1 Lab 7: UART: Buffers and receiving characters

1.1 Summary

This lab is completes basic communication your education technology by receiving data instead of just sending them. You will configure the USART of the microcontroller to receive incoming single characters, and use pointers and buffers to receive and process entire strings. Thereby you will likely encounter buffer overflows and others problems and learn about proper memory management.

1.2 Objectives

By the end of the lesson, you shall have completed the provided files. They extend the serial communication code from the last lab with a two new functions, which either receive a single character or an entire string. Furthermore, you will have to configure your terminal program for properly sending characters and string termination characters.
The working example program, the counterpart in your terminal program and the source code have to be shown and explained to the supervisors until the end of the lesson.

1.3 Duration and evaluation

One lab of 80 minutes. You have to present the running software, the sending and receiving of messages in the terminal program and your source code to the supervisors until the end of the lesson. Prepare to explain your code and change communication parameters upon request. Do not expect supervisors to send messages shorter than your buffer. Take precautions to handle messages of any arbitrary length without messing up your memory!

1.4 Further Information

The microcontroller used in this lab is an ATmega328P on a developer board called ATmega328P Xplained mini. LEDs, resistors, NTCs etc. are available as in the previous labs, but actually not needed this time.
To complete this lab, it is important to understand buffer concepts and what the difference between a bunch of characters and a string is. Pay special attention to things like string termination characters, whitespace characters and other things that can possibly be sent over the line, but not be displayed as a visible ASCII characters.
Hint: Think about how to encode your data. Do you really need to cover the binary transmission of a full byte (any value from 0-255) and its processing? May it be easier to use

ASCII or another encoding, where some byte values are reserved for metadata or control operations rather than information / content?
On a microcontroller, your RAM is very limited. You will not be able to store strings of arbitrary length in them. How could you possibly react when the incoming message exceed the buffer length? There are plenty of different concepts for that. Which one(s) would you apply here? Why?

1.5 Recommended reading

Atmega328 [1]**:** contains the datasheet and further information about the ATMega328p microcontroller. Datasheets with register descriptions are a thrilling lecture and should not be missed. Seriously knowing which registers there are (and what they can do) is essential for embedded systems programming and especially for using peripheral modules. Have a look at it!
Xmini [2]**:** provides the full developer board hardware documentation.
Man minicom: gives you the manual of the minicom terminal program (if it is installed type this command on your CLI). Despite minicom, there are several other terminal programs which you can use, e.g. gtkterm or CuteCom. They are all installed in the VM image provided to you for this course. Use whichever you want.

[1] `http://www.atmel.com/devices/atmega328p.aspx`
[2] `http://www.atmel.com/tools/mega328p-xmini.aspx`

2 Task1: Complete the provided source code and receive data from a terminal program on your PC

2.1 Required preparation

- You need to bring your own device (e.g. laptop) with a working installation of the avr-gcc cross-compiler and the avr-libc library. See Lab 2: Hello, World!" on Atmel AVR Microcontrollers for details.
- Understand how serial communication works.
- Understand the difference between characters and strings.
- Learn about string termination and buffer concepts.
- Check the provided source code. You are supposed to extend the file USART/USART_implement_me.c. You can reuse the code from the last lab, but you will also need to bring some more. Prepare these lines - configuring the USART to receive characters is actually easy, if you know what you are doing.
- Have an appropriate buffer concept ready. A proof-of-concept implementation or pseudo code will help.

2.2 Class work

You are supposed to add the missing code lines in USART/USART_implement_me.c. First, add support for the receiving of single characters. When this works, add support for receiving entire strings, too. Extend this function with a proper buffering concept. Try what happens if you receive more characters than your buffer can hold (undefined behaviour, writing to memory cells outside of your buffer, spontaneous resets etc. are no options).
Note that you might or might not have to extend your USART_Init() function from the previous labs in order to properly initialise the receiving of characters. This depends on how you configured the according registers before.

2.3 CLI instruction cheat sheet

```
avr-gcc -Wall -Wextra -Wpedantic --std=gnu99 -mmcu=atmega328p -Os -o USART_3.elf USART_3.c USART/USART_implement_me.c

avr-objcopy -j .text -j .data -O ihex USART_3.elf USART_3.hex

avrdude -p atmega328p -F -c stk500v1 -P /dev/ttyACM0 -b57600 -D -V -U flash:w:USART_3.hex:i

minicom -s
man minicom
cutecom
gtkterm
```