OBJECTIVES

- Learn about asynchronous communication, as well as about the Universal Synchronous/Asynchronous Receiver Transmitter (USART) system within the ATxmega128A1U.
- Create an application to browse and edit any data memory location within the ATxmega128A1U via a serial terminal graphical user interface (GUI).

INTRODUCTION

Beyond parallel communication, where multiple bits of data are transferred together via a bus of pins¹, serial communication is another method in which a processor can communicate with external devices. Serial communication involves the process of sending one bit of data at a time, while following a specific protocol. Serial communication is generally classified as either being asynchronous or synchronous. Unlike synchronous serial communication, where a common clock signal is used to determine when to send, receive, or sample data between multiple devices, there exists no synchronization signal for asynchronous serial communication. Instead, asynchronous communication protocols rely on the hope that a common transfer rate will be upheld by any systems using a chosen protocol; if this is not achieved, data transferred or received could be wrongly interpreted, or even entirely missed.

To facilitate asynchronous serial communication, a device known as a Universal Asynchronous Receiver/Transmitter (UART) is generally used in conjunction with both a transmitting device and a receiving device. A UART utilizes a clock signal to generate a transfer rate, denoted as the baud rate², and also uses two physical connections to communicate data: one pin to receive data (Rx), and one pin to transmit data (Tx). Within the ATxmega128A1U, several UART modules are available by way of the Universal Synchronous/Asynchronous Receiver Transmitter (USART) system.³

LAB STRUCTURE

Within this lab, you will begin to explore the asynchronous capabilities of the USART system within the ATxmega128A1U. In § 1, you will research relevant information regarding the USART system. In § 2, you will learn to transmit a text character from your microcontroller to a connected computer, via the USART system. In § 3, you will use an oscilloscope to measure the transmission signal generated from a relevant USART module. In § 4, you will learn to transmit a character string of arbitrary length⁴. In § 5, you will determine how to receive a character from a connected computer via the USART system. In § 6, you will learn to receive a character string of arbitrary length⁴. In § 7, you will configure the reception of data via the USART system to be interrupt-based. In § 8, you will create a program that implements a graphical user interface (GUI) to allow any data memory location within the ATxmega128A1U to be viewed and edited via a serial terminal program.

REQUIRED MATERIALS

- Atmel ATxmega128A1U AU Manual (doc8331)
- Atmel ATxmega128A1U Manual (doc8385)
- OOTB µPAD v2.0 with USB A/B cable
- OOTB Switch & LED Backpack
- Digilent Analog Discovery (DAD) kit with WaveForms software

SUPPLEMENTAL MATERIALS

- Using the XMEGA USART (doc8049)
- Getting Started Writing C-Code for XMEGA (doc8075)

¹ You have previously utilized parallel communication whenever using an I/O port. Later in this semester, you will work with another example of parallel communication when interfacing with an external bus interface (EBI) system.
² To learn more about baud rates in the context of this course, see Appendix B of this document.
³ In addition to providing the functionality of a UART, the USART system also provides a form of synchronous serial communication to the ATxmega128A1U.
⁴ By “arbitrary length”, we mean any length of available contiguous data memory locations.
In this section, you will explore documentation for the Universal Synchronous/Asynchronous Receiver Transmitter (USART) system within the ATxmega128A1U, which provides a form of both synchronous as well as asynchronous serial communication to the microcontroller. However, in this lab, we will only learn about the asynchronous form of the system.

1.1. Read any relevant parts of §23 (USART) within the 8331 manual to learn about the asynchronous abilities of the USART system available within the ATxmega128A1U.

On the μPAD, the ATxmega128A1U uses a specific USART module to transfer data, via an EDBG chip, to and from the USB type B port also located on the μPAD; this allows communication between the USB type B port and a USB type A port on a computer, whenever the appropriate USB A/B cable is connected. (Note that the EDBG chip is not shown on the μPAD schematic due to the request of the manufacturer.)

