TECHNISCHE UNIVERSITÄT
CHEMNITZ

Professur
Betriebssysteme

Embedded Software Lab

Lab 1 - Hello ATMega!

Christine Jakobs, Martin Richter

The Gertboard is the hardware foundation for all upcoming work in the Embedded Software Lab. The following tasks will help you to get familiar with both the development environment and the target hardware.

The Gertboard has, beside several input and output connectors, an ATMEL ATMega 328P microcontroller on board. This chip provides an 8-Bit RISC Harvard processor, 2kByte of SRAM for runtime data, 32kByte of non-volatile Flash memory for code, and 1kByte of non-volatile EEPROM for storing information at run-time.

Flash and EEPROM memory can be modified through the 6-pin header (J23) on the Gertboard, or directly by code running on the chip. Furthermore, most microcontroller pins are connected through to the Gertboard (PC0-PC7, PD0-PD7, PB0-PB5).

The Gertboard is powered by the RaspberryPi computer it is attached to. The execution of flashed code starts immediately when the controller is powered on or reset.

Initial Remarks

The tasks should be solved in the given order. If you do not manage to solve all tasks in preparation of the next lab meeting, make sure that you solve them afterwards. Your final grade at the end of the semester depends on the availability of *all* solutions in your code repository (see below).

The presentation of your results during the lab slot is oral. There is *no need for preparing slides* or any other kind of written material.

Be ready to answer questions. Have your code, positive and negatives experiences and according documentation available.

For multi-person teams, all must participate in the presentation and discussion of results.

For the coding part, please avoid playing with the boot / read / write lock bits, playing with the Fuse bits or writing to the boot loader memory. Writing to EEPROM regions is also not needed at the moment.

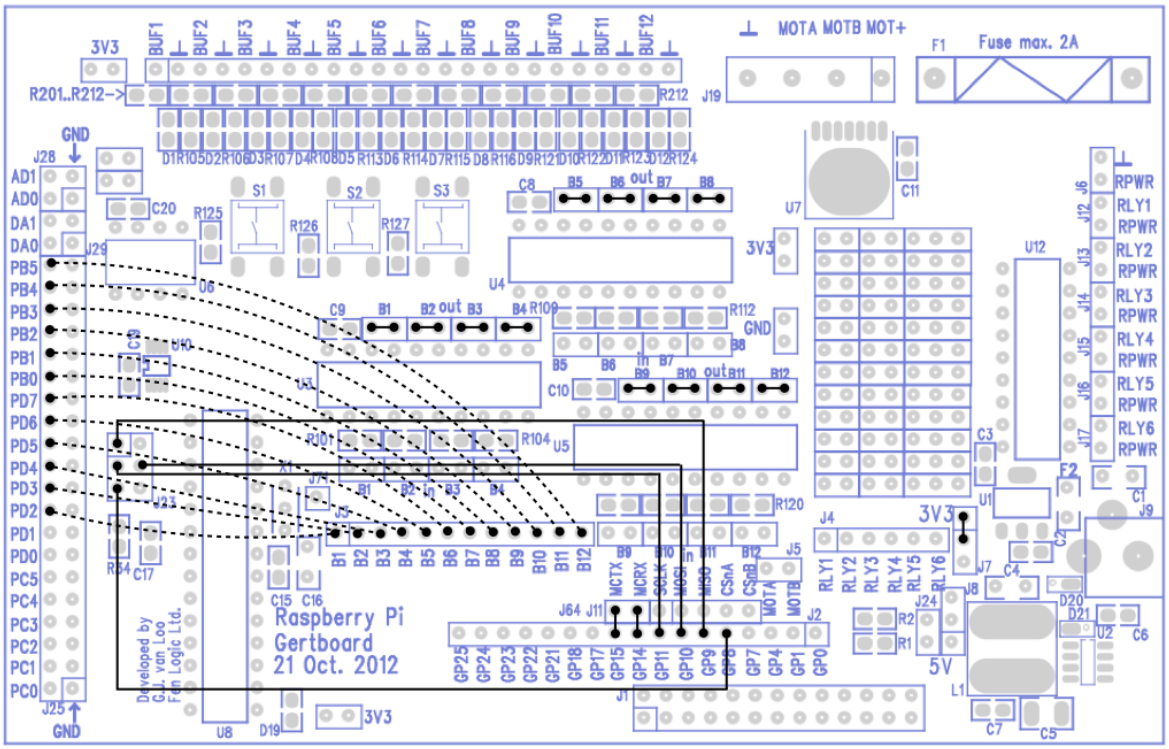

Figure 1: Gertboard wiring scheme.

