ESE381 Embedded Microprocessor Systems Design II

Spring 2024, K. Short   revised March 1, 2024 8:42 pm

Laboratory 06: Circular Buffers and AVR128DB48 USART Module in Asynchronous Serial (RS232) Mode

To be performed the week starting March 3rd.

Prerequisite Reading

1. Lecture 10: Serial Interrupts and Circular Buffers
2. AVR306: Using the AVR USART on tinyAVR and megaAVR devices. You can skip section 3 of this document.
3. Termite Terminal Emulator Help.
4. AVR128DB48 Data Sheet, Section 27: USART - Universal Synchronous and Asynchronous Receiver and Transmitter.
5. <avr/interrupt.h>: Interrupts

Overview

There are two ways for the AVR128DB48 CPU to transfer data serially using one of its USARTs. The first is polling, which you became familiar with in Laboratory 05. The drawback with using polling is that the CPU must constantly expend clock cycles doing the polling. The second way is to use interrupts. The advantage to using interrupts is that once you set up and enable the interrupts, the CPU can go on to other tasks until it is interrupted. In response to an interrupt, the appropriate interrupt service routine (ISR) can read or write the data to be transferred and then immediately go back to other system tasks.

When using interrupts, the efficiency of the system can often be improved by using **circular buffers**. Characters being received can be placed in a software receive buffer until an appropriate amount of data has been received. Then this data can be processed and the results transferred to a software transmit buffer as a block. Characters in the transmit buffer are transferred as a block using data register empty or transmit complete interrupts.

This laboratory will demonstrate both the use of interrupts and the use of circular buffers.

Design Tasks

The designs in this laboratory involve writing programs to configure USART3 to receive and transmit asynchronous serial protocol data using interrupts and circular buffers. You will verify the results of your work using the Saleae logic analyzer and Termite terminal emulator.

Design Task 1: A More Flexible USART Initialization Function.

A simple function to configure a USART might have a single parameter that specifies the desired baud rate.

The USART initialization function that you must write for this task goes further, it allows both baud rate and part of the frame format to be specified. Write a function named `USART3_Init`. The function prototype is:

```
void USART3_Init(uint16_t baud, uint8_t data_bits, unsigned char parity);
```

The parameter `baud` is the baud rate in bits per second. To minimize complexity, write this function to only handle `data_bit` values from 0 through 8, with the number of stop bits fixed at 1.

Write a test program that uses this function to initialize the AVR128DB48's USART3 and to transfer characters at the specified speed and format. Name the program `usart3_init_test`. After configuring USART3 using the above function, the program only needs to transmit a single specified byte. This allows the characteristics of the transmitted byte to be observed on the Saleae logic analyzer to confirm the correctness of the configuration. The correctness of the configuration of USART3 can also be verified in the I/O window. Note, this program must not use the circular buffers.

Submit your C source file for this program as part of your prelab.

Design Task 2: Modifying AVR306 Code to Work with AVR128DB48

Write a program named `usart3_avr128_cir_buff` that is a modification of the program in the AVR306 Application Note to work with the AVR128DB48 and compile using Studio 7. This program receives characters via USART3, places them in a circular receive buffer. The main loop transfers the characters from the receive butter to the transmit buffer and enables the transmit data buffer interrupt so that the transmit ISR writes data in the transmit buffer to the USART for transmission.

Submit your C source file for this program as part of your prelab.

Design Task 3: Verifying Operation of Your Program with a Block of Data Being Transferred

This task requires you to modify the program for Task 2 so that you can do a more thorough verification of the operation of the circular buffers. The modification is given in Laboratory Lecture 06 which is posted in the Laboratory 06 folder on Blackboard.

Laboratory Activity

Laboratory Task 1: A More Flexible USART Initialization Function.

Create a project named `usart3_init_test` using the program `usart3_init_test` that you wrote. Set the compiler optimization to -Og. Build the program. Place any program variables in a watch window to observe their values. Single step through each instruction in the program.

Prior to single-stepping each instruction, determine what changes you expect in the values of the USART3 SFRs and any variables you created.

Use the I/O window to monitor the SFRs of USART3 to determine if the USART is properly configured by your `USART3_Init` function.

Use the Saleae Logic Analyzer to capture your test program's output for different baud rates and formats.

When your program is working correctly and generates the required USART3 configuration and traces, have a TA verify that your program performs as required. Get the TA's signature.

Laboratory Task 2: Modifying AVR306 Code to Work with AVR128DB48

Create a project named `usart3_avr128_cir_buff` using the program `usart3_avr128_cir_buff` that you wrote. Set the compiler optimization to -Og. Build the program. Place the buffers and any other variables in a watch window to observe their values. Single step through each instruction in the program. Prior to single-stepping each instruction, determine what changes you expect in the values of the USART3's SFRs and any variables you created. Change the receive and transmit buffer sizes to 8 and put the buffers in a watch window to see if the buffers operate properly.

Use the Saleae logic analyzer to monitor the transfer and receipt of frames during serial transfers.

When your program seems to be correct when single stepped, run it full speed and verify its operation. Use the TeraTerm terminal emulator to send and receive characters using a format of 8N1.

When your program is working correctly and echoes the characters typed into TeraTerm using interrupts, have a TA verify that your program performs as required. Get the TA's signature.

Laboratory Task 3: Verifying Operation of Your Program with a Block of Data Being Transferred

At this point, I have not been able to find a way to configure TeraTerm to buffer data and transmit it as a block or burst of data. However, I have found another free terminal emulator called Termite that will allow this. Information on its use is given in Prerequisite 3, Laboratory 06 Lecture, and at the end of this document.

Termite allows you to send data that is typed into a text box at the bottom of the applications window. However, a character is not sent as soon as you type it, like in TeraTerm. You have to click an icon to the right of the text box to transmit a character. If you type a block of characters into the text box and then click on the icon, the block of characters will be transmitted at once. In other words, Termite has its on transmit buffer and will transmit the contents of this buffer when you click on the icon.

A number of the questions at the end of this laboratory are based on you using Termite to send different size blocks of data and verify the results of the transfers in the Watch window of Studio and in Termite's text area.

Using Termite

You can download Termite from: https://www.compuphase.com/software_termite.htm
Once you are at this site, click on the “Termite version 3.4 complete setup” link to download the setup exe file. Open this file to download Termite.

The document at the above link gives you the basic information you need to use Termite. Additional information is given in the posted Laboratory 06 Lecture.

Questions

1. After reset, index `USART_RxHead` has the value 0. Why is the first byte of data received placed in the `USART_RxBuf[1]` rather than `USART_RxBuf[1]`?

2. Why is it that when 5 characters are typed into Termite for transmission, 7 characters are actually sent? What are the extra two characters and where did they come from?

3. For Task 3, if the receive and transmit buffer sizes are set to 8 bytes, does the system work when you type in 6 bytes and send them? If not why not.

4. For Task 3, if the receive and transmit buffer sizes are set to 8 bytes, does the system work when you type in 7 bytes and send them? If not why not.

5. What can you conclude from the answers to questions 3 and 4 about the required relationship between the number of characters you type in and the required size of the buffers for the system to work?