ESE381 Embedded Microprocessor Systems Design II

Spring 2022, K. Short   revised March 2, 2022 8:26 pm

PRELIMINARY PRELIMINARY PRELIMINARY PRELIMINARY PRELIMINARY

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

To be performed the week starting March 6th.

Prerequisite Reading

1. Lecture 10: Circular Buffers
2. AVR306: Using the AVR USART on tinyAVR and megaAVR devices. You can skip section 3 of this document.
3. AVR128DB48 Data Sheet, Section 27: USART - Universal Synchronous and Asynchronous Receiver and Transmitter.

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 the interrupts, the appropriate interrupt service routine (ISR) can read or write the data to be transferred and go back to the other system tasks.

When using interrupts, the efficiency of the system can often be improved by using circular buffers. Characters to be transmitted can be placed in a software transmit buffer and transferred as a block when the microcontroller has time. Characters being received can be placed in a software receive buffer until an appropriate amount of data has been received and then processed as a block.

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 transmit and receive asynchronous serial protocol data using interrupts and circular buffers. You will verify the results of your work using the Saleae logic analyzer and TeraTerm terminal emulator.

Design Task 1: A More Flexible USART Initialization Function.
In Laboratory 05 you were not required to organize your program using functions, even though this approach was discussed in class. A simple function to configure a USART might have a single parameter that specifies the desired baud rate.

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

```
void USART3_init (uint16_t baud. uint8_t data_bits, uchar parity);
```

Write a test program that uses this function to initialize the AVR128DB48's USART3. Name the program `usart3_init_test`.

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 `usart_avr128.c` that modifies the program for the AVR306 application note to work with the AVR128DB48 and compiles using Studio 7.

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

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 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 port and USART3 registers and any variables you created.

(a) Verification with Saleae Logic Analyzer
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 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 `usart_avr128` using the program `usart_avr128` that you wrote. Set the compiler optimization to -Og. Build the program. Place any 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 port and USART3 registers and any variables you created.

When your program seems to be correct from single stepping run it full speed.

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.

Problems

1.