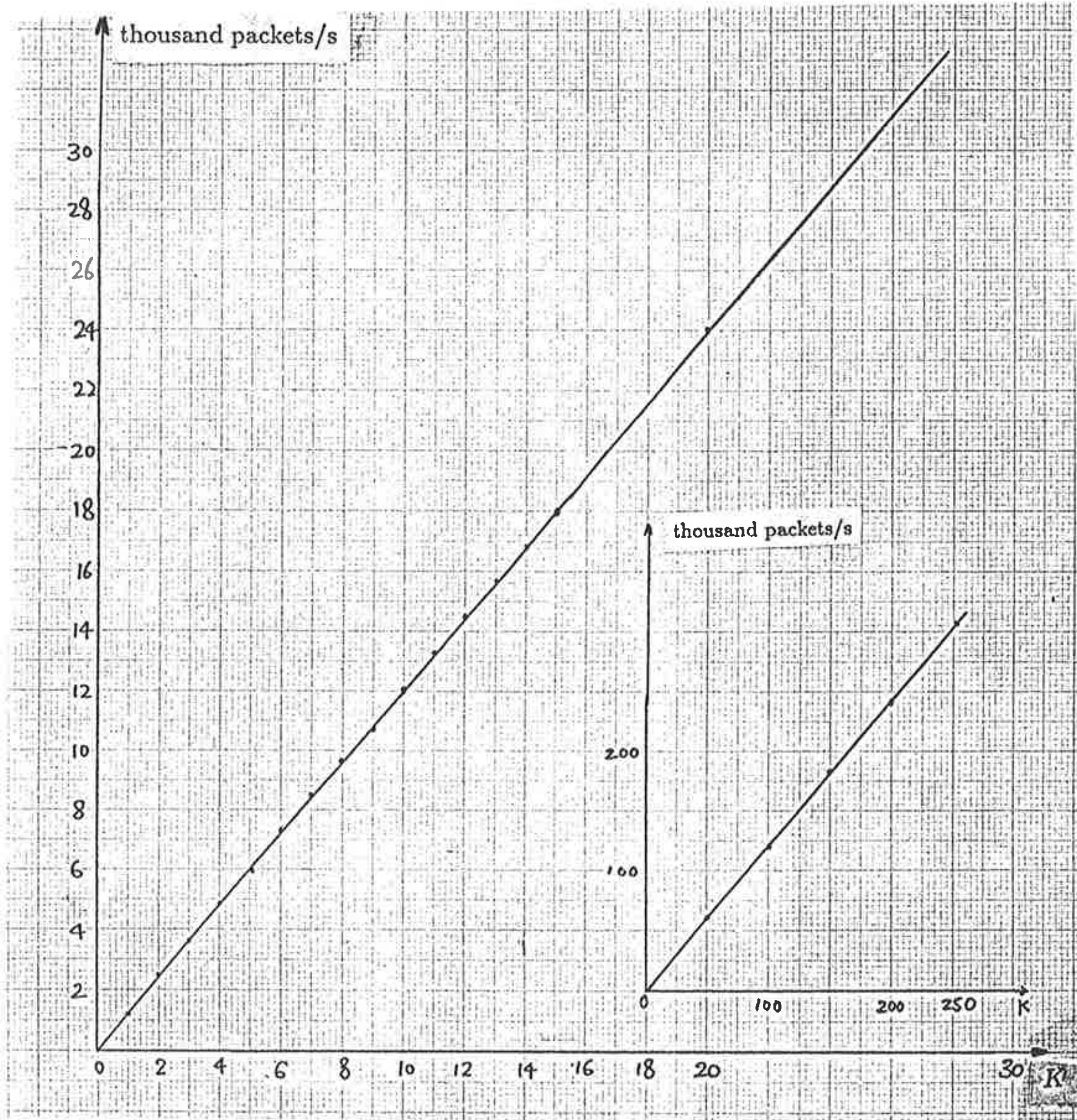


Figure 6.9: The rate of packet generation in the counter mode

Chapter 7

Conclusions

7.1 General

This dissertation covers the complete design of the source generator board, which simulates the video packet source generator based on the Maglaris' packet video model. This includes both the hardware design and the software development.

The source generator operates at 5 MHz frequency. This limits the rate of the packet generation to 312500 packets per second or 10416 packets per frame. This rate is enough to simulate the Maglaris' video model which has a maximum rate of 20654 packets per seconds. The Maglaris' video model simulates video frames which are transmitted using variable length encoding techniques. Thus each frame has a variable number of packets. The frame frequency is 30 Hz.

This source generator generates 16-bit packet headers in parallel which contain only the information needed for determining the nature of the packets generated. This includes packet class parameter, destination address pa-

rameters, packet identification and packet source identification. The packet generation can be done in two ways: the comparator mode or the counter mode. When using the comparator mode, the frame model is devised to specify a probability for generation of randomly distributed packets. Thus the packets are spread over the whole frame interval. When using the counter mode, the frame model specifies a packet count for generation of packets in a single burst at start of each frame.

The source generator operates under the testbed system control. The external control computer performs some controls by writing the control number to the control register, writing the packet header parameter number to the packet parameter register, and writing the parameters to the microcontroller to replace the software model parameter A, B, C to modify the rate of packet generation. This allows the source generator to generate packet traffic in a flexible way. The control computer also can read the rate parameter from the source board for use in analyses of the software model performance.

By testing the performance of the Maglaris' software model and the source board hardware, we determined that this generator had adequate performance to simulate the Maglaris' video packet model.

7.2 Future work

The source generator hardware may be used as a general purpose traffic generator hardware platform. By changing the model software in the microcontroller, the source generator may be used to simulate different type of sources, such as voice, image, interactive computer, database access and some specific sources.

To simulate very high speed traffic, it may be necessary to change the

source board internal clock. This source generator has an estimated upper operation frequency of 22 MHz (untested). This would allow us to simulate a source with a maximum rate of packet generation of 1,375,000 packets per second ($22,000,000/16 = 1,375,000$).

By changing the frame clock, we can obtain different frame models according to the requirements of the simulation. We may also use the internal counters in the MC68HC11A8 to update the rate parameter K rather than use the frame pulse. Thus a larger class of traffic models may be simulated.

Bibliography

[1] K.W Sarkies, "A Testbed for Generation of Multiservice Traffic", Internal Report, Department of Electrical and Electronic Engineering, The University of Adelaide, October 1989.

[2] John J. Kulzer and Warren A. Montgomery, "Statistical Switching Architecture For Future Services", Session 43 A Paper 1, ISS '84 Florence, 7-11 May 1984.

[3] J. Turner, "New Directions in Communications", IEEE Communications Mag., Vol24, No.10, Oct. 1986, pp.8-15.

[4] P. Temple, "An Event Capture Board for a Network Source Emulator", Final Year Report, Department of Electrical and Electronic Engineering, The University of Adelaide, 1989.

[5] B.Maglaris, et al., "Performance Models of Statistical Multiplexing in Packet Video Communications", IEEE Transactions on Communications, Vol. 36, No.7, July 1988, pp.834-843.

[6] Motorola INC, "MC68HC11A8 HCMOS Single-Chip Microcontroller", 1988.

[7] D. Knuth, "The Art of Computer Programming", Vol. 2, Seminumerical Algorithms, Addison-Wesley, 1981, pp.25-33.

[8] Motorola Inc., "SCHOTTKY TTL data", 1983.

[9] Motorola Inc., "VMEbus specification manual", Oct. 1981.

[10] Mitsuru Nomura, T.Fuji and N Ohta, "Basic Characteristics of Variable Rate Video Coding in ATM Environment", IEEE Journal on Selected Areas in Communications, Vol. 7, No.5 July 1989, pp.752-760.

[11] D. Knuth, "The Art of Computer Programming", Vol. 2, Addison-Wesley, 1969, pp.103-110.

[12] INTERSIL, "INTERSIL data manual", 1983, pp.11.36-11.41.

[13] E.J. Rupp, Motorola Inc., "A software translates assemble program", Nov. 1984.

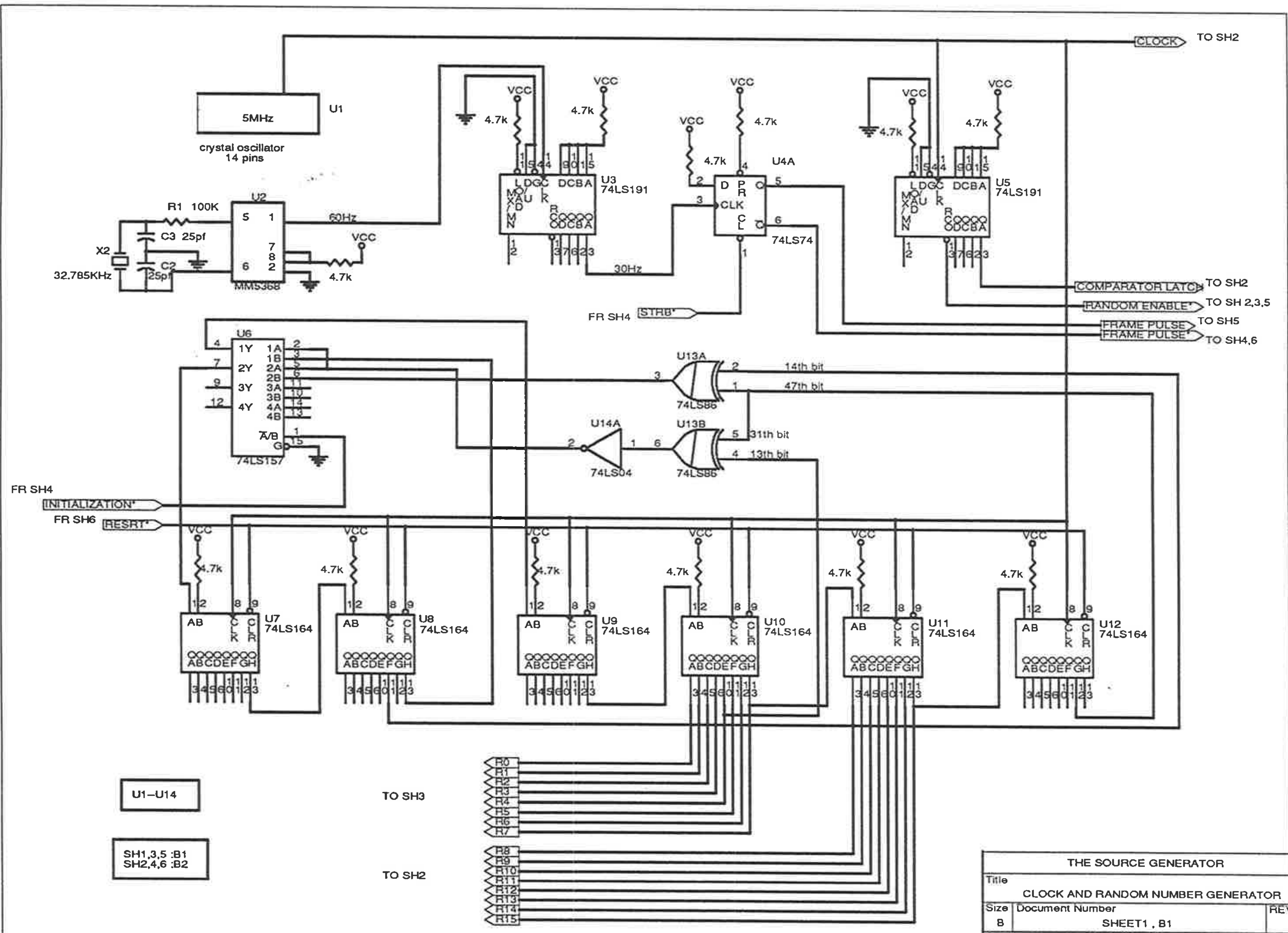
[14] Ironics Incorporated, "IV-1602 VMEbus single board computer", User's Manual, 1988.

