Principle of operation and environment of the AVR Boot-loader optiboot

Karl-Heinz Kübbeler
kh_kuebbeler@web.de

November 29, 2025

Contents

Preface

My interest for the AVR boot-loaders begun, as some users had told me their interest to run the transistor tester software at some boards of the Arduino family. Of course the transistor tester software does not run as Arduino Sketch. The Arduino development environment Sketch is only used to show the output of the serial interface. The transistor tester software do not use the Arduino library. This is not necessary to use the boot-loader.

The boot-loader is a little program, which can receive program data from a serial interface from a host (PC) and can put this data in the instruction storage (flash) of the micro-controller. Because the transistor tester software use very much program storage, the boot-loader should use as less program memory as possible for his program. The boot-loader should also be able to write data to the nonvolatile data storage of the AVR, the EEprom. So the target was specified. I search a boot-loader, which support the writing of flash and EEprom, but use only little space in the flash memory for it's own code.

Chapter 1

Principle function of a boot-loader

A boot-loader is a little program, which can receive program data from a interface and store this data in the instruction memory of a processor. Typically the received program is also started at the end of transmission. With this method a computer with writable instruction memory is able to run any application program.

In principle the BIOS of a PC is also a boot-loader, but the BIOS is extended for the function to select a interface to fetch the program data. You can select a chain of peripheral equipment, which is tested for existence of program data. Often is a second stage of loader started, which can choose more selections like different operating systems or boot options.

For the micro-controllers the function of a boot-loader is designed more simple. Only one interface is preselected and there are no further options selectable during operation. A characteristic for the mode of operation of a boot-loader is the type of instruction memory. If the instruction memory of the computer is build with RAM (Random Access Memory), the boot-loader must be sure to start any application program only, if it is loaded just before.

For a micro-controller with non-volatile instruction memory (flash), the boot-loader can assume, that a application program has been loaded some time ago to the instruction memory. Therefore the boot-loader try to start a application program every time after waiting for new program data a appropriate time. It doesn't matter, if new program data are received before or not. Even if there was never loaded any application program, the result is not fatal. The facility to load a application program later is still available. The lack of any application program let the boot-loader resume with the next try to get program data from the serial interface.

The figure 1.1 shows the principle function of boot-loaders, which receive their data from a serial interface.

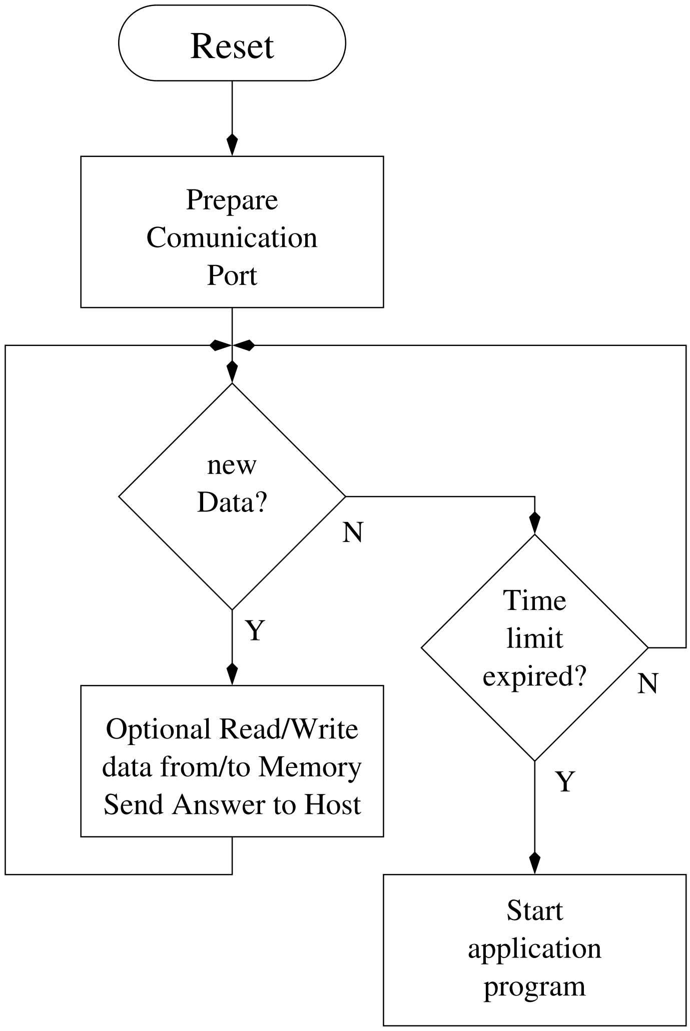

Figure 1.1. Principle function of a boot-loader

The transmitter process at the PC will reset the AVR target processor at the beginning of transmission. If the reset is not done automatically, you must reset the AVR processor manually. The PC tries to start the communication with the AVR processor by sending a data byte to the serial output and wait for any serial response of the AVR processor. If the answer is not received in a appropriate time, the procedure is repeated some times. The boot-loader program at the AVR processor wait only a limited time for new data. If the wait time is exceeded, the boot-loader tries to start a application program in the flash memory.

Chapter 2

The Hardware of the AVR 8-bit micro controllers

2.1 CPU and memory access

You can find any thing at the chip of a AVR 8-bit micro-controller, what is needed to run a digital computer. You find a clock generator, registers, data storage (RAM), program storage (flash), input registers and output registers. The content of registers and data storage is loosed with every restart. The content of the instruction memory (flash) and the often available additional nonvolatile data storage (EEprom) is preserved for long time. The figure 2.1 shows a simplified block diagram of a 8-bit AVR micro-controller.

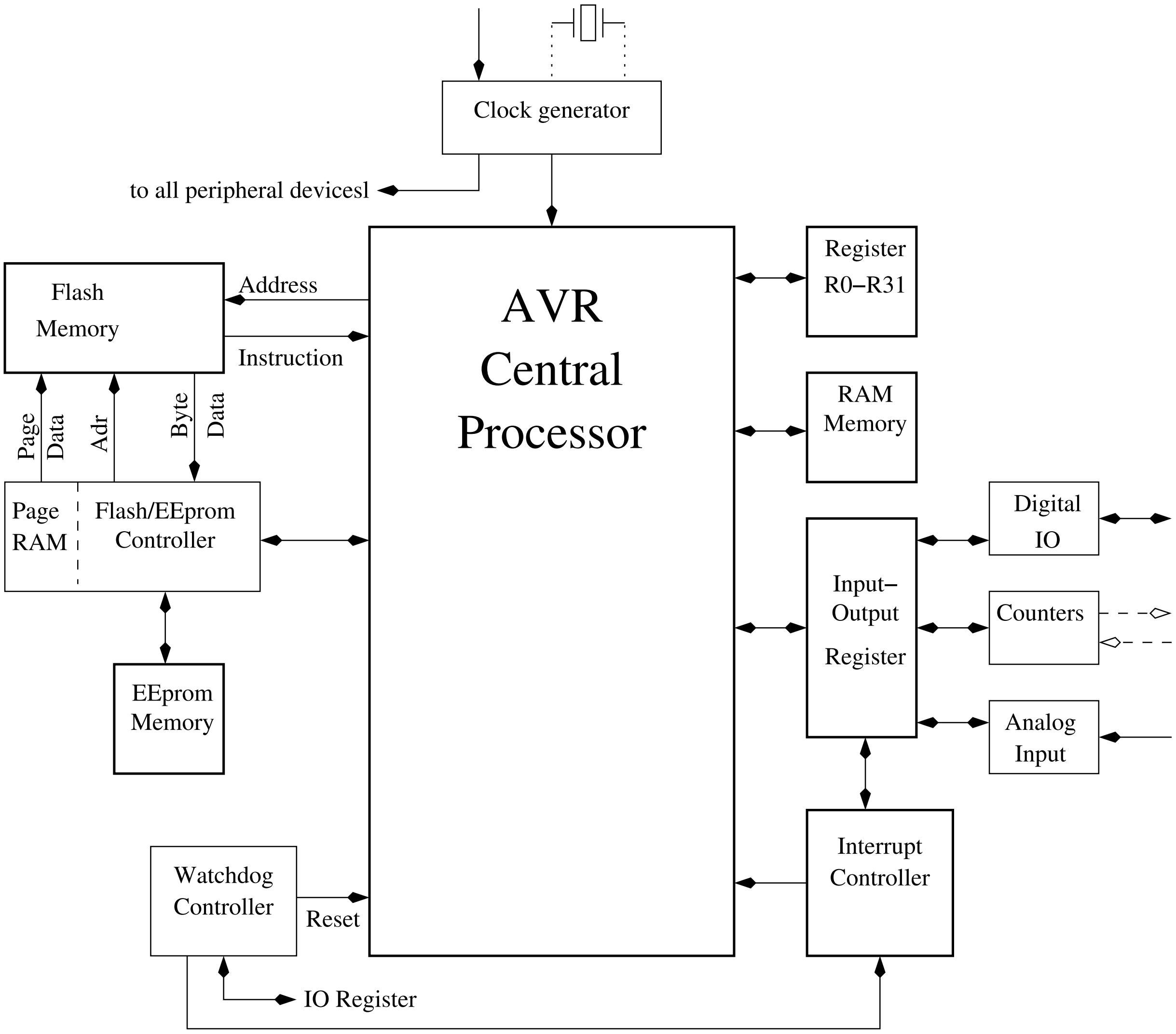

Figure 2.1. Simplified block diagram of a AVR mikrocontroller

You can see at the diagram, that the CPU (Central Processor Unit) can access easy the registers R0-R31 and the RAM memory. Also the access to the input and output resisters is easy possible. But the access to the instruction memory (flash) is only possible with a special controller and more complex.

Only the instruction engine can easy access the flash data for the selected program address. With the Load Immediate (LDI) instruction you can transfer parts of the instruction word to the upper registers (R16-R31). Also with the instructions ADIW, ANDI, CPI, ORI, SBCI, SBIW and SUBI parts of the 16-bit instruction word are processed.

Usually after every instruction the program counter will be increased by one word or two words, depending on the instruction length. A exception to this rule for the normal operation is only caused by the conditional or unconditional jump instructions (RJMP, JMP, IJMP, RCALL, CALL, ICALL, RET, RETI).

Also a reset event or interrupt event can be the reason for a discontinuity of the program counter increase. A Reset will reset the whole processor and the program counter will be set to a previous selected address. A interrupt will set the program counter to a associated address. Normally the start address for the reset event is set to 0. For starting the boot-loader many AVR processors have special configuration bits is fuses to select a different start address.

A random address to the instructing memory content is only possible with the flash-controller. For that access you must tell the controller the requested byte address. Afterward the requested byte can be read with a special instruction.

More complex is a write operation to the flash memory. The write access is only possible for a total page of flash. The flash page must be erased before any write access and you must load load the total page data to the controller buffer storage before you can select the write operation. You can compare this method with printing a page with a stamp. The stamp can be configured with replaceable letters before the next print. So you can print any text. But you need a empty sheet of paper and you must configure the whole text for the page. Only if all is prepared right, you can stamp a page.

Also the access to the nonvolatile data memory (EEprom) is only possible with the special controller. The writing to the EEprom is more simple compared to the flash memory access, but you need the controller access too. You can not use the EEprom and flash memory access together, because some common parts of the controller is used.

2.2 Input and Output function

The CPU can access the external pins with IO-registers. The IO-register are organized with Byte access, so that you can select up to 8 pins with one IO register. The figure 2.2 shows the structure of a port pin circuit.

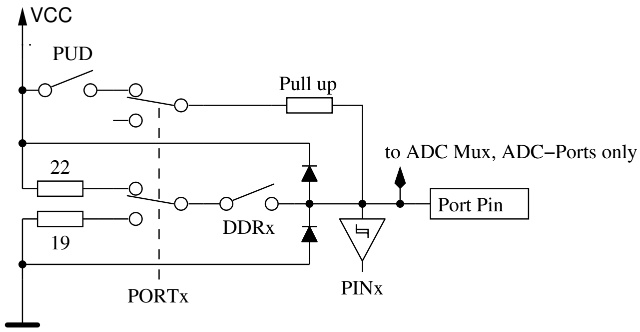

Figure 2.2. Simplified circuit of every AVR Port pin

Every pin is associated with one port bit and can be used as output pin (PORT) or as input pin (PIN). The 8 bits of the Data Direction Register (DDR) are used to select the mode of every port bit. For every pin you will find a associated bit number in three different registers. The DDR register is used to select the direction of the associated bits. The PIN register shows the voltage level of the associated bits. If the voltage is below the half operating voltage, the PIN bit is switched to 0 and above the half operating voltage the PIN bit is switched to 1. If the associated bit in the DDR register is set to 1, the associated bit in the PORT register select the level of the output signal. A 0 select a voltage level near to GND and a 1 select a voltage level near to the operating voltage (VCC). If the output mode for one bit is deselected with a 0 in the DDR register, a setting of the corresponding bit in the PORT register enables a Pull-Up resistor. But if the PUD bit in the MPU configuration register MCUPR is set, all Pull-Up registers are deactivated. All associated registers and bits of a group (8-bit) use always the same code letter. For the second group this letter is a B. The output port of this group has the name PORTB, the input port can be accessed with the name PINB and the register for the data direction has the name DDRB. For the identification of a single bit a number between 0 and 7 will be appended. For example the bit 0 of the input port B would be named PINB0. This terminology is used in the Atmel documentation and is used also with program languages.

2.3 The start of AVR micro-controllers

With the factory set configuration of the AVR micro-controller a first pass over of the minimal operating voltage will cause a reset of the processor. All IO-register are set to predefined values and after waiting some time to stabilize the operating voltage the instruction unit is started with the flash address 0. Normally all pins are set to input mode. Beside this reason for a reset event there exist three other reasons for a reset of the processor. The reason for the reset event is saved in the MCU status register (MCUSR) with four bits.

Name of Flag	Reason for the Reset
PORF	Power-on Reset This Reset is caused by switching on the operating voltage. This reason can not be deactivated.
BORF	Brown-out Reset This reason can only occur, if the function is selected with the BODLEVEL bits of a Fuse and no Brown-out Interrupt is selected.
EXTRF	External Reset is caused by a 0 level at the Reset Pin, if the fuse RSTDISBL is not activated.
WDRF	Watchdog Reset can only be set, if the corresponding Interrupt is not enabled.

Table 2.1. Different Reset reasons in the MCUSR register

By setting the right configuration bits in the fuses of the AVR micro-controller you can select another start address as the usual 0. The figure 2.3 shows the options for a ATmega168. This processor has a total instruction memory (flash) capacity of 16384 Byte. The instruction interpreter of the micro-controller can access a 16-bit parallel instruction code of the flash memory. So the largest program counter is only 8190 for optiboot! It can not be 8192 because the counting begin with 0.

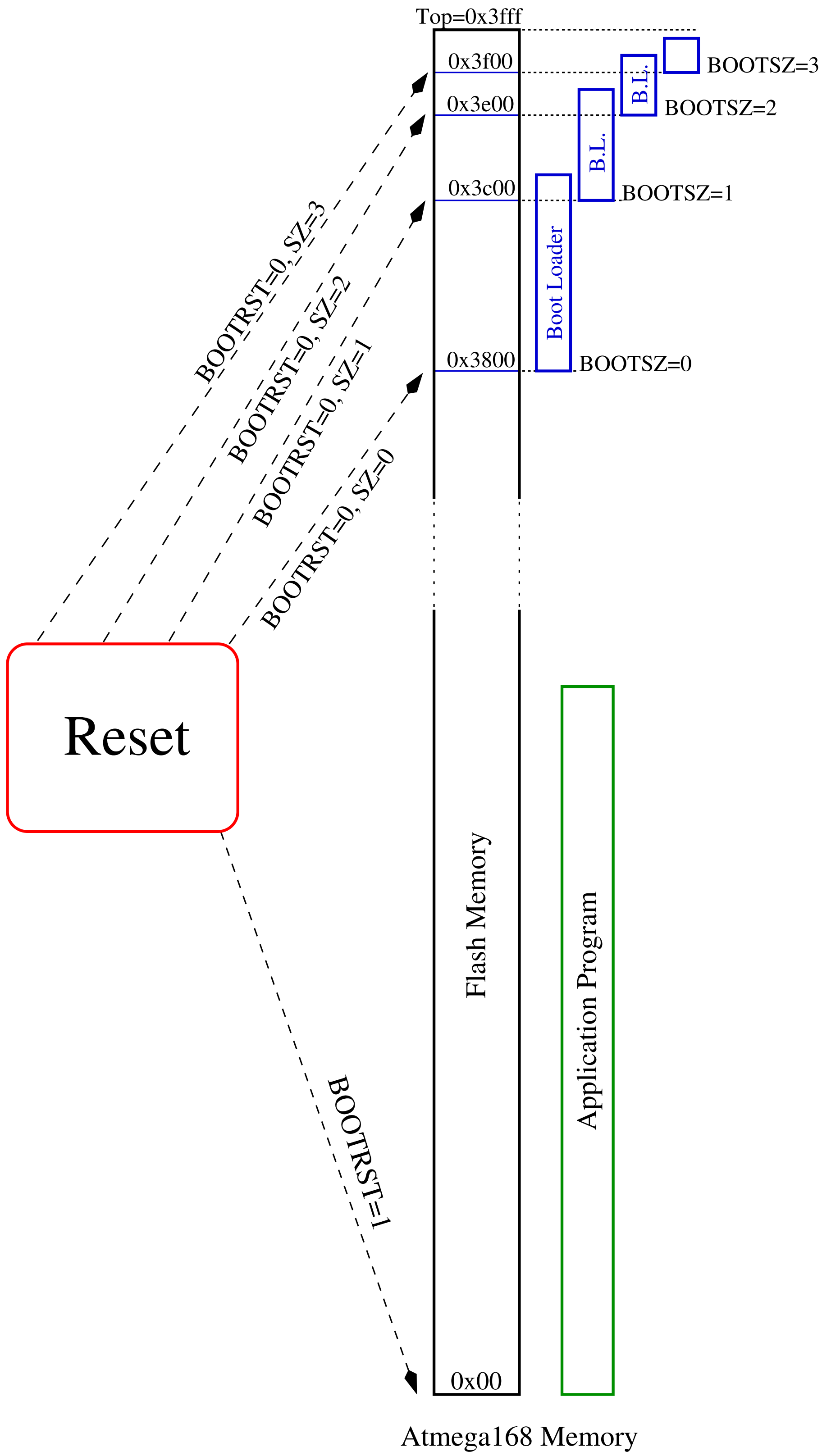

Figure 2.3. The different Start-Options for the ATmega168

The ATmega168 can select a boot-loader size of 256 bytes (BOOTSZ=3), 512 Bytes (BOOTSZ=2), 1024 Bytes (BOOTSZ=1) and 2048 Bytes (BOOTSZ=0). The application program would like to use as many program space as possible, so the boot-loader space should be as low as possible. The boot-loader code is placed at the highest starting address possible. The activating of Lock-bits of the AVR micro-controller can protect the boot-loader area against overwrite. Once activated Lock-bits can only be reset with a total erase of the AVR memories.

For this processor, also for the Mega48 and Mega88, the control bits for the boot-loader start are located in the extended fuse (efuse). This is also true for the BOOTRST fuse, which can be used to switch the start address from 0 to the boot-loader start. For most other AVR micro-controller, also for the ATmega328, the same control bits are located in the high fuse (hfuse). The table 2.2 shows the memory sizes of different AVR-micro-controllers and additionally the options for the Boot-loader area. The boot-loader options are located in the same bit numbers, whatever fuse is selected, the high or extended fuse.

Processor Type	Flash size	EEprom size	RAM size	UART	Boot Config Fuse	BOOTSZ			
						=3	=2	=1	=0
ATmega48	4K	256	512	1	Ext.	256	512	1K	2K
ATtiny84	8K	512	512	-	-	(N x 64)			
ATmega8	8K	512	1K	1	High	256	512	1K	2K
ATmega88	8K	512	1K	1	Ext.	256	512	1K	2K
ATmega8U2	8K	512	512	1	Ext.	512	1K	2K	4K
ATmega16	16K	512	1K	1	High	256	512	1K	2K
ATmega168	16K	512	1K	1	Ext.	256	512	1K	2K
ATmega164	16K	512	1K	1	High	256	512	1K	2K
ATmega16U2	16K	512	512	1	Ext.	512	1K	2K	4K
ATmega16U4	16K	1.25K	512	1	Ext.	512	1K	2K	4K
ATmega32	32K	1K	2K	1	High	512	1K	2K	4K
ATmega328	32K	1K	2K	1	High	512	1K	2K	4K
ATmega324	32K	1K	2K	2	High	512	1K	2K	4K
ATmega32U2	32K	1K	1K	1	Ext.	512	1K	2K	4K
ATmega32U4	32K	2.5K	1K	1	Ext.	512	1K	2K	4K
ATmega644	64K	2K	4K	2	High	1K	2K	4K	8K
ATmega640	64K	4K	8K	4	High	1K	2K	4K	8K
ATmega1284	128K	4K	16K	2	High	1K	2K	4K	8K
ATmega1280	128K	4K	8K	4	High	1K	2K	4K	8K
ATmega2560	262K	4K	8K	4	High	1K	2K	4K	8K

Table 2.2. Boot-loader configurations for different micro-controllers

By the way the boot-loader will work for the first time, even if the BOOTRST fuse bit is not activated. In this case the reset vector is still set to address 0, where usually the application program is located. Because for the first time no application program is loaded, the CPU execute the instructions in the cleared flash memory until it reach the boot-loader code. For the ATmega168 this are less than 8000 instructions, which executes the CPU in less than 1 ms with a 8 MHz clock. But if any application program was loaded, the Reset will start the application program, if the BOOTRST bit is not activated (still set). The boot-loader program can no longer work, because it is not addressed by the reset.

