1 Lab 5: UART: Sending single characters

1.1 Summary

This lab introduces an important features of MicroController Units (MCUs): Serial communication. It allows you to transmit text and binary content between different places, which allows for a much greater level of control than with single GPIO ins or analog signals.
Most microcontrollers have one or several USARTs, which stands for Universal Synchronous and Asynchronous Receiver and Transmitter. You will learn in the lectures what synchronous and asynchronous communication is. You will also learn about different bus systems and protocols.
For the moment and in this lab, we just want to transmit a single character at a time. You will complete the necessary code, configure a terminal program on your computer and transmit a text message from your microcontroller to your computers screen.

1.2 Objectives

By the end of the lesson, you shall have completed the provided files. By adding the proper setup code and character output routine, the test program can transmit a message to a terminal program on your computer. The working hardware, the appearing text message in the terminal program and the source code have to be shown and explained to the supervisors until the end of the lesson.

1.3 Further Information

The microcontrollers used in this lab is an ATmega328P on a developer board called ATmega328P Xplained mini and MSP430F5529. LEDs, resistors, NTCs etc. are available as in the previous labs, but actually not needed this time.
In this lab, you are a supposed to transmit a single character (per time) over a serial interface to the computer. A character in this sense is an arbitrary 8bit value. Humans tend to dislike raw numbers like 0x44, which is why we assign humanreadable symbols to these values. The de facto standard for this is the ASCII code. For example, in ASCII the value 0x44 stands for the letter D. We use terminal programs to send and receive those values, and we can also tell those programs to translate the raw information into ASCII or other codes for us (in the same way, you can type E instead of 0x45 if you want to send something to the microcontroller).
In this lab, you will be given a prewritten example program. You have to complete it the code for using the USART interface is missing. Then you have to configure a terminal pro-

gram on your computer and to connect it to the microcontroller.
The developer board uses a physical USB connection. Your microcontroller and terminal program do not know this, though, and act as if there would be a direct serial link between them. On the Atmel board, there is a second microcontroller (an ATMega32U4, here also called mEDBG chip), which translates all the serial data from your microcontroller into USB packages. Similarly, the driver in your operating systems receives these USB packages, extracts the pure serial data and forwards them to a serial interface. Your terminal program can connect to this serial interface and does not need to care about USB at all. This is why this serial connection is also called a virtual serial link. On Linux, this serial port is typically /dev/ttyACM0 or /dev/ttyUSB0. On Windows it could show up as COM3 or COM12 in the device manager.
Some microcontrollers can also speak USB directly, or one of many other bus systems. The ATMega328P has no USB support (in fact, there are ways to make it work anyways, but this is far above the level of this course). For the moment be happy with the serial port it is the simplest system which you will find.
Note: The mEDBG chip on these boards is limited to 57600 baud/s maximum. Slower baud rates work, higher dont.

1.4 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 establish communication with 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 avrgcc crosscompiler and the avrlibc library. See Lab 2: Hello, World!" on Atmel AVR Microcontrollers for details.
- Understand how serial communication works. What are baud rates, parity, data bits, start and stop bits about? What are they good for?
- Understand what ASCII is. You do not need to learn the entire ASCII table by heart, but having an idea which symbols it contains is never wrong.
- Check the provided source code. There are only a few lines missing, all in the file USART/USART_implement_me.c. Prepare these lines its mostly about setting some register values. You shall configure your serial interface to 57600 8N1 (see the files if you are not sure what this means).
- Familiarize yourself with a terminal program of your choice (suggestions see above). If nothing is received, you should be very sure that it is not the terminal programs fault. Otherwise you have to search the error on both sides avoid that by proper preparation!
- The main file USART_1.c demonstrates how you can repeatedly send a single character to form a string. This is clumsy, but sufficient for the moment. Sending real strings and more follows in the next labs.
- Tip: Your code should ensure that you do not attempt to send another character before the transmission of the first one is finished. Otherwise youll mess up your message.
- This time the code consists of more than one file. The provided example code demonstrates how to use headers (.h) and source (.c) files, how to properly include them etc. have a look at that!
- Also note that each .c file has to be compiled separately (but no .h file). Otherwise your code wont work. Pay attention to always compile all files in some cases, you might change things, but forget to recompile this particular file. If your code then links with an old version of this compiled file (that was still stored somewhere), things

can get pretty nasty. You dont want to debug this, so always doublecheck your CLI commands.

2.2 Class work

You are supposed to add the missing code lines, configure your terminal program and see how the test string is received. Feel free to modify this message and try out different things.. The result and the source codes must be shown to a supervisor before the end of the lesson.

2.3 [Linux] CLI instruction cheat sheet

```
avr-gcc -Wall -Wextra -Wpedantic --std=gnu99 -mmcu=atmega328p -Os -o USART_1.elf USART_1.c USART/USART_implement_me.c

avr-objcopy -j .text -j .data -O ihex USART_1.elf USART_1.hex

avrdude -p atmega328p -F -c stk500v1 -P /dev/ttyACM0 -b57600 -D -V -U flash:w:USART_1.hex:i

minicom -s
man minicom
cutecom
gtkterm
```