NOTE: Be aware that the microcontroller receives data transmitted from the EDBG chip and transmits data to the receiving pin on the EDBG. More specifically, the signal labeled EDBG_USART_CDC_RX denotes the Rx signal for the relevant USART module and the signal labeled EDBG_USART_CDC_TX denotes the Tx signal.

1.2. Review the relevant μPAD schematic to identify which USART module is used to communicate with a connected computer.

PRE-LAB EXERCISES

i. What is the maximum possible baud rate that you could use for asynchronous communication within the USART system of the ATxmega128A1U, if the microcontroller is configured for a clock frequency of 2 MHz? Support your answer.

ii. Explain the purposes of buffering in the context of the USART system within the ATxmega128A1U.

iii. If an asynchronous serial communication protocol of 8 data bits, one start bit, one stop bit, no parity, and baud rate of 1 MHz was chosen, calculate how many seconds it would take to transmit the ASCII character string “Dr. Schwartz saw seven slick slimy snakes slowly sliding southward.” (This string has 67 characters.) Support your answer.

2. USART, CHARACTER TRANSMISSION

In this section, you will write an assembly program (lab5_2.asm) to configure the appropriate USART module within your microcontroller to send data to your computer via its USB port. In this lab, we will use the following asynchronous serial communication protocol: even parity, 8 data bits, 1 start bit, 1 stop bit, and 57,600 bps (bits per second) baud rate.

2.1. Create an assembly program (lab5_2.asm). In this program, first create the following two subroutines.

2.1.1. USART Initialization (USART_INIT). This subroutine should initialize the necessary the necessary USART module.

2.1.1.1. Set the data direction of the appropriate USART transmit pin.

2.1.1.2. Configure the USART module for the appropriate mode (synchronous, asynchronous, etc.), as well as configure the necessary amount of data bits, type of parity, and number of stop bits.

2.1.1.3. Set the baud rate by storing the appropriate value in the baud rate registers. See Appendix B for a discussion regarding baud rates. You can use the excel workbook available on our website (Baud Calculator) to verify any baud rate calculations, but be sure that you know how to calculate by hand the necessary values for your lab quiz and exams. (When configuring baud rate registers, it is useful to use assembler directives when programming in assembly, or macros when programming in C, which will be relevant later.)

2.1.2. Output Character (OUT_CHAR). This subroutine will output a single character to the transmit pin of a chosen USART module. It is assumed that a character is passed into the subroutine via a general-purpose register (e.g., R16 or R17).

2.1.2.1. At the beginning of this subroutine, check if there is currently an ongoing transmission in the relevant USART module; if there is, wait until it has been completed. An appropriate interrupt flag should be polled to handle this, i.e., do not use an interrupt.

2.1.2.2. Transmit the character passed into the subroutine.

2.2. Next, create a main routine within the assembly program to continually transmit the ASCII character U (i.e., the capital letter U) utilizing the OUT_CHAR subroutine.

You will need to use a serial terminal program on your computer to view any data transmitted by your microcontroller. Basic information on how to configure/use PuTTY, a popular serial terminal program, is given in Appendix A of this document. Some other popular terminal programs are X-CTU, RealTerm, Bray Terminal (also known as Br@y++ Terminal), MobaXterm, and HyperTerminal. There is even a terminal within the Data Visualizer extension of Atmel Studio. You may use any serial terminal program for this course, as long as it has all features needed.
2.3. Use a serial terminal program on your computer, e.g., PuTTY, to test your assembly program and verify that the ASCII character U is continually transmitted.

3. USART, MEASURING BAUD RATE

In this section of the lab, you will use your DAD to measure the configured baud rate of your USART module, and to view the transmission frame for the ASCII character U when using the asynchronous serial communication protocol defined above. Unfortunately, on the µPAD, there is no practical way to access the pins used for the USART module that communicates with a connected computer. Therefore, you will need to configure a separate USART module.

NOTE: You will NOT be able to communicate with a connected computer while using a separate USART module, as no other module is connected to the available EDBG chip.

