INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book. These are also available as one exposure on a standard 35mm slide or as a 17" x 23" black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 1337428

A parallel adapter for a high-speed serial bus

Gray, Terrence Patrick, M.S.

The University of Arizona, 1989

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

A PARALLEL ADAPTER FOR A HIGH-SPEED SERIAL BUS

by

Terrence Patrick Gray

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE
WITH A MAJOR IN ELECTRICAL ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

1989

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotations from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: Leanne P. Gray

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

F.J. Hill
Professor of Electrical and
Computer Engineering

25 May 1989
Date

ACKNOWLEDGMENTS

I would like to thank the members of my defense committee, Dr. William Sanders, Dr. Ralph Martinez, and Dr. Fredrick Hill, for their help with this thesis. I would also like to give special thanks to Dr. Donald L. Shirer without whose patience this work would not have been completed.

TABLE OF CONTENTS

TABLE OF CONTENTS--Continued

LIST OF ILLUSTRATIONS

Page

Figure

LIST OF ILLUSTRATIONS--Continued

LIST OF TABLES

Page

Table

ABSTRACT

This paper describes the building of a parallel converter for a high-speed serial bus. The high-speed serial port of the Macintosh personal computer is used to implement the bus, while an MC68000 Educational Computer Board is used to perform the serial-to-parallel conversion. The device's performance is evaluated, and possible methods for improving its performance are discussed.

CHAPTER 1

INTRODUCTION

The Macintosh personal computer, introduced in early 1984, represented, in many ways, a revolutionary approach to the design of small computers. As one of the first computers with the ability to execute commands by "dragging a mouse and "clicking" over icons of the program to be executed, it has set a standard for user friendliness that many other companies are attempting to emulate (Williams 1984, pp 30-31). Coupled with this innovative approach to the software, many unique hardware features were also designed into the Macintosh achieving great economy in the chip count, with a corresponding increase in the machine's reliability.

The most notable feature of the Macintosh hardware, however, was the replacement of the usual expansion slots and input-output ports with two high-speed serial ports provided by a single chip. This approach was touted by the designers, because it reduced the chip count by replacing the usual serial and parallel port chips with a single chip. The high speed of the ports had the added benefit of allowing expansion by daisy-chaining the peripherals rather than having the peripherals plug into expansion slots. This freed the designers from having to deal with supplying power to as yet unspecified peripheral devices, and reduced radio-frequency interference problems by enabling the designers to enclose the computer in a seamless case (Williams 1984, p. 43). Since many peripherals require parallel data input and output, there is a great need for a device to convert

the high-sped serial data stream into parallel outputs for a number of peripheral devices simultaneously. This paper describes the design and building of such a device.

CHAPTER 2

DESIGN OF THE SERIAL-TO-PARALLEL CONVERTER

A block diagram of the converter is shown in Figure 1. The transceiver interfaces to the high-speed serial data bus, and controls the receiving of messages from the Macintosh at the hardware level. The serial-to-parallel converter simply takes the data stream and outputs it eight bits at a time. The port controller sends the data to the appropriate port, while controlling activation and monitoring of the ports' handshake lines. It would also be desirable to have this process work in reverse. Data received from each port could be converted to serial form, and sent by the transceiver back to the Macintosh. With this overall description in mind it is now possible to discuss the hardware and software design of the converter in detail.

Hardware Design of the Serial-to-Parallel Converter

Now that the basic operational characteristics of the converter have been determined, it is possible to examine the hardware options for the basic functional blocks of the converter in more detail. The converter can be viewed as having four parts as shown in Figure 1: the transceiver, the serial-to-parallel converter, the port controller, and the input-output ports. Any one of these sections or all of them could be implemented as discrete logic or by more complex integrated circuits. There were three major factors mitigating against an implementation using discrete logic. First, the chip count would be greater, in general, in an

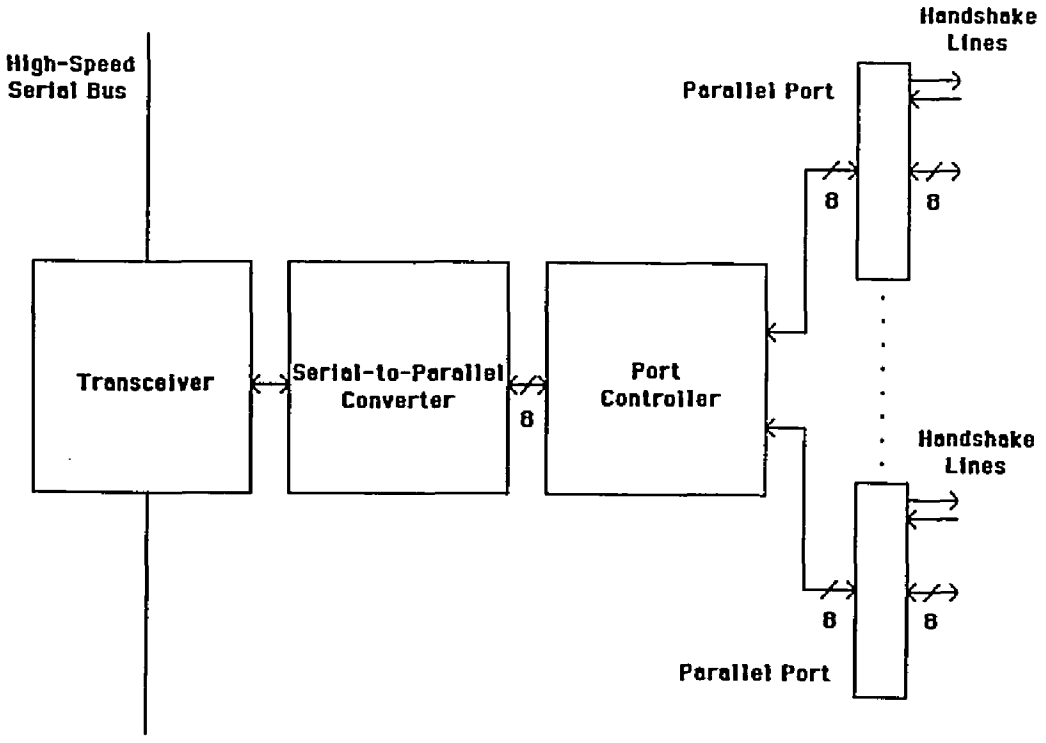

Figure 1. Functional Block Diagram of the Serial-to-Parallel Converter

implementation using discrete logic than in one using more complex integrated circuits. Second, the Macintosh supplies little power for external devices, and, therefore, any discrete logic implementation would require the building or purchasing of a power supply. Third, a discrete logic implementation would require a complex control sequencer. Such functions as the handling of messages, error detection and recovery, and port input-output control are relatively complex. The sequencer would require either a large number of discrete logic chips to implement or the use of a ROM-based microcoded sequencer, with the enormous complexity induced by the need to develop and debug the microcode. Since these factors would add a great deal of complexity, expense and building time, it was decided to use the more complex integrated circuit chips whenever possible.

With these considerations in mind, the Zilog 8530 Serial Communications Controller (hereafter referred to as the Z8530SCC or, simply, as the SCC) was chosen to implement the transceiver and serial-to-parallel converter portions of the converter. This same chip is used in the Macintosh to implement the two high-speed serial ports (Guterl 1984, pp. 38-39). This extremely versatile device supports two full-duplex channels in asynchronous and a variety of synchronous protocols. It also supports a number of data encoding methods or the clocking for the data can be supplied externally. The Z8530SCC also supports a number of DMA and modem control lines. In addition, numerous other features are available (Zilo. 1983, pp. 1.1-1.2).

Needless to say, a device this sophisticated has a complicated control system. Each channel contains sixteen write and eight read registers. The various operational modes are controlled by programming the sixteen write registers, while various status conditions can be monitored by using the eight read registers (Zilo. 1983 p. 3.8). The task of programming and monitoring the Z8530SCC almost requires the use of a microprocessor.

In spite of the fact that a ROM-based sequencer could be used to drive the Z8530SCC, a microprocessor would probably be a better choice for several reasons. Dealing with the messages, for instance, is best done in the assembly language of the microprocessor, either directly or as compiled code from some higher-level language, rather than in microcode. The port controller block of the converter is also a complex system which is responsible, among other things, for synchronizing the data flow between the Z8530SCC and the parallel input-output ports. The easiest way to accomplish this task is again by programming a microprocessor rather than trying to write microcode. The microprocessor, by doubling as the port controller, eliminates any further need to consider the question of a hardware realization for it. Additionally, choosing a microprocessor as the Z8530SCC driver and port controller does not rule out the possibility of later implementing the converter using a microcoded ROM. Once a microcoding scheme and corresponding hardware have been developed, all that is necessary is the translation of the converter's software from the microprocessor's assembly language to the microcode. The flexibility and power of the microprocessor make it the best choice for controlling the converter.

In many ways the MC68000 was the logical choice as the processor to use. It is the processor used in the Macintosh (Williams 1984, p.35) which helps to ensure that programs developed for the converter could be transported easily to the Macintosh and vice versa. The 68000's speed and rich instruction set were additional factors in its favor. An HP64000 Logic Development System was available making system development and documentation much easier and more orderly.

Moreover, the MC68000 was available as part of the MC68000 Educational Computer Board (hereafter referred to as the MC68000 ECB or, simply, as the ECB). This board contains, among other things: a 4MHz MC68000 microprocessor; a 32K ROM that supports the TUTOR assembler and debugger; a 32K RAM; two serial ports supporting data rates up to 9600 baud; one MC68230 Parallel Interface and Timer; and a 15 watt power supply (Moto. 1982, pp. 7.2-7.6). Use of this board had a distinct advantage in that building time for a prototype of the converter would be reduced considerably. The only hardware that had to be built would be the interface between the Z8530SCC and the MC68000 ECB.

In summary, the converter hardware was chosen to have the block structure shown in Figure 2. The transceiver and serial-to-parallel converter would be implemented using the Z8530SCC; the port controller would be implemented by the MC68000; and two 8-bit parallel ports would be implemented by the MC68230 Parallel Interface and Timer (or MC68230 PIT). The next issue was to decide on the software to be used.

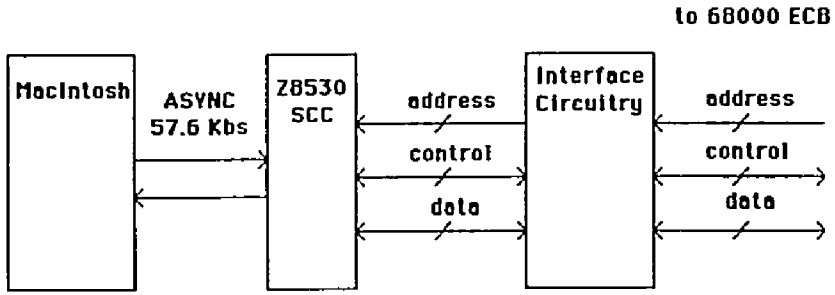

Figure 2. Block Diagram of the Hardware Realization of the Serial-to-Parallel Converter

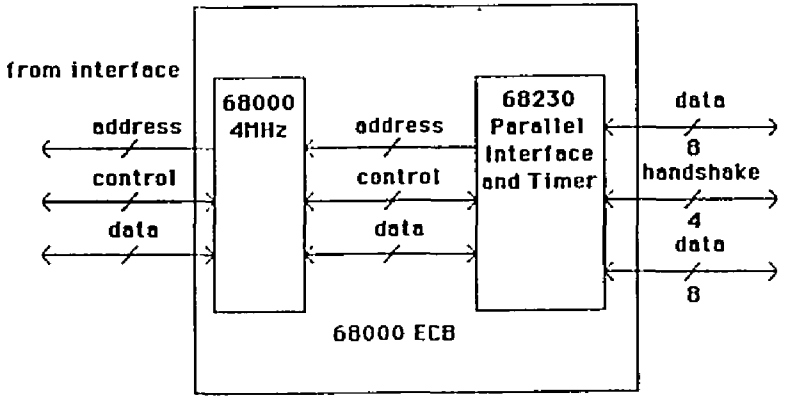

Figure 2 -- continued

Software Design of the Serial-to-Parallel Converter

The choice of the Z8530SCC and the MC68000 ECB to implement the converter presents a number of alternatives in the software design as both devices are extremely versatile and powerful. Some of these choices influence the hardware configuration also, and will be dealt with first.

