AN INTELLIGENT COMMUNICATION CONTROLLER FOR THE VME BUS

by

Dileep Raghunath Idate

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

APPROVED:

[illegible]

Dr. J. G. Tront, Chairman

[illegible]

Dr. A. G. Phadke

[illegible]

Dr. C. E. Nunnally

February 1989

Blacksburg, Virginia

AN INTELLIGENT COMMUNICATION CONTROLLER FOR THE VME BUS

by

Dileep Raghunath Idate

Dr. J. G. Tront, Chairman

Electrical Engineering

(ABSTRACT)

This document explains the design of the microcontroller based Intelligent Communication Controller (ICC) for the Motorola VMEbus. The card transmits and receives serial data on T1 medium at a rate of 1.544 Mbits/sec. This ICC card is a part of the communication system used in a current differential protection scheme for power distribution systems.

Acknowledgements

I wish to express my sincere gratitude and appreciation to Dr. J. G. Tront for his guidance, suggestions and constant encouragement which has made this project a possibility. I would like to thank Dr. A. G. Phadke for giving me the opportunity to work on this interesting project, and Dr. C. E. Nunnally for his time and attention as a member of my graduate committee. I would also like to thank Bob Lineberry for his invaluable suggestions during the development of this project.

I would like to express my gratitude to all my friends for their constant support and encouragement and to my parents, to whom I dedicate this work.

Table of Contents

List of Illustrations

List of Tables

Chapter I

INTRODUCTION

1.1 The Project Background

American Electric Power (AEP) and the ASEA RELAYS of the ASEA group have started a joint project to examine the possibilities of using fiber optic systems in protective relay applications. Protective relays are the units which sense the abnormal conditions on a power line, such as, very high current or voltage and in response energize a circuit breaker to isolate the faulty power line. Today's protective relays are implemented with microprocessors and other digital hardware. Electromagnetic coils and other similar analog equipment are no longer used. Optical fiber technology has already made its entry into the communications field and it will probably play an important role in future protective relays [1].

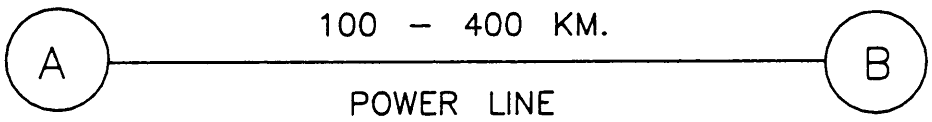
A network consisting of a single power line, about 100 to 400 Km. long, is considered here. Points A and B are the end points of the transmission line. There is a need to exchange information between these points, and therefore a communication link is needed between the two substations at the end points. The information communicated consists of digitized samples of various parameters such as voltage, current, phase angle, etc. The synchronizing pulses, containing the timing information describing the sampling of the terminal point parameters, are also sent. By comparing the sampled patterns, it is possible to detect a fault in the power network. If a fault is present, then it must be located, the faulty phase must be selected, and the faulty line isolated. This must occur promptly and safely. Therefore, the demands on the communication system, in terms of speed, will be considerably high [1].

The proposed system communicating between two power stations is shown in Figure 1 on page 4. Each station has a VMEbus-based system, which is typically used to monitor the power line and gather the appropriate data. The data needs to be exchanged between the two VMEbus-based systems at the two stations. The communication link between the two stations will partially consist of a twisted shielded pair and partially of an optical fiber, with the later constituting a major portion in terms of length. The communication interface performs the conversion of electrical signals to optical signals and vice versa. The fiber optic system has a high bandwidth and very low noise susceptibility, which allows large amounts of data to be transferred rapidly and faultlessly.

This thesis describes the design of an Intelligent Communication Controller (ICC) for the VMEbus-based systems at the end stations. The ICC does the job of transmitting and receiving data from one station to another. The initial functional requirements of the ICC card are given below. The design has been tailored to fulfill these requirements.

- The ICC shall be capable of continuous data transmission and reception at a rate of 192,000 bytes per second.
- The serial data channel shall be full duplex.
- The ICC will notify the application program upon receipt of a block of data.
- Similarly, the ICC will also notify the application program upon completion of the transmission of a block of data.
- The interface to the VMEbus backplane shall be accomplished through the P1 connector only.
- Any address on the communication card shall be user configurable.
- The electrical input/output signal levels, timing and bit framing shall correspond to Bell T1 Time Division Multiplexing (TDM) data communication standards.

The next two sections give an overview of Bell T1 TDM standard and the VMEbus.



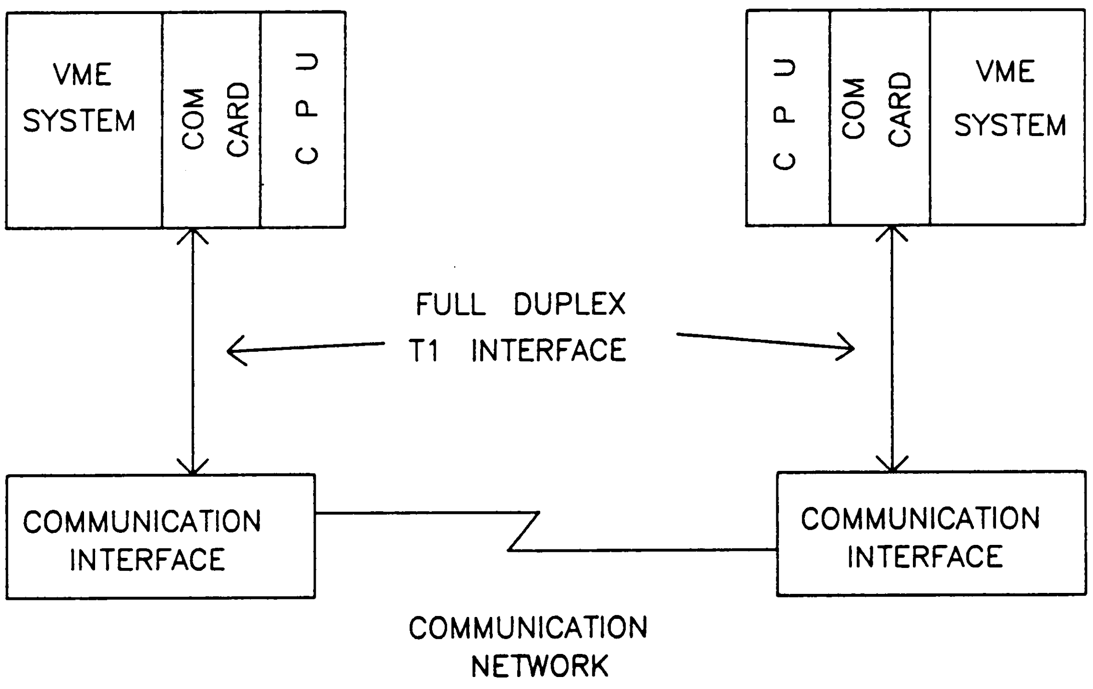


Figure 1. Block diagram of the basic communication system

1.2 Overview of Bell T1 TDM

T1 is a Pulse Code Modulation (PCM) format for time division multiplexing 24 voice channels (telephone) or data circuits onto a single transmission path. This path is normally a twisted pair cable with digital repeaters at intervals of 6000 feet [2].

T1 presently has two major formats. The older D4 format and the new Extended Superframe Format (ESF). The major differences between them are in the multiframe structure and the definition of the F bit pattern. The ESF contains 24 frames per multiframe while the D4 format has 12 frames per multiframe. At this time, the ICC only supports the D4 format.

There is a hierarchy of PCM formats within the T carrier system that defines further time division multiplexing of multiple T1 lines. Two T1 lines are multiplexed to produce T1C, four T1 lines produce T2, seven T2 lines produce T3, and six T3 lines are multiplexed to produce the T4 carrier. This hierarchy is shown in Figure 2 on page 6 [3].

The basic T1(D4) format is shown in Figure 3 on page 7. Each of the 24 voice channels are sampled at a rate of 8 kHz and coded as an 8-bit word. The resultant channel messages are word interleaved to form an uninterrupted sequence of 192 bits to make a frame. An extra bit (193rd bit or F bit) is inserted at the beginning of each frame to define the frame boundaries. Since the voice is sampled at 8 kHz, the frame rate is 125 μs. To transmit 193 bits in 125 μs requires a bit rate of 1.544 Mbits/second. Hence the standard T1 frequency is 1.544 MHz [2].

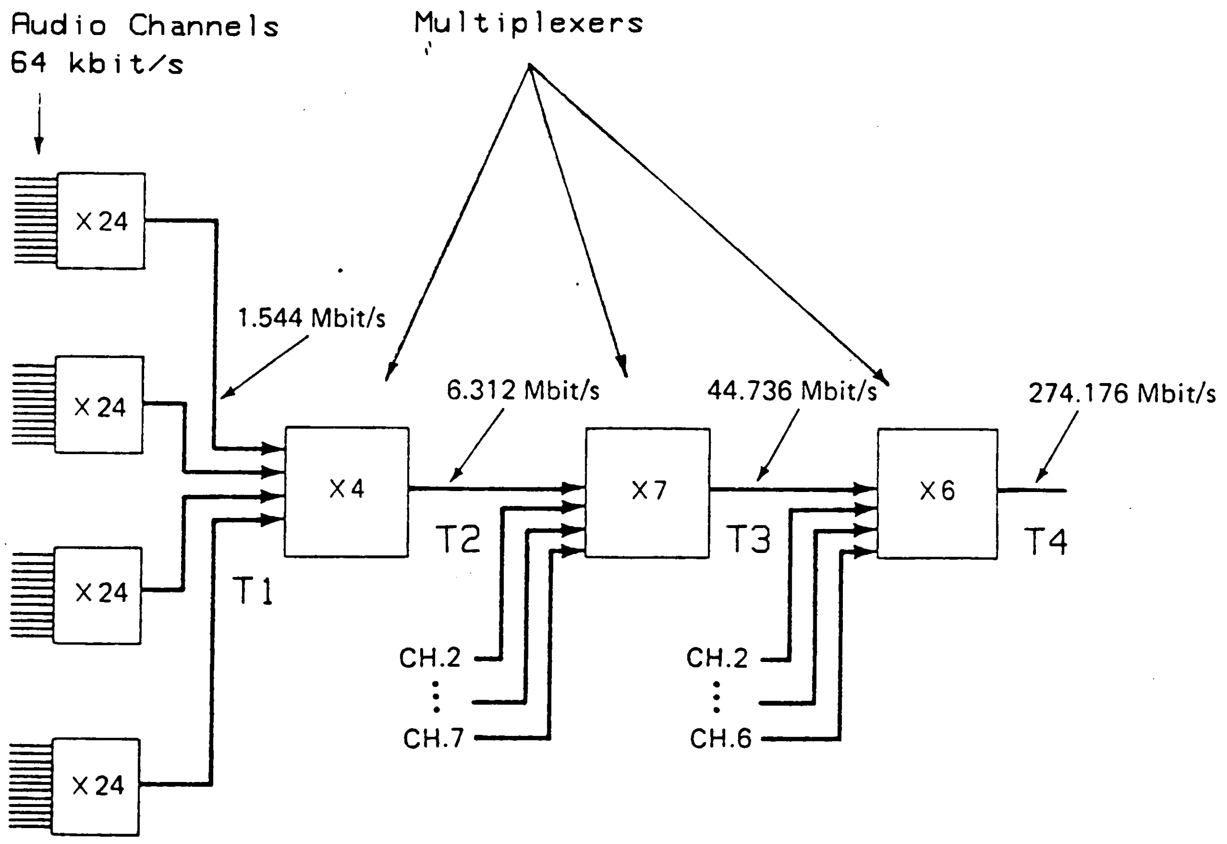
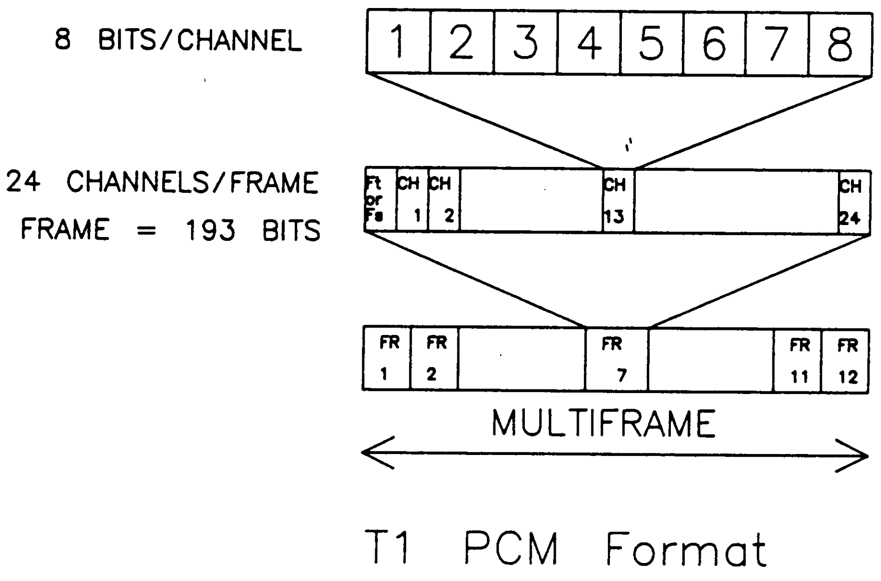


Figure 2. Hierarchy in the T carrier system



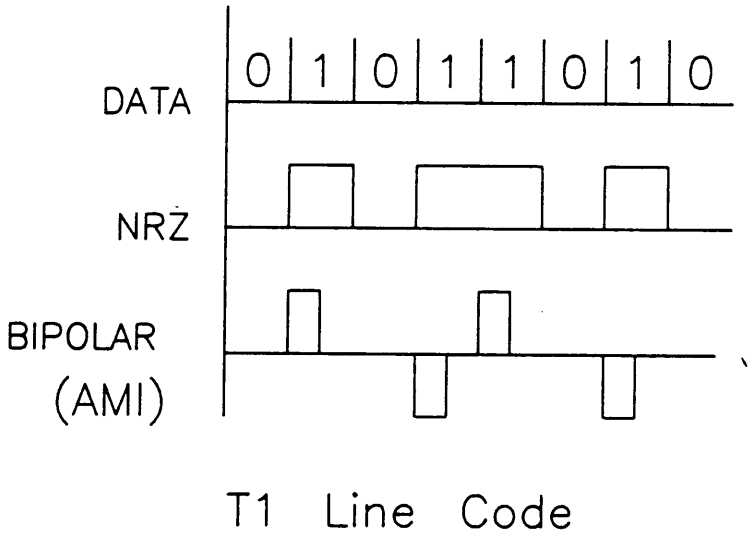


Figure 3. The T1 line format and line code

A multiframe is comprised of 12 frames. The structure of a frame and a multiframe is defined by the F bit pattern. The F bit is designated alternately as an F(t) bit (terminal framing bit) and an F(s) bit (signalling framing bit). The F(t) bit carries a pattern of alternating 0s and 1s (101010) and defines frame boundaries so that one channel may be distinguished from another, while the F(s) bit carries a pattern of 001110 and define the multiframe boundaries. The F(t) bit is attached at the beginning of odd frames (1st, 3rd, 5th, ...) and the F(s) bit is attached at the beginning of even frames (2nd, 4th, 6th, ...) [2].

The line code is the mapping from data bits to signal elements. The line code for T1 is Alternate Mark Inversion (AMI) or Bipolar. It is also shown in Figure 3 on page 7. In this method, zeroes are transmitted as no pulses and a 1 corresponds to a pulse in the first half of a bit interval, alternating polarity from pulse to pulse. Thus, the Bipolar has the bandwidth centered on one half the bit rate and there is no direct-current (dc) component, which is an advantage. The lack of a dc component facilitates alternating-current (ac) coupling which uses transformers and provides excellent electrical isolation between data communicating devices and their environment. If the number of consecutive zeroes exceeds some limit, then there is a chance of the receiver loosing synchronization with the transmitter. To avoid this, the Bipolar with Eight Zero Substitution (B8ZS) scheme is used. In this scheme, any sequence of eight consecutive zeroes is replaced by a 00 0VB 0VB code. Here B represents a normal bipolar pulse and V represents a pulse violating the bipolar rule (V has the same polarity as the preceding pulse). This helps the receiver in the process of clock recovery and subsequently in maintaining synchronization [2,3].

1.3 Overview of the VMEbus (IEEE P1014)

The VMEbus is an asynchronous parallel bus developed from VERSAbus, a microprocessor bus created by Motorola to meet the needs of 16-bit microprocessors, particularly the MC68000. Although close in functionality, the two buses have different factors (card size, connectors, etc.) and different timing. The VMEbus uses a Eurocard format with DIN pin-in-socket connectors. This mechanical configuration is particularly suited for applications where resistance to mechanical vibrations is important [4].

The VMEbus can be divided into 8 functional modules as noted below [5,6]:

1) System clock driver : This driver provides a 16 MHz clock (SYSCLK) to all boards that need to use it. All transactions across the bus occur independently although some modules may derive their internal timing from the system clock.

2) Power module : This module provides a 'system reset' (SYSRESET*) signal and an ACFAIL* signal. It provides +5, +12, -12 and +5 V standby voltages to the system.

3) Data Transfer bus (DTB) Master : It controls all the data transfer operations over the bus between itself and the selected slave. A master gains control of the bus through a bus requester. The master is generally a CPU or a DMA controller or any other circuitry capable of directing data transfers on the bus.

4) DTB Slave Module : It decodes the address, address modifier lines, and the strobe lines provided by a DTB master and supplies or accepts data from the master module. Slaves are generally memories or I/O devices.

5) Bus Requester : This module requests control of the data transfer bus whenever a master or an interrupt handler needs the bus. Upon getting control of the bus, it notifies the requesting device about the availability of the bus. The VMEbus supports two types of requesters: Release When Done (RWD) and Release On Request (ROR). First one releases control of the bus when the master no longer needs it. These requesters are generally associated with DMA controllers. ROR requesters release the bus only when some other board needs the bus. These requesters are associated with the CPUs.

6) Bus Arbiter : This module arbitrates requests for VMEbus mastership from various bus requesters. There are three types of arbiters: Single level, Fixed priority and Round robin priority. The VMEbus has 4 levels of bus requests. The first type, a single level arbiter, arbitrates solely the highest priority level. The fixed priority arbiter arbitrates among all four levels and gives the bus to the highest priority requester. The Round Robin arbiter gives all four levels equal access to the bus. This is accomplished by changing the priorities with each arbitration cycle in such a manner that the highest priority level in the last cycle becomes the lowest priority and the priorities of the rest of the levels increment by one step. Any number of sub-levels may exist in each level. These sub-levels are accommodated by using a daisy chain mechanism.

7) Interrupter : This module activates an interrupt signal on the VMEbus upon receipt of an interrupt request from a local module. It responds to the interrupt acknowledgement cycle by placing an 8-bit vector on the data bus. There are 7 levels of interrupts on the VMEbus. Again, sub-levels exist in each level through a daisy chain mechanism. Thus, a module closest to the master has the highest priority on that particular level.

8) Interrupt Handler : This module responds to the interrupt requests by first gaining control of the bus through a bus requester and then by conducting an interrupt acknowledge cycle. A 3-bit code is placed on the address bus to indicate the level of acknowledgement. The module finally passes an 8-bit vector placed by an interrupter to the master and terminates the interrupt acknowledge cycle.

This concludes the discussion of the VMEbus modules. The VMEbus can be divided into four sub-buses, with each performing a specific function. These sub-buses are briefly discussed below:

1) Data Transfer Bus : This bus contains 32 data lines (D00 - D31), 31 address lines (A01 - A31), 6 address modifier lines (AM0 - AM5) and control signals WRITE*, AS*, DTACK*, DS0*, DS1*, BERR*, LWORD* and IACK*. The * indicates the 'active low' nature of a signal. A data width of 8, 16 or 32 is allowed and is set up dynamically on a cycle by cycle basis. The type of data transfer is defined by an address modifier code. Out of 64 possible codes, 14 are defined as part of the bus standard, 16 are user-defined and 34 are reserved for future use. Data transfers are performed asynchronously, with the address and data presented in a non-multiplexed format.

2) Arbitration Bus : The arbitration scheme consists of the following signals:

- Bus Request lines (BR0* - BR3*)
- Bus Grant In lines (BG0IN* - BG3IN*)
- Bus Grant Out lines (BG0OUT* - BG3OUT*)
- Bus Busy (BBSY*) and
- Bus Clear (BCLR*)

The bus request lines are used by masters to signal a request for control of the bus. The BGxIN* and BGxOUT* forms a daisy chain at each level. They are used to signal granting of the bus to the active bus requester. The BBSY* line is used by the requester to show that it has control of the bus. Re-arbitration starts when BBSY* is deactivated by the requester. The use of BCLR* is optional. It is activated by the arbiter when a request of higher priority is pending. Masters can be designed to release the bus when the BCLR* line becomes active. This will prevent the master from 'hogging' the bus and will allow the higher priority master to perform its work.

3) Interrupt Bus : The following signal lines constitute the interrupt bus.

- Interrupt Request lines (IRQ1* - IRQ7*)
- Interrupt Acknowledge (IACK*)
- Interrupt Acknowledge In (IACKIN*)
- Interrupt Acknowledge Out (IACKOUT*)

Interrupt request lines are driven by interrupters to request an interrupt to the host processor. The IACK* signal is driven by the interrupt handler to indicate that an Interrupt Acknowledge Cycle is in progress. The IACKIN* and IACKOUT* signals form a daisy chain from one board to another.

4) Utility Bus : The utility bus contains SYSCLK* (16 MHz), System Fail (SYSFAIL*), AC power fail (ACFAIL*), serial clock (SERCLK), serial data (SERDAT*) and system reset (SYSRESET*). The utility bus supplies periodic timing, initialization and diagnostic capability for the VMEbus.

The VMEbus has two 96-pin connectors (J1/P1 and J2/P2). Connector J1 contains 16 data pins, 24 address pins and the pins for the arbitration, utility,interrupt and the control bus. Address and data buses can be expanded to 32-bits each through the J2 connector. J2 also contains some power and ground pins. The rest of the pins of J2 are user defined. The J1 connector diagram is shown in Figure 4 on page 14.

1.4 Organization of the Thesis

This thesis documents the design and development of an Intelligent Communication Controller for the VMEbus system. Chapter 2 describes the approach taken in a design procedure. The necessity of different layers in computer networks and their corresponding protocols is discussed in this chapter. Next, the protocol for the ICC is formulated. Finally, Chapter 2 discusses necessary hardware support for the protocol implementation. Chapter 3 explains the hardware design for the ICC. The selection of different chips is justified and the hardware is discussed in detail. Chapter 4 explains the software for operation and testing of the ICC. Chapter 5 gives infor-

PIN NUMBER	ROWa SIGNAL MNEMONIC	ROWb SIGNAL MNEMONIC	ROWc SIGNAL MNEMONIC
1	D00	BBSY*	D08
2	D01	BCLR*	D09
3	D02	ACFAIL*	D10
4	D03	BG0IN*	D11
5	D04	BG0OUT*	D12
6	D05	BG1IN*	D13
7	D06	BG1OUT*	D14
8	D07	BG2IN*	D15
9	GND	BG2OUT*	GND
10	SYSCLK	G3IN*	SYSFAIL*
11	GND	BG3OUT*	BERR*
12	DS1*	BR0*	SYSRESET*
13	DS0*	BR1*	LWORD*
14	WRITE*	BR2*	AM5
15	GND	BR3*	A23
16	DTACK*	AM0	A22
17	GND	AM1	A21
18	AS*	AM2	A20
19	GND	AM3	A19
20	IACK*	GND	A18
21	IACKIN*	SERCLK(1)	A17
22	IACKOUT*	SERDAT*(1)	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	+5VSTDBY	+12V
32	+5V	+5V	+5V

Figure 4. J1/P1 Pin Assignments

mation about usage of the card. Finally, Chapter 6 summarizes the design procedure and discusses possible future modifications. Chapter 6 also estimates the efficiency of the ICC.

Chapter II

THE DESIGN APPROACH

The aim of this thesis is to design a high speed interface card for the VMEbus to reliably transmit and receive serial data. The data is exchanged between two VMEbus-based systems at opposite ends of a power transmission line.

The design approach consists of two parts: 1) Conceptualization and 2) Implementation.

Conceptualization includes:

1. Definition of the ICC's interaction with the outside world and
2. Development of the ICC's architecture on a block diagram level

Implementation is the detailed design of the hardware and software. It is done in the following steps:

1. Selection of chips and other circuit elements
2. Circuit design using these chips
3. Development of Software
4. Validation of the design

The first part is discussed in this chapter. The hardware implementation and the software development are discussed in the following chapters.

In a system, two modules should follow a certain set of rules for proper communications to take place. This set of mutually agreed upon rules or conventions is called a protocol. The first section of the conceptualization process is the development of a suitable protocol for the ICC's interaction with the outside world. The second section defines the necessity for different functional modules to implement these protocols. The necessity of some additional modules has also been discussed in this section.

2.1 The Protocol Design

As defined above, a protocol is a set of mutually agreed upon rules that govern the exchange of data between two entities. The ICC is required to exchange information with two modules. One is the host processor of its own VMEbus system and the other is the ICC at the other end of the communication link. Two protocols are needed: 1) Host - Card protocol and 2) Card - Card protocol.

The role of these protocols can be understood with the help of the Open System Interconnection (OSI) model of a computer network architecture, as specified by the

International Standards Organization (ISO). The OSI model partitions the communication process into vertical set of seven layers. These seven layers, in the lowest to highest order, are:

1. Physical
2. Data Link
3. Network
4. Transport
5. Session
6. Presentation
7. Application

Each layer performs a subset of functions required to communicate with other system. Both the systems involved in the communication have a similar set of layers. The functions of each layer are briefly discussed below [3,6].

The Physical Layer: This layer is concerned with transmitting raw bits over a communication channel. The layer defines the mechanical, electrical and procedural interface to the net.

The Data Link Layer: This layer provides for the reliable transfer of information across the physical link. It carries out functions such as flow control, error recovery and link management. It manages the initiation, maintenance and termination of sustained data exchange.

The Network Layer: This layer controls the operation of the sub-net. It provides upper layers with independence from the data transmission and switching technologies used to connect systems. It establishes, maintains and terminates the physical route from one system to another in a network.

The Transport Layer: It provides reliable, transparent transfer of data between end points. It accepts data from the layers above it, splits it and passes it to the network layer. It also ensures that the packets of data reach the other end in an orderly manner.

The Session Layer: This layer provides the control structure for the communication between applications. It establishes, manages and terminates connections (sessions) between cooperating applications running on machines at the either end of a communication link.

The Presentation Layer: It provides independence to the application processes from the differences in data representation (syntax). It provides services such as data compression, data encryption for security and conversion between character codes such as ASCII and EBCDIC.

The Application Layer: This layer provides access to the OSI environment to the users and also provides distributed information services. The content of the application layer is determined by the individual user. When two user programs communicate, those programs alone determine the set of allowed messages to be exchanged and the action to be taken upon receipt of a message.

All seven layers of the OSI model are necessary only in the very sophisticated, multiuser, timesharing systems connected to a complex network. The simpler the computer, the simpler the network, and thus fewer the number of layers required. In the design of the ICC, only three layers are considered: The Application, Data Link and the Physical layer. Other network layers are unnecessary for the following reasons:

- The computer system is a very simple uniprocessor, single user system.
- The two communicating systems are not connected through a network, but rather by a simple point to point link. Therefore, any switching and routing decisions are not needed.
- The two systems are identical and therefore compatible with each other in the use of data format. Also, data encryption is not needed.

The host-card protocol consists of the Application layer protocol. This protocol will be used by the application program running on the host or the ICC when either needs to communicate with the other. The card-card protocol consists of the Data Link layer and the Physical layer protocols. These protocols are used by the ICC to establish a communication channel with another ICC. The nature of these three protocols is discussed below.

2.2 Host-Card Protocol

2.2.1 The Application Layer

The information is exchanged between the host and the ICC over the VMEbus. Thus, the physical medium has already been defined. The kind of information that needs to be exchanged is primarily defined by the initial requirements:

- The host will issue command to the ICC to transmit a block of data.
- The card will notify the host upon completion of the transmission of a block of data and upon receipt of a block of data.

Thus, the host must pass commands and information to the ICC while the ICC needs to pass certain information back to the host. The primary command issued by the host is 'send data'. The host also gives some miscellaneous commands to control the configuration of the ICC to suit its needs. For example, the information passed to the ICC with the send command consists of the number of words to be transferred and an address of the beginning of a block of data. The nature of the other commands and the parameters passed with them are hardware dependent and therefore are discussed in the chapter on hardware.

The information that is passed from the ICC to the host is of following types:

- Positive acknowledgement on completion of the transmission of a block of data.

- An error code, if an error occurs during the transmission. The code explains the nature of an error.
- Number of bytes received upon the reception of a block of data.

This exchange of information between the application program and the ICC constitutes the application layer.

2.3 Card-Card Protocol

2.3.1 The Physical Layer

The physical layer defines following characteristics for the physical link:

MECHANICAL SPECIFICATIONS: The physical lines carrying the serial data are twisted shielded copper pairs. Thus, there is only one two-pin output terminal and one two-pin input terminal.

ELECTRICAL SPECIFICATIONS: The electrical input to the twisted pairs, electrical output levels from the twisted pairs, pulse shape and pulse jitter correspond to Bell system DS1 TDM data communications standard. Data is formatted as per the Bell T1 carrier.

2.3.2 The Data Link Layer

The Data Link protocol is the most important protocol in the design of the ICC. This protocol carries out various functions such as flow control, error recovery and link management. It manages the initiation, maintenance and termination of sustained data exchange. These functions are examined below.

The term flow control means controlling a flow of data from a transmitter to a receiver. Flow control is necessary when the transmitter transmits data at a rate faster than the receiver can receive it. In this case, overflow can occur at the receiver end resulting in a loss of data if flow control is not applied. In the case of the ICC, the need for flow control can be eliminated through proper buffering of data at the receiver. In other words the problem of overflowing can be solved by putting enough buffer space to hold the received block of data. Therefore, flow control is not implemented in the Data Link protocol.

Error checking and recovery is of importance if the transmission medium is unreliable and noisy. Noisy lines result in the corruption of data. The communication link between the two VMEbus based systems consists mostly of optical fiber. The optical fiber has very high noise immunity and is reliable. Hence, the received data is assumed to be reliable and an error recovery scheme is not included.

Link management is concerned with the orderly exchange of data over the link. The Data Link protocol of the ICC essentially does the function of link management. It initiates, maintains and terminates the process of transfer of data from one card to another.

Two types of special messages are used in the link management. One type is 'Flag' message and the other is an 'Acknowledgement (ACK)' message. The Flags are control characters sent by the transmitter to the receiver and the ACKs are the response of the receiver to the transmitter's Flags. To understand the usage of Flags and ACKs let us examine a simple example. Consider a station A and a station B at opposite ends of a transmission link. Suppose station A wants to send data to station B. Then A will first send an 'Enquiry' flag to station B asking if it is ready to receive data. After sending this flag, A will wait for B's response. Station B will send either 'Yes' or 'No' ACK. If station B sends 'Yes' ACK then A will start transmission otherwise it will either abort the transmission process or make the enquiry again. Station A will note the start of data with a 'Start' flag and the end of data using a 'Stop' flag. The receiver will start accepting data from the 'Start' flag and stop the reception upon receipt of the 'Stop' flag.