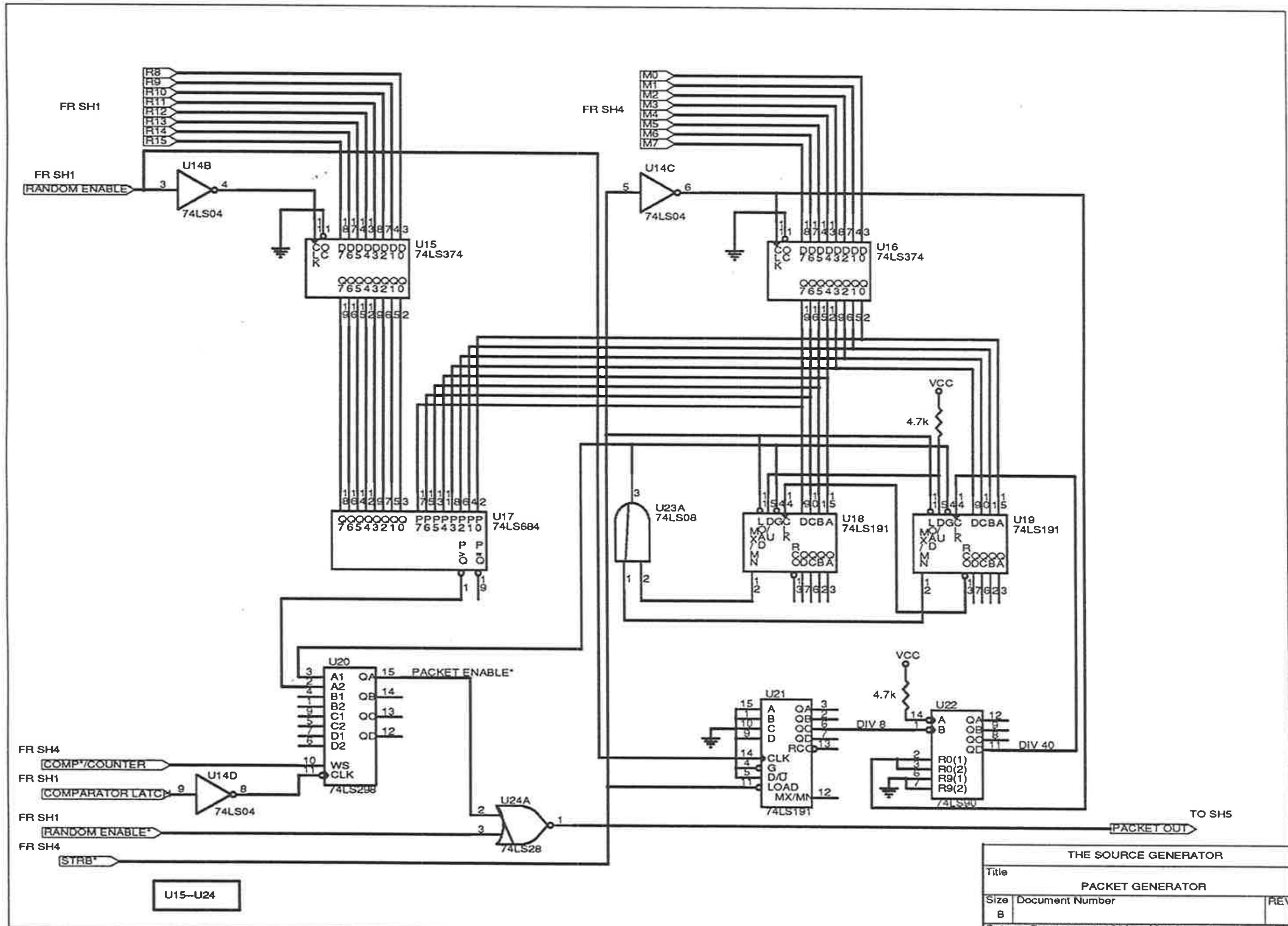
[15] Ironics Incorporated, "IMON68 debug monitor user's guide", 1986.

[16] Motorola Inc., "MC68000 16-bit microprocessor user's manual", 1982.

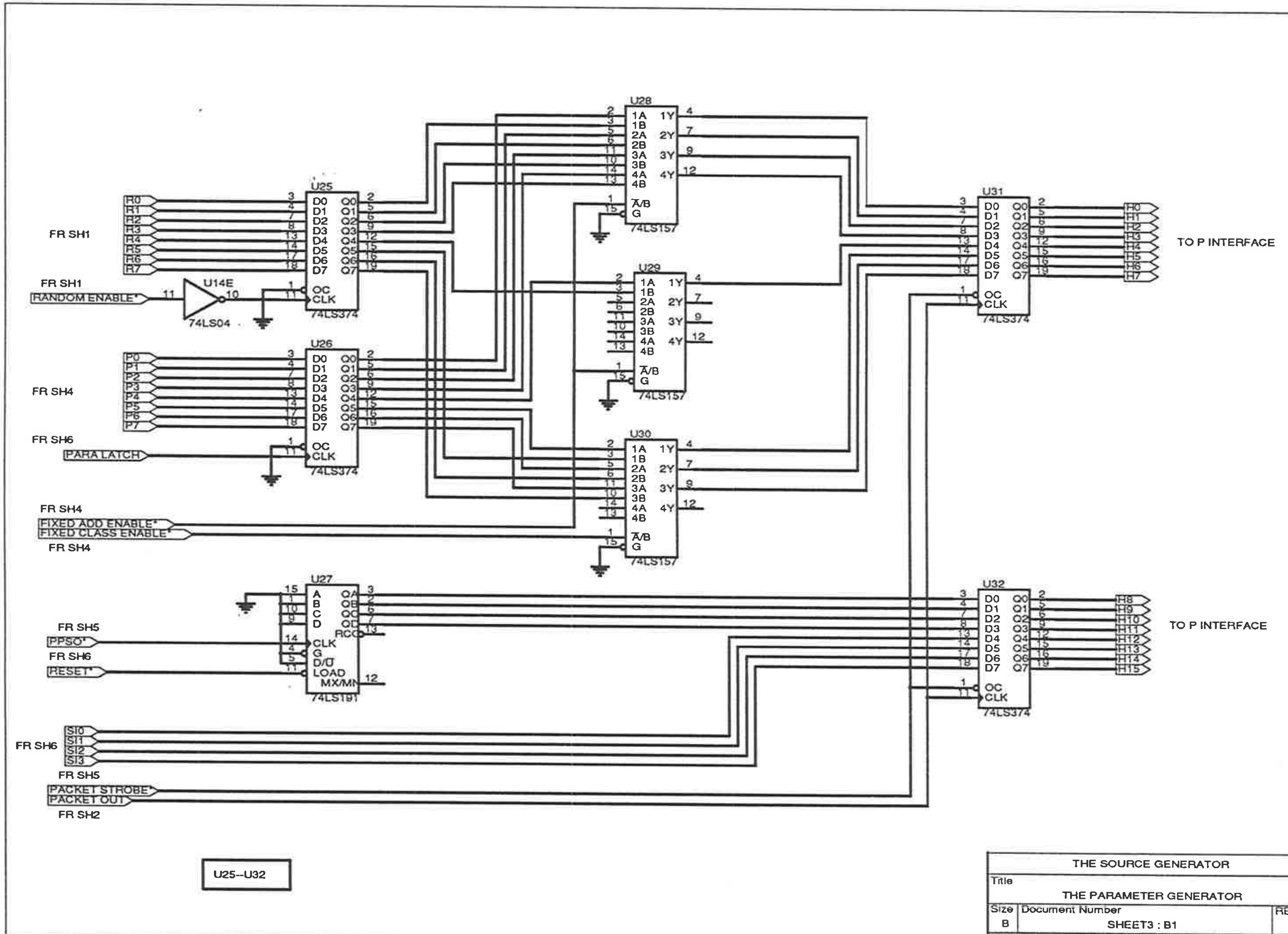
[17] Motorola Inc., "Linear and interface integrated circuits", 1983.