The first such issue was raised by an interesting property of the Z8530SCC. In order to prevent possible metastable conditions, there must be a minimum delay between successive accesses to it. The formula (Adva. 1985, pp. 2.177-2.178) for deriving the length of this delay is:

required delay (in nsec.) = 6 x clock rate (in nsec.) + 200 nsec.

The clock rate for the MC68000 ECB is 4MHz. Using this clock as the converter's Z8530SCC clock the required delay between successive accesses is found to be:

delay = 6 x 250 nsec. + 200 nsec. = 1700 nsec.

Although the delay could be produced by hardware, it can also be enforced in the software. The principle reason for using this approach is that it is the one used in the Macintosh (Williams 1984, p. 36), and the software portability is increased by using the same approach in both programs. However the Macintosh's Z8530SCC clock is slower (only 3.7MHz, Williams 1984, p.36), and the required delay between successive accesses to it on the Macintosh is at least 2.25 usec. For the sake of consistency, a uniform 3 usec. delay is used in the software at both ends of the link.

Another software issue that affects the hardware design is the question of whether to use polling or interrupts to control program flow. To guarantee simplicity in the initial program design, it was decided to have the converter's MC68000 poll the Z8530SCC and the ports.

Now it is possible to focus on purely software issues in the converter's design. The first of these was the question of whether to use assembly language or a high-level language to write the Macintosh software. This question was easily settled by the lack of an assembly language development system for the Macintosh which forced the software to be written in a high-level language. Another reason for using the high-level language, however, was that if it were desired to eventually upgrade the converter's software to a Macintosh application program (a program that reproduces the Macintosh desktop, allows execution of user programs in the foreground while the converter's software ran in the background, and so forth), the upgrade could be more easily accomplished using one of the programming languages available for the Macintosh (Pascal, C, BASIC, etc.). Since the only high-level language available for the Macintosh at the University of Arizona during this time was BASIC it was chosen to implement the Macintosh software.

The choice of BASIC as the high-level language forced a speed restriction on the converter. It was originally desired to have the converter operate at 230 kilobits per second, however this leaves only:

$$\frac{1 \text{ x } 106}{2.3 \text{ x } 10^5} \text{ usec. x } 8 \frac{\text{bits}}{\text{byte}} = 32 \frac{\text{usec.}}{\text{byte}}$$

This small amount of time between successive bytes of a message combined with the 3 usec. minimum delay necessary between successive accesses to the Z8530SCC mentioned previously requires that all code accessing the Z8530SCC be carefully timed, a process more easily done in assembly language. In BASLIC, however, the interface to assembly language is quite complicated. All assembly code must be entered by hand into BASIC DATA statements as machine code and loaded into arrays at execution time. It was decided, therefore, to use the BASIC supplied file COM1 (Willen 1985, p.116) which allows access to the Macintosh's Z8530SCC at speeds of up to 57.6 kilobits per second using an asynchronous protocol. Using COM1 avoids the necessity of having to access the Macintosh's Z8530SCC directly which helps to simplify the Macintosh software.

Despite using a high-level language for the Macintosh software, it was decided to use assembly language to develop the code for the MC68000 ECB end of the converter. The motivation for using assembly language for the ECB end of the converter needs to be mentioned. Since it was decided to create a converter that could be realized as discrete hardware if so desired, the use of assembly language for the ECB software is called for, because it makes the process of conversion to an all hardware design easier. Microcode or discrete digital logic can be more easily developed from software hand written in assembler than from either a high-level language or from assembler generated by a compiler.

To summarize, the software of the Macintosh would be written in BASIC while the software for the ECB would be written exclusively in assembly

language. This assembly code would include special modifications to insure that there would be at least 3 usec. between successive accesses to the ECB's Z8530SCC. Finally, the BASIC file COM1 would be used to control the Macintosh's Z8530SCC, and to operate the serial port at 57.6 kilobits per second (see Figure 1).

Message Design of the Serial-to-Parallel Converter

The next issue to consider is the design of a set of messages for the converter. Although it was already decided to use an asynchronous protocol, the structure and types of messages to be sent between the Macintosh and the ECB still needed to be determined. The basic structure of the messages is shown in Figure 3. The one part of this protocol that needs further discussion is the structure of the port control byte.

The port control byte is used to determine what type of operation is to be performed on the register. Bit 7 of the byte determines whether a read or a write is to be performed. The remaining seven bits give the register number. This number identifies the various control and data registers of a given port chip, so , in general, corresponding to each port will be a series of register numbers. For example, bits 6 and 5 might be used to point to one of four MC68230 port chips, while the remaining five bits (4 trough 0) would point to one of the twenty-three registers of this chip. The idea behind this concept is to implement a "virtual" port within the Macintosh.

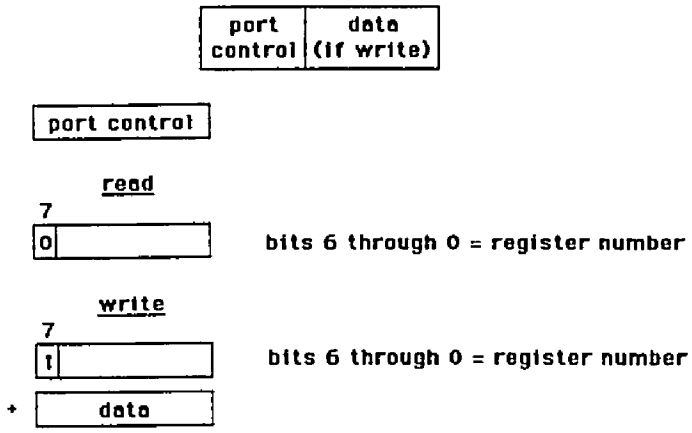

Figure 3. Protocol Used for Messages Sent Between the Macintosh and the Converter at 57.6 Kilobits Per Second

By a "virtual" port it is meant that the Macintosh would write to and read from the various port registers as if these registers were a part of the Macintosh hardware. The following routines would be used to accomplish this:

1. popen--initializes the Macintosh Z8530SCC for converter operation.
2. pread--sends a message to the ECB telling it to read the port register specified by the instruction, and return the data to the Macintosh.
3. pwrite--sends a message to the ECB telling it to write the data (included in the message) to the specified port register.
4. pclose--returns Macintosh's Z8530SCC to its original state.

An application program designed to use the converter need only call these routines as subroutines in much the same way that similar routines are used to control files and such hardware devices as the serial ports and the screen.

Finally, the flowcharts for these routines are shown in Figures 4, 5, 6, and 7. Flowcharts for the software at the ECB's end of the converter are shown in Figures 8 and 9. Several observations are warranted at this point. There are only two message types: read and write. Messages are initiated by the Macintosh, and executed sequentially with no new messages sent until the previous message is dealt with. The ECB runs as a slave to the Macintosh; simply waiting for each message sent by it, and then taking the appropriate action. These steps were taken to make the code for the ECB simpler with the advantages already mentioned in the previous section.

The next issue to be dealt with is how the ECB converts the port number sent to it by the Macintosh into the port's physical address. At the Macintosh end

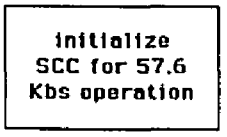

Figure 4. Flowchart of the Subroutine popen

Figure 5. Flowchart of the Subroutine pclose

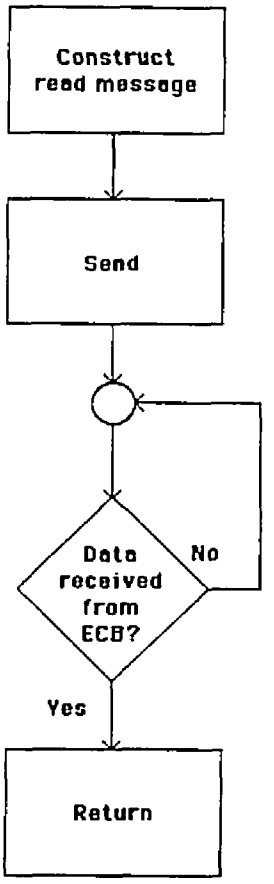

Figure 6. Flowchart of the Subroutine pread

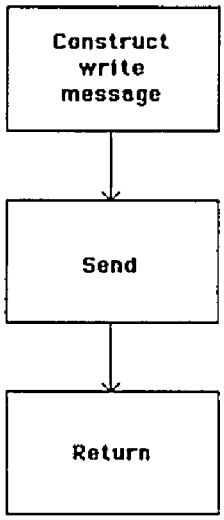

Figure 7. Flowchart of the Subroutine pwrite

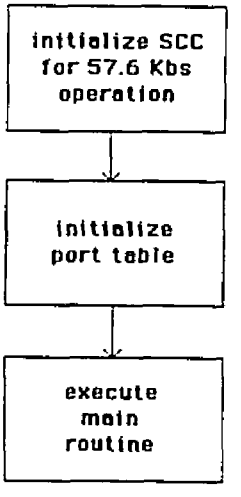

Figure 8. Overview of the MC68000 Software

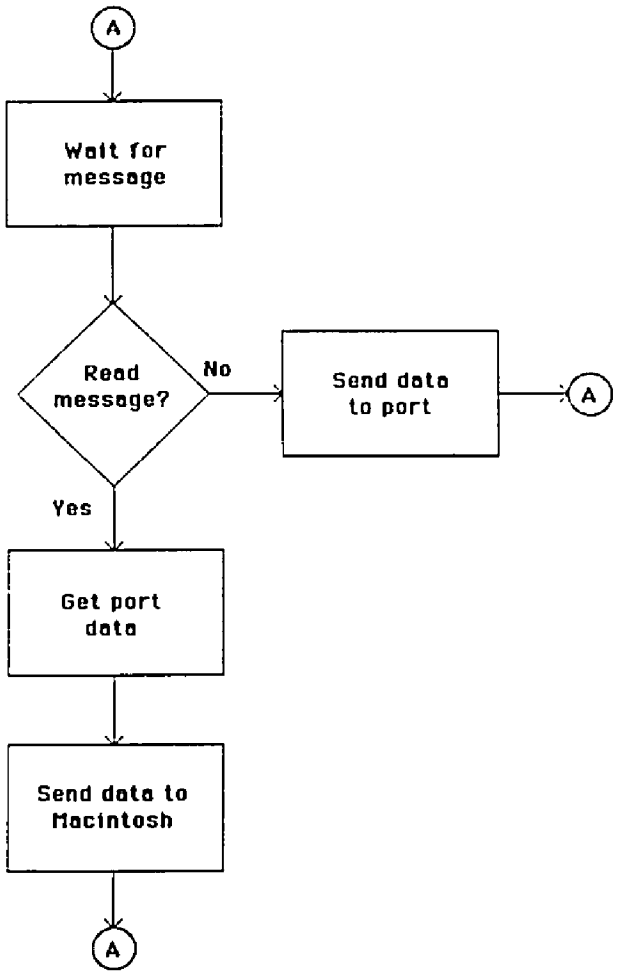

Figure 9. Flowchart of the MC68000's Main Program

of the link the driver program (the program calling popen, pclose, etc.) assigns numbers to the various port registers of interest as shown in Table 1. In addition, values have been assigned to some of the port's registers. In this particular case the values have been assigned to the MC68230 PIT's registers in order to implement a Centronix style interface (Moto. 1982, pp. 6.17-6.18). Meanwhile, at the ECB end of the link, the port number is hashed to the address of a table entry that contains the physical address of the port register. The hashing formula is shown below:

address of table entry = (4 x register number) + base address of table

The base address of the table is 900H. This information is summarized in Table 2. Each entry is four bytes long to hold the 24-bit addresses of the MC68000. The 7-bit register numbers allow 128 registers to be accessed. The Macintosh can now manipulate these registers as if they were local. In other words, the register numbers used by the Macintosh could be viewed as "pointers" to the physical registers of the MC68230 PIT located on the ECB. A message structure for a 230 kilobits per second version of the converter is described in Appendix A. With the message structure determined it was now possible to write the software for both ends of the converter.