The Data Link protocol used in the ICC is very similar to the one explained in the example above. The protocol is shown in Figure 5 on page 25. Instead of using an explicit 'Stop' flag, the transfer count is sent to the receiver along with the enquiry flag. Here the 'Init' flag does the job of enquiry. The receiver stops receiving when the number of bytes received equals the transfer count. The Data Link protocol is explained below.

- The transmitter starts the process of transmission by sending INIT flag.
- Right after INIT flag, the transmitter also send the transfer count.
- The transmitter then waits for an acknowledgment from the receiver and repeatedly sends a HOLD flag.

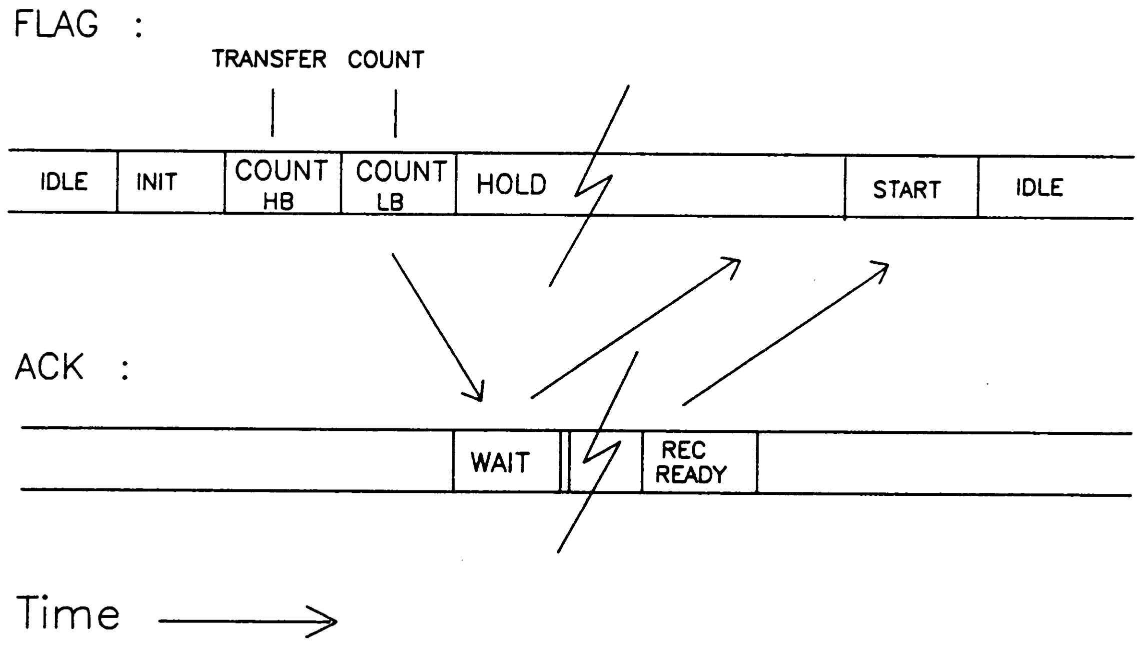


Figure 5. The Data Link Protocol

- If the transmitter does not get an ACK within some fixed time period, it aborts the transmission process.
- On receiving INIT flag, the receiver sends back WAIT flag telling the transmitter to wait.
- The receiver then checks to determine if its buffer is empty. If it is empty then the receiver sends a REC-READY ACK; otherwise,it waits until the buffers are emptied and then sends a REC-READY ACK.
- On getting REC-READY ACK from the receiver, the transmitter sends START flag and starts the transmission of data. The transmitter goes idle after completing transmission.
- On receiving START flag, the receiver starts accepting data. It also keeps track of the number of bytes received. When the number equals to the transfer count, receiver stops reception and goes idle.

The protocols have thus been defined. The necessity of different functional modules for an implementation of the protocol is discussed in the next section. The next section defines the architecture of the ICC card.

2.4 *Architectural Definition*

The necessity of a Central Processing Unit (CPU) from the previous discussion is obvious. The ICC needs a CPU to implement the Application layer and the Data Link layer protocols. The communication between the host and the ICC is really the communication between the host CPU and the ICC's CPU. A CPU is also needed to decide the transmission of Flag and ACK characters and to interpret the received Flags and ACKs. This CPU is the 'intelligent' unit on the communication card and hence the name 'Intelligent Communication Controller'. The ICC relieves the host processor of performing the underlying processes involved in data communication, such as transfer of data to and from the main memory and to and from the transceiver unit and the management of the link. Thus the CPU makes the Communication Controller card independent and truly intelligent. Each of the protocols will be separately considered. The need of any supporting hardware for the CPU to carry out these protocols will be discussed.

2.4.1 Implementation of the Application layer

The host CPU and the ICC's CPU need to pass information to each other. This can be done in the following ways:

1. A part of the global memory can be reserved for message passing. This memory will act as mailbox. One CPU can put information in these mailboxes and the other CPU can read that information from the mailboxes.
2. The second way is to keep this memory in the ICC itself. This memory can either be single ported or dual ported. If a single ported memory is used, the host CPU will have to first gain control of the local bus when it needs to access this memory. If a dual port memory is used, one port can be connected to the VMEbus and the other to the local bus. Thus each processor will have independent access to the memory.

A dual port memory on the ICC is a better scheme. If a global memory is used, then the local CPU will need to have access to the VMEbus, and therefore it will need extra circuitry such as a bus requester. The access will be slow due to arbitration time for the VMEbus. Also, the host processor cannot get control of the VMEbus for its own activity at a time when the local processor is accessing the mailbox memory. All these disadvantages are neutralized by having the memory on the ICC itself. So, this scheme is used in the ICC.

The other factor to be considered in the host-card communication is the way in which each of the processors should know about new information in the mailboxes. This can be done in two ways:

1. Through Polling: Each processor can set a 'Flag' when it writes data to the mailboxes. The other processor can keep polling this flag at regular intervals, and read the data when the flag is set. The flag will be reset by the processor after reading the data.

2. Through Interrupts: One processor can generate an interrupt to the other processor when it has written new data to the mailboxes. The other processor will read the data on receiving an interrupt.

The first scheme is very inefficient since the processor will have to keep polling to check for new data. This wastes a lot of processor's time. Also, if one processor writes new data, just after the other has polled, the new data will be recognized only after time equal to the polling interval. The second method does not have these disadvantages and thus is quick and efficient. The second method is used in the ICC.

The hardware required to implement the Application layer protocol is as below:

- A dual port memory is needed. This memory element can be a RAM or registers. The size of this memory depends on the size of the information to be passed. The message from the host typically consists of a byte long command and a few bytes of parameters, while that from the local processor consists of a byte long acknowledgement to the host's command and some parameters. In both the cases, the largest parameters are the transfer count (two bytes) and the starting address of the block (three bytes), which adds up to five bytes. Thus each CPU needs at the most six bytes of space in the dual port memory.
- Two interrupter modules are needed: One for generating an interrupt from the local processor to the VMEbus; one for an interrupt from the VMEbus to the local processor.
- Address and address modifier decoders are needed on the VMEbus side for the host's access to the mailbox memory.
- VMEbus buffers are needed for the address, data and the control lines from the VMEbus to the dual port memory.

2.4.2 Implementation of the Data Link Layer

The nature of the Data Link protocol has already been discussed. This section explains implementation of this protocol. The basic idea involved in this protocol is to send Flag characters from the transmitter to the receiver and ACK characters from the receiver to the transmitter for an orderly transfer of data.

The input/output connections of the transceiver consist of four wires: Two wires for a transmission loop and two wires for a reception loop. The transmission loop of one ICC is the reception loop of the other and vice versa. So, it is evident that Flag and ACK characters have to be sent on the same physical link along with data.

Flag and ACK characters are sent along with data through time multiplexing. The characters are inserted in the data stream at regular intervals and at fixed locations with reference to frames and multiframes of the T1 format. Therefore, some space in these data frames has to be reserved for Flag and ACK characters. There are four types of Flag characters: viz. IDLE, INIT, START, and HOLD. There are three type of ACK characters: viz. IDLE, WAIT, and READY. These characters are coded in nibbles. A Flag nibble and an ACK nibble are combined to make a single, byte wide, FLAG/ACK character. Along with the Flag characters, the transmitter also needs to send a transfer count to the receiver. Transfer count is two bytes long. Transfer count also needs to be sent in the flag space. Therefore, it is broken in to four nibbles and sent in four consecutive flag spaces. Figure 6 on page 31 shows the insertion of FLAG/ACK into data stream. The FLAG/ACK characters have been inserted as the first channel of every n'th frame. The characters are removed at the receiver end from the same channels in data. When there is no data transmission, the data space is filled with 'idle' code, which is 7FH. The modules which are necessary for implementation of this scheme are as below:

1. Registers: Two 8-bit registers are needed. One to store the FLAG/ACK character to be transmitted and the other to store the received character.

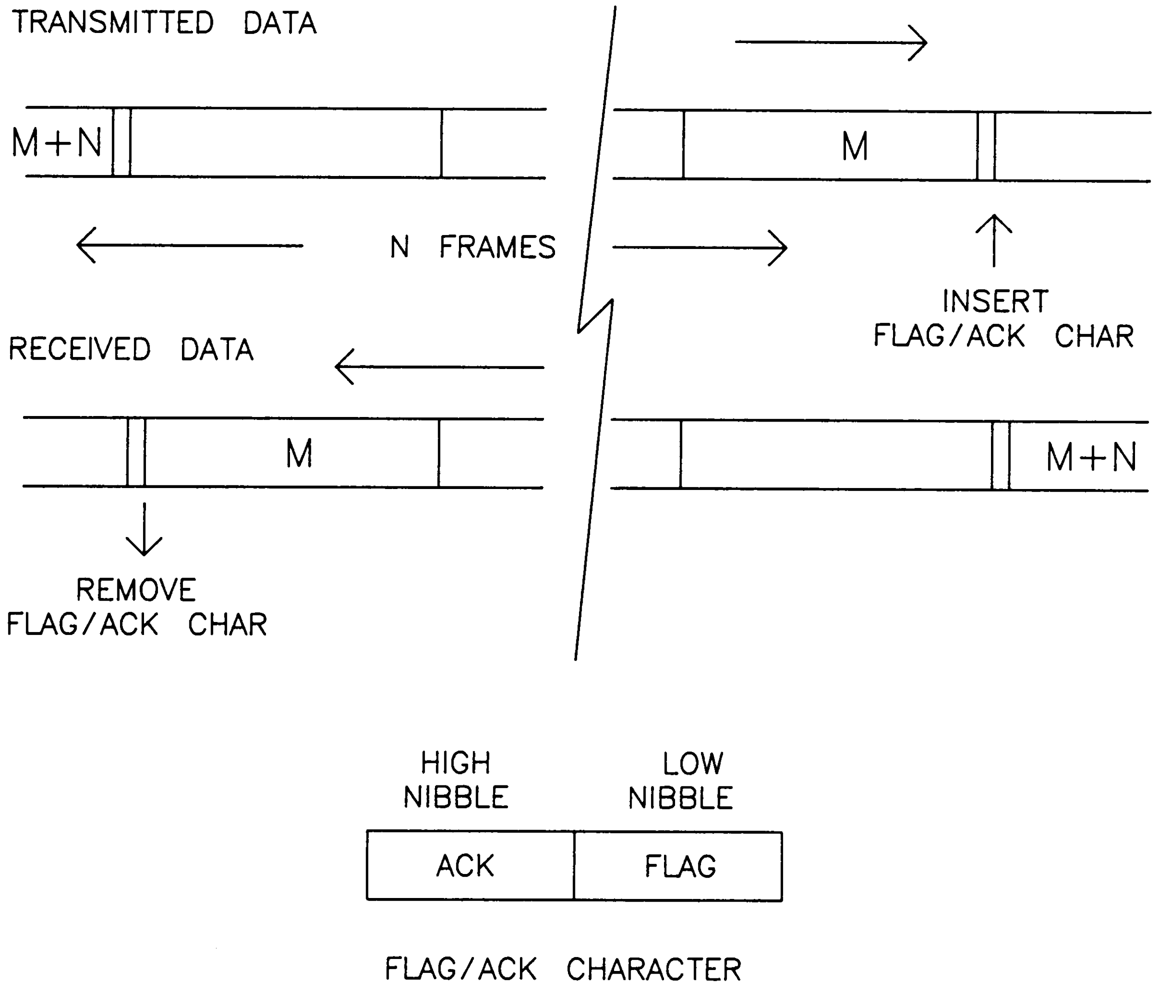


Figure 6. Insertion of the FLAG/ACK character in the data stream

2. Pulse Generator: It is needed to generate the timing pulses necessary to insert and remove the FLAG/ACK characters from data stream at the predetermined moment.
3. Synchronizer: The synchronizer is needed to synchronize the pulse generator with the frame and multiframe boundaries of the T1 format. This is necessary for inserting and removing the FLAG/ACK characters from the data at the same channels.
4. CPU: It is responsible for refreshing the transmitter's FLAG/ACK character buffer with a new character when the previous has character been transmitted. The CPU also processes the received FLAG/ACK character.

These four modules together implement the Data Link protocol.

2.4.3 Implementation of the Physical Layer

As per the requirements, the data to be transmitted must be formatted according to the T1 carrier specifications and sent on a twisted shielded pair (ABAM cable) with the line code corresponding to Bell system DS1 TDM standard.

The module which implements this layer has some digital circuitry as well as some analogue circuitry. The digital circuitry performs the following jobs:

- For transmission
    - Accept parallel data, put it into frames and multiframes as per the T1 format and insert the F(t) and F(s) bits.
    - Transmit data serially.

    - Implement the B8ZS scheme to avoid transmission of 8 consecutive zeroes.
- For reception
    - Accept serial data stream of bits and synchronize itself to the frame and multiframe boundaries by decoding the F(s) and F(t) bit patterns.
    - Convert serial data into parallel form.

Analogue circuitry is needed to interface to the input/output lines. The transmitter needs to drive the line and maintain the DS1 standard parameters such as pulse shape, electrical signal levels, etc. The receiver needs to recover the clock from the input data.

The functional modules necessary for the Physical layer, Data Link layer and the host-card communication have been defined above. The way to transfer data to and from the main memory to the on-board buffer memory is formulated below.

We have already seen the necessity of a buffer memory to avoid the necessity of the flow control mechanism in the Data Link layer. Typically a Direct Memory Access controller (DMAC) is used to transfer data. The buffer memory is necessary for one more reason. That is, without the buffer memory, the DMAC will have to transfer data on a byte by byte basis. To do this the DMAC will need the whole of the VMEbus bandwidth and it cannot have it since the host needs the bus periodically to service interrupts. Buffer memory allows the DMAC to work in a block transfer mode, which is much faster than the byte by byte transfer, requiring less bus bandwidth. The DMAC can easily do the job of block transfer when the host does not need the bus.

The buffer memory can be implemented in two ways:

- Single Port Memory
- Dual Port Memory

If a single port memory is used to store the data temporarily, then the maximum length of the block of data will be limited by the amount of buffer memory available on the card. In this scheme, if the host wants to transmit data, then the DMAC will have to move the whole block of data from the main memory to the buffer memory before the transmission can be started. Similarly, at the time of reception of data the DMAC will have to wait until the whole block of data has been received before it can start transferring the received data to the main memory.

If a dual port memory is used, the DMAC has exclusive access to one port and the transceiver is connected to the other port. This configuration has the following advantages over the single port memory system.

1. Since both the DMAC and the transceiver have access to the buffer memory at the same time, both can work simultaneously. In the process of transmission, the DMAC can first bring a fraction of the whole block of data into the memory and let the transmitter start the transmission. The DMAC can bring in the rest of data while the transmission is going on. The DMAC can do this job by transferring data in small packets instead of moving the rest of the block in one process. Thus, the DMAC will not hold the VMEbus continuously for a long time and let other boards in the system use it. This results in quick starting of transmissions and increased performance of the overall system.

2. The other advantage is that, the dual port memory does not place any restriction on the total length of the data block to be transmitted or received. This is because

as the memory is being emptied by the transmitter or filled by the receiver, the DMAC can simultaneously bring in new data to fill the empty memory and keep the transmission from being interrupted. At the same time it can transfer data from the receiver's buffer to the main memory to make space for incoming data.

Because of these advantages, a dual port memory has been used in the ICC card. To implement the DMAC-memory scheme, the following modules were needed.

1. A DMA controller
2. Dual Port RAM
3. VMEbus Requester. DMAC needs this unit to request mastership of the VMEbus.
4. Bus buffers between the VMEbus and the DMAC for data, address and control lines.

Programming of the DMAC is done by the local CPU. The CPU needs control of its local bus to carry out the Data Link protocol. So, the CPU cannot halt at a time when the DMAC is transferring data. Therefore, the CPU and the DMAC should be able to work independently. This is allowed by isolating the two buses with tristate bus transceivers. These transceivers need to be activated only at the time when the CPU is programming the DMAC or reading its internal registers.

Until now, we have discussed the way to implement different layers of the OSI communication standard. The central controller of all the modules is the CPU. The CPU does various jobs, such as, interpreting commands from the host processor, interpreting the received FLAG/ACK characters, refreshing the transmitter's FLAG/ACK buffer, programming the DMAC, etc. All these jobs need to be timed precisely . For example, the CPU should know when a new FLAG/ACK character has been received or when to write a new character to the transmitter's FLAG/ACK buffer. The necessity of these events can be informed to the CPU through interrupts. For example, there is a need of a piece of circuitry that can be associated with the receiver buffer to generate an interrupt when the particular number of bytes have been received. On receiving this interrupt, the CPU can program the DMAC to transfer those many bytes to the main memory. Similarly, a circuit is needed to generate an interrupt on the reception and on the transmission of a FLAG/ACK character and on the completion of a DMA transfer. The exact nature of this circuitry depends on the type of CPU and hence it is discussed in the chapter on hardware.

In summary, architecture of the Intelligent Communication controller (ICC) include following modules.

1. A CPU
2. Dual port memory or registers for the communication between the host and the ICC.
3. Address and address modifier decoders.
4. Interrupt unit to generate an interrupt to and from the VMEbus to the local processor.
5. A DMA Controller
6. A dual port buffer RAM.
7. A bus requester.
8. A T1 transceiver.
9. Buffer registers to hold FLAG/ACK characters.
10. Pulse generator.
11. Interrupt circuitry for the CPU.

12. Bus buffers to the VMEbus and buffers to isolate the CPU bus and the DMAC bus.

Thus, the ICC architecture has been defined on the functional level. The block diagram of the ICC architecture is shown in Figure 7 on page 38. The block diagram shows the interconnections amoung the different modules of the ICC. The realization of these functional modules at the chip level is discussed in the next chapter.

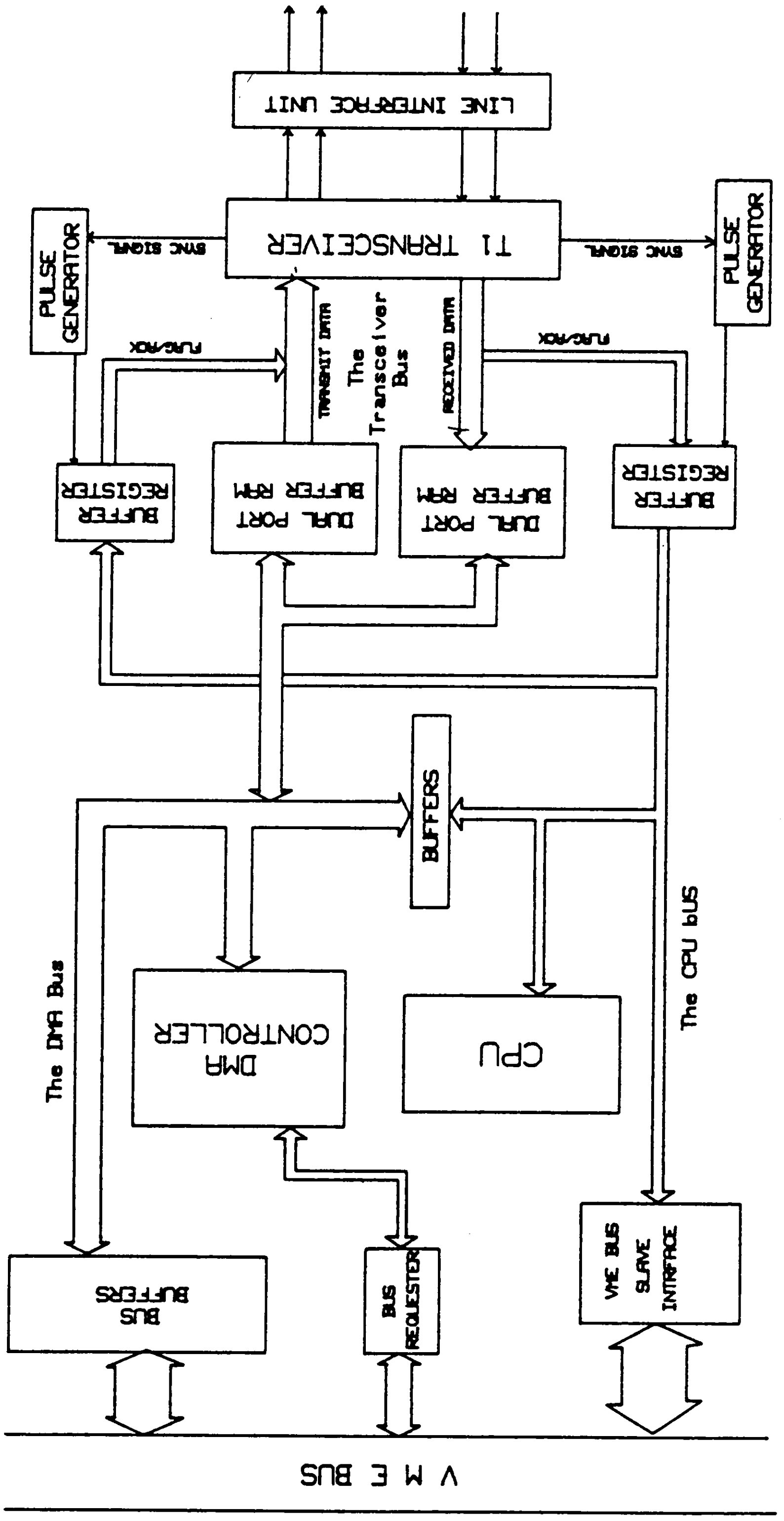


Figure 7. Block Diagram of the ICC Architecture

Chapter III

HARDWARE DESIGN

In the last chapter the architecture of the ICC card was defined. This chapter presents hardware design based on that architecture. The hardware design can be broadly partitioned into following sections:

1. Selection of chips
2. System Design
3. Development of a Prototype Card
4. Testing of the Prototype Card

Each of these sections is discussed below in detail.

3.1 Selection of Chips

Every module necessary for the operation of the ICC card is considered one by one and suitable chips are chosen to implement it.

1) CPU: The Intel 8751 Microcontroller Unit (MCU) is used as a CPU. This 8-bit MCU has four 8-bit ports. Two are used for the address and the data bus, one for the control lines and one port is used for some special control signals. These control signals are explained later in this chapter. Some salient features of the microcontroller are [8]:

- The chip has a powerful Boolean processor.
- The chip has two internal timers.
- The chip has 4K bytes of on-chip EPROM and 128 bytes of data memory. Part of the RAM and all the ports can be addressed on a bit level giving flexibility in the usage.

2,3 & 4) The VME bus interface: The dual port buffer registers, interrupts and the address and address modifier code decoders are implemented with the usage of the Performance Technologies chip PT-VSI. The chip exclusively implements the VMEbus slave interface. Its major features are [9]:

- 15 dual port mailbox registers.
- An Interrupt generator to the local processor on a read or a write to any of these mailbox registers from the VMEbus.

- On-chip address and address modifier decoders.
- Full seven level VMEbus interrupter.

The interrupt vector and an interrupt level are programmable by the local CPU. The address and the address modifier code to which the chip responds, when acting as a slave to the VMEbus, are also programmable by the local CPU. The mailbox register can be configured to generate an interrupt to the local CPU either on a read or a write cycle from the VMEbus. The local CPU can enable or disable the interrupt from any of the 15 mailbox registers. The mailbox interrupt can be routed to any of the 4 interrupt pins on the chip through programming.

5) DMAC: The Motorola MC68440 dual channel DMA controller is used. It satisfies all the requirements of the speed, address-data width and has a capacity for the required memory to memory transfers. The features of the chip are given below [10].

- 16-bit data bus, 24-bit address bus.
- Programmable function code lines (FC0 - FC2). They are used to generate the address modifier codes.
- Two independent DMA channels with programmable priority.
- Flexible request generation methods
- Transfer rate up to five Megabytes per second at 10 MHz.

The requirements for the size of address and data bus (24-bit address and 16-bit data) are obvious from the VMEbus size. The required minimum speed limit is explained below.

To find the required speed for the DMA operation, the VMEbus system operation is taken into consideration. The system typically has one master card with Motorola MC68020 processor and an Analog to Digital (A/D) converter card. The A/D card generates an interrupt to the host processor every 1.4 ms. The host processor needs 600 to 900 μs to service the interrupt. It is assumed that the host needs control of the VMEbus while servicing the interrupts. On an average host needs the VMEbus for 800 μs to service an interrupt. Therefore, the DMAC can get the bus for at the most 600 μs per 1.4 ms. In other words, the DMAC can get 42.8 % of the bus bandwidth. The DMAC needs the maximum bus bandwidth at the time when the transmission as well as the reception is going on. The rate of data transfer at this time is 192000 * 2 = 384000 bytes per second. Therefore, the DMAC should be fast enough to transfer 384000 bytes in 0.428 seconds.

The main memory on the host's card has a typical access time of 650 ns. The access time for the local buffer RAM about 100 ns. For these access times, the MC68440 DMAC needs at the most 16 clock cycles to transfer data from one memory to another. The DMAC transfers two bytes in one transfer. Thus, with the clock speed of 10 MHz, the DMAC needs 1.6 μs to transfer 2 bytes. At this rate, the DMAC can transfer 384,000 bytes in 0.31 seconds, which is faster than the minimum required rate. This justifies the selection of MC68440 DMAC.

6) T1 Transceiver: This module is implemented using two chips: The Rockwell R8069 Line Interface Unit and the Rockwell R8070 T1 Tranceiver chip. The R8070 converts the 8-bit parallel data into a serial data stream in the T1 format and receives the serial T1 data and converts it into a parallel data. It also implements the B8ZS scheme. Transmitter in the R8069 line interface unit converts the serial data in the Non Return to Zero (NRZ) format from the R8070 to the bipolar format. It also performs the line equalization up to 650 feet of cable and takes care of the pulse shaping and the jitter tolerance. On the receiver side, the R8069 extracts the clock from the received data. The I/O interface of the R8069 to the twisted shielded copper pair is through a set of pulse transformers. The pulse transformers F27.1 and F28 from the Pulse Engineering are used in the ICC [2,12,16]. The F27.1 is used for the transmission of data and the F28 is used for the reception of data.

7) Bus Requester: The bus requester is implemented using flip-flops and primitive gates. The block diagram is shown in the Figure 8 on page 44. The signals Bus Request (BR*), Bus Grant Acknowledge (BGACK*) and Bus Grant (BG*) interface to the DMAC. The signals Bus Busy (BBSY*), Bus Request (BRQ*), Bus Grant In (BGIN*) and Bus Grant Out (BGOUT*) interface to the VMEbus through proper buffering. The bus requester makes a request for the VMEbus mastership when the DMAC activates BR*. The bus requester receives the bus grant from the VMEbus arbiter (BGIN* low) and passes it to the DMAC by activating BG* and driving BBSY* low. In response to BG*, the DMAC deactivates BR* and activates BGACK*. The requester finally deactivates BBSY* when the DMAC deactivates BGACK*. The requester passes BGIN* line onto BGOUT* if the DMAC does not need the bus.

8) Dual port buffer RAM: The Integrated Devices' IDT 7434, 4K x 8, 70 ns dual port static RAM chips are used. Two chips are used, each for the transmitter buffer and for the receiver buffer, totalling 8K bytes of space for each buffer. The chip has two sets of an address, data and a control bus. One set connects to the DMAC and the other connects to the transceiver. Since the T1 transceiver chip R8070 does not have an address bus, separate synchronous counters are used to generate data addresses for

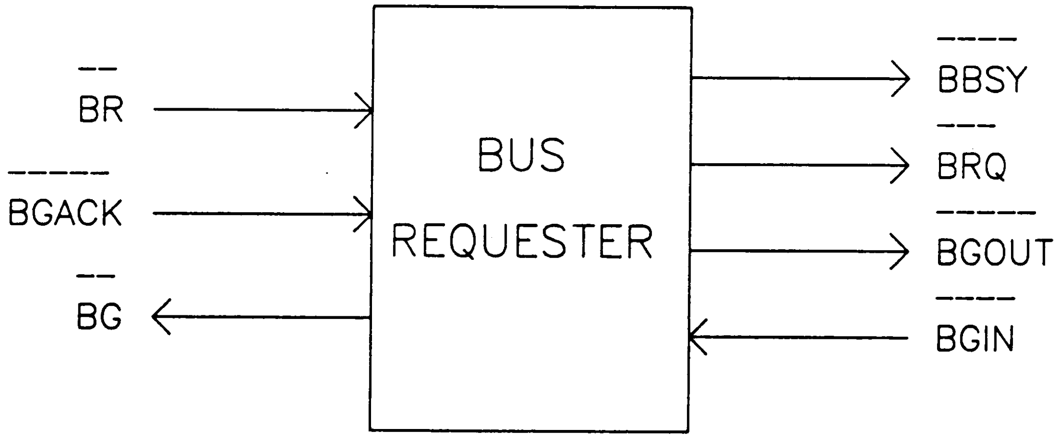


Figure 8. The Bus Requester

the transmitter and the receiver section. To generate an address up to 8K, 13 bits are required. Therefore, two 74AS867 8-bit synchronous binary counters are used to make an address generator [14]. Two such address generators are used: One for the transmitter and one for the receiver.

9) FLAG/ACK Buffers: The TTL octal D type edge triggered flip-flops 74LS374 are used to hold the FLAG/ACK characters. The transmitter's buffer is written by the MCU and read by the transceiver while the opposite is true for the receiver's FLAG/ACK buffer.

10) Pulse Generator: The Intel 8253 Programmable Timer chip is used to generate the timing pulses required to remove or insert the FLAG/ACK character in the data stream. The 8253 has six modes of operation. It works as a rate generator in mode 2. The duration between the pulses generated in this mode is programmable. A synchronizer circuit has also been designed to synchronize the 8253's operation to the frame/multiframe boundaries of the outgoing and the incoming data. The Rockwell R8070 transceiver provides the clock signals to the timers and also provides the signals required by the synchronizer.

11) Interrupt Circuit: Any event on the ICC which needs to be executed by the MCU is initiated through an interrupt. There are six types of interrupts on the ICC. Every interrupt and its purpose is explained below.