3.1. Create an assembly program (lab5_3.asm) to configure a USART module for which you have access to probe via the µPAD and then continually transmit the ASCII character U within a main routine. (Other than the different USART module, this program should be unchanged from your previous program.)

3.2. Use the Scope feature within WaveForms, along with your DAD, to measure the width of both a single data bit and a single character transmission frame; verify that the defined protocol is met (i.e., 57,600 Hz baud rate, even parity, 8 data bits, 1 start bit, and 1 stop bit).

PRE-LAB EXERCISES

iv. Why might the ASCII character U be chosen for the above tasks? Recall that the ASCII character U is equivalent to 0x55 = 0b 01010101.

4. USART, STRING TRANSMISSION

Now that you have a method to output a single character via the USART system, you should be able to easily create a routine to output a character string of arbitrary length.

4.1. Create an assembly program (lab5_4.asm). First, copy the subroutines used in § 2. Then, write the following subroutine.

4.1.1. Output character string (OUT_STRING). This subroutine should output a character string stored in program memory, using the appropriate USART module. When this subroutine is called, it will be assumed that the Z register already points to the beginning of a character string within memory, i.e., any main program utilizing this subroutine must properly configure the Z register before calling the subroutine.

4.1.1.1. Read the character pointed to by Z and increment the pointer.

4.1.1.2. For each non-null character, call the subroutine OUT_CHAR; when a null character is found, return from the subroutine.

4.2. Create a main routine within the relevant assembly program to output your complete name, using the OUT_STRING subroutine. Use a terminal program on your computer to test your assembly program.

NOTE: Recall that ASCII characters can be referenced in Atmel Studio individually, by using single quotes (e.g., 'A'), and as a string, by using double quotes (e.g., "this is a string of ASCII characters").

5. USART, CHARACTER INPUT

In this section, you will begin to configure the appropriate USART module to receive serial data from your computer.

5.1. Create an assembly program (lab5_5.asm). First, copy the subroutines used in § 4, and edit the USART_INIT subroutine to additionally enable the receiver within the appropriate module. Then, write the following subroutine.

5.1.1. Input character (IN_CHAR). This subroutine should receive a single character with the relevant USART module and return the received character to the calling procedure via a specified general-purpose register (e.g., R16 or R17).

5.1.1.1. Check if a character has been received (by polling an appropriate interrupt flag), and if not, keep checking until one has been received.

5.1.1.2. Read the received character from the appropriate buffer and return the character to the calling procedure.

5.2. Design a main routine within the relevant assembly program to continually echo (i.e., re-transmit) any character received by your microcontroller back to your computer, utilizing the IN_CHAR and OUT_CHAR subroutines. Use a terminal program on your computer to test your program.

6. USART, STRING INPUT

Now that you have a method to input a single character via the USART system, you should be able to create a routine to input a character string of arbitrary length.

6.1. Create an assembly program (lab5_6.asm). First, copy the subroutines used in § 5. Then, write the following subroutine.
6.1.1. **Input character string (IN_STRING).** This subroutine should receive a character string of arbitrary length with the relevant USART module and store the received character string to some memory location(s) within data memory via the Y index. Whenever this subroutine is called, it will be assumed that the Y index already points to the beginning of some pre-allocated contiguous memory locations, i.e., any main program utilizing this subroutine must properly configure data memory and the Y index before calling the subroutine.

6.1.1.1. Continually read characters from the appropriate USART module with the IN_CHAR subroutine. For each character not equal to the carriage return$^5$ ASCII character (CR, 0x0D) nor the backspace character (BS, 0x08), store the character in the next appropriate data memory location with the Y index; when a backspace character is found, decrement the Y index to allow for another character to be written, and when a carriage return character is found, store a null character at the end of the input string and return from the subroutine.

6.2. Create a main routine within the relevant assembly program to input your complete name, using the IN_STRING subroutine, and then echo the relevant input string. To echo this string stored in data memory, you will not be able to use the OUT_STRING designed in § 4 (since this subroutine was designed to read from program memory), however another subroutine with very similar functionality could be created. Make sure to allocate an appropriate amount of data memory as well as configure any necessary indices, e.g., Y. Use a terminal on your computer to test your program. Verify that backspace functionality is correct.

