Lab 8: Asynchronous Serial I/O: Interfacing to an External UART and using the Internal UART

OBJECTIVES
In this lab, we will learn how to create additional asynchronous serial ports by memory-mapping a Texas Instruments (or substitute) TL16C550 UART to our 68HC12 development board. This interface will allow characters to be received and transmitted between the microprocessor and a PC serial port. This is an additional serial port to the one built into our 68HC12. In an early experiment you will use polling for the receive and transmit flags in your TL16C550 UART’s internal registers; later you will replace this mechanism with an interrupt-driven communication system. You will also write code for the 68HC12’s internal UART.

REQUIRED MATERIALS
1. 16C550 UART Chip (40-pin)
2. Socket for UART (40-pin wide)
3. 1.8432 MHz Clock
4. RS232 Line Driver (20-pin) either DS229 Chip or SP232 Chip
5. Socket for Clock (8-pin wide)
6. Socket for RS232 Line Driver (20-pin narrow)
7. Socket for the 4 charge pump capacitors (8-pin)
8. 0.1 uF bypass capacitors (UART & RS232 chips)
9. Female DB9 Header
10. Socket for DB9 Header (only need a single row of 5 pins)

PART I – WRITE USING POLLING
The first step in being prepared for this lab is to study the UART/RS232 notes attached to this lab as well as the data sheets for the TL16C550 and the RS232 line driver chips, both on our web site. We will be using the TL16C550 in TL16C450 mode. The C550 chip is a functional upgrade of the C450 with optional FIFO mode (which we will not use). You will also be using a Sipex SP232A RS-232 Line Driver/Receiver device (or similar). The web sites for the manufacturers of the two above devices are www.ti.com and www.sipex.com, respectively.

Upon studying these materials, proceed as follows:

1. Wire the UART to its clock source and directly to the microprocessor as shown in the example schematic in the UART/RS-232 Notes handout.

2. Map the UART in the 68HC12’s memory map starting at $8000 (mirrored up to $8FFF). Doing so will require you to re-program the CPLD on your board as was done in earlier labs. The chip-enable equation should take into account that this is a read and write device (just like an SRAM). Generate the appropriate chip-enable equation and print out the GDF file as part of your pre-lab.

3. Wire the Sipex SP232A (or similar) driver/receiver to the UART as shown in the example schematic. Note: Pin numbers corresponding to certain signals should be obtained directly from the data sheet. Also connect a DB9 serial port connector to the SP232A RS-232 TX & RX signals.

4. Use monitor commands (MM, MD, etc…) to write to and read from the Line Control Register (LCR) in the UART at address $8003. Note: You must pass this test before continuing on to any additional test. If there are discrepancies between what was written and what was read, then check your wiring and decode circuitry. An LSA can be used to trigger on write & read cycles to view all important bus signals (Data Bus, A2:0, R/~W, Chip Enable, etc.).

5. Upon successful writing and reading of the UART’s above internal register, connect a serial cable between the new DB9 serial port connector and a free serial port on your PC. Note: It may be advantageous to use another student’s serial cable in addition to your serial cable so that you can still talk to the monitor and test the new port.

6. Write a program that reads 0-9 on your keypad and then sends this value (ASCII equivalent) out your new serial port. The steps to perform this are as follows:

   a. Initialize the UART to match your settings in the HyperTerminal software on the PC (i.e. 9600 baud, 8 data bits, no parity, 1 stop bit). Note: The UART document describes how to program these settings in the internal registers of the TL16C550. The register initialization should only be a few lines of assembly code.

   b. Test (i.e. poll) bit 5 (i.e. the Transmitter Holding Register Empty or THRE flag) in the Line Status Register (LSR). If it is high, then check/read a key pressed on the keypad (0-9) and write it to the Transmitter Holding Register (THR).

   c. Upon sending out a character, check for a new key pressed, poll the THRE flag, and write the value to the THR when allowed. Repeat continuously forever. Important Note: Remember to turn your keypad value (0-9) to an ASCII character by adding $30 to it.

You should now see a new character every time a key is pressed in the new HyperTerminal window connected to the serial port you are using with the UART & SP232A. If you do not, then use an LSA to check the serial bit pattern being sent to the SP232A. Verify that the TX & RX lines of the UART are wired to the correct signals in the SP232A.
Part II – Read Using Polling

Use your earlier software to initialize the UART's internal registers to again match the HyperTerminal settings. Next, write a short routine to test (i.e. poll) the Data Ready (DR) flag in the Line Status Register. When the flag is set (high), read the Receiver Buffer Register (RBR) and display the character on your LCD added in the last lab. Don’t forget to initialize the LCD.

Force this code to repeat in an infinite loop such that we can type keys in the HyperTerminal window and see them show up one at a time on your LCD.

Part III – Write/Read via Interrupts

Wire the 68HC12’s IRQ signal to the UART as shown in the UART/RS232 notes. Do not forget to add the appropriate CPLD jumper. The UART will now be used to interrupt the 68HC12 whenever a character is sent out or received. Here are the specific steps to achieve this objective:

1. Initialize the UART as was done in Parts I & II.
2. Enable the RX interrupt in the UART's Interrupt Enable Register (IER). Enable the IRQ interrupt in the 68HC12, redirect its vector, and do not forget to "CLI".
3. Write an ISR that first tests the Interrupt Identification Register (IIR) to see if it is an RX or TX interrupt.

