ESE381 Embedded Microprocessor Systems Design II

Spring 2022 - K. Short revised February 5, 2022 4:31 pm

Laboratory 04: Software Implementation of a UART Transmitter and Receiver and Use of a Saleae Logic Analyzer and Tera Term Terminal Emulator

To be performed the week starting February 22nd.

Prerequisite Reading

1. Maxim Integrated, *Serial Digital Data Networks* (on Blackboard).
2. Maxim Integrated, *Determining Clock Accuracy Requirements for UART Communications* (on Blackboard).
3. Dallas Semiconductor, *Application Note 83 Fundamentals of RS–232 Serial Communications* (on Blackboard).
4. *Saleae User's Guide 11-12-19, (on Blackboard).*

Overview

Microcontrollers have universal synchronous/asynchronous receiver/transmitter (USART) modules that provide, among other things, all the functionality of a universal asynchronous receiver and transmitter (UART). These hardware modules provide a hardware option for the hardware/software trade-offs in implementing asynchronous serial communications.Their major advantage is that they can perform the operations of serializing and deserializing data concurrently with the microcontroller CPU performing other tasks. Using hardware is typically the preferred approach to implementing asynchronous serial communications and you will use that approach in the Laboratory 07.

However, nothing clarifies the basic concepts of asynchronous serial transfer like writing your own C functions to implement the serial transmitter and receiver. In doing so, you make use of C's bit manipulation instructions and library delay functions.

With any serial transfer protocol, debugging a design is greatly simplified if you have the appropriate test equipment. A key piece of equipment for debugging serial communications is a protocol aware logic analyzer. In this laboratory you will learn to use Saleae's 16-channel logic analyzer. This is a protocol aware analyzer that can analyze a large number of different serial protocols, including asynchronous serial. You will be using this tool extensively as you study a few of the most used serial protocols in embedded system design.

When using the asynchronous serial protocol, another key tool is a terminal emulator software application that emulates an ASCII terminal on your PC. The emulated ASCII terminal allows you to send and receive ASCII information in the asynchronous protocol and see this information displayed on the PC's monitor. There are many different terminal emulators available. In this laboratory your will be using Tera Term.

Design Tasks Summary

Design Task 1: Software UART Transmitter Function
Design Task 2: Software UART Receiver Function
Design Task 3: Software UART Receiver Interrupt Service Routine
Design Task 4: Interrupt Echo Program

Design Tasks

Design Task 1: Software UART Transmitter Function
You must write a function named `UART_sw_write`.

```
void UART_sw_write(char c);
```

This function has a `char` parameter and has the return type `void`. The `char` passed to the function will be an ASCII character that is to be transmitted according to the asynchronous serial protocol. The frame format for the asynchronous transfer is fixed at 8N1. The baud rate BAUD_RATE must be defined as a #define macro preprocessor directive at the beginning of your program, so that the baud rate can easily be changed.

Write a program named `asynch_sw_send` that allows you to send ASCII characters at any of the following three baud rates: 4800, 9600, and 19200 baud using the `UART_sw_write` function. This program must use PB0 as the TX pin for your software transmitter. In the laboratory, this program will be used to test your `UART_sw_write` function. The characters transmitted will be observed using the oscilloscope, Saleae logic analyzer, and Tera Term. You must use a loop in your program to continuously send the same character with a 1 ms delay between the characters sent.

Your program's duration of a single bit (bit time) needs to be as accurate as reasonably possible for each of the baud rates. You can use information from Laboratory 05 to help you with bit time accuracy.

Submit your program C source file and program verification strategy as part of your prelab.

Design Task 2: Software UART Receiver Function
You must write a function named `UART_sw_read`.

```
uint8_t UART_sw_read();
```

This function has the return type `uint8_t`. The `uint8_t` returned by the function must be the right justified ASCII character received. The frame format for the asynchronous transfer must be fixed at 8N1. The baud rate BAUD_RATE must be defined by a #define macro preprocessor directive at the beginning of your program, so that the baud rate can easily be changed.