Table 1. The Macintosh Register Numbers and Settings Used to Implement the Centronix Interface

Macintosh Register Number	MC68230 Register Name and Abbreviation	Register Setting for Centronix Interface (in hexadecimal)
0	Port General Control Register(PGCR)	00
1	Port Service Request Register(PSRR)	00
2	Port A Data Direction Register(PADDR)	FF
3	Port B Data Direction Register(PBDDR)	00
4	Port A Control Register(PACR)	60
5	Port B Control Register(PBCR)	A0
6	Port A Data Register(PADR)	
7	Port B Data Register(PBDR)	
8	Port Status Register(PSR)	

Table 2. The MC68000 Port Table Structure and Entries

Macintosh Register Number	MC68230 Register Name and Abbreviation	Address of Register's Entry	Physical Address of Register (in hexadecimal)
0	Port General Control Register(PGCR)	900	010001
1	Port Service Request Register(PSRR)	904	010003
2	Port A Data Direction Register(PADDR)	908	010005
3	Port B Data Direction Register(PBDDR)	90C	010007
4	Port A Control Register(PACR)	910	01000D
5	Port B Control Register(PBCR)	914	01000F
6	Port A Data Register(PADR)	918	010011
7	Port B Data Register(PBDR)	91C	010013
8	Port Status Register(PSR)	920	01001B

CHAPTER 3

BUILDING OF THE SERIAL-TO-PARALLEL CONVERTER

Hardware

The building of the converter's hardware can now be discussed. Schematics for the converter are shown in Figures 10 and 11. Figure 10 details the hardware used to interface the MC68000 ECB and the Z8530SCC, and Figure 11 shows the hardware used to connect the ECB and the Z8530SCC to the Macintosh. In addition, the parts for the interface are shown in Table 3. Before discussing these areas in detail, a few general remarks can be made. First, the ECB's extension bus was used to interface the Z8530SCC to the MC68000 (Moto 1982, pp. 7.10-7.12). The pin connections between the Z8530SCC's and the ECB's data busses are described in Table 4. And second, the modem port (Willen 1985, p.117) of the Macintosh was used for the connections between it and the Z8530SCC.

Referring to Figure 10 and Table 3 the address decoding and the read and write operation of the interface will now be discussed. The address decoding is accomplished by chip E, which is enabled when *AS is low and when both address lines A_{15} and A_{14} are high. Lines A_{13} and A_{12} are used to select one of four outputs. Output 0 is used as the chip select signal for the Z8530SCC, and it is also fed to chip J which generates a *DACK signal for the MC68000. This signal appears 500 nsec. after the assertion of *CS and the MC68000 will

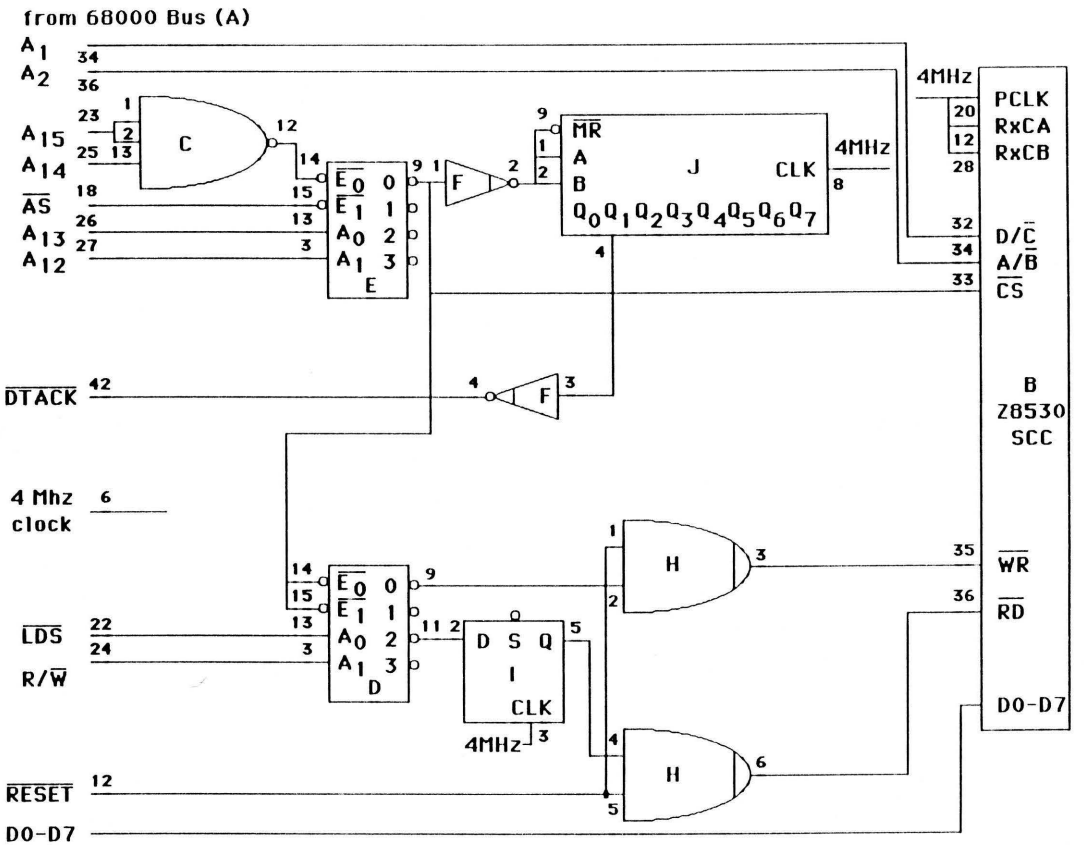

Figure 10. Schematic of Interface's Read and Write Circuitry

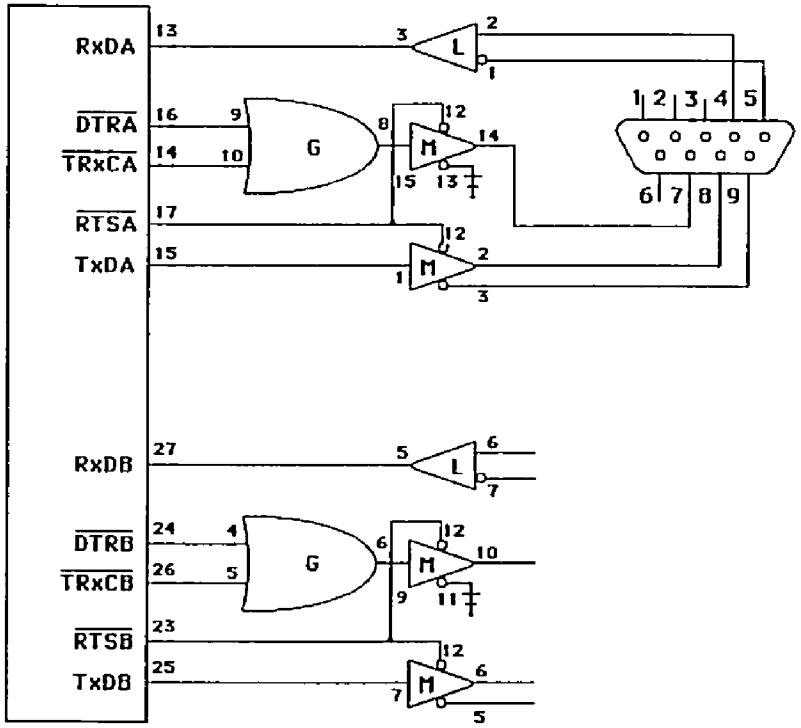

Figure 11. Schematic of the Connection Between the Z8530SCC and the Macintosh

Table 3. The List of Parts Used in the Schematic of the Serial-to-Parallel Converter

Identifying Letter	Part Type
A	MC68000 Extension Bus
B	Z8530 SCC
C	LS7410
D	LS74155
E	LS74155
F	LS7405
G	LS7432
H	LS7408
I	LS7474
J	LS74164
K	LS74164
L	LS2932
M	LS2930

Table 4. The Wiring List of the Connections Between the MC68000 Extension Bus and the Z8530SCC

Data Line Number	Pin Number	
	MC68000 Extension Bus	Z8530 SCC
D0	21	40
D1	14	1
D2	4	39
D3	2	2
D4	1	38
D5	3	3
D6	5	37
D7	7	4

respond to *DACK by lowering *AS 500 nsec. later. Therefore, each read or write operation lasts approximately one microsecond.

Additionally, the output of chip E is also used to enable chip D, which is used to assert either the read or the write input of the Z8530SCC. This is accomplished by using the MC68000's *LDS and R/*W as inputs to the decoder. Output 0 asserts the write, while output 2 asserts the read. The flip flop I is used to delay the assertion of the read until 250 nsec. after the assertion of the chip select to the Z8530SCC (the chip's specifications require at least an 80 nsec. delay, Adva. 1985, pp. 2.173-2.174). A similar delay is not necessary for the write signal, because the MC68000 automatically delays the assertion of *LDS until 250 nsec. after the assertion of *AS (Pren. 1982, p. 40). The MC68000 address line A_1 is used to select either the data or control registers,and A_2 is used to select either channel A or B of the Z8530SCC. This means that the following addresses can be used to select the various Z8530SCC registers:

possible addresses	addresses actually used	register name
1100 XXXX XXXX 0001	C001	channel B control
1100 XXXX XXXX 0011	C003	channel B data
1100 XXXX XXXX 0101	C005	channel A control
1100 XXXX XXXX 0111	C007	channel A data

However, only channel A is used by the converter's software. Additionally, the read and write signals are AND'ed with the ECB's *RESET line to allow the possibility of a hardware reset of the Z8530SCC.

The next schematic, Figure 11, shows the connections between the ECB and the Macintosh. The lines RxD+, RxD-, TxD+, and TxD- are used to implement the RS422 connection using a 9-wire twisted pair cable and a 9-pin D-

type plug. These lines are connected from the Z8530SCC to the corresponding Macintosh lines through the 29LS30 quad differential driver and the 29LS32 quad differential receiver. The RS422 connection is used for the possible future extension of the converter to 230 kilobits per second, and to cut down on errors due to noise.

Software

The software produced for the converter will now be discussed. The flowcharts in Figures 4 through 9 describing the communications protocol also describe the operation of the software written to implement it. The Macintosh half was written in BASIC, while the ECB half was written in the MC68000 assembly language.