Important Note: Reading the IIR automatically clears the pending interrupt ID. Therefore, this register can only be read ONCE!

   a. If it is an RX interrupt, read the RX buffer and save the character to memory. Then disable the RX interrupt, enable the TX interrupt, and branch to the end of the handler.

   b. If it is a TX interrupt, read memory (where the last character is stored) and write the character to the transmit buffer. Then disable the TX interrupt, enable the RX interrupt, and branch to the end of the handler.

When this code is running properly, we should simply see the character typed in the HyperTerminal echoed back to the HyperTerminal window. We are waiting for a character, receiving it, saving it to memory and then sending it back out via interrupts.

Note: Make sure that you do not have the “Echo typed characters locally” option checked under the Hyperterminal ASCII Setup. Having it enabled will cause characters to be echoed back twice. Your TA will also check that this option is not checked.

Part IV – 68HC12 SCI System

The 68HC12 has an Asynchronous Serial Communication Interface (SCI). This system works just like the C550 chip. It only requires an external RS-232 driver. This is the system that is connected to the on-board DB-9 header.

Read Chapter 11 in the S&HE book (soft-cover) sections 11.1, 11.2, and 11.3. These sections discuss the SCI system in detail.

You are to write a program similar to that of Part III (reading and writing using interrupts) using the SCI system. The program is described below.

1. Initialization
   a. Set the data direction of PORTS Bits 1 and 0 to the corresponding values for Tx and Rx.
   b. Set the baud rate to 9600 bps by storing the appropriate divisor into the SCI0BDH and SCI0BDL registers.
   c. Set SCI0CR1 to indicate 8 data bits, 1 start bit, and 1 stop bit with wired-or, loop mode and parity off.
   d. Enable the transmitter and receiver by setting the appropriate bits in SCI0CR2.

2. Baud Rate Selection Menu
   a. Use POLLING (TDRE bit in SCI0SR1) to display the baud rate selection menu. The menu should be as follows:

   Please select a baud rate
   0 – 300 baud
   1 – 1200 baud
   2 – 2400 baud
   3 – 4800 baud
   4 – 9600 baud
   >

   Note: It will be VERY useful to create a subroutine to print menus. The subroutine could take the starting address of the string in Register X and would expect it to be terminated by an End-of-String character (EOS). (Note: It will make it easier to write menus in future labs.)

   b. Use POLLING (RDRF bit in SCI0SR1) again to take a 1-character input through the SCI. Make sure that the character is 0, 1, 2, 3, or 4 (remember the input is in ASCII). Ignore any other input. Note: The input polling can also be made into a subroutine. For example, the subroutine would return a single input character in Register A.
   c. Display the input character on the computer screen.
Lab 8: Asynchronous Serial I/O: Interfacing to an External UART and using the Internal UART

d. Then display a message (using POLLING) indicating the new baud rate. For example, pressing the number 3 should show:

   Please change your terminal to 4800 baud.
   Hit <ENTER> when you are ready.

e. Change the divisor in the SC0BDH and SC0BDL registers.
f. Use POLLING to take a 1-character input. If it is equal to <ENTER> (called CR or “carriage return” in ASCII), proceed to the echo section. Otherwise, ignore the input and poll again until CR appears.

3. ECHO back characters
   a. Display the following message using POLLING:

   Begin typing. Press <ESC> to quit.

   b. Store $FF into a reserved memory location called UTYPE. This indicates you are in TYPE mode. (The location should be initialized to $00 at the beginning of the program).

   c. Enable the Rx interrupt by setting the RIE bits in SC0CR2. Clear the global interrupt flag using the CLI instruction.

   d. Enter a loop that constantly checks the UTYPE. This is the working loop for your SCI ISR.

   e. The RX portion of your SCI ISR should wait for input. If the input is equal to ESC, it should clear the UTYPE flag and disable all interrupts. Otherwise, it should save the value for the TX portion of the code, disable the RX interrupt, and enable the TX (the TIE bit) interrupt.

   f. The TX portion of your SCI ISR should take the input value and show it on the screen. Then, disable the TX interrupt and enable the RX interrupt.

4. HITTING ESC
   The program should return to the baud rate selection menu upon hitting the ESC key while typing.

PRELAB REQUIREMENTS
1. Read the ENTIRE lab handout.
2. Create a hand-drawn wiring schematic of the entire UART system. This includes the 68HC12 pins, DB9 Header, Clock, C550, and DS229 chips. Use labels instead of wires to show connections. (2.5%)
3. Part I step 2, provide the UART chip-enable equation and GDF printout of the added CPLD logic. (2.5%)
4. Part I step 6, provide ASM and error-free LST for the input read/pollled write program. (2.5%)
5. Part II, provide ASM and error-free LST for the polled read/write display program. (2.5%)
6. Part III, provide ASM and error-free LST for the interrupt-based UART read/write program. (2.5%)
7. Part IV, provide ASM and error-free LST for the Baud Rate Selection program. (2.5%)

IN-LAB REQUIREMENTS
1. A disk containing your LATEST ASM and LST files.
2. Part I, demonstrate the polled write program (read switches and output to screen) to your TA and answer your TA’s questions (20%).
3. Part II, demonstrate the polled read/write program (read the keyboard and output to LCD) to your TA and answer your TA’s questions (25%).
4. Part III, demonstrate the interrupt-based UART program to your TA and answer your TA’s questions (20%).
5. Part IV, demonstrate the Baud Rate Selection program to your TA and answer your TA’s question (20%).