**7. USART, INTERRUPT-BASED RECEIVING**

In this section of the lab, you will learn how to configure interrupt-based receiving within the USART system by creating an interrupt-driven echo program for the appropriate USART module.

7.1. Create an assembly program (**lab5_7.asm**) that utilizes the receive complete (RxC) interrupt within the appropriate USART module to echo, i.e., re-transmit, any character received by your microcontroller back to your computer. Additionally, to demonstrate that your serial interrupt is independent from the rest of your program, continually toggle the **GREEN_PWM** LED within the main routine of your program. See the relevant µPAD schematic, if necessary. Use a serial terminal program to test your assembly program.

**NOTE:** Recall that interrupt service routines should generally be as short as possible; thus, it would generally be unwise to call a subroutine within an ISR.

**8. MEMORY GUI**

In this final section, you will combine concepts learned from within this lab and other material to create a **C program** that implements a graphical user interface (GUI) to allow some data memory locations within the ATxmega128A1U to be viewed and edited via a serial terminal program. You may choose whether or not your program utilizes interrupts.

Upon program start, an introductory prompt should be output to the terminal, describing in detail the purpose of the application. For example, the following text would suffice:

*Welcome to the Memory GUI!*

*This program allows a user to view as well as edit any data memory location within the connected ATxmega128A1U.

**NOTE:** When applicable, text should be followed by a blank line (i.e., a carriage return followed by a line feed), or anything else to make the output easily readable.

After the introductory prompt, the program should continually prompt the user to either to read from, or write to, data memory locations. The following text may be used as such.

5 A carriage return (CR, 0x0D, 'v') ASCII character is generated when you hit the Enter key on a connected computer keyboard. In the Windows® operating system, a carriage return generally causes the cursor to just move back to the beginning of a line. To advance one line in *Windows*, a line feed (LF, 0x0A, ‘n’) ASCII character is normally needed. In *Unix®* and *Linux®*, a carriage return character generally causes the cursor to both advance to a new line and to move back to the beginning of the respective line. The separation in *Windows* stems from typewriter times, when starting a new line involved the two-step process of turning the platen (wheel) and paper before returning the carriage to the position denoting the beginning of a line.
either ‘A’ or ‘a’, either ‘B’ or ‘b’, etc. In this context, any input other than characters representing the possible hexadecimal values should be ignored.

Once a memory address has been entered, data should either be read from or written to the corresponding memory location.

Whenever data is to be read, the value within the corresponding memory location should be output to the terminal window on a new line, in a two-digit hexadecimal format. For example, if the data read from a memory location is equivalent to 0x37, the terminal output should read similar to the following.

“0x37”

Whenever data is to be written, the user should be prompted to enter an 8-bit value in hexadecimal format. The following text may be used as such.

“Data: 0x”

Again, invalid input should be prevented, i.e., any input other than characters representing the possible hexadecimal values should be ignored. Whenever valid data is entered, it should be stored to the specified memory location.

Once data has been either read from or written to the relevant memory location, the user should be prompted to either read from or write to another memory location.

8.1. Create a C program (lab5_8.c) to implement the memory GUI described above. For simplicity, you may utilize the usart.h and usart.c files provided on the course website, although, similar to most provided information, you are responsible for fully understanding them.

---

**PRE-LAB PROCEDURE SUMMARY**

1) Answer pre-lab exercises when applicable.
2) Become introduced to the USART system within the ATxmega128A1U in § 1.
3) Learn how to transmit a character via the USART system in § 2.
4) Measure USART character transmissions in § 3.
5) Learn to transmit a character string of arbitrary length via the USART system in § 4.
6) Create a subroutine to receive a character via the USART system in § 5.
7) Implement a subroutine to receive a character string of arbitrary length via the USART system in § 6.
8) Create an interrupt-driven echo program in § 7.
9) Construct a graphical user interface (in the C programming language) to be able to both view as well as edit all data memory locations within an ATxmega128A1U in § 8.
APPENDIX

A.  PUTTY

*Putty* is a lightweight terminal program with many features and settings, but for the purposes of this course, we will only need to use its serial operating mode. To start using *Putty*, you will first need to download the program to your PC.

