

Lab #6

Character I/O

PURPOSE

This lab introduces the low-level details of character input/output on the 68KMB. Upon completion of this lab, students will be able to do the following:

- Write subroutines to perform character input/output on an asynchronous serial interface.
- Use character I/O subroutines in programs that receive input from a keyboard and send output to a CRT display.

PREPARATION

Prior to the scheduled lab session, read the following sections from Chapter 9 (Interface Examples) of your textbook:

- Section 9.1 (Introduction)
- Section 9.2 (The 68681 DUART)
- Section 9.3 (RS232C Interface)

MATERIALS

Hardware:

- 68KMB 68000-based computer
- PC host computer
- RS232C serial interface cable

MS-DOS Software:

- A68K.EXE 68000 cross assembler
- XLINK.EXE 68000 linker, locator, conversion utility
- PC-VT.EXE VT100 terminal emulator
- EDIT.COM MS-DOS text editor (or equivalent)\

68000 Programs:

- MYNAME2 to be written
- ALPHA to be written

INTRODUCTION

The 68KMB's monitor program, MON68K, includes a variety of built-in routines for input/output, data conversion, etc. These are accessed through trap instructions.

As an example, the program called MYNAME in lab #4 sent a string of ASCII characters to a terminal by placing the address of the string in register A1 and then executing a TRAP #2 instruction:

```
MOVEA.L    #MESSAGE, A1
TRAP      #2
```

There is nothing unique about TRAP #2 in the architecture of the 68000. TRAP #2 provides access to special routines in MON68K; but this trap could be used for a different purpose (or not at all) in other implementations using the 68000 microprocessor. This point is mentioned because it is important to distinguish details of the 68000 architecture from details of an implementation – the 68KMB.

As students of computer organization and the 68000 microprocessor, we want to get as close to the hardware as possible using machine language or assembly language programs. Traps hide details of input/output. This is convenient for systems' programmers; however, to uncover the details of input/output subsystems, we want to get inside the low-level details of character input/output as implemented on typical computer systems such as the 68KMB.

Asynchronous Input/Output

The interface between the MON68K and a terminal (or host computer) uses a serial RS232C communications channel. Each character is transmitted in the following sequence:

- a start bit (low)
- 7 or 8 data bits (LSB first)
- a parity bit (optional)
- a stop bit (high)

This type of communications is called **asynchronous**, since the receiver must re-synchronize itself with each new character. Usually 7 data bits are used since this is the size of ASCII codes. Figure 6-1 shows the ASCII code for the letter *a* framed by a start bit, an odd parity bit, and a stop bit. (**Odd parity** means the total number of bits equal to one is an odd number. Only the data bits and the parity bit are counted.) The reciprocal of the transmission time for each bit is called the **baud rate**. For the 68KMB, the baud rate is 9600, so the period of transmission for each bit is 104 μ s.

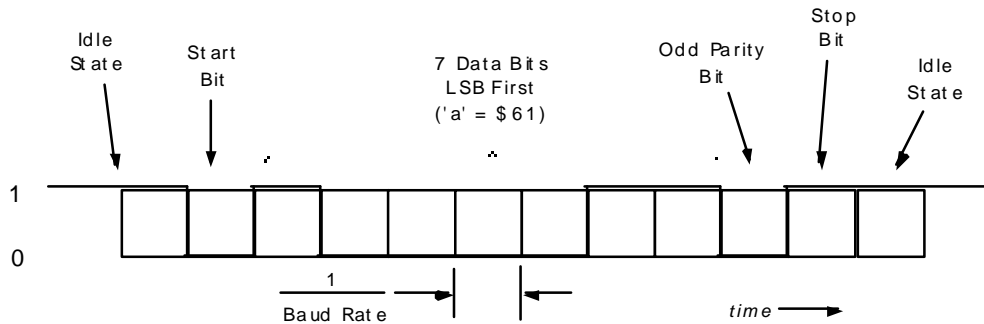


Figure 6-1. Asynchronous character transmission

Of course, characters are stored in parallel in registers or memory locations, so transmission on an RS232C interface requires special devices at each end to perform parallel-to-serial conversion or vice versa. On the 68KMB, this device is a 68681 DUART (Dual Universal Asynchronous Receiver/Transmitter). Although the 68681 includes two separate serial interface channels, we will only use Channel A in this lab. Our discussion of the 68681's features is very limited in this lab. Complete details are found in *The 68000 Microprocessor*. The hardware interface to the 68000 is presented in Chapter 8, programming examples are found in Chapter 9, and the 68681 data sheet is found in Appendix H.

A simplified version of the CPU interface is shown in Figure 6-2.