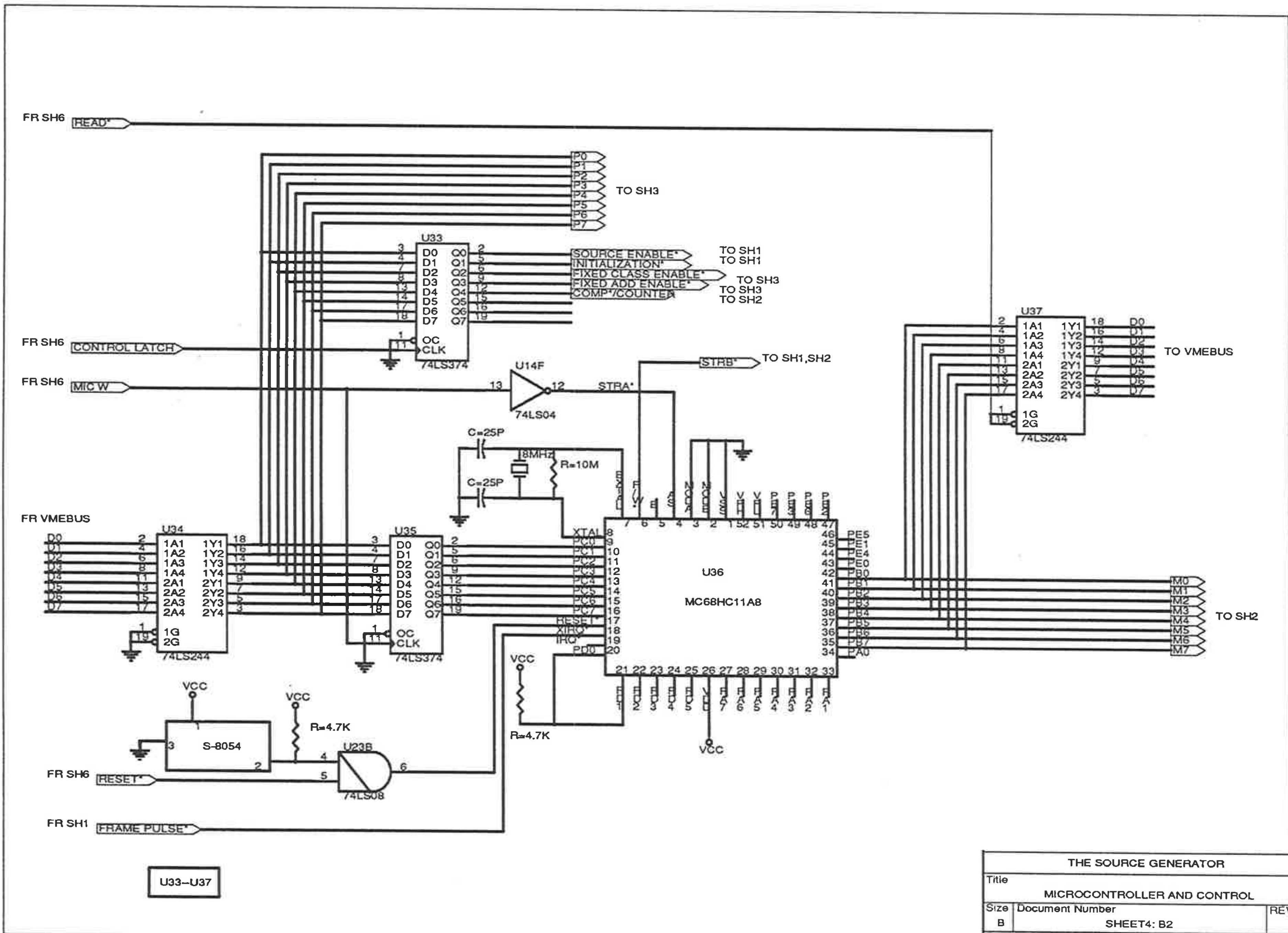
Appendix 3.1


CLOCK
TO SH2
5MHz
U1
crystal oscillator
14 pins
VCC
4.7k
U3
74LS191
U4A
74LS74
U5
74LS191
R1 100K
U2
C3 25pf
X2
32.785KHz
C2
25pf
MM5368
60Hz
30Hz
FR SH4
STRB*
COMPARATOR LATCH
TO SH2
RANDOM ENABLE*
TO SH 2,3,5
FRAME PULSE
TO SH5
FRAME PULSE*
TO SH4,6
U6
74LS157
U13A
74LS86
14th bit
47th bit
U14A
74LS04
U13B
74LS86
31th bit
13th bit
FR SH4
INITIALIZATION*
FR SH6
RESRT*
U7
74LS164
U8
74LS164
U9
74LS164
U10
74LS164
U11
74LS164
U12
74LS164
R0
R1
R2
R3
R4
R5
R6
R7
TO SH3
R8
R9
R10
R11
R12
R13
R14
R15
TO SH2
U1–U14
SH1,3,5 :B1
SH2,4,6 :B2
THE SOURCE GENERATOR
Title
CLOCK AND RANDOM NUMBER GENERATOR
Size
B
Document Number
SHEET1 , B1
REV

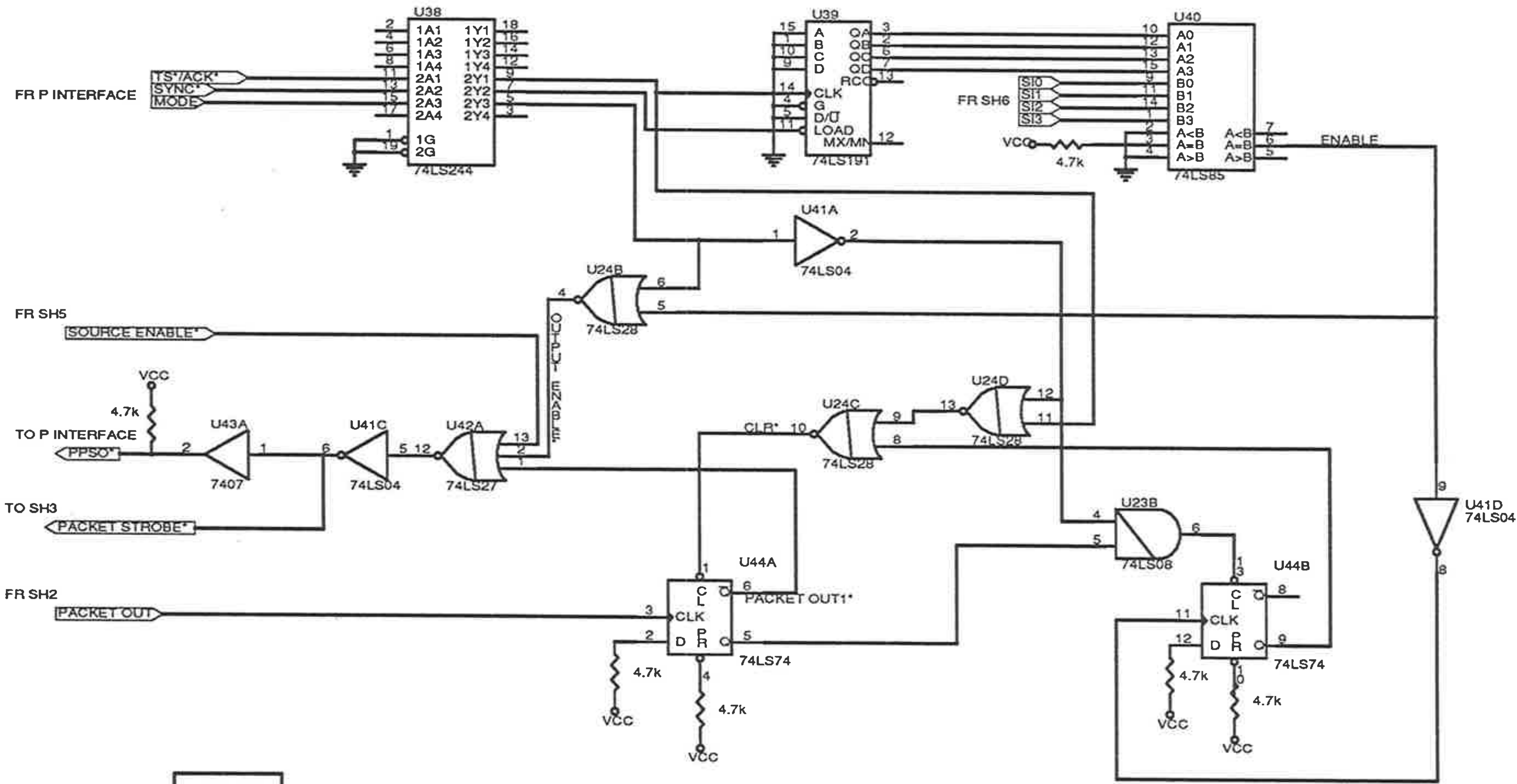


FR SH1
FR SH4
R8
R9
R10
R11
R12
R13
R14
R15
M0
M1
M2
M3
M4
M5
M6
M7
U14B
74LS04
U14C
74LS04
RANDOM ENABLE
U15
74LS374
U16
74LS374
U17
74LS684
U23A
74LS08
U18
74LS191
U19
74LS191
VCC
4.7k
U20
74LS298
PACKET ENABLE*
COMP*/COUNTER
COMPARATOR LATCH
U14D
74LS04
RANDOM ENABLE*
STRB*
U24A
74LS28
U21
74LS191
DIV 8
U22
74LS90
DIV 40
TO SH5
PACKET OUT
U15--U24
THE SOURCE GENERATOR
Title
PACKET GENERATOR
Size
B
Document Number
REV