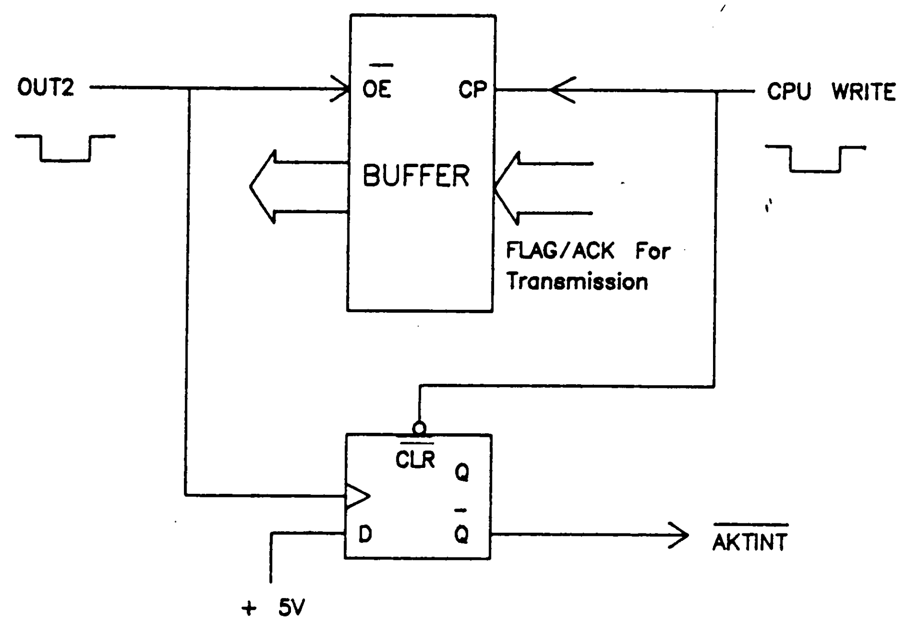
VMEINT* This interrupt is generated by the PT-VSI chip, when its mailbox register MB00 is written by the host processor. It tells the MCU that a new command has been written by the host. The interrupt is level active and is reset on the MCU's reading of the mailbox register MB00.

DMAINT* This interrupt is generated by the DMAC on completion of a data transfer operation. The interrupt is level active and is reset when the MCU reads the interrupt vector from the DMAC.

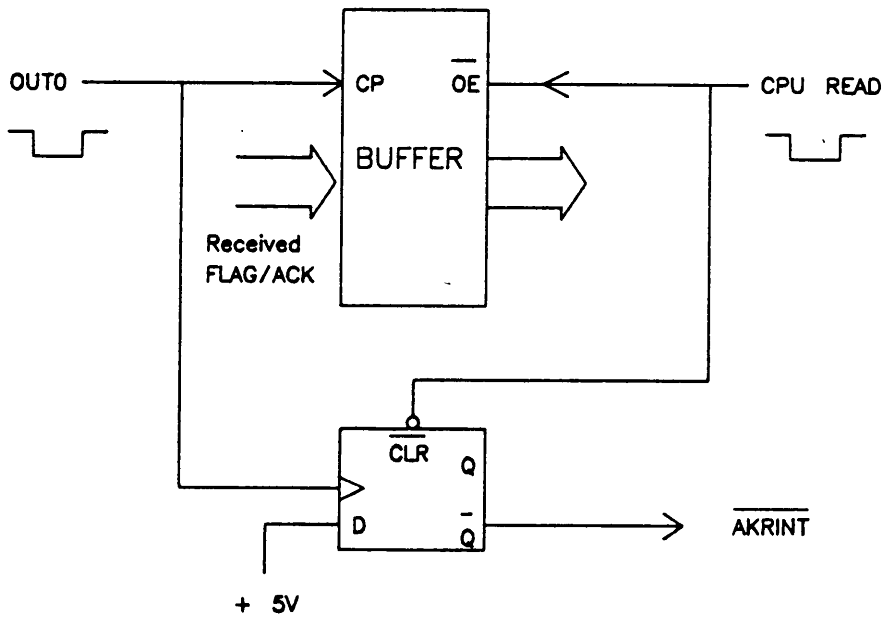
AKTINT* This interrupt tells the MCU that an FLAG/ACK character has been sent and the buffer needs to be refreshed with a new character. The pulse from the pulse generator is used to enable the buffer at the time of transmission of the character. The rising edge of this pulse is used to generate the interrupt through a D flip-flop. Input D of the flip-flop is always tied to a high level. The pulse is used as the clock to this flip-flop. The pulse strobes this high level into the flip-flop making the output Q* low. The signal Q* is used as an interrupt. The circuit is shown in Figure 9 on page 47. The interrupt is reset when the MCU writes a new character to the buffer, since the write signal to the buffer is also used to clear the flip-flop.

AKRINT* This interrupt is quite similar to AKTINT*. It tells the MCU that a new FLAG/ACK has been received and it needs to be read and processed. Interrupt mechanism is similar to that of AKTINT* and is also shown in Figure 9 on page 47. The flip-flop is reset when the MCU reads the character from the buffer.

TRINT* This interrupt signifies that a 2K bytes of data has been transmitted and the empty buffer space can be filled by transferring new data from the main memory to the buffer. Since the local buffer can store up to 8K bytes, the transmitter keeps transmitting data uninterrupted from the rest of the memory. This gives enough time to the MCU to program the DMAC and start the data transfer. The falling edge of an address line #10 of the transmitter's address counter is used to recognize a 2K byte boundary. The interrupt circuit is similar to that of AKTINT* or AKRINT* except, a



Interrupt From FLAG/ACK Transmission



Interrupt From FLAG/ACK Reception

Figure 9. Interrupt generation from the FLAG/ACK buffers

negative edge triggered JK flip-flop is used instead of a D flip-flop. The falling edge of the address line A10 strobes the input high level into the flip-flop making Q* low. The signal Q* is used as an interrupt. The interrupt is reset by clearing the flip-flop with a separate signal.

RCINT* This interrupt is generated when 2K bytes of data is received by the receiver, and needs to be transferred to the main memory from the receiver's buffer. The interrupt generation and its clearing is similar to the TRINT*'s mechanism. The address line #10 of the receiver's address counter is used to generate the interrupt.

The 8751 MCU has only two interrupt inputs: viz. INT0* and INT1*, while there are six sources on the card. These 6 interrupts have been partitioned into two groups. The first interrupt, VSIINT*, is connected to INT0* and the remaining five interrupts have been ANDed together and the output is connected to INT1*. Thus, if any of these five interrupts becomes active, ie. goes low, then the output of the AND gate will go low generating the INT1* interrupt. The individual interrupts are then recognized through software polling. A parallel port is needed for this purpose. The Intel 8255 Programmable Peripheral Interface chip provides this port [13]. The 8255 has three 8-bit I/O ports: A,B and C. Port C can be addressed on a bit level. Port A is used for polling of the interrupts. Two bits from port C are used as the reset signals for interrupts TRINT* and RCINT*.

12) Bus Buffers to the VMEbus: Buffers are used to interface the address, address modifier, data and the control buses of the DMAC to the VMEbus. The VMEbus Specification manual suggests the types of buffers to be used with each of the VMEbus lines [7]. The 74ALS645-1 bus transceivers are used for the address, address modifier and the data buses [15]. The 74LS244 octal tristate bus buffers are used to drive the control bus and to receive some of the control signals. The 74AS760 open collector buffer is used to drive the interrupt request and the data acknowledge lines. The 74S38 open collector NAND buffers are used to drive the bus request and the bus busy lines.

This completes selection of the chips. The ICC card design using these chips is discussed in the next section.

3.2 The ICC Hardware Design

A microprocessor system consists of different functional modules connected to a standard microprocessor bus and some supporting circuitry for these modules. This section explains the hardware design of the ICC card. First, the different bus structures in the card are discussed and then the design of the supporting circuitry is explained. The detailed schematic of the hardware is shown on the two sheets attached on the back cover.

3.2.1 The Bus Structure

The ICC contains three important bus structures. They are :

1. The DMAC bus system
2. The microcontroller (MCU) bus system and

3. The transceiver bus system

3.2.1.1 The DMAC Bus

The job of the DMAC is to transfer data to and from the main memory to the local buffer memory. Therefore, the DMAC is connected to the VMEbus through appropriate bus buffers. The Bus Request (BR*), Bus Grant (BG*) and the Bus Grant Acknowledge (BGACK*) lines of the DMAC are connected to the bus requester. These connections are shown in the sheet #1. One port of all the dual port buffer RAMs is also connected to the DMAC bus. The RAMs have an 8-bit data bus and the DMAC has a 16-bit data bus. Therefore, one chip of the transmitter's and the receiver's buffer is connected to the D0 - D7 bus and the other chip is connected to the D8 - D15 bus. The 3 to 8 line decoder 74LS138 is used to generate the chip select signals for the buffer RAMs. The transmitter's RAM is located at an address from 010000H to 011FFFH, while the receiver's RAM is located from 020000H to 023FFFH. The DMAC's function code lines specify the type of the current bus cycle, that is, whether the main memory is being accessed or the buffer memory is being accessed. The function code for the main memory cycle is used to generate an appropriate address modifier code for the VMEbus. The function code for the local buffer access is used to enable the chip select signal generator and to generate the DTACK* signal for the local RAM. The chip select signal generator and the DTACK* signal generator circuit is shown on the sheet #2 in the top left corner. The DMAC bus is connected to the microcontroller bus through four bus buffers. Two 74LS245 bus transceiver chips are used as an interface between the 16-bit data bus of the DMAC and the 8-bit data bus of the MCU. The 74LS244 tristate bus driver is used between the address buses and the 74AS760 open collector bus buffer is used between control buses of the two bus systems. These buffers are controlled by the MCU.

The pull up resistors are provide for all the output pins with an open collector configuration. The 8 to 3 line encoder 74LS148 is used to encode HALT*, RESET* and Bus Error (BERR*) lines onto the BEC0 - BEC2 lines of the DMAC. The 10 MHz clock is generated with a 10 MHz TTL clock oscillator and a schottky inverter. The schottky inverter is used to enhance the clock waveform by reducing the rise and the fall time. The same clock is also supplied to the PT-VSI slave interface chip, bus requester circuit and the local DTACK* generator circuit. Since the DMAC has a multiplexed address/data bus, two 74LS373 latches are used to latch the address.

3.2.1.2 The Microcontroller Bus

The MCU has a 16-bit address bus and an 8-bit data bus. The lower eight bits of an address are multiplexed with the eight bits of data. The 74LS245 bus transceiver is used to buffer the data to and from the MCU to the data bus and the 74LS373 latch is used to latch an address. Only eight bits of the address are used in the system. The address bits A8 to A11 are used to generate the chip select signals for various chips. The 3 to 8 decoder 74LS138 is used as an address decoder. The control bus of the MCU consists of Read (RD*) and Write (WR*) lines. The bus structure is shown in the sheet #1 and #2.

Various chips such as the PT-VSI VME slave interface, 8255 Programmable Peripheral Interface, 8253 timer, two buffers (74LS374) for the FLAG/ACK characters and the buffer transceivers which isolate the DMAC bus and the MCU bus, interface to the

MCU bus. Port 1 of the microcontroller is used on a bit level for various control signals. These signals are discussed below in ascending order from Bit #0 to Bit #7.

DMAHLT*	This is an active low signal used by the MCU to halt the DMAC operation.
RESET	This signal is used to reset the system
TRTMR	This signal is used to start the timer operation of the sync pulse generator for the transmission of the FLAG/ACK characters.
RECTMR	Similar to the TRTMR, but for the reception of the FLAG/ACK character.
RCSTRT*	This signal is used to start address counters of the receiver's buffer RAM for the reception of a data.
TCSTRT*	Similar to RCSTRT*, but for the transmitter's data.
REQ0*	This signal is used to make a request for a transfer of data to DMAC channel #0.
REQ1*	Request to DMAC channel #1.

Port B of the 8255 chip is used to collect data regarding the status of the transceiver. The signals RRED, MS1 and MS0 from the R8070 and the signals ES1 and ES0 from the R8069 denote different states of the receiver such as 'In the Process of Synchronization', 'Loss of Signal', etc. The active RYEL signal denotes the reception of yellow alarm. A bit from port C of the 8255 is used to enable the transmission of yellow alarm if the 'Loss of Signal' condition occurs at the receiver. The yellow alarm is denoted by sending an F(s) bit in the frame 12 as high instead of a normal low. Port A of the 8255 is used to poll the interrupts.

3.2.1.3 *The Transceiver Bus*

The transceiver bus is shown in the sheet #2. It consists of two data buses, two address buses and two control buses. One of the each is required for the data transmission and the other one for the data reception. The data buses connect the transceiver to one of the ports of a dual port buffer RAM. The address buses are driven by the address counters made up of the two 74AS867 8-bit synchronous binary counters. The control bus consists of Read, Write and Chip Select signals. These signals are derived from some of the R8070 transceiver's output timing signals by using a combinational logic. These circuits are explained in the next section.

- The R8070 T1 transceiver's input/output and clock signals are directly compatible to the R8069's corresponding signals. The connections of these signals are shown in the sheet #2. The TTL clock generator 74LS321 is used to generate an external clock of 1.544 MHz for the R8069. The serial input/ouput pins of the R8069 are connected to the pulse transformers. The pulse transformer F27.1 is used to drive the transmission line while the F28 is used to receive the data from the transmission line [14,16].

Thus, various chips are interconnected to form the ICC hardware. Some supporting circuitry is also needed in addition to this hardware. This circuitry is explained in the next section.

3.2.2 The Supporting Circuits

Three major circuits are needed in addition to the system discussed above. These are:

1. The pulse generator and the synchronizer.
2. The address counter synchronizer and controller and
3. The control signal generator for the buffer RAM.

3.2.2.1 The Pulse Generator and the Synchronizer

Timers in the chip 8253 are used as pulse generators. The pulse generator circuit is shown in Figure 10 on page 55. These pulses are used to strobe the FLAG/ACK characters into the outgoing data stream and to remove them from the incoming data stream. The process of strobing and removal should occur at regular intervals from the particular channels in the particular frames.

The first character in the first, fifth and the ninth frame has been chosen to be sent as a FLAG/ACK character. Thus, there is one FLAG/ACK character per four frames, or every ninety sixth character in the data stream is a FLAG/ACK character.

The pulses generated by the pulse generator are equal in width to the width of the channel. The timing of these pulses match with the timing of the selected channels. These pulses do three jobs :

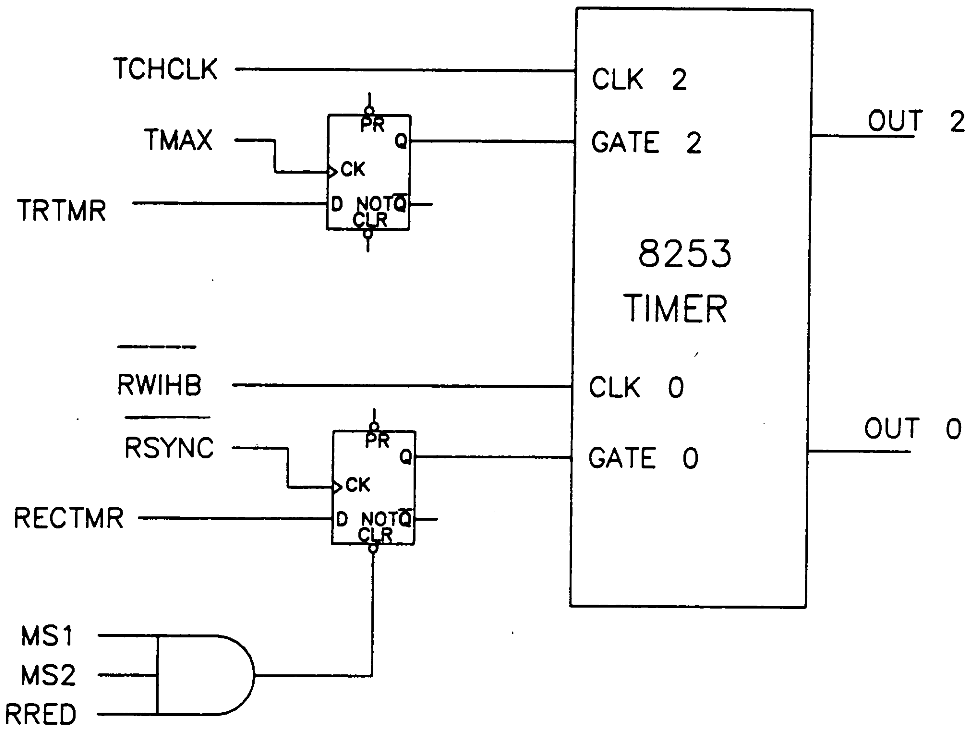
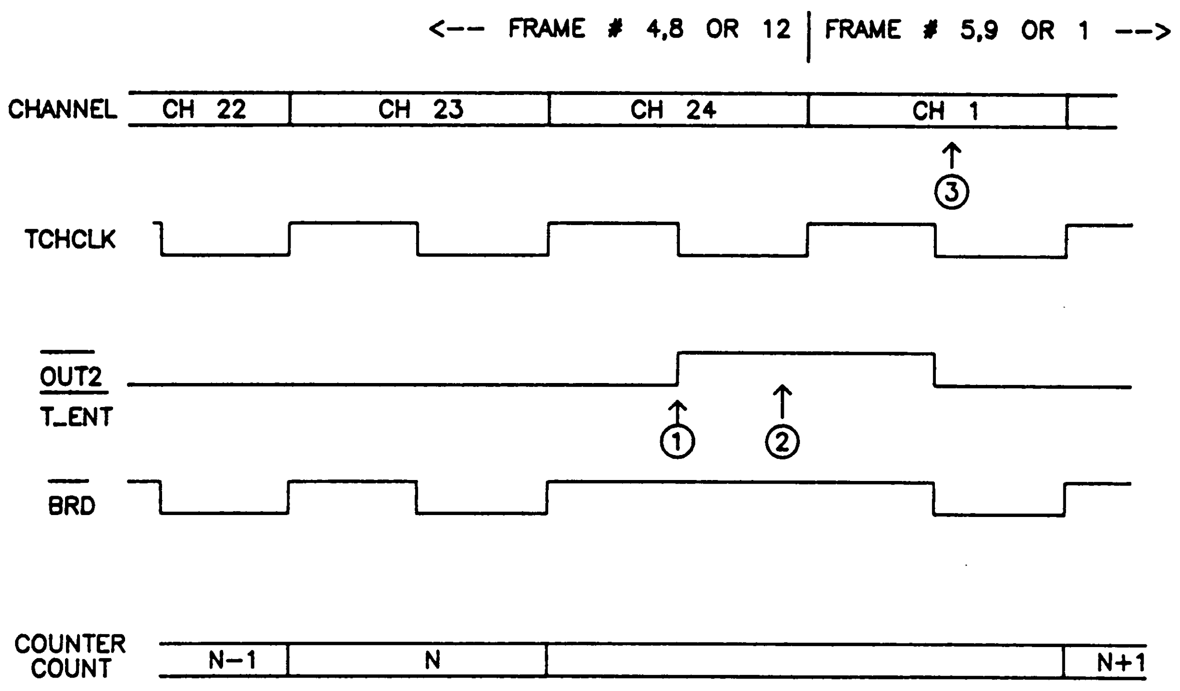


Figure 10. Timer circuit for the generation of pulses

1. They enable the FLAG/ACK buffer for insertion of the characters in the outgoing data or strobe the characters from the incoming data into the buffers.
2. They also disable the local buffer memory. This avoids bus contention in the case of transmission and avoids writing of the FLAG/ACK characters to the receiver's buffer RAM in the case of reception.
3. They freeze the address counters of the buffer RAM for that period.

The pulses are achieved by programming the 8253 timers in mode 2, which is a rate generator [13]. In this mode the output is low for one clock period after every n-1 clocks where n is a programmed count. Therefore, the timers are programmed with a count of 96. The clock should have a period equal to the channel width. The R8070 transceiver provides the suitable clocks. For the transmission of characters signal TCHCLK has is used while for the reception signal RWIHB is used as a clock. Both these signals have the same frequency as that of data: viz. 192,000 Hz. The process of insertion and deletion of the FLAG/ACK characters from data is shown in Figure 11 on page 57 and Figure 12 on page 58 respectively. The timer 2 is used for insertion and the timer 0 for deletion of the characters. The figures show the activation of buffers, disabling of the buffer memory and freezing of the address counters.

The crucial part is to synchronize these pulses with the proper channel slots. This synchronization is achieved by using D flip-flops triggered by the multiframe timing signals: viz. TMAX and RSYNC from the R8070 transceiver. The synchronization of timer 2 and timer 0 is shown in Figure 13 on page 60 and Figure 14 on page 61 respectively. The positive pulse of signals TMAX and RSYNC signify the starting of a new multiframe for the transmitter and the receiver respectively. The output of D flip-flops is given to the gate controls of the timers. The circuit is shown in Figure 10

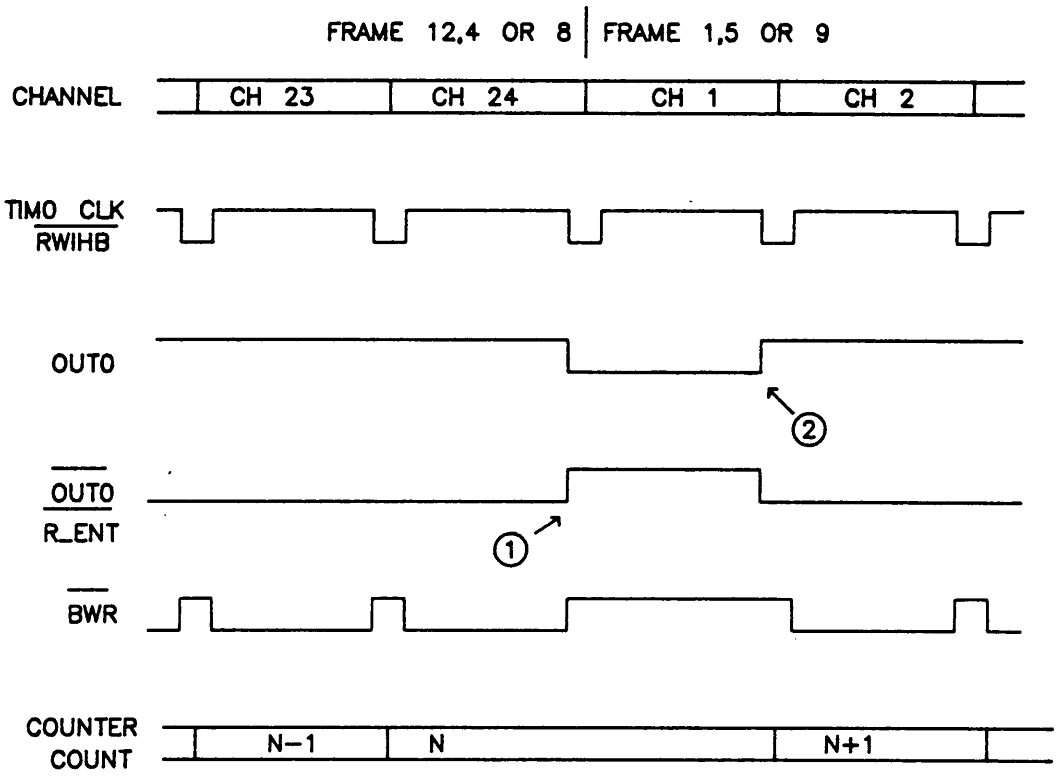


(1) FLAG/ACK BUFFER ENABLED AND MEMORY DISABLED

(2) CHARACTER READ FROM BUFFER

(3) CHARACTER TRANSMITTED IN CH. 1 SLOT

Figure 11. Insertion of FLAG/ACK character in the data stream



① $\overline{\text{BWR}}$ FORCED TO HIGH ,ADDRESS COUNTER DISABLLED

② RISING EDGE STROBES CH.1 INTO BUFFER

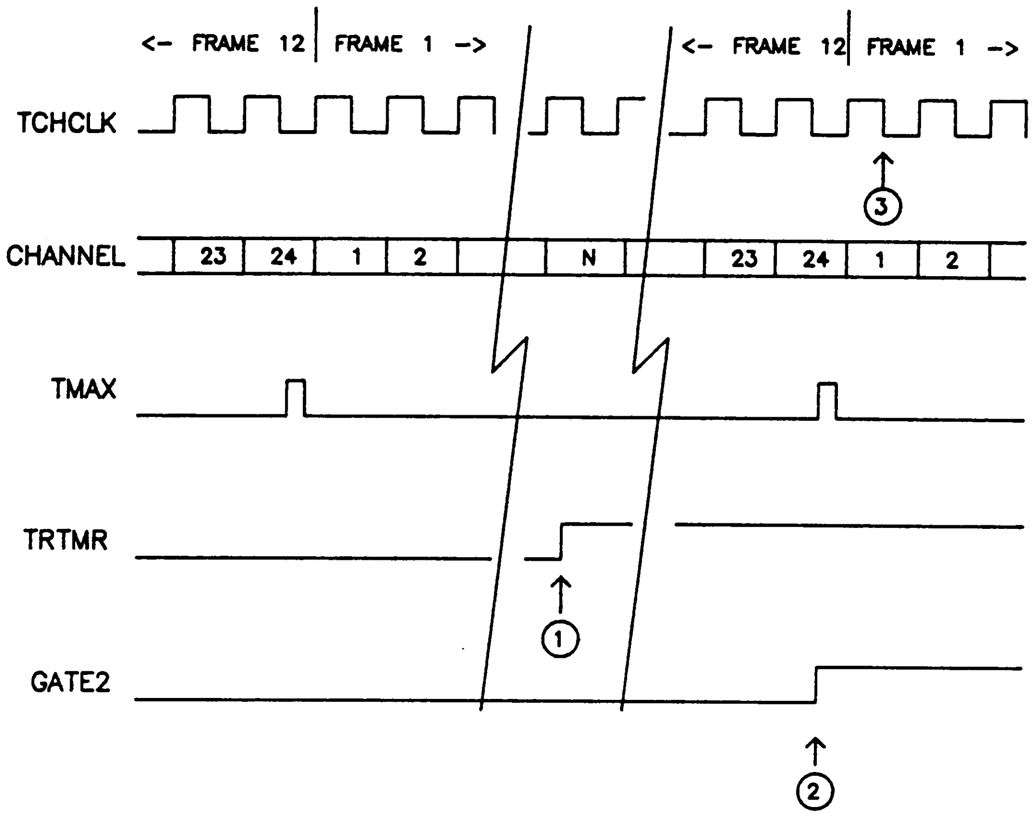
Figure 12. Removal of FLAG/ACK character from the data stream

on page 55. Therefore, the timers start counting from the start of a multiframe. The timers count on the falling edge of clock. With a count of 96 the timers produce 3 pulses per multiframe synchronized with the first channel of the first, fifth and the ninth frame. The signals MS0, MS1 and RRED signify the state of the receiver's synchronization to incoming data. All the signals are high if the receiver is properly synchronized. These signals are ANDed and the output given to the 'clear' input of the D flip-flop. This mechanism works as an auto-synchronizer. If the receiver looses synchronization, one or more of the signals MS0, MS1 and RRED will go low clearing the flip-flop and disabling the timer. The 'clear' signal will get deactivated upon restoration of synchronization. The next pulse of RSYNC will then strobe '1' into the flip-flop enabling the timer at the proper moment.

Consider Figure 13 on page 60 for the starting of the transmitter's pulse generator. Assume that the signal TRTMR, which is an input to a D flip-flop, goes high randomly at some instant. It will be strobed into the D flip-flop by the next TMAX pulse as shown in this figure. The gate will be thus enabled and the timer starts counting from the falling edge of TCHCLK in the channel 1. Thus, the starting is synchronized with the beginning of a multiframe. The same principle is used for the receiver's timer. Instead of signals TMAX, TCHCLK and TRTMR, signals RSYNC*, RWIHB* and RECTMR are used. The timer 0 starts counting from the falling edge of the signal RWIHB* in the channel #1.

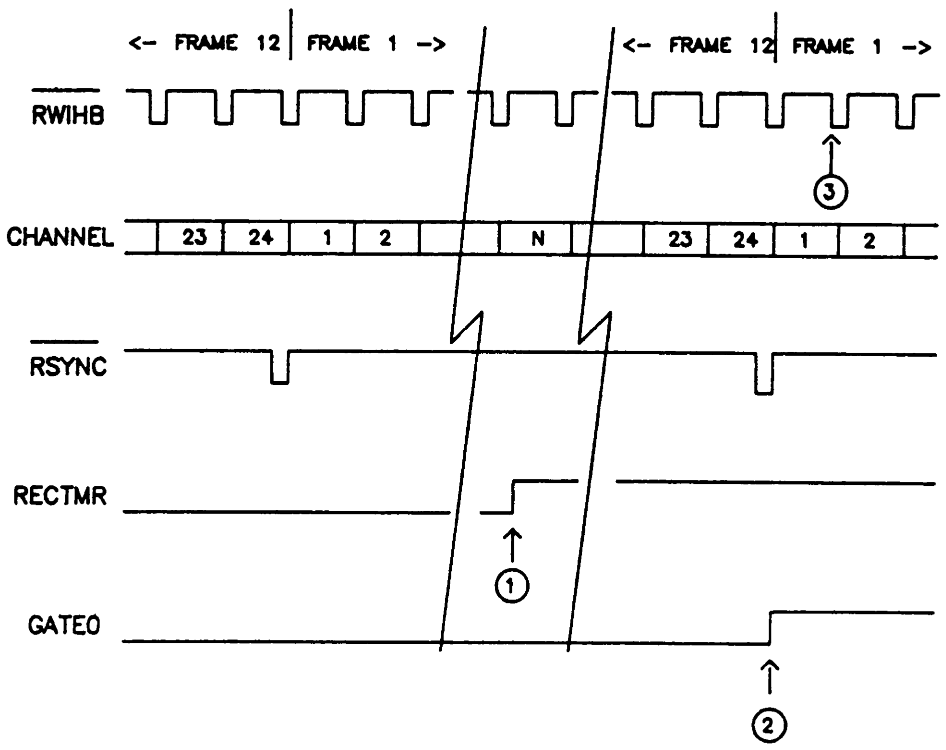
3.2.2.2 *The address counter synchronization and control*

Just as the starting of the pulse generators needs to be synchronized, the starting of the address counters also needs to be synchronized. The address counters must be



1. TIMER CONTROL ACTIVATED
2. TIMER GATE ENABLED
3. TIMER STARTS COUNTING FROM THIS FALLING EDGE

Figure 13. Synchronized starting of the timer 2



(1) TIMER CONTROL ACTIVATED

(2) TIMER GATE ENABLED

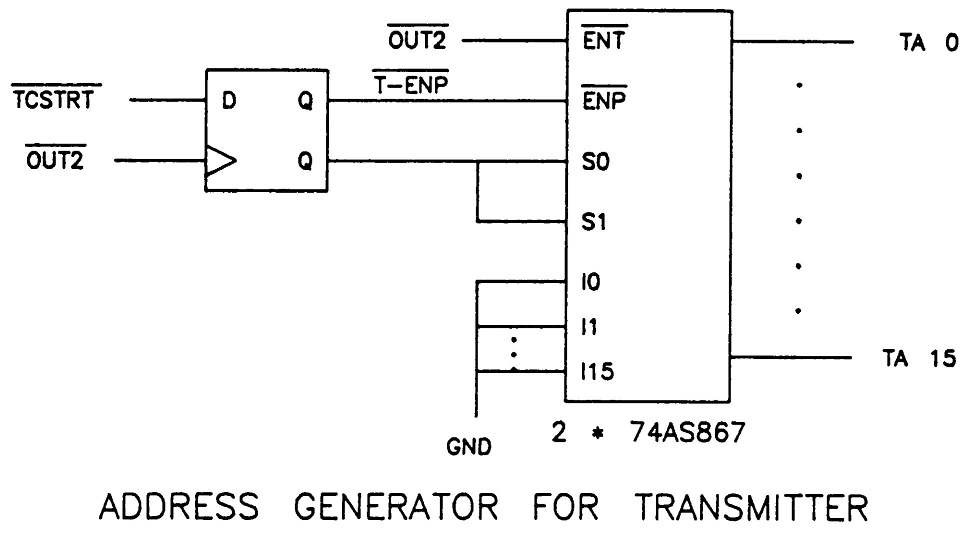
(3) TIMER STARTS COUNTING FROM THIS FALLING EDGE

Figure 14. Synchronized starting of the timer 0

synchronized with the beginning of data. The control circuitry for both the address counters is shown in Figure 15 on page 63.