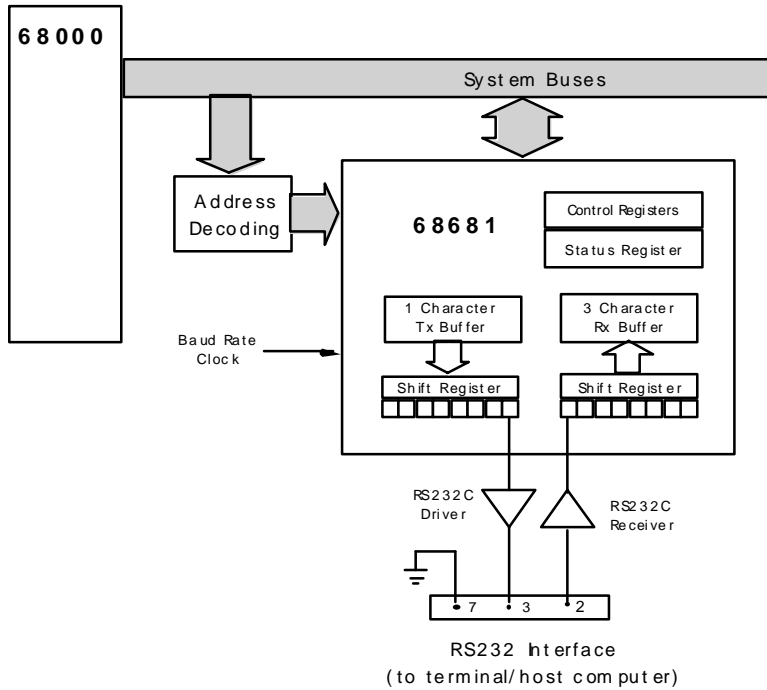


Figure 6-2. Interface to the 68681 DUART

The 68681 includes a double-buffered transmitter and a quadruple-buffered receiver. During transmission, one character can be waiting in the Transmit Buffer while the previous character is being transmitted out the shift register (see Figure 6-2). For character reception, a three character FIFO (first-in, first-out) buffer holds characters waiting to be read through software while the next character is clocked into the shift register.

In Figure 6-2, note the special interface ICs between the 68681 and the RS232C interface. An RS232C line driver converts the voltage of outgoing signals from TTL (transistor-transistor logic) levels to RS232C levels. An RS232C line receiver converts the voltage of incoming signals from RS232C levels to TTL levels. This conversion is summarized in Table 6-1.

Table 6-1. TTL-RS232C Signal Conversions

TTL			RS232C	
Logic	Voltage		Logic	Voltage
high (1)	2.4 to 5 volts	=	MARK	-3 to -25 volts
low (0)	0 to 0.8 volts	=	SPACE	+3 to +25 volts

Since data are transmitted on pin 3 of the RS232C interface, the 68KMB is configured as a DCE. A straight-through cable can be used as long as the terminal/host computer at the opposite end is configured as a DTE (see lab #2).

The address decoding provides access to the 68681's registers through odd-byte addresses \$00C001 to \$00C01F. In this lab we are only concerned with the three registers shown in Table 6-2.

Table 6-2. 68681 Registers

Address	Read	Write
\$00C005	Status Register A (SRA)	-
\$00C007	Receive Buffer A (RBA)	Transmit Buffer A (TBA)

Each bit in the status register has a different purpose, as shown in Table 6-3.

Table 6-3 68681 Status Register Bit Assignments

7	6	5	4	3	2	1	0
Received Break	Framing Error	Parity Error	Overrun Error	TxE _{MT}	TxRDY	FFUL	RxRDY
0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes

The bits we are concerned with in this lab are TxRDY (bit 2) and RxRDY (bit 0). TxRDY indicates either that the transmitter is ready to accept a new character for transmission (TxRDY = 1), or that the transmit buffer is full (TxRDY = 0). RxRDY indicates either that a character has been received and is waiting in the FIFO to be read by the CPU (RxRDY = 1), or that the receive buffer is empty (RxRDY = 0). Both TxRDY and RxRDY are status *flags* that are checked through software to determine the status of an input or output port.

Character Reception

When inputting a character, RxRDY must be interrogated continually (i.e., in a loop) until it equals 1. This indicates a character has been received and is sitting in the receive FIFO waiting to be read. After reading the character, RxRDY is cleared automatically unless there is another character in the FIFO waiting to be read. A flowchart of the steps just described is shown in Figure 6-3a.