U28
74LS157
U29
74LS157
U30
74LS157
U25
74LS374
U26
74LS374
U31
74LS374
U32
74LS374
U27
74LS191
U14E
74LS04
FR SH1
R0
R1
R2
R3
R4
R5
R6
R7
FR SH1
RANDOM ENABLE*
FR SH4
P0
P1
P2
P3
P4
P5
P6
P7
FR SH6
PARA LATCH
FR SH4
FIXED ADD ENABLE*
FIXED CLASS ENABLE*
FR SH4
FR SH5
PPSO*
FR SH6
RESET*
FR SH6
SI0
SI1
SI2
SI3
FR SH5
PACKET STROBE*
PACKET OUT
FR SH2
H0
H1
H2
H3
H4
H5
H6
H7
TO P INTERFACE
H8
H9
H10
H11
H12
H13
H14
H15
TO P INTERFACE
U25--U32
THE SOURCE GENERATOR
Title
THE PARAMETER GENERATOR
Size
B
Document Number
SHEET3 : B1
REV

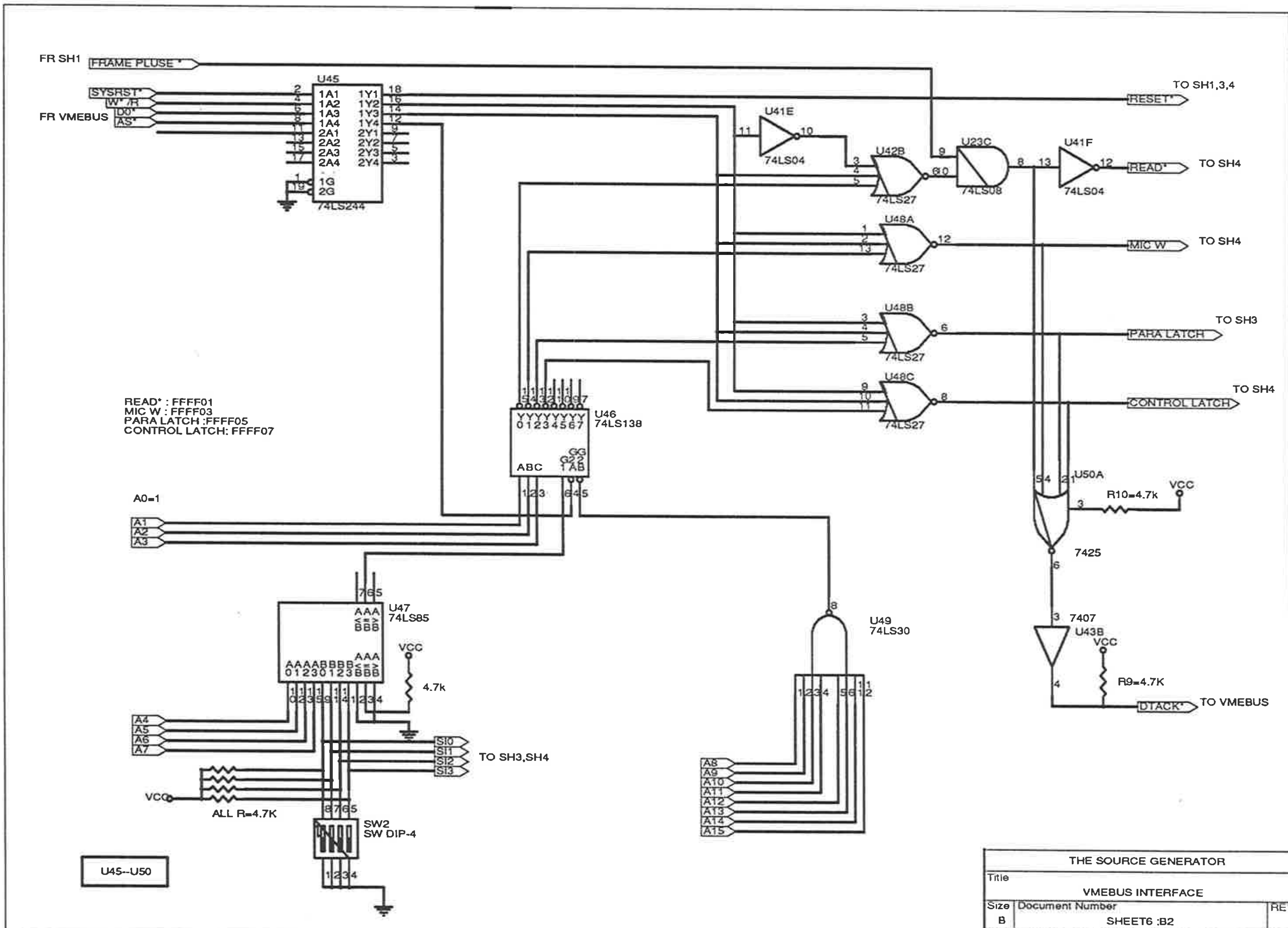


FR SH6 READ*
P0
P1
P2
P3
P4
P5
P6
P7
TO SH3
U33
74LS374
SOURCE ENABLE* TO SH1
INITIALIZATION* TO SH1
FIXED CLASS ENABLE* TO SH3
FIXED ADD ENABLE* TO SH3
COMP*/COUNTER TO SH2
FR SH6 CONTROL LATCH
FR SH6 MIC W
U14F
74LS04
STRA*
STRB* TO SH1,SH2
U37
74LS244
TO VMEBUS
C=25P
8MHz
R=10M
FR VMEBUS
U34
74LS244
U35
74LS374
U36
MC68HC11A8
M0
M1
M2
M3
M4
M5
M6
M7
TO SH2
VCC
S-8054
R=4.7K
U23B
74LS08
FR SH6 RESET*
FR SH1 FRAME PULSE*
R=4.7K
U33--U37
THE SOURCE GENERATOR
Title
MICROCONTROLLER AND CONTROL
Size B
Document Number
SHEET4: B2
REV



THE SOURCE GENERATOR

Title	THE PACKET INTERFACE	
Size B	Document Number SHEET5 :B1	REV



FR SH1
FRAME PLUSE*
SYSRST*
W*/R
D0*
AS*
FR VMEBUS
U45
74LS244
U41E
74LS04
U42B
74LS27
U23C
74LS08
U41F
74LS04
TO SH1,3,4
RESET*
READ*
TO SH4
U48A
74LS27
MIC W
TO SH4
U48B
74LS27
PARA LATCH
TO SH3
U48C
74LS27
CONTROL LATCH
TO SH4
READ* : FFFF01
MIC W : FFFF03
PARA LATCH :FFFF05
CONTROL LATCH: FFFF07
U46
74LS138
A0=1
A1
A2
A3
U50A
R10=4.7k
VCC
7425
U47
74LS85
VCC
4.7k
U49
74LS30
7407
U43B
VCC
R9=4.7K
DTACK*
TO VMEBUS
A4
A5
A6
A7
SI0
SI1
SI2
SI3
TO SH3,SH4
VCC
ALL R=4.7K
SW2
SW DIP-4
A8
A9
A10
A11
A12
A13
A14
A15
U45--U50
THE SOURCE GENERATOR
Title
VMEBUS INTERFACE
Size
B
Document Number
SHEET6 :B2
REV

Appendix 4.1

```
**********************************************************************
*             VIDEO SOURCE SIMULATOR PER MAGLARIS                    *
*                                                                    *
*          For the MC68HC11 microcontroller chip.                    *
*                                                                    *
*         This program is to generator 8 bit rate parameter K(n)     *
*         according to the Maglaris videophon model for encorded     *
*         single source videophone traffic.                          *
*                                                                    *
*         Port B is the output port. Port C is the input port.       *
*                                                                    *
*         References are to the Motorola technical data MC68HC11xx.  *
*                                                                    *
*         Parameter A is single precision 0-255.                     *
*         Parameter B is single precision 0-255.                     *
*         Parameter C is double precision 0-32768.                   *
*                                                                    *
**********************************************************************
*         BY ZHIJIE TAN 1990. CLEANED UP BY DR. K. SARKIES 1990      *
**********************************************************************
*              EQUATES                                               *
**********************************************************************
*              Memory map address                                    *
RAM          EQU     $0000       START OF 256 BYTE RAM
REGS         EQU     $1000       START OF 64 BYTE REGISTER BLOCK
EEPROM       EQU     $B600       START OF 512 BYTE EEPROM
BOOT         EQU     $BF40       START OF SPECIAL BOOT ROM
*              Register assignments                                  *

OUTPUTB      EQU     REGS+$04    OUTPUT PORT B FOR THE RATE PARAMETER
PORTCL       EQU     REGS+$05    INPUT ALTERNATE LATCHED PORT C
PORTC        EQU     REGS+$03    INPUT PORT C
PORTCON      EQU     REGS+$02    PORT C I/O CONTROL REGISTER

RATE         EQU     RAM+$20     FINAL VALUE OF K(n) FOR OUTPUT
ALEAST       EQU     RAM+$22     MODEL PARAMETER A
BLEAST       EQU     RAM+$23     MODEL PARAMETER B
CMOST        EQU     RAM+$24     MODEL PARAMETER C HIGH BYTE
CLEAST       EQU     RAM+$25     MODEL PARAMETER C LOW BYTE

GSUMH        EQU     RAM+$26     SUM OF THE RANDOM NUMBER HIGH BYTE
GSUML        EQU     RAM+$27     SUM OF THE RANDOM NUMBER LOW BYTE
XSH          EQU     RAM+$28     TEMPORARY STORE FOR IX REG HIGH BYTE
XSL          EQU     RAM+$29     TEMPORARY STORE FOR IX REG LOW BYTE
YSH          EQU     RAM+$2A     TEMPORARY STORE FOR IY REG HIGH BYTE
YSL          EQU     RAM+$2B     TEMPORARY STORE FOR IY REG LOW BYTE
RATEH        EQU     RAM+$2C     TEMPORARY STORE FOR THE RATE HIGH BYTE
RATEL        EQU     RAM+$2D     TEMPORARY STORE FOR THE RATE LOW BYTE
GCOUNT       EQU     RAM+$30     GAUSSIAN SUMMATION COUNTER VALUE
REMDERH      EQU     RAM+$32     REMAINDER HIGH BYTE
REMDERL      EQU     RAM+$33     REMAINDER LOW BYTE
RARRAY       EQU     RAM+$40     RANDOM NUMBER ARRAY 55 VALUES
STACK        EQU     RAM+$A0     TOP OF STACK
```

