CIS*2030 – Lab#10

Serial and Parallel Interfaces

OBJECTIVES

This lab introduces serial and parallel I/O with 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 out to a console.
- Write programs that input data from the parallel input port on the 68681 DUART.
- Write programs that output data to the parallel output port on the 68681 DUART.

PREPARATION

Prior to the scheduled lab session, review your lecture notes and read the following from Chapter 9 of your text book:

- Section 91. (Introduction)
- Section 9.2 (The 68681 DUART)
- Section 9.3 (RS232C Interface)
- Section 9.4 (Switches and LEDs)
- Section 9.5 (68681 Timer)
- Appendix H (DUART)

SERIAL INTERFACE - 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.

The example below sends 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 68000 ISA. TRAP #2 simply 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 ISA from details of an implementation – the 68KMB.

As students of computer architecture and the 68000 microprocessor, we need to get as close to the hardware as possible writing assembly-language programs. Traps hide details of input/output. This is convenient for systems' programmers; however, to uncover the details of I/O subsystems, we want to get inside the low-level details of character I/O, as implemented on typical computer system.

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 communication is called asynchronous since the receiver must resynchronize itself with each new character. Usually 7 data bits are used since this is the size of ASCII data. Figure 1 shows the ASCII code for the letter *a* framed by a start bit, an odd parity, and a stop bit. (Only the data bits and the parity bit are counted when computing parity.) 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 µs.

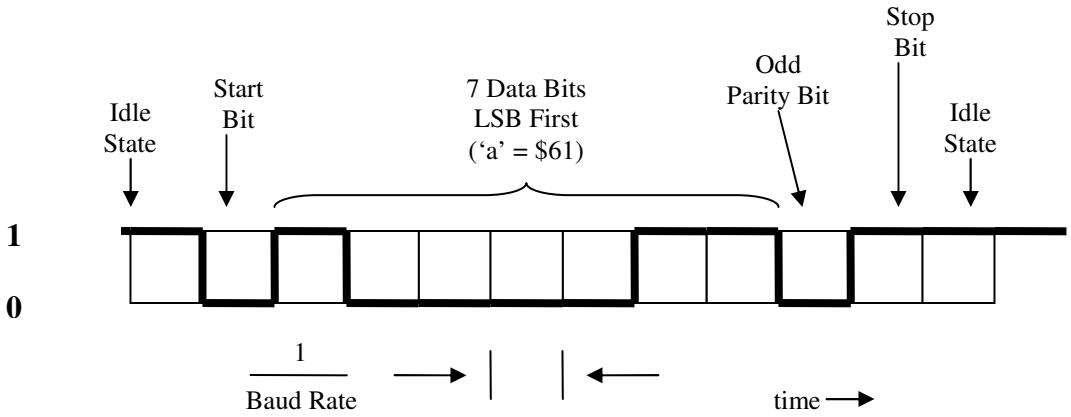

Figure 1: Asynchronous character transmission

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 visa-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 your textbook, *The 68000 Microprocessor.* The hardware interface to the 68000 is presented is 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 Fig. 2.

