1 Lab 2: "Hello, World!" on Microcontrollers

1.1 Summary

This lab introduces C programming on MicroController Units (MCUs). You shall learn how to write a valid example program in C on a regular computer, how to crosscompile it for another architecture and how to load and execute it on a given microcontroller. Thereby, this lab introduces one possible C development toolchain as commonly used for microcontroller development. The software created in this development process is also called firmware.
As microcontrollers usually come without a display or other means of text output, this example uses a blinking LED attached to the microcontroller instead.

1.2 Objectives

By the end of the lesson, you shall have written, crosscompiled, loaded and executed a C program on a microcontroller. This program shall let an LED attached to this microcontroller blink at a frequency of approximately 1Hz (or another frequency).

1.3 Evaluation

You have to present the working Hello, World! program (blinking LED) to the supervisors until the end of the lesson.

1.4 Further Information

The microcontroller used in this lab is an ATmega328P on a developer board called ATmega328P Xplained mini. It stems from the widespread AVR line of the manufacturer Atmel. The according avr toolchain consists of avrgcc with the avrlibc for writing and compiling code, the avr binutils for numerous smaller steps like adjusting the output file formats and avrdude for loading the final binary code into the microcontroller. The Recommended Reading section contains material giving a first overview over the avr toolchain. Understand how it works in principal, but do not worry about single parameters. Its details will be explained in the lab.
The avrgcc toolchain differs from the AVR Studio integrated development environment (IDE) mentioned in the developer boards user guide.
The actual microcontroller development board and accessories (cables and components) will be given to you at the beginning of the lab session.

1.5 Recommended reading

Xmini [1]**:** is the manufacturer documentation for the hardware used in this lab. Note the "documents" tab containing a user guide and the links to the datasheets of the microcontroller used.
Compilation [2]**:** introduces the very basics of the gcc based toolchain for the microcontrollers that can be used in this lab. Reading it helps understanding the cross-compilation process. Really, you will need those tools! Invest an hour to understand what they are and research a bit more, if you are in doubt or just curious.
Atmel Studio [3], the integrated development environment. **Arduino** [4], open-source electronics platform.
Google search, "Getting Started AVR": there is a lot of material that explains how to use AVR microcontrollers.

[1] `https://www.microchip.com/DevelopmentTools/ProductDetails/ATMEGA328P-XMINI`
[2] `http://www.nongnu.org/avr-libc/user-manual/overview.html`
[3] `https://www.microchip.com/mplab/avr-support/atmel-studio-7`
[4] `https://www.arduino.cc/en/Main/Software`

2 Task1: Write a "Hello, World!" program on AVR

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. You can also use Atmel Studio or Arduino.

You must familiarize yourself with the used hardware by reading the manual and with the according software tools (see recommended reading). A basic understanding of how to operate the development board is necessary in order to complete the lab in time. This includes viewing the schematic in order to understand how to control the LEDs on this board. Furthermore, you should have a look at the example code below and understand its basic mode of operation in order to keep up with the explanations of the instructor.

2.2 Class work

All students will write an identical Hello, World! example program on their computer. It lets an LED attached to the microcontrollers pin blink. You will crosscompile this program, load it into the provided developer board and execute the program there. The result must be shown to a supervisor.

2.3 Source Code and CLI instruction cheat sheet

```
1  #define F_CPU 16000000UL
2  #define LED_PIN 5
3  #include <avr/io.h>
4  #include <util/delay.h>
5
6  int main (void)
7  {
8    // sets the direction of this pin to output
9    DDRB |= (1 << LED_PIN);
10
11   while(1)
12   {
13     // toggles the state of this pin
14     PORTB ^= (1 << LED_PIN);
15     // busy waits the specified time
16     _delay_ms(500);
17   }
18 }
```