Task 1.1

Consult the Gertboard manual [6], the microcontroller manual [1] and self-chosen sources to learn how the ATMega can be programmed through the *Serial Peripheral Interface (SPI)* protocol.

Explain the basic concept during your presentation, including the purpose of the `avrdude` tool. What is the relationship between such tools running on the Raspberry, GPIO pins, ATMEL pins and the different operation modes of the microcontroller?

Task 1.2

Prepare the Gertboard wiring according to Figure 1. If a wiring already exists, check it for correctness.

What is the purpose of the different connections? Consult the Gertboard manual [6] and explain your insights during your lab presentation.

Task 1.3

The RaspberryPi installation offers the `avr-gcc` cross-compiler for creating microcontroller code. Write a first C-program that performs some non-I/O *libc* test calls (e.g. string handling), based on `avr-libc` [3].

Compile the binary and send it to the microcontroller memory with the

```
avrdude ...  -U flash:w:<filename.hex> ...
```

command. The usage of the Arduino IDE is NOT allowed here.

Prepare a `Makefile` for the code compilation, linking and flashing procedure. It is supposed to be used in the upcoming tasks. The `Makefile` must be shown and explained during the presentation of your results.

Task 1.4

Create a Git repository for your sources. Make sure to give read-only access to the supervisors. Make sure that you store all your source code and the `Makefile` in the repository. Send its URL to the supervisors by e-mail.

Explain the basic ideas of Git (push, pull, clone, checkout, commit) in your presentation.

Task 1.5

Write a microcontroller program that turns the Gertboard LEDs on and off.

First, all LEDs should be switched on, one after the other. When all are on, switch them off again, one after the other, and start over. Ordering and speed are not relevant here.

The wiring plan in Figure 1 is suitable for this task.

Push the code to your Git repository. Show the working code in your presentation.

Task 1.6

Extend your program so that it changes the LED animation speed, based on character input read from a serial connection between the ATMega chip and RaspberryPi.

The wiring plan in Figure 1 is suitable for this task.

This task demands the implementation of serial communication on the ATMega chip. Consult the microcontroller manual [1] and additional sources such as [2, 4] to learn all the relevant details.

It is ok to rely on `avr-libc` functions to solve this task [5].

On the RaspberryPi side, you can use the installed `minicom` tool to send characters of the serial connection. The serial GPIO device should be available at `/dev/ttyAMA0`.

The usage of the Arduino IDE is NOT allowed here. Using wrapper libraries (e.g. for Arduino) is also NOT allowed.

Push the code to your Git repository. Show the working code in your presentation.

Task 1.7

Extend your program so that the chosen blinking mode is indicated as debug output on the serial `minicom` console.

Push the code to your Git repository. Show the working code in your presentation.

References

[1] Atmel. ATmega48A/PA/88A/PA/168A/PA/328/P Data Sheet. `https://osg.informatik.tu-chemnitz.de/lehre/emblab/atmel.pdf`.

[2] Mikrocontroller.net. AVR-GCC-Tutorial/Der UART. `http://www.mikrocontroller.net/articles/AVR-GCC-Tutorial/Der_UART`.

[3] AVR Libc Project. avr-libc 2.0.0 - Standard C library for AVR-GCC. `http://www.nongnu.org/avr-libc/`.

[4] Yash Tambi and Mayank Prasad. The USART of the AVR. `http://maxembedded.com/2013/09/the-usart-of-the-avr/`.

[5] Mika Tuupola. Simple Serial Communications With AVR Libc. `http://www.appelsiini.net/2011/simple-usart-with-avr-libc`.

[6] Gert van Loo and Myra VanInwegen. Gertboard User Manual, Revision 2.0. `https://osg.informatik.tu-chemnitz.de/lehre/emblab/gertboard_manual_2.0.pdf`.