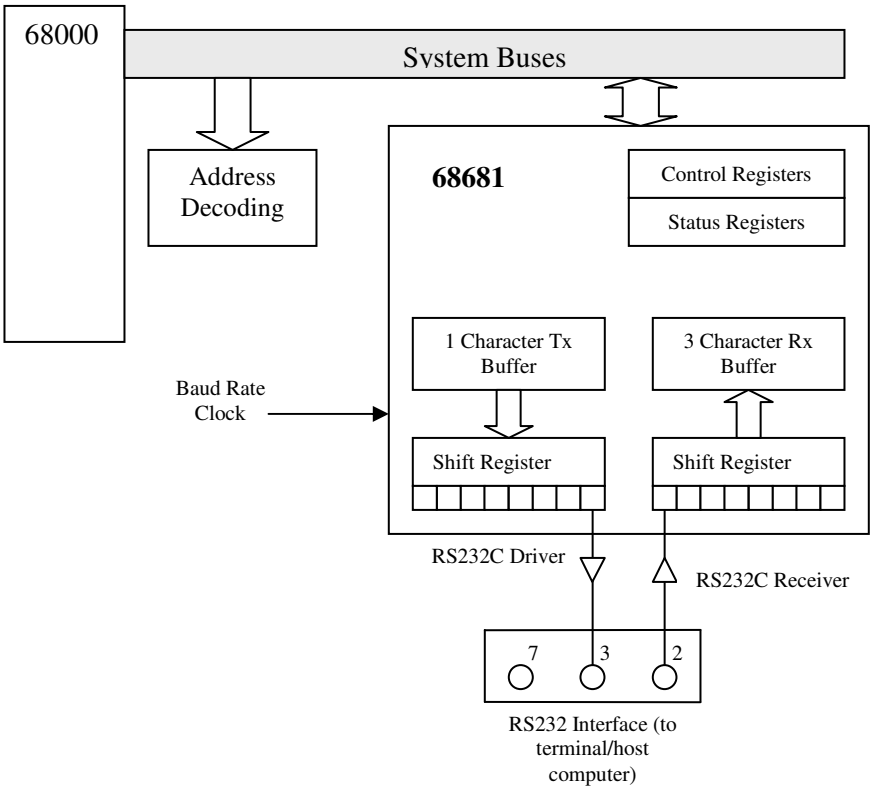

Figure 2: Interface to the 68681 DUART

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 of the shift register (see Figure 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 Fig. 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 1.

Table 1: TTL-RS232C Signal Conversions

TTL			**RS232C**	
Logic	**Voltage**		**Logic**	**Voltage**
high (1)	2.4 to 5 volts	=	MARK	-3 to -25 volts
low (2)	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 #1).

As explained in class, the address decoding provides access to the 68681's registers through **odd-byte addresses** $00C001 to $00C01F. Before the DUART can be used for transmitting/receiving characters it must be first configured. (The details of how to configure the DUART were given in class. However, there is no need to configure the DUART yourself as it has already been done for you by the monitor program.) Once configured and turned on, the transmit and receive buffers can be used to send and receive characters, respectively, using a simple polling strategy.

REQUIRED PROBLEMS

Following the procedure described in class, use the DUART's serial interface in conjunction with a simple polling strategy to implement the following programming problems.

1. Write a subroutine called OUTSTRING to output a null-terminated ASCII string pointed at by A1 to the console (screen). Note: Do not use any trap instructions except TRAP #14 at the end of your program to return to MON68K.

2. Write a subroutine called INSTRING to input a line of characters from the keyboard. The subroutine should store the string in the location in memory pointed at by A1. (You will need to reserve storage for this RAM buffer.) The subroutine should also place a null character (i.e., 0) at the end of the string. Note: Do not use trap instructions except at the end of your program to return to MON68K.

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?

PARALLEL INTERFACE - INTRODUCTION

This next part of the lab explores interfacing with the 68KMB. The lab uses an I/O board that connects to J1 on the 68KMB. In particular, this lab uses I/O board #1 which interfaces to the 68KMB through J1 as illustrated in Fig. 3.

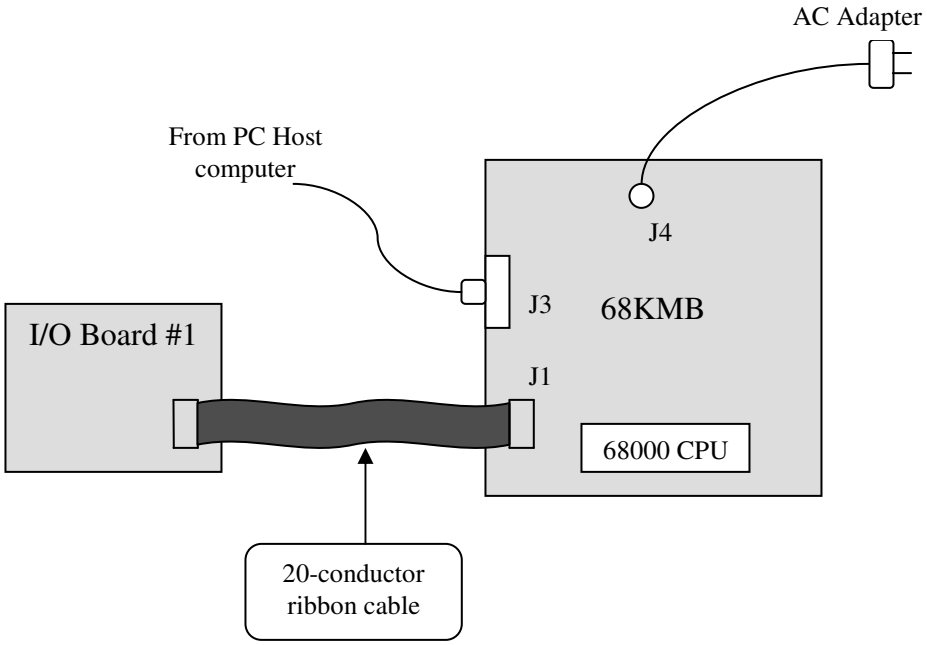

Figure 3: Connecting an I/O board to J1 on the 68KMB

J1 provides access to the 68681's parallel input port and parallel output port, as shown in Fig. 8-15 in *The 68000 Microprocessor.* I/O Board #1 contains a very simply interface to LEDs and switches (see Fig. 4). The output port signals on the 68681 are labeled OP0 to OP7. Each signal interfaces to a LED through a buffer and a resistor. Writing a *zero* to a port pin turns the corresponding LED *on.* Writing a one to a port pin turns the corresponding LED off. Upon reset, all port pins are one, therefore all LEDs are off.