A discussion of the main program is now in order. This program was written on the Macintosh, and calls the other Macintosh routines (popen, pclose, pread, and pwrite) as subroutines. The main routine is divided into two parts as shown in Figure 12. The first part initializes the Macintosh half of the port table (see Table 1), while the second part runs the Benchmark driver program. Figure 13 shows a detailed flowchart of the Benchmark program while Figure 14 shows the hardware configuration necessary to run it. The program functions in the following manner. Data is output through port A of the MC68230 PIT. One bit of the output byte is then sent to port B. The byte is then read from port B, the data is inverted, and the byte is resent through port A. Although the MC68230's status registers are read and written to as they would be if a fully-interlocked handshake were used, no checking of the input handshaking lines is done,

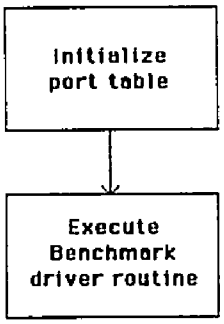

Figure 12. Overview of the Benchmark Program

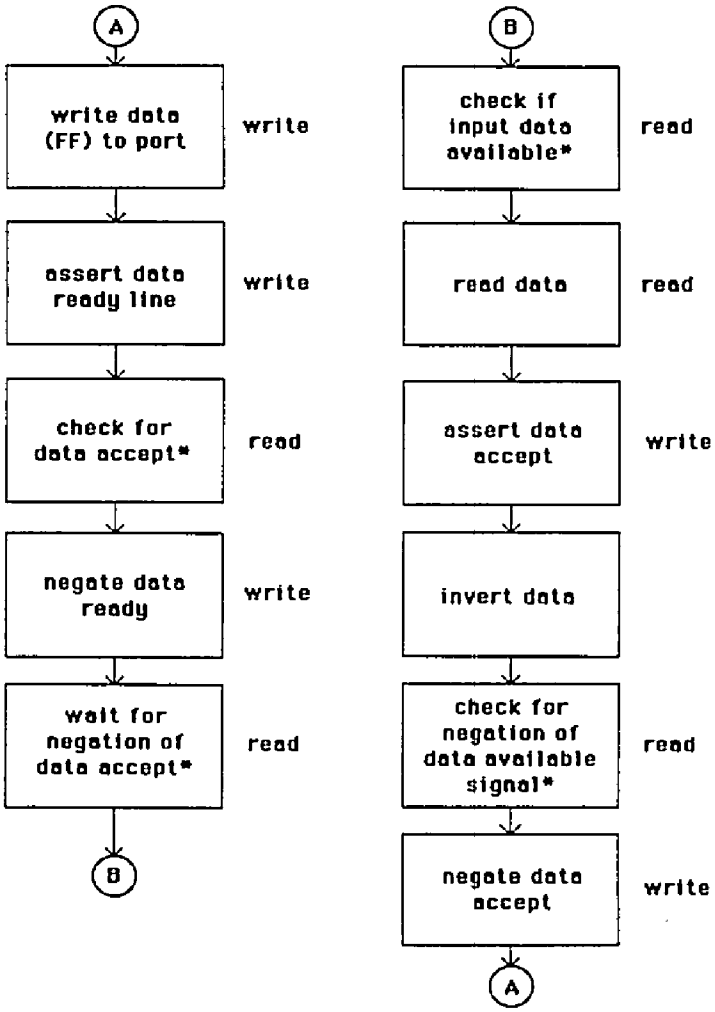

Figure 13. Flowchart of the Benchmark Routine

*This program assumes that input data is always available when checked, and that output data has always been accepted when checked

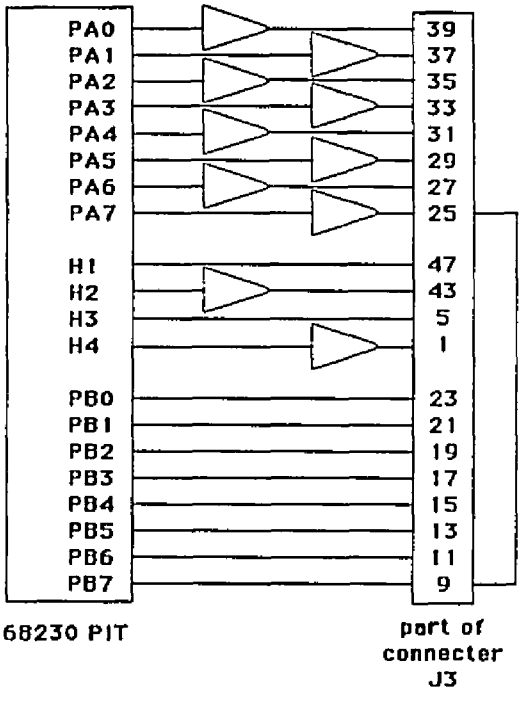

Figure 14. Hardware Configuration Necessary to Run the Benchmark Program

because all input and output data is assumed to be immediately available. The purpose of this program is to acquire an idea of the maximum speed that the program can input and output data. By hooking an oscilloscope to either pin 9 or 25 of the MC68230 PIT a square wave should be observable. The frequency of the wave indicates the maximum speed that data can be input and output through the MC68230 under the Macintosh's control. Since running a Centronix printer requires that one byte of data be output to the printer, and a byte of data indicating the printer's status be input to the Macintosh (Moto. 1982, p. 6.17), this program will also give an indication of the number of printers that can be driven simultaneously by the Macintosh. This concludes the discussion of the building of the converter.

CHAPTER 4

TESTING OF THE SERIAL-TO-PARALLEL CONVERTER

Hardware

While building the converter hardware and programming its software, it was also necessary to begin the testing of the various components of the converter. First, there will be a discussion of the hardware tests performed during the various stages of building the converter. The first area tested was the timing of the write and read operations between the Z8530SCC and the ECB. The results of these timing tests are shown in Figures 15 and 16. The most important times to consider were the *CS (chip select) to the *R (read) or *W (write) setup times of 80 nsec. (Adva 1985, pp. 2.173-2.174). In the case of the *CS to *W setup time a delay of 250 nsec. is achieved because the *LDS (lower data strobe) signal of the MC68000 (which helps toggle the *W line of the Z8530SCC) is not asserted until one clock period after the assertion of the MC68000's *AS (address strobe) line (Pren. 1982, p. 40). The *CS to *R delay, however, is not automatically provided by the *LDS line of the MC68000, but must be provided by the flip-flop labeled I in the schematic of the previous chapter (Figure 10). The delay until the read data is valid and the length of time the data must be held until it is valid are also met by this interface (Adva. 1985, pp. 2.173-2.174). Once the timings had been guaranteed, it became possible to begin operating the ECB half of the converter.

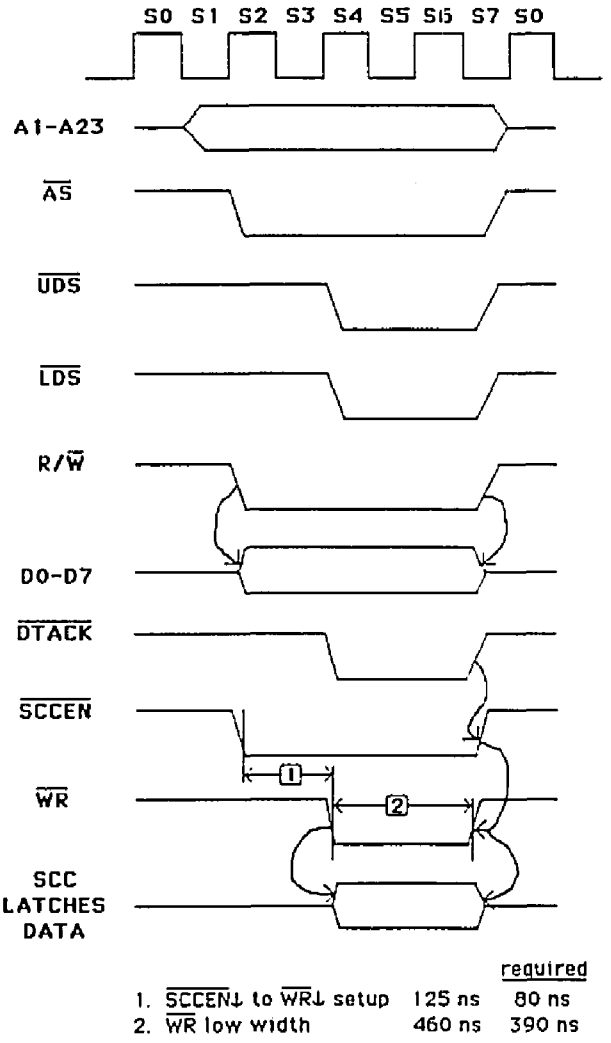

Figure 15. Z8530SCC Write Cycle Timing Diagram

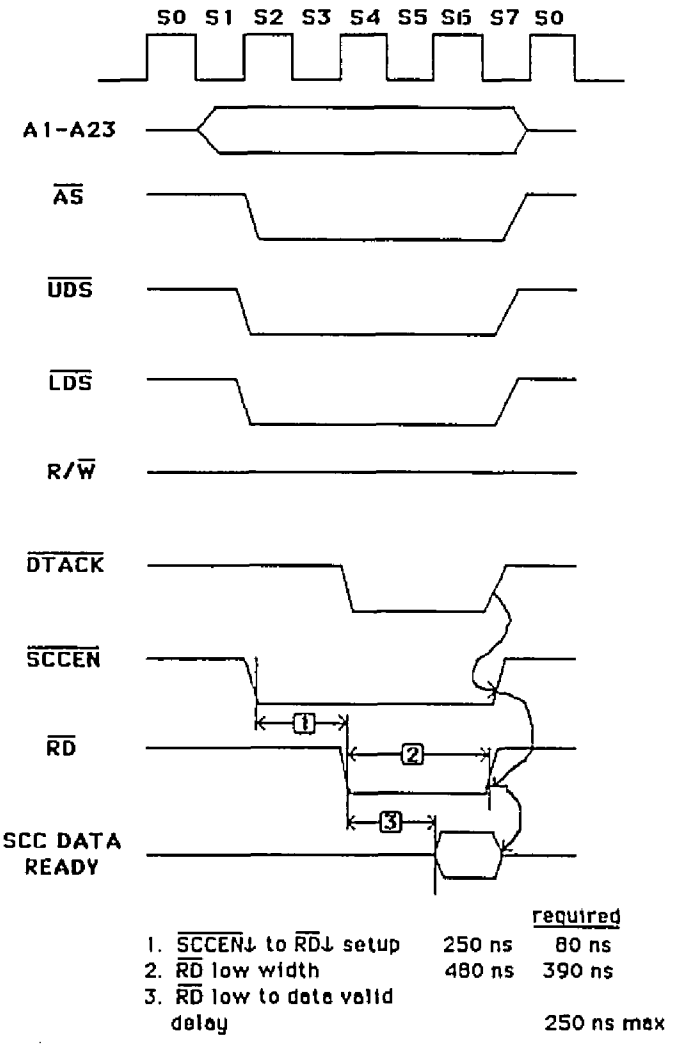

Figure 16. Z8530SCC Read Cycle Timing Diagram

The next step in the testing was to connect the transmit and receive lines of the ECB's end of the converter to one another to produce a closed loop with the ECB talking to itself. The principle reason for doing this was that the HP64000 Logic Development System made programming modifications easier to do on the ECB than on the Macintosh, and allowed various tests of the hardware and software to be performed quickly.

The first problem encountered was that the connection between the data busses of the ECB and the SCC were faulty. A number of the data lines were frozen at the high logic level and could not be forced to zero. To solve this problem a program called BR_TEST was developed in order to check the data bus connections. A flowchart of the BR_TEST program is shown in Figure 17. Basically, this program tries to write 00H to the baud rate counter of the SCC, and then tries to read back this data. If a value other than 00H is received, then the bus connection is faulty and must be corrected.

Once the data bus worked reliably, it was possible to test the ECB's *RESET line to see if it reset the SCC as planned (see schematic of Figure 10). The SCC was set to operate in the high-speed synchronous mode (SDLC at 230 kilobits per second), and the *RESET line was asserted. The SCC was found to have reverted to its reset state (Zilo. 1983, p. 3.9), therefore, the test was successful. With the hardware tests completed successfully, it was now possible to begin testing the software.