Function `UART_sw_read` must use PB1 as the RX pin for this software receiver. This function must poll PB1 to detect the start of an asynchronous frame. It must then perform a false start check at the middle of the start bit. The function does not return until it has received a character.

Write a program named `asynch_sw_read` that allows you to read ASCII characters at any of the following three baud rates: 4800, 9600, and 19200 baud. The received character returned by the function must be placed in a global variable. In the laboratory, this program will be used to test your `UART_sw_read` function. The characters read will be observed using the Saleae logic analyzer. In the main loop of your program simply place a call to the `UART_sw_read` function and place a breakpoint on this call. When the breakpoint is reached you can view the global variable in Studio's Watch window to see the value received.

Your program's duration for a single bit (bit time) needs to be as accurate as reasonably possible.

Submit your program C source file and program verification strategy as part of your prelab.

Design Task 3: Software UART Receiver Interrupt Service Routine

A drawback of the software receiver in Task 2 is that it polls to detect and receive an asynchronous frame. The function does not return until it has detected and received a frame. If the function is used in a larger program that must carry out other tasks in real time, the program locks up waiting for an asynchronous frame to be received.

A better approach would be to use an interrupt to detect the frame's start bit at PB1. You had some experience with pin change interrupts in ESE280. More information on using pin change interrupts is in the Laboratory 6 Lecture.

Modify the function `UART_sw_read` from Task 2 to be an interrupt service routine that services a pin change interrupt request generated by a start bit at PB1.

Write a program named `asynch_sw_read_interrupt` that allows you to read ASCII characters at any of the following three baud rates: 4800, 9600, and 19200 baud. In the laboratory, this program will be used to test your interrupt service routine. The main loop in your program does not need to do anything. However, after your configuration and initialization code is executed you want execution to stay in the main loop until an interrupt occurs and return to the main loop after the ISR completes. You can write your main loop as follows to keep it from being optimized away.

```
while (1)
{
    //a nop so while loop is not optimized away
    asm volatile ("nop");
}
```

You can observe the character received by your program by storing it in a global variable in your

ISR and placing this variable in Studio's Watch window. When a character is typed in Tera Term you can observe the character sent to the Curiosity board using the Saleae logic analyzer. You can then pause program execution to read the global variable's value in the Watch window.

Design Task 4: Interrupt Echo Program

Write a program named `interrupt_echo` that combines the code from Task 1 and Task 3 to create a program that uses an interrupt to receive an alphabetic character from Tera Term and echoes the character back to the Tera Term with its case changed. So, if you send a lower case 'a' it sends back an upper case 'A'. The characters type into Tera Term are assumed to be limited to alphabetic characters.

Submit your program C source file and program verification strategy as part of your prelab.

Design Task 5: Message Relay Program

Modify your program from Design Task 4 to create a program named `interrupt_echo_line` that uses interrupts to receive a line of ASCII characters from the Tera Term. The characters received must be buffered in an 80 character array until a carriage return 'CR' control character (0x0D) is received. Once a carriage return character is received, the program must send (echo or relay) the entire line back to the Tera Term. After a line is echoed back, the Tera Term's cursor must be moved to the beginning of the next line.

Submit your program's C source file and program verification strategy as part of your prelab.

Laboratory Activities

Laboratory Task 1: Software UART Transmitter Function

Create a project named `asynch_sw_send`. Connect the Oscilloscope and Saleae Logic Analyzer to your TX signal at PB0 (Curiosity Nano pin 16). The Curiosity connects to the Tera Term via its CDC TX pin and its USB connection, You do not have to do any wiring for the USB connection.

Load, debug, and run your program. Measure the bit times for the bits in your transmitted ASCII characters using the Oscilloscope and Saleae Logic Analyzer. Use the Snipping Tool to get a screen capture of the transmitted characters on the Saleae Logic Analyzer with the bit times measured and added to the waveform. Compute the percent error in your generated bit width compared to the specified time.