By the convention of the Data Link protocol, data stream starts right after the FLAG/ACK character which follows START flag. The pulses from the pulse generator are used to insert or delete the FLAG/ACK characters. Since data starts right after the FLAG/ACK character, these pulses can be used to start the address counters too. The usage of these pulses to start the address counters is shown in Figure 16 on page 64 and Figure 17 on page 65. Consider the starting of the address counter for the transmitter. The MCU activates the TCSTRT* signal right after the transmission of START flag. The inverted output of timer 2 OUT2* is used to strobe this signal into the D flip-flop. The rising edge of the next OUT2* pulse strobes TCSTRT* into the D flip-flop and activates the signal TENP* enabling the counter. The counter has an initial value of 0000H and thus the next character to be transmitted is read from the buffer memory location 0000H. The signal TCHCLK is used as a clock for the transmitter's address counters. The counter is incremented after the transmission of a character. A similar approach has been adopted for the starting of the receiver's address counter. Refer to Figure 17 on page 65 for the timing diagram. The MCU activates the RCSTRT* signal right after receiving the START flag. The rising edge of OUT0* strobes this signal into the flip-flop and enables the counter. The signal RCHCLK is used as a clock. It increments the counter with the reception of a character.

The address counters are subsequently shut off by the MCU, by deactivating the signals TCSTRT* and RCSTRT*, on completion of data transmission or reception.



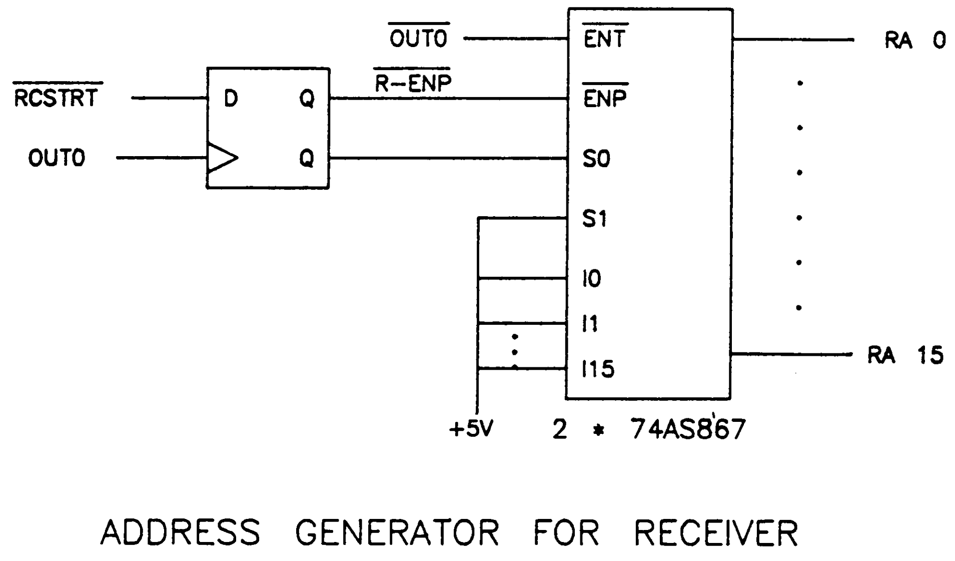
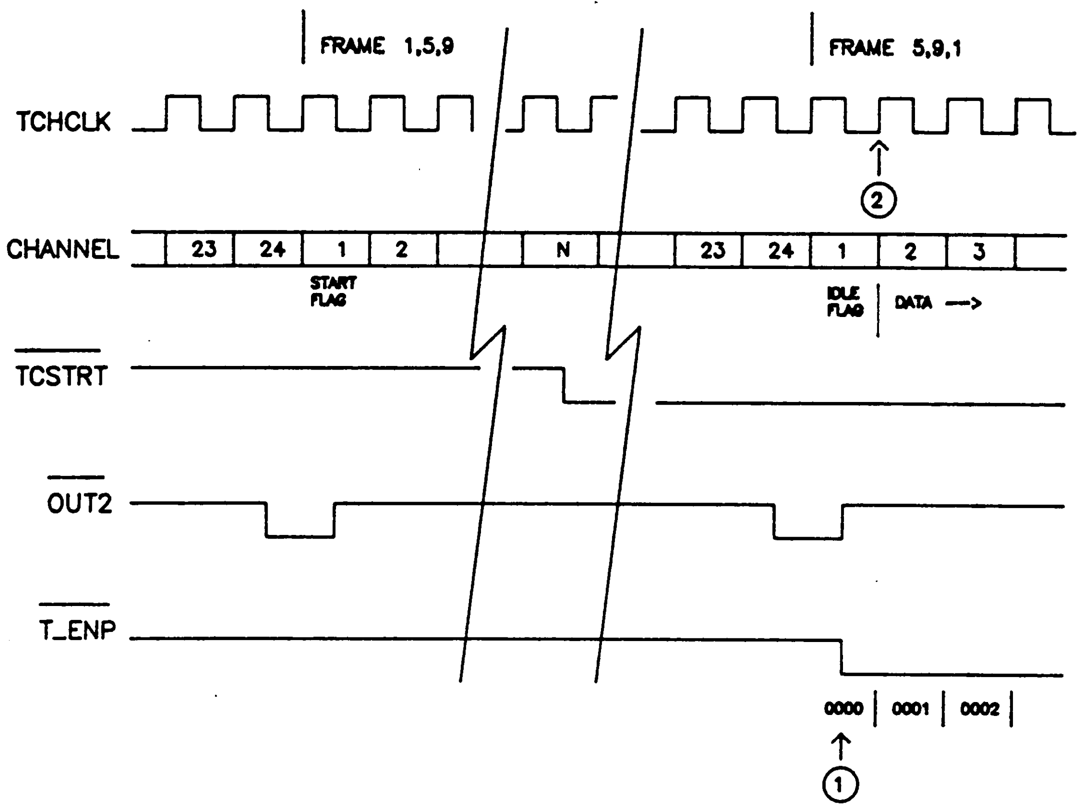


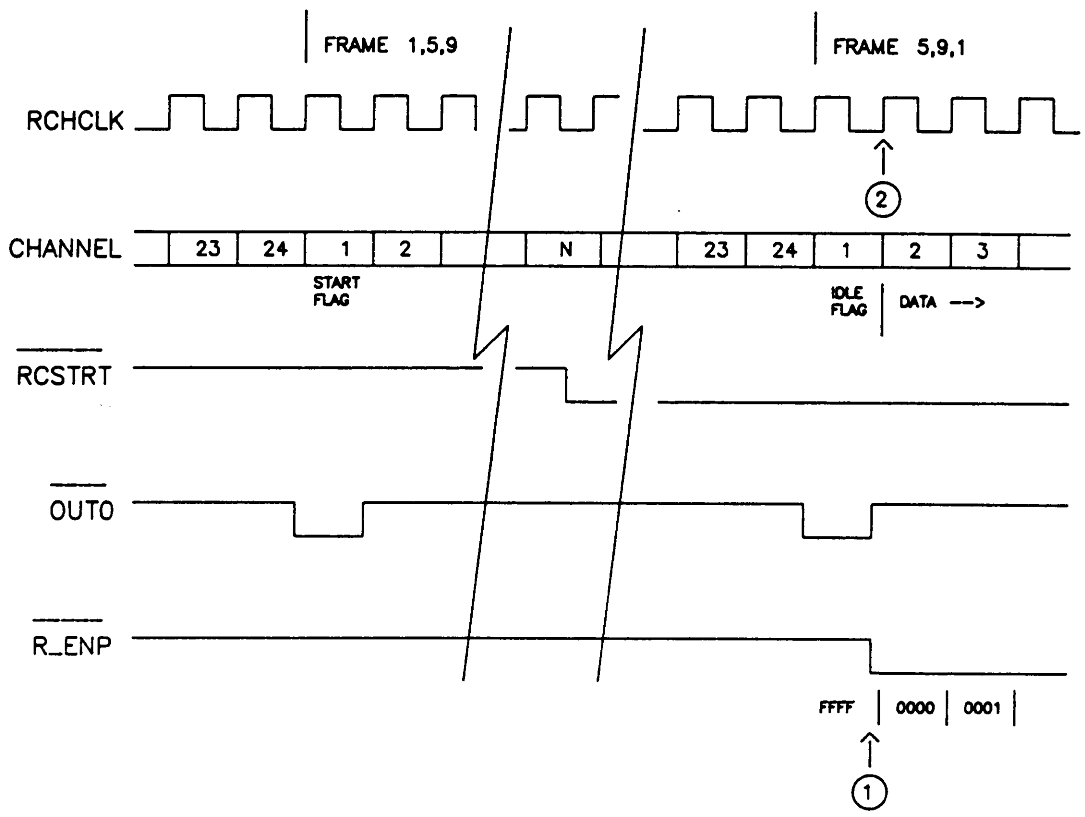
Figure 15. Control circuit for the address counters



(1) RISING EDGE OF $\overline{\text{OUT2}}$ ENABLES COUNTER

(2) RISING EDGE OF TCHCLK INCREAMENTS COUNTER

Figure 16. Synchronized starting of the transmitter's address counter



(1) RISING EDGE OF $\overline{\text{OUTO}}$ ENABLES THE COUNTER

(2) RISING EDGE OF RCHCLK INCREAMENTS THE COUNTER

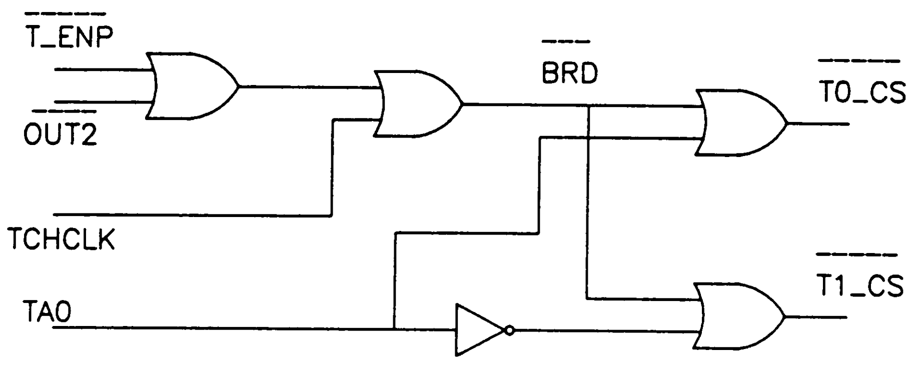
Figure 17. Synchronized starting of the receiver's address counter

3.2.2.3 Generation of control signals for the buffer RAM

The control signals considered here are for the transceiver's port of the dual port buffer RAM. The signals of the other port are connected to the DMAC bus. The RAM chip has three control lines: viz. Chip Enable (CE*), Read/Write (R/WR*) and Output Enable (OE*). The R/WR* signal of the transmitter's buffer is permanently connected to a high level since data is only read from this buffer and not written to. Similarly, the OE* signal on the receiver buffer is also connected to a high level since data is always written to this buffer. The remaining control signals of each buffer RAM chip have been combined into a single control signal. These control signals for the transmitter's buffer chips are T0_CS* and T1_CS* and those for the receiver's buffer chips are R0_CS* and R1_CS*.

The circuits to generate the control signals for the transmitter's and the receiver's buffer memory are shown in Figure 18 on page 67 and Figure 19 on page 68 respectively. Data is read from the transmitter's buffer alternately from each chip on low level of the signal TCHCLK. Data is written to the receiver's buffer alternately in each chip on a low level of the signal RWIHB. The even and odd addressed bytes reside in different chips due to the 16-bit bus on the other port of the RAM. The chips are alternately selected by using an address line A0.

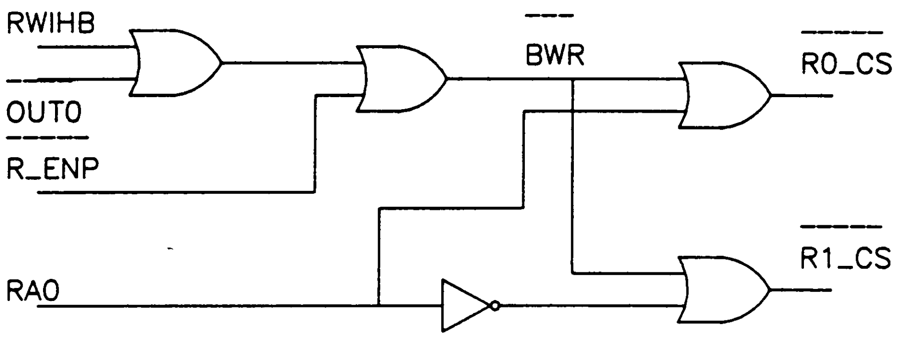
It can be seen from the truth table that the memories are accessible only at the time when the address counters are enabled (signals TENP* and RENP* at low level). It can also be seen that the inverted timing pulses from the timers, which are active high, disable the buffer memories momentarily at the time when the FLAG/ACK characters are being accessed or written.


$\overline{\text{T\_ENP}}$
$\overline{\text{OUT2}}$
TCHCLK
TA0
$\overline{\text{BRD}}$
$\overline{\text{T0\_CS}}$
$\overline{\text{T1\_CS}}$

TRUTH TABLE

INPUT				OUTPUT	
$\overline{\text{T\_ENP}}$	$\overline{\text{OUT2}}$	TCHCLK	TA0	$\overline{\text{T0\_CS}}$	$\overline{\text{T1\_CS}}$
0	0	0	0	0	1
0	0	0	1	1	0
1	X	X	X	1	1
X	1	X	X	1	1
X	X	1	X	1	1

Figure 18. Circuit to generate control signals for the transmitter's buffer


RWIHB
$\overline{\text{OUT0}}$
$\overline{\text{R\_ENP}}$
RA0
$\overline{\text{BWR}}$
$\overline{\text{R0\_CS}}$
$\overline{\text{R1\_CS}}$

TRUTH TABLE

INPUT				OUTPUT	
$\overline{\text{R\_ENP}}$	$\overline{\text{OUT0}}$	RWIHB	RA0	$\overline{\text{R0\_CS}}$	$\overline{\text{R1\_CS}}$
0	0	0	0	0	1
0	0	0	1	1	0
1	X	X	X	1	1
X	1	X	X	1	1
X	X	1	X	1	1

Figure 19. Circuit to generate control signals for the receiver's buffer

This concludes the discussion on the control signals as well as on the hardware design.

3.3 Prototype Development

Two prototype ICC boards have been made. To start with, two blank wirewrap boards with a grid of holes equally spaced at 0.1 inch were purchased. The required chips, wirewrap sockets and other miscellaneous circuit components were also purchased. The list of parts required for a single board is given in the Appendix C.

First, the wirewrap sockets were installed on the blank board. Special care was taken while preparing the floor plan of the board so that the wirelengths were minimized. Clock sources were placed as close to chips as possible to minimize the line inductance and capacitance effects on the clocks. This guaranteed optimal rise and fall time of the clock signals.

The boards were wirewrapped together in an identical fashion according to the circuit diagram shown on sheets in the Appendix E. These circuit diagrams were drawn with the help of 'PCCAPS' program of the 'P-CAD' software. The 'P-CAD' is an integrated software package for the IBM PC from the Personal CAD Systems, Inc. to aid in the development of digital systems. It contains various programs which do various jobs such as schematic capture, simulation, PCB design, etc. PCCAPS is the schematic capture program which was used to draw the two sheets of the circuit diagram in the Appendix E. Schematics for the standard TTL chips were available in the libraries but those for the chips such as microcontroller, DMAC, etc. had to be made. The corresponding footprints for these chips were also made. These footprints are needed for the PCB design.

After the circuit connections were completed, power and ground connections were done. Wirewrap pins for the decoupling capacitors were placed by the side of each chip. Special care was taken to minimize the area of ground loops. All the connections were then verified again. The process of debugging and testing of the hardware was then started. The next section explains this procedure.

3.4 Design Validation

The approach taken for testing was to test the circuit on a module by module basis. Testing was started with a minimum possible number of chips. More and more chips were eventually added so as to test more and more complex modules.

A multimeter, an oscilloscope and a logic analyzer were used in the process of testing. The multimeter was used to measure the power supply at different points in the circuit to check a drop in the supply voltage from the main supply points to a chip. It was also used to detect the level of the stable signals while the tests were in progress. The oscilloscope was primarily used for checking the clock signal waveforms. The logic analyzer was most extensively used to monitor and debug the hardware. It was used to monitor address, data and control buses and various other signals during a run of the test software.

Over a dozen test programs were written to test the ICC card. Two of these programs are shown in the Appendix A. The initial programs are small and simple, written to test a limited amount of circuit. The later programs are complex, written to test complex modules consisting of number of chips. Testing was started with the MCU. Once the operation of the MCU was tested, operation of different chips interfacing to the microcontroller such as 8255 and 8253 was tested. In all the tests, the MCU was single stepped with the help of a simple circuit shown in Figure 20 on page 72. The test input and the test output lines were connected to two unused pins of port 1 (unused in that particular test) of the MCU. To single step, one of the interrupts of the MCU was kept active continuously. The interrupt service routine for this interrupt was written to go into loop until it observed a high to low and a low to high transition on the test input pin. The test output pin was used to indicate execution of the interrupt service routine by making it low and thus turning the Light Emitting Diode (LED) on. Since the MCU executes at least one instruction in between two consecutive interrupt service routines, the scheme worked as single stepping. Each closing and opening of the push button executed one instruction of a code. The execution of all the instructions was observed on the logic analyzer.

Once all the modules were working satisfactorily on a work bench, the boards were tested on the VMEbus system. The hardware interfaces to the VMEbus such as PT-VSI slave interface chip and the DMAC, which could not be tested on a workbench, were tested first. After the completion of this testing, software for the Data Link protocol was tested and debugged to make the ICC fully functional. The next chapter explains this software in detail.

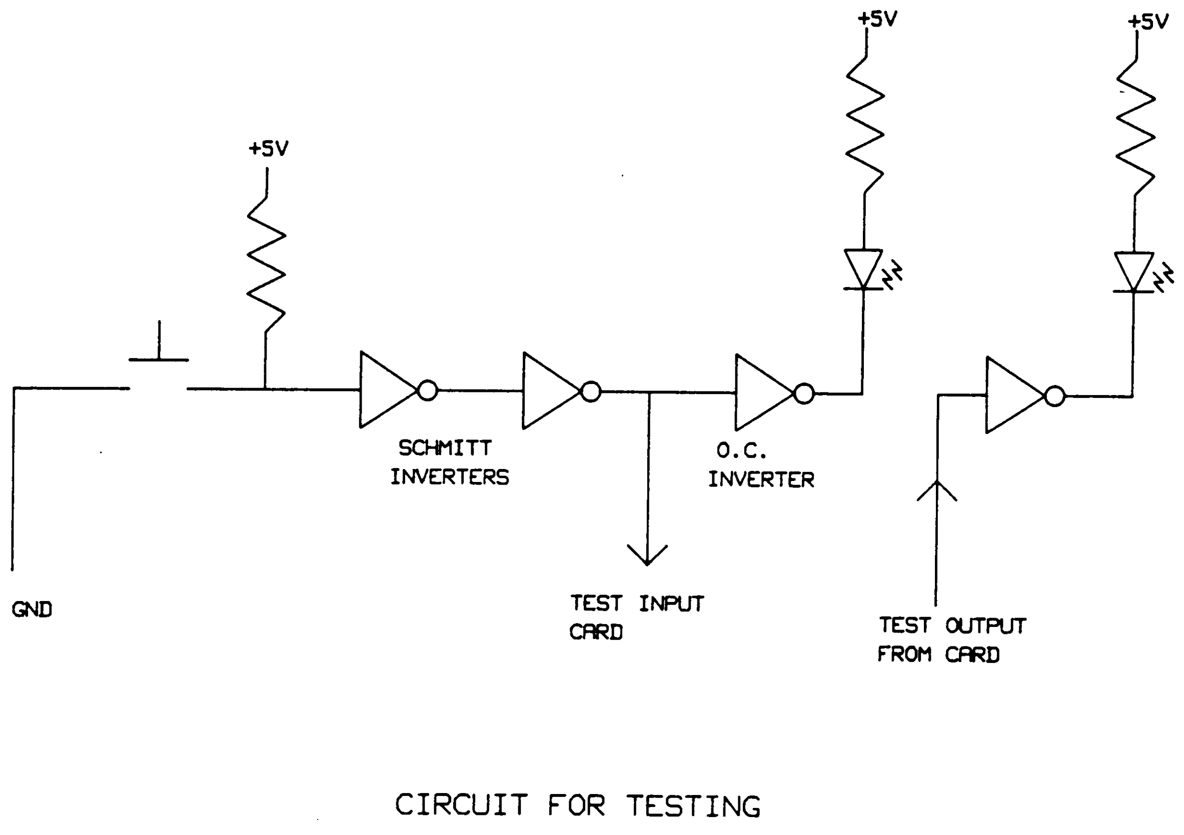


Figure 20. Circuit for single stepping the microcontroller

Chapter IV

SOFTWARE

The software needs to carry out two major functions. First is to communicate with the host and the second is to carry out the Data Link layer protocol for an orderly transfer of data. In the chapter on hardware the interrupt structure which is used to initiate an event in the protocol was discussed. The software consists of a main program and the Interrupt Service Routines (ISR). The functions carried out by these programs and the conventions used in the program are explained below. The program listing is attached in the Appendix B.

4.1 The Main Program

The main program starts executing on a power up or the microcontroller's reset. The main program resets the system and then initializes it. The initialization process consists of initializing the variables used in the program to some particular value and programming of the various programmable chips on the card. The variables used by the program reside in the on-chip RAM of the 8751 microcontroller.

After the initialization phase, the main program goes into an infinite loop. In this loop the program monitors the status of the transceiver and checks for the 'Yellow Alarm' or the 'Loss of Signal' condition. These conditions are explained later in this chapter. The main program then updates the Global Status Register (GSR) as per its findings. The mailbox register #10 in the PT-VSI is used as a GSR. It is shown below.

X	X	X	X	LOS_SIG	Y_ALRM	RC_ON	TR_ON

The usage of bits is as follows:

TR_ON	Status of Transmitter: 1 = Transmission on, 0 = No transmission
RC_ON	Status of Receiver: 1 = Reception on, 0 = No reception
Y_ALRM	Yellow Alarm: 1 = Yes, 0 = No
LOS_SIG	Loss of Signal: 1 = Yes, 0 = No
X	Don't Care

The Global Status Register (GSR) can be monitored by the host to know the status of the ICC.

4.2 Interrupt Service Routines

4.2.1 INT0_SER

The PT-VSI chip generates this interrupt when the host writes a command byte in the mailbox register # 0. The numerical value of the commands and their meaning is as below:

08: Change Interrupt Vector: This command changes the value of an interrupt vector which is used in the interrupt acknowledge cycle. The host passes the new value in the mailbox register #1. The microcontroller reads this value and updates the local variable VMEINT_NUM which stores the interrupt vector value. The default value of the interrupt vector is 64.

04: Change Interrupt Level: This command changes the level at which an interrupt to the VMEbus is generated by the ICC. The host passes the new value in the mailbox register #1. The local variable VMEINT_LEV is updated to the new value. The default value is 4.

02: Change Main Memory Address for the Received Data Block: The address of the location at which the received data block is stored is changed with this command. The three byte long new address is passed by the host in the mailbox registers 1 to 3, with the lowermost byte in MB01. The local variables RECMEM_ADR_L (Low byte), RECMEM_ADR_M (Middle byte) and RECMEM_ADR_H (High byte) which store the address are updated to their new values. The default address is 030000H.

01: Start Transmission: This is the command to start the transmission of data. The parameters are passed as follows:

Starting Address:	High Byte	MB03
	Middle Byte	MB02
	Low Byte	MB01
Transfer Count:	High Byte	MB05
	Low Byte	MB04

Here, the transfer count is a number of words and not bytes. The MCU reads the address and the count and then programs the DMAC for the data transfer. For the first transfer, the DMAC is programmed with a count of 2K words if the total block length is greater than 2K words. If not, the actual transfer count is programmed. The subsequent transfers are made in blocks of 1K word. Once the first data block is transferred to the buffer, the Data Link protocol is initiated to establish the logical link with the other card. The transmission begins on the reception of REC-READY acknowledgement from the receiver.

The way in which the transmitter and the receiver keep track of the number of bytes transmitted and the number of bytes received is explained below.

The host passes a transfer count to the ICC as two bytes. These bytes specify the number of words to be transmitted. The MCU breaks up this count into two parts. One is the number of 1K word blocks and other is the remainder count. The MCU stores this partitioned count into three variables as follows:

TRBLKS:	Number of 1K blocks
TRCNT_H:	High Byte of the Remainder count
TRCNT_L:	Low Byte of the Remainder count.

For example, let a transfer count be 9.5K words (0010 0110 0000 0000 B). For this count, the number of 1K blocks is 9H (higher 6 bits) and the remainder count is 200H (lower 10 bits). Therefore, TRBLKS = 09H, TRCNT_H = 02H and TRCNT_L = 00H.

One more variable has been defined to hold the number of DMA transfers required to bring data into the buffer memory. This variable is TRBLKS_2_XFER. Since the DMAC transfers 1K words of data in a single transfer, this variable contains the number of 1K blocks obtained by rounding the transfer count to the next higher 1K boundary. For all the transfers except the first and the last the DMAC is programmed to transfer 1K words and the TRBLKS_2_XFER is decremented by 1 with each transfer. For the first transfer the DMAC transfers 2K words and for the last transfer, that is, when the value of TRBLKS_2_XFER has become 1, the DMAC is programmed to transfer the 'remainder count' of data. For example, for a transfer count of 9.5K, TRBLKS_2_XFER = 10. The DMAC will transfer 2K words in the first transfer, 7K in the next seven transfers of 1K each and 0.5K in the last transfer.

The MCU keeps track of the number of words transmitted by counting the number of 1K word blocks that have been transmitted. The MCU counts the number of 1K blocks by using the interrupt TRINT* from the transmitter's address counter. This count is stored in the variable TRBLKS_SENT. The initial value of TRBLKS_SENT is 0. It is incremented by 1 with each TRINT* interrupt. The MCU simply rounds the transfer count to the next higher 1K word boundary and stops the transmission when

TRBLKS_SENT equals this boundary. For example, for the transfer count of 9.5K the transmitter transmits 10K words. The last 0.5K of the data will be some random data from the buffer.

The track of the received data is also kept by counting the number of 1K word blocks that have been received. The receiver also rounds the transfer count to the next higher 1K word boundary and receives that many words. This rounded transfer count is stored in RCBLKS_2_REC. The MCU uses the interrupt RCINT* from the receiver's address counter to count the number of 1K blocks that have been received. The RCBLKS_2_REC is decremented by 1 on each RCINT* interrupt The data reception is stopped when RCBLKS_2_REC becomes 0. The receiver stores the transfer count into three variables as follows:

RCBLKS_2_REC	: # of blocks to receive
	= Number of 1K blocks + 1
RCCNT_H	: High byte of the remainder count.
RCCNT_L	: Low byte of the remainder count.

The variable RCBLKS_2_XFER is used to keep the count of number of DMAC transfers required to transfer the received data to the main memory. Its value is same as the RCBLKS_2_REC. For example, for a count of 9.5K RCBLKS_2_XFER will initially contain 10 and the data will be transferred as 9 transfers of 1K each and the last transfer with the count equal to the remainder count which is 0.5K. It can be seen that the variables RCBLKS_2_REC and RCBLKS_2_XFER are analogous to the variables TRBLKS_2_SEND and TRBKLS_2_XFER.

The transmitter sends a transfer count to the receiver by splitting the two bytes into 4 nibbles. The first nibble contains the number of 1K blocks and the rest of the three nibbles contain the remainder count. Since a nibble can hold a maximum value of 15, the maximum number of 1K blocks is 15. The 10 bit remainder count can hold a maximum value of 1K-1. Therefore, a transfer count cannot exceed 16K-1 words. This is the limit on the amount of data that can be transmitted in a single transmission. This limit can be increased by increasing the number of nibbles allocated for a transfer count in the protocol.

4.2.2 FAT_SER

This Interrupt Service Routine (ISR) services the AKTINT* interrupt. It updates the value of the transmitter's FLAG/ACK character buffer. The working of this ISR is analogous to a Moore type of Finite State Machine (FSM). In this type of FSM the output depends upon the state of the machine. The FAT_SER ISR can be viewed as two finite state machines implemented in a software. One machine's output is a FLAG character and another's is an ACK character. The machines are named TR_FLG_FSM and TR_ACK_FSM. These machines form a part of the Data Link protocol. The FSM implemented in the hardware typically uses flip-flops as memory elements. The FSM implemented in a software uses a variable (a memory location in RAM) as a memory element. The value of this variable defines the state of a machine. The variables used by TR_FLG_FSM and TR_ACK_FSM are:

TR_FLG_STT:	Transmission Flag State and
TR_ACK_STT:	Transmission Acknowledgement State.

Nine states (0-8) have been defined for the transmission of a FLAG while three states (0-2) have been defined for the transmission of an ACK character. Operation of each machine is explained below.

TR_FLG_FSM:

TR_FLG_STT	MEANING
00 :	Send IDLE Flag
01 :	Send INIT Flag
02 :	Send Number of 1K Blocks
03 :	Send High nibble of a Remainder Count
04 :	Send Middle nibble of a Remainder Count
05 :	Send Low nibble of a Remainder Count
06 :	Send HOLD Flag
07 :	Send START Flag
08 :	Send IDLE flag and start data transmission.

The state is 0 in the Idle state. The Idle state can either be 'No transmission of data' or 'Steady transmission of data'. The machine goes through states 1 to 8 only during the process of establishment of the logical link to the receiver at the opposite end of the transmission link. The initiation of the Data Link protocol starts after the state becomes 1. The state is made 1 after the transmitter's buffer memory has been filled with the initial 2K words of data. After the state becomes 1, the routine itself keeps incrementing the state along with the transmission of each flag until it reaches the state 6. In the state 5 the routine sends the low nibble of a remainder count and starts the 8751's (MCU) internal timer #0. This timer is used as a watchdog timer. The timer's interrupt terminates the process of data transmission if the receiver does not respond with the WAIT ACK within certain time limit. The timer is aborted on the reception of the WAIT acknowledgement from the receiver. In the state 6 the routine keeps sending the HOLD flag until the REC-READY ACK changes the state to 7. The routine sends START flag for the state 7 and then updates the state to 8. In the state 8, the routine starts the transmission and resets the state to 0. The state diagram for the TR_FLG_FSM is shown in Figure 21 on page 82. The circles show the state and the arrows show a transition from one state to another. The event which causes the transition is shown by the side of an arrow.

TR_ACK_FSM:

TR_ACK_STT	MEANING
00 :	Send IDLE Acknowledgement
01 :	Send WAIT Acknowledgement
02 :	Send REC-READY Acknowledgement

The transmission acknowledgement state changes to 01 from 00 after the reception of the last nibble of a transfer count. The WAIT ACK character is sent in this state. The state changes to 02 when the receiver is ready to receive data. The state is restored to 00 on the reception of the START flag. The state diagram for TR_ACK_FSM is shown in Figure 22 on page 83.