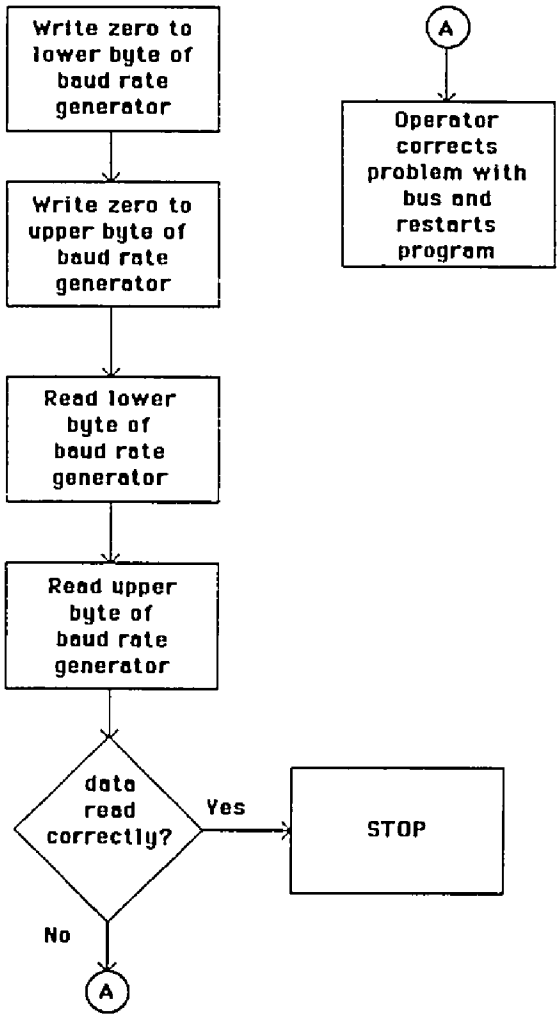

Figure 17. Flowchart of the Routine BR_TEST

Software

Once the hardware worked correctly, and it was possible to reliably read and write to the SCC, software for the ECB end of the converter could be written and tested. As mentioned in the previous section, the ECB was hooked up in loopback mode for the first series of test programs.

The first program written, however, operated the SCC at 9600 baud in order to acquire skill at using the SCC. A flowchart of this first program is shown in Figure 18. After the successful running of the 9600 baud loopback program, the speed of transmission was increased to 230 kilobits per second. This program had essentially the same structure as the 9600 baud program with the exception that the speed was higher. Even though no termination impedances were used in the loop, it was possible to send messages of 2 kilobytes reliably.

Now that the ECB loopback program had been run at both 9.6 and 230 kilobits per second, the final step involved the connecting of the Macintosh to the ECB, and getting the Benchmark program to operate successfully. Referring again to Figure 13 to the right of the Benchmark program's flowchart is a description of the type of message that must be sent. With this information it is possible to make an estimation of the Benchmark program's speed (a similar calculation is made in Appendix A for a 230 kilobits per second version of the converter). The results of the calculations are given in Table 5. An estimated speed of 288 cycles per second is obtained, where one cycle consists of one pass through the Benchmark program. This is quite slow; multiple Centronix printers could not be driven by the Macintosh. Therefore, it was decided to make no

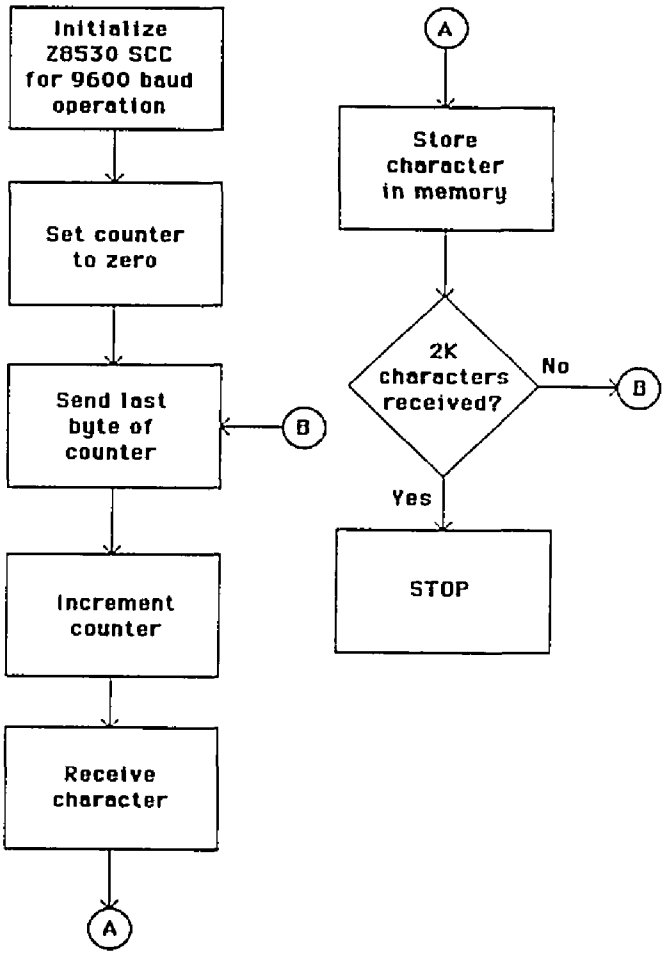

Figure 18. Flowchart of the 9600 Baud Loopback Program

Table 5. The Cycle Time for the Benchmark Program Operating at 57.6 Kilobits Per Second

One cycle consists of:

5 reads
5 writes

Each read message contains the following message types:

Message Type	Number of Bytes
1 read from Macintosh	1
1 write from ECB	2

Each write message contains the following message types:

Message Type	Number of Bytes
1 write from Macintosh	2

The total number of bytes in these message types is equal to:	5
The total number of bytes in 5 read and 5 write messages is equal to:	25
The total amount of time one cycle takes (where it takes 138 usec to send one byte of data at 57.6 Kbs) is equal to 25 bytes x 138 usec or:	3450 usec
The total number of cycles that can be completed in one second is equal to:	288 cycles per second

attempt to connect the Macintosh and the ECB at this point (although the software for both machines had been written); but, rather, to try to find some way to improve the converter's transfer rate. Some of these methods will be discussed in the final chapter.

CHAPTER 5

INITIALIZING THE CONVERTER AT SYSTEM STARTUP

Two more issues that need to be dealt with in this paper are how to initialize the ECB at power on and how to load the ECB's half of the converter program to it at system startup time. Presently, a terminal is connected to Port 1 of the ECB, and this terminal is used to manually load the ECB's converter program to it, and to start the program running. The first step is necessary, because the ECB has no static RAM, so the program is lost whenever the power is shut off. The second step is needed, because the ECB idles in the TUTOR firmware when power is first supplied to it, and a command must be sent to the ECB to get it to leave the TUTOR program, and begin executing a user program. This chapter describes the design of the hardware and software needed to allow the Macintosh to upload the ECB's program to it, and to start it operating at system power on.

The first modification needed is the connection of the Macintosh's modem port not only to the Z8530SCC on the ECB, but also to the ECB's Port 1. This modified connection is shown in Figure 19. This change would allow the Macintosh to be viewed as a terminal by the ECB. It would then be possible for the Macintosh to be used to develop programs on the ECB using the TUTOR firmware. A flowchart of the Macintosh terminal program is shown in Figure 20.

This program contains two subroutines. The first, download, is described in Figure 21. This subroutine would be used to transport a program developed on

the ECB to the Macintosh using Motorola's S-record message format (Moto. 1982, pp. A.1-A.4). Once a program had been stored in a Macintosh BASIC file, the subroutine upload can then be used to transport the contents of the Macintosh's BASIC file to the ECB in S-record format. A flowchart of upload is shown in Figure 22.

Once these subroutines worked correctly, the subroutine upload could then be used in an automated procedure, pinitialize, which would be used at system startup time to upload the ECB's converter program to it, and start it operating. The subroutine pinitialize would be called by a main program in the same manner as popen, pclose, and so forth. Pinitialize is described in Figure 23. With these programs in place the converter would become a stand-alone device using only the Macintosh and the ECB.

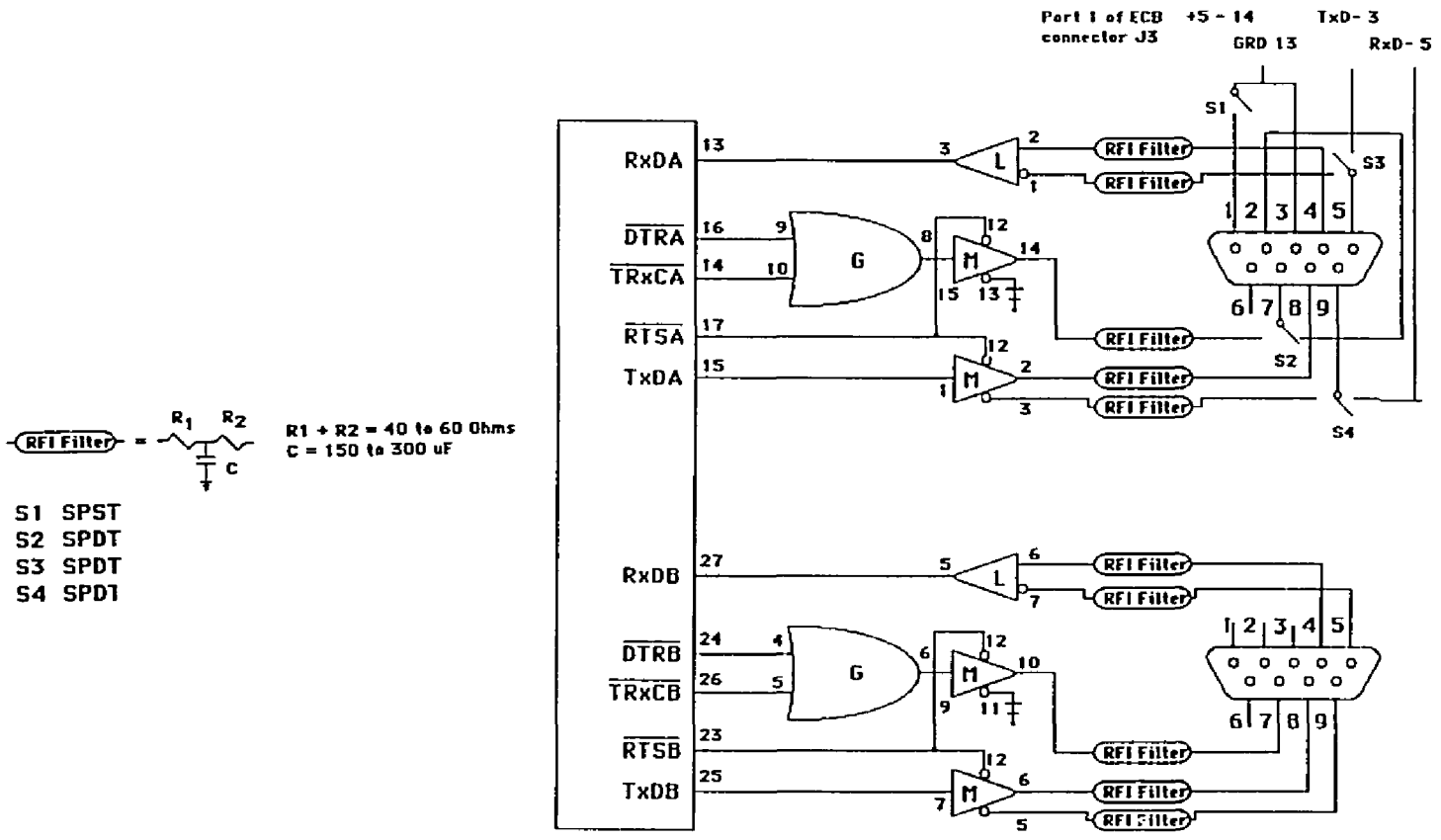

Figure 19. Modifications that Allow the Converter to be Connected to Port 1 of the ECB and that Include Termination Impedances