The waveform on the Saleae logic analyzer will indicate to you with white dots where it expects the center of each bit time to be. Those white dots indicate where the Saleae sampled the incoming data signal.

Have a TA verify that your program works properly and that your 9600 baud waveform's timing is correct. Get that TA's signature.

Laboratory Task 2: Software UART Receiver Function

Create a project named `UART_sw_read`. Connect the Oscilloscope and Saleae Logic Analyzer to your RX signal at PB1 (Curiosity Nano pin 17).

Load, debug, and run your `UART_sw_read` program. Measure the bit times for the bits in your received ASCII characters using the Oscilloscope and Saleae Logic Analyzer. Use the Snipping Tool to get a screen capture of the received characters on the Saleae Logic Analyzer with the white dots indicating when the Saleae Logic Analyzer expects the receiver to sample the data.

Have a TA verify that your program performs properly and decodes the character read to the correct value. Get that TA's signature.

Laboratory Task 3: Echo Program

Create a project named `asynch_sw_read_interrupt`. Connect the Saleae Logic Analyzer to your RX signal at PB1 (Curiosity Nano pin 17) and TX signal at PB0 (Curiosity Nano pin 16).

Load, debug, and run your `asynch_sw_read_interrupt` program. Measure the bit times for the bits in your received ASCII characters using the Oscilloscope and Saleae Logic Analyzer. Use the Snipping Tool to get a screen capture of the received characters on the Saleae Logic Analyzer with the white dots indicating when the Saleae Logic Analyzer expects the receiver to sample the data.

Have a TA verify that your program performs properly by generating an interrupt at the beginning of the start bit and decodes the character read to the correct value. Get that TA's signature.

Laboratory Task 4: Interrupt Echo Program

Create a project named `interrupt_echo`. Connect the Saleae Logic Analyzer to your RX signal at PB1 (Curiosity Nano pin 17) and TX signal at PB0 (Curiosity Nano pin 16).

Load, debug, and run your `interrupt_echo` program. After you type an alphabetic character into the Tera Term you should see the character displayed on the Tera Term monitor with the case changed. You can place a breakpoint in your ISR after it has received the character and store it in a global variable. This will verify that the interrupt occurs and you can verify the received character by viewing its global variable in the Watch window.

Have a TA verify that your program echoes back each character entered on the Tera Term with its case changed.

Laboratory Task 5: Message Relay Program
Create a project named `interrupt_echo_line`. Connect the Saleae Logic Analyzer to your RX signal at PB1 (Curiosity Nano pin 17) and TX signal at PB0 (Curiosity Nano pin 16).

Load, debug, and run your `interrupt_echo_line` program. After you type a character into the Tera Term, you can pause execution of your program and verify the received character using a global variable in the Watch window. You can also place your array in the Watch window and observe it fill with the characters you have typed into the Tera Term. As you type characters into the Tera Term, they will not display individually on the Tera Term's monitor. When you type the Enter key on your keyboard a carriage return character will be sent by the Tera Term. When your program receives the CR, it should send back the entire line of characters that you typed including the CR and then it should send a LF so that the Tera Terms cursor is on a new line.

Have a TA verify that your program buffers and relays back each line of characters entered on the Tera Term.

Questions
1. What would be the preferred hexadecimal value to send as an initial test of your `UART_sw_write` function from Task 1 and why? Preferred in the sense that it would be the easiest to use to verify the bit times using an oscilloscope or logic analyzer.

2. From the article *Determining Clock Accuracy Requirements for UART Communications* what would you use as the maximum allowable percent error in your bit times for a software transmitter or receiver for the "nasty link" case and the "normal link" case respectively? Assume that the device you are communicating with has no bit time error.
?
3. If we were to go to very high baud rates, how might that effect the avr-libc delay function(s) that you use in your program?

4. If you were implementing an interrupt driven software receiver, as in Task 4, for fairly low baud rates, how could you improve its performance by using one of the counters on the AVR128DB48? Provide a one paragraph explanation of how you would use the counter in the overall implementation of the receiver.