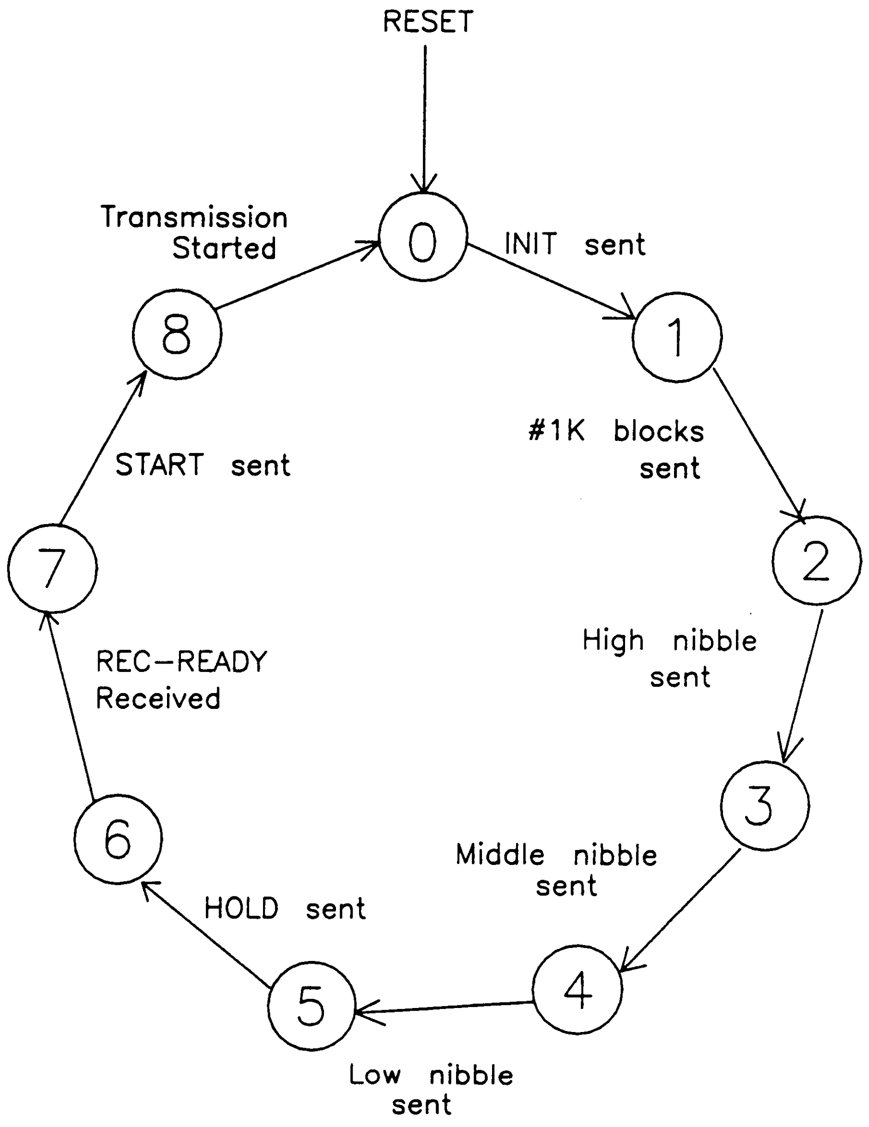


Figure 21. State diagram for the transmission of a FLAG character

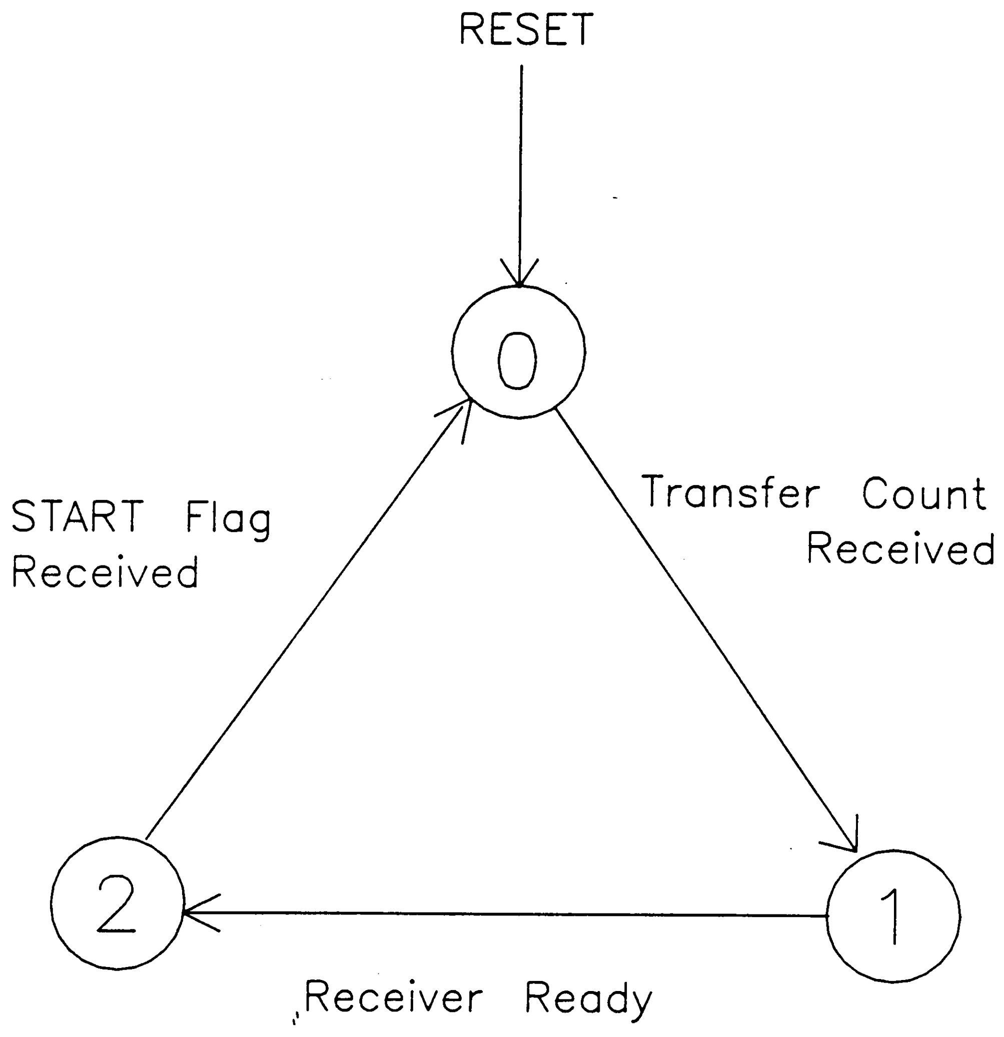


Figure 22. State diagram for the transmission of an ACK character

4.2.3 FAR_SER

This ISR services the AKRINT* interrupt. It processes the received FLAG/ACK character. The received Flag character can be either a control character or a part of the transfer count. The concept of an FSM is also used for the recognition of a Flag character. This FSM, RC_FLG_FSM, has 5 (0-4) states. The variable RC_FLG_STT holds the state of the machine. The nature of the received Flag character is identified by the state. The states are required to distinguish between the control characters and a transfer count since a transfer count can imitate the control characters. The value of RC_FLG_STT and the interpretation of Flag is as below.

0 :	The flag is a control character. It can be IDLE, INIT, HOLD or START depending upon its value.
1 :	The flag is a nibble of a transfer count containing the number of 1K blocks.
2 :	The flag is a High nibble of a remainder count.
3 :	The flag is a Middle nibble of a remainder count.
4 :	The flag is a Low nibble of a remainder count.

The state is normally 0. It goes to 1 on the reception of INIT flag. The state is subsequently updated with reception of each nibble of a transfer count. The state returns to 0 after reception of the last nibble. The state diagram for the RC_FLG_FSM is shown in Figure 23 on page 86.

After receiving the HOLD flag, the MCU checks whether the card is ready to receive data by checking the receiver's buffer. If the buffer is empty, it sets up the transmitter's acknowledgement state (TR_ACK_STT) to send REC-READY ACK, otherwise it does nothing. The process will then be repeated when the next HOLD flag arrives.

After receiving the START flag, the MCU enables the receiver's address counters to start the data reception. It also resets the TR_ACK_STT to 0.

An FSM is not needed for the interpretation of an received ACK character, since it solely consists of the control characters. That is, an ACK nibble does not carry any data which can imitate the control character. Therefore, the ACK characters are processed by the value itself.

If the ACK character is WAIT then the MCU turns off the watchdog timer. If it is REC-READY then the MCU updates the TR_FLG_STT to 7 to send a START flag.

4.2.4 TC_SER

This ISR services the interrupt TRINT* from the the transmitter's address counter. It manages data in the transmitter's buffer. To carry out this task it uses two variables: viz. TRBLKS_MTY and TRBLKS_SENT. The variable TRBLKS_MTY contains the number of empty blocks of size 1K word in the transmitter's buffer and TRBLKS_SENT equals the number of 1K word blocks transmitted until then.

The ISR first updates these variables by incrementing them by 1. It then compares TRBLKS_SENT with the total number of blocks to be transmitted (TRBLKS + 1) and stops the transmission if they are equal. It also notifies the host about completion of the transmission. If TRBLKS_SENT and (TRBLKS+1) are not equal, the routine

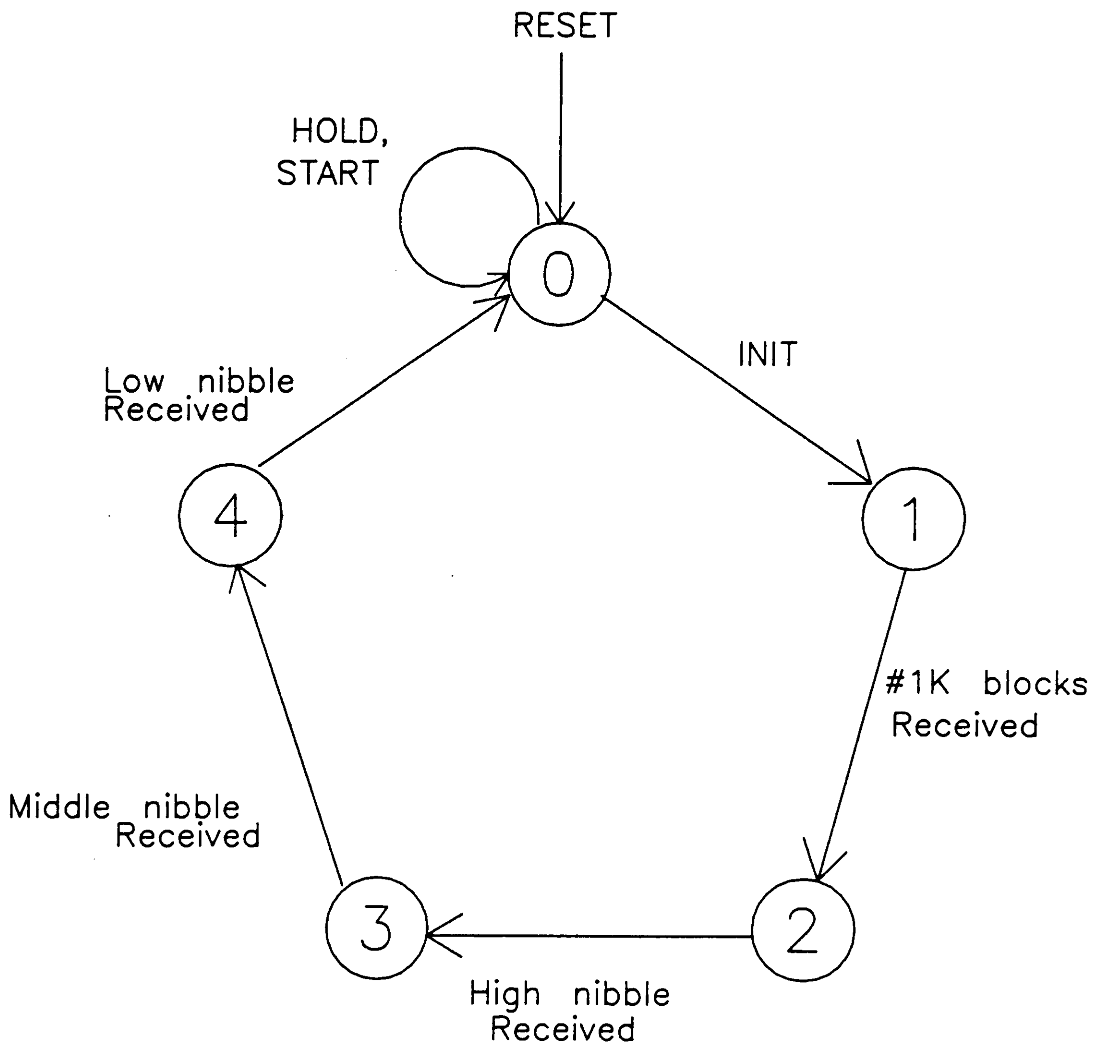


Figure 23. State diagram for the interpretation of a Flag character

checks whether there are any more data blocks to be transferred to the local buffer from the main memory. If there are, the routine programs the DMAC for a data transfer provided that the DMAC is available for the transfer. If the DMAC is currently active, the routine simply quits leaving the task of transfer to the DMAC's ISR.

4.2.5 RC_SER

This ISR services the interrupt RCINT* from the receiver's address counter. It manages the receiver's buffer memory. It keeps two variables to keep track of data traffic in the receiver's buffer. They are RCBLKS_FULL and RCBLKS_2_REC. The variable RCBLKS_FULL specifies the number of currently occupied 1K word data blocks in the receive's buffer and RCBLKS_2_REC initially contains the number of 1K word of data blocks to receive. With every interrupt, RCBLKS_2_REC is decremented by 1 since a block is received and RCBLKS_FULL is incremented by 1 since the buffer gets a block. The varable RCBLKS_FULL is subsequently decremented by the DMAC's ISR when it transfers a data block from the buffer to the main memory.

Since the buffer size is 4K words, the buffer can hold at the most four data blocks of size 1K word. Therefore, this ISR aborts the data reception process if RCBLKS_FULL becomes 4 signifying an overflowing of the buffer.

The RC_SER ISR stops the data reception when RCBLKS_2_REC decrements to 0. The ISR also starts a transfer of data from the buffer to the main memory by programming the DMAC provided that the DMAC is not currently active. Otherwise it quits leaving the job of transfer to the DMAC's ISR.

4.2.6 DMA_SER

This ISR services two types of interrupts from the DMAC. One is the interrupt on completion of the receiver's buffer to the main memory data transfer (Device to Memory: DEV2MEM) and the other on completion of the main memory to the transmitter's buffer data transfer (Memory to Device: MEM2DEV). Both these interrupts are discussed below.

DEV2MEM: The ISR first decrements the variables RCBLKS_FULL and RCBLKS_2_XFER since 1K word of data is transferred to the main memory. If RCBLKS_2_XFER reduces to 0, the routine notifies the host about reception of a block of data. It also passes the transfer count through the mailbox registers.

Then the routine checks for any pending memory to device transfers. The pending transfer can either be the 'First Transfer' or a normal transfer. The First Transfer is a transfer of the first block of data from the main memory to the transmitter's local buffer. It is different from the normal transfer with respect to a transfer count. Two Kilowords of data are transferred in the first transfer as opposed to 1K word in the normal transfer. If any of these transfers is pending, then the routine programs the DMAC accordingly and starts it. If none of the transfers is pending, the routine checks for any pending device to memory transfers and programs the DMAC for that transfer if a request is present.

MEM2DEV: The routine first updates the variables TRBLKS_MTY and TRBLKS_2_XFER according to the type of transfer that had taken place, that is, First Transfer or the normal transfer. For the first case the variables are decremented by two while for the later one they are decremented by one, since 2K words of data are transferred in the first transfer while 1K word is transferred in all the other transfers. The routine initiates the Data Link protocol on completion of the First Transfer by setting TR_FLG_STT to 01.

The routine then checks for any pending device to memory transfers or any pending memory to device transfers in the respective order and starts the DMAC appropriately.

In both the types of interrupts discussed above, the routine aborts the process of transmission or reception if an error occurs in the corresponding DMA transfer and then notifies the host about it.

4.2.7 TIM0_SER

This routine services MCU's internal timer #0, the watchdog timer, interrupt. It aborts the process of transmission and notifies the host processor about the action.

This completes the discussion of all the interrupt service routines. The ISRs discussed above use many other subroutines. These miscellaneous subroutines are explained below.

4.3 *Miscellaneous Subroutines*

4.3.1 FUST_MOV

This subroutine is used by either INT0_SER or DMA_SER to make the First Transfer of a data block form the main memory to the buffer memory. The routine programs the DMAC channel #1 with a transfer count of 2K words if the length of data is greater than 2K, else with an actual length of data block if the length is less than 2K.

4.3.2 TBLK_XFER

This routine is used to program the DMAC channel for a memory to device transfer, that is, a transfer of data from the main memory to the tranmitter's buffer. It first sets the buffer memory address in the DMAC's register to the beginning of the buffer if the current address is that of the end of the buffer. This results in usage of the buffer in a circular manner. After the address correction, the routine programs the DMAC with a transfer count. The 'remainder count' is used if the transfer is the last transfer. If not, the count used is 1K.

4.3.3 RBLK_XFER

This routine is very similar to the routine TBLK_XFER, except that it is used for a transfer of data from the receiver's buffer memory to the main memory. The address correction and the count selection is done in the same manner as that of TBLK_XFER.

4.3.4 TR_ABORT

This routine is used to abort the process of data transmission. It initializes all the variables, flag bits and the hardware control signals which are used in the process of transmission. It also initializes the DMAC registers. If the process of abortion is initiated due to some error condition, there is a possibility of the DMAC channel being active in transferring data from the main memory to the transmitter's buffer. In this case, the routine first halts the DMAC and aborts the channel operation before initializing the registers.

4.3.5 RC_ABORT

This routine is used to abort the process of data reception. The routine is very similar to the routine TR_ABORT. It initializes all the variables, flag bits and the hardware control signals used in the process of data reception. It initializes the DMAC registers in the same manner as that of TR_ABORT.

4.3.6 YELALRM

The routine sets the YEL_ALRM bit and the corresponding bit in the GSR if yellow alarm is received in more than three consecutive samplings of the transceiver status. The yellow alarm signifies that the ICC card at opposite end of the transmission link has lost signal from the first card. This alarm can be used to recognize a fault on the transmission link. The routine aborts the process of transmission if yellow alarm is received while the transmission is in progress.

4.3.7 ERRCHEK

This routine sets the LOSS_SYN bit and the appropriate bit in the GSR if the 'loss of signal' condition persists for more than three consecutive samplings of the transceiver's status. It also sends yellow alarm to the other card to notify it about the loss of signal condition. It aborts the process of data reception if the signal is lost in the middle of data reception.

4.3.8 INT_MAIN

This routine programs the PT_VSI registers to generate an interrupt to the VMEbus. The message code for that interrupt is passed through the mailbox register #15. The message code specifies the purpose of the interrupt, such as 'Transmission Complete', 'Data Received', etc.

The Intel 8751 microcontroller has 4K bytes of EPROM and 128 bytes of RAM. The software is programmed into the EPROM and the RAM is used to store the variables and different bit flags. The program source code in the assembly language of 8051. It is attached in the Appendix B. It also contains the files which contain the the addresses for the 8751 registers and the registers of all the programmable chips. One file contains the declarations of all the bit and byte variables and constants used in the program.

Chapter V

USERS MANUAL

This document explains the method of using the Intelligent Communication Controller (ICC). The ICC is a double high Eurocard module for the VMEbus which is used for high speed reliable data transmission. The ICC transmits and receives serial data on the T1 link and supports full duplex operation. It has an on-board microcontroller, DMA Controller and 16K bytes of dual port fast static buffer RAM. The microcontroller has an on-chip EPROM loaded with the firmware to execute the Data Link protocol and to communicate with the host. The ICC is capable of functioning independently with a minimal support from the host.

5.1 Abbreviations

DMAC	Direct Memory Access Controller
DTB	Data Transfer Bus
ICC	Intelligent Communication Controller
MCU	Microcontroller Unit
RAM	Random Access Memory
VMEbus	Interfacing system for use in interconnecting data processing, data storage and peripheral control devices in a closely coupled configuration.

5.2 Features

- VMEbus compatible.
- Intelligent slave operation.
- On-board DMA controller.
- 16K bytes of on board dual port buffer RAM.
- Efficient host-ICC communication through dual port registers.
- Full Duplex operation.

5.3 Specifications

Description:	Intelligent Communication Controller
Configuration:	
DTB Slave:	A24, D8
DTB Master:	A24, D16
Interrupter:	7 level, software selectable
Interrupt vector:	Programmable
Bus Request Level:	Jumper selectable
Data Communication:	Full duplex, 4 wire
Data Formatting:	T1 (D4), 1.544 Mbits/s

5.4 Jumper Options

The ICC has a total of three jumper groups:

1) To select the bus request level.

2) To control DMA_HALT line.

3) To select the length of data transmission line.

The jumper groups are explained below.

1) The bus request level jumper is the part #X7 on the sheet #1. It is located in the bottom left corner of the sheet. Part (a) of Figure 24 on page 98 shows the jumper configuration. The bus requester is used by the DMAC when it needs to transfer data

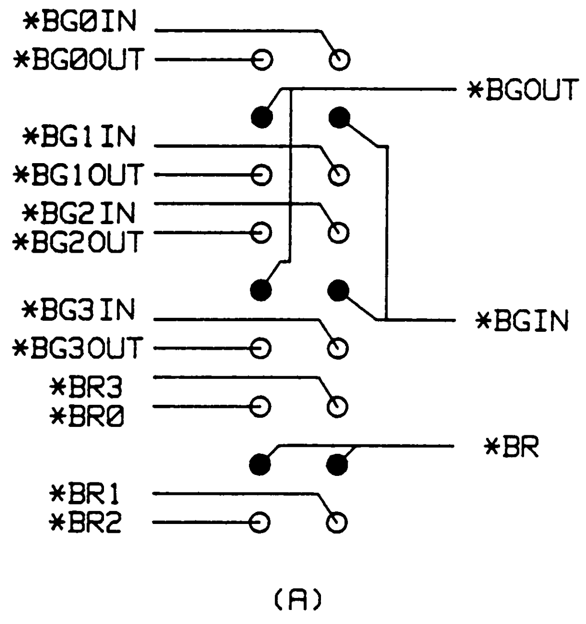
between the main memory and the on-board buffer RAM. Any one of the four request levels: viz. BR0, BR1, BR2 and BR3 can be selected. The pins shown with the filled circles are connected to the bus requester while those shown with the hollow circles are connected to the VMEbus. The desired level can be selected by placing the set of three jumpers vertically, shorting the corresponding BGxIN, BGxOUT and BRx lines. The BGxIN and BGxOUT of the levels which are not used should be shorted by placing horizontal jumpers.

The configuration to select level 1 is shown in the part (b) of the figure.

2) Jumper group 2 is used to halt the DMAC when any other board in the system places a request for the VMEbus. This jumper group is also located in the bottom left corner of sheet #1. Figure 25 on page 99 shows the the jumper group connections along with the required logic circuitry.

The four bus request signals are connected to one end of the jumpers and the other end of the jumpers is connected to a 4-input nand gate. When the jumpers are not placed, all the input lines of the nand gate are pulled high by the pull up resistors causing the output low and the signal DMA_HALT* (* indicates active low) high . When the jumpers are installed on any of these lines, the active low bus requests on those lines will make the output of the 4-input nand gate high. This will make DMA_HALT* low provided that the DMAC itself is not requesting the bus. The 2-input nand gate is used to avoid self locking, that is, to prevent the DMAC's own bus request from activating the DMA_HALT*.

By configuring these jumpers, one can select a priority of the ICC in the system. For example, if all the jumpers are installed, the DMAC will relinquish the VMEbus for a



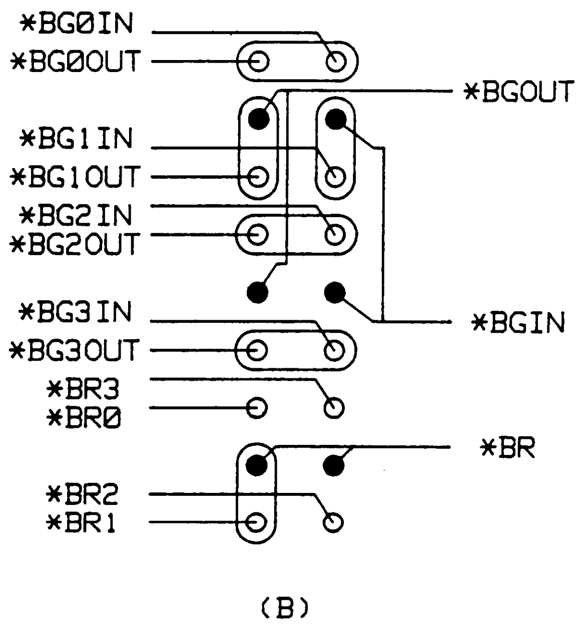


Figure 24. Jumpers for the Bus Request Level

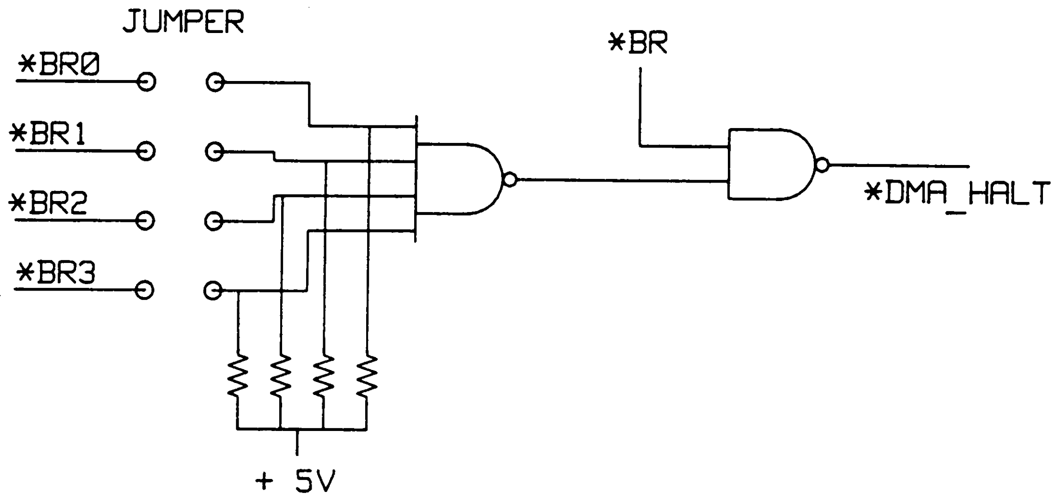


Figure 25. Jumpers to halt the DMAC operation

request on any level. If none of the jumpers is installed, the DMAC will not relinquish the bus until the task is finished. Installing the jumpers on the selected lines will cause the DMAC to give up the bus for the bus requests only on those lines. This is useful when any of the other boards needs immediate service from the host and can not afford to wait until the DMAC finishes its task.

3) The jumper group #3 is used to select the length of the transmission cable. It is the part #X9 on the right side of sheet #2.

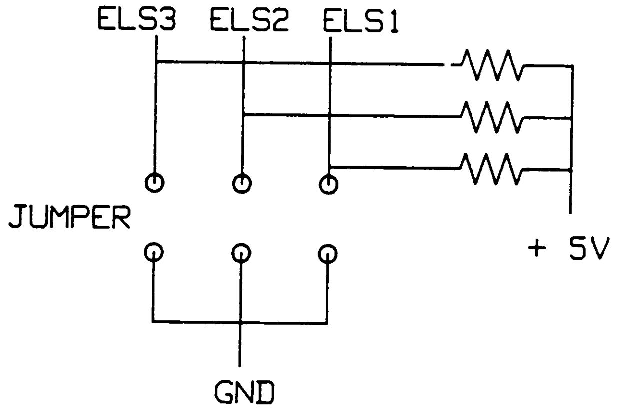
Part (a) of Figure 26 on page 101 shows the connections of the jumper group #3. The table in the part (b) gives the settings and the corresponding transmission line length. The settings which are shown are the valid ones and the rest are invalid.

The upper row of jumper pins is connected to the pins ELS1 to ELS3 of the line interface unit. These pins are pulled high by pull up resistors. The other row of jumper pins is grounded. Installing the jumper pulls the pin low. The jumper configuration and the corresponding length of line equalization is shown in part (b).

5.5 Programmable Options

This section explains the programmable options on the ICC. The options are listed below.

1. VMEbus interrupt level.



(A)

ELS3	ELS2	ELS1	CODING
J	J	J	0 -110 FT
J	J	N	110-220 FT
J	N	J	220-330 FT
J	N	N	330-440 FT
N	J	J	440-550 FT
N	J	N	550-660 FT

J = JUMPER
N = NO JUMPER

(B)

Figure 26. Jumpers to select the line equalization length

2. VMEbus interrupt vector .
3. VMEbus addresses to which the ICC responds in the slave mode.
4. Address Modifier (AM) code to which the ICC responds in the slave mode.
5. Address of the location in the main memory where the received block of data is stored.
6. AM code generated by the ICC in the master mode while accessing the main memory.
7. AM code generated by the ICC in the master mode while addressing the on-board RAM.

The first four options are selected through programming of the VME slave interface chip: PT-VSI. The last three options are controlled through programming of the DMAC registers. The on-board Intel 8751 MCU does the programming of these chips. The default settings can be changed by changing the appropriate parameters in the assembly language code for the 8751. The code should then be assembled and programmed into the on-chip EPROM of the 8751. Therefore, the 8751 assembler, EPROM eraser and the EPROM programmer are needed to make any changes.

The interrupt level, the interrupt vector and the main memory's address for the storage of received data can be dynamically changed by the application program running on the host by passing new parameters to the ICC along with an appropriate command. This is possible, because the MCU uses the values in the variables stored in its internal RAM to program the corresponding registers in the corresponding chips. The application program can pass a new value to the MCU through the mailbox registers and give a command to update the required variables. The command codes and their meaning is as follows:

CODE	MEANING
01	Change the Interrupt Level.
02	Change the Interrupt Vector.
04	Change the main memory address for storage of the received data.
08	Start Transmission

The Appendix D gives a sample code of the programs that can be used by the Motorola 68020 host to change the interrupt level, interrupt vector and the address of the main memory to store the received block of data.

The possible parameter values and the default values for the above options are given below.

1) Slave address: Address bits A9 to A23 can be programmed for a match. The default value of the address to which the card responds is 9000XXH.

2) Interrupt level: Any one of the seven. The default is level 4.

3) Interrupt vector: Any 8-bit value. The default value is 64.

4) Slave address modifier (AM): Any value permitted by the host processor card. Refer to the host card's Users Manual for the possible options. The default value is 39H with the bit #2 set to 'don't care' condition. Therefore, the ICC responds to the codes 39H and 3DH which correspond to Standard Non-Privileged Data Access and the Standard Supervisory Data Access respectively.

5) Received Block's Address: Any 23 bit address in the range of the main memory. The default address is 030000H.

6) Master AM code for VMEbus: Two values: viz. 3DH and 39H are possible which correspond to standard supervisory data access and standard non-privileged data access. The default value is 39H.

7) Master AM code for the on-board RAM: Two values: viz. 1DH and 19H are possible. These lie in the 'User Defined' address modifier codes range of the VMEbus specifications. The default value is 1DH. Care should be taken that none of the other cards on the VMEbus respond to this address modifier code.

The ICC acknowledges the execution of the command given to it by the host by interrupting the host. An error occurred during the process of transmission or the reception is also notified to the host through an interrupt. The host can interpret the purpose of the interrupt from the ICC by reading its mailbox register MB01. The value in MB01 specifies the purpose of the interrupt. The possible values and their meaning is given below.