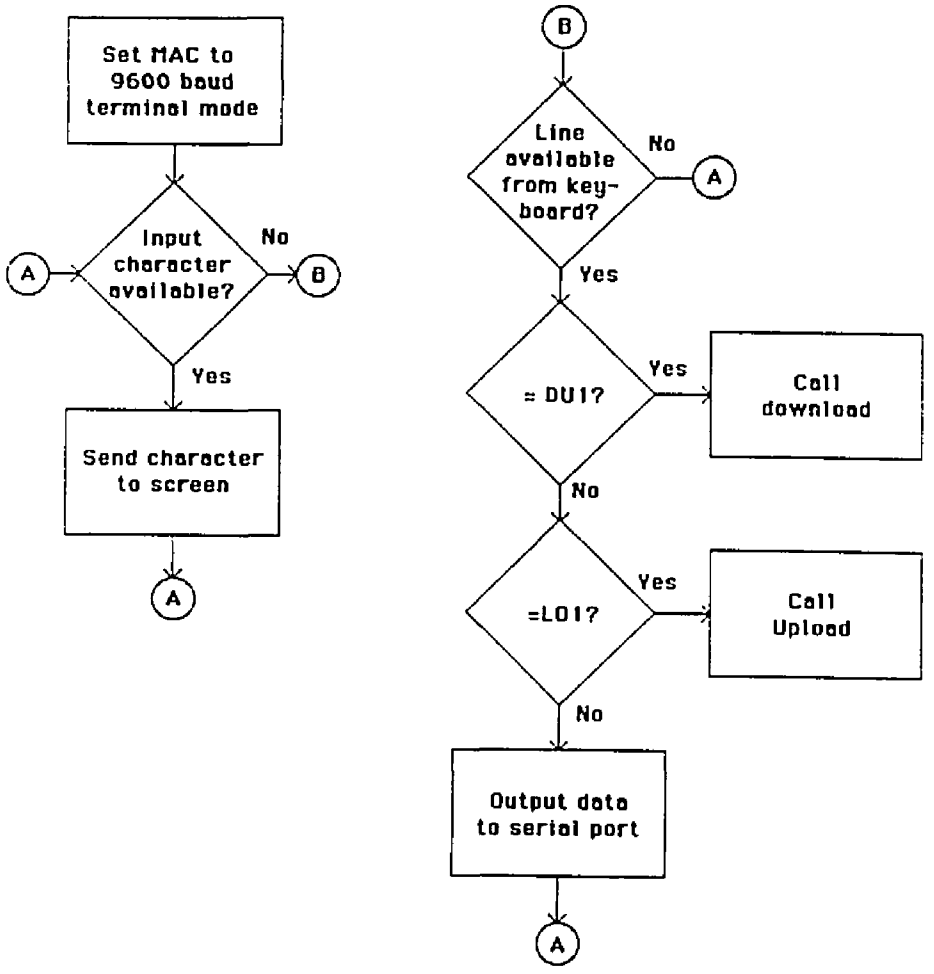

Figure 20. The Main Body of the Macintosh Terminal Program

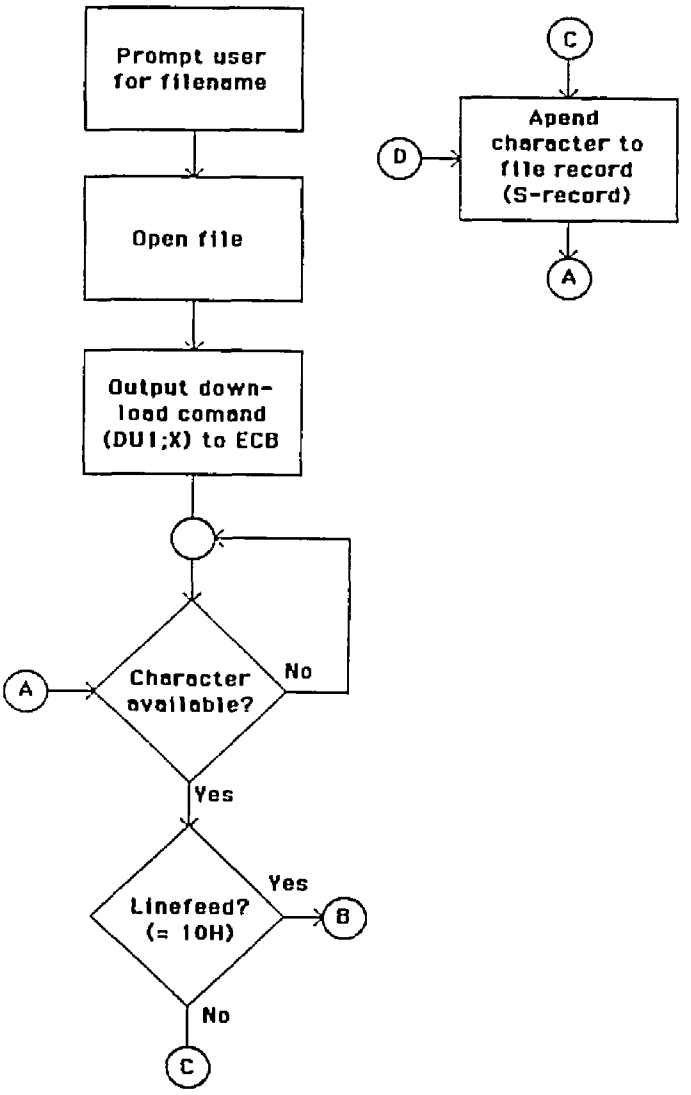

Figure 21. Flowchart of the Subroutine Download

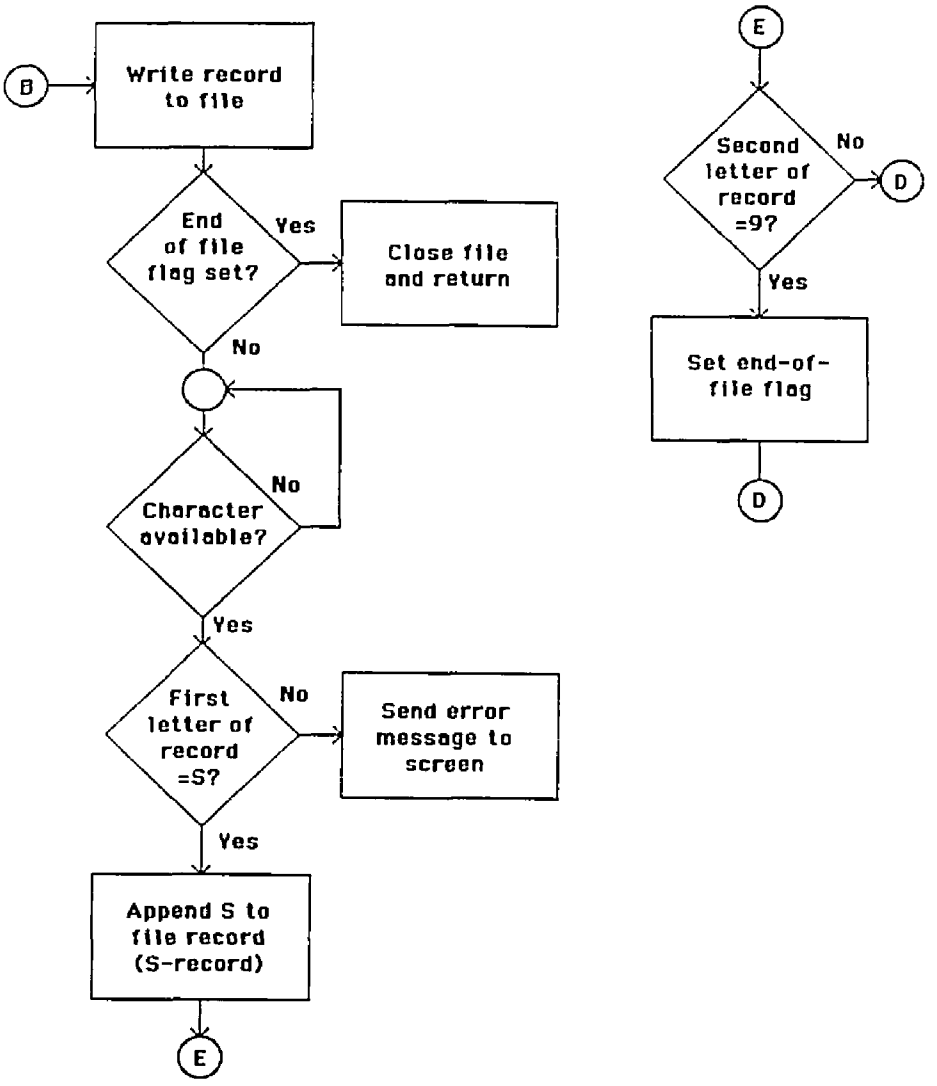

Figure 21 -- continued

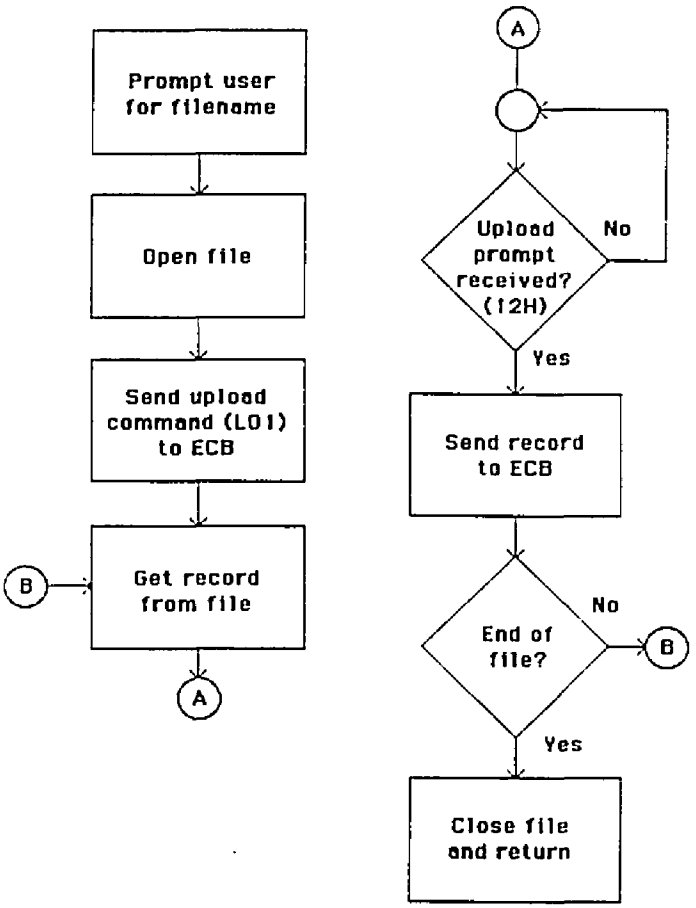

Figure 22. Flowchart of the Subroutine Upload

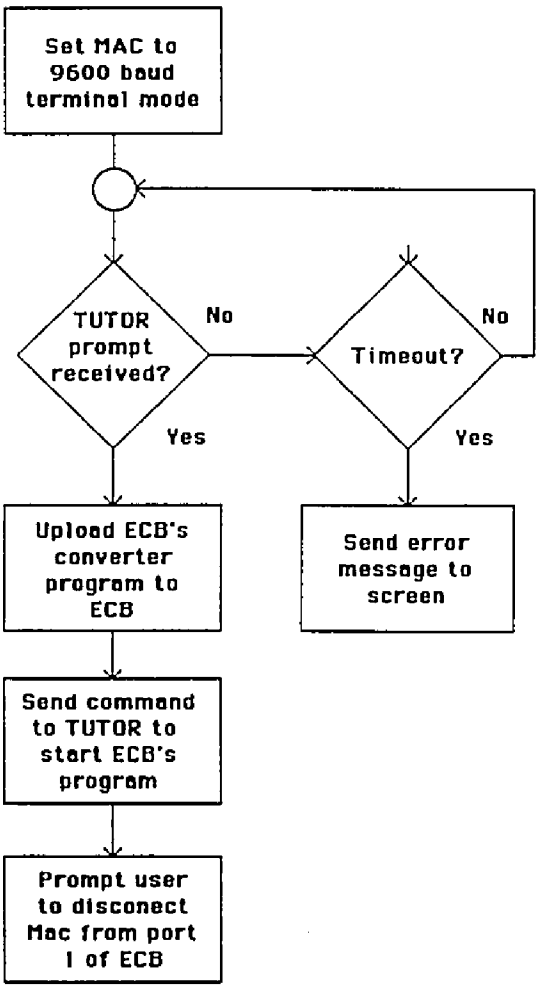

Figure 23. Flowchart of the Subroutine Pinitialize

CHAPTER 6

CONCLUSIONS

Current System Limitations

As noted in Chapter 4, the present design, although workable, is much too slow to be of practical value. With a throughput of only 288 cycles (i.e. bytes) per second the device cannot run multiple parallel ports. In order for it to run multiple parallel ports, some method must be found to increase its speed.