A.1. To download *Putty* for 64-bit operating systems, click [here]; for 32-bit operating systems, click [here]. Once downloaded, run the executable.

Upon the program opening, the *Putty* configuration menu should be displayed, as shown in Figure 2. This configuration menu is used to select the operation(s) of the terminal application. There are a few things that must be changed before we can start communication between our computer and the microcontroller.

A.2. In the configuration menu of *Putty*, do the following:

A.2.1. Select the *Session* tab at the top left. Choose *Serial* as the connection type (on the right, next to *SSH*).

A.2.2. Select the *Serial* sub-tab on the left of the *Putty* configuration menu (located at the bottom of the *Connection* tab list).

A.2.3. Choose and enter the correct COM (communication) port that corresponds to the µPAD. To determine which COM port on your computer represents the µPAD,

A.2.3.1. Open the preinstalled *Device Manager* application in *Windows*, expanding the *Ports* section.

A.2.3.2. Disconnect your µPAD USB cable and make a note of the COM ports that are available. (It is possible that no ports may be shown.)

A.2.3.3. Re-connect your µPAD USB cable and notice the COM port was added to the list. This is the COM port that you should use in *Putty*, e.g., *COM1*, *COM2*, etc.

A.2.4. Enter the correct baud rate in the *Speed (baud)* textbox, select the correct number of data and stop bits, and also select the correct type of parity. Additionally, set *Flow Control* to *None*. (Make sure that the data bits, stop bits, and type of parity are all configured as they are in the USART system within the ATxmega128A1U.)

A.2.5. Once everything is configured, you can save your configuration settings so that you do not have to change them every time. To do this, do the following:

A.2.5.1. Navigate back to the *Session* menu, and in the textbox located under *Saved Sessions*, type something such as *3744 UART Config*. This will be the name used for your configuration.

A.2.5.2. Next, click the button to the right labeled *Save*. This will save your current configuration, so that you can access it for the next time you use *Putty*. (To load a saved configuration, you will need to first click on the appropriate configuration listed within *Saved Sessions* and then click the button to the right labeled *Load*.)

Now that everything is configured, you can open the terminal window by clicking the *Open* button located at the bottom right of the window.

NOTES:

- Configure *Putty* and open the terminal window **BEFORE** you debug/run your program in *Atmel Studio*.
- It is possible that the COM channel will change if you have different USB devices connected to your PC, or if you connect any USB devices in a different order. If this occurs, just repeat items 2 and 3 above to determine the proper COM port for your microcontroller.

Additionally, there is a setting in *Putty* that causes characters typed to the terminal to be echoed, i.e., displayed to the terminal automatically. This can be mistaken as a properly-working echo program, when in fact *Putty* might be the only source of echoing.

A.3. When applicable, to turn off the automatic echo setting, do the following:

A.3.1. Open the configuration menu of *Putty*.

A.3.2. Select the *Terminal* tab.

A.3.3. Under *Local echo*, select *Force off*.

You can find more detailed information on *Putty*’s website, or by clicking on the following link: *Putty User Manual*.
B. BAUD RATE VS. HZ VS. BITS/SECOND

In general, a **baud rate** represents the rate of **symbols per second**, where a **symbol** represents a relevant unit. In this course, the relevant symbol for a baud rate is a **bit**. Therefore, the overall unit of any baud rate in this course is **bits/second** (bps).

Additionally, since one bit of data is transmitted per one cycle of our processor clock, and since the unit of hertz (Hz) represents **cycles per second**, we can state that a baud rate of 1 bps is equivalent to 1 Hz.

To learn more about communication theory, take **EEL4514: Communication Systems and Components**.