VALUE	MEANING
0	Transmission Complete
1	Data Received
2	Request for transmission is not accepted
3	Interrupt Level changed
4	Interrupt Vector changed
5	Storage address for the received data changed
6	Transmission aborted due to DMA transfer error
7	Data Reception aborted due to DMA transfer error
8	Transmission aborted due to watchdog timer timeout
9	Transmission aborted due to reception of the Yellow Alarm

5.6 Using The Card

The card is a 'Plug-In' type card for the VMEbus. The selection of the jumpers should be done before plugging the card into the bus. Refer to the System Master's Users Manual for the selection of the 'Bus request level' selection jumpers and the jumpers for the automatic halting of the DMAC on any of the external board's request for the bus. From the System Master's Users Manual, find the request levels that are arbitrated by the bus arbiter on that card. Select any of these levels for the ICC card. Also find the level at which the host processor makes a request for the VMEbus. The jumper for that level must be put. Failure to do so might result in the failure of the host processor to service the interrupts while the DMAC is the bus master. Put other jumpers in this groups as per the necessity. Select the jumpers in the jumper group #3 as per the length of the transmission cable.

Switch off the power and plug the ICC card into the system. After plugging the card in the system attach the two cables: one for the data transmission and the other for the data reception to the corresponding ports on the card. Then power up the system and reset it. If the default values of the interrupt vector, interrupt level, etc. need to be changed, run the required routines on the host. These routines can also be embedded in the application program. The card is then ready for the usage.

The prototype card does not have any jumpers. The jumpers could not be provided due to insufficient space. Therefore, the corresponding jumper connections have been wired. The settings of the jumpers for the prototype card are as follows.

- The bus requester is connected to the level 3 of the arbitration bus.

- All the 4 bus request levels have been connected in the automatic halt mechanism for the DMAC.
- The transmission wire length of up to 110 feet has been selected.

Software routines are required to give a command for transmission and to service the card's interrupt. A 'Send' routine can be used by the application program to start the data transmission. The interrupt service routine can be written to read the message code associated with the interrupt and to take the appropriate action on it.

The sample codes of these routines along with a simple application program are attached in the Appendix D. The 'Send' routine checks the status register of the ICC for any error condition and for the availability of the ICC. If the ICC is not currently engaged in the transmission and if there is no error then the 'Send' routine makes a request for the transmission. The 'Send' routine is used by the application program to transmit data. The sample interrupt service routine in the Appendix D simply prints a message on the screen of the terminal attached to the host processor's card about the nature of an interrupt.

These routines are given as samples. More sophisticated routines can be written for a particular application program.

Chapter VI

Conclusion

The purpose of this thesis was to design an Intelligent Communication Controller for the VMEbus. The ICC card will be used in a Protective Relay application for a power system.

The design process was started with the aim of achieving the specified functional requirements of the ICC card. The first step in the design process was to define the working of the ICC on a functional level. The protocols governing the communication of the ICC with the host processor and with another ICC were defined. Next, the functional modules needed to carry out these protocols were defined. Interfacing of these modules to each other resulted in the architectural definition of the ICC. Once the architecture was finalized, proper chips were selected and the circuit was designed. Two wirewrap prototype boards were prepared. Special test routines were used to test the hardware. Next, the software was written for the Application layer protocol and the Data Link protocol to make the ICC fully functional.

6.1 Performance Estimation

Performance estimation can be categorized into two parts:

1. Amount of bus bandwidth used by the ICC and
2. Time required for the transmission of data

6.1.1 Bus Usage

The ICC uses the VMEbus for the sole purpose of transferring data to and from the main memory to the local buffer memory. The data transfer is done by the on-board DMAC.

The DMAC transfers data at a rate of 1 word per 1.6 μs. The maximum usage of the VMEbus is done at the time of simultaneous data transmission and reception (full duplex operation). Since the rate of transmission and reception is 192,000 bytes/s, data transferred on the bus per second is 384,000 bytes/s. Therefore the bus bandwidth used by the ICC is

$192000 \times 1.6 \times 10^{-6} \times 100 = 30.72$

6.1.2 Time for Transmission

The time required for the transmission of data is calculated from the time when the 'Send' command is given by the host to the ICC, to the time when the data reaches the host at the other end of a communication link.

Time required for the data transmission =

Time required to transfer the first block of data from the main memory to the buffer memory

+ Time required to set up a logical connection

+ Time required to send a data

+ Time required to transfer the last block (2K or less) of data from the receiver's buffer memory to the main memory

The equation can be understood with the help of an example. The time required to transmit a block of 5K bytes is explained in Figure 27 on page 110. The four parts of the equation are shown in this figure. The first part consists of the time required to move 4K bytes of data from the main memory to the buffer RAM. It can be seen that the time for the transfer of the rest of the data overlaps with the second part, which is the time required to set up a logical link. The Setup time is calculated with the following equation:

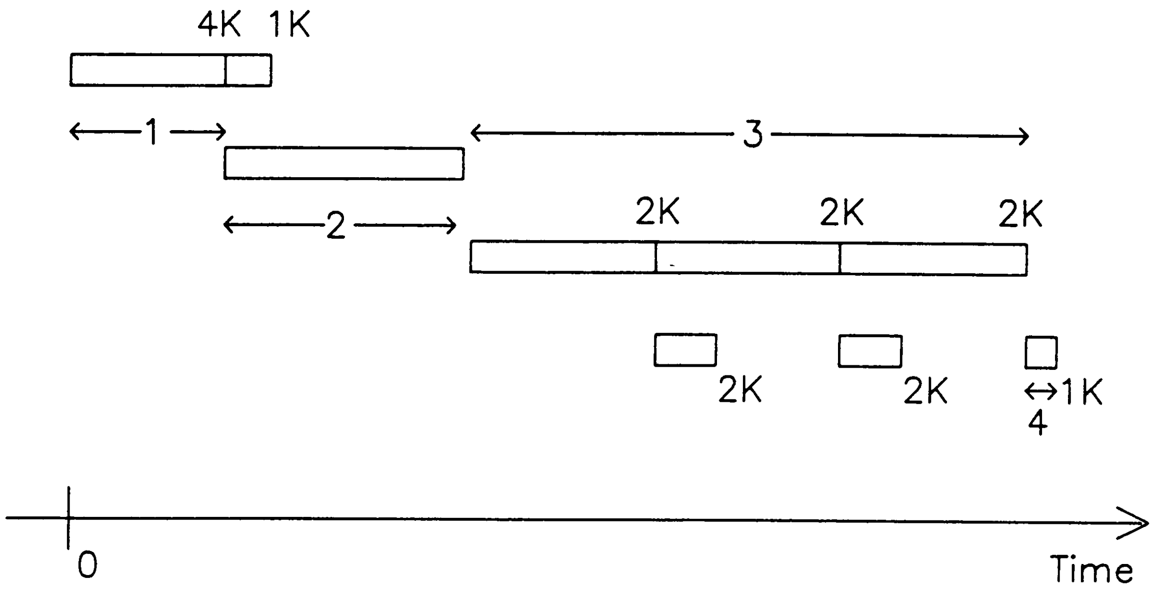
Setup Time =

Time required to transmit INIT and Transfer Count Flags by the transmitter

+ Time required to transmit REC-READY by the receiver upon reception of the above flags

+ Transit Time for Flags and ACKs in both directions

Transit time in this equation is the time required to travel the distance between the two systems. The speed of data travel is taken as 0.6C where C is the speed of light.



1. DMA transfer from main memory to transmitter buffer
2. Setup time
3. Time for data transmission
4. Time for transferring last 2K of data from receiver buffer to main memory

Figure 27. Time required for the transmission of data

The Setup time is constant for constant distance and is independent of the length of a data block. The third part of the main equation consists of the time required for data transmission. This time depends solely upon the length of data. Since the receiver recognizes the incoming data on a 2K byte boundary, 6K bytes need to be transmitted even though the data length is 5K bytes. The fourth part of the total time of transmission is the time required to transfer last 1K bytes of data from the receiver's buffer to the main memory. The transfer of first 4K bytes overlaps with the third part as shown in the figure. Table 1 shows the transmission time for data lengths of 0.5K, 2K and 5K for distances of 100 Km., 200 Km. and 500 Km. The unit of transmission time is a millisecond.

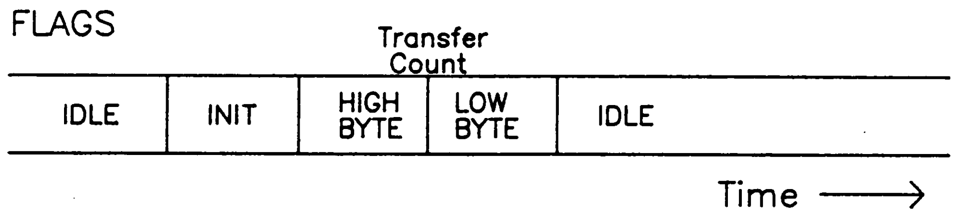
DATA LENGTH	DISTANCE (Km.)		
	100	200	500
0.5K	17.67	19.32	24.33
2K	20.13	21.78	26.79
5K	43.11	44.76	49.77

Table 1. Time required for data transmission

6.2 *Future Modifications*

6.2.1 Synchronous Protocol

In the last section we have calculated the time required for data transmission using an asynchronous protocol. The protocol is called asynchronous because, the process of 'handshaking' takes place between the transmitter and the receiver through an exchange of Flags and ACKs, prior to the transmission of data. The handshaking takes substantial time. This overhead of handshaking may not be acceptable in some applications. The time required to establish a logical connection can be reduced by using a synchronous protocol instead of an asynchronous one. In the synchronous protocol, it is assumed that the receiver is always ready to accept data and hence the handshaking is not necessary. Since the ACKs from the receiver are not needed, only Flag characters are used in a synchronous protocol. The synchronous protocol is shown below:



In a synchronous protocol, only INIT and Transfer Count Flags are needed. The data can start right after the low byte of a transfer count. The INIT and Transfer Count Flags occupy only one multiframe. With such a small overhead, the synchronous protocol is much faster than the asynchronous protocol.

Another way to reduce the transmission time is to bring down the boundary at which the transmitter and the receiver count the outgoing or the incoming data. The present boundary of 2K bytes can be reduced to 0.5K bytes. This will result in a reduced overhead of extra characters which need to be appended at the end of a block of data to make the total length a multiple of 2K. For example, for the transmission of 2.25K bytes of data, 4K bytes will need to be transmitted for a 2K byte boundary scheme while only 2.5K bytes will need to be transmitted with a boundary of 0.5K. This modification saves time in the third part of the equation for the total transmission time. Table 2 shows the time required for data transmission using a synchronous protocol and the boundary set at 0.5K bytes. The time is in milliseconds.

DATA LENGTH	DISTANCE (Km.)		
	100	200	500
0.5K	5.55	6.10	7.77
2K	13.96	14.51	16.18
5K	29.96	30.51	32.18

Table 2. Time required for data transmission using a synchronous protocol

6.2.2 Error Detection and Correction

Transmission errors can always occur regardless of the design of a communication system. Therefore, an error detection and correction mechanism is mandatory for a data communication system.

Error detection and correction can be accomplished either in a hardware or in a software. The ICC does not have an error detection scheme built in the hardware. Therefore, the application program needs to take care of the error detection. Error detection can be done in a software by calculating a checksum of the data and appending it at the end. The host processor at the receiver's end can recalculate the checksum and compare it with the received checksum. If the checksums differ, the processor can reject the data and send a message back to the first system to re-transmit the data. Such a scheme is practical only for a small amount of data. As the data size increases, the software overhead for calculating the checksum increases. Also, the error recovery is very slow since it requires the re-transmission of the whole block of data. The error recovery schemes implemented in a hardware are a lot faster and efficient compared to those implemented in software.

The error recovery scheme could not be included in the ICC due to the complexity of the circuit involved. It is impossible to build such a circuit using discrete logic gates and fit it into available space on the VMEbus card. These circuits are generally available in form of a VLSI chip. Today, a new chip from Rockwell is available to carry out error detection and correction for the data transmission on a T1 link. The chip, R8071, interfaces directly to chips R8069 and R8070. The chip implements the HDLC (High level Data Link Control) data link protocol. The HDLC protocol has an error detection and recovery as a part of its standard. The chip can be used on the ICC card with some changes in the hardware and the software. The R8070 manual [18] gives details about usage of the chip. The R8070 will improve the ICC's reliability and efficiency tremendously.

BIBLIOGRAPHY

1. T. Eklund, "A Study of a Fiber Optic Communication System used for Protective Relaying ", Masters Thesis.

2. Rockwell, "R8070, T1-CEPT PCM Tranceiver Data Sheet ", Rev. 1, June 1986, Document number : 29300N14.

3. W. Stallings, "Data and Computer Communications", Macmillan Publishing Company, New York, 1985.

4. T. Balph "VMEbus - A Microprocessor Bus for the future", *Digital Design* , August 1982.

5. W. Fischer, "IEEE P1014 - A Standard for the High Performance VMEbus", *IEEE Micro*, February 1985.

6. A. Tanenbaum, "Computer Networks", *EDN*, June 14, 1984.

7. The VMEbus Specifications Manual, revision C.1, October 1985, Motorola Microsystems.

8. Microcontroller Handbook, 1983, Intel Corp.

9. VMEbus Slave Interface (VSI), 1987, Performance Technologies Inc.

10. MC68440 Dual-Channel Direct Memory Access Controller: Advace Information, February 1984, Motorola Semiconductors.

11. High Performance CMOS: Data Book, 1988, Integrated Device Technologies

12. R8069 Line Interface Unit: Data Sheet: Document No. 29300N36, September 1987, Rockwell.

13. Microprocessor and Peripheral Handbook, Volume 2, 1988, Intel Corp.

14. The TTL Data Book, Volume 2, 1985, Texas Instruments.

15. The TTL Data Book, Volume 3, Texas Instruments, 1984.

16. Information Bulletin: Preliminary Product Information, 1988, Pulse Engineering, Inc.

17. PC-LOGS User's Manual, Personal CAD Systems, Inc.

18. R8071 ISDN/DMI Link Layer Controller: Data Sheet: Document No. 29300N18, August 1987.

Appendix A

Sample Test Programs

Many software routines were written to test the hardware design. Two of these routines are given in this appendix.

```
****************************************************************************

* This is a test program for the 8051 microcontroller and the 8255.
* It initializes the registers inside the MC and the bits of the output port.
* The system is reset and the 8255 is programmed. Port B of 8255 is read and
* output to Port A. Port bits P1.6 and P1.7 of 8751 generate a square wave.
* Port c of 8255 is set to 00h and then the bits 4 and 5 are set to 1 to
* test the bit level function of port C.
*
* Chips placed : Microcontroller, 8255, Address latch 74ls373 and data
*   buffers 74ls245 for the MC, Chip select decoder 74ls138
*   OR gates 74ls32.
* Input to port B can be given through R8069 and R8070 socket pins which
* connect to port B. The pins can be pulled high through a resistor or
* can be grounded. Output at port A can be checked with a voltmeter.
* Reading ang writting to the ports can also be observed on logic analyser.
* Square wave on bits P1.6 and P1.7 can be observed on an oscilloscope.

****************************************************************************

#include "I8051.WRD"
#include "ADDRSS.WRD"

N_HALT    .equ  P1.0            port P1 bit declarations
RESET     .equ  P1.1
RECTMR    .equ  P1.2
TRTMR     .equ  P1.3
N_RCSTRT  .equ  P1.4
N_TCSTRT  .equ  P1.5
N_REQ0    .equ  P1.6
N_REQ1    .equ  P1.7

TM0_RUN   .equ  TCON.4          declaration of the timer
TM1_RUN   .equ  TCON.5          control bits.


   .ORG  0000H

   LJMP  START


START   .ORG  0100H

********* INITIALIZATION SECTION *********


   MOV   IE,#00H                Disable All Interrups
   MOV   SP,#220D               Init Stack Pointer
   MOV   PSW,#00H               Clear Flags, Select Reg. Bank 0
```

```
    MOV    P1,#0F1H                     Init P1 Bits
    MOV    TCON,#00H                    Set Interrups Level Triggerd
                                        Clear Timer Overflow Flags
    MOV    TMOD,#11H                    Timer Mode, No Gate Control
    MOV    SCON,#00H                    Disable Serial Reception
    MOV    IP,#02H                      Int. Priority = TIM_0 : INT0 : INT1

* RESET THE SYSTEM

    SETB   RESET
    MOV    R1,#10H
    DJNZ   R1,$
    CLR    RESET

* PROGRAM THE 8255

    MOV    DPTR,#PORT_MODE
    MOV    A,#82H                       Port A,C = O/P , B = I/P
    MOVX   @DPTR,A

    MOV    DPTR,#PORT_C
    MOV    A,00H
    MOV    @DPTR,A                      Init Port C to 00H

    MOV    DPTR,#PORT_MODE
    MOV    A,#09H
    MOVX   @DPTR,A                      Port C.4=1
    ORL    A,02H
    MOVX   @DPTR,A                      Port C.5=1

    MOV    DPTR,#PORT_A

LOOP   INC  DPTR
    MOVX   A,@DPTR                      Read Port B
    DEC    DPTR
    MOVX   @DPTR,A                      Write to Port A
    CPL    C
    MOV    N_REQ1,C                     Square Wave
    MOV    N_REQ0,C
    SJMP   LOOP                         Loop Infinitely

    .END
```

```
*************************************************************************

*    Testing the 8253 Timers with Transceiver and LIU.
*    Testing the insertion and deletion of flag characters.

* This is a test program for the 8253 timer synchronization to the
* transceiver signals. The timer for the transmitter section is started
* right away after initialization of 8253 while that for the reception
* is started on the receiver synchronization to the incomming signal.
*
* Interrupt 1 is always kept enabled, since it is used for single
* stepping. Using the circuitry for single stepping, single step
* through the program. Observe the write cycles on logic analyser.
* After the timer is programmed, observe the timing pulses on the
* Logic Analyser along with the other transceiver signals.
* Check their relationship to verify the design.

*************************************************************************

#include "I8051.WRD"
#include "ADDRSS.WRD"

N_HALT    .equ  P1.0
RESET     .equ  P1.1
RECTMR    .equ  P1.2
TRTMR     .equ  P1.3
N_RCSTRT  .equ  P1.4
N_TCSTRT  .equ  P1.5
N_REQ0    .equ  P1.6
N_REQ1    .equ  P1.7

TM0_RUN   .equ  TCON.4
TM1_RUN   .equ  TCON.6


   .ORG  0000H

   LJMP  START


START  .ORG  0050H

********* INITIALIZATION SECTION *********


   MOV  IE,#00000000B           Disable All Interrups
   MOV  SP,#15D                 Init Stack Pointer
   MOV  PSW,#00000000B          Clear Flags, Select Reg. Bank 0
   MOV  P1,#0F1H                Init P1 Bits
                                All bit functions deactivated
```

```
    MOV   TCON,#00000000B                Set Interrupts Level Triggerd
                                         Clear Timer Overflow Flags
    MOV   TMOD,#00100010B                Timer Mode 2, No Gate Control
    MOV   SCON,#00H                      Disable Serial Reception
    MOV   IP,#00H

* RESET THE SYSTEM

    SETB  RESET
    MOV   R1,#10H                        Provide enough time for RESET
    DJNZ  R1,$
    CLR   RESET

    MOV   IE,#10000100B                  Enable Interrupt 1 (Single Stepping)

******** PROGRAM THE 8255 *****************

    MOV   DPTR,#PORT_MODE
    MOV   A,#92H                         PORT A,B : INPUT  C: OUTPUT
    MOVX  @DPTR,A                        MODE 0


******** PROGRAM THE 8253 TIMERS **********

* Set up Modes

    MOV   A,#34H                         TIMER 0 : RATE GENERATOR
    MOV   DPTR,#TIM_MOD
    MOVX  @DPTR,A                        TIMER 2 : RATE GENERATOR
    MOV   A,#0B4H
    MOVX  @DPTR,A

* Load Count Values

    MOV   DPTR,#TIMER_0                  COUNT OF 0096D IN BOTH TIMERS
    MOV   A,#96D
    MOVX  @DPTR,A
    MOV   A,#00H
    MOVX  @DPTR,A

    MOV   DPTR,#TIMER_2
    MOV   A,#96D
    MOVX  @DPTR,A
    MOV   A,#00H
    MOVX  @DPTR,A

    SETB  TRTMR                          Start transmitter timer
    MOV  DPTR,#PORT_B

CHECK   MOVX  A,@DPTR                    Check If Receiver is Synchronized
```

```
    ANL    A,#01110000B
    CJNE   A,#00110000B,CHECK           if not, go back and check again
    SETB   RECTMR                       if yes, start receiver timer

    MOV    A,#33H                       Fill flag/ack Transmit buf with 33h
    MOV    DPTR,#FLG_ACK
    MOVX   @DPTR,A

READ    MOV   IE,#00H                   Disable Interrupts

    MOV    DPTR,#FLG_ACK
    MOVX   A,@DPTR                      read receiver flag/ack buf.
    MOV    DPTR,#PORT_C
    MOVX   @DPTR,A                      output it to port c
    MOV    R2,#20D                      Wait for some time
    DJNZ   R2,$
    MOV    IE,#10000100B                Enable Interrupt 1 (Single Stepping)
    SJMP   READ                         keep doing it infinitely.

IDLE    MOV   PCON,#01H                  Go Idle


********* INT 1 Interrupt Service Routine *********

    .org   0013H

    LJMP   INT1_SER

INT1_SER  .org  0300H

AGAIN1    JNB  N_REQ0,$                 WAIT HERE UNTIL REQ0 GOES HIGH
    MOV    R2,#0FFH                     AVOID FALSE SIGNAL
    DJNZ   R2,$                         DELAY OF APPRX. 1.2 mSec.
    JNB    N_REQ0,AGAIN1

    SETB   N_REQ1                       SET FLAG

AGAIN2    JB  N_REQ0,$                  NOW WAIT HERE TILL IT GOES LOW
    MOV    R2,#0FFH
    DJNZ   R2,$
    JB     N_REQ0,AGAIN2

    CLR    N_REQ1                       Clear Flag
    RETI                                GO BACK AND EXECUTE ONE INSTR.


    .END
```