2.4 Writing to the AVR memories

The AVR micro-controllers know three different nonvolatile memories. The most important is the instruction memory, the so called Flash memory.

In addition there are some configuration bits, which can be used to select some features of the processor. This configuration bits are organized in some bytes, the lfuse (low fuse), the hfuse (high fuse), the efuse (extended fuse), the lock byte and the calibration byte. The calibration byte is used to calibrate the frequency of the internal RC oscillator. The lock byte can be used to restrict the access to the memories. Once activated lock bits can not be reset by rewriting the lock byte. The only way to deactivate the lock bits is a complete clear of all memories. Note, that the lock function will be activated by clearing the appropriate bits (write to 0). With the complete clear of all Memories the lock bits will be set to 1 (full access). The layout of the configuration bits to the different fuse bytes differ for the several AVR processor models and should be read in the specific data sheet. You can select the way of clock generation, the delay of start and a monitoring of operating voltage with the fuses.

Most AVR micro-controllers are also equipped with a nonvolatile data memory, the EEprom. This memory type has no special function for the processor. It is just a way for a application program so save data for the next program start.

2.4.1 Parallel programming

All three nonvolatile memory types can be written or read with different technique. Usually the parallel programming method is rare used for writing the nonvolatile memories. Sometimes this method is the only way to reactivate processors, which can not accessed with other methods. For example you can not use the serial programming, if the fuse bit for the Reset pin usage is deactivated (RSTDISBL=0). The voltage at the reset pit is raised to higher voltage level (12V) for this parallel programming method. Therefore this method is also called HV-programming.

2.4.2 serial download with ISP

The normal way to program any non volatile memory is the serial programming. The Atmel documentation call this method also serial download. For that way a SPI (Serial Parallel Interface) interface is used. The SPI interface is build with three signals, MOSI, MISO and SCK. Additionally the Reset pin of the AVR processor must hold to 0V to force this special download mode. Together with two additional power signals GND and VCC (about 2.7 to 5V) this four signals build a ISP (In System Programming) interface, which is often integrated at many boards. The figure 2.4 shows the layout of two usual plugs, which are often integrated at boards with AVR micro-controllers.

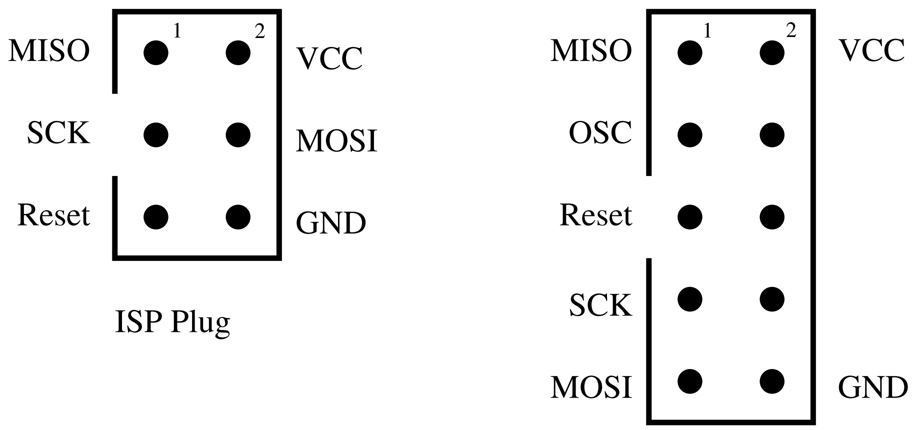

Figure 2.4. Two different types of ISP Plugs

The 10-pin version of the ISP-plug can additionally support a clock signal OSC for feed-in a clock signal to the AVR micro-controller. One of this two plug versions are usually required to program a boot-loader in the flash memory of a AVR micro-controller. The program data for the flash memory are usually created with a PC. To transfer the program data to the AVR micro-controller a ISP programmer is required, which use often a USB interface to the host computer side. But the host computer can also use a serial or parallel interface to connect a ISP programmer. The USB interface has the advantage, that the power (5V or 3.3V) for the micro-controller can be taken easy from the interface. You can choose some types of specific ISP programmers at the electronic market, the manufactor of the controllers offer the Atmel AVR-ISP MK2 programmer for example. But you can also use a Arduino UNO or a similar Arduino with a special program for the connected ISP interface. I use a DIAMAX ALL-AVR, which is equipped with both plug types and has some additional features.

2.4.3 Self programming with serial interface

Because the AVR processor can write flash and EEprom memories with special instructions, you can write a little program to the flash memory with one of the two programming methods, which receive data from a serial interface and can write this data to the flash or EEprom memory. Exactly this is the feature of the boot-loader optiboot. Setting of fuses or lock bytes is often not possible with this method and is not supported by the boot-loader. You must set the fuses and the lock byte with one of the other methods. The STK500 Communication Protocol from Atmel is used for the serial data transfer. Because up-to-date computers often has no more any serial interfaces, a USB - serial converter like the FTDI chip of Future Technology Devices International Ltd is used. A module with this chip is for example a UM232R.

The Chips PL2303 from Prolific Technology Inc. and CP2102 from Silicon Laboratories Inc. satisfy the same purpose. Also a suitable programmed ATmega16U2 can be used for the same function. All of these chips have a selectable Baud-rate and a TTL level for the serial signals. You have to add level converter to get real RS232 signals. But the AVR micro-controllers don't need RS232 signal level. One of these chips is integrated at any Arduino board with USB interface. For a fast answer the serial interface should also connect the DTR signal of the converter with a serial 100nF capacitor to the Reset input pin of the AVR processor. The figure 2.5 shows a typical way of connection.

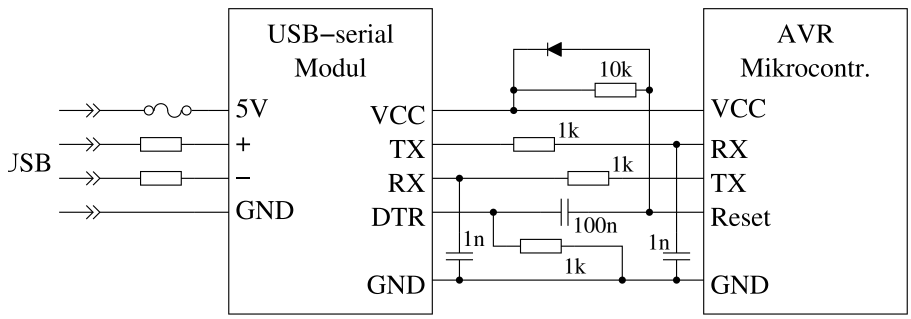

Figure 2.5. Connection of a USB-serial converter to the micro-controller

You should select the right power voltage for the USB - serial converter. Most modules can select a 3.3V or a 5V signal level with a jumper. If you have a Arduino UNO with a ATmega328p at a socket, you can remove the ATmega328p and use the board as USB - serial converter. So you can transfer program data to another AVR processor with a already installed boot-loader. If you frequently use this serial interface, a separate USB - serial interface make sense.

You can select with the Arduino development tool Sketch with the menu entry "Tools - serial Port" a detected serial port. Then you can open a monitor window with the menu entry "Tools - Serial Monitor", which can show you the serial output of your AVR micro-controller at the screen. The Baud rate of the serial interface can be selected at the monitor window. Both 1 nF capacitors at the RX-inputs removes spikes from the serial signals. The test of the software UART program was only successfull with a little capacitor at the RX input of the AVR processor. The probe of a scope was sufficient as "filter" for the spikes. The hardware UART tolerates the spikes without any filter and runs proper with or without the capacitor.

The running Serial Monitor of the Arduino Sketch can disturb the program download with the serial interface, if the same USB - serial module is used for download and the monitor program. But you can insert a additional USB - serial module to the host computer and connect only the RX-Signal to the TX-Port of the AVR micro-controller. This second serial input listen to the output of the AVR without any problems for the serial communication of the program download, if you select this second interface with the Serial Monitor tool.

2.4.4 Diagnostic Tools

At the Linux operating system you can install a tool with the strange name jpnevulator, which can monitor two serial inputs at the same time. Any received data are shown in a hexadecimal format and with the option -a also as ASCII characters. With the option –timing-print the system time of the serial data packets are shown. To prevent any affect to the data communication, you should select two separate USB - serial modules for this monitoring. You should connect the serial input (RX) of one module to the TX signal and the serial input of the other module to the RX signal of the AVR micro-controller. Together with the module for the program download there are three USB - serial modules connected to the PC. Of course all three modules must be set to the same baud rate (stty ... -F /dev/ttyUSB1). Without the speed parameter (stty -F /dev/ttyUSB1) the current settings of the specified device are shown by stty. Some applications like avrdude can set the baud rate itself. The full command line with the start of the protocol can be look like:

```
jpnevulator -a --timing-print --read --tty "/dev/ttyUSB1" --tty "/dev/ttyUSB2"
2016-05-29 11:05:06.589614: /dev/ttyUSB0
30 20                                           0
2016-05-29 11:05:06.589722: /dev/ttyUSB1
14 10                                           ..
2016-05-29 11:05:06.593593: /dev/ttyUSB0
41 81 20                                        A.
2016-05-29 11:05:06.594581: /dev/ttyUSB1
14 74 10                                        .t.
2016-05-29 11:05:06.597583: /dev/ttyUSB0
41 82 20                                        A.
2016-05-29 11:05:06.598574: /dev/ttyUSB1
14 02 10                                        ...
2016-05-29 11:05:06.601586: /dev/ttyUSB0
42 86 00 00 01 01 01 01 03 FF FF FF FF 00 80 04 B...............
00 00 00 80 00 20                               .....
2016-05-29 11:05:06.603608: /dev/ttyUSB1
14 10                                           ..
2016-05-29 11:05:06.605639: /dev/ttyUSB0
45 05 04 D7 C2 00 20                            E.....
2016-05-29 11:05:06.606576: /dev/ttyUSB1
14 10                                           ..
...
```

The avrdude program also has the option of logging activities. This function is switched on with the -v option. Multiple v's make the log more verbose. With -vvvv almost every detail is logged.

Chapter 3

The optiboot boot-loader for AVR Micro-controllers

The optiboot Boot-loader has been created with C language by Peter Knight and Bill Westfield. I have used the version 6.2 as base for the here described revised Assembler version. I would like to underline, that I did not reinvent the optiboot boot-loader. I have just done some optimizing. Many adaptions to several target processors and special board level systems are present with the version 6.2. The program use parts of the STK500 communication protocol, which is released with AVR061 [21] from Atmel.

3.1 Changes and enhancements to the version 6.2

Basically I have translated the total program in the assembler language and have rebuild the process for generating the .hex file with a bash shell script, so that the program length will be processed automatically to select the start address of the boot-loader and set the right fuses for that program length. The selected solution generates some variables during some interim steps, which are required to solve the steps to select the right start address and the right fuses. The start address of the boot-loader for any target processor depends on the present flash size, the flash requirement of the boot-loader code and the tile size, which is supported by the target processor for bootlace. The tile size means the smallest boot-loader size, which can be supported by the selected target processor.

For processors like the ATtiny84, which don't support the boot-loader start function, the page size of the flash memory is used for this calculation. For the ATtiny84 this are 64 Bytes. Therefore the start address of the boot-loader is always located at the begin of a flash page.

For all other supported target processors the boot-loader area can be selected with the fuse bits BOOTSZ1 and BOOTSZ0 (each with the values 0 and 1). If you put together the both bits, you get a coded boot-loader size with values between 0 and 3. Always the value of 3 select the smallest possible boot-loader area. A value of 2 select a double size, the value 1 the quadruple size and the value 0 select a size of eight times the smallest size. The table 2.2 at page 11 shows a overview for the several target processors. The picture 3.1 shows most of the test boards and processores, which are used for the optiboot tests.

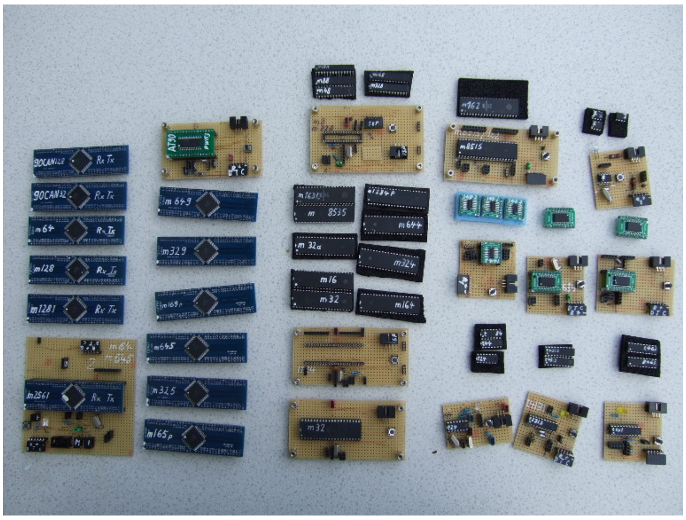

Figure 3.1. Some Boards and AVRs for the optiboot tests

3.2 Features of the optiboot Assembler version

To avoid many pitfalls when creating an executable boot loader, the creation of the boot loader file was largely automated. In addition, settings are also checked and the generation is canceled with a corresponding error message, if, for example, the selected operating frequency (AVR_FREQ) does not match the fuse setting (CKSEL, CKDIV8) for the clock. Receiving program data for optiboot is currently only possible via a serial interface.

Below are some of the features listed for the optiboot bootloader:

- Supports a large number of AVRs, allmost all have been tested for functionality.
- Support the data load for the EEprom memory.
- Fits in 512 Byte Flash in most cases.
- The required fuse setting is changed automatically according to the actual program size.
- The start address, which depends on the program size and the AVR model, is automatically calculated and displayed on the screen.
- All serial interfaces available with the respective AVR processor can be chosen freely. The first UART is always the default.
- Can emulate the serial interface via software. With that options you can also use ATtiny processores without a UART. In addition, the TX and RX pins can be freely selected with SOFT_UART.
- With the SOFT_UART feature you can select a serial communication with only on AVR pin (RXD and TXD are same). The Hardware UART can not use this feature, because the RXD and TXD pins are fixed to the manufactor selected pins.
- The Baud rate can be adjusted automatically by the optiboot program. For that feature the Baud time is measured by analysing the first incomming serial data byte. Different methods of this baud time measurement can be choosed. Normally you must deselect the LED Flash function to fit the optiboot program data within 512 byte Flash.

- Can be used with AVRs without the bootloader support.
  This VIRTUAL_BOOT_PARTITION feature can of course also be used for AVRs, which has the bootloader support. The optiboot program gets bigger, but can then start at every flash memory page.

- By default, a connected LED flashes three times (LED_START_FLASHES=3). When serial data arrives, the blinking is stopped immediately.

- Instead of the LED flashing at startup, you can also let the LED light up when waiting for serial data (LED_DATA_FLASH=1). By setting LED_DATA_FLASH=4 you call select a permanent lit LED. The watchdog reset will turn the LED off in this case.

- The length of the generated optiboot depends only on the selected AVR processor and the selected options. The version of the installed avr-gcc has no effect, if you use the assembler source. So a later check of the installed bootloader can be easier done, even with another PC.

- Supports the frequency adjustment of the build in AVR RC-oscillator. For some problematic processors this feature enables the use of a bootloader with a fix baud rate in this operation mode.

- Installing the optiboot bootloader at a respective AVR can be done with the program **avrdude** and a connected ISP-programmer. To initiate this task only a additional ISP=1 is required at the **make** call. With a ISP=2 at the **make** call you can operate a verify run of **avrdude**. Additionally a read out of the total flash memory of the AVR is possible with ISP=3. A ISP=4 will read the complete EEprom data from your AVR processor.

The next features affects only the build process of the boot bootloader:

- You can also choose the adapted C-sources to generate the optiboot bootloader. Most of the features are possible with this selection also. Of course the generated optiboot program will be bigger! Please note, that you have to select the EEprom support with the C-source.

- The amount of the screen output iduring the generating of the optiboot program can be adjusted with the system variable VerboseLev. The value for VerboseLev can be between 1 and 4, the normal value is 2.

- The screen output can also be colored with the system variable WITH_COLORS=1. If the variable is set to WITH_COLORS=2, you get a unformatted text on screen.

- The generationg of the optiboot program is controlled by the bash script build_hex.sh. This script will read the required AVR data from the text file avr_params.def . In the same avr_params.def the script will find the default setting for the fuses and other parameters like operating frequency and baud rate for every supported AVR.

- For every supported AVR processor the build_hex.sh script can find a matching file in the avr_pins directory, where the pin layout of this processor is fixed. Because some processors have the same pin layout, some of the AVRs are grouped together. In the matching file the default pin for the LED can be found. For processors without UART the default pins for RXD and TXD for the software UART solution is specified.

- The generating of the optiboot program runs at a Linux System with installed avr packages. You can also use a Windows10 system, when you install the Arduino package und additional one or more packages, which install the required commands like bash, bc, echo and other tools. Tested is the generation at a Windows10 laptop with installed Arduino and Cygwin64 package.

- The selected parameters for the generating of the optiboot bootloader are logged at the end of the .lst file and also in a separate .log file. You can see the report with the command cat <filename>.log .

3.3 Automatic size adaption in the optiboot Makefile

The boot-loader start address and the required boot-loader size will be adapted automatically with the bash script build_hex.sh, wich is called by the Makefile. For the calculation some interim variables are created, which is only possible together with some Linux tools:

bash a powerfull command interpreter for running the script files.

bc a simple calculator, which can operate with input and output- values in decimal and hexadecimal values.

cat put the file content to the standard output.

cut can select part of lines of a text.

echo shows the specified text at standard output.

grep shows only lines of a text file which contain the specified string.

tr can replace or erase characters.

Until now the functions of the bash script is only tested with a Linux System and with Windows10 together with installed Cygwin64 and Arduino packages. The new optiboot system does not create .dat interim Files.

Here are the names of the used script files:

build_hex.sh take the settings from the Makefile and produce a matching optiboot hex-file in Intel format. The build_hex.sh scripts call some helping scripts as get_avr_params.sh, avr_family.sh, show_led_pin.sh, show_rx_pin.sh and show_tx_pin.sh . If the variable ISP is set, also the script program_target is called.

program_target.sh is called from build_hex.sh, to check and correct the hfuse and efuse settings. If the configuration result to another number of used bootpages, the BOOTSZ bits of the ATmega must be changed. The bits are depending on the AVR model in the hfuse or efuse byte, which does not **make** the correction easier. After the required corrections the **avrdude** program is called with the script only_avrdude.sh .

only_avrdude.sh makes only one avrdude call with the params given by the variable DUDE_PARAMS . For check of the parameters the call is reported at the terminal. If the **avrdude** call returns with an error, the script give some common hints for error search. At a Linux system a additional search of probably matching serial interfaces is done.

find_serials.sh will search possible interface named for USB-serial converters at a Linux system. For this task are matching integrated chips available, which are named by the Linux driver beginning with /dev/ttyUSB . There are also some software emulations available at special mikrocontrollers, which has a integrated USB interface. You can access these Interfaces with names beginning with /tty/ACM . For each found Interface the script will check, if a access is possible for you. For that you must be a member in the group, which is allowed to access the interface (in most cases dialout). This script will only called from only_avrdude, if avrdude returns with error. You can call this script also by command bash ./find_serials.sh For a Windows system this scipt is useless, use the device manager of Windows instead to find serial interfaces with name COMx .

get_avr_params.sh find for the MCU_TARGET specified AVR processor all required parameters like flash size, the size of a flash page and the size of a bootloader page. Additionally some default values like operating frequency, baud rate and fuse settings are set, if not specified by user. The source file for these settings is avr_params.def.

avr_family.sh build a group name for some AVR processors with identical pin layout. Usually the group name is identical to the member with the biggest flash (atmega328 for atmega16 or atmega88).

show_led_pin.sh find the LED pin, which is the default for this processor or group in the file avr_pins/<group>.pins A message is shown to explain this selection. If the user has selected another pin for the LED, this pin is reported.

show_rx_pin.sh report the selected RX pin for incomming serial data. If the user did not specify a special selection for SOFT_UART, the pin is taken from the avr_pins/<group>.pins file.

show_tx_pin.sh report the selected TX pin for outgoing serial data. If the user did not specify a special selection for SOFT_UART, the pin is taken from the avr_pins/<group>.pins file.

baudcheck.tmp.sh is created during the run of the build_hex.sh script with a C-Preprocessor call from the baudcheck.S file. It is executed in the same run to give you information about the selected baud rate state.

3.4 target selection for the optiboot Makefile

There can exist different configurations for the same processor type. The table 3.1 shows some basic configuration for several target processors. You can select some parameters also with the **make** call or by setting a environment variable of the shell. These settings will allways replace the default selections. There are two main variable names, which are given to the bash shell script build_hex.sh , TARGET and MCU_TARGET. The MCU_TARGET must specify a valid AVR processor name. The TARGET variable specifies a free selectable name, but is usually same as the MCU_TARGET of specify a board name.