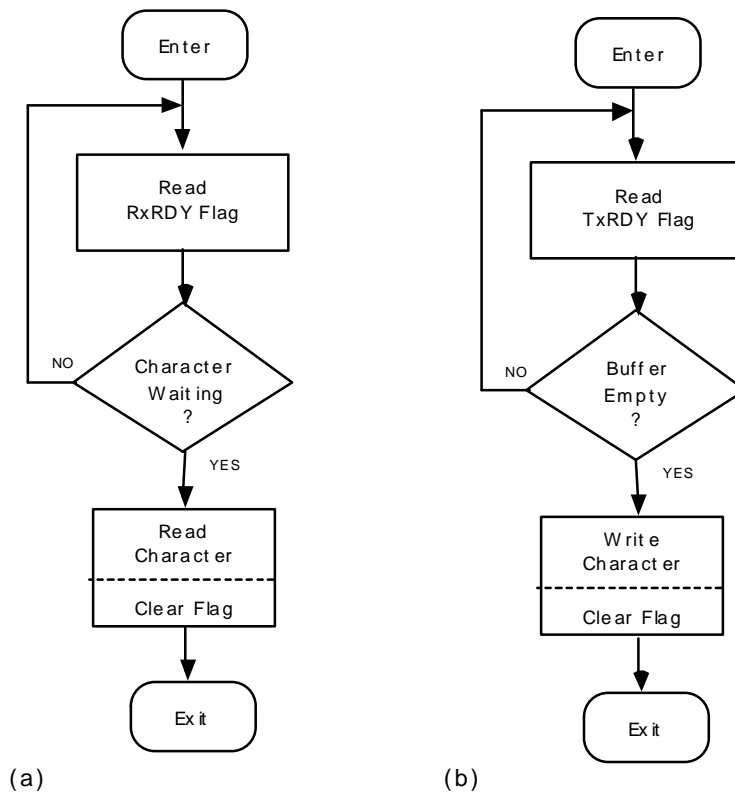


Figure 6-3. Flowcharts for (a) character input and (b) character output

The instructions to input a character from the terminal/computer attached to the 68681 into register D0 are shown below:

```

DUART    EQU        $00C001    ;base address for 68681
SRA      EQU        2          ;offset for Status Register A
RBA      EQU        6          ;offset for Receive Buffer A
LOOP     MOVEA.L    #DUART,A0   ;A0 points to 68681
         MOVE.B     SRA(A0),D7  ;get Status Register A
         ANDI.B     #1,D7       ;RxRDY = 1?
         BEQ       LOOP         ;no: check again
         MOVE.B     RBA(A0),D0  ;yes: input character

```

Note the effective use of equates to make the instructions easier to understand. As well, the addressing mode used to access the 68681 registers is address-register-indirect-with-offset. Although absolute long addressing could also be used, this would increase the size of each instruction by one word.

The ANDI instruction uses a mask of 00000001_2 to clear all bits except bit 0, which corresponds to the RxRDY bit. After the ANDI instruction, the entire low-byte of $D7 = 0$ if $RxRDY = 0$; so the appropriate branch to repeat the test is *branch-if-equal-zero* (BEQ).

Once a character has arrived, $RxRDY = 1$ and the branch test fails. The move instruction following BEQ reads a character from memory location \$00C007 (Receive Buffer A) and places it in D0.

Usually the instruction sequence shown above is part of an input character subroutine or trap.

Character Transmission

When outputting a character, TxRDY must be interrogated continually to determine when the last character written has been moved into the shift register. At such time, the next character can be loaded into the transmit buffer. A flowchart illustrating this is shown in Figure 6-3b. The following output character sequence is similar to the input character sequence. In this instance however, we assume an ASCII code has been loaded into D0 in advance. Note also that bit 2 of the 68681 status register must be interrogated in the output character sequence.

```

DUART    EQU        $00C001    ;base address for 68681
SRA      EQU        2          ;offset for Status Register A
TBA      EQU        6          ;offset for Transmit Buffer A
        MOVEA.L     #DUART,A0   ;A0 points to DUART
LOOP     MOVE.B     SRA(A0),D7   ;get Status Register A
        ANDI.B     #4,D7        ;TxRDY = 1?
        BEQ       LOOP         ;no:  check again
        MOVE.B     D0,TBA(A0)   ;yes:  send character

```

PROCEDURE

1. Make a copy of MYNAME.SRC (from lab #4), and save it in a file called MYNAME2.SRC. Put your name and the data in comment lines at the top. Modify the new program as follows. Place the code to output a character to the terminal in a subroutine called OUTCHR. This subroutine should be called from inside the subroutine OUTSTR. Do not use trap instructions except at the end of the program to return to MON68K.

Use equates near the top of the source program to define symbols for the 68681 registers. Use these symbols as appropriate in your subroutine.

Obtain a printout of the listing file and show it to your lab instructor. Demonstrate your program.

6.1 

2. Write a program called ALPHA that outputs a continuous stream of alphabetic characters to the terminal as follows:

```
abcdefghijklmnopqrstuvwxy  
abcdefghijklmnopqrstuvwxy  
abcdefghijklmnopqrstuvwxy  
(etc.)
```

Monitor the keyboard for input as follows:

- If "U" or "u" is entered, the output switches to uppercase.
- If "L" or "l" is entered, the output switches to lowercase.
- If "Q" or "q" is entered, the program terminates to MON68K.
- Ignore any other input characters.

Do not use trap instructions except at the end of the program to return to MON68K.

Put your name and the date in comment lines at the top. Demonstrate the program to your lab instructor.

6.2 

CONCLUSION

This lab has introduced character I/O on the 68KMB using a 68681 DUART.

QUESTIONS

1. If odd parity is enabled, what is the state of the parity bit for the following character codes:

F _____

e _____

z _____

% _____

2. At 4800 baud, what is the duration of each bit transmitted? _____
3. How many characters per second can be continuously transmitted at 1200 baud using 7 data bits, no parity, 1 start bit, and 1 stop bit? _____
4. An asynchronous serial interface is configured to operate at 4800 baud with 1 start bit, 7 data bits, 1 stop bit, and no parity. What is the maximum number of characters that can be transmitted across this interface in one minute? _____