Appendix B

Firmware

This file contains the software required to communicate with the host and to implement the Data Link protocol. The software is written in an assembly language and runs on the Intel 8751 microcontroller on the ICC. It is resident in the on-chip EPROM of the microcontroller.

```
***************************************************************************

* This file contains the 8751 port and special function registers'
* addresses. Bit addresses of the ports and registers are also definied.

***************************************************************************

      ACC   .equ 0E0h                     Accumulator
      B     .equ 0F0h                     B Register
      PSW   .equ 0D0h                     Program Status Word
      SP    .equ 081h                     Stack Pointer
      DPL   .equ 082h                     Data Pointer Low
      DPH   .equ 083h                     Data Pointer high
      DPTR  .equ 082h

      P0    .equ 080h                     Port 0
      P1    .equ 090h                     Port 1
      P2    .equ 0A0h                     Port 2
      P3    .equ 0B0h                     POrt 3

      TMOD  .equ 089h                     Timer Mode Control
      TCON  .equ 088h                     Timer Control Register
      TH0   .equ 08Ch                     Timer 0 Word
      TL0   .equ 08Ah
      TH1   .equ 08Dh                     Timer 1 Word
      TL1   .equ 08Bh

      IP    .equ 0B8h                     Interrupt Priority
      IE    .equ 0A8h                     Interrupt Enable

      SCON  .equ 098h                     Serial Port Control
      PCON  .equ 097h                     Power Control

*   Port Bit Addresses

      P1.0 .equ 090h                      Port 1 Bit 0
      P1.1 .equ 091h
      P1.2 .equ 092h
      P1.3 .equ 093h
      P1.4 .equ 094h
      P1.5 .equ 095h
      P1.6 .equ 096h
      P1.7 .equ 097h

      TCON.4 .equ 08Ch
      TCON.6 .equ 08Eh

      ACC.0 .equ 0E0H                     Accumulator bits
      ACC.1 .equ 0E1H
      ACC.2 .equ 0E2H
ACC.3 .equ 0E3H
ACC.4 .equ 0E4H
ACC.5 .equ 0E5H
ACC.6 .equ 0E6H
ACC.7 .equ 0E7H
```

```
************************************************************************

* This file contains addresses of the various programmable registers,
* ports and timers in the on-board chips. Some buffer addresses are
* also included.

************************************************************************

********** PTVSI REGISTERS **************
*
       MB00 .equ 0F101H                Mailbox Register # 0
       MB01 .equ 0F103H
       MB02 .equ 0F105H
       MB03 .equ 0F107H
       MB04 .equ 0F109H
       MB05 .equ 0F10BH
       MB06 .equ 0F10DH
       MB07 .equ 0F10FH
       MB08 .equ 0F111H
       MB09 .equ 0F113H
       MB10 .equ 0F115H
       MB11 .equ 0F117H
       MB12 .equ 0F119H
       MB13 .equ 0F11BH
       MB14 .equ 0F11DH
       MB15 .equ 0F11FH                Mailbox Register # 15

       INT_MASK_1 .equ 0F121H          Mailbox Interrupt Enable (MB07-MB00)
       INT_MASK_2 .equ 0F123H          Mailbox Interrupt Enable (MB015-MB07)
       INT_PEND_1 .equ 0F121H          Mailbox Interrupt Pending(MB07-MB00)
       INT_PEND_2 .equ 0F123H          Mailbox Interrupt Pending(MB015-MB08)

       MB_INT_SEL1 .equ 0F125H         ( LIRQ# ) Pin Select ( MB03-MB00 )
       MB_INT_SEL2 .equ 0F127H                 ( MB07-MB04 )
       MB_INT_SEL3 .equ 0F129H                 ( MB11-MB08 )
       MB_INT_SEL4 .equ 0F12BH                 ( MB15-MB12 )

       AM_COMP     .equ 0F12DH         Address Modifier Compare
       AM_X        .equ 0F12FH         Address Modifier Don't Care
       ADDR_COMP_1 .equ 0F131H         Address Compare ( A31-A24 )
       ADDR_COMP_2 .equ 0F133H            ( A23-A16 )
       ADDR_COMP_3 .equ 0F135H            ( A15-A09 )

       INT_REQ_LEV .equ 0F137H         VMEbus Interrupt Request Level
       INT_VEC_NUM .equ 0F139H         Interrupt Vector Number

       GLB_INT_STS .equ 0F13DH         Global Interrupt Status
       MA_ENBL     .equ 0F13DH         Master Enable Register
       VSI_MOD_SEL .equ 0F13FH         Mode Selection Register
```

```
* ********** MC68440 DMA CONTROLLER REGISTERS **********

        CSR_0     .equ    0F000H          Chanell Status Register : Channel 0
        CSR_1     .equ    0F040H                 : Channel 1

        CER_0     .equ    0F001H          Channel Error Register
        CER_1     .equ    0F041H

        DCR_0     .equ    0F004H          Device Control Register
        DCR_1     .equ    0F044H

        OCR_0     .equ    0F005H          Operation Control Register
        OCR_1     .equ    0F045H

        SCR_0     .equ    0F006H          Sequence Control Register
        SCR_1     .equ    0F046H

        CCR_0     .equ    0F007H          Channel Control Register
        CCR_1     .equ    0F047H

        MTCR_L_0   .equ    0F00BH         Memory Transfer Cnt. Reg. Low Byte : CH.0
        MTCR_H_0   .equ    0F00AH                  High Byte
        MTCR_L_1   .equ    0F04BH             : CH.1
        MTCR_H_1   .equ    0F04AH

        MAR_BA_0   .equ    0F00FH         Mem. Addr. Reg. : Base Address : CH.0
                                          Total 4 bytes ( Long word ) BA-1,-2,-3
        MAR_BA_1   .equ    0F04FH                : CH.1

        DAR_BA_0   .equ    0F017H         Dev. Addr. Reg. : Base Address : CH.0
                                          Total 4 bytes ( Long word ) BA-1,-2,-3
        DAR_BA_1   .equ    0F057H                : CH.1

        NIVR_0    .equ    0F025H          Normal Interrupt Vector Register CH.0
        NIVR_1    .equ    0F065H

        EIVR_0    .equ    0F027H          Error Interrupt Vector Register
        EIVR_1    .equ    0F067H

        MFCR_0    .equ    0F029H          Memory Function Code Register
        MFCR_1    .equ    0F069H

        DFCR_0    .equ    0F031H          Device Function Code Register
        DFCR_1    .equ    0F071H

        CPR_0     .equ    0F02DH          Channel Priority Register
        CPR_1     .equ    0F06DH
        GCR       .equ    0F0FFH          General Control Register

*********** INTEL 8253 TIMER REGISTERS ************

        TIMER_0    .equ    0F300H          Timer # 0 Address
        TIMER_1    .equ    0F301H          Timer # 1
        TIMER_2    .equ    0F302H          Timer # 2
        TIM_MOD    .equ    0F303H          Timer Mode Register

* ****** INTEL 8255 PROGRAMMABLE PERIPHERAL INTERFACE PORTS *****

        PORT_A    .equ    0F200H           Port A Address
        PORT_B    .equ    0F201H           Port B
        PORT_C    .equ    0F202H           Port C
        PORT_MODE .equ     0F203H          Mode Control Register

******** MISCLLENIOUS ADDRESSES **********

        FLG_ACK    .equ    0F400H          Flag / Acknowledge Character Buffer

        INTVEC     .equ    0F500H          DMA Interrupt Vector Read
```

```
**************************************************************************

*This file contains various Constants and Bit & Byte Variable decalrations.
*All the variables reside in the internal data ram of 8751.

**************************************************************************

********** Bit Constants ***************

        N_HALT    .equ   P1.0          DMA halt
        RESET     .equ   P1.1          Reset System
        RCTMR     .equ   P1.2          Receiver Timer Control
        TRTMR     .equ   P1.3          Transmitter Timer Control
        N_RCSTRT  .equ   P1.4          Start Receiver Counter
        N_TRSTRT  .equ   P1.5          Start Transmitter Counter
        N_REQ0    .equ   P1.6          DMA Request 0
        N_REQ1    .equ   P1.7          DMA Request 1

        TM0_RUN   .equ   TCON.4        8751 Timers run control bits
        TM1_RUN   .equ   TCON.6

*********** Byte Constants **************

        INT_MASK  .equ 03EH            Mask off unwanted bits after polling

        FL_IDLE   .equ 00H             Idle Flag
        FL_INIT   .equ    02H          'Initiating Transmission' Flag
        FL_WAIT   .equ    04H          'Ready Wait' Flag
        FL_START  .equ 06H             Start of data flag
        FL_ABORT  .equ 08H             Transmission aborted

        AK_IDLE  .equ 00H              Idle aknowledge
        AK_WAIT  .equ    20H           'WAIT' signal to transmitter
        AK_READY .equ 40H              Receiver Ready Acknowledgement
        AK_ABORT .equ 60H              Block Received Ack.

*********** Bit Variables ***************

        DIR_BIT      .equ 40H          Direction of current or last data Xfer
                                       Set = Bus to card,Reset= Card to Bus
        CH_ACTIVE    .equ 41H          Set = DMA channel active
        TR_1ST_XFER  .equ 42H          Set = First data tranfer to Transmit
                                       buffer (4 Kbytes)
        TRBUF_READY  .equ 43H          Transmitter buffer ready
        TR_FINISH    .equ 44H          Transmission finished

        YEL_ALRM     .equ 45H          Yellow alarm received
        LOSS_SYN     .equ 46H          Loss of signal

        TR_ON        .equ 47H          Transmission ON
        RC_ON        .equ 48H            Reception ON
        XFER1        .equ 49H            First transfer in progress

        RCNT_1STINT .equ 4AH

*********** Byte Variables **************

        VMEINT_LEV .equ 30H              VME bus Interrupt level
        VMEINT_NUM .equ 31H              VME bus Interrupt number

        TR_FLG_STS .equ 32H              Transmission Flag Status
        TR_ACK_STS .equ 33H              Transmission Acknowledge Status
        RC_FLG_STS .equ 34H              Received Flag Status

        TRBLKS          .equ 35H         Total 1K word blocks to transmit
        TRBLKS_2_XFER .equ 36H           # of data blocks to transfer to
                                         transmitter buffer memory
        TRBLKS_MTY     .equ 37H          # of empty transmitter buffer blocks
        TRBLKS_SENT    .equ 38H          # of blocks transmitted

        TRCNT_L    .equ 39H              Transmission count low
        TRCNT_H    .equ 3AH              Transmission count high
        TR_COUNT   .equ 3BH

        TRMEM_ADR_H .equ 3CH             Main memory address of transmit
                                         data block
        TRMEM_ADR_M .equ 3DH
        TRMEM_ADR_L .equ 3EH

        TRBUF_ADR_H .equ 3FH             Transmitter buffer address
        TRBUF_ADR_M .equ 40H             High and Middle bytes


        RCBLKS_2_REC   .equ 45H          # of data blocks to receive
        RCBLKS_2_XFER .equ 46H           # of data bloks to transfer to mem.
        RCBLKS_FULL    .equ 47H          # of receiver buffer blocks full

        RCBUF_ADR_H .equ 48H             High and Middle byte of
        RCBUF_ADR_M .equ 49H             Receiver buffer address

        RCMEM_ADR_H .equ 4AH             Main memory address of received
                                         data block
        RCMEM_ADR_M .equ 4BH
        RCMEM_ADR_L .equ 4CH

        RCCNT_H   .equ 4DH               Received data count
        RCCNT_L   .equ 4EH
        RC_COUNT .equ 4FH
```

```
FAK_CHR   .equ 50H                 Flag ack character

Y_ALARMS .equ 51H                 number of consecutive yellow alarms
LSYNCS    .equ 52H                 # of consecutive 'loss of signal's.
```

```
**************************************************************************
* Program Name : DATA-LINK
* This program controls the Data Link protocol fro the VMEbus
* Intelligent Communication Controller. It also carries out
* the communication with the host.
* Written by   : Dileep R. Idate
* DATE         : December 1988
*
**************************************************************************


       #include "I8051.WRD"          8051 register declarations
       #include "ADDRSS.WRD"         address declarations
       #include "CONST.WRD"          constants and variables
                                     declarations

         .ORG  0000H                 Reset vector

         LJMP  START

         .org  000BH                 Timer 0 interrupt vector

         LJMP  TIM0_SER

START   .ORG  0050H

********* INITIALIZATION SECTION *********


         MOV   IE,#00000000B         Disable All Interrupts
         MOV   SP,#08D               Init Stack Pointer
         MOV   PSW,#00000000B        Clear Flags, Select Reg. Bank 0
         MOV   P1,#0F1H              Init P1 Bits
                                     All bit functions deactivated
         MOV   P3,#0FFH              Enable Secondary functions
         MOV   TCON,#00000000B       Set Interrups Level Triggerd
                                     Clear Timer Overflow Flags
         MOV   TMOD,#00010001B       Timer Mode 1, No Gate Control
         MOV   SCON,#00H             Disable Serial Reception
         MOV   IP,#00000110B         TIM0 =1, INT0 = 0 , INT1 = 1

* RESET THE SYSTEM

         SETB  RESET
         MOV   R5,#10H               Provide enough time for RESET
         DJNZ  R5,$
         CLR   RESET

         MOV   A,#00H
         MOV   DPTR,#FLG_ACK         Make dummy write and read to
```

```
        MOVX  @DPTR,A                    reset the interrupts Flip flops
        MOVX  A,@DPTR                    of FLAG/ACK buffers

        MOV   28H,#00H                   Initialize all bit variables
        MOV   29H,#00H                   to 0.
        CLR   CH_ACTIVE
        MOV   VMEINT_LEV,#04H            Interrupt level = 4
        MOV   VMEINT_NUM,#64D            Interrupt Number = 64

        MOV   RCMEM_ADR_L,#00H           Init the primary storage
        MOV   RCMEM_ADR_M,#00H           address for the received data
        MOV   RCMEM_ADR_H,#03H           address = 030000h

        MOV   Y_ALARMS,#00H
        MOV   LSYNCS,#00H

******** PROGRAM THE 8255 *******************


        MOV   DPTR,#PORT_MODE
        MOV   A,#92H                     Port A,B : Input  C : Output
        MOVX  @DPTR,A

        MOV   A,#00H
        MOV   DPTR,#PORT_C
        MOVX  @DPTR,A                    Clear Address Counter Interrupts
        MOV   A,#03H
        MOVX  @DPTR,A


* ************** PROGRAM 8253 TIMERS ****************

* Set up Modes

        MOV   A,#34H                     TIMER 0 : Rate Generator
        MOV   DPTR,#TIM_MOD
        MOVX  @DPTR,A
        MOV   A,#0B4H                    TIMER 2 : Rate Generator
        MOVX  @DPTR,A

* Load Count Values

        MOV   DPTR,#TIMER_0              Count of 144 in both timers
        MOV   A,#90H                     ie. 1 FLAG/ACK character after
        MOVX  @DPTR,A                    143 data bytes.
        MOV   A,#00H                     ( 2 FLAG/ACK per T1 multiframe )
        MOVX  @DPTR,A

        MOV   DPTR,#TIMER_2
        MOV   A,#90H
        MOVX  @DPTR,A
        MOV   A,#00H
        MOVX  @DPTR,A


* ************* PROGRAM PTVSI REGISTERS ***************


        MOV   DPTR,#INT_MASK_1        Program Int. enable registers
        MOV   A,#01H                  Enable Interrupt for MB00
        MOVX  @DPTR,A                 Disable the rest
        MOV   DPTR,#INT_MASK_2
        MOV   A,#00H
        MOVX  @DPTR,A

        MOV   DPTR,#MB_INT_SEL1       Select pin LIRQ0 for MB00 Int.
        MOV   A,#0FCH
        MOVX  @DPTR,A

        MOV   DPTR,#AM_COMP           Set address modifier code to 39h
        MOV   A,#39H                  standard nonpriviledged data
        MOVX  @DPTR,A                 access

        MOV   DPTR,#AM_X
        MOV   A,#04H
        MOVX  @DPTR,A                 Ignore bit 2

        MOV   DPTR,#ADDR_COMP_2       Set the address of VME slave at
        MOV   A,#90H                   XX9000XX H
        MOVX  @DPTR,A
        MOV   DPTR,#ADDR_COMP_3
        MOV   A,#00H
        MOVX  @DPTR,A

        MOV   DPTR,#MA_ENBL           Init Master Enable Reg.
        MOV   A,#60H                  Aux addr comp = disable
        MOVX  @DPTR,A                 Slave addr. comp. = enable
                                      MB Int = enbl.
                                      Int. Done = Disabled
        MOV   DPTR,#VSI_MOD_SEL       Don't care A31-A24
        MOV   A,#0D9H                 Reset Local Int on MB read.
        MOVX  @DPTR,A

        MOV   DPTR,#MB00               Dummy Read of mailbox 0 to
        MOVX  A,@DPTR                  deactivate interrupt(if any!)

        MOV   DPTR,#MB10
        MOV   A,#00H
        MOVX  @DPTR,A
```

```
* ****** PROGRAM THE DMAC CHANNELS   ********

        MOV   A,#08H
        MOV   DPTR,#DCR_0               Burst transfer mode,
        MOVX  @DPTR,A                   68000 compatible ,16 bit port
        MOV   DPTR,#DCR_1
        MOVX  @DPTR,A

        MOV   A,#05H
        MOV   DPTR,#MFCR_0              Program memory func. code reg.
        MOVX  @DPTR,A
        MOV   DPTR,#MFCR_1
        MOVX  @DPTR,A

        MOV   A,#02H
        MOV   DPTR,#DFCR_0              Program Dev. func code reg.
        MOVX  @DPTR,A
        MOV   DPTR,#DFCR_1
        MOVX  @DPTR,A

        MOV   DPTR,#SCR_0               Program sequence control reg.
        MOV   A,#05H                    MAR and DAR both count up.
        MOVX  @DPTR,A
        MOV   DPTR,#SCR_1
        MOVX  @DPTR,A

        MOV   A,#00H                    Program both channels at a
        MOV   DPTR,#CPR_0               same priority of 0
        MOVX  @DPTR,A
        MOV   DPTR,#CPR_1
        MOVX  @DPTR,A

        MOV   A,#0FFH
        MOV   DPTR,#CSR_0               Init CSR
        MOVX  @DPTR,A
        MOV   DPTR,#CSR_1
        MOVX  @DPTR,A

        MOV   DPTR,#NIVR_0              Program the Normal and error
        MOV   A,#00H                    interrupt vector registers
        MOVX  @DPTR,A
        MOV   DPTR,#EIVR_0               00 , 01
        MOV   A,#01H
        MOVX  @DPTR,A

        MOV   DPTR,#NIVR_1
        MOV   A,#80H
        MOVX  @DPTR,A                    80 , 81
```

```
         MOV   DPTR,#EIVR_1
         MOV   A,#81H
         MOVX  @DPTR,A

         MOV   DPTR,#OCR_0                Program operation control
         MOV   A,#91H                     registers
         MOVX  @DPTR,A                    Dev to Mem, Word Operand
                                          Internal req at max rate
         MOV   DPTR,#OCR_1                Program operation control
         MOV   A,#11H                     registers
         MOVX  @DPTR,A                    Mem to Dev, Word Operand
                                          Internal req at max rate


         ACALL  TR_ABORT                  Init all the variables and flags
         NOP
         ACALL  RC_ABORT                  used for transmission and
                                          reception
         MOV   R5,#0FFH                   wait for some time
         DJNZ  R5,$

         SETB  TRTMR                      Start transmitter and receiver
         SETB  RCTMR                      timers
         MOV   IE,#10000101B              Enable Interrupts 0 and 1
         NOP

REPEAT   MOV  DPTR,#PORT_B
         MOVX  A,@DPTR
         MOV   R6,A
         JNB   ACC.1,YELOK                check for Yellow Alarm

         JB    YEL_ALRM,NXT_CHK           Quit if yellow alarm flag
         INC   Y_ALARMS                   already set
         MOV   A,Y_ALARMS
         CJNE  A,#05D,NXT_CHK
         SETB  YEL_ALRM                   else set it and proceed
         MOV   DPTR,#MB10                 update the status register
         MOVX  A,@DPTR
         SETB  ACC.2
         MOVX  @DPTR,A
         SJMP  NXT_CHK

YELOK    JNB  YEL_ALRM,NXT_CHK
         CLR   YEL_ALRM                   else reset 'yellow alarm' flag
         MOV   DPTR,#MB10                 Update the status register
         MOVX  A,@DPTR
         CLR   ACC.2
         MOVX  @DPTR,A
         MOV   Y_ALARMS,#00H
```

```
NXT_CHK  MOV  A,R6
         ANL  A,#01110000B            check for loss of signal
         CJNE A,#00110000B,CONT1      synchronization
         JNB  LOSS_SYN,CONT2
         CLR  LOSS_SYN                else reset 'loss of sync' flag
         MOV  DPTR,#MB10              update the status register
         MOVX A,@DPTR
         CLR  ACC.3
         MOVX @DPTR,A
         MOV  LSYNCS,#00H
         MOV  DPTR,#PORT_MODE         stop yellow alarm transmission
         MOV  A,#0EH
         MOVX @DPTR,A
         SJMP CONT2

CONT1    JB   LOSS_SYN,CONT2          Quit if loss_syn already set
         INC  LSYNCS
         MOV  A,LSYNCS
         CJNE A,#05H,CONT2
                                      SET it if 5 consecutive samplings
         SETB LOSS_SYN                show loss of signal
         MOV  DPTR,#MB10              update the status register
         MOVX A,@DPTR
         SETB ACC.3
         MOVX @DPTR,A

GO_ON    MOV  DPTR,#PORT_MODE         Send yellow alarm
         MOV  A,#0FH
         MOVX @DPTR,A

CONT2    MOV  R5,#0F0H                wait for some time before next
         DJNZ R5,$                    sampling


         SJMP REPEAT
         NOP
         SJMP REPEAT


********************* SUBROUTINE INT0_SER ********************************
*
*Function : Interrupt sevice routine for interrupt 0
*Entry Requirement : None
*Registers Altered : Acc, R0, R2, R5
*Called By    : Hardware Interrupt # 0
*Calls     : INT_MAIN, FUST_MOV
*
*-----------------------------------------------------------------
```

```
        .org   0003H

        LJMP   INT0_SER

INT0_SER  .org  0200H

        NOP
        PUSH   ACC
        PUSH   PSW
        PUSH   DPH
        PUSH   DPL

        MOV    DPTR,#MB00              Read control word
        MOVX   A,@DPTR

        JB     ACC.0,INT_LEVL          If 08 : Request transmission
        NOP
        JB     ACC.1,INT_NUM           if 04 : change address of
        NOP                             primary storage
        JB     ACC.2,CHG_RCAD          if 01 : change Int. level
        NOP
        JB     ACC.3,REQ_TR            if 02 : change Int. vector
        NOP
        AJMP   EXIT


INT_LEVL MOV   DPTR,#MB01              Read and update interrupt level
        MOVX   A,@DPTR
        MOV    VMEINT_LEV,A
        MOV    R0,#3D
        LCALL   INT_MAIN               Give the acknowledgement
        NOP
        AJMP   EXIT

INT_NUM  MOV   DPTR,#MB01              Read and update interrupt number
        MOVX   A,@DPTR
        MOV    VMEINT_NUM,A
        MOV    R0,#4D
        LCALL   INT_MAIN               Give the acknowledgement
        NOP
        AJMP   EXIT

CHG_RCAD MOV   DPTR,#MB01              New address in MB01 to MB03
        MOVX   A,@DPTR                 MB01 = lowermost byte
        MOV    RCMEM_ADR_L,A           MB02 = middle byte
        MOV    DPTR,#MB02              MB03 = high byte
        MOVX   A,@DPTR
        MOV    RCMEM_ADR_M,A           Save the address
        MOV   DPTR,#MB03
        MOVX  A,@DPTR
        MOV   RCMEM_ADR_H,A

        LCALL  RC_ABORT
        NOP
        MOV   R0,#5D
        LCALL  INT_MAIN
        NOP
        AJMP  EXIT

REQ_TR  NOP
        JB    TR_ON,SKIP             If transmission ON, Ignore
        NOP                          the request
        JB    LOSS_SYN,SKIP          If 'yellow alarm or loss of
        NOP
        JB    YEL_ALRM,SKIP          signal' then ignore request

        JNB   CH_ACTIVE,PR_CH        If the DMA channel is active
        CLR   N_HALT                 (ch. for received data ) then
        MOV   R5,#05H                first halt DMA and then program
        DJNZ  R5,$

PR_CH   MOV   DPTR,#MB01             Read the memory address from
        MOVX  A,@DPTR                mailbox registers and program
        MOV   DPTR,#MAR_BA_1         memory address registers of the
        MOVX  @DPTR,A                DMA
        MOV   DPTR,#MB02
        MOVX  A,@DPTR
        MOV   DPTR,#MAR_BA_1 - 1
        MOVX  @DPTR,A
        MOV   DPTR,#MB03
        MOVX  A,@DPTR
        MOV   DPTR,#MAR_BA_1 - 2
        MOVX  @DPTR,A

        SETB  N_HALT

        MOV   DPTR,#MB04             Read the transfer count
        MOVX  A,@DPTR                break it into 1k word blocks and
        MOV   TRCNT_L,A              the remainder and store it.
        MOV   DPTR,#MB05
        MOVX  A,@DPTR                MB04 = Low Byte
        MOV   TR_COUNT,A             MB05 = High Byte of the count

        MOV   A,TRCNT_L              Check if transfer count = 0
        ORL   A,TR_COUNT             if yes , quit.
        JZ    SKIP

        SETB  TR_ON
```

```
         MOV    DPTR,#MB10                   Set the TR_ON bit in the status
         MOVX   A,@DPTR                      register
         SETB   ACC.0
         MOVX   @DPTR,A

         MOV    A,TR_COUNT
         ANL    A,#3CH                       Split the count in 3 bytes
         RR     A                            1 byte = # of 1K blocks
         RR     A                            2 bytes = remainder count
         MOV    TRBLKS,A
         MOV    A,TR_COUNT
         ANL    A,#03H
         MOV    TRCNT_H,A

         JNZ    ADDONE                       if remainder = 0
         MOV    A,TRCNT_L                    then DMA blocks to transfer
         JNZ    ADDONE                       = # of 1K blocks
         MOV    TRBLKS_2_XFER,TRBLKS
         MOV    TRCNT_H,#04H
         SJMP   GETSET
                                             else
ADDONE   MOV    A,TRBLKS                     DMA blocks to transfer
         INC    A                            = totak 1K blocks + 1
         MOV    TRBLKS_2_XFER,A

GETSET   SETB   TR_1ST_XFER
         JB     CH_ACTIVE,EXIT               check if the DMA is active,
         NOP
         LCALL  FUST_MOV                     if no, start the transfer of
         NOP
         AJMP   EXIT

SKIP     MOV    R0,#02D                      Notify the main processor
         LCALL  INT_MAIN                     'Request not accepted'

EXIT     NOP
         POP    DPL
         POP    DPH
         POP    PSW
         POP    ACC                          data
         NOP
         RETI


******************** SUBROUTINE INT1_SER ******************************
*
*Function : Interrupt service routine for interrupt # 1
*Entry Requirement : None
*Registers Altered : Acc, R1
```

```
*Called By    : Hardware Interrupt 1
*Calls      : FAT_SER, FAR_SER, TC_SER, RC_SER, DMA_SER
*
*-----------------------------------------------------------------------

         .org  0013H

         LJMP  INT1_SER

INT1_SER  .org  0300H

         NOP
         PUSH  ACC
         PUSH  PSW
         PUSH  DPH
         PUSH  DPL

         MOV   DPTR,#PORT_A           Read the Interrupt port
         MOVX  A,@DPTR
         ANL   A,#INT_MASK            Make unused port bits = 0
         RR    A

         SETB  C
         MOV   R1,#0FFH

FIND     RRC   A
         INC   R1
         JC    FIND                   find the interrupt

         MOV   A,R1
         RL    A
         RL    A
         ADD   A,R1
         MOV   DPTR,#ISR_TBL
         JMP   @A+DPTR                Select the corresponding ISR

ISR_TBL  ACALL  FAT_SER
         NOP
         AJMP  OUT

         ACALL  FAR_SER
         NOP
         AJMP  OUT

         ACALL  TC_SER
         NOP
         AJMP  OUT

         ACALL  RC_SER
         NOP
```

```
          AJMP   OUT

          ACALL  DMA_SER
          NOP
          AJMP   OUT

OUT    NOP
          POP    DPL
          POP    DPH
          POP    PSW
          POP    ACC                    data
          NOP
          RETI


******************** SUBROUTINE FAT_SER ********************************
*
*Function : To handle transmission of the Flag/Acnowledge character
*Entry Requirement : None
*Registers Altered : Acc, R0,
*Called By     : INT1_SER
*Calls      : None
*
*-----------------------------------------------------------------


FAT_SER  MOV   A,TR_FLG_STS             Read flag status
         RL    A
         MOV   DPTR,#JP_TBL             First select the flag
         JMP   @A+DPTR

JP_TBL   AJMP  FSTS_0
         AJMP  FSTS_1
         AJMP  FSTS_2
         AJMP  FSTS_3
         AJMP  FSTS_4
         AJMP  FSTS_5
         AJMP  FSTS_6
         AJMP  FSTS_7
         AJMP  FSTS_8

FSTS_0   MOV   FAK_CHR,#FL_IDLE         Send Idle flag
         AJMP  NEXT

FSTS_1   MOV   FAK_CHR,#FL_INIT         Send Init flag
         INC   TR_FLG_STS
         AJMP  NEXT

FSTS_2   MOV   FAK_CHR,TRBLKS           send # of 1k blocks
         INC   TR_FLG_STS
         AJMP  NEXT

FSTS_3   MOV   FAK_CHR,TRCNT_H
         INC   TR_FLG_STS                 send High nibble of the
         AJMP  NEXT                       remainder count

FSTS_4   MOV   A,TRCNT_L                  send Middle nibble
         SWAP  A
         ANL   A,#0FH
         MOV   FAK_CHR,A
         INC   TR_FLG_STS
         AJMP  NEXT

FSTS_5   MOV   A,TRCNT_L                  send Low nibble
         ANL   A,#0FH
         MOV   FAK_CHR,A

         INC   TR_FLG_STS
         MOV   DPTR,#TH0                  Program the watchdog timer
         MOV   A,#0CCH                    for approx. 20mS delay
         MOVX  @DPTR,A
         MOV   DPTR,#TL0
         MOV   A,#00H
         MOVX  @DPTR,A
         SETB  TM0_RUN                    Start the timer
         AJMP  NEXT

FSTS_6   MOV   FAK_CHR,#FL_WAIT           send 'Ready wait 'flag
         AJMP  NEXT

FSTS_7   MOV   FAK_CHR,#FL_START          send 'Start' flag to indicate
         INC   TR_FLG_STS                 starting of data transmission
         AJMP  NEXT

FSTS_8   MOV   FAK_CHR,#FL_IDLE
         MOV   TR_FLG_STS,#00H
         CLR   N_TRSTRT                   Start the data transmission
         AJMP  NEXT


NEXT     MOV  A,TR_ACK_STS                Read acknowledge status
         ANL   A,#03H
         RL    A                          multiply A by 5
         RL    A
         ADD   A,TR_ACK_STS
         MOV   DPTR,#AKSTS                select the proper ack nibble
         JMP   @A+DPTR

AKSTS    ORL   FAK_CHR,#AK_IDLE           send 'IDLE '
         SJMP  SEND
```

```
        ORL   FAK_CHR,#AK_WAIT        or 'WAIT'
        SJMP  SEND

        ORL   FAK_CHR,#AK_READY       or 'READY'
        SJMP  SEND


SEND    MOV   DPTR,#FLG_ACK           Update the flag/ack buffer
        MOV   A,FAK_CHR
        MOVX  @DPTR,A
        RET

******************** SUBROUTINE FAR_SER ********************************
*
*Function : To Process the received Flag/Ack character
*Entry Requirement : None
*Registers Altered : Acc, R0, R1
*Called By    : INT1_SER
*Calls      : None
*
*-----------------------------------------------------------------------


FAR_SER MOV   DPTR,#FLG_ACK           Read the character
        MOVX  A,@DPTR
        MOV   R1,A                    save the character


        MOV   A,RC_FLG_STS            First check flag
        RL    A                       select proper routine depending
        ANL   A,#0EH                  upon the status
        MOV   DPTR,#RCF_STS
        JMP   @A+DPTR

RCF_STS AJMP  RCFS_0                  selection of the routines
        AJMP  RCFS_1
        AJMP  RCFS_2
        AJMP  RCFS_3
        AJMP  RCFS_4


RCFS_0  MOV   A,R1                    retrive the character
        ANL   A,#06H                  mask off the acknowledge part
        MOV   DPTR,#FCHK
        JMP   @A+DPTR

FCHK    AJMP   CHEK_ACK               Do nothing if flag is 'Idle'
        SJMP  FC_INIT
```

```
         SJMP  FC_RWET
         SJMP  FC_STRT

FC_INIT  MOV   RC_FLG_STS,#01H          Get set to receive
                                        the transfer word count
         AJMP  CHEK_ACK                 quit

FC_RWET  MOV   A,RCBLKS_FULL            check if receiver is ready ie.
         JNZ   CHEK_ACK                 check if receiver buffer empty
         MOV   TR_ACK_STS,#02H          if not then quit else send
                                        'AK_READY' acknowledgement
         SETB  RC_ON
         MOV   DPTR,#MB10               update the status register
         MOVX  A,@DPTR
         SETB  ACC.1
         MOVX  @DPTR,A

         AJMP  CHEK_ACK                 quit

FC_STRT  CLR   N_RCSTRT                 'START' flag : start the
         SETB  RCNT_1STINT              receiver
         MOV   TR_ACK_STS,#00H
         AJMP  CHEK_ACK


RCFS_1   MOV   A,R1                     if status is 1
         ANL   A,#0FH
         INC   A
         MOV   RCBLKS_2_REC,A           then store the nibble at two
         MOV   RCBLKS_2_XFER,A          places
         INC   RC_FLG_STS               increament the status
         AJMP  CHEK_ACK

RCFS_2   MOV   A,R1
         ANL   A,#0FH
         MOV   RCCNT_H,A                Store the higher byte of count

         MOV   A,RCBLKS_2_REC           Combine the # of blocks and
         DEC   A
         RL    A                        RCCNT_H to get the high byte of
         RL    A                        the transfer count
         ORL   A,RCCNT_H
         MOV   RC_COUNT,A               store it in RC_COUNT

         INC   RC_FLG_STS
         AJMP  CHEK_ACK

RCFS_3   MOV   A,R1                     Store higher nibble of the lower
         ANL   A,#0FH                   byte of the count
         SWAP  A
```

```
          MOV    RCCNT_L,A
          INC    RC_FLG_STS
          AJMP   CHEK_ACK

RCFS_4    MOV    A,R1
          ANL    A,#0FH
          ORL    RCCNT_L,A                 store lower nibble of the
          MOV    RC_FLG_STS,#00H           lower byte
          MOV    TR_ACK_STS,#01H           send 'AK_WAIT' acknowledgment


CHEK_ACK  MOV    A,R1                      Now check the Acknowledge char.
          ANL    A,#0F0H                   retrive the acknowledge char.

          JZ     GOOUT                     If 0 ie. if 'IDLE' then quit

          SWAP   A
          SUBB   A,#02H                    if 2 ie.
          JNZ    CHK_NXT                   if the ack is 'WAIT' then
          CLR    TM0_RUN                   switch off the watchdog timer
          RET

CHK_NXT   SUBB   A,#02H                    if 4 ie.
          JNZ    GOOUT                     if the ack is 'READY' then
          NOP
          JNB    TRBUF_READY,GOOUT         if Transmitter buffer is ready,
          MOV    TR_FLG_STS,#07H           then Next flag = 'START'
          CLR    TRBUF_READY

GOOUT   RET


********************* SUBROUTINE TC_SER **********************************
*
*Function : To manage transfer of data between the primary storage
*     and the transmitter buffer
*Entry Requirement : None
*Registers Altered : Acc, R0
*Called By    : INT1_SER
*Calls      : TBLK_XFER, TR_ABORT, INT_MAIN
*
*-----------------------------------------------------------------------

TC_SER    MOV    DPTR,#PORT_MODE           Reset the interrupt
          MOV    A,#00H
          MOVX   @DPTR,A
          NOP
          MOV    A,#01H
          MOVX   @DPTR,A
          INC    TRBLKS_MTY
          INC    TRBLKS_SENT
          MOV    A,TRBLKS
          INC    A
          CLR    C
          SUBB   A,TRBLKS_SENT          Check if any more data blocks
          JNZ    XFER_BLK               to send : If yes, check if any
                                        blocks to transfer
          LCALL  TR_ABORT               If no, stop transmission of data
          MOV    R0,#00H
          LCALL  INT_MAIN               Notify main processor of
          NOP                           completion of transmission
          RET

XFER_BLK  MOV    A,TRBLKS_2_XFER        Check if any blocks present to
          JZ     NO                     transfer
          JB     CH_ACTIVE,NO
          LCALL  TBLK_XFER              if yes, call the routine to
NO        NOP                           transfer block
          RET


******************** SUBROUTINE RC_SER ********************************
*
*Function : To manage transfer of data between the primary storage
*     and the receiver buffer
*Entry Requirement : None
*Registers Altered : Acc, R0
*Called By     : INT1_SER
*Calls      : RBLK_XFER, RC_ABORT
*
*-----------------------------------------------------------------


RC_SER    MOV    DPTR,#PORT_MODE        Reset the interrupt
          MOV    A,#02H
          MOVX   @DPTR,A
          NOP
          MOV    A,#03H
          MOVX   @DPTR,A

          JNB    RCNT_1STINT,GOXFER
          CLR    RCNT_1STINT
          RET

GOXFER    DEC    RCBLKS_2_REC           Decreament count for blocks to
          INC    RCBLKS_FULL            receive add 1 to # of full
                                        receive buffer blocks
          MOV    A,RCBLKS_2_REC
          JNZ    FUL_CHK                check if any more blocks
```

```
        SETB   N_RCSTRT                    to receive if no, stop receiver
                                           address counters
FUL_CHK MOV    A,RCBLKS_FULL               check if buffer is full
        CJNE    A,#04H,BUF_OK
        LCALL  RC_ABORT                    If full abort the reception
        NOP
        RET
BUF_OK  JB     CH_ACTIVE,FIN               Start the block transfer if the
        LCALL  RBLK_XFER                   DMA is currently not active
FIN     NOP
        RET


********************* SUBROUTINE DMA_SER ********************************
*
*Function : To service the DMA interrupt
*Entry Requirement : None
*Registers Altered : Acc, R0
*Called By    : INT1_SER
*Calls      : TBLK_XFER, RBLK_XFER, FUST_MOV, TR_ABORT, RC_ABORT
*     : INT_MAIN
*
*-----------------------------------------------------------------------


DMA_SER  CLR  CH_ACTIVE
        MOV    DPTR,#INTVEC
        MOVX   A,@DPTR                     Read interrupt vector
        JB     ACC.0,DMA_ERR
        NOP
        JB     ACC.7,MEM2DEV


DEV2MEM MOV    DPTR,#CSR_0                 Reset the Interrupt
        MOV    A,#0FFH
        MOVX   @DPTR,A

        DEC    RCBLKS_FULL                 Decreament # of full receiver
        DEC    RCBLKS_2_XFER               blocks and # of blocks needed to
                                           transfer by 1.
        MOV    A,RCBLKS_2_XFER             check if the reception is
        JNZ    MOV_ON                      complete by checking if
                                           RBLKS_2_XFER = 0
        MOV    DPTR,#MB13                  Put the number of received words
        MOV    A,RC_COUNT                  in MB13 and MB14
        MOVX   @DPTR,A
        MOV    DPTR,#MB14
        MOV    A,RCCNT_L
        MOVX   @DPTR,A
```

```
          LCALL  RC_ABORT
          NOP
          MOV    R0,#01D                   notify main processor of the
          LCALL  INT_MAIN                  reception of data
          NOP

MOV_ON    JNB    TR_1ST_XFER,CHKFRTR       Check if first block transfer
          LCALL  FUST_MOV                  needed : if yes , do it
          NOP
          RET

CHKFRTR   MOV    A,TRBLKS_2_XFER           else check for normal mem to dev
          JZ     CHKFRRC                   transfer : if yes , do it
          MOV    A,TRBLKS_MTY
          JZ     CHKFRRC
          LCALL  TBLK_XFER
          NOP
          RET

CHKFRRC   MOV    A,RCBLKS_FULL             else check for dev to mem
          JZ     GTO                       pending xfer : if yes , do it
          LCALL  RBLK_XFER
GTO       NOP
          RET


MEM2DEV   MOV    DPTR,#CSR_1               Reset the Interrupt
          MOV    A,#0FFH
          MOVX   @DPTR,A

          JB     XFER1,FUST_XFR            check if it was the first block
                                           transfer
          DEC    TRBLKS_MTY                update necessary variables
          DEC    TRBLKS_2_XFER
          SJMP   TST_RCB

FUST_XFR  CLR    XFER1
          SETB   TRBUF_READY
          MOV    TR_FLG_STS,#01H

TST_RCB   MOV    A,RCBLKS_FULL             check for any pending buf to mem
          JZ     CK_TBLK                   transfer : if yes, do it
          LCALL  RBLK_XFER
          NOP
          RET

CK_TBLK   MOV    A,TRBLKS_2_XFER           else check for any pending mem
          JZ     GOGO                      to buf transfers , if yes do it
          MOV    A,TRBLKS_MTY
          JZ     GOGO
```

```
        LCALL  TBLK_XFER
GOGO    NOP
        RET


DMA_ERR NOP
        JB     ACC.7,M2D_ERR

        MOV    DPTR,#CER_0          Read error vector and store it
        MOVX   A,@DPTR              in MB14
        MOV    DPTR,#MB14
        MOVX   @DPTR,A
        MOV    DPTR,#CSR_0
        MOV    A,#0FFH              Reset the Interrupt
        MOVX   @DPTR,A

        MOV    R0,#7D               Notify the host about
        LCALL  INT_MAIN             transmission abortion
        NOP
        LCALL  RC_ABORT
        NOP
        AJMP   MOV_ON

M2D_ERR MOV    DPTR,#CER_1          Read error vector and store it
        MOVX   A,@DPTR              in MB14
        MOV    DPTR,#MB14
        MOVX   @DPTR,A
        MOV    DPTR,#CSR_1
        MOV    A,#0FFH              Reset the Interrupt
        MOVX   @DPTR,A

        LCALL  TR_ABORT             Abort the data reception if error
        MOV    R0,#06D              in DMA transfer
        LCALL  INT_MAIN             Notify Host

        NOP
        AJMP   TST_RCB

******************** SUBROUTINE FUST_MOV ********************************
*
*Function : To transfer the first block of data for transmission
*Entry Requirement : None
*Registers Altered : Acc
*Called By     : INT0_SER, DMA_SER
*Calls       : None
*
*----------------------------------------------------------------------
```

```
FUST_MOV SETB  CH_ACTIVE
         CLR   TR_1ST_XFER
         SETB  XFER1
         SETB  DIR_BIT

         MOV   A,TRBLKS_2_XFER         Program the 'Remainder' count if
         DEC   A                        TRBLKS_2_XFER <= 2
         JZ    TR_REMAIN
         DEC   A
         JZ    TR_REMAIN

         MOV   DPTR,#MTCR_L_1          Init memory transfer count
         MOV   A,#00H                  Count of 2000 D
         MOVX  @DPTR,A
         DEC   DPTR
         MOV   A,#08H
         MOVX  @DPTR,A

         DEC   TRBLKS_2_XFER
         DEC   TRBLKS_2_XFER
         MOV   TRBLKS_MTY,#02H
         SJMP  GO_ST

TR_REMAIN MOV   DPTR,#MTCR_L_1         Init memory transfer count
         MOV   A,TRCNT_L               Count is the remainder count
         MOVX  @DPTR,A
         DEC   DPTR
         MOV   A,TR_COUNT
         MOVX  @DPTR,A
         MOV   TRBLKS_2_XFER,#00H

GO_ST    MOV   DPTR,#CCR_1             Program channel control
         MOV   A,#88H                  Start channel, Enbl. interrupt
         MOVX  @DPTR,A


         RET

******************** SUBROUTINE TBLK_XFER ******************************
*
*Function : To program the DMA for the transfer of data from the
*   : primary memory to the local transmitter buffer memory
*Entry Requirement : None
*Registers Altered : Acc
*Called By    : DMA_SER, TC_SER
*Calls      : None
*
*----------------------------------------------------------------
```

```
TBLK_XFER SETB  CH_ACTIVE
          SETB  DIR_BIT

          MOV   DPTR,#DAR_BA_1 - 1      correct the device base address
          MOVX  A,@DPTR                 if outside the buffer range
          ANL   A,#1FH
          MOVX  @DPTR,A

          MOV   A,TRBLKS_2_XFER         Check if it is a last transfer
          DEC   A                       if last then
          JNZ   TTR_1K                       xfer cnt = remainder
                                              else  xfer cnt = 1 K
          MOV   DPTR,#MTCR_L_1          Init memory transfer count
          MOV   A,TRCNT_L               Count is the remainder count
          MOVX  @DPTR,A
          DEC   DPTR
          MOV   A,TRCNT_H
          MOVX  @DPTR,A
          SJMP  CH1_CTR

TTR_1K    MOV   DPTR,#MTCR_L_1          Init memory transfer count
          MOV   A,#00H                  Count of 1000 D
          MOVX  @DPTR,A
          DEC   DPTR
          MOV   A,#04H
          MOVX  @DPTR,A

CH1_CTR   MOV   DPTR,#CCR_1             Program channel control
          MOV   A,#88H                  Start channel, Enbl. interrupt
          MOVX  @DPTR,A


          RET


********************* SUBROUTINE RBLK_XFER *******************************
*
*Function : To program the DMA for the transfer of data from the local
*   : receiver buffer memory to the primary memory
*Entry Requirement : None
*Registers Altered : Acc
*Called By    : DMA_SER, RC_SER
*Calls      : None
*
*-----------------------------------------------------------------


RBLK_XFER MOV   DPTR,#DAR_BA_0 - 1      correct the device address if
```

```
         MOVX  A,@DPTR                    it is outside the buffer range
         ANL   A,#3FH
         JZ    CRRECT
         MOVX  @DPTR,A
         SJMP  GOCNT
CRRECT   MOV   A,#20H
         MOVX  @DPTR,A

GOCNT    MOV   A,RCBLKS_2_XFER            if last transfer then
         DEC   A                            count = remainder
         JNZ   RTR_1K                       else  count = 1 K

         MOV   DPTR,#MTCR_L_0             Init memory transfer count
         MOV   A,RCCNT_L                  Count is the remainder count
         MOVX  @DPTR,A
         DEC   DPTR
         MOV   A,RCCNT_H
         MOVX  @DPTR,A
         SJMP  CHO_CTR

RTR_1K   MOV   DPTR,#MTCR_L_0             Init memory transfer count
         MOV   A,#00H                     Count of 1000 D
         MOVX  @DPTR,A
         DEC   DPTR
         MOV   A,#04H
         MOVX  @DPTR,A

CHO_CTR  MOV   DPTR,#CCR_0                Program channel control
         MOV   A,#88H                     Start channel, Enbl. interrupt
         MOVX  @DPTR,A

         SETB  CH_ACTIVE
         CLR   DIR_BIT

         RET


******************** SUBROUTINE TR_ABORT ********************************
*
*Function : To abort the data transmission
*Entry Requirement : None
*Registers Altered : Acc, R5
*Called By    : TIMO_SER, DMA_SER, YELALRM
*Calls      : None
*
*------------------------------------------------------------------------


TR_ABORT  PUSH  ACC
```

```
          SETB  N_TRSTRT                  Reset varios bits
          CLR   TR_1ST_XFER
          CLR   TRBUF_READY
          CLR   TR_ON
          MOV   DPTR,#MB10                update the status register
          MOVX  A,@DPTR
          CLR   ACC.0
          MOVX  @DPTR,A

          MOV   TRBLKS_MTY,#04H           Set the Transmitter buffer Empty

          MOV   TRBLKS_SENT,#00H          Init varios variables
          MOV   TRBLKS_2_XFER,#00H
          MOV   TR_FLG_STS,#00H

          JNB   CH_ACTIVE,PRGRM           If DMA not active then go ahead
          CLR   N_HALT                    and program the registers
          MOV   R5,#05H                   else Halt it.
          DJNZ  R5,$
          JNB   DIR_BIT,PRGRM             If ch.1 is not active
          MOV   DPTR,#CCR_1                then program it's registers
          MOV   A,#80H                     else first abort it
          MOVX  @DPTR,A
          NOP
          NOP
          MOV   A,#00H                    Reset channel control register
          MOVX  @DPTR,A
          MOV   DPTR,#CSR_1               Reset channel status register
          MOV   A,#0FFH
          MOVX  @DPTR,A
          CLR   CH_ACTIVE

PRGRM     MOV   DPTR,#DAR_BA_1            Initialize Transmitter buffer
          MOV   A,#00H                    address in DMA
          MOVX  @DPTR,A
          DEC   DPTR
          MOVX  @DPTR,A                   address = 010000 h
          DEC   DPTR
          MOV   A,#01H
          MOVX  @DPTR,A

          SETB  N_HALT
          POP   ACC
          RET


******************** SUBROUTINE RC_ABORT ******************************
*
*Function : To abort the data reception
*Entry Requirement : None
```

```
*Registers Altered : Acc
*Called By    : DMA_SER, ERRCHEK
*Calls        : None
*
*-----------------------------------------------------------------------


RC_ABORT PUSH  ACC
         SETB  N_RCSTRT
         CLR   RC_ON
         MOV   DPTR,#MB10                update the status register
         MOVX  A,@DPTR
         CLR   ACC.1
         MOVX  @DPTR,A

         MOV   RCBLKS_FULL,#00H
         MOV   RCBLKS_2_XFER,#00H
         MOV   TR_ACK_STS,#00H
         MOV   RC_FLG_STS,#00H

         JNB   CH_ACTIVE,PRGRM2          If DMA not active then go ahead
         CLR   N_HALT                    and program the registers
         MOV   R5,#05H                   else Halt it.
         DJNZ  R5,$
         JB    DIR_BIT,PRGRM2            If ch.1 is not active
         MOV   DPTR,#CCR_0                then program it's registers
         MOV   A,#80H                     else first abort it
         MOVX  @DPTR,A
         NOP
         NOP
         MOV   A,#00H                    Reset channel control register
         MOVX  @DPTR,A
         MOV   DPTR,#CSR_0               Reset channel status register
         MOV   A,#0FFH
         MOVX  @DPTR,A
         CLR   CH_ACTIVE

PRGRM2   MOV   DPTR,#MAR_BA_0            Init main memory address for the
         MOV   A,RCMEM_ADR_L             received data
         MOVX  @DPTR,A
         DEC   DPTR
         MOV   A,RCMEM_ADR_M
         MOVX  @DPTR,A
         DEC   DPTR
         MOV   A,RCMEM_ADR_H
         MOVX  @DPTR,A

         MOV   DPTR,#DAR_BA_0            Initialize Receiver buffer
         MOV   A,#00H                    address in DMA
         MOVX  @DPTR,A
         DEC   DPTR
         MOV   A,#20H                   address = 012000 h
         MOVX  @DPTR,A
         DEC   DPTR
         MOV   A,#01H
         MOVX  @DPTR,A

         SETB  N_HALT
         POP   ACC
         RET


******************** SUBROUTINE TIMO_SER *********************************
*
*Function : To service the internal timer# 0 interrupt
*Entry Requirement : None
*Registers Altered : R0
*Called By    : Timer 0 Interrupt
*Calls      : INT_MAIN, TR_ABORT
*
*-----------------------------------------------------------------------


TIMO_SER CLR   TMO_RUN                  Abort transmission process
         LCALL  TR_ABORT                since no response from
         MOV   R0,#08H                  receiver within the time limit
         LCALL  INT_MAIN                Notify Host
         NOP
         RETI


*************** SUBROUTINE INT_MAIN ************************************
*
*Function : To generate an interrupt to the main processor
*Entry Requirement : Message code in reg. R0
*Registers Altered : None
*Called By    : INT0_SER, TC_SER, DMA_SER, TMO_SER, YELALRM
*Calls      : None
*
*-----------------------------------------------------------------------

INT_MAIN NOP
         PUSH  ACC

         MOV   A,R0
         MOV   DPTR,#MB15
         MOVX  @DPTR,A

         MOV   DPTR,#INT_REQ_LEV
         MOV   A,VMEINT_LEV
```

```
          MOVX  @DPTR,A
          MOV   DPTR,#INT_VEC_NUM
          MOV   A,VMEINT_NUM
          MOVX  @DPTR,A

          POP   ACC
          RET


*---------------------------------------------------------------

          .ORG  800H                      These instructions are
                                          put to provide a recovery on
          LJMP  START                     a random program jump into
                                          unused area of the EPROM
          .ORG  0B00H                     due to any noise.

          LJMP  START
                                          The unused area of the EPROM
          .ORG  0FFDH                     is initialised to 00h (NOP)
                                          through 'fill' option at the
          LJMP  START                     time of assembly.


          .END
```