2.4 Optional (for Linux)

```
avr-gcc -Wall -Wextra -Wpedantic --std=gnu99 -mmcu=atmega328p -Os -o hello_avr.elf
    hello_avr.c

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

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

2.4.1 Notes

- Use less v (verbose) flags in order to reduce the output produced by avrdude.
- avrudes F (force) is a parameter that tells the program to go on, even if it detects errors during programming. This is normally a very bad idea, as errors almost always mean that something is wrong. In this particular case it is the Optiboot bootloader on the developer board itself. At the time of writing (201511), Optiboot contains a bug which prevents avrdude from correctly reading the device signature of the microcontroller. This signature is used to verify that the chip you are trying to program is actually the same one that you specified on the command line. By using F, you ignore the outcome of this check (and any other problem). If all other parameters are correct, avrdude can still flash the user program as intended, so using it is okay. As soon as Optiboot is fixed, the boards can be updated and F should not be used anymore.

3 Task2: Write a "Hello, World!" program on Texas Instruments MSP430 Microcontrollers

3.1 Summary

This lab repeats the previous exercise on a microcontroller from a different manufacturer. Instead of an Atmel AVR, you will use an MSP430 series microcontroller by Texas Instruments (TI) this time. Changing your target platform always requires some adaptations in the way you program. Different manufacturers have different ways of organising their datasheets, and the mnemonics (names and shortcuts you use during programming) change as well. But a variable is still a variable, and a bit is still a bit. You will see the similarities and fundamental concepts that all microcontrollers build on. The vast majority of the required knowledge is platform agnostic, and you will see that changing the brand is quite a harmless process. Always structure your code well and separate generic parts from hardware specific parts then porting your software is easy and fast.
Just as the last time, we will make an LED blink. The toolchain is still gcc this time in the flavour of ms430gcc instead of avrgcc. You also can use Code Composer Studio or Energia.

3.2 Further Information

The microcontroller used in this lab is an MSP430F5529 on a developer board called Texas instruments LaunchPad. It is also referred to by its product number MSPEXP430F5529LP. This hardware uses a programming and debug interface called ezFET lite. A driver library called libmsp430.so from TI is required to use it (see recommended reading).

3.3 Recommended reading

MSP-EXP430F5529LP [5]**:** is the manufacturer documentation for the hardware used in this lab. Have a look at the hardware before coming to the lab!
MSP Wiki [6]**:** is a wiki containing plenty of information regarding the hardware, but also software examples.
GCC [7]**:** provides information about the gcc based toolchain. msp430gcc has been a community project for a long time, but is developed under leadership from Texas Instruments by now. It comes with a distribution of the newlib standard C library.

[5] `http://www.ti.com/tool/MSP-EXP430F5529LP`
[6] `http://processors.wiki.ti.com/index.php/MSP430F5529_LaunchPad`
[7] `http://www.ti.com/tool/msp430-gcc-opensource`

Launchpad [8]**:** contains the full software example for a blinky LED on this board. **Code Composer Studio** [9], integrated development environment.
Energia [10], open-source electronics prototyping platform.
Energia

3.4 [Linux] Obtaining the libmsp430.so:

Simplest way: A statically linked version of it, build on another machine before, can be used (the supervisors have one for 64 bit systems).
Better way: See http://energia.nu/ Energia is TIs answer to the Arduino environment. The download packages contain a workign copy of libmsp430.so in the 32 and 64 bit editions.
The hard way: See http://www.ti.com/tool/mspds and get the mspds open source package. It contains all means to build the libmsp430.so by yourself. Note that it is particularly difficult to build this library on an Ubuntu system, as several dependencies have to be compiled from source as well (they are not available in the repositories, e.g. the hdiapi).
In all cases, make sure to set your library path correctly (e.g. export it in .bashrc) so that mspdebug can find it. Alternatively, place the library in /usr/lib. Type mspedebug tilib to see if it complains about the missing library or not.

3.5 Required preparation

You need to bring your own device (e.g. laptop) with a working installation of the msp430gcc crosscompiler, the mspdebug utility and the newlib library. You can also use Code Composer Studio or Energia.

[8] `http://wisense.in/docs/TI-2013-MSP430F5529-LaunchPad-CCS.pdf`
[9] `http://www.ti.com/tool/CCSTUDIO`
[10] `http://energia.nu/download/`

3.6 Source Code and CLI instruction cheat sheet

```c
#include <msp430.h>
#include <stdint.h>
int main(void)
{
WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer, it runs by default.
P1DIR |= 0x01; // Port1 direction register, Pin0 as output

while (1) // Infinite loop
{
  P1OUT ^= 0x01; // Toggle P1.0 (XOR)
  for(volatile uint32_t i=0; i< 20000; i++); // Busy wait delay
}

return 0;
}
```

3.7 Optional (for Linux)

```
msp430-gcc -Wall -Wextra --std=gnu99 -mmcu=msp430f5529 -Os -o hello_msp.elf hello_msp.c

//optional, one can also  flash the .elf file directly
msp430-objcopy --output-target=elf32-msp430 hello_msp.elf hello_msp.bin

mspdebug tilib -d /dev/ttyACM0 "prog hello_msp.bin"
```

3.8 Notes

- On this developer board, after flashing, you have to press the RESET button in order to start your program! Don't rely on the output of *mspdebug* for that.