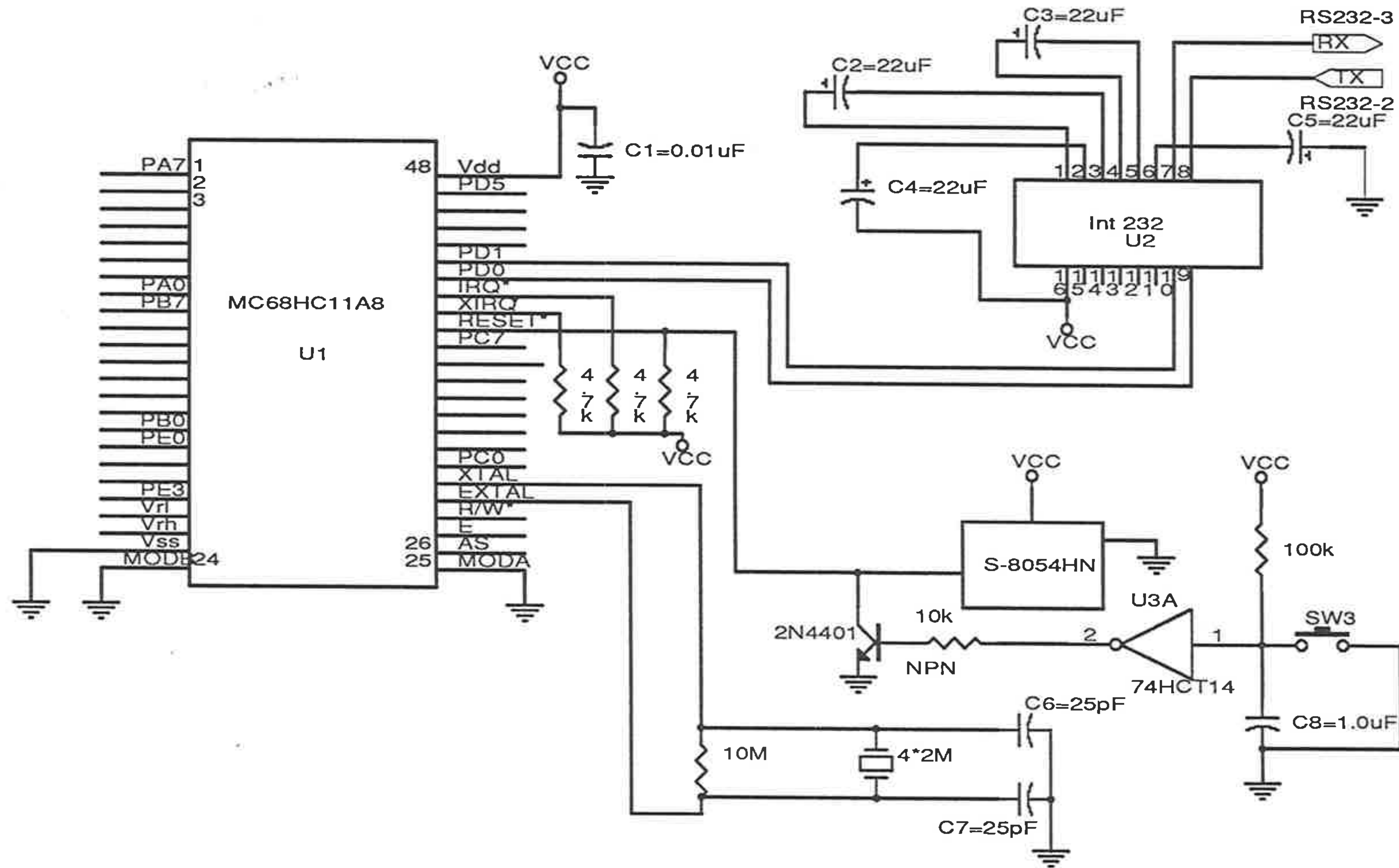
```
********************************************************************
*             PROGRAM START                                        *
********************************************************************
ORG            $B600
               LDAA #$80        SET S BIT,
               TAP              CLEAR X,I BIT TO ENABLE INTERRUPT
********************************************************************
*              INITIALISE THE RANDOM NUMBER                        *
*              ARRAY OF 55 NUMBERS NOT ALL EVEN                    *
*              FOR THE MASTERSON ALGORITHM                         *
********************************************************************
               LDX  #0055       LENGTH OF ARRAY TO FILL
               LDAA #16         INITIAL SEED
LOOP           ADDA #191        SEED VALUE
               STAA RARRAY,X    STORE IN ARRAY AT INDEXED LOCATION
               DEX              NEX ARRAY LOCATION
               BNE  LOOP
********************************************************************
*             SET INTERRUPT VECTOR                                 *
*                                                                  *
********************************************************************
*       IRQ INTERRUPT TO REPLACE DEFAULT A,B,C.                    *
*       THE VECTOR STARTS $B720                                    *
********************************************************************
               LDAA #$7E
               STAA $00EE
               LDAA #$B7
               STAA $00EF
               LDAA #$20
               STAA $00F0
********************************************************************
*      XIRQ INTERRUPT FOR OUTPUT THE RATE PARAMETER K(n).          *
*      THE VECTOR STARTS $B700.                                    *
********************************************************************

               LDAA #$7E
               STAA $00F1
               LDAA #$B7
               STAA $00F2
               LDAA #$00
               STAA $00F3
********************************************************************
*                                                                  *
*              SET INITIAL VALUE                                   *
********************************************************************

               LDAA #176
               STAA ALEAST      SET DEFAULT A= 176
               LDAA #01
               STAA BLEAST      SET DEFAULT B= 1
               LDAA #$05
               STAA CMOST
               LDAA #$6A
               STAA CLEAST      SET DEFAULT C= 1386
               LDAA #$40
```

```
                STAA PORTCON    SET SIMPLE STROBE HANDSHAKE OPERATION
                LDAA #6
                STAA RATE       SET INITIAL K(N)
                LDAA #12
                STAA GCOUNT     SET COUNTER #12
                LDX  #0055
                STX  XSH        SET INITIAL IX STORE #55
                LDY  #0024
                STY  YSH        SET INITIAL IY STORE #24
*******************************************************************
*               MAIN PROGRAM                                      *
*******************************************************************
*               GENERATE B*(X{1}+..+X{12})                        *
*******************************************************************
RESTART          CLR GSUMH
                 CLR GSUML         INITIASE GAUSSIAN SUM =0
                 LDS #STACK        SET STACK POINTER
                 CLI               CLEAR I BIT
                 LDX XSH           LOAD IX FROM IX STORE
                 LDY YSH           LOAD IY FROM IY STORE
MAIN1            LDAB  RARRAY,X    LD Y[55]
                 ADDB  RARRAY,Y    GET (Y[55]+Y[24]| )MODULO 256
                 STAB  RARRAY,X    SAVE TO Y[55]
                 LDAA BLEAST
                 MUL               b*X{i}
                 ADDD GSUMH        SUM=SUM+b*X{i}
                 STD  GSUMH        STORE SUM TO $32 MOST,$33 LEAST
                 DEX               IX -1
                 BNE  MAIN2
                 LDX  #0055        IF ZERO RESET IX TO TOP OF ARRAY
MAIN2            DEY               IY -1
                 BNE  MAIN3
                 LDY  #0055        IF ZERO RESET IY TO TOP OF ARRAY
MAIN3            DEC  GCOUNT       COUNTER -1
                 BNE  MAIN1        REPEAT 12 TIMES TO GET RANDOM NUMBER
                 STX  XSH          SAVE IX TO IX STORE
                 STY  YSH          SAVE IY TO IY STORE
                 LDAA #12
                 STAA GCOUNT       LD COUNTER #12

                 LDAA ALEAST       LD A
                 LDAB RATE         LD K(n)
                 MUL               A*K(n)
                 ADDD GSUMH        A*K(n)+ B*(X{1}+..+X{12})
                 SUBD CMOST        A*K(n)+ B*(X{1}+..+X{12}) -C
                 BPL  BUFF1
                 CLRB              IF <0  THEN SET K(n)=0
                 STAB RATE
                 JMP RESTART

BUFF1            LDX #0200         LOAD  IX 200
                 IDIV              [A*K(n)+B*(X{1}+..+X{12})-C]/200
                 STD REMDERH       SAVE RENAINDER
                 STX RATEH         SAVE RESULT TO THE TEMPORARY RATE REG
```

```
                LDAA RATEH
                TSTA              CHECK UPPER BYTE
                BEQ  BUFF2        SHOULD BE 0
                LDAB #$FF         IF >0 OVERFLOW, OUTPUT 255
                BRA  BUFF4

BUFF2           LDAA REMDERL
                SUBA #100         REMAINDER - 100
                BMI  BUFF3        IF REMAINDER <100, PUT RESULT TO RATE
                INC  RATEL        IF REMAINDER >= 100, RATE +1
BUFF3           LDAB RATEL
BUFF4           STAB RATE         SAVE AS K(n+1)
                WAI               WAIT FOR INTERRUPT TO OUTPUT NEXT K(n)
                JMP  RESTART      REPEAT
********************************************************************
*             XIRQ INTERRUPT ROUTINE : OUTPUT K(n)                 *
********************************************************************
                ORG $B700
                LDAA RATE
                STAA OUTPUTB
                RTI
********************************************************************
*            IRQ INTERRUPT ROUTINE: MODIFY PARAMETER A,B,C         *
*            THIS ROUTINE WORKS THROUGH SERVAL STATES TO READ      *
*            DATA TO REPLACE DEFAULT A,B,C.                        *
********************************************************************
                ORG $B720
                LDAA PORTCON       CLEARS THE STROBE FLAG
                LDAA PORTCL
                LDAA PORTC         LOAD FROM THE PORT C
                STAA ALEAST        REPLACE DEFAULT A
                LDAA #$40
                STAA $F0           SET NEXT IRQ INTERRUPT VECTOR
                RTI

                ORG $B740
                LDAA PORTCON       CLEARS THE STROBE FLAG
                LDAA PORTCL
                LDAA PORTC         LOAD FROM THE PORT C
                STAA BLEAST        REPLACE DEFAULT B
                LDAA #$60
                STAA $F0           SET NEXT IRQ INTERRUPT VECTOR
                RTI

                ORG $B760
                LDAA PORTCON      CLEARS THE STROBE FLAG
                LDAA PORTCL
                LDAA PORTC        LOAD FROM THR PORT C
                STAA CLEAST       REPLACE C LOW BYTE DEFAULT
                LDAA #$80
                STAA $F0          SET NEXT INTERRUPT VECTOR
                RTI
```

```
        ORG $B780
        LDAA PORTCON      CLEARS THE STROBE FLAG
        LDAA PORTCL
        LDAA PORTC        LOAD FROM THR PORT C
        STAA CMOST        REPLACE C HIGH BYTE DEFAULT
        LDAA #$20
        STAA $F0          COMPLETE WRITE,
        RTI               SET ORIGINAL INTERRUPT VECTOR

*******************************************************************
*           END                                                   *
*******************************************************************
```