Recommended Future Work

The principle technique for improving the converter's speed would be to have the input and output handshaking controlled by the ECB end of the converter, rather than by the Macintosh as is done now. An example of the Benchmark routine with the ECB controlling the handshaking is shown in Figure 24.

With the handshaking controlled by the ECB the only responsibilities of the Macintosh would be to send and receive data. Since the rate of transmission is 57.6 kilobits per second (or 7200 bytes per second), this fact implies that the Macintosh will be sending 3600 bytes per second and receiving 3600 bytes per second. In addition, since in this configuration only write messages are sent by the Macintosh and ECB, and since a write message requires 2 bytes of data (a port number and the data to be sent), the Macintosh would be sending and receiving

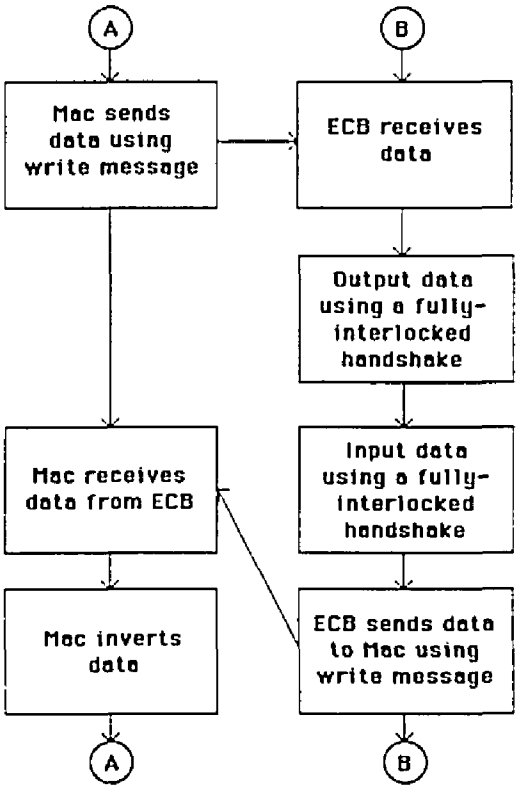

Figure 24. Benchmark Routine with the Handshaking Being Handled by the ECB Rather than the Macintosh

1600 messages per second; more than enough throughput to control multiple parallel ports.

Another approach to increasing the speed would be to make the connection full-duplex rather than half. Of course the Benchmark program could no longer be run, because it requires the Macintosh to wait for a received message after it transmits one (i.e. half-duplex operation). However, in full-duplex operation the Macintosh could send and receive 7200 bytes per second or 3600 messages per second. Again, this rate of transmission makes it possible for the Macintosh to control multiple parallel ports. These changes in the converter's software would make it practical to use.

APPENDIX A

PROPOSED SOFTWARE DESIGN OF A SERIAL-TO-PARALLEL CONVERTER OPERATING AT 230 KILOBITS PER SECOND

In this appendix the design of a serial-to-parallel converter that operates at 230 kilobits per second using an SDLC style protocol will be discussed. The converter will operate in SDLC point-to-point mode using the protocol shown in Figure 25. At the Macintosh end the same four subroutines popen, pclose, pread, and pwrite described in Chapter 2 would be used. The flowcharts for these routines are shown in Figures 26 through 29. The software for the ECB end of the converter is shown in Figures 30 and 31. These programs are more complex than those in Chapter 2, since error detection and control functions are now included in the software.

The software can be run using the Benchmark program described in Chapter 3, and tested for speed as was done in Chapter 4 for the 57.6 kilobits per second software. An estimation of the speed can also be computed as was done in Chapter 4. The results of this estimation are described in Table 6. The outcome is quite disappointing; a throughput of only 152 cycles (i.e. bytes) per second is obtained. This is even slower than the 57.6 kilobits per second result. Considering the complexity of this software in comparison with the 57.6 kilobits per second software, there was no value in trying to implement the 230 kilobits per second software. Clearly, however, it is necessary to have the ECB rather than the Macintosh control the handshaking, and to make the connection full-

duplex rather then half-duplex, as was mentioned in the conclusion of this paper. In addition a more sophisticated protocol could be used than the simple stop-and-wait protocol described here. These changes may speed up the converter sufficiently to make it practical to use.

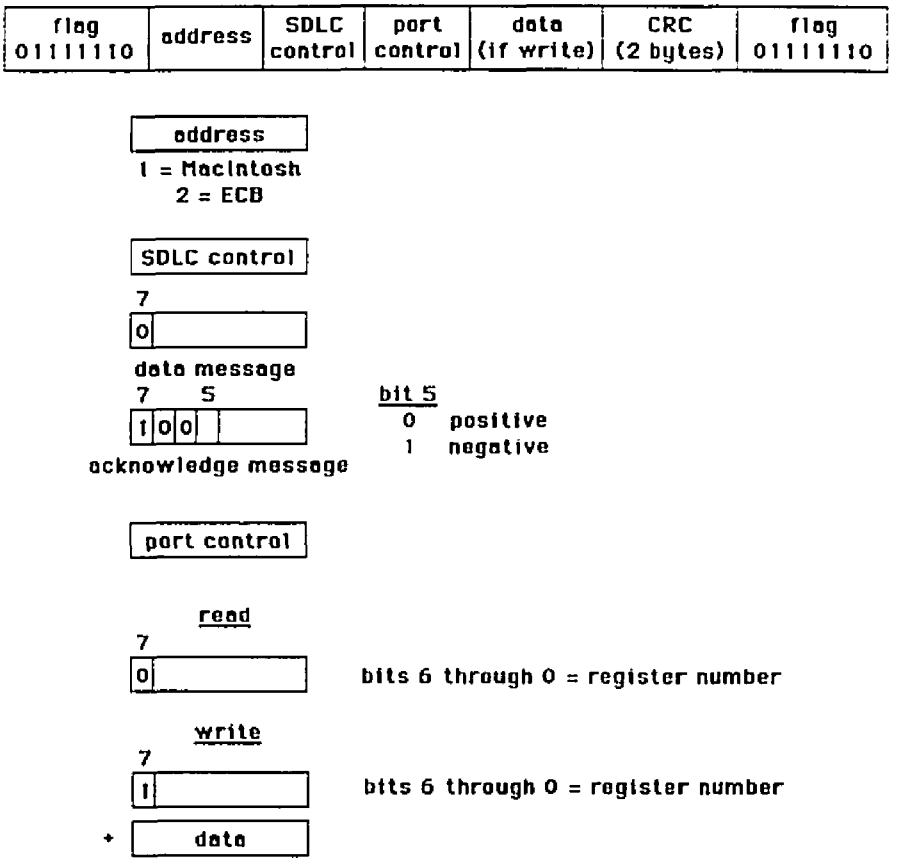

Figure 25. Protocol for Messages Sent Between the Macintosh and the Converter at 230 Kilobits Per Second

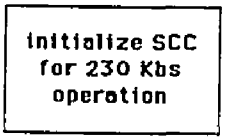

Figure 26. Flowchart of the Subroutine popen for the 230 Kilobits Per Second Version of the Converter

Figure 27. Flowchart of the Subroutine pclose for the 230 Kilobits Per Second Version of the Converter

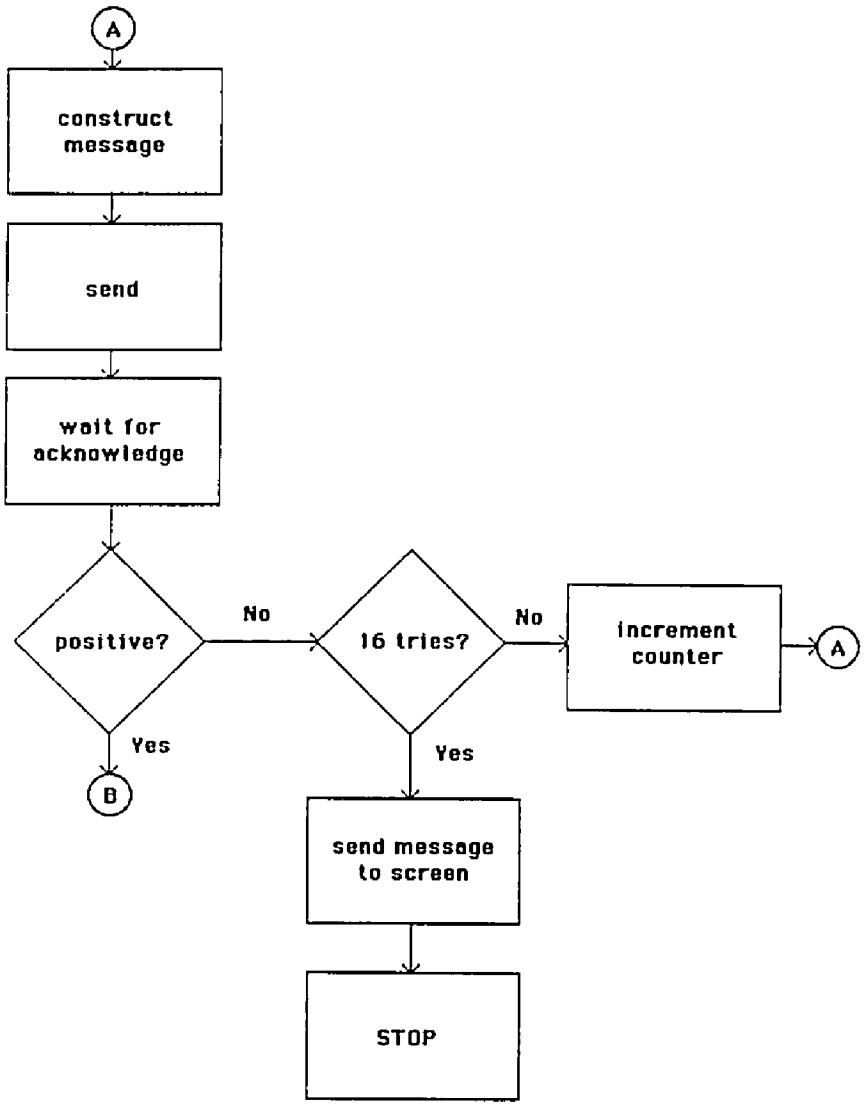

Figure 28. Flowchart of the Subroutine pread for the 230 Kilobits Per Second Version of the Converter