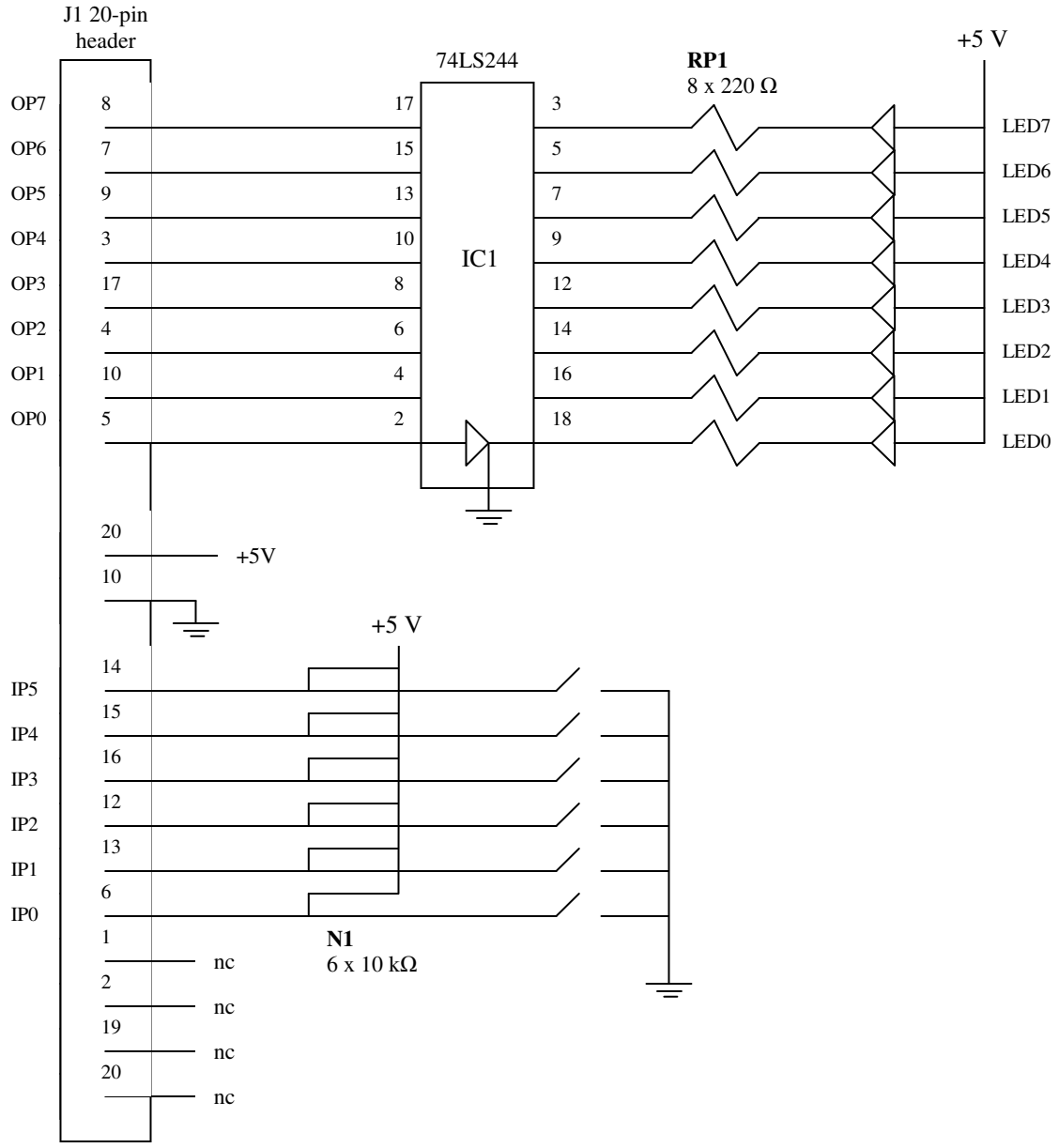

Figure 4: I/O Board #1

The 68681's parallel input port is only six bits wide. The input port signals are labeled IP0-IP5, as seen in Figure 4. Each input bit is connected to a single-pole single-throw switch. One terminal of each switch connects to ground, and the other terminal connects to an input port pin through a 10K pull-up resistor.

Recall from class that the base address of the 68681 is $00C001 and that internal registers reside at consecutive odd addresses from this address up to $00C01F. To access the 68681's parallel ports, three registers are used. These are summarized in Table 2.