Appendix 5.1



Title	DOWNLOAD DIAGRAM	
Size A	Document Number	REV
Date: December 4, 1988	Sheet of	

Appendix 6.1

rdrate

```
************************************************************************
*       This program is to read the packet rate parameter from         *
*       the output B of the microcontroller of the source board.       *
*                                                                      *
*       20,000 rate parameter values will be read, the values will     *
*       be saved to $6000--$AE1F and be written to the monitor.        *
*                                                                      *
*       The values are hex numbers from $00 to $ff.                    *
*                                                                      *
*       Read address is the VMEbus address $ffff01.                    *
************************************************************************
*              EQUATES                                                 *
************************************************************************
 org         $5000

 sio1base    equ.l   $f7c001
 sio2base    equ.l   $f7c011
 sstat       equ     2
 sdata       equ     6
 *status register bit difinitions*
 rxrbit      equ     0
 txrbit      equ     2
 monitor     equ.l   $f00014
 micportb    equ.l   $ffff01
************************************************************************
*              PROGRAM START                                           *
************************************************************************
 main1:         movea    #$2ffe,sp          *set up stack point*
                movea    #con1,a0
                jsr      write              *write start information *
                clr.l    d7                 *clear counter d7*
                clr.l    d6
                movea.l  #$6000,a3
                jsr      micread            *read the rate parameter *
                movea    #con2,a0           *get string*
                jsr      write              *display information*
                jsr      monitor            *to monitor*
************************************************************************
*           SUBROUTINE: READ RATE PARAMETER FROM THE SOURCE BOARD     *
************************************************************************
micread:        move.b   micportb,d0        *read 8 bit rate parameter*
                move.b   d0,d3
                move.b   d0,(a3)+           *save number to $6000+ *
                lsr.l    #4,d0
                jsr      wrnum              *wrt high 4 bits to monitor*
                move.b   d3,d0
                jsr      wrnum              *wrt low 4 bits to monitor*
                move.b   #$20,d0
                jsr      wri                * write a space*
                add.l    #1,d6
                cmp.l    #20,d6
                beq      rten
mk:             add.l    #1,d7
```

```
                cmp.l     #20000,d7           *until 20000 values are read*
                beq       micrdend
 micrdloop:     move.b    micportb,d0         * read next and save to d2*
                cmp.b     d0,d3               * compare with last one*
                beq       micrdloop           *wait until different data*
                bra       micread             *read again*

 micrdend:      clr.l     d7
                rts                           * end read*
*******************************************************************
*          SUBROUTINE: WRITE HEX RATE PARAMETER TO MONITOR        *
*******************************************************************
 wrnum:         and.l     #$0f,d0             *mask out high bits*
                cmp.b     #$0a,d0
                blt       hexch
                add.b     #7,d0
 hexch:         add.b     #$30,d0
                jsr       wri                 *write one number*
                rts
*******************************************************************
*          SUBROUTINE: WRITE TO MONITOR                           *
*******************************************************************
 write:         move.l    a1,-(sp)            *push a1*
                movea     a0,a1
 wsloop:        move.b    (a1)+,d0            *get string *
                beq       wend                *IF 0 then end*
                jsr       wri                 *write character to monitor*
                bra       wsloop              * and again*
 wend:          movea.l   (sp)+,a1            *pull a1,then return *
                rts
*******************************************************************
*          write character to monitor                             *
*******************************************************************
 wri:           lea       siolbase,a0         *load base port address*
 wrloop:        move.b    sstat(a0),d1        *move status to d1*
                btst      #txrbit,d1          *test status*
                beq       wrloop              *wait until ready*
                move.b    d0,sdata(a0)        *write character*
                rts

 rten:          move.b    #$0a,d0
                clr.l     d6
                jsr       wri
                move.b    #$0d,d0
                jsr       wri
                bra       mk
*******************************************************************
*               INFORMATION AND INSTRUCTIONS                      *
*******************************************************************
 con1:          dc.b      $0a,$0d,'This program is to read the '
                dc.b      'rate parameter from '
                dc.b      'the source board, 20,000 numbers '
                dc.b      'are:',$0a,$0d,$0
 con2:          dc.b      $0a,$0d,'Read is completed.',$0a,$0d,$0
 end
```

** wrabc **

```
*********************************************************************
*   This program is to write A,B,C into the microprocessor to       *
*   modify the model parameter.                                     *
*                                                                   *
*   Parameter A, B are single precision 0-255, C is double          *
*   precision 0-32768.                                              *
*   Write the parameters to the VMEbus address $ffff03.             *
*                                                                   *
*   Must write 0 first for the test, then A, B, C.                  *
*                                                                   *
*   References are to the Motorola 16-bit microprocessor user's     *
*   manual and IV-1602 VMEbus single board computer user's manual.  *
*********************************************************************
*                    EQUATES                                        *
*********************************************************************
org   $004000

sio1base  equ.l $f7c001
sio2base  equ.l $f7c011
sstat     equ   2
sdata     equ   6
*status register bit difinitions*
rxrbit    equ   0
txrbit    equ   2
monitor   equ.l $f00014
micportc  equ.l $ffff03
*********************************************************************
*             MAIN PROGRAM                                          *
*********************************************************************

 main1:    movea    #$2ffe,sp         *set up stack pointer*
           movea    #con1,a0
           jsr      write             *display instructions: input A*
           jsr      read              *get 0 from user*
           jsr      micwrite          *write A to microcontroller*
           movea    #con2,a0
           jsr      write             *display information 'input B' *
           jsr      read              *get B from user   *
           jsr      micwrite          *write A to the microprocessor*
           movea    #con3,a0
           jsr      write             *display 'input C' *
           jsr      read              *get C*
           jsr      micwrite          *write C low byte to the mic *
           movea    #con4,a0
           jsr      write             *display 'write C low*
                                      *byte is completed.*
           jsr      micwrtc           *write C high byte to the mic*
           movea    #con5,a0
           jsr      write             *display information*
           jsr      monitor           *to monitor*
*********************************************************************
*            SUBROUTINE: WRITE TO MICROCONTROLLER                   *
*********************************************************************
 micwrite: move.w   d1,d3             *move d1 to d3*
```

```
            move.w    d1,d4               *move d1 to d4*
            and.l     #$00ff,d3           *make lowest byte*
            move.b    d3,micportc         *write to microcontroller*
            movea     #con6,a0            * write information to monitor*
            jsr       write
            rts                           *return*

 micwrtc:   and.l     #$ff00,d4           *make hige byte*
            lsr.w     #8,d4               *logical right shift*
            move.b    d4,micportc         *write to microcontroller*
            movea     #con7,a0            *write to monitor*
            jsr       write
            rts                           *return*
*********************************************************************
*           SUBROUTINE: WRITE TO MONITOR                            *
*********************************************************************
 write:     move.l    a1,-(sp)            *push a1*
            movea     a0,a1
 wsloop:    move.b    (a1)+,d0            *get string*
            beq       wend                *if 0 then end*
            jsr       wri                 *write character to monitor*
            bra       wsloop              * and again*
 wend:      movea.l   (sp)+,a1            *pull a1,then return *
            rts
* write character to monitor*

 wri:       lea       sio1base,a0         *load base port address*
 wrloop:    move.b    sstat(a0),d1        *move status to d1*
            btst      #txrbit,d1          *test status*
            beq       wrloop              *wait until ready*
            move.b    d0,sdata(a0)        *write character*
            rts
*********************************************************************
*           READ FROM USER AND CONVERT DECIMAL NUMBER TO HEX NUMBER *
*********************************************************************
 read:      clr.w     $004500             *clear*
            clr.w     $004502
            clr.w     $004504
            clr.w     $004506
            clr.w     $004508
            lea       sio1base,a0         *load base port address*
            move.l    a1,-(sp)            *push a1*
            lea       $004500,a1          *load address*
 rdloop:    move.b    sstat(a0),d0        *load status to d0*
            btst      #rxrbit,d0          *test status*
            beq       rdloop              *wait until ready*
            move.b    sdata(a0),d0        *get character then put to d0*
            cmp.b     #$0d,d0             *test whether 'carriage return'*
            beq       rdend               *yes, end read*
            jsr       wri                 *echo to screen*
            andi.w    #$000f,d0           *convert ASCII decimal to decimal*
            move.w    4(a1),6(a1)         *move,highest number put in 6(a1)*
            move.w    2(a1),4(a1)         *lowest number put in (a1)*
            move.w     (a1),2(a1)
            move.w    d0,(a1)
```

```
          bra       rdloop              *loop *

rdend:    jsr       convert             *to convert subroutine*
          movea.l   (sp)+,a1            *pull a1*
          rts                           *return,end read*
*******************************************************************
*            CONVERT DECIMAL TO HEX                                *
*******************************************************************
convert: move.w     6(a1),d0            *load highest number to d0*
          lsl.w     6(a1)               *logical shift left*
          lsl.w     #3,d0               *logical shift left 3 bits*
          add.w     6(a1),d0            *add and put to d0*
          add.w     4(a1),d0            *add*
          move.w    d0,d1               *move d0 to d1*
          lsl.w     #1,d1               *logical shift left d1*
          lsl.w     #3,d0               *logical shift left d0*
          add       d1,d0               *add*
          add       2(a1),d0            *add*
          move.w    d0,d1               *move d0 to d1*
          lsl.w     #1,d1               *logical shift left d1*
          lsl.w     #3,d0               *logical shift left 3 bits*
          add       d1,d0               *add*
          add       (a1),d0             *add*
          move.w    d0,d1               * put hex number to d1*
          rts                           *return*
*******************************************************************
*            INFORMATION                                           *
*******************************************************************
con1:     dc.b      $0a,$0d,'This program is to write '
          dc.b      'A,B,C to the microcontroller of the source '
          dc.b      'board, please input a decimal number:'
          dc.b      $0a,$0d,'Input A (0 to 255):',$0a,$0d,$0
con2:     dc.b      $0a,$0d,'Input B (0 to 255):',$0a,$0d,0
con3:     dc.b      $0a,$0d,'Input C (0 to 32768):',$0a,$0d,0
con4:     dc.b      ' --write C low byte is completed.',$0a,$0d,0
con5:     dc.b      $0a,$0d,'Write A,B, and C to the '
          dc.b      'microcontroller is completed.',$0a,$0d,0
con6:     dc.b      $0a,$0d,'Write to the microcontroller '
          dc.b      'is completed '
con7:     dc.b      $0a,$0d,'Write C high byte is completed.',0

end
```