Appendix C

Part List

This appendix lists all the parts used to build one prototype card. The prices are aprroximate and they apply to a purchase of a single component.

C.1 TTL/CMOS Chips

CHIP NAME	QUANTITY	PRICE	DESCRIPTION
74S04	1	0.47	Schottky Hex Inverter
74LS04	3	0.52	Hex Inverter
74LS08	2	0.36	Quad AND Gate
74LS10	1	0.32	Triple 3 input NAND
74LS20	1	0.34	Dual 4 input NAND
74LS30	1	0.36	8 input NAND
74LS32	5	0.36	Quad 2 input OR
74S38	1	0.45	Quad O.C. NAND
74LS74	7	0.52	Dual D Flip Flop
74LS112	1	0.47	Dual JK Flip Flop
74LS138	2	0.72	3 to 8 Demultiplexer
74LS148	1	0.72	8 to 3 Encoder
74LS244	3	1.42	Tristate Octal Buffers
74LS245	4	1.42	Tristate Bus Transceiver
74LS321	1	2.34	Crystal Controlled Oscillator
74LS373	3	1.17	Octal Latches
74LS374	2	1.17	Octal Flip Flops
74ALS645-1	7	2.55	Tristate Bus Transceivers
74AS760	2	3.33	O.C. Bus Buffers
74AS867	4	4.76	8 Bit Synchronous Counters
Intel 8253	1	1.89	Prog. Interval Timer
Intel 8255	1	1.69	Prog. Peripheral Interface
Intel 8751-8	1	40.00	Microcontroller
MC68440RC10	1	60.00	DMA Controller
PT-VSI	1	100.00	VME Slave Interface
IDT7134L70P	4	55.00	8k * 8 Dual Port RAM
R8069	1	30.00	T1 Line Interface Unit
R8070J	1	50.00	T1 Transceiver

C.2 Miscelleneous Parts

NAME/DESCRIPTION	QUANTITY	PRICE
Crystalls		
6.176 MHz	2	10.00
8.000 MHz	1	1.62
10 MHz TTL Oscillator	1	5.75
Pulse Transformers		
Pulse Engineering F27.1	1	
Pulse Engineering F28	1	
Capacitors		
10 pF	1	0.10
27 pF	2	0.10
47 pF	1	0.10
1000 pF	55	0.10
0.1 MuF	10	0.20
10.0 MuF	1	0.20
330 MuF	2	4.0
Inductor 15 MuH	1	1.30
Common Terminal SIP Resistors		
5 Element 1.0K	3	0.16
5 Element 2.2K	3	0.16
9 Element 2.2K	1	0.27
Resistor 10 K, 0.25 W	1	0.10

In addition to these components, a prototype wirewrap board, wirewrap sockets and wires were needed.

Appendix D

Software for the Host Processor

This appendix contains six programs. The programs have been written for the Motorola 68020 microprocessor. The programs are :

1. Program to change the Interrupt Vector
2. Program to change the Interrupt Level
3. Program to change the address of the location to store the received data.
4. A trivial Apllication Program
5. A 'Send' routine
6. The Interrupt Service Routine.

All the above programs use the same interrupt service routine. The programs are listed below.

```
*******************************************************************************

*  Address Declarations

*******************************************************************************

        MB00       EQU       $00900001    ; PT-VSI Mailbox Adresses
        MB01       EQU       $00900003
        MB10       EQU       $00900015
        MB14       EQU       $0090001D
        MB15       EQU       $0090001F
        IFLAG      EQU       $00004F00    ; ISR Execution Flag

*******************************************************************************

* Program to change the Interrupt Vector

*******************************************************************************

        ORG        $5000

        MOVEA.L    #INTSER,$100         ; Init ISR address
        MOVE.L     #$2100,SR            ; Init status Register
        MOVEA.L    #IFLAG,A2            ;
        MOVE.B     #$00,(A2)            ; Init Flag

        MOVEA.L    #MB01,A1
        MOVE.B     #64D,(A1)            ; New Interrupt Vector (64) in MB01
        MOVEA.L    #MB00,A1
        MOVE.B     #02D,(A1)            ; Command to change Int. Vector

LOOP:   MOVE.B     (A2),D0
        CMPI.B     #$0FF,D0             ; Check if Flag set
        BNE        LOOP                 ; No : wait in the loop
        MOVE.B     #$00,(A2)            ; Reset Flag
        TRAP       #15                  : Return to DEBUG
        DC.W       $0063


*******************************************************************************

* Program to change the Interrupt Level

*******************************************************************************

        ORG        $5100

        MOVEA.L    #INTSER,$100         ; Init ISR address
        MOVE.L     #$2100,SR            ; Init status Register
        MOVEA.L   #IFLAG,A2           ;
        MOVE.B    #$00,(A2)           ; Init Flag

        MOVEA.L   #MB01,A1
        MOVE.B    #04D,(A1)           ; New Interrupt Level (4) in MB01
        MOVEA.L   #MB00,A1
        MOVE.B    #01D,(A1)           ; Command to change Int. Level

NEXT:   MOVE.B    (A2),D0
        CMPI.B    #$0FF,D0            ; Check if Flag set
        BNE       NEXT                ; No : wait in the loop
        MOVE.B    #$00,(A2)           ; Reset Flag
        TRAP      #15                 : Return to DEBUG
        DC.W      $0063


*****************************************************************************

* Program to change the Storage Location For The Received Block

*****************************************************************************

        ORG       $5200

        MOVEA.L   #INTSER,$100        ; Init ISR address
        MOVE.L    #$2100,SR           ; Init status Register
        MOVEA.L   #IFLAG,A2           ;
        MOVE.B    #$00,(A2)           ; Init Flag

        MOVEA.L   #MB01,A1
        MOVE.B    #$00,(A1)           ; New Address in Registers
        MOVE.B    #$00,(2,A1)         ; MB01 to MB03
        MOVE.B    #$02,(4,A1)

        MOVEA.L   #MB00,A1
        MOVE.B    #04D,(A1)           ; Command to change Location
                                      ; Address
NEX:    MOVE.B    (A2),D0
        CMPI.B    #$0FF,D0            ; Check if Flag set
        BNE       NEX                 ; No : wait in the loop
        MOVE.B    #$00,(A2)           ; Reset Flag
        TRAP      #15                 : Return to DEBUG
        DC.W      $0063


*****************************************************************************

* A SImple Application Program
```

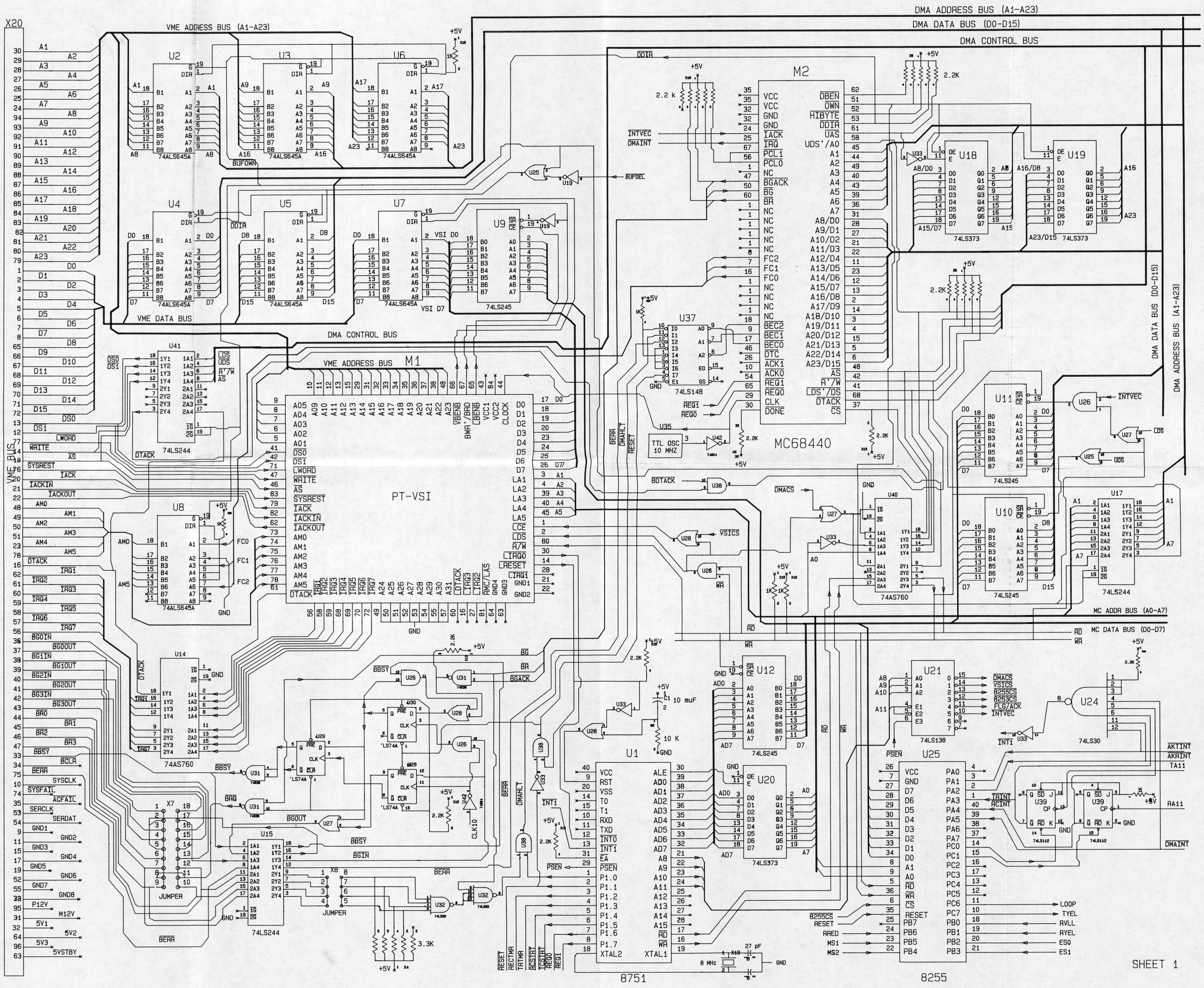
```
*********************************************************************

        ORG         $5300

        MOVEA.L     #INTSER,$100        ; Init ISR address
        MOVE.L      #$2100,SR           ; Init status Register
        MOVEA.L     #IFLAG,A2           ;
        MOVE.B      #$00,(A2)           ; Init Flag

        MOVE.B      #$00,D1             ; Block Address (010000H) in
        MOVE.B      #$00,D2             ; registers D1 to D3
        MOVE.B      #$01,D3
        MOVE.B      #$0F0,D4            ; Transfer Count (04F0H) in
        MOVE.B      #$04,D5             ; registers D4 and D5
        BRS         SEND                ; Call the subroutine SEND


GO:     MOVE.B      (A2),D0
        CMPI.B      #$0FF,D0            ; Check if Flag set
        BNE         GO                  ; No : wait in the loop
        MOVE.B      #$00,(A2)           ; Reset Flag
        TRAP        #15                 : Return to DEBUG
        DC.W        $0063

*********************************************************************

*  The SEND Subroutine

*********************************************************************

        ORG         $5400


SEND:   MOVEA.L     #MB10,A3
        MOVE.B      (A3),D0             ; Read The Status Register
        CMPI.B      #$01,D0             ; Check for error or busy status
        BGT         OUT                 ; Quit if error or busy

        MOVEA.L     #MB01,A1
        MOVE.B      D1,(A1)             ; Block Address to Registers
        MOVE.B      D2,(2,A1)           ; MB01 - MB03
        MOVE.B      D3,(4,A1)
        MOVE.B      D4,(6,A1)           ; Transfer Count to Registers
        MOVE.B      D5,(8,A1)           ; MB04 - MB05

        MOVEA.L     #MB00,A1
        MOVE.B      #$08,(A1)           ; Command to Transmit data
        RTS

OUT:    MOVE.B    #$0FF,(A2)           ; Give message if the request
        MOVEA.L   #MSG,A0              ; can not be accepted
        MOVE.L    A0,A1
        ADD.W     #65D,A1
        MOVEM.L   A0/A1,-(SP)
        TRAP      #15
        DC.W      $0022
        RTS

MSG:    DC.B      'REQUEST NOT ACCEPTED.'
        DC.B      'TRANSMISSION IN PROGRESS OR ERROR CONDITION.'


*************************************************************************

*  Interrupt Service Routine

*************************************************************************

        ORG       $5500

INTSER: MOVEA.L #MB15,A1              ; Read the Message Code
        MOVE.B    (A1),D1
        MULS.W    #28D,D1              ; Calculate the address of
        MOVEA.L   #STR1,A0             ; the message string
        ADD.W     D1,A0
        MOVE.L    A0,A1
        ADD.W     #28D,A1
        MOVEM.L   A0/A1,-(SP)          ; Push the addresses on stack
        MOVEA.L   #MB14,A4
        MOVE.B    (A4),D3

        MOVEA.L   #IFLAG,A4
        MOVE.B    #$0FF,(A4)

        TRAP      #15                  ; Print the message
        DC.W      $0022
        RTE


STR1:   'TRANSMISSION COMPLETE       '
        'DATA RECEIVED               '
        'TRANMISSION ABORTED         '
        'INT. LEVEL CHANGED          '
        'INT. VECTOR CHANGED         '
        'STORAGE ADDRESS CHANGED     '
        'TR ABORT: DMA TRANSFER ERROR'
        'RC ABORT: DMA TRANSFER ERROR'
```

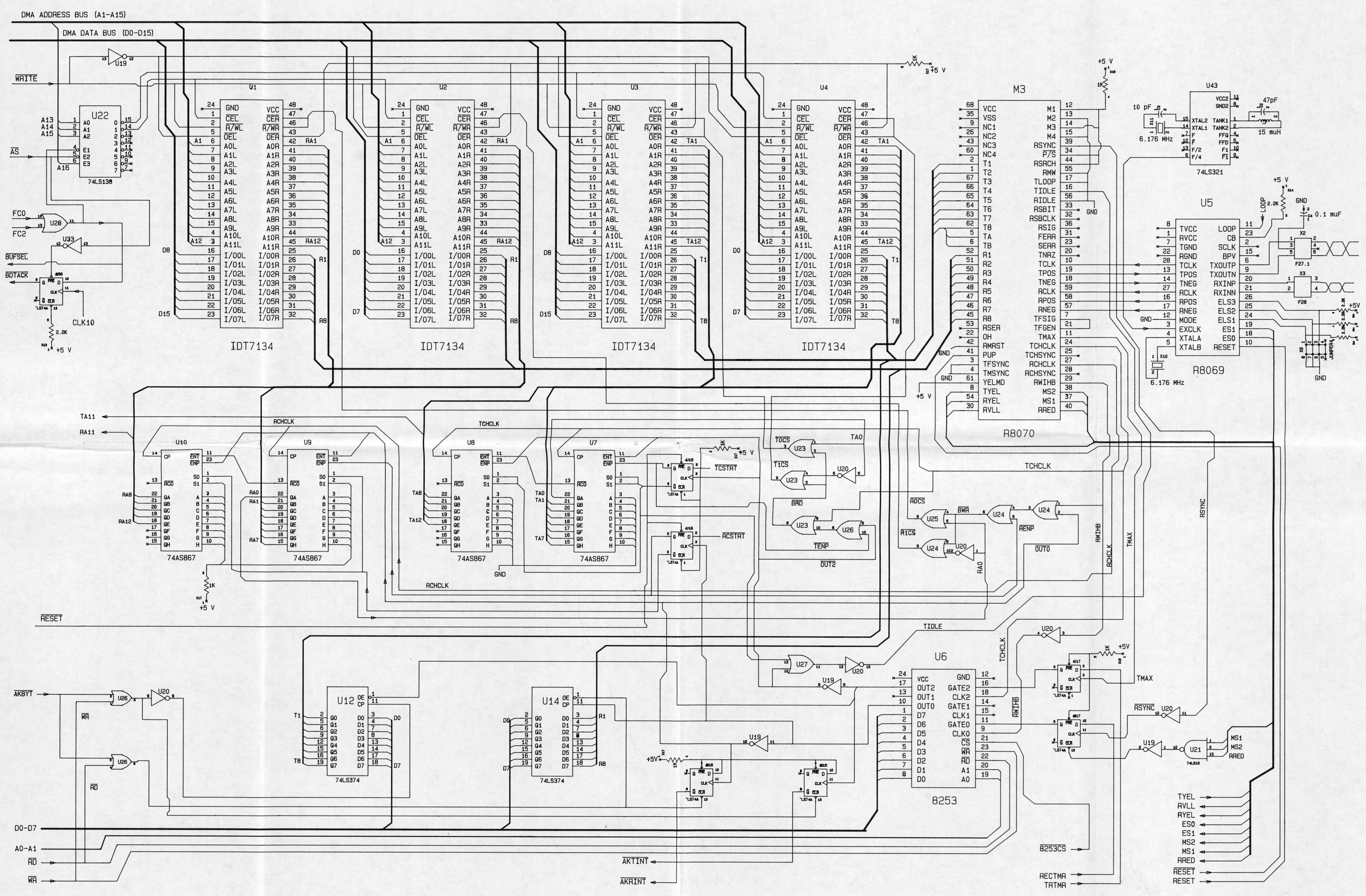
```
'TR ABORT: WATCHDOG TIMEOUT  '
'TR ABORT: YEL ALRM RECEIVED '
```

Appendix E

The Circuit Diagram

This appendix contains the detailed circuit diagram of the card. It is divided in to two sheets.


X20
VME BUS
VME ADDRESS BUS (A1-A23)
VME DATA BUS
DMA ADDRESS BUS (A1-A23)
DMA DATA BUS (D0-D15)
DMA CONTROL BUS
MC ADDR BUS (A0-A7)
MC DATA BUS (D0-D7)
U2
U3
U6
U4
U5
U7
74ALS645A
U9
74LS245
U41
74LS244
U8
M1
PT-VSI
M2
MC68440
U37
74LS148
TTL OSC 10 MHZ
U18
U19
74LS373
U11
U10
U17
U40
74AS760
U14
U15
X7
X8
JUMPER
U12
U20
U21
74LS138
U24
74LS30
U39
74LS112
U1
8751
U25
8255
'LS74A
8 MHz
27 pF
10 muF
10 K
2.2K
3.3K
+5V
GND
SHEET 1


DMA ADDRESS BUS (A1-A15)
DMA DATA BUS (D0-D15)
WRITE
AS
FC0
FC2
BUFSEL
BDTACK
CLK10
U22
74LS138
U1
U2
U3
U4
IDT7134
M3
R8070
U43
74LS321
6.176 MHz
U5
R8069
U10
U9
U8
U7
74AS867
TA11
RA11
RCHCLK
TCHCLK
RESET
TCSTRT
RCSTRT
TIDLE
U6
8253
U12
U14
74LS374
AKBYT
D0-D7
A0-A1
RD
WR
AKTINT
AKRINT
8253CS
RECTMR
TRTMR
TMAX
RSYNC
TYEL
RVLL
RYEL
ES0
ES1
MS2
MS1
RRED
RESET
RESET

SHEET 2