Table 2: 68681 Registers used with Parallel Interface

	Read		Write	
Address	**Name**	**Function**	**Name**	**Function**
$00C01B	IPR	Input Port	-	-
$00C01D	-	-	OPR_SET	Set Output Register Bits (clear pins)
$00C01F	-	-	OPR_CLR	Clear Output Register Bits (set pins)

Reading input port pins is a simple matter of reading memory address $00C01B. Bits 0-5 reflect the state of the input port pins IP0-IP5, and bits 6-7 are read as ones. For example, the instruction

```
MOVE.B          $00C01B,D0
```

reads the state of IP0-IP5 into bits 0-5 of data register D0.

Writing to the output port is more difficult. To clear output pins, a mask byte must be written to address $00C01D with a one in each position where an output port pin is to be cleared. To set output pins, a mask byte must be written to address $00C01F with a one in each position where an output port pin is to be set. As noted in Table 2, there is an inverse relationship between the output *register bits* and the output *pins.*

The operation of the 68681's output port may seem odd, but there is a good reason for it to work as it does. Consider that the outputs might be divided among two or more interfaces. If the software driver for one interface needs to set or clear a few bits in the output port while leaving others as is, there is no easy way to do so (since the output port is write-only). The 68681 implementation is a simple solution to this, allowing output pins to be set or cleared independent of other pins.

Writing an 8-bit value to the output pins requires three steps. First, the value is written to OPR_CLR. Second, the value is complemented. Third, the complemented value is written to OPR_SET. For example, to copy the contents of the low-byte in D0 to the output port, the following instruction sequence could be used:

```
MOVE.B          D0,$00C01F      ;write to OPR_CLR
NOT.B           D0              ;complement data
MOVE.B          D0,$00C01D      ;write to OPR_SET
```

PROCEDURE

1. With the 68KMB powered-off, connect I/O Board #1 to J1.

2. Power-on the 68KMB and the PC host computer. Execute hyper-terminal and obtain the MON68K prompt from the 68KMB.

3. Use MON68K's memory modify command to read the state of the switches connected to the 68681's parallel input port:

   V4.4>MM C01B

4. Use MON68K's commands to write to OPR_CLR and OPR_SET to turn LEDs on and off

   Make sure you are comfortable using MON68K commands to read the input switches and write to the LEDs.

5. An example program called WIRE681 is presented in Example 9-3 in your textbook. WIRE681 simulates a wire connection between the six input switches and the six least-significant LEDs. Review the software listing and the description of the program to gain an understanding of its operation.

   Implement WIRE681 and run the program. Be prepared to answer questions on its operation.

REQUIRED PROBLEM

6. Okay, now its your turn! Write a program called **KEYTEST** that inputs characters from the keyboard (using the code you wrote above for the serial interface) and displays the ASCII code in binary on the LEDs. Put the program in a loop and terminate to MON68K when *q* is detected. Put your name and the date in comment lines at the top of the source program.

An example program called TWOHZ is presented in Example 9-5 in your textbook. The program makes LED #3 flash at a rate of 2 Hz. Unlike, Example 9-4, which uses a software delay to synchronize output, TWOHZ uses the 68681's built-in timer. Review the software listing and the description of the program to gain an understanding of its operation. Implement and run TWOHZ and be prepared to answer questions on its operation.

REQUIRED PROBLEM

7. Write a modified version of TWOHZ that causes all 8 LEDs to flash at a rate of 8 Hz. Call the new program **EIGHTHZ**.

Continue to use the 68681 timer in EIGHTHZ. A new approach is required, however. In Example 9-5, we directed the timer output to OP3 by writing $04 to the output port configuration register. This will not work here because we want all 8 LEDs to flash. The solution is to interrogate the timer status directly to determine when the counter reaches its terminal count. Each time the terminal count is reached, the output data should be complemented. (Note: It takes *two* toggles of the output data to achieve one period of flashing.) The terminal count is tested through the counter/timer ready bit of the interrupt status register (ISR). See Chapter 9, Table 9-2 and Appendix H, Table 6. Each time the counter/timer ready bit changes from 0 to 1, a terminal count has been reached and data can be written to the LEDs. A stop command is required to clear the counter/timer ready bit. This is performed through the 68681's STOP register, which is address-triggered (see Chapter 9, Table 9-2). Any read of this register (e.g., a TST instruction) will work fine. Note that when the 68681 is in "timer mode," as is the case here, the stop command does not actually stop the timer; it just clears the counter/timer ready bit in the interrupt status register.

CONCLUSION

This lab has introduced simple serial and parallel I/O on the 68KMB using the 68681's serial interface, parallel input port, parallel output port, timer, and interrupts. **Congratulations! You have not completed the required lab component for CIS 2030.** ☺