wrcontrol

```
*********************************************************************
*     This program is to write the control number to the control    *
*     register of the source board.                                 *
*                                                                   *
*     The control number is a single precision 0-255. The program   *
*     will convert the decimal number to 8-bit binary number, then  *
*     write to the control register. The control numbers refer      *
*     to Table 6.2.                                                 *
*                                                                   *
*     Write the control number to the VMEbus address $ffff07.       *
*                                                                   *
*     Address $4500--$4520 is reserved.                             *
*********************************************************************
*                    EQUATES                                        *
*********************************************************************
 org          $004300

 sio1base     equ.l    $f7c001
 sio2base     equ.l    $f7c011
 sstat        equ      2
 sdata        equ      6
 *status register bit difinitions*
 rxrbit       equ      0
 txrbit       equ      2
 monitor      equ.l    $f00014
 contreg      equ.l    $ffff07
*********************************************************************
*            PROGRAM START                                          *
*********************************************************************
 main1:       movea    #$2ffe,sp          *set up stack point*
              movea    #con1,a0
              jsr      write              *display information *
              jsr      read               *get control bits from user*
              jsr      conwrite           *write control bits *
              movea    #con2,a0           *get string*
              jsr      write              *display information*
              jsr      monitor            *to monitor*
*********************************************************************
*          SUBROUTINE: WRITE TO THE CONTROL REGISTER OF THE BOARD   *
*********************************************************************
 conwrite:    move.w   d1,d3
              and.l    #$00ff,d3          *mask out high bits*
              move.b   d3,contreg         *wrt to the control register*
              rts                         *return*
*********************************************************************
*          SUBROUTINE: WRITE TO MONITOR                             *
*********************************************************************
 write:       move.l   a1,-(sp)           *push a1*
              movea    a0,a1
 wsloop:      move.b   (a1)+,d0           *get string*
              beq      wend               *IF 0 then end*
              jsr      wri                *write character to monitor*
              bra      wsloop             * and again*
```

```
wend:        movea.l  (sp)+,a1            *pull a1,then return *
             rts
***********************************
* write character to monitor      *
***********************************
wri:         lea      sio1base,a0         *load base port address*
wrloop:      move.b   sstat(a0),d1        *move status to d1*
             btst     #txrbit,d1          *test status*
             beq      wrloop              *wait until ready*
             move.b   d0,sdata(a0)        *write character*
             rts
***********************************************************************
*          READ FROM USER AND CONVERT DECIMAL NUMBER TO HEX NUMBER    *
***********************************************************************
read:        clr.w     $004500            *clear*
             clr.w     $004502
             clr.w     $004504
             clr.w     $004506
             clr.w     $004508
             lea       sio1base,a0        *load base port address*
             move.l    a1,-(sp)           *push a1*
             lea       $004500,a1         *load address*
rdloop:      move.b    sstat(a0),d0       *load status to d0*
             btst      #rxrbit,d0         *test status*
             beq       rdloop             *wait until ready*
             move.b    sdata(a0),d0       *get character then put to d0*
             cmp.b     #$0d,d0            *test whether 'carriage return'*
             beq       rdend              *yes, end read*
             jsr       wri                *echo to screen*
             andi.w    #$000f,d0          *convert ASCII decimal to decimal*
             move.w    4(a1),6(a1)        *move,highest number put in 6(a1)*
             move.w    2(a1),4(a1)        *lowest number put in (a1)*
             move.w    (a1),2(a1)
             move.w    d0,(a1)
             bra       rdloop             *loop *

rdend:       jsr       convert            *to convert subroutine*
             movea.l   (sp)+,a1           *pull a1*
             rts                          *return,end read*
**********************************************************************
*          CONVERT DECIMAL TO HEX                                    *
**********************************************************************
convert:     move.w    6(a1),d0           *load highest number to d0*
             lsl.w     6(a1)              *logical shift left*
             lsl.w     #3,d0              *logical shift left 3 bits*
             add.w     6(a1),d0           *add and put to d0*
             add.w     4(a1),d0           *add*
             move.w    d0,d1              *move d0 to d1*
             lsl.w     #1,d1              *logical shift left d1*
             lsl.w     #3,d0              *logical shift left d0*
             add       d1,d0              *add*
             add       2(a1),d0           *add*
             move.w    d0,d1              *move d0 to d1*
             lsl.w     #1,d1              *logical shift left d1*
             lsl.w     #3,d0              *logical shift left 3 bits*
```

```
            add      d1,d0               *add*
            add      (a1),d0             *add*
            move.w   d0,d1               * put hex number to d1*
            rts                          *return*
*******************************************************************
*           INFORMATION                                           *
*******************************************************************
con1:    dc.b     $0a,$0d,'This program is to write '
         dc.b     'the control number to the control '
         dc.b     'register of the source board, please '
         dc.b     'input decimal number.',$0a,$0d
         dc.b     'Control number(0--255):'
         dc.b     $0a,$0d,$0
con2:    dc.b     $0a,$0d,'Write to the control register '
         dc.b     'is completed.',$0a,$0d,0
end
```

wrparameter

```
*******************************************************************
*     This program is to write the packet header parameter to     *
*     the parameter register of the source board.                 *
*                                                                 *
*     The parameter number is single procision 0-255. The program *
*     will convert decimal parameter number to 8-bit binary number,*
*     then write to the parameter register. The lower 5 bits are  *
*     the packet header destination address, and upper 3 bits are *
*     packet class parameter.                                     *
*                                                                 *
*     Write the parameter to the VMEbus address $ffff05           *
*                                                                 *
*     Reference for the parameter is to Chapter3.4.3 parameter    *
*     generator.                                                  *
*                                                                 *
*     Address $004500--$004520 is reserved.                       *
*******************************************************************
*              EQUATES                                            *
*******************************************************************
 org          $4600

 sio1base     equ.l   $f7c001
 sio2base     equ.l   $f7c011
 sstat        equ     2
 sdata        equ     6
 *status register bit difinitions*
 rxrbit       equ     0
 txrbit       equ     2
 monitor      equ.l   $f00014
 parareg      equ.l   $ffff05
*******************************************************************
*              PROGRAM START                                      *
*******************************************************************
 main1:       movea   #$2ffe,sp          *set up stack point*
              movea   #con1,a0
              jsr     write              *display information *
              jsr     read               *get control bits fr user*
              jsr     parawrite          *write parameter byte *
              movea   #con2,a0           *get string*
              jsr     write              *display information*
              jsr     monitor            *to monitor*
*******************************************************************
*           SUBROUTINE: WRITE TO THE CONTROL REGISTER OF THE BOARD *
*******************************************************************
 parawrite:   and.l   #$00ff,d1          *mask out high bits*
              move.b  d1,parareg         *write to para register*
              rts                        *return*
*******************************************************************
*           SUBROUTINE: WRITE TO MONITOR                          *
*******************************************************************
 write:       move.l  a1,-(sp)           *push a1*
              movea   a0,a1
 wsloop:      move.b  (a1)+,d0           *get string*
              beq     wend               *if 0 then end*
```

```
          jsr       wri                 *write char to monitor*
          bra       wsloop              * and again*
wend:     movea.l   (sp)+,a1            *pull a1,then return *
          rts
***********************************
* write character to monitor      *
***********************************
wri:      lea       sio1base,a0         *load base port address*
wrloop:   move.b    sstat(a0),d1        *move status to d1*
          btst      #txrbit,d1          *test status*
          beq       wrloop              *wait until ready*
          move.b    d0,sdata(a0)        *write character*
          rts
*****************************************************************
*        READ FROM USER AND CONVERT DECIMAL NUMBER TO HEX NUMBER   *
*****************************************************************
read:     clr.w     $004500             *clear*
          clr.w     $004502
          clr.w     $004504
          clr.w     $004506
          clr.w     $004508
          lea       sio1base,a0         *load base port address*
          move.l    a1,-(sp)            *push a1*
          lea       $004500,a1          *load address*
rdloop:   move.b    sstat(a0),d0        *load status to d0*
          btst      #rxrbit,d0          *test status*
          beq       rdloop              *wait until ready*
          move.b    sdata(a0),d0        *get character then put to d0*
          cmp.b     #$0d,d0             *test whether 'carriage return'*
          beq       rdend               *yes, end read*
          jsr       wri                 *echo to screen*
          andi.w    #$000f,d0           *convert ASCII decimal to decimal*
          move.w    4(a1),6(a1)         *move,highest number put in 6(a1)*
          move.w    2(a1),4(a1)         *lowest number put in (a1)*
          move.w    (a1),2(a1)
          move.w    d0,(a1)
          bra       rdloop              *loop *

rdend:    jsr       convert             *to convert subroutine*
          movea.l   (sp)+,a1            *pull a1*
          rts                           *return,end read*
*****************************************************************
*          CONVERT DECIMAL TO HEX                                *
*****************************************************************
convert:  move.w    6(a1),d0            *load highest number to d0*
          lsl.w     6(a1)               *logical shift left*
          lsl.w     #3,d0               *logical shift left 3 bits*
          add.w     6(a1),d0            *add and put to d0*
          add.w     4(a1),d0            *add*
          move.w    d0,d1               *move d0 to d1*
          lsl.w     #1,d1               *logical shift left d1*
          lsl.w     #3,d0               *logical shift left d0*
          add       d1,d0               *add*
          add       2(a1),d0            *add*
          move.w    d0,d1               *move d0 to d1*
```

```
               lsl.w      #1,d1                *logical shift left d1*
               lsl.w      #3,d0                *logical shift left 3 bits*
               add        d1,d0                *add*
               add        (a1),d0              *add*
               move.w     d0,d1                * put hex number to d1*
               rts                             *return*
********************************************************************
*            INFORMATION                                           *
********************************************************************
 con1:     dc.b      $0a,$0d,'This program is to write '
           dc.b      'the packet header parameter to the '
           dc.b      'parameter register of the source board,'
           dc.b      'please input a decimal number:',$0a,$0d
           dc.b      'Packet header parameter(0--255):',$0a,$0d,$0
 con2:     dc.b      $0a,$0d,'Write to the parameter register '
           dc.b      'is completed.',$0a,$0d,0
 end
```