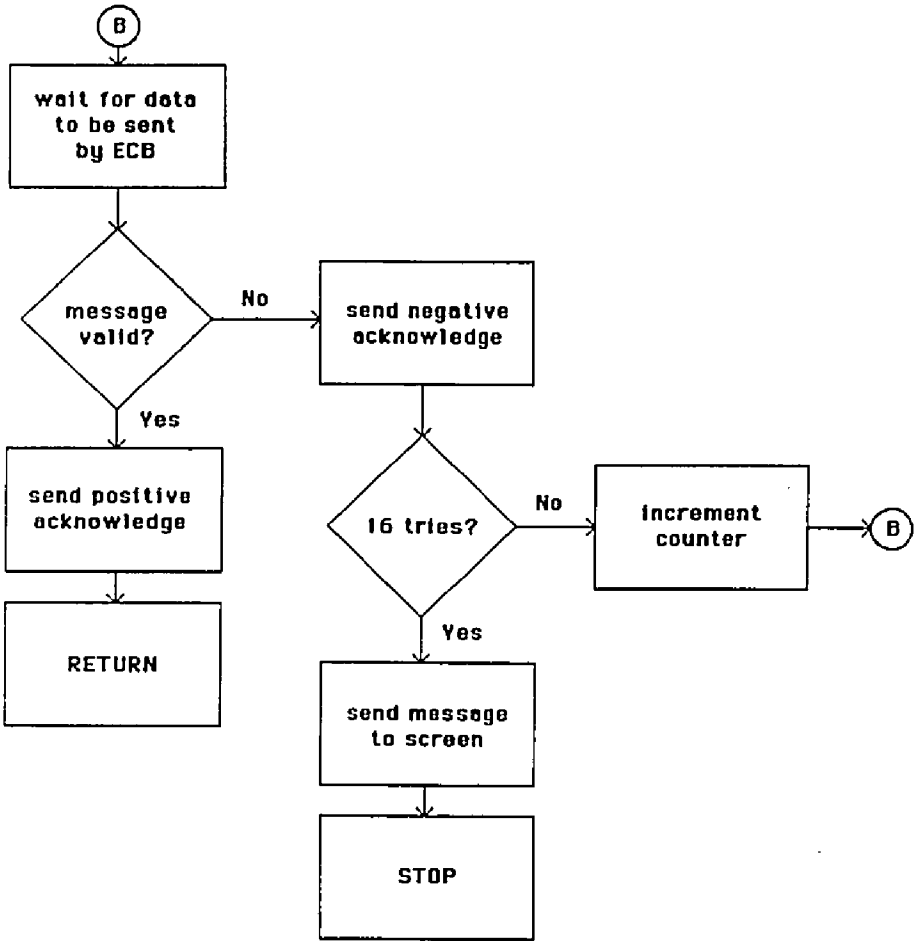

Figure 28 -- continued

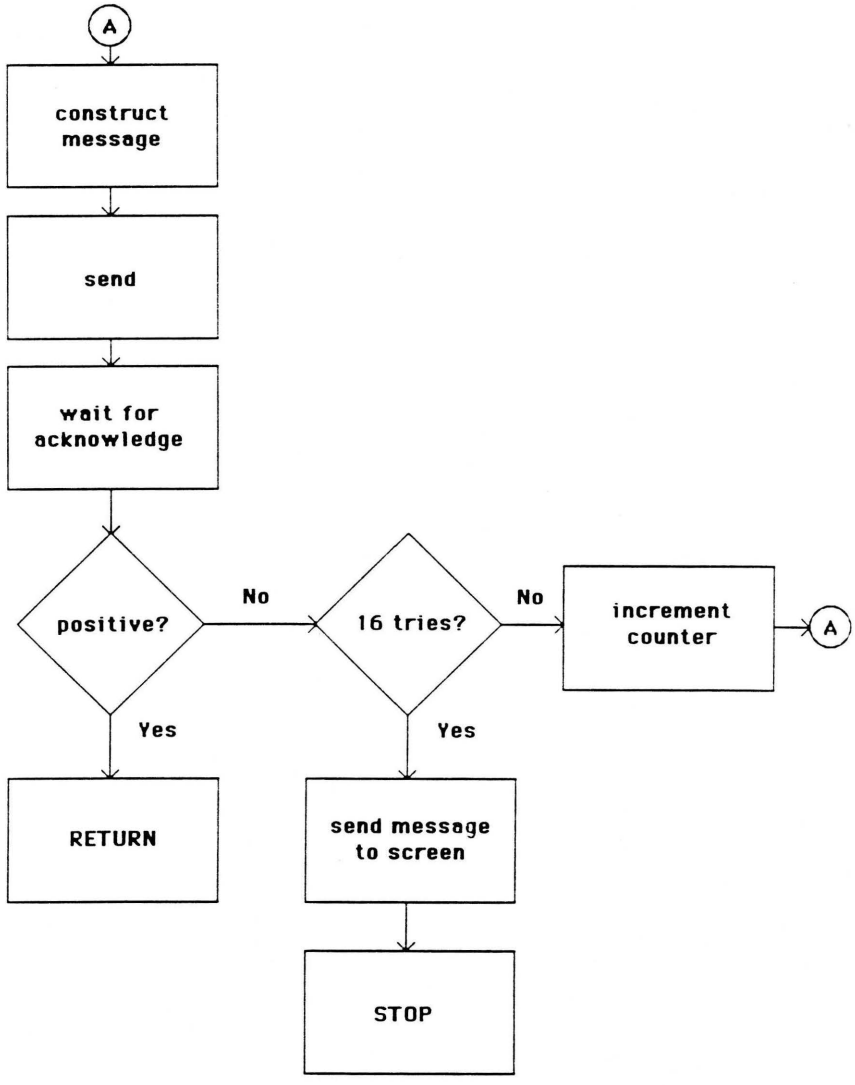

Figure 29. Flowchart of the Subroutine pwrite for the 230 Kilobits Per Second Version of the Converter

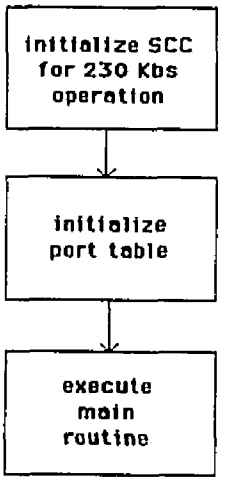

Figure 30. Overview of the MC68000 Software for the 230 Kilobits Per Second Version of the Converter

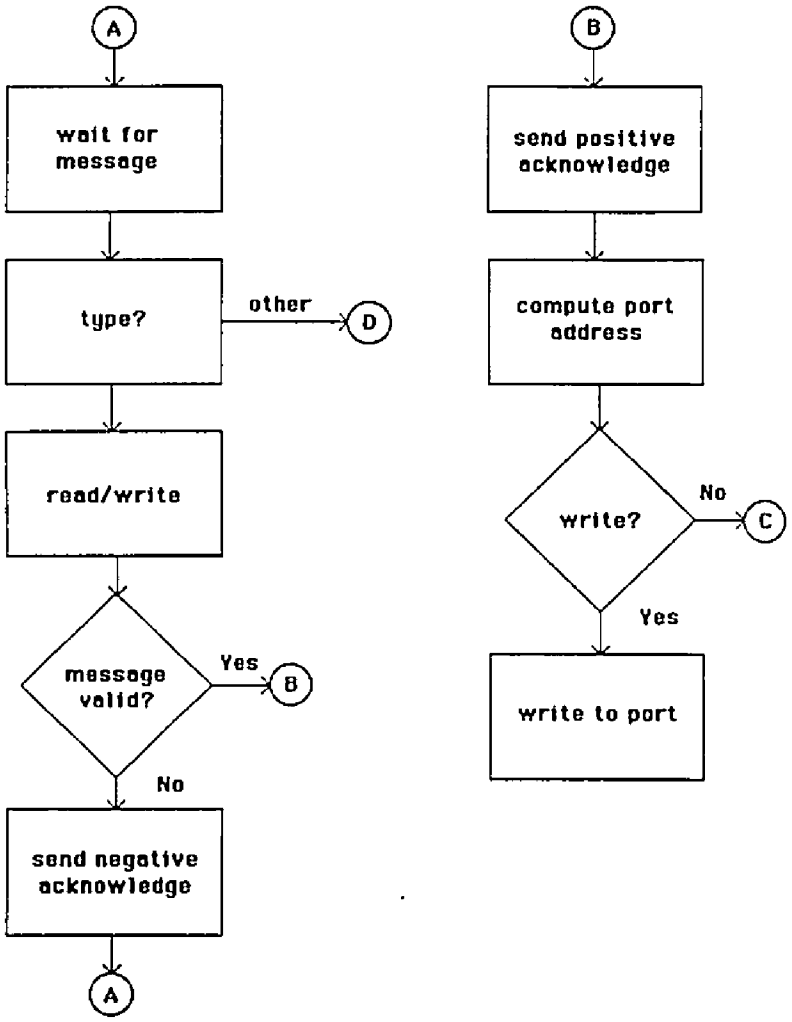

Figure 31. Flowchart of the MC68000's Main Routine for the 230 Kilobits Per Second Version of the Converter

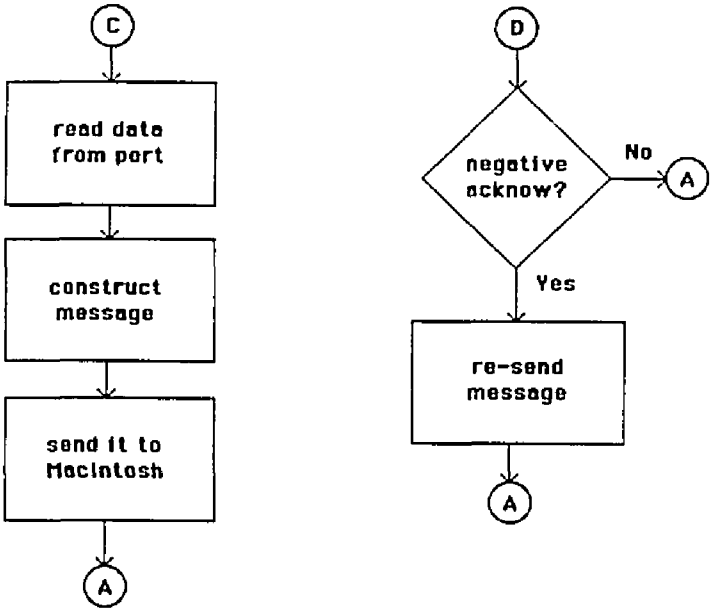

Figure 31 -- continued

Table 5. The Cycle Time for the Benchmark Program Operating at 57.6 Kilobits Per Second

One cycle consists of:

5 reads
5 writes

Each read message contains the following message types:

Message Type	Number of Bytes
1 read from Macintosh	7
1 acknowledge from ECB	6
1 write from ECB	8
1 acknowledge from Macintosh	6

Each write message contains the following message types:

Message Type	Number of Bytes
1 write from Macintosh	8
1 acknowledge from ECB	6

The total number of bytes in these message types is equal to:	41
The total number of bytes in 5 read and 5 write messages is equal to:	205
The total amount of time one cycle takes (where it takes 32 usec to send one byte of data at 230 Kbs) is equal to 205 bytes x 32 usec or:	6560 usec
The total number of cycles that can be completed in one second is equal to:	152 cycles per second

REFERENCES

Advanced Micro Devices, Inc. MOS Microprocessors and Peripherals. Sunnyvale, California. 1985.

Guterl, Fred. "Design Case History: Apple's Macintosh," IEEE Spectrum. Vol. 21, No. 12 (Dec. 1984).

Motorola, Inc. MC68000 Educational Computer Board User's Manual. 2nd Ed. Tempe, Arizona. 1982.

Prentice-Hall, Inc. 16-Bit Microprocessor User's Manual. 3rd. Ed. Englewood Cliffs, New Jersey. 1982.

Willen, David C. Macintosh Programming Techniques. Indianapolis, Indiana. Howard W. Sands & Co., Inc., 1985.

Williams, Greg. "The Apple Macintosh Computer," Byte. Vol.9, No.2 (Feb. 1984).

Zilog Corporate Publications. Z8030/8530 SCC Serial Communications Controller Technical Manual. Campbell, California. 1983.