Name	MCU	AVR_ FREQ	total Flash size	Flash page size	BP_ LEN	LFUSE	HFUSE	EFUSE
attiny84	t84	16M?	8K	64	(64)	62	DF	FE
atmega8	m8	16M	8K	64	256	BF	CC	-
atmega88	m88	16M	8K	64	256	FF	DD	04
atmega16	m16	16M	16K	128	256	FF	9C	-
atmega168	m168	16M	16K	128	256	FC	DD	04
atmega168p	m168p	16M	16K	128	256	FC	DD	04
atmega32	m32	16M	16K	128	256	BF	CE	-
atmega328	m328	16M	32K	128	512	FF	DE	05
atmega328p	m328p	16M	32K	128	512	FF	DE	05
atmega644p	m644p	16M	64K	256	512	F7	DE	05
atmega1284p	m1284p	16M	128K	256	512	F7	DE	05
atmega1280	m1280	16M	128K	256	1K	FF	DE	05

Table 3.1. some Processor targets for optiboot Makefile

All size values are shown in byte units, the values for fuses are shown with hexadecimal values. The frequency values must be specified in Hz units, 16M is the same as 16000000 Hz. The standard baud rate of the serial interface is 115200 in most cases for the $16MHz$ operation.

Additional to the universal processor configurations you can also select configurations for special boards or operational environment. The table 3.2 shows the different adjustments.

Name	MCU	AVR_ FREQ	BP_ LEN	L FUSE	H FUSE	E FUSE	BAUD_ RATE	LED	SOFT_ UART
luminet	t84	1M	64v	F7	DD	04	9600	0x	-
virboot8	m8	16M	64v						
diecimila	m168	(16M)		F7	DD	04		3x	-
lilypad	m168	8M		E2	DD	04	-	3x	-
pro8	m168	16M		F7	C6	04	-	3x	-
pro16	m168	16M		F7	DD	04	-	3x	-
pro20	m168	16M		F7	DC	04	-	3x	-
atmega168p_lp	m168	16M		FF	DD	04	-		-
xplained168pb	m168	(16M)					57600		
virboot328	m328p	16M	128v						-
atmega328_pro8	m328p	8M		FF	DE	05	-	3x	-
xplained328pb	m168	(16M)					57600		
xplained328p	m168	(16M)					57600		
wildfire	m1284p	16M					-	3xB5	
mega1280	m1280	16M		FF	DE	05	-		-

Table 3.2. configured targets for the optiboot Makefile

3.5 The Options for the optiboot Makefile

With the options you can select the feature of the optiboot boot-loader. For example you can select with the option SOFT_UART, that a software solution is used for the serial communication.

Without this option a integrated hardware UART is used for serial communication. The pin TX (Transmit) is used for serial output and the pin RX (Receive) is used for serial input. If more than one UART is present at the target processor, the first interface with the number 0 is used. But you can also select every other present UART by specify the number with the option UART (UART=1 for the second present UART). For the hardware UART interfaces the pins for transmit and receive are fixed to the specific pins. For the serial communication with software you can select any pins, which are able to do digital input and output. More details for the available options you can find in the tables 3.3, 3.4 and 3.5

Name of the Option	Example	Function
F_CPU	F_CPU=8000000	Tell the program the clock frequency of the processor. The value is specified in Hz units (cycles per second). The example specifies a frequency of 8 MHz.
BAUD_RATE	BAUD_ RATE=9600	Specifies the baud-rate for the serial communication. Always 8 data bits without parity is used. Values below 100 will select a measurement and adaptation of the baudrate with different technique.
SOFT_UART	SOFT_UART=1	Select a software solution for the serial communication.
UART_RX	UART_RX=D0	Specifies the port and bit number used for the serial input. The example select bit 0 of PIND as serial input. You can use this option only with the software UART.
UART_TX	UART_TX=D1	Specifies the port and bit number used for the serial output. The example select bit 1 of PORTD as serial output. You can use this option only with the software UART.
INVERSE_UART	INVERSE_UART=1	Inverse the logic level for RX and TX data. This option can only used with software UART.
UART	UART=1	Select a hardware UART used for the serial communication You can only select a UART if more than one is present. This option will also change the default setting for SOFT_UA
LED_START_ FLASHES	LED_START_ FLASHES=3	Select a repetition count of flashing cycles for the control LED. A count of 1 or -1 will only flash once without a loop. Negative values will switch off a additional check of the RX pin of the serial interface. The loop is interrupted immediately, if any incoming serial data is detected. Please note, that the start of application program is delayed with the blink cycles.
LED	LED=B3	Select a port and bit number for the control LED. The example would select the bit number 3 of the port B for the LED connection. With the option LED_START_FLASHES this LED will flash the specified count before the communication start.
LED_DATA_ FLASH	LED_DATA_ FLASH=1	The control LED will glow during waiting for serial input data, if the value is 1. If you set the variable to 4, the LED ist switch on once at the beginning of the bootloader So you can also see, that the bootloader has started, but this setting to 4 with a zero LED_START_FLASHES will save a lot of flash memory.

Table 3.3. Important options for the optiboot Makefile

When operating with the internal RC generator, it is quite possible that a serial data transfer not immediately succeed. This is principle independent of whether the hardware UART interface or a software solution (SOFT_UART) is used. Without additional measurement you can only try with estimated OSCCAL_CORR values. Probably the data sheet of the AVR processor can help a little bit. Here is described at which operating voltage and at what temperature the RC oscillator was calibrated. In addition here is also described the gradient of frequency change with operating voltage, temperature and OSCCAL modification.

More options are listed in tables 3.4 and 3.5. Some of these options are only interesting for software checks, the frequency adjusting of the RC-generator and for processors without the boot-loader support.

Name of the Option	Example	Function
TIMEOUT_MS	TIMEOUT_MS=2000	This option specifies a time limit in ms units for receiving boot data. After this time without data the boot process is aborted and the processor tries to start the user program. Possible values for TIMEOUT_MS are 500, 1000, 2000, 4000 and 8000. The efffective value can be limited to 2s because of processor limits for the watchdog. If no TIMEOUT_MS is specified, the time limit is set to 1 second.
SUPPORT_ EEPROM	SUPPORT_ EEPROM=1	Select the EEprom read and write function for the boot-loader. If the assembly language is selected as source, the EEprom support is enabled without this option, but can be switched off by setting the SUPPORT_EEPROM Option to 0. For the C-source the function must be switched on (default = off).
C_SOURCE	C_SOURCE=1	Select the C language as source instead of the assembly language (option 0 = assembly). The assembly version requires less program space.
LFUSE	LFUSE=EF	Specifies a desired value for the AVR Low Fuse. Only two Hex characters are allowed.
HFUSE	HFUSE=D9	Specifies a desired value for the AVR High Fuse. Only two Hex characters are allowed.
EFUSE	EFUSE=FC	Specifies a desired value for the AVR Extended Fuse. Only two Hex characters are allowed.

Table 3.4. More options for the optiboot Makefile

Name of the Option	Example	Function
BIGBOOT	BIGBOOT=512	Select additional space usage for the compiled program. This is used only for tests of the automatic adaption to the program size.
VIRTUAL_ BOOT_ PARTITION	VIRTUAL_ BOOT_ PARTITION	Changes the interrupt vector table of a user program, that the boot-loader is called with a Reset. For the start of the user program another interrupt vector is used.
save_vect_ num	save_vect_ num=4	Choose a interrupt vector number for the VIRTUAL_BOOT_PARTITION method.
OSCCAL_ CORR	OSCCAL_ CORR=5	With the option OSCCAL_CORR you can adjust the internal 8 MHz RC-generator of the AVR. Is effectless with crystal-operation or external clock! The correction value will be subtracted from the actual OSCCAL byte. The frequency will be lower with a positive correction value. Because the produced Baud rate is directly derived from the processor clock, a correct selected processor clock is important for a successful serial communication. The value must be between -15 and +15.
NO_EARLY_ PAGE_ERASE	NO_EARLY_ PAGE_ERASE=1	Prevents the erasing of the flash page before the data is received via the serial interface. Programming the flash with this option is about 30% slower, because the deletion otherwise runs parallel to the data reception. But the time loss is not so significant, because the also carried out data verification takes about the same time and is uneffected. This saves about 14 bytes of space on the bootloader side of the optiboot, which in practice can also mean that the space requirement can be halved due to the AVR technology.

Table 3.5. More options for the optiboot Makefile

3.6 Usage of optiboot without a boot-loader area

For processors without a special boot-loader area in the flash memory, for example the ATtiny84, a solution is selectable to use the optiboot anyway. This function can be selected with the VIRTUAL_BOOT_PARTITION option. To start the boot-loader first with every Reset of the processor, the interrupt vector table of the application program is changed. At the reset vector location a jump to the optiboot program is registered. The original start address of the application program will be moved to another interrupt vector the "replacement reset vector". This interrupt vector should not be used by the application program. If the boot-loader does not receive any data from the serial interface within a appropriate time, the boot-loader jump to the location of the replacement reset vector and start the application program. The figure 3.2 should illustrate these changes.

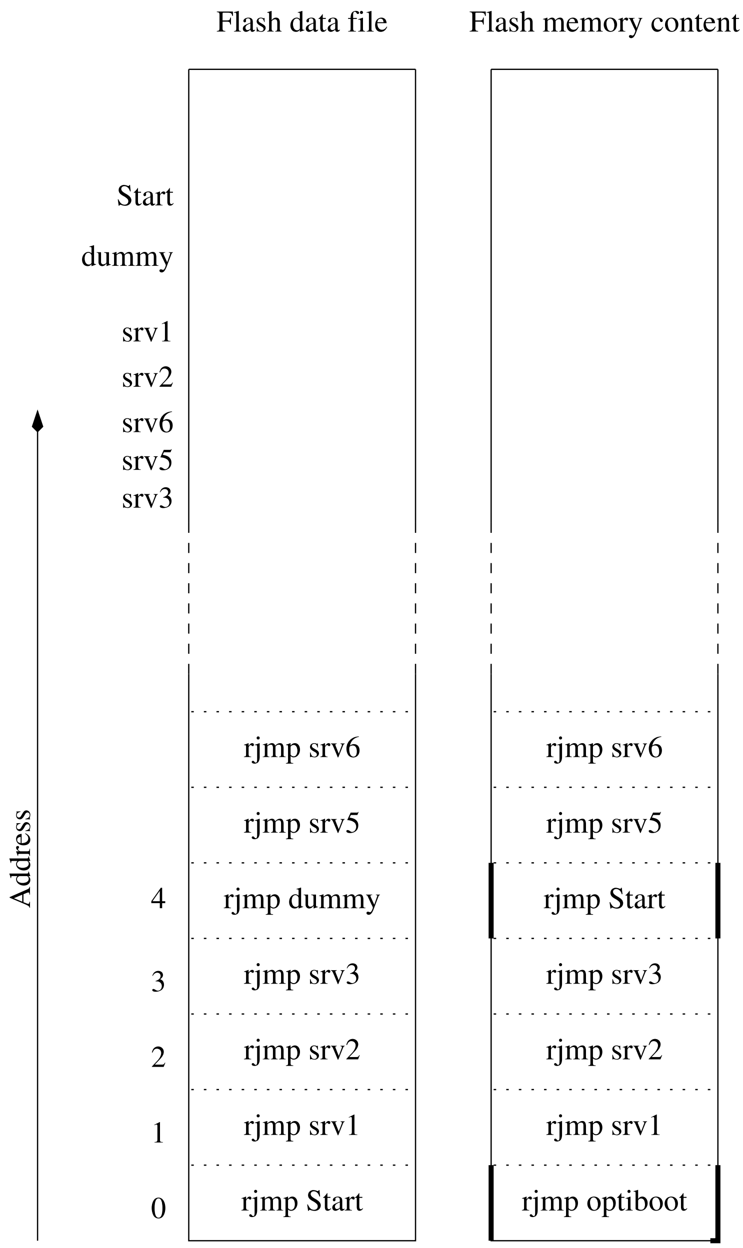

Figure 3.2. Changes of program data by optiboot

At the left side the content of the program data file (.hex) is shown. Just to the right the content of the flash memory is shown, as it is modified by the optiboot boot-loader. At two interrupt vector addresses the content is changed. At the reset vector address 0 the jump is modified to select the optiboot start address as jump target. At the ”replacement vector address” 4 the original jump target address of the application program’s reset vector is used as new jump target address of this vector. One of the problems with this modification is, that usually the program data is verified by the host after write is finished. To provide any error message by verify the program data, the optiboot return the program data without its own modification, not the real content of the interrupt vector table. The jump target address of the reset vector can be reconstructed with the content of the replacement vector address. But the original content of the replacement vector would be lost because there is no place to save the original content in the flash memory. Therefore optiboot use the last two places of the EEprom memory to save this original content of the replacement vector. So the verify of the program data is possible without errors, as long as the application program do not use one of the last two EEprom locations. Even if the application program use one of the last two EEprom locations, the boot-loader will be unaffected. Only the program verify by the host is no longer possible without a error message. An error message will occur at the location of the replacement interrupt vector.

For processors with more than 8 kByte flash memory two instruction words are used for every interrupt vector. Normally every of this double words hold one JMP instruction with the proper jump target address. The optiboot program can respect these JMP vector table too. But if you use the linker avr-ld with the option –relax, all JMP instructions are replaced by a RJMP, if this is possible for the target address. This replacement of JMP instruction in the vector table by RJMP is not respected by the optiboot program. The optiboot program assume, that all interrupt vector numbers of a processor with more than 8 kByte flash hold a JMP instruction. For that reason a optiboot program with the VIRTUAL_BOOT_Partition option will not work with a application program, which is linked with the –relax option. The same problem exist, if the application program itself use a RJMP instruction in one of the two critical interrupt vector positions.

Further you should notice, that you don't activate the BOOTRST fuse together with with the usage of the VIRTUAL_BOOT_PARTITION option. The reason is, that the start address of the boot-loader can be located to other addresses with the VIRTUAL_BOOT_PARTITION option than without this option. With the VIRTUAL_BOOT_PARTITION the start address can be placed to every begin of a flash page. For the normal boot-loader support of the AVR the start address can only respect the single, double, quadruple or octuple size of a minimum boot-loader size as shown in figure 2.3 at page 10.

3.7 Capabilities of the serial interface with the applied software

Das Programm für die Erzeugung und Verarbeitung der elektrischen Signale ist in AVR-Assembler geschrieben. The way of operation is taked over from the Application Note AVR305 of the ATmel Corporation. However here are some special features build in. For example it is respected, that we can not use the special bit-instructions SBI, CBI and SBIC for any port address. You can use this instructions only up to the address 31 (0x1f). For some higher port addresses up to 63 (0x3f) you can use special input (IN) and output (OUT) instructions. If the addresses of the port are higher than this value, you must use the instructions LDS and STS to access this ports. This instructions need 2 processor clocks for execution and use the double flash memory (2 words or 4 byte) of the other instructions. The changed cycle number of one loop pass without any additional delay will be automatically determined by the program. This number of cycles is then taken into account for the calculation of the delay loop, to achieve a correct time for the transmission of a bit. The diagrams 3.3 and 3.4 should be used to explain the work of the C-proprocessor.

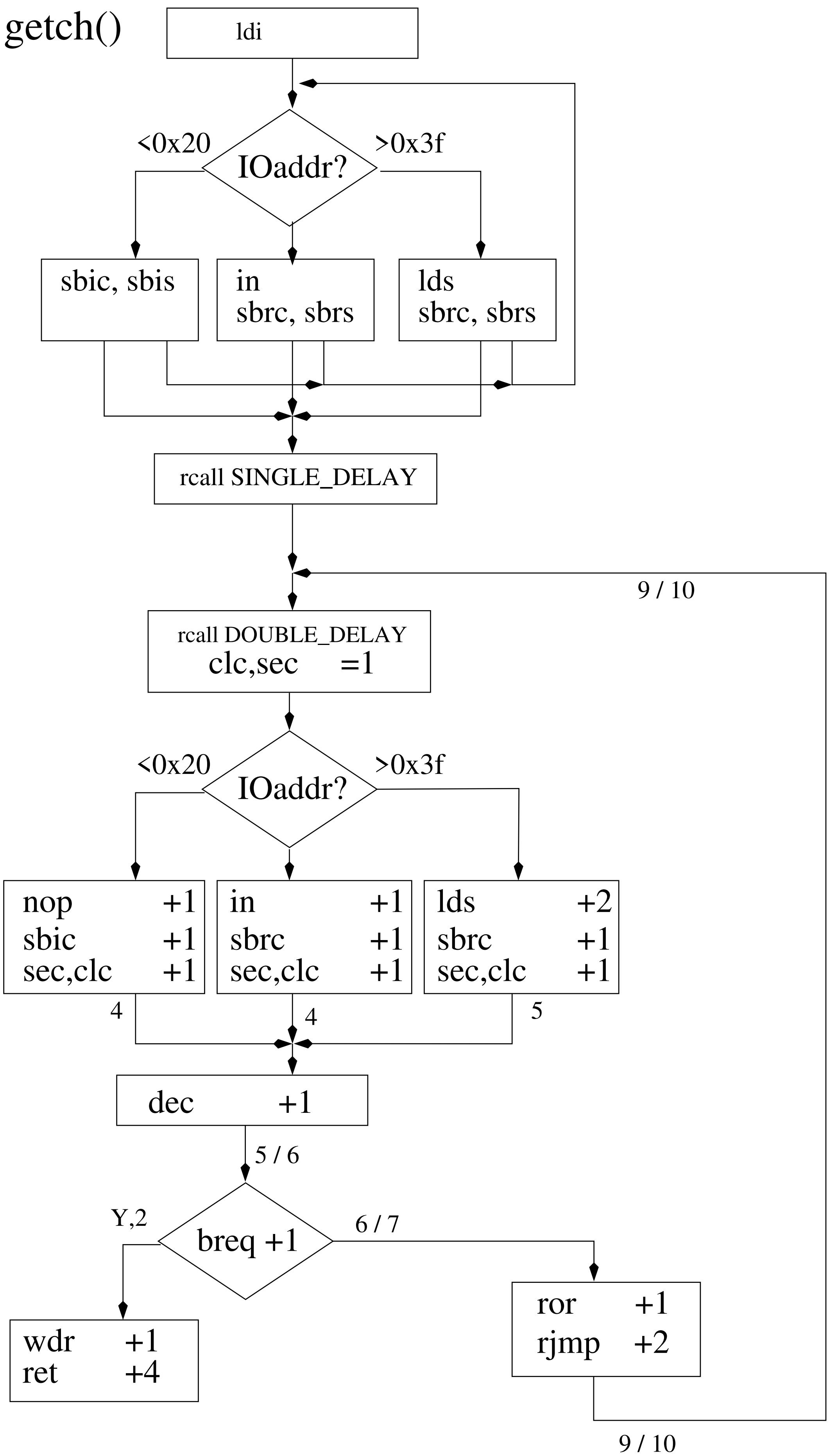

Figure 3.3. Possible variants of the getch function

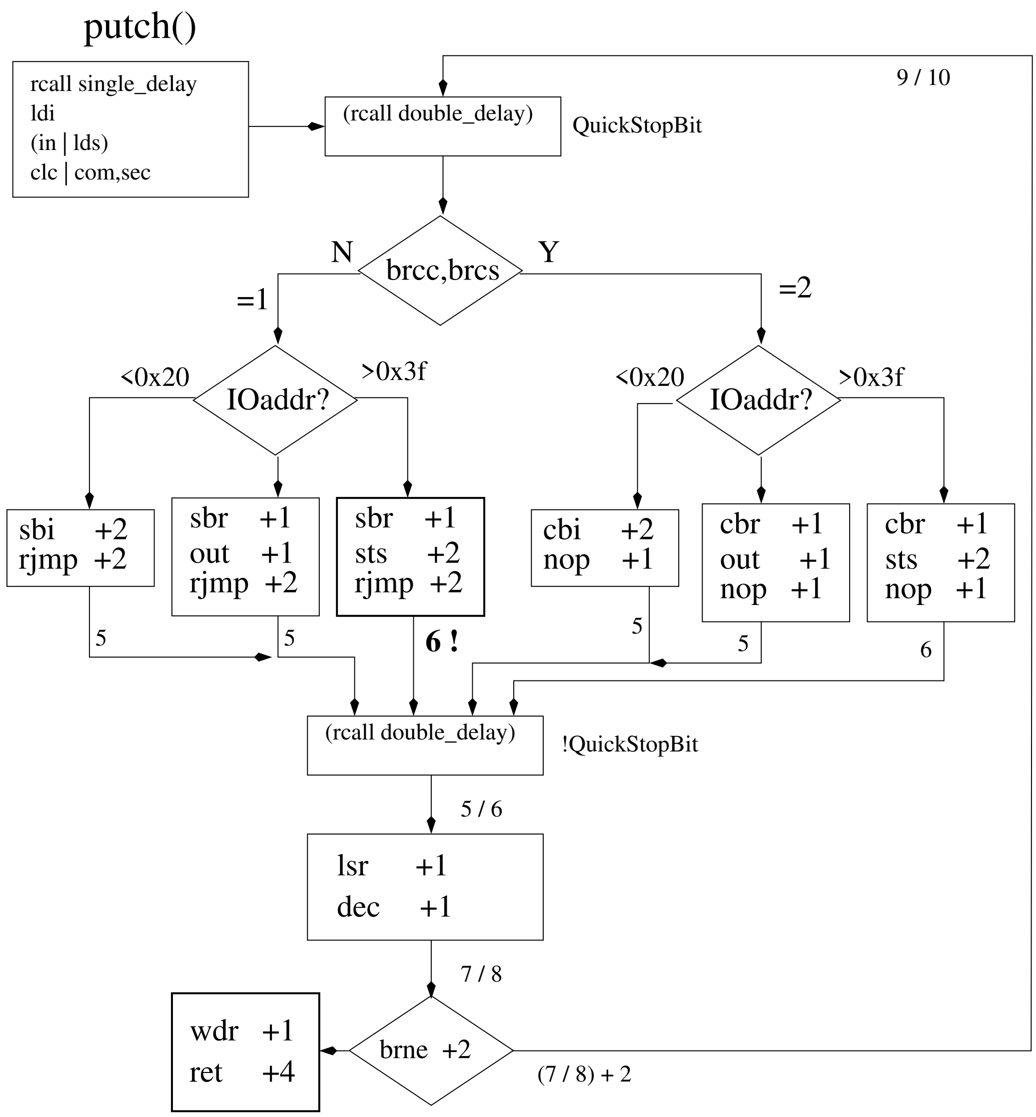

Figure 3.4. Possible variants of the putch function

Both loops are generated so that one cycle with the same conditions of INVERSE_UART option and address range of the used ports use the same tic count. So both functions can use the same function for generating the necessary delay.

3.7.1 Computing of the delay time

For the read-in function getch() is the half baud time needed. From the detection of the start bit a total time of 1.5 times of the baud-time is delayed to read the first data bit. Therefore the base loop is designed for the half duration of a bit transmission. This base loop is called twice in a special way, so that exactly twice the time is get. Should one clock have been lost by the building of half the time and subsequent doubling, this clock is compensated by adding a additional NOP command, so that the total time is exactly correct for the transmission of one bit. Because of the limited time resolution of the counter loop of 3 tics, a remainder of up to 2 clock tics is compensated by adding a additional instruction with the right tic count (RJMP or NOP). Of course, all this happens automatically, because only the loop time of the input function (getch) and the output function (putch), the clock frequency of the processor and the desired baud rate must be known. All other parameters like the number of clocks for a subroutine call (RCALL, RET) are known for the traget processor. A disadvantage of the base delay loop is the limited number of clocks for the delay. With the used 8-bit counter only a maximum of 256*3 clocks is possible for the loop delay. In addition there is the subroutine call of 7 tics, resulting in a half delay time of 775 tics. This value must be doubled (delay for a whole bit time) and the loop-time of the input or output function must be added to get the total time possible. Thus the highest achievable delay time is 1559 clocks. With a $16MHz$ clock frequency you can not get a baud rate of 9600 Baud $104.17\mu s$, because the limit is only $97.4\mu s$. If a 16-bit counter is used by the delay loop, you would get an even worse resolution of time because one loop cycle would take more time. In addition, the 16-bit loop probably need to use the carry bit. The 8-bit counter don't use the carry bit. This problem is solved by gradually doubling the delay times by doubling the delay loop call. The C-prepocessor checks, if the initial value of the loop counter would match the 8-bit limit (255) at the selected clock frequency and the desired baud rate. In this cace the calculation is repeated for a double call of the base delay loop. If the resulting initial value for the counting loop is still too high, the double call is doubled again. Currently this procedure is repeated up to a factor 64 of the base time with the 8-bit counter. At a clock frequency of $16MHz$ or $20MHz$ it is now possible to set the serial interface to 300 baud. For every doubling of the delay time, a additional instruction (2 bytes) is required. With the maximum count of doubling 6 additional instructions (12 byte) are required in the flash memory. There is no attempt to add additional instructions to compensate the missing tics caused by the prescaler for saving flash memory. The baud time error remains clearly below 1%, because the scaler is used only if necessary. So the base loop has at least 127 passes with approximately 381 tics, The double delay time makes no error because of the "NOP" compensation. Thus the error stays below 1:762 0.13%. The higher baud rates tend to get higher error for the transmission time, because the time frame of the CPU clock does not match to the desired baud time. The hardware UART has then the same problem, if it use the same CPU-clock. If you wish to generate a baud rate of 230400 with a CPU clock rate of $16MHz$, your UART can operate with $2MHz$ in the best case. So you can use either 8 tics with a baud time of $4\mu s$ or 9 tics with a baud time of $4.5\mu s$. For the first case your baud time is 7.84% too short, for the second case the baud time is 3.68% too long.

3.7.2 Using more than one serial interface

The assembler file soft_uart.S is designed to be included by a different file which hold a normal assembler source for the AVR family. For the optiboot application this is done by the optiboot.S file. The included file soft_uart.S use many instructions of the GNU C-preprocessor and includes another file uart_delay.S for producing a delay loop for the desired baud rate. Because this include can be repeated with other parameters, you can produce up to 4 different delay loops for 4 different baud rates. This feature use the file soft_uart.S for generatung a getch and a putch function. For both functions the file uart_delay.S is included. But for the second call is usually no new code generated because the calling parameters are the same. Only if the parameters differ, a new delay loop would be generated. Please note, that the callings for the delay loop are named with C-preprocessor macros. This macros are set to a matching delay loop named DOUBLE_DELAY_CALL and SINGLE_DELAY_CALL, if you include the uart_delay.S file before the code of the serial input or output function.

Three constants must be set before any include of uart_delay.S, named F_CPU, BAUD_RATE and LOOP_TICS. The LOOP_TICS must be set to the count of tics of the serial input or output loop for one cycle without any additional delay (usually 9 tics). For every generated delay loop, the total count of delay tics are saved in one of four different constant names of the C-preprocessor BIT_CLOCKS_0, BIT_CLOCKS_1, BIT_CLOCKS_2 and BIT_CLOCKS_3 . Before a new delay loop is generated, the C-preprocessor checks, if any of the already generated code for a delay match the new requirement. Because also the file soft_uart.S must be included to generate the code for the serial input and serial output function, you can repeat this include for another serial interface. But you must set a additional constant for differing the name in the functions. If you set the SOFT_UART_NUMBER to 1 (#define SOFT_UART_NUMBER 1) before the #include, the serial input function is named getch_1 and the serial output function is named putch_1. If you define a constant named NO_SOFT_UART_TX before the #include of soft_uart.S, no serial output function is generated by this include. The same is done with the serial input function, if you specify the constant NO_SOFT_UART_RX.

3.7.3 Serial Input and Output with only one AVR Pin

Sometimes it makes sense to operate the serial communication only with one pin, to unlock one of the few IO pins of small AVRs for other use. With a special circuit technology can be achieved that you can read in data in the output pauses. The software solution of optiboot can only use the half-duplex operation in any case. Thus, at one time, only either data can be send or data can be received. Normally the output pin with the TX function becomes high in the transmission pauses, which prevent a data reading on the same pin. But if the TX output pin is switched to input mode instead of the high level, a external pull-up resistor can provide the required high level. In contrast to the fixed high level now a externally connected TX signal can pull down the level and let the input function read the low level. A serial resistor in the connection between the common TX/RX pin of the AVR with the external TX output can serve as pull-up resistor, because the idle state if the interface is a high signal. In addition, this serial resistor provides current limiting, if both TX interfaces send at the same time. The external RX interface must be connected directly to the common TX/RX pin to enable the reading of the external RX interface. The figure 3.5 should illustrate this simplest connection.

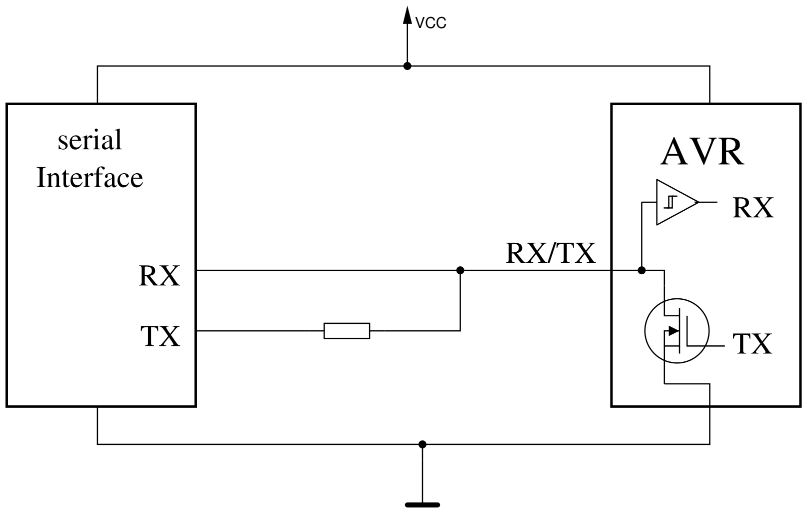

Figure 3.5. Possible serialconnection to a AVR with one Pin

Now is the problem, that the mostly full-duplex capable external interface will read back the own output. The communication program like **avrdude** at the external side is usually unprepared for this condition. One solution for this problem is to adapt the program at the external side to handle this read back. But I believe, that this is not a good idea, because this special version will have no maintenance or you must repeat this adaption for new versions constantly. That's why I prefere a hardware solution. A electronic circuit between the both endpoints must differ, from which side the data are send. A corresponding circuit suggestion is shown in figure 3.6

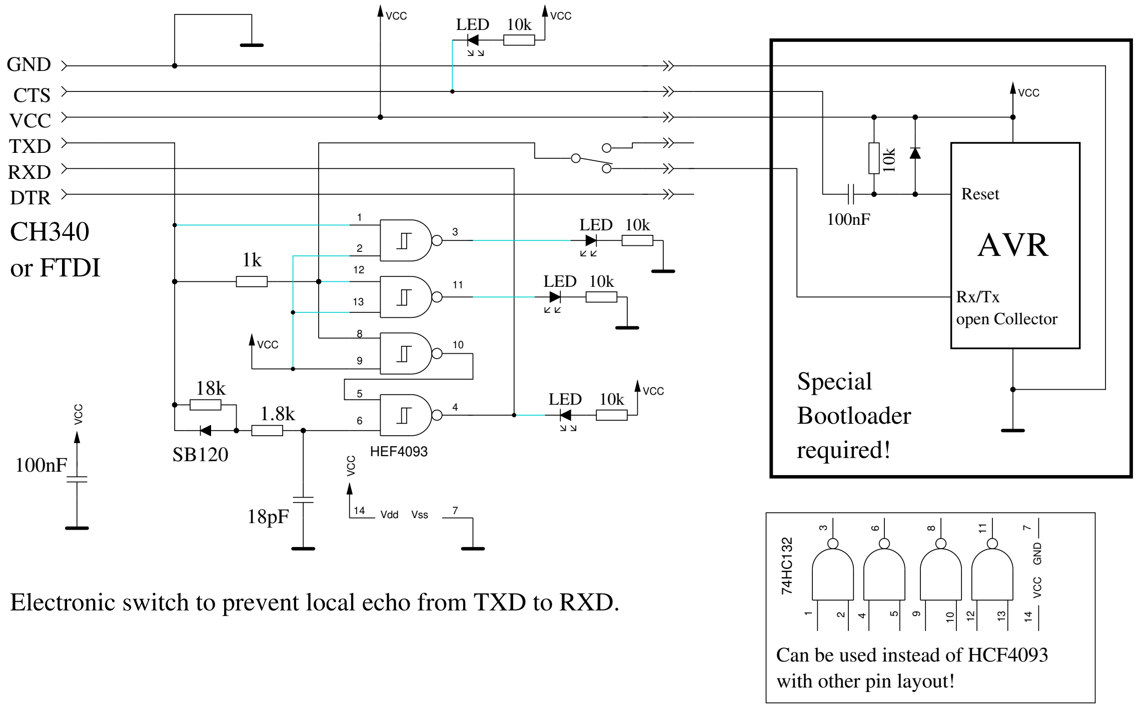

Figure 3.6. Circuit for suppressing the echoes

The TX data from the PC are transfered directly with the $1k\Omega$ resistor to the combined RX/TX input pin of the AVR. If the AVR sends data back to the PC, the output can be switch the level close to 0V, because the $1k\Omega$ resistor can not avoid this, even though the PC's TX level remains switched to VCC level. The date are passed with the both NAND gates of the HEF4093 chip to the PC's RX input a little bit delayed. The 0V level of the PC's TX signal will prevent this The 0V level on the PC-TX prevents exactly this passing on of data and the RX level remains at VCC, i.e. on the idle signal of the serial transmission. Problems are given only by the signal change of the PC-TX signal. The circuit consisting of a diode, two resistors and a $18pF$ capacitor forward the falling edge faster to NAND input 6 than the rising edge. only two NAND gates are required for the function. the other two NAND gates are used for a optional display of TX/RX activity. With this circuit you can use an unchanged transfer program like **avrdude** for example. The circuit is largely independent of the selected baud rate. I expect only problems with very high baud rates. At 115200 baud the circuit operates well. Much higher baud rates can not be used in most cases with the software serial interface. A simular circuit with a 74HC14 Hex Schmitt Inverter is shown in figure 3.7.

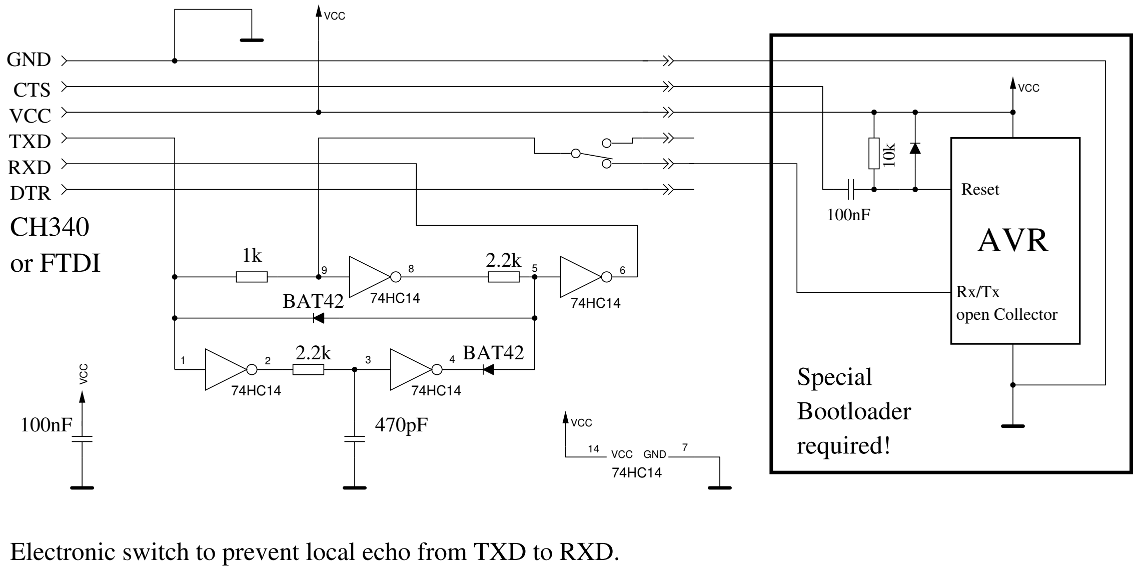

Figure 3.7. Alternative circuit for suppressing the echoes

3.7.4 Use of the automatic baud rate detection

If you specify a baud rate below $100 Baud$ for generating the bootloader code, the program will be prepared to detect the baud rate from the first received character. At the begin if the transmission with the STK500 protocol the computer send a command STK_GET_SYNC (0x30) followed by the control character CRC_EOP (0x20). For the serial transmission protocol the High level (1) is defined as the idle state. The begin of a transmission is started by s start bit with Low level (0). Directly after the end of the start bit the first data bit is send beginning with the low order bit. After the last of the aggreed count of bits (8) is transmitted, the transmission is finished with one or more Stop bits at High level (1). The level of every bit is hold exactly the aggreed baud time. The time between the end of the last Stop bit and the begin of the next Start bit is unspecified. The figure 3.8 shows the expected results for all possible time measurement. At the time axis "t" four possible start positions are marked for a complete time measurement of the assumed character. The counter is started in each case of a detected 1-0 slope. The right start position for the STK_GET_SYNC character is marked with "1". The marks "2", "3" and "4" show the situation for three possible wrong Start bit detections. For the mark "2" and "4" a 1-0 slope of data bits is detected as Start bit in a wrong way. For the mark '3' is a Start bit correctly detected, but for the wrong character (CRC_EOP). The wrong detections are allways possible, if the bootloader is started not before the transmission was running. The expected counter values are marked at the counter level graph for all four start positions in figure 3.8. The "b" means the counter reading for one bit transmission time (baud time), the "d" stand for a possible time delay between the end of the Stop bit transfer and the start of a new Start bit. The "D" represents a expected long time delay to the begin of the next message. The transmit sequence of the message has reached it's end and the computer wait for an answer of the AVR. In the "Rx"-row the data bits are labled with "0" (least significant bit) to "7" (highest significant bit). The Start bit has the label 'A' and a Stop bit has the label 'E'. The first byte is the coding of the STK_GET_SYNC character and the second byte is the coding of the control character CRC_EOP.

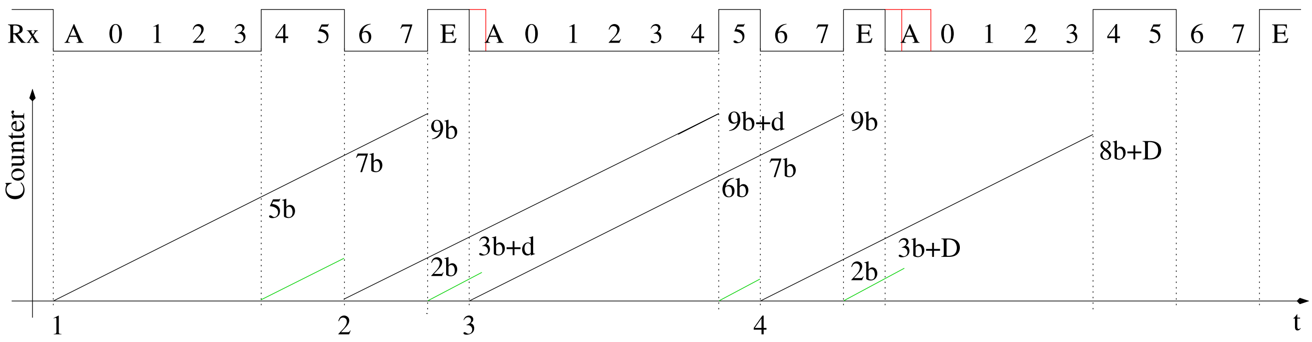

Figure 3.8. Possibel timer measurements for the STK_GET_SYNC sequence

- Simplest way of baud rate measurement, BAUD_RATE 20-29

  The simplest form of baud rate measurement doesn't take into account the wrong start bit choise. It is assumed, that the detected Start bit belongs to the STK_GET_SYNC character. After the Start bit detection the program waits to the next 0-1 slope and starts a 16-bit counter with the frequency F_CPU/8 with the stating value -1. At the next 1-0 slope the counter value is read out and the half of this value is taken as frequency divider for the UART. The building of the half value is required because the time of two data bits (4-5) was measured. Normally 1 must be subtracted for the UART frequency divider. But when halving should be rounded, a 1 must be added before the halving. Because also 2 can be subtracted from the counter result instead of 1 from the division result, totally only one must be subtracted from the counter result. In order to save computation steps, the subtraction (2-1=1) from the count is done by a start value of the counter of -1.

  This method works well, if the bootloader is started automatically just before the transmission by generating a reset pulse from the DTR (Data Terminal Ready) signal of the serial interface. You can also be successfull without the automatically generated Reset, if a Reset key is connected to the AVR, which should be hold pressed until the transmission program is started. But this procedure requires some feeling for the right release time of the button.

  The bootloader only waits a maximum time for the beginning of the transmission, which is specified from the watchdog timer. When the watchdog timer triggers a reset, usually the user program is started. Only if no user program has been loaded yet, the boot loader starts again for a retry.

- Improved way of baud rate measurement, BAUD_RATE 30-39

  Same as the simple way of baud rate measurement this method does not check the correctness of the bit change sequence. But the software wait for the next 0-1 slope until the counter value is read. So this counter reading is the time of four data bits (4-7), which is then used for the baud rate determination. Because the time recording is slightly inexact because of the polling loop, the baud time result will be better with dividing by four than by dividing by two. For that reason this method should be prefered, if little flash memory use is important. This improved method use only 4 byte more than the simple way.

- Simplest way of baud rate measurement with time limit, BAUD_RATE 40-49

  The same method as the simplest way is used for this method. Only a additional limitation for the timer value is used here. If the time limit is passed over, the program starts again with the search for a new Start bit. Without this limitation the program can stay in a loop for detecting a low level for the start condition "4". Probably the bad situation with the start position "4" in the diagram 3.8 can be handled a little better. The restart of the start bit search with the time limit would delay a reset from the watchdog timer only a little bit. The bootloader program may not reset the watchdog timer inside the search loop, because then the user program is never started without the transmission of a new program. For this reason the time limit of the watchdog timer must be greater as the time delay for repeating the STK_GET_SYNC sequence without a answer to the previous STK_GET_SYNC sequence.

- Improved way of baud rate measurement with time limit, BAUD_RATE 50-59

  The baud rate measuring is done in the same way as described for the improved way (30-39). Simular to the method "40-49" the wait time for the 1-0 slope is limited to the overflow of the 16-bit counter. But because the counter value is normally read at the next 0-1 slope, the result is equal to the improved way without the time limit.

- Complex check of baud rate measurement, BAUD_RATE 60-69

  With this method the counter is started immediately with the detection of a Start bit and the counter value is read with every of the next 3 bit changes. The time sequence of bit changes for the STK_GET_SYNC character is measured with this method, if the start position is well ("1"). With this measured time sequence some plausibility checks are done. For a first is checked, that the difference between the third counter reading and the first counter reading is less than the value of the first reading. For the start position "1" this is the case $((9b - 5b) < 5b)$. For the wrong start position "3" this is unfortunately also the case $((9b - 6b) < 6b)$ But this test will fail for start condition "2", if the time delay "d" of the second start bit of the STK_GET_SYNC sequence is sufficiently small. For the wrong start position "4" can help only a time limitation for the second bit change wait loop, because the repetition of the STK_GET_SYNC is done after a longer wait time for the answer ("D").

  A second test checks, if the difference between the third and the second counter reading isn't significant greater than the difference between the second and the first counter reading. For the correct start posistion "1" this result to the equation $((9b\text{-}7b) < (7b\text{-}5b+4))$ or $(2b < 2b+4)$. For the false start position "2" you get the equation $((9b\text{-}3b) < (3b+d\text{-}2b+4))$ or $(6b < (b+d+4))$. For the false start position "3" we get the equation $((9b\text{-}7b) < (7b\text{-}6b+4))$ or $(2b < (b+4))$. This exams are relatively safe for detecting the right baud rate, but they require a lot of additional space for the program. This method is especially recommended, if at least 1024 bytes are reserved for the boot loader anyway (boot loader page size). For setting the correct UART frequency divider, the difference between the third counter reading and the second counter reading is divided by 2 $(9b\text{-}7b\text{-}1)/2 = (2b+1)/2\text{-}1$.

- Complex check of baudrate measuring, BAUD_RATE 70-79

  All counter readings and checks are done in the same way as the BAUD_RATE 60-69 method. Only the calculation of the UART divider is based on the time of four bits instead of two bits. Consequently the equation for the correct starting position look like (9b-5b-2)/4 = (4b+2//4-1.

- Complex check of baudrate measuring, BAUD_RATE 80-89

  All counter readings and checks are done in the same way as the BAUD_RATE 60-69 method. Only the calculation of the UART divider is based on the time of nine bits instead of two bits. Consequently the equation for the correct starting position look like $(9b\text{-}4)/9 = (9b+5)/9\text{-}1$.

- Simple way of baud rate measurement, BAUD_RATE 10-19

  With this selection the time of 9 serial bits is measured without checks to build the baud rate. Usually the measurement of the time of 4 serial data bits should be sufficient to determine the baud rate.

I recommend for all methods of the baud rate determination to omit the LED flashing at the begin of the bootloader program to prevent a delay of the start bit detection. Unfortunately even the simplest way of baud rate measurement need so much additional program memory, that the bootloader will not fit into the 512 byte limit, if the EEprom support is selected (SUPPORT_EEPROM=1) with the flashing LED function. For some processors you can select the additional function LED_DATA_FLASH without exceeding the 512 byte limit, when the EEprom support is deselected. If the 512 byte limit is overshoot by a required function, the next limit of 1024 byte give enough space for all additional selections. You can select the SOFT_UART function together with the automatic baud rate detection (BAUD_RATE <100) only for the assembler version of optiboot, not for the C-version.

The following table 3.6 summarize the different options. The specified program sizes in bytes refer to an ATmega328 without LED flashing function, but with the EEprom support. The program sizes in brackets result from the operation of the serial interface with software.

BAUD_RATE	Bit Base	Clock Base	Time Limit	Check	mega328 size (HW)
10-14	9	clk/8	-	simple	502
15-19	9	clk	-	simple	502
20-24	2	clk/8	-	simple	490
25-29	2	clk	-	simple	494
30-34	4	clk/8	-	simple	494
35-39	4	clk	-	simple	494
40-44	2	clk/8	Yes	simple	496
45-49	2	clk	Yes	simple	500
50-54	4	clk/8	Yes	simple	500
55-59	4	clk	Yes	simple	500
60-64	2	clk/8	Yes	complex	550
65-69	2	clk	Yes	complex	554
70-74	4	clk/8	Yes	complex	554
75-79	4	clk	Yes	complex	554
80-84 90-94	9	clk/8	Yes	complex	564
85-89 95-99	9	clk	Yes	complex	564

Table 3.6. Setting options for the baud rate measurement.

There are some differencies and special features for the baud rate measuring together with the Soft UART solution compared to the hardware UART solution. Usually a 8-Bit counter loop is used to generate the delay for the half baud time. Because of the used instructions in this loop, you can select the time only with a solution of 3 clock tics. To get the full baud time the loop must be called twice, so that the resolution is doubled too. With a known fixed baud rate this error will be compensated by extra instructions. But this is impossible, if the baud rate is unknown in advance. To get the best possible rounding of the selected baud rate, the time measurement should be done with the 16-bit counter at the full CPU clock rate (F_CPU). For the hardware UART the counter can be used with F_CPU/8, because the UART use a identical clock rate and the time is measured for 2 or 4 bits.

The use of a 8-bit delay loop result to a upper limit of the baud time depending on the CPU clock rate. The resulting minumum baud rate is definite higher with the 8-bit loop compared to the minimum baud rate with the hardware UART. With the limited resolution of the period selection for the baud rate the upper limit can be specified with a guaranteed baud rate error below 2%. If the AVR processor is operated with a RC clock generator, the clock rate is usually imprecise. By measuring the baud rate with a counter controlled with the same clock, this error is compensated. But the error by reason of the limited resolution of the baud rate divider can not be calculated in advance. I would like to show you what I mean by an example for the hardware UART. The baud rate $250kHz$ can be used with a clockrate of exactly $8MHz$. If you select the divider of 4, the desired baud rate is together with the factor 8 prescaler produced without a deviation. If I assume a inexact real clock rate of $7.6MHz$, the best selectable divider is still 4. For this case the actual baud rate is now $237.5kHz$ with the same error of -5% as the base clock rate. To overcome at least the lower baud limit with the software UART solution, all baud rate selections between 20 and 99, which are odd-numbered, will generate a code with a 15-bit delay loop. Unfortunately the pass time of this loop is 5 tics for the 15-bit loop. Because of the double delay call for the full baud time, the resolution is only 10 tics and thereby more worth than the 8 tics resolution of the hardware UART caused by the 8:1 prescaler. You should use this option only, if you need the support of low baud speeds and deside not to use very high baud rates. The following table 3.7 is intended to clarify the use of the different auto baud functions at the operating frequency $8MHz$. The table does not respect the time error, which can be caused by the polling loops for the RX signal.

BAUD_RATE Option	SOFT_ UART	Minimum Baud	BAUD-Err <4% @ Baud	Comment
10-14,20-34 40-54,60-64 70-74,80-84	0	244	80.0k	HW_UART 2-Bit Time or CLK/8 measurement
35-39,55-59	0	488	80.0k	HW_UART 4-Bit Time, CLK
15-19,85-89 25-29,45-49 65-69,75-79	0	1098	80.0k	HW_UART 9-Bit Time, CLK HW_UART 2-Bit Time, CLK HW_UART 2/4-Bit Time, CLK
42	1	5151	81.6k	Simple, 2-Bit Time, 8-Bit Loop
62	1	5151	81.6k	Complex, 2-Bit Time, 8-Bit Loop
72	1	5151	81.6k	Complex, 4-Bit Time, 8-Bit Loop
47	1	244	53.3k	Simple, 2-Bit Time, 15-Bit Loop
67	1	1220	53.3k	Complex, 2-Bit Time, 15-Bit Loop
77	1	1220	53.3k	Complex, 4-Bit Time, 15-Bit Loop

Table 3.7. Limits for the automatic baud rate selection with a $8MHz$ clock.

Usually a limit for the baud error is set to only 2%, because the transmitter and the receiver can have a baud rate error to the opposite direction. The Auto-baud function measure the actual baud rate of the transmitter, so that the error rate of the Auto-baud function can take the double value of 4%. If you select a hardware UART interface, you can select the standard baud rates 1200, 2400, 4800, 9600, 19200 and 38400 with all arbitrary options. Higher baud rates than the specified 40 kBaud are not safe to use, although 57600 baud also worked well in tests. Likewise, a test with 115.2 kBaud at $16MHz$ crystal operation was still successful when using a software UART with a 15-bit delay loop. If only a 8-bit delay loop is used with the software UART solution, the baud rates 1200, 2400 and 4800 can not be used. The lower baud rates are only usable with the software UART, if you configures the optiboot with a 15-bit delay loop. The different limit for the 49 and 69 or 89 baud rate selection is caused by the different use of the 16-bit AVR counter. With the 49 selection (all below 60) only the time of two data bits is measured with the counter. For selections above 59 the complete byte sequence is measured from the Start bit to the Stop bit with the 16-bit counter. Probably baud rated below 9600 are rarely used anyway. Of course the limits of the baud rates change with other processor clock rate.

There are numerous setting options for the UART interface software, which on the one hand relate to the measurement base of the baud time, but also relate to the type of delay calculation. In the table 3.8 the program lengthes of the bootloade are shown for a ATmega328 target processor. Unfortunately the programs in the colums labled with _0 and _5, this means all Baud rate selections ending with the digit 0 or 5, are practical unusable. The reason herefore is, that the division operation by successive subtraction is too slow to finish before the serial data stop bit ends. A workaround would be to let the transmitter of the data (**avrdude**) send all data with 2 stop bits. Because this is not intended by the **avrdude** version 6.3, the other columns use an accelerated subtraction loop, which then causes a longer program. If possible, the faster shift operation for division by powers of 2 is used instead of the subtraction loop. To enable this shift operations, the colums labled with _3, _4, _8 and _9 has increased the number of cycles for one delay loop pass (8T or 16T, T for tics). Unfortunately the program length is only shorter compared to the subtraction loop, if the count of shift operation is below 3, which would spare the loop operation for the shifts. But you can save 14 bytes of flash with the option NO_EARLY_PAGE_ERASE , so that about 30 versions more match to a 512 byte limit.

BAUD_RATE ,Measurement base	_0 6T/8T	_1 10T	_2 6T	_3 16T	_4 8T	_5 10T	_6 6T	_7 10T	_8 8T	_9 16T
	simple CLK/8					simple CLK/1				
1_ , 9Bit	512	526	518	530	518	522	518	528	520	532
2_ , 2Bit	514	530	520	522	**508**	518	514	524	**512**	522
3_ , 4Bit	510	526	516	524	**512**	520	516	528	514	532
4_ , 2Bit	520	536	526	528	514	524	520	530	518	528
5_ , 4Bit	516	532	522	530	518	526	522	534	520	538
	komplex CLK/8					komplex CLK/1				
6_ , 2Bit	574	590	580	582	578	578	574	584	572	582
7_ , 4Bit	572	588	578	592	580	580	576	588	580	592
8_ , 9Bit	574	588	580	592	580	584	580	590	582	594

Table 3.8. Bootloader program length with automatic Baud rate selection for Software UART

Finally I would like to show the results from tests, which I have done with a ATmega1281 with the internal $8MHz$ RC-generator clock at a specified baud rate and also above the specified baud rate. The frequency was tuned with the OSCCAL_CORR option in steps of two and of course measured. The functionality was tested by loading a small user program. All tests within the specified baud rate was successfull. Because all tests run well, additionally tests with a higher baud rate than specified ($115.2kBaud$) was also done.

OSCCAL _CORR	Freq. MHz	HW-UART 57600 Mode 82	SW-UART 57600 Mode 82	HW-UART 115200 Mode 56	SW-UART 115200 Mode 52	SW-UART 115200 Mode 57
20	7.18	OK	OK	Err	Err	Err
18	7.22	OK	OK	Err	Err	Err
16	7.35	OK	OK	OK	OK	Err
14	7.45	OK	OK	OK	OK	OK
12	7.56	OK	OK	OK	OK	OK
10	7.63	OK	OK	OK	OK	OK
8	7.75	OK	OK	OK	OK	OK
6	7.84	OK	OK	Err	OK	Err
4	7.98	OK	OK	Err	Err	Err
2	8.04	OK	OK	Err	OK	Err
0	8.18	OK	OK	OK	OK	Err
-2	8.29	OK	OK	OK	OK	Err
-4	8.42	OK	OK	OK	OK	Err
-6	8.51	OK	OK	OK	OK	OK
-8	8.64	OK	OK	OK	Err	OK
-10	8.77	OK	OK	OK	OK	OK
-12	8.92	OK	OK	Err	OK	OK

Table 3.9. Test for the automatic baud rate at $8MHz$ clock.

For the operating mode 82 I have additionally checked the odd OSCCAL_CORR settings without any noticeable difficulties. The also checked simplest mode 42 has not shown any difficulties with the tested even OSCCAL_CORR values of the table.

Only at the extemely high baud rate of $115,2k$ for this CPU frequency the operation mode 52 shows four failures and with the operation mode 57 eight failures are stated. The increase in failures in mode 57 with the 15-bit delay loop is expected and caused by the coarser grid of adjustable delay times.

A test with a chinese Arduino UNO board, which use a CH340G chip as USB-serial converter, could operate only up to $38.4kBaud$ correctly. With higher baud rates the read back of the flash data caused problems. Probably the last byte of a package is sometimes not transmitted to the host and the communication blocks. The same test with a other Arduino UNO board, which use a Mega16U2 controller for USB-serial conversion, didn't show the same problem. This board could run with $115.2kBaud$ and also with $230.4kBaud$. Probably the reason is, that both processors (ATmega328p and Mega16U2) use a real baud rate of $250kBaud$.

3.7.5 Special features of the serial interface

Usually optiboot send the serial data with a prolongate stop bit. The hardware UART switch on a second stop bit for transmitted data and the software solution use a additional half stop bit. This seems to be usefull with little differences of the baud rate between sender and receiver, because there is more time for the receiver to synchronize with the next start bit. The PC side with **avrdude** version 6.3 could send data with two stop bits. So I had pached the sources for **avrdude** to add this feature with the additional option -S. But unfortunately this feature is missing with **avrdude**.

You can disable the two stop bit selection of optiboot with the option TWO_STOP_BITS=0. The difference in speed is insignificant, so I would recommend to use the longer stop bit.

The software for the serial interface (SOFT_UART=1) use the delay time for the stop bits at the begin of the serial transmission. With that feature the program has more time at the end of transmission to prepare the receiving of serial data. You can also disable this feature with the option QuickStopBit=0. But I know no problems or disadvantages with this feature, not even when operating with just one pin.

3.8 Some examples of building a optiboot bootloader

The first example is the building of a bootloader for the popular ATmega328P:

```
make atmega328p
```

```
Optiboot for 16000000 Hz (16.00 Mhz) operation with Baudrate 115200 and EEprom support \
 configured.
 >>> Start building for AVR atmega328p:
LED-Pin PB5 use Pin 19-PDIP28 17-TQFP32, with special functions: SCK PCINT5
 RX-Pin PD0 use Pin  2-PDIP28 30-TQFP32, with special functions: PCINT16 RXD
 TX-Pin PD1 use Pin  3-PDIP28 31-TQFP32, with special functions: PCINT17 TXD
avr-gcc -g -Wall -Os -fno-split-wide-types -mrelax -mmcu=atmega328p -fno-diagnostics-show-caret \
 -DBAUD_RATE=115200 -DLED_START_FLASHES=3 -DSUPPORT_EEPROM=1 -DLED=pB5 -DUART=00 -DSOFT_UART=0 \
 -DUART_RX=pD0 -DUART_TX=pD1 -DF_CPU=16000000 -DHFUSE=hexDE -DLFUSE=hexFF -DBOOT_PAGE_LEN=512 \
 -DVerboseLev=2 -c -o optiboot.o optiboot.S

--------------------------------------------------------------------------------
BAUD RATE CHECK: Desired: 115200,  Real: 117647, UBRR = 16, Difference=2.12%
--------------------------------------------------------------------------------
# # # # # # # # # # # # # # # # # # # # # # #
Boot Loader start address: 0x7E00 = 32256
# # # # # # # # # # # # # # # # # # # # # # #

   text    data     bss     dec     hex filename
    488       0       0     488     1e8 optiboot.elf
Requires 1 Boot Page of 512 Bytes, which is 1.5% of Flash Memory
BOOTSZ=3, which means 1 Boot Pages
```

With no additional option a baudrate of 115200 with a clock frequency of $16MHz$ is selected. For serial output the hardware interface is selected. You should notice, that the systematic baud rate error is above 2% with the hardware UART. The second example with the same processor is done with a software solution for the serial interface.

```
make atmega328p SOFT_UART=1
```

```
Optiboot for 16000000 Hz (16.00 Mhz) operation with Baudrate 115200 and EEprom support \
 configured.
 >>> Start building for AVR atmega328p:
LED-Pin PB5 use Pin 19-PDIP28 17-TQFP32, with special functions: SCK PCINT5
 RX-Pin PD0 use Pin  2-PDIP28 30-TQFP32, with special functions: PCINT16 RXD
 TX-Pin PD1 use Pin  3-PDIP28 31-TQFP32, with special functions: PCINT17 TXD
avr-gcc -g -Wall -Os -fno-split-wide-types -mrelax -mmcu=atmega328p -fno-diagnostics-show-caret \
 -DBAUD_RATE=115200 -DLED_START_FLASHES=3 -DSUPPORT_EEPROM=1 -DLED=pB5 -DUART=00 -DSOFT_UART=01 \
 -DUART_RX=pD0 -DUART_TX=pD1 -DF_CPU=16000000 -DHFUSE=hexDE -DLFUSE=hexFF -DBOOT_PAGE_LEN=512 \
 -DVerboseLev=2 -c -o optiboot.o optiboot.S

--------------------------------------------------------------------------------
```

```
BAUD RATE CHECK: Desired: 115200, SoftUART_Real: 115107, Delay: 116*1, Difference=-.07%
--------------------------------------------------------------------------------
# # # # # # # # # # # # # # # # # # # # # # #
Boot Loader start address: 0x7E00 = 32256
# # # # # # # # # # # # # # # # # # # # # # #

   text    data     bss     dec     hex filename
    504       0       0     504     1f8 optiboot.elf
Requires 1 Boot Page of 512 Bytes, which is 1.5% of Flash Memory
BOOTSZ=3, which means 1 Boot Pages
```

Please note, that the software solution for the serial interface requires a little more flash memory, but take use of only 1 boot page too. The systematic baud rate error is much smaller than the error with the hardware UART. But the hardware UART has the advantage, that the input and output could be done simultaneous and is more tolerant against short disturbance of the input signal. For the software serial interface you can specify every digital IO-pin for input (UART_RX) and output (UART_TX). In this example the feature is used to automatically select the RX and TX of the hardware UART. The automatic IO-pin selection depends on the selected processor type and the selected UART number, if more than one UART is available.

The last examples shows a configuration with the new automatic selection of the baud rate by measuring the rate of the first incoming data. The flashing of the LED at the program start is deselected in the first example to save flash memory.

```
make atmega328p LED_START_FLASHES=0 BAUD_RATE=52
```

```
Optiboot for 16000000 Hz (16.00 Mhz) operation with Auto-Baudrate and EEprom support \
 configured.
 >>> Start building for AVR atmega328p:
LED-Pin not used!
 RX-Pin PD0 use Pin  2-PDIP28 30-TQFP32, with special functions: PCINT16 RXD
 TX-Pin PD1 use Pin  3-PDIP28 31-TQFP32, with special functions: PCINT17 TXD
avr-gcc -g -Wall -Os -fno-split-wide-types -mrelax -mmcu=atmega328p -fno-diagnostics-show-caret \
 -DBAUD_RATE=52 -DLED_START_FLASHES=0 -DSUPPORT_EEPROM=1 -DLED=p -DUART=00 -DSOFT_UART=0 \
 -DUART_RX=pD0 -DUART_TX=pD1 -DF_CPU=16000000 -DHFUSE=hexDE -DLFUSE=hexFF -DBOOT_PAGE_LEN=512 \
 -DVerboseLev=2 -c -o optiboot.o optiboot.S

--------------------------------------------------------------------------------
Simple Baudrate measurement with time limit implemented in optiboot! (4-bit, CLK/8)
UART Minimum 976 Baud, Difference surely less than 4% up to 160.0 kBaud
--------------------------------------------------------------------------------
# # # # # # # # # # # # # # # # # # # # # # #
Boot Loader start address: 0x7E00 = 32256
# # # # # # # # # # # # # # # # # # # # # # #

   text    data     bss     dec     hex filename
    500       0       0     500     1f4 optiboot.elf
Requires 1 Boot Page of 512 Bytes, which is 1.5% of Flash Memory
BOOTSZ=3, which means 1 Boot Pages
```

for the last example the most complex methode of baud rate measurement was selected because the limit of 512 byte would be exceeded with selecting the LED flashing and the simplest measurement method (534 byte).

```
make atmega328p BAUD_RATE=76
```

```
Optiboot for 16000000 Hz (16.00 Mhz) operation with Auto-Baudrate and EEprom support \
 configured.
 >>> Start building for AVR atmega328p:
LED-Pin PB5 use Pin 19-PDIP28 17-TQFP32, with special functions: SCK PCINT5
 RX-Pin PD0 use Pin  2-PDIP28 30-TQFP32, with special functions: PCINT16 RXD
 TX-Pin PD1 use Pin  3-PDIP28 31-TQFP32, with special functions: PCINT17 TXD
avr-gcc -g -Wall -Os -fno-split-wide-types -mrelax -mmcu=atmega328p -fno-diagnostics-show-caret \
 -DBAUD_RATE=82 -DLED_START_FLASHES=3 -DSUPPORT_EEPROM=1 -DLED=pB5 -DUART=00 -DSOFT_UART=0 \
 -DUART_RX=pD0 -DUART_TX=pD1 -DF_CPU=16000000 -DHFUSE=hexDE -DLFUSE=hexFF -DBOOT_PAGE_LEN=512 \
 -DVerboseLev=2 -c -o optiboot.o optiboot.S

--------------------------------------------------------------------------------
Complex Baudrate measurement implemented in optiboot! (4-bit, CLK/1)
UART Minimum 2197 Baud, Difference surely less than 4% up to 160.0 kBaud
--------------------------------------------------------------------------------
# # # # # # # # # # # # # # # # # # # # # # #
Boot Loader start address: 0x7C00 = 31744
# # # # # # # # # # # # # # # # # # # # # # #

   text    data     bss     dec     hex filename
    612       0       0     612     264 optiboot.elf
Requires 2 Boot Pages, 512 Bytes each, which is 3.1% of Flash Memory
BOOTSZ=2, which means 2 Boot Pages
```

3.9 Clock Frequency Correction of the internal RC-Generator

The use of the serial interface is only possible, if the selected baudrate is matched by both interfaces with only two percent deviation. The actual baudrate is given by the processor clock and the selected scaling factor for the serial IO-clock. The hardware UART interface scaled the processor clock with factor 8 or 16 and a additional selectable divider between 1:1 and 1:4096 for generating the clock for the serial IO. For lower baudrates additional dividers with power of 2 can be additional selected. If the relationship between the processor clock and the baudrate clock is sufficiently high, the desired baudrate can be selected with low deviation. By generating the optiboot bootloader code the systematic error is shown at the terminal protocol. Usually errors lower than 2% are uncritical. The implemented coding of the serial interface with software (SOFT_UART) produce lower systematic errors as the hardware UART. The problems with the software solution of the serial interface is caused with no filter for input data and the missing feature to organize the interface full duplex. From the output of the last bit to the ability to receive the next data is allways a little time delay. For this reason you can expect fewer difficulties with lower baudrates for the software UART. All this considerations assume however, that the processor clock itself is selected with sufficient accuracy. With a crystal or ceramic resonator the clock frequency is usually accurate enough without special activities. But for the internal RC-generator of the AVR processors the situation is different. The actual processor clock can differ too much from the desired value. The processors are precalibrated at factory. But this calibration is valid only for one temperature and operating voltage. The sensivity of frequency changes with temperature and operating voltage differ for the different processor types. To enable a correction for the user of the processor, the calibration value of the clock frequency is copied to a special IO register with the name OSCCAL at every start of the processor. The Optiboot bootloader can use the option OSCCAL_CORR to correct a known residual error of the clock frequency.

If the actual clock rate is uncritical for your application and you can spend the additional memory space to the bootloader, you can also use the automatic baud rate selection of optiboot. The automatic baud rate selection is implemented by the optiboot bootloader, if you select a baud rate below 100 at the generation time. You can find more details about the automatic baud rate selection in subsection 3.7.4 at page 32. But you should keep in mind, that the deviation to the expected clock rate will also affect the application program (if it will use a serial interface).

In the following subsections I have analysed the correction of the RC clock frequency for some AVR examples.

3.9.1 RC-generators check of the ATmega8

The Atmega8 can select 4 different frequencies for the internal RC generator with the Low-fuse, $1MHz$, $2MHz$, $4MHz$ and $8MHz$. In the table 3.10 I have analysed all 4 selections.

AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
			Corr	Freq	Corr	Freq	Corr	Freq
1M	0xA1	9.6k	-8	1050k	4	983k	0	1004k
2M	0xA2	19.2k	-8	2098k	4	1967k	0	2008k
4M	0xA3	19.2k	-2	4201k	10	3927k	7	3999k
8M	0xA4	57.6k	0	8231k	13	7723k	6	7990k

Table 3.10. Possible OSCCAL_CORR selections for the RC-frequencies of the ATmega8

The table 3.10 shows, that for the $1Mhz$ and $2MHz$ operation a correction of the OSCCAL register is not required. This ATmega8 is calibrated for these frequencies very good at the factory. For the $4MHz$ clock frequency a operation without correction is still possible, but the correct clock frequency is better approximated with a OSCCAL_CORR value of 7. For the $8MHz$ clock frequency the serial interface was still possible withou the correction, but the serial interface runs more safely with the OSCCAL_CORR value 6.

3.9.2 RC-Generators check of the ATmega8535

The ATmega8535 can select 4 different frequencies for the internal RC generator with the Low-fuse, $1MHz$, $2MHz$, $4MHz$ and $8MHz$. The table 3.11 shows the results for one example for all 4 frequencies.

AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
			Corr	Freq	Corr	Freq	Corr	Freq
1M	0xA1	9.6k	-10	1053k	3	982k	0	1001k
2M	0xA2	19.2k	-9	2095k	4	1965k	1	1998k
4M	0xA3	19.2k	-5	4204k	8	3932k	4	4012k
8M	0xA4	19.2k	-7	8420k	6	7901k	3	8003k

Table 3.11. Possible OSCCAL_CORR selections for the RC-frequencies of the ATmega8535

3.9.3 RC-Generators check of the ATmega8515 and the ATmega162

The ATmega8515 can select 4 different frequencies for the internal RC generator with the Low fuse, $1MHz$, $2MHz$, $4MHz$ and $8MHz$. The table 3.12 shows the results of one exemplar for all 4 frequency selections.

AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
			Corr	Freq	Corr	Freq	Corr	Freq
1M	0xA1	9.6k	-10	1053k	2	985k	-1	997k
2M	0xA2	19.2k	-10	2099k	3	1963k	-1	1999k
4M	0xA3	38.4k	-3	4192k	10	3928k	7	3979k
8M	0xA4	38.4k	-3	8396k	10	7860k	7	7966k

Table 3.12. Possible OSCCAL_CORR selections for the RC-frequencies of the ATmega8515

The ATmega162 with a simular pin layout can only operate with the $8MHz$ RC generator frequency. Table 3.13 shows the result of one exemplar.

AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
			Corr	Freq	Corr	Freq	Corr	Freq
8M	0xE2	38.4k	0	8190k	6	7718k	2	8000k

Table 3.13. Possible OSCCAL_CORR selections for the RC-oscillator of the ATmega162

3.9.4 RC-Generators check of the ATmega328 family

For the ATmega328 family only a RC oscillator frequency of $8MHz$ can be selected. This Frequency can be divided by factor 8 with a fuse-bit, so that a operation with $1MHz$ can be also selected. The table 3.14 shows the results for the checked processors.

Typ	AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
				Corr	Freq	Corr	Freq	Corr	Freq
mega48P	8M	0xE2	57.6k	-6	8230k	8	7720k	0	8010k
mega88	8M	0xE2	57.6k	-2	8250k	10	7770k	4	7990k
mega168	8M	0xE2	57.6k	-5	8263k	8	7720k	1	7970k
mega328P	8M	0xE2	57.6k	-5	8250k	9	7723k	1	7992k

Table 3.14. Possible OSCCAL_CORR selections for the ATmega328 family

For all checked processors the serial interface can be used with the internal RC generator without any OSCCAL correction. Only for the checked ATmega88 a correction would be worthwhile (OSCCAL_CORR=4).

3.9.5 RC-Generators check of the ATmega32 / 16

You can select 4 different frequencies with the internal RC-generator for the ATmega32 and the ATmega16, $1MHz$, $2MHz$, $4MHz$ and $8MHz$. The tables 3.15 and 3.16 shows the results with one test exemplar each.

Typ	AVR_ FREQ	LFUSE	Baud-rate	Minimum Corr	Minimum Freq	Maximum Corr	Maximum Freq	Best Corr	Best Freq
m32	1M	0xA1	9.6k	-13	1049k	-1	980k	-5	1001k
m32a				-7	1046k	4	984k	1	998k
m32	2M	0xA2	19.2k	-12	2102k	0	1968k	-3	1997k
m32a				-7	2105k	6	1966k	2	2005k
m32	4M	0xA3	19.2k	-5	4169k	6	3942k	3	3993k
m32a				2	4192k	14	3939k	10	4015k
m32	8M	0xA4	19.2k	-7	8425k	6	7888k	3	7983k
m32a				2	8408k	14	7921k	11	8014k

Table 3.15. Possible OSCCAL_CORR selections for the RC-frequencies of the ATmega32

AVR_ FREQ	LFUSE	Baud-rate	Minimum Corr	Minimum Freq	Maximum Corr	Maximum Freq	Best Corr	Best Freq
1M	0xA1	9.6k	-11	1047k	1	982k	-2	998k
2M	0xA2	19.2k	-12	2099k	0	1971k	-3	1995k
4M	0xA3	19.2k	-9	4291k	3	3932k	0	4002k
8M	0xA4	19.2k	-11	8415k	2	7857k	-2	8013k

Table 3.16. Possible OSCCAL_CORR selections for the RC-frequencies of the ATmega16

Whenever positive values appear in the MinCorr column or negative values in the MaxCorr column, it is impossible to use the serial interface with this processor at this frequency without a frequency correction. If a 0 appear in any of the Corr columns, the operation of the serial interface is just possible.

3.9.6 RC-Generator check of the ATmega163L

The ATmega163L has only one $1MHz$ RC-generator, which can be adjusted with the OSCCAL register. My exemplar had no preselection of the OSCCAL value. Therefore, exceptiionally high correction values are required to select a clock frequency of about $1MHz$.

AVR_ FREQ	LFUSE	Baud-rate	Minimum Corr	Minimum Freq	Maximum Corr	Maximum Freq	Best Corr	Best Freq
1M	0x92	9.6k	-88	1026k	-62	964k	-77	998k

Table 3.17. Possible OSCCAL_CORR selections for the RC-frequency of the ATmega163L

3.9.7 RC-Generator check of the ATmega64 / 128

The ATmega64 and the ATmega128 can select 4 different frequencies for the internal RC-generator with the Low-fuse, $1MHz$, $2MHz$, $4MHz$ and $8MHz$. In then tables 3.18 and 3.19 all 4 frequencies are checked. At this point it should also be noticed, that the program data is loaded via the ISP interface not with the signals MISO and MOSI but via the signals TXD (PE1) and RXD (PE0). Of course this must be taken into account when connecting the processor to the programmer.

AVR_ FREQ	LFUSE	Baud-rate	Minimum Corr	Minimum Freq	Maximum Corr	Maximum Freq	Best Corr	Best Freq
1M	0xA1	9.6k	-4	1024k	6	975k	1	1000k
2M	0xA2	19.2k	-4	2047k	6	1952k	0	2015k
4M	0xA3	19.2k	4	4070k	10	3939k	8	3976k
8M	0xA4	57.6k	6	8028k	10	7847k	7	8005k

Table 3.18. Possible OSCCAL_CORR selections for the RC-frequencies of the ATmega64

AVR_ FREQ	LFUSE	Baud-rate	Minimum Corr	Minimum Freq	Maximum Corr	Maximum Freq	Best Corr	Best Freq
1M	0xA1	9.6k	-9	1051k	3	985k	0	999k
2M	0xA2	19.2k	-9	2102k	3	1971k	0	2000k
4M	0xA3	19.2k	-3	4209k	9	3960k	6	4006k
8M	0xA4	57.6k	0	8225k	13	7723k	7	8005k

Table 3.19. Possible OSCCAL_CORR selections for the RC-frequencies of the ATmega128

You can see at the tables, that there is no correction of the OSCCAL register necessary at $1MHz$ and $2MHz$ operation for using the serial interface. For operation at $4MHz$ or $8MHz$ the checked ATmega64 can not use the serial interface without a frequency correction. The $4MHz$ frequency is about 4% too high Without the correction and the $8MHz$ frequency is about 4.3% too high. You can find a hint in the Atmel documentation, that the RC-generator of the ATmega64 and ATmega128 is calibrated at $1MHz$. It should be noted once again, that the the tabular data is the test result of a single copy of the ATmega. Outside the specified minimum or maximum values of the OSCCAL-corrections, it was not possible to operate the serial interface at thespecified baud rate.

3.9.8 RC-Generator check of the ATmega644 family

For the ATmega644 family an RC oscillator frequency of $8MHz$ can be selected. In addition a $128kHz$ generator can be selected as the clock, which otherwise supplies the watchdog timer. The selected frequency can be pre-divided with a factor 8 fuse bit, so that a $1MHz$ operation is also possible. The table 3.20 shows the results for the checked processors.

Typ	AVR_ FREQ	LFUSE	Baud-rate	Minimum Corr	Minimum Freq	Maximum Corr	Maximum Freq	Best Corr	Best Freq
mega1284p	8M	0xC2	19.2k	-8	8416k	7	7882k	4	7989k
mega644p	8M	0xC2	19.2k	-12	8416k	3	7871k	-1	8009k
mega324p	8M	0xC2	19.2k	-12	8398k	3	7885k	0	7976k
mega164p	8M	0xC2	19.2k	-5	8401k	4	7888k	2	8012k

Table 3.20. Possible OSCCAL_CORR selections for the ATmega644 family

3.9.9 RC-Generator check of the ATmega645 family

For the ATmega645 family an RC oscillator frequency of $8MHz$ can be selected. Bei der ATmega645 Familie kann nur eine RC-Oszillatorfrequenz von 8MHz gewählt werden. The selected frequency can be pre-divided with a factor 8 fuse bit, so that a $1MHz$ operation is also possible. The table 3.21 shows the results for the checked processors.

Typ	AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
				Corr	Freq	Corr	Freq	Corr	Freq
mega165p	8M	0xE2	57.6k	-6	8235k	7	7718k	-1	8015k
mega325	8M	0xE2	38.4k	-10	8403k	5	7868k	1	7992k
mega645	8M	0xE2	57.6k	0	8253k	12	7726k	5	8012k

Table 3.21. Possible OSCCAL_CORR selections for the ATmega645 family

Beim ATmega645 ist der Betrieb der seriellen Schnittstelle ohne OSCCAL Korrektur gerade noch möglich. Sicherer ist aber der Betrieb mit OSCCAL_CORR=5, da dann die 8MHz besser eingehalten werden.

3.9.10 RC-Generator check of the ATmega649 family

For the ATmega649 family an RC oscillator frequency of $8MHz$ can be selected. The selected frequency can be pre-divided with a factor 8 fuse bit, so that a $1MHz$ operation is also possible. The table 3.22 shows the results for the checked processors.

Typ	AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
				Corr	Freq	Corr	Freq	Corr	Freq
mega169	8M	0xE2	57.6k	-9	8250k	2	7864k	-2	8010k
mega329	8M	0xE2	38.4k	-2	8330k	7	7877k	4	8013k
mega649	8M	0xE2	38.4k	-2	8370k	8	7895k	6	7988k

Table 3.22. Possible OSCCAL_CORR selections for the ATmega649 family

3.9.11 RC-Generator check of the ATmega2560 family

For the ATmega2560 family an RC oscillator frequency of $8MHz$ can be selected. The selected frequency can be pre-divided with a factor 8 fuse bit, so that a $1MHz$ operation is also possible. The table 3.23 shows the results for the checked processors.

Typ	AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
				Corr	Freq	Corr	Freq	Corr	Freq
mega1281	8M	0xC2	38.4k	-5	8405k	5	7871k	2	8012k
mega2561	8M	0xC2	38.4k	-8	8363k	4	7870k	1	7990k

Table 3.23. Possible OSCCAL_CORR selections for the ATmega2560 family

Loading of more than 128Kbyte data was successfully tested with the ATmega2561. Normally the user data for the flash memory starts withh the address 0. This is not absolutely nessesary for data download via the serial interface. But the initial address must be below 128K (0x20000), so that the loading of data into the upper flash memory half works. The option VIRTUAL_BOOT_PARTITION can not be used by processors with more than 128Kbyte flash memory.

3.9.12 RC-Generator check of the ATtiny4313 family

The ATtiny4313 und the ATtiny2313 can select a RC-generator frequency of $8MHz$ and $4MHz$ with the Low-fuse. In addition a $128kHz$ generator can be selected as the clock, which otherwise supplies the watchdog timer. The table 3.24 shows the results of the frequeny measurement at $4MHz$ and $8MHz$ operation for the checked processors.

Typ	AVR_ FREQ	LFUSE	Baud-rate	Minimum Corr	Minimum Freq	Maximum Corr	Maximum Freq	Best Corr	Best Freq
tiny4313	8M	0xE4	38.4k	-4	8342k	0	7975k	-1	7983k
				-2	8326k	3	7905k	1	8010k
tiny2313	8M	0xE4	38.4k	-4	8400k	3	7909k	2	7980k
tiny4313	4M	0xE2	38.4k	-6	4193k	-3	3976k	-3	3976k
				1	4169k	6	3961k	5	4017k
tiny2313	4M	0xE2	38.4k	0	4160k	6	3960k	5	3998k

Table 3.24. Possible OSCCAL_CORR selections for the ATtiny4313 family

For all three checked ATtinys of this series the setting of the frequency was difficult because a small OSCCAL correction results to a relatively strong frequency change.

3.9.13 RC-Generator check of the ATtiny84 family

The ATtiny84 family can select the $128kHz$ clock of the watch dog additional to the $8MHz$ internal RC-generator as the main clock. But the $128kHz$ clock can not be adjusted. If you use this clock, you can only correct the generated baud rate by selecting another baud rate value or you can use the automatic baud rate selection. For a ATtiny24a I have checked the generated baud rate. Instead of the selected 2400 Baud I could measure only 2170 Baud. This results to a frequency error of about 9.6%, which is much too high for using it without a correction. If I select a baud rate of 2640 Baud for the optiboot, the download could operate with 2400 Baud. The measured clock frequency of the processor was $115.2kHz$ instead of the $128kHz$. The table 3.25 shows the results of the frequeny measurement at the $8Mhz$ operation for the checked processors.

Typ	AVR_ FREQ	LFUSE	Baud-rate	Minimum Corr	Minimum Freq	Maximum Corr	Maximum Freq	Best Corr	Best Freq
attiny84	8M	0xE2	19.2k	-6	8453k	14	7673k	5	8019k
attiny44a	8M	0xE2	19.2k	-16	8367k	3	7673k	-7	7984k
attiny24a	8M	0xE2	19.2k	-4	8388k	11	7685k	4	7992k

Table 3.25. Possible OSCCAL_CORR selections for the ATtiny84 family

3.9.14 RC-Generators check of the ATtiny85 family

The ATtiny84 family can select a $8MHz$ and a $6.4MHz$ RC-generator and a $128kHz$ clock of the watchdog circuit. The $6.4MHz$ RC-generator clock is allways scaled to $1.6Mhz$ for the processor clock. The $128kHz$ clock can not be calibrated. If you wish to use this clock, you can adjust the baud rate only by selecting a corrected baud rate value or you can use the automatic baud rate selection. The first table 3.26 shows the results of the frequeny measurement at the $8Mhz$ operation for the checked processors.

Typ	AVR_ FREQ	LFUSE	Baud- rate	Minimum		Maximum		Best	
				Corr	Freq	Corr	Freq	Corr	Freq
attiny85	8M	0xE2	38.4k	-4	8370k	9	7714k	3	8012k
attiny45	8M	0xE2	38.4k	-4	8400k	9	7706k	3	8030k
attiny25	8M	0xE2	38.4k	-9	8424k	46	7724k	40	8034k
attiny25	8M	0xE2	38.4k	-12	8399k	7	7680k	-2	7992k

Table 3.26. Possible OSCCAL_CORR selections for the ATtiny85 family at $8MHz$ operation

The setting values for the ATtiny25 look strange, but in the case of correction 3 the OSCCAL value has fallen below the number 128 and is therefore in a different setting range. Not before a correction value of 34 a frequency of $8364kHz$ was reached again, at which a operation of the serial interface was possible. A simular frequency could be selected with the correction value -6 in the other setting range. The next table 3.27 shows the results of the frequeny measurement at the $1.6Mhz$ operation for the checked processors. The RC-generator operates at $6.4Mhz$, but this frequency is allways divided ba factor 4 for the processor.

Typ	AVR_ FREQ	LFUSE	Baud- rate	Minimum		Maximum		Best	
				Corr	Freq	Corr	Freq	Corr	Freq
attiny85	1.6M	0xD3	9.6k	-7	1684k	9	1547k	2	1603k
attiny45	1.6M	0xD3	9.6k	-5	1684k	11	1559k	4	1603k
attiny25	1.6M	0xD3	9.6k	-7	1689k	10	1543k	3	1602k
attiny25	1.6M	0xD3	9.6k	-10	1680k	3	1550k	-3	1609k

Table 3.27. Possible OSCCAL_CORR selections for the ATtiny85 family at $1.6MHz$ operation

The operation with $1.6MHz$ clock frequency has not shown the anomaly of the OSCCAL setting for the ATtiny25. All checked examples can use the serial interface without any correction at this frequency. The ATtiny84 processor family can also use a PLL-oscillator, which is controlled with the internal $8MHz$ RC generator. The PLL-oscillator can operate at $64MHz$ or at $32MHz$ clock, which is typically used for the T1 counter. If you use the PLL-clock for the processor, you can only select the $64MHz$ operation and the clock is scaled by factor 4. So you will get a resulting $16MHz$ clock for the processor. The table 3.28 shows the measured results. As expected, these results do not differ significantly from the $8MHz$ results.

Typ	AVR_ FREQ	LFUSE	Baud- rate	Minimum		Maximum		Best	
				Corr	Freq	Corr	Freq	Corr	Freq
attiny85	16M	0xF1	38.4k	-4	16.87M	10	15.41M	4	16.02M
attiny45	16M	0xF1	38.4k	-4	16.87M	10	15.41M	4	15.95M
attiny25	16M	0xF1	38.4k	-9	16.91M	47	15.38M	41	16.03M
attiny25	16M	0xF1	38.4k	-11	16.82M	7	15.43M	-2	16.07M

Table 3.28. Possible OSCCAL_CORR selections for the ATtiny85 family at $16MHz$ operation

3.9.15 RC-Generators check of the ATtiny841 family

The ATtiny841 and the ATtiny441 can use also a internal $8MHz$ RC-generator, which can be adjusted. For the support of this family some special modifications must be done at the optiboot bootloader source. The table 3.29 shows the results of some checked examples.

Typ	AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
				Corr	Freq	Corr	Freq	Corr	Freq
attiny841	8M	0xE2	38.4k	-4	8369k	10	7861k	6	8003k
				-5	8389k	9	7874k	6	7990k
attiny441	8M	0xE2	38.4k	-4	8399k	10	7870k	7	7985k
				-4	8380k	9	7900k	7	7985k

Table 3.29. Possible OSCCAL_CORR selections for the ATtiny841 family at $8MHz$ operation

For all checked examples the serial interface can be used without the OSCCAL correction.

3.9.16 RC-Generators check of the ATtiny861 family

The ATtiny861 family can use a $8MHz$ internal RC generator, a PLL oscillator and the $128kHz$ clock of the watchdog circuit. The $128kHz$ clock of the watchdog circuit can not be calibrated und is therefore limited for use with the bootloader application. The PLL-Oscillator has a resulting frequency of $16MHz$ for the processor, which can only synchronized by the internal RC-generator. Therefore you can not use the PLL-oscillator with the T1 counter for precise time measurements. The first table 3.30 shows the OSCCAL correction results of the checked examples for the $8MHz$ operation.

Typ	AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
				Corr	Freq	Corr	Freq	Corr	Freq
attiny861	8M	0xE2	38.4k	-2	8415k	18	7693k	9	8007k
				-1	8436k	19	7678k	10	8011k
attiny461	8M	0xE2	38.4k	-2	8418k	17	7690k	9	7995k
				-4	8380k	14	7695k	5	8030k
attiny261	8M	0xE2	38.4k	-4	8403k	17	7710k	9	7986k

Table 3.30. Possible OSCCAL_CORR selections for the ATtiny861 family at $8MHz$ operation

For the ATtiny261 I have omitted the optional LED-flashing at the start of the optiboot to get enough space for the test program (option LED_START_FLASHES=0).

3.9.17 RC-Generators check of the ATtiny87 family

The ATtiny87 family can select a internal $8MHz$ RC-generator and a internal $128kHz$ generator as processor clock. You can also select a factor 8 scaler for the processor clock with the Low-fuse. The table 3.31 shows the calibration results of two examples of this family.

Typ	AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
				Corr	Freq	Corr	Freq	Corr	Freq
attiny87	8M	0xE2	57.6k	-1	8270k	7	7940k	3	8035k
attiny167	8M	0xE2	57.6k	-5	8227k	2	7839k	-1	8009k

Table 3.31. Possible OSCCAL_CORR selections for the ATtiny87 family

3.9.18 RC-Generators check of the ATtiny88 family

The ATtiny88 family can select a internal $8MHz$ RC-generator and a internal $128kHz$ generator as processor clock. You can also select a factor 8 scaler for the processor clock with the Low-fuse. The table 3.32 shows the calibration results of two examples of this family.

Typ	AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
				Corr	Freq	Corr	Freq	Corr	Freq
attiny88	8M	0xE2	38.4k	-4	8397k	15	7682k	6	8013k
attiny48	8M	0xE2	38.4k	-5	8385k	12	7739k	5	7995k

Table 3.32. Possible OSCCAL_CORR selections for the ATtiny88 family

Both examples can operate the serial interface without a frequency correction. But the processor clock will match the $8MHz$ better, if you choose a correction value of 5 (6).

3.9.19 RC-Generator check of the ATtiny1634

I have checked the $8MHz$ internal RC-generator of the ATtiny1634 with two examples. The ATtiny1634 support two additional calibration register for adjusting the temperature drift of the $8MHz$ RC-generator. In the table 3.33 I have not checked the effect of the temperature drift compensation. Additional to the $8MHz$ RC-generator the ATmega1634 can also adjust the internal $32kHz$ generator with a additional calibration register (OSCCAL1). This adjustment is currently unsupported by the Optiboot bootloader.

AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
			Corr	Freq	Corr	Freq	Corr	Freq
8M	0xE2	19.2k	-5	8404k	9	7867k	6	7983k
8M	0xE2	19.2k	-7	8410k	7	7867k	4	7986k

Table 3.33. Possible OSCCAL_CORR selections for the ATtiny1634

3.9.20 RC-Generators check of the AT90PWM family

Beside the usual $8MHz$ internal RC-generator the AT90PWM family can also select a PLL-oscillator with a resulting processor clock of $16MHz$. The PLL-oscillator is synchronized by the internal $8MHz$ RC-generator. The table 3.34 shows the OSCCAL correction results of two examples.

Typ	AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
				Corr	Freq	Corr	Freq	Corr	Freq
at90pwm2b	8M	0xE2	38.4k	-13	8350k	-1	7862k	-5	8020k
at90pwm3	8M	0xE2	38.4k	-10	8359k	4	7885k	1	7991k
at90pwm2b	16M	0xE3	38.4k	-14	16.74M	-1	15.74M	-4	15.97M
at90pwm3	16M	0xE3	38.4k	-10	16.79M	4	15.79M	2	15.97M

Table 3.34. Possible OSCCAL_CORR selections for the AT90PWM family

The AT90PWM2B can not use the serial interface without a OSCCAL correction.

3.9.21 RC-Generators check of the AT90CAN family

I have examined only one copy of a AT90CAN32 and AT90CAN132. Both AT90CAN examples can only select one internal 8 MHz RC-oscillator. All other choises require an external crystal or external clock generator. The clock frequency can optionally be divided by the factor 8, so that also a 1 MHz operation is possible with the internal clock.

Typ	AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
				Corr	Freq	Corr	Freq	Corr	Freq
at90can32	8M	0xE2	38.4k	0	8379k	6	7920k	5	8019k
at90can128	8M	0xE2	38.4k	0	8303k	3	7922k	2	8057k

Table 3.35. Possible OSCCAL_CORR selections for the AT90CAN family

3.10 Examination of the cooperation with USB-serial converters

3.10.1 Preface to the test environment

At first the environment for the series tests of the behavior of automatic baud rate adjustment are bad. The baud rate of the AVR-UART interface can be selected only with great steps for higher baud rates. Also the choise of baud rates for the different USB-serial chips is limited by the chip and the capability of the driver. With the AVR processors it is possible to detune the clock frequency of the local RC generator with OSCCAL, but the result of the detuning vary with each particular model. The resulting oscillating frequency of the local RC generator can be measured, but the step size of the detuning cannot be changed to get a specific frequency exactly. I know only one way out of this difficulty is to use a external clock source for the AVR processor. For this reason I got the relatively inexpensive Kkmoon FY6900 function generator, which can generate almost any waveform with frequencies up to 60 MHz. Of course, this function generator can also generate square-wave signals, which can serve as a clock for the AVR. The function generator can be operated both via the front panel and via a USB interface. The USB interface consists of an internal CH341 USB-serial converter, so that no special USB driver is required. The control can be done with well-documented ASCII character strings. iThis fulfills the essential requirements for systematic tests of the interaction between **avrdude**, the USB-serial converter and the optiboot bootloader. In particular, the behavior of the automatic baud rate adjustment should be examined more intensively. For this, downloads with **avrdude** should be repeated with different clock frequencies, so that problems are detected. I choosed a number of 40 repititions as a acceptable compromise between expenditure of time and statistical significance. Since 41 different frequencies are to be examined with the automatic baud rate adjustment, there are already 1640 write processes for one variant. With more than 10 of these variants, the guaranteed number of cycles of the AVR flash memory would be exceeded. Since I didn't want to examine the cycle stability of the AVR flash memory, I provided a write protection pin in the optiboot software for this. This additional function of optiboot is only useful for these examinations, therefore this option (WRITE_PROTECT_PIN) is not further documented.

3.10.2 Autobaud measurements with different USB-Serial chips

The data for the results presented here were generated with the bash script file test_download in the subdirectory ee_test and then combined in .tab files in the directory Doku/results with an editor. In the course of the investigations, an improvement in the baud rate measurement with the divided clock frequency could be built into optiboot. The improvement consists of resetting the prescaler for the AVR's counters. The other source of error for the time measurement is caused by the query loop for the RX data pin and results to at least 3 clocks for each measuring point. But first I will show the count of errors with different USB-serial chips with 40 downloads for each clock frequency between 14 MHz and 18 MHz with a step size of 100 kHz and a baud rate of 115200. A total of 1640 downloads of 10 kByte with **avrdude** was the base for each table entry. A special modified **avrdude** program was used for these tests, which can select a serial output with two stop bits with the new parameter -S. The differences of the **avrdude** sources are notices in the file ee_test/avrdude.diff.

Baud rate	FT232 RL	FT232 Clone	CH340	CP2102	CP2104	PL2303 TA	Pololu AVR	Arduino UNO
22	0	1	40	0	0	0	264	0
26	0	0	69	0	0	0	250	0
62	150	181	186	95	155	145	521	129
32	0	0	72	0	0	0	217	0
36	0	0	12	0	0	0	261	0
72	10	44	24	4	4	3	427	0
76	1	0	2	0	0	0	331	0
12	0	0	7	0	0	0	281	0
82	0	0	7	0	0	0	285	0

Table 3.36. Error rate of 1640 downloads with 10 kByte each using the HW-UART

As a reminder, the key for the Optiboot auto baud rate setting should be mentioned here. Settings 22, 26 and 62 all select a 2-bit time measurement. With settings 22 and 62 the counter, which is used for the time measuring, works with /8 frequency prescaler, only with setting 26 the counter runs at full clock rate.

Correspondingly, the settings 32, 36, 72 and 76 select a 4-bit time measurement. With settings 32 and 72 a counter with /8 frequency prescaler is used, settings 36 and 76 use a counter without prescaler for the time measurement. The other two settings 12 and 82 use 9 bits for time measurement, both with the /8 prescaler for the counter.

The other two settings 12 and 82 use 9 bits for time measurement, both with the /8 prescaler for the counter.

The USB-serial converter FT232 appears twice in the table 3.36, once as FT232RL,s the original chip from Future Technology Devices International Limited (FTDI) and once as FT232clone, a Chinese replica of unknown origin with the serial number A50285BI. The replicas usually have the same serial number. The original chips all have a different serial number. From the outside, the replicated chips are difficult to identify because the manufacturer's logo FTDI was copied also. The examined Chinese replica achieved similarly good test results as the original chips.

You can see in the table 3.36, that the number of errors in the autobaud settings 62, 72 and 82 are noticeably high. The reason is that the counter with the /8 prescaler for the additional checks is already started with the start bit of the serial receive data and the state is further only read for the time measurement.

As a result, the condition of the prescaler during the actual time measurement is undefined. With settings 22 and 32, the counter is started with a prescaler reset with the first data bit used for measurement. As a result, the time measurement gains measurement reliability. Settings 26, 36 and 76 use the counter without a prescaler and therefore the problem with the undefined prescaler does not exist here. I would like to mention here that earlier versions of the optiboot autobaud measurement with settings 22 and 32 would result to simular high error rates as the settings 62 and 72, because the prescaler was never reset. With normal use, you can expect fewer problems with autobaud than with this tests, because the clock frequency of the AVR is better suited to the desired baud rate. The tests here are designed to show the limits of the automatic baud rate adjustment.

Two USB-serial converters stand out in the table 3.36 with a higher error rate, the CH340 and the Pololu AVR. In order to detect the reason, further measurements were made with a fixed UBBR divider, which are shown in the next subsection.

3.10.3 Measurement of USB-serial converters mit fixed optiboot baud rate

For this tests, the optiboot baud rate was fixed to 115200 at a clock rate of 16 MHz. The USB-serial converters speed is always set to 115200 baud. The clock frequency of the ATmega has been increased from 14.7 MHz in 100 kHz steps to 17.1 MHz instead of the fixed 16 MHz. The optiboot baud rate changes proportionally with the change in frequency. For the tests downloads of 10 kByte each with **avrdude** are repested 40 times for each frequency. The number of unsuccessful attempts for each frequency are recorded in a table from the script file test_download. From the tables obtained in this way for the various USB-serial converters, an editor was used to create an overall table for the test series with 1 stop bit (Doku/results/HWfix_1_115200_USBserial_0.tab) and for the test series with 2 stop bits (Doku/results/HWfix_1_115200_USBserial_1.tab). The table 3.37 was then created from these tables.

	115200 Baud, 2 Stop Bits			115200 Baud, 1 Stop Bit		
USB-Seriell Typ	Mindest-Frequenz	Maximal-Frequenz	Frequenz-Bereich	Mindest-Frequenz	Maximal-Frequenz	Frequenz-Bereich
FT232RL	15.1 MHz	16.5 MHz	9.3%	15.4 MHz	16.5 MHz	7.9%
FT232clone	15.2 MHz	16.5 MHz	8.7%	15.5 MHz	16.5 MHz	7.2%
CH340G	15.1 MHz	16.1 MHz	6.8%	15.4 MHz	16.1 MHz	5.0%
CP2102	15.0 MHz	16.4 MHz	9.3%	15.3 MHz	16.4 MHz	7.5%
CP2104	15.1 MHz	16.6 MHz	10%	15.3 MHz	16.6 MHz	8.6%
PL2303TA	15.1 MHz	16.6 MHz	10%	15.1 MHz	16.6 MHz	10%
PololuAVR	15.4 MHz	16.5 MHz	7.5%	15.4 MHz	16.5 MHz	7.5%
ArduinoUNO	15.4 MHz	16.6 MHz	8.1%	15.7 MHz	16.6 MHz	6.2%

Table 3.37. Usable clock frequency range with Hardware-UART

A permissible deviation of the baud rate of ±4% was expected with this tests, resulting to a total area of 8%. Only two USB-serial converters, the CP2104 and the PL2303TA, match to the expected 8% area with the normal 1 stop bit operation. With 2 stop bits selected for the serial interface, the FT232 converters, the CP2102, CP2104 and PL2303TA converters as well as the Arduino UNO reach the expected 8% range. The CH340G converter significantly improves the usable area to 6.8%, but remains below 8%. The PololuAVR and the PL2303TA shows no reaction to increasing the number of stop bits from 1 to 2. But the time measurement with the Linux command "time" of the output of more than 10000 characters results to about 10% more time usage with the selection of 2 stop bits instead of 1 stop bit. The PololuAVR does not show any significant time change with this test. So I assume, that the interaction between driver and chip does not work with the PololuAVR.

In any case, the possible cause for the poor performance of the CH340G chip and the PololuAVR in table 3.36 on page 52 is found. With the PololuAVR you should consider that it is designed as an ISP programmer and the USB-serial converter is only an additional function.
The clear front runners in this special comparative test are the CP2104 from Silicon Laboratories Inc. and the PL2303TA from Prolific Technology Inc. , which also achieved better values in this test than the FT232RL from Future Technology Devices International. According to the data sheet, the FT232RL supports a maximum baud rate of 3 Mbaud, the PL2303TA even 6 Mbaud, which neither the CP2102 (1 Mbaud) nor the test winner CP2104 (2 Mbaud) can achieve according to the data sheet.

In addition, I also examined the work area of the USB-serial converter in conjunction with the software UART solution from optiboot. The software uses about 1.5 stop bits when sending data, the USB-serial converters use either 1 (Doku/results/SWfix_1_115200_USBserial_0.tab) or 2 stop bits (Doku/results/SWfix_1_115200_USBserial_0.tab). The results are summarized in the table 3.38.

USB-Seriell Typ	115200 Baud, 2 Stop Bits			115200 Baud, 1 Stop Bit		
	Minimum-Frequency	Maximum-Frequency	Frequency-area	Minimum-Frequency	Maximum-Frequency	Frequency-area
FT232RL	15.3 MHz	16.8 MHz	9.8%	15.5 MHz	16.8 MHz	8.7%
FT232clone	15.4 MHz	16.9 MHz	9.8%	15.55 MHz	16.9 MHz	9.0%
CH340G	15.2 MHz	16.5 MHz	8.5%	15.5 MHz	16.5 MHz	6.8%
CP2102	15.2 MHz	16.7 MHz	10%	15.5 MHz	16.8 MHz	8.7%
CP2104	15.2 MHz	16.9 MHz	11.2%	15.3 MHz	16.9 MHz	10.6%
PL2303TA	15.3 MHz	16.9 MHz	10.6%	15.4 MHz	16.9 MHz	10%
PololuAVR	15.5 MHz	16.9 MHz	9.3%	15.5 MHz	16.9 MHz	9.3%
ArduinoUNO	15.4 MHz	16.9 MHz	10%	15.8 MHz	16.9 MHz	7.5%

Table 3.38. Usable clock frequency range with Software-UART

All measurement series with the software UART solution show a larger frequency range without errors compared to hardware UART operation from the table 3.37. This is probably due to the fact that the software solution only scans the input signal per bit once and tries to place this reading in the middle of the transmission time expected with the baud rate. The hardware UART solution of the AVR processors use 3 samples per data bit. In the "Double Speed Mode" used here, scan numbers 4, 5 and 6 of the 8 scans are used for each data bit or stop bit. The three scans before (1-3) and the two after (7-8) are not used. At least 2 samples of the 3 used readings must result in a 1 for the stop bit sampling, otherwise a "framing" error is generated. This oversampling process copes better with disturbed transmission, but allows less tolerance in the baud rate.

I have not found any more detailed information about the type of scanning of the serial data of any USB-serial converter. But I assume that they all use a form of oversampling similar to the AVR-UART interface. However, the scanning method of the individual USB-serial converters must differ. There is no other way to explain the different test results because the optiboot side was the same for all test candidates.

3.10.4 Extreme test with 115200 Baud at 8 MHz clock

The wide baud rate range for a functioning serial communication of some USB-serial converters has encouraged me to investigate the behavior of the autobaud function in practice for a baud rate of 115200 baud at a clock rate of 8 MHz. For this I have chosen the USB-serial converter with the greatest baud rate tolerance, the CP2104. I provided the ATmega1281 used for the tests with an optiboot bootloader with Autobaud mode 32 without the LED flashing. In the Autobaud group 30-39, 4 data bits are used to measure the baud rate. With this setting 40 downloads of 10 kByte with avdrude for 41 frequencies from 7 MHz to 9 MHz in 50 kHz steps were done. To evaluate the result, the same series of measurements was repeated with a fixed baud rate of optiboot. The baud setting to 111111 generates an UBRR setting of the hardware UART to 8, with the selection 125 kBaud a UBRR setting of 7 is used. Of course the actual baud rate changes proportionally with the frequency change.

Frequency	Type Mode Period	CP2104 HW32 auto	HW111111 UBRR=8	HW125000 UBRR=7	SW32 auto	SW34 auto	CH340G HW32 auto
7000000		40	40	40	0	0	40
7050000		40	40	40	0	0	40
7100000		0	40	0	0	0	0
7150000		0	40	0	4	0	0
7200000		0	40	0	0	0	0
7250000		0	40	0	0	0	0
7300000		0	40	0	0	0	0
7350000		0	40	0	0	0	0
7400000		0	40	0	0	0	0
7450000		0	40	0	0	0	0
7500000		0	40	0	0	0	0
7550000		0	40	0	0	0	0
7600000		0	40	0	0	7	40
7650000		0	40	0	0	40	40
7700000		0	40	0	0	40	40
7750000		0	40	0	0	0	40
7800000		16	40	0	0	0	40
7850000		40	40	40	0	0	40
7900000		40	40	40	0	0	40
7950000		40	40	40	0	0	40
8000000		0	0	40	0	0	0
8050000		0	0	40	0	0	0
8100000		0	0	40	0	0	0
8150000		0	0	40	0	0	0
8200000		0	0	40	0	0	0
8250000		0	0	40	0	0	0
8300000		0	0	40	0	0	0
8350000		0	0	40	0	0	0
8400000		0	0	40	0	0	0
8450000		0	0	40	0	0	0
8500000		0	0	40	0	24	0
8550000		0	0	40	0	40	40
8600000		0	0	40	0	0	40
8650000		0	0	40	0	0	40
8700000		0	0	40	0	0	40
8750000		15	0	40	0	0	40
8800000		40	40	40	0	0	40
8850000		0	40	40	0	0	0
8900000		0	40	40	0	0	0
8950000		0	40	40	0	0	0
9000000		0	40	40	0	0	0

Table 3.39. Error rate of the extreme tests with 115200 Baud and 8 MHz clock

All test results in the table 3.39 represent the number of failed attempts at the respective clock rate (column 1) of 40 download attempts of 10 kByte with **avrdude**. The USB-serial converters always use 2 stop bits for these tests. The unsuccessful attempts for a CP2104 USB-serial converter are recorded in the first data column. The PL2303TA chip has shown the same result for this test. As you can see there are three total failures between 7850 kHz and 7950 kHz, none of the 40 downloads worked. The explanation is provided in the next two columns. The 40 downloads were attempted for the entire frequency range with the constant UART divider setting 9 (UBRR = 8) and 8 (UBRR = 7). Neither of the two settings works for the frequencies 7850 kHz to 7950 kHz. However, no further setting can be selected between UBRR=7 and UBRR=8. It is therefore clear why no Autobaud variant can work here. With these preconditions it is simply not possible to set the desired baud rate of 115200 with these clock frequencies with sufficient accuracy. The partial failure of the download at 7800 kHZ with the Autobaud variant 32 is probably due to the uncertainty of the baud rate measurement.

In the fourth data column, the download was attempted with the same settings with the software UART solution from optiboot. The table 3.38 on page 54 with the software UART showed a larger Frequency working range compared to the CP2104 hardware UART tests in table 3.37 on page 53. In addition, the delay time of the software UART is formed with waiting loops that allow a setting of multiples of 6 tics. Due to the pre-divider of the hardware UART, the UBRR setting results to delay times with multiples of 8 clock tics. Both circumstances lead to the fact that with the CP2104 the software solution shows only 4 unsuccessful download attempts in the entire frequency range at the clock frequency 7150 kHz, a really amazing result! Above a clock rate of 7200 kHz, the baud rate 115200 can be used without restriction with this CP2104 USB-serial converter, when the 2 stop bit operation and software UART of the optiboot with BAUD_RATE=32 is used.

As a further test with the software UART solution from optiboot, I tested the BAUD_RATE=34 setting. With this setting, the delay loop is lengthened by one clock, so that only a multiple of 8 clocks can be selected here, just like the hardware UART can do. This option only exist, because it result to a slightly shorter optiboot program. Although this software UART setting has exactly the same multiple of 8 clocks for the period as the hardware UART, the result is slightly better here. This is probably due to the already mentioned larger frequency working range of the CP2104 converter with optiboot in software UART operation. In addition, the critical area here is not as close to the selected working frequency of 8 MHz as with the hardware UART.

In the 6th and last data column I tested the same measurement as in the first data column with the CH340G converter. You see significantly more frequencies with total failures of the download attempts here, what the investigation in table 3.37 on page 53 expected.

3.10.5 Extreme test with 230400 Baud at 16 MHz clock rate

After the preliminary investigations at 8 MHz clock and 115200 baud, I also examined all available USB-serial converters with 230400 baud and an autobaud setting 32 of the AVR. The baud rate is calculated from the transmission time for 4 data bits. A counter with / 8 prescaler is used to measure the time. The clock of the AVR was starting in steps of 100 kHz at 14 MHz increased to 18 MHz. 40 downloads of 10 kByte each with the modified **avrdude** (2 stop bits) are tried for each frequency. The table 3.40 shows the results for the existing USB-serial converters.

Typ Frequenz	FT232 RL	FT232 clone	CH340G	CP2102	CP2104	PL2303 TA	Pololu AVR	Arduino UNO
14000000	40	40	40	40	40	40	40	0
14100000	40	40	40	40	40	40	40	0
14200000	4	40	0	0	0	0	40	0
14300000	0	0	0	0	0	0	40	0
14400000	0	0	0	0	0	0	40	0
14500000	0	0	1	0	0	0	0	0
14600000	0	0	0	0	0	0	0	0
14700000	0	0	0	0	0	0	0	0
14800000	0	0	0	0	0	0	0	0
14900000	0	0	0	0	0	0	0	40
15000000	0	0	0	0	0	0	0	40
15100000	0	0	0	0	0	0	0	40
15200000	0	0	40	0	0	0	0	40
15300000	0	0	40	0	0	0	0	40
15400000	0	0	40	0	0	0	0	0
15500000	0	0	40	40	0	0	0	0
15600000	35	23	40	40	19	15	40	0
15700000	40	40	40	40	40	40	40	0
15800000	40	40	40	40	40	40	40	0
15900000	40	40	40	0	40	40	40	0
16000000	0	29	0	0	0	0	40	0
16100000	0	0	0	0	1	0	40	0
16200000	0	0	0	0	0	0	40	0
16300000	0	0	0	0	0	0	0	0
16400000	0	0	0	0	0	0	0	0
16500000	0	0	0	0	0	0	0	0
16600000	0	0	0	0	0	0	0	0
16700000	0	0	0	0	0	0	0	40
16800000	0	0	0	0	0	0	0	40
16900000	0	0	0	0	0	0	0	40
17000000	0	0	0	0	0	0	0	40
17100000	0	0	40	0	0	0	0	0
17200000	0	0	40	0	0	0	0	0
17300000	0	0	40	0	0	0	0	0
17400000	0	0	40	8	0	0	0	0
17500000	18	4	40	40	18	20	17	0
17600000	40	40	40	40	40	40	40	0
17700000	40	40	0	0	0	0	40	0
17800000	0	6	0	0	0	0	40	0
17900000	0	0	0	0	0	0	40	0
18000000	0	0	0	0	0	0	40	0
Total:	337	382	641	328	278	275	697	360

Table 3.40. Error rate of the extreme test with 230400 Baud at 16 MHz Clock

Again the CP2104 and the PL2303 show the fewest failures in the table ref tab: ExtremErrs16 over the entire frequency range. The serial output of the USB-serial converter was operated with two stop bits in this investigation and the optiboot bootloader worked with the hardware UART. If the software UART from optiboot had been used, the result would have been even better. Unfortunately, apart from the Arduino UNO with the ATmega16U2 as a USB-serial converter, all other USB-serial converters show failures just below the clock rate 16 MHz. The FT232clone itself has problems with the download even at 16 MHz, with the Pololu an entire frequency range around 16 MHz cannot be used. Almost all USB-serial converters try to adhere to the baud rate of 230400 as precisely as possible, which works quite well because of the higher internal clock rate of around 48 MHz to 96 MHz. In contrast, the ATmega16U2 uses a baud rate of 222222, the same as generated by the ATmega1281 used for the tests at 16 MHz. This is the reason why the Mega16U2 installed on the Arduino UNO shows no problems with the download to the ATmega1281 in the clock frequency range around 16 MHz.

If the existing USB-serial converter supports operation with 250000 baud, this setting is probably more favorable with an AVR clock rate of 16 MHz, since this baud rate can be generated without any deviation. With an optiboot with the autobaud function, you can simply try out if the higher baud rate is possible.

3.10.6 Summary and recommendations

- USB-Serial converters
  The USB-serial converter FT232RL, CP2102, CP2104 and also the ATMega16U2 used by an Arduino UNO board variant has shown good results with the tests. For a new purchase, I would recommend the CP2104, which operates with the largest baud rate range at the 115200 baud setting in the tests. With the PL2303 module I had difficulties during the tests, the Linux driver reported errors and the results of the tests were inconsistent. I also had to retrofit a DTR output pin because this pin was missing on my module. Only after I swapped the PL2303HX chip for a PL2303TA did the results improve. The GND (Pin 7) of the PL2303TA chip also had to be rewired on the board. After refitting the PL2303 modul I get the same good results as CP2104 did with the PL2303TA. Actually I noticed some Chinese offers of a Arduino Nano clone build with the PL2303TA chip instead of the usual CH340G.

- 2 stop bits
  The extensive tests, which are summarized in the tables 3.37 on page 53 and 3.38 on page 54, showed that the use of a second stop bit for the transmit data increases the usable baud rate difference significantly. This shows that the occurrence of unfavorable baud rate combinations with 2 stop bits is reduced. Unfortunately the standard **avrdude** with version 6.3 does not allow the use of a second stop bit. You can try to patch the **avrdude** version 6.3 sources with the diff documentation in the file ee_test/avrdude.diff. The loss of speed is insignificant, you gain more speed, if you can select a higher baud rate instead.

- Selection of a automatic baud mode
  With the last improvement of the autobaud function, the simplest measurement of the baud rate with the setting BAUD_RATE=22 can also be used with good results. This is interesting because this version uses the least amount of memory. Unfortunately, one more assembler command is required for this useful improvement (prescaler reset), so that 2 bytes of flash are more used for all autobaud modes with clock prescaler for the counter (One-digit of the baud rate <5). The memory consumption is nevertheless lower compared to using the counter at full clock rate (One-digit of the baud rate >4). **I recommend measuring the baud rate over 4 data bits (mode 32).** The additional control function of mode 52 can be omitted. The use of full control of the data byte with baud rate >59 is not recommended, especially when using the counter with prescaler, the baud rate measurements are significantly worse. The measurement

of the baud rate over 9 bits (BAUD_RATE=12) can be used without restrictions, but the measurement over 4 bits is completely sufficient and requires less memory.

- Usage of software UART
  There are several reasons that make the use of the software UART solution necessary, such as using other pins for serial communication or one-pin communication (same selection for RX and TX pin). The autobaud function can also be used in this case. Because both the software UART solution and the autobaud function require more flash memory, it is becoming more and more difficult to fit the optiboot in 512 bytes flash. Of course you don't have to worry about the 512 byte limit, if the target processor has a larger bootloader page such as 1024 bytes. You should try the configurations of optiboot to fit in 512 byte flash with **make** calls without the ISP option or with the ISP option set to 0. You can try out the different options until you find a suitable configuration that fits in 512 bytes. By the way, the size of the bootloader page of the target processor is displayed in the terminal log with every **make** call that specifies a target processor.

  First you should switch off the blinking function of the LED with LED_START_FLASHES=0 in order to save space. If you absolutely need something to light up, you can use the LED_DATA_FLASI function, because it only needs 4 bytes of flash instead of the 56 bytes of the flashing function. The table 3.8 on page 37 gives advance information about the memory required for all setting options.
  With the option NO_EARLY_PAGE_ERASE=1 additional 14 bytes can be saved, whereby this only makes the loading of the user program a little slower.
  You should leave the possibility of writing data to the EEprom enabled, if possible. Switching off the function is also possible with SUPPORT_EEPROM=0, but saves only about 24 bytes.

  If more interference is to be expected in the transmission, for example because longer cables are used, the hardware UART should be used if possible and higher baud rates should be avoided.

3.11 Final remark

When I started working on converting optiboot to assembler and installing EEprom support, I had no idea how much could be examined and expanded here. The first task was to build an optiboot that supports loading EEprom data and fit to 512 bytes of flash. That seemed a manageable effort to me.

But there was also the support of many AVR processors to be built in, which had to be purchased first. The different bootloader page sizes and flash page sizes had to be taken into account. Of course everything also had to be tested too. Many processors with many options results to many program variants with different program lengths, which are now always automatically taken into account. This avoids incorrect setting of the AVR fuses for all supported processors.
The software UART solution has been optimized so that it works well and can now also operate with one IO pin. The automatic baud rate adjustment (autobaud) was installed and tested. Many autobaud variants can now be selected, whereby the autobaud function works with both, the hardware UART and the software UART solution of optiboot.

The possibility of adjusting the internal RC oscillator as a clock for the supported AVRs was examined and the setting option OSCCAL_CORR was integrated in optiboot.

The 512 byte limit for the flash is a particular challenge when combining autobaud and software UART, even if you deselect the LED flashing function. Of course, optiboot also runs if more than 512 bytes are required. The limit was only my goal, I wanted to integrate most functions into the 512 bytes. With the option NO_EARLY_PAGE_ERASE a way was found to save a few bytes if necessary. This makes the flash writing a little slower, but if, for example, a higher baud rate can be selected in the autobaud mode, this is more than compensated. The advantage of the autobaud function is that you can try out the higher baud rate such as 115200 baud at 8 MHz. If that doesn't work, you can switch to a lower baud rate without changing the bootloader.

So I was able to prove in extensive test series that a larger deviation of the baud rate is tolerated from **avrdude** with the use of a second stop bit for the transmitted serial data. It is a little surprising that this has not been noticed so far. Otherwise the second stop bit should at least be selectable as an option from **avrdude**.

Even when I thought there was nothing more to improve, I got the idea to reset the frequency prescaler for counters during the final tests of the autobaud function with various USB-to-serial converters. This is rarely used because the prescaler is shared by all integrated counters. But here in optiboot this function can be used and leads to a reduction in autobaud time measure errors if the counter is used with /8 prescaler.

I am sorry that the many setting options for optiboot are confusing for a user. As far as possible, I have taken over the options of the C version from Peter Knight ands Bill Westfield to make it easier for users of the original version of optiboot. Otherwise I hope that the examples from chapter 3.8 on page 39 facilitate the first introduction. In addition, a **make** call with the desired target processor as a parameter is sufficient to generate an executable optiboot as a .hex file. The terminal log shows which setting has been selected. That is always a matching clock frequency and a (fixed) baud rate preselected for this specific processor.
If necessary, the presettings can of course be changed with options.

Chapter 4

Data of the AVR 8-Bit Microcontrollers

4.1 Signature Bytes and default Fuse setting

The following tables show the content of the signature bytes and the default Fuse setting for different AVR families.

	Signature			Default Fuses			Default
Typ	1	2	3	Low	High	Extended	Clock
ATtiny11	0x1E	0x90	0x05	0x52	-	-	1MHz RC
ATtiny102	0x1E	0x90	0x0C	0xFF	-	-	8/8MHz RC
ATtiny104	0x1E	0x90	0x0B	0xFF	-	-	8/8MHz RC
ATtiny12	0x1E	0x90	0x05	0x52	-	-	1.2MHz RC
ATtiny13	0x1E	0x90	0x07	0x6A	0xFF	-	9.6/8MHz RC
ATtiny15	0x1E	0x90	0x06	0x50	-	-	1.6MHz RC
ATtiny22	0x1E	0x91	0x06	0xBE?	-	-	1MHz RC
ATtiny2313A	0x1E	0x91	0x0A	0x64	0xDF	0xFF	8/8MHz RC
ATtiny24	0x1E	0x91	0x0B	0x62	0xDF	0xFF	8/8MHz RC
ATtiny25	0x1E	0x91	0x08	0x62	0xDF	0xFF	8/8MHz RC
ATtiny26	0x1E	0x91	0x09	0xE1	0xF7	-	8/8MHz RC
ATtiny261A	0x1E	0x91	0x0C	0x62	0xDF	0xFF	8/8MHz RC
ATtiny28	0x1E	0x91	0x07	0xFF	-	-	1.2MHz RC
ATtiny4313	0x1E	0x92	0x0D	0x62	0xDF	0xFF	8/8MHz RC
ATtiny44	0x1E	0x92	0x07	0x62	0xDF	0xFF	8/8MHz RC
ATtiny441	0x1E	0x92	0x15	0x62	0xDF	0xFF	8/8MHz RC
ATtiny461A	0x1E	0x92	0x08	0x62	0xDF	0xFF	8/8MHz RC
ATtiny45	0x1E	0x92	0x06	0x62	0xDF	0xFF	8/8MHz RC
ATtiny48	0x1E	0x92	0x09	0x6E	0xDF	0xFF	8/8MHz RC
ATtiny84	0x1E	0x93	0x0C	0x62	0xDF	0xFF	8/8MHz RC
ATtiny841	0x1E	0x93	0x15	0x62	0xDF	0xFF	8/8MHz RC
ATtiny861A	0x1E	0x93	0x0D	0x62	0xDF	0xFF	8/8MHz RC
ATtiny85	0x1E	0x93	0x0B	0x62	0xDF	0xFF	8/8MHz RC
ATtiny87	0x1E	0x93	0x87	0x62	0xDF	0xFF	8/8MHz RC
ATtiny88	0x1E	0x93	0x11	0x6E	0xDF	0xFF	8/8MHz RC
ATtiny167	0x1E	0x94	0x87	0x62	0xDF	0xFF	8/8MHz RC
ATtiny1634	0x1E	0x94	0x12	0x62	0xDF	0xFF	8/8MHz RC

Table 4.1. Signature Bytes of ATtiny processors and default fuses

Typ	Signature 1	Signature 2	Signature 3	Default Fuses Low	Default Fuses High	Default Fuses Extended	Default Clock
ATmega103	0x1E	0x97	0x01	0xDF?	-	-	Crystal
ATmega128	0x1E	0x97	0x02	0xC1	0x99	0xFD	1MHz RC
ATmega48A	0x1E	0x92	0x05	0x62	0xDF	0xFF	8/8MHz RC
ATmega48PA	0x1E	0x92	0x0A	0x62	0xDF	0xFF	8/8MHz RC
ATmega8	0x1E	0x93	0x07	0xE1	0xD9	-	1MHz RC
ATmega88A	0x1E	0x93	0x0A	0x62	0xDF	0xF9	8/8MHz RC
ATmega88PA	0x1E	0x93	0x0F	0x62	0xDF	0xF9	8/8MHz RC
ATmega8515	0x1E	0x93	0x06	0xC1	0xD9	-	1MHz RC
ATmega8535	0x1E	0x93	0x08	0xC1	0xD9	-	1MHz RC
ATmega16	0x1E	0x94	0x03	0xE1	0x99	-	1MHz RC
ATmega161	0x1E	0x94	0x01	0xDA	-	-	Crystal
ATmega162	0x1E	0x94	0x04	0x62	0x99	0xFF	8/8MHz RC
ATmega163	0x1E	0x94	0x02	0xF2	0xF9	-	8/8MHz RC
ATmega164A	0x1E	0x94	0x0A	0x42	0x99	0xFF	8/8MHz RC
ATmega165A	0x1E	0x94	0x10	0x62	0x99	0xFF	8/8MHz RC
ATmega165PA	0x1E	0x94	0x07	0x62	0x99	0xFF	8/8MHz RC
ATmega168A	0x1E	0x94	0x06	0x62	0xDF	0xF9	8/8MHz RC
ATmega168PA	0x1E	0x94	0x0B	0x62	0xDF	0xF9	8/8MHz RC
ATmega169	0x1E	0x94	0x05	0x62	0x99	0xFF	8/8MHz RC
ATmega32	0x1E	0x95	0x02	0xE1	0x99	-	1MHz RC
ATmega323	0x1E	0x95	0x01	0xE1	0x99	-	1MHz RC
ATmega324A	0x1E	0x95	0x08	0x42	0x99	0xFF	8/8MHz RC
ATmega325A	0x1E	0x95	0x05	0x62	0x99	0xFF	8/8MHz RC
ATmega325PA	0x1E	0x95	0x0D	0x62	0x99	0xFF	8/8MHz RC
ATmega3250A	0x1E	0x95	0x06	0x62	0x99	0xFF	8/8MHz RC
ATmega3250PA	0x1E	0x95	0x0E	0x62	0x99	0xFF	8/8MHz RC
ATmega328	0x1E	0x95	0x14	0x62	0xD9	0xFF	8/8MHz RC
ATmega328P	0x1E	0x95	0x0F	0x62	0xD9	0xFF	8/8MHz RC
ATmega329	0x1E	0x95	0x03	0x62	0x99	0xFF	8/8MHz RC
ATmega3290	0x1E	0x95	0x04	0x62	0x99	0xFF	8/8MHz RC
ATmega64	0x1E	0x96	0x02	0xC1	0x99	0xFF	1MHz RC
ATmega640	0x1E	0x96	0x08	0x42	0x99	0xFF	8/8MHz RC
ATmega644A	0x1E	0x96	0x0A	0x62	0x99	0xFF	8/8MHz RC
ATmega645A	0x1E	0x96	0x05	0x62	0x99	0xFF	8/8MHz RC
ATmega645P	0x1E	0x96	0x0D	0x62	0x99	0xFF	8/8MHz RC
ATmega6450A	0x1E	0x96	0x06	0x62	0x99	0xFF	8/8MHz RC
ATmega6450P	0x1E	0x96	0x0E	0x62	0x99	0xFF	8/8MHz RC
ATmega649	0x1E	0x96	0x03	0x62	0x99	0xFF	8/8MHz RC
ATmega6490	0x1E	0x96	0x04	0x62	0x99	0xFF	8/8MHz RC
ATmega1280	0x1E	0x97	0x03	0x42	0x99	0xFF	8/8MHz RC
ATmega1281	0x1E	0x97	0x04	0x42	0x99	0xFF	8/8MHz RC
ATmega1284	0x1E	0x97	0x05	0x62	0x99	0xFF	8/8MHz RC
ATmega2560	0x1E	0x98	0x01	0x42	0x99	0xFF	8/8MHz RC
ATmega2561	0x1E	0x98	0x02	0x42	0x99	0xFF	8/8MHz RC

Table 4.2. Signature Bytes of ATmega Processors and default fuses

Typ	Signature 1	2	3	Default Fuses Low	High	Extended	Default Clock
AT90S1200	0x1E	0x90	0x01	0xFF	-	-	Crystal
AT90S2313	0x1E	0x91	0x01	0x64	0xDF	0xFF	Crystal
AT90S2333	0x