Appendix 6.2

```
(*  This program will get S-record data from the file           *)
(*  'fread', and remove all unwanted elements. The remain       *)
(*  data will be hex number, and be placed to the output        *)
(*  file 'hexdata' and output to the monitor.                   *)

(*  By Linh Nguyen.  Cleaned up by Zhijie Tan 1991              *)

program getdatafromfread(input, output, hexdata, fread);
var     fread,hexdata:text;
        c:array[1..100] of char;
        ch:char;
        i,int:integer;

begin

  writeln('This program is to convert S record data contained in');
  writeln('file fread to hex data, these data is displayed while');
  writeln(' saved to file hexdata.');
  writeln('The hex rate parameter values are:');
  reset(hexdata); rewrite(hexdata);
  reset(fread);

  while not eof(fread) do
    begin
      i:=1;
      while not eoln(fread) do
        begin
          read(fread,ch);          (* get chars fr 'fread'*)
          c[i]:=ch;
          i:=i+1;
        end;
      int:=i-3;
      readln(fread);                (* next line*)

      if ((c[1]='S') and (c[2]='1')) then
        begin
          for i:=9 to int do        (*cleans unwanted elements*)
            write(hexdata,c[i]);    (* write to 'hexdata'*)
            writeln(hexdata);

          for i:=9 to int do
            write(c[i]);            (* write to the monitor*)
            writeln;
         end;
       end;
end.
```

```
(*  This program will get hex numbers from 'hexdata'to        *)
(*  convert to decimal numbers, and then put to the           *)
(*  output file 'data', while the data is output to           *)
(*  the monitor.                                              *)

(*  By Linh Nguyen. Cleaned by Zhijie Tan 1991.               *)

program hextodec(input, output, hexdata, data);
const   hexval=16;
var     hexdata,data:text;
hexarray:array[1..80] of char;
hexno:array[1..2] of char;
no,yes,count1,count2:integer;
hexchar:char;
hexerror:boolean;
decno:integer;

function convert(hexch:char):integer; (*convert hex char to dec*)
 begin
    case hexch of
       '0','1','2','3','4','5','6','7','8','9':
        convert:=ord(hexch)-ord('0');
       'A','B','C','D','E','F':
       convert:=ord(hexch)-ord('A')+10;
       'a','b','c','d','e','f':
       convert:=ord(hexch)-ord('a')+10;
     end;
 end;
(*convert hex number to decimal number*)

function power(x,a:integer):integer;
  var   count1:integer;
        product:integer;
  begin
    product:=1;
    for count1:=1 to a do
        product:=product*x;
        power:=product;
  end;

 (* Main program   *)
 begin
     write('This program converts hexadecimal numbers');
     writeln('to decimal numbers.');
     write('The input file is hexdata and the resultant');
     writeln('output file is data.');
     writeln('Please wait..........');
     reset(hexdata);
     reset(data); rewrite(data);

     (*    Check whether hex char error    *)

     hexerror:=false;
     while (not eof(hexdata)) and (not hexerror) do
       begin
```

```
          no:=0;
          while (not eoln(hexdata)) and (not hexerror) do
            begin
              no:=no+1;
              read(hexdata,hexchar);
              if not (hexchar in ['a','b','c','d','e',
              'f','A','B','C','D','E','F','0','1','2',
              '3','4','5','6','7','8','9']) then
              hexerror:=true;
              hexarray[no]:=hexchar;
            end;
          readln(hexdata);

 (*If not error, convert hex number to decimal number. *)

          if not hexerror then
            for count2:=1 to no do
            begin
              if odd(count2) then yes:=1 else yes:=2;
              hexno[yes]:=hexarray[count2];
              if yes=2 then
                 begin
                   decno:=0;
                   for count1:=yes downto 1 do
                   decno:=decno+power(hexval,yes-count1)
                          *convert(hexno[count1]);
                   write(decno:4);
                   writeln(data,decno);
                  end;
            end

(*   If error, terminal the program.      *)

      else writeln('It is not a correct hexadecimal form...');
      writeln;
     end;
end.
```

```
(*  This program is to get the MEAN and STANDARD DEVIATION    *)
(*  of the bit rate, and the bit rate histogram as well as    *)
(*  the autocovariance function. The decimal rate parameter   *)
(*  will be obtained from the file 'data'.                    *)

(* Written by Mr Linh Nguyen, modified by Z. Tan.            *)

program correlation(input, output, data);
 const   n=100000; {arbitrary large number}
         h=20;  h1=40;
         interval=1;
 var     data:text;
         K:array[1..n] of integer;
         C:array[0..n] of real;
         scale:array[1..h1] of integer;
         N,i,j:integer;
         mean,sd:real;
         hist:array[1..h] of integer;

(*  This function is to obtain the MEAN of the bit rate  *)
    function average:real;
      var   sum, aver:real;
            count1:integer;

      begin
       sum:=0;
       for count1:=1 to N do
         sum:=sum+K[count1];
       aver:=sum/N;     (* The mean of the rate parameter*)
       average:=aver/12;    (* The mean of the bit rate*)
      end;

(* The function is to obtain the STANDARD DEVIATION
   of the bit rate  *)
   function  standdev:real;
      var    stddev,squaresum:real;
             count1:integer;
      begin
        squaresum:=0;
        for count1:=1 to N do
          squaresum:=squaresum+sqr(K[count1]-12*mean);
        stddev:= sqrt(squaresum/(N-1));
        standdev:= stddev/12;
          (*The standard deviation of the bit rate*)
      end;

 (* This procedure is to obtain the bit rate histogram *)
    procedure histogram;
       var    x, m:integer;
              scale:real;
       begin
         for x:=1 to h do
           begin
             scale:=(x-1)/12;
             write(scale:1:2);(*scale bit rate in bits/pixel*)
```

```
          write('|');
          for m:=1 to hist[x]do
            write(' ');
          write('*');
          writeln;
        end;
    end;

(* This procedure is to obtain autocovariance function     *)
(* of the bit rate                                         *)
   procedure autocovar;
     var     k:integer;
             count1:integer;
             productsum:real;
             {c:real;}
     begin
       writeln('The CORRELATION of the data.....');
       writeln;
       for k:=0 to h1 do
         begin
           productsum:=0;
           for count1:=1 to N-k do
             productsum:=productsum+(K[count1])*(K[count1+k]);
           C[k]:=(productsum/(N-k))/sqr(12)-sqr(mean);
             (* autocov of the bit rate*)
           write('C[',k:1,']= ', C[k]:1:3); writeln
         end;
     writeln('Scale normalised to the standard deviation...');
     writeln;
     write(  '        0.0                                       ');
     write(C[0]:1:3);
     writeln;
     writeln('+--------+---------------------------------------+');
     for k:=0 to h1 do scale[k]:=trunc(40*(C[k]/C[0]));
     for k:=0 to h1 do
       begin
         if scale[k]<0 then
           begin
             write('*':(10+scale[k]));
             write('|':-scale[k]);
           end;
         if scale[k]>0 then
           begin
             write('|':10); write('*':scale[k]);
           end;
         if scale[k]=0 then
           write('*':10);
           writeln;
       end;
   end;

     (*  Main Program   *)

   begin
     reset(data);
```

```
   N:=0;
   for i:=1 to h do hist[i]:=0;
   while not eof(data) do
{storing contents of result onto the array 'lamda'}
       begin
         N:=N+1;
         readln(data,j);
         K[N]:=j;
         hist[trunc(K[N]/interval)+1]:=
                hist[trunc(K[N]/interval)+1]+1;
       end;

   writeln('Total number, N, of data is: ',N); writeln;

   mean:=average;
   writeln('The MEAN of the data is: ',mean:5:2,'.');
   writeln;

   sd:=standdev;
   writeln('The STANDARD DEVIATION is: ',sd:5:2,'.');
   writeln;

   writeln('The HISTOGRAM of the data....');
   writeln;
   for i:=1 to h do
     hist[i]:=round(140*(hist[i]/N));
   histogram;

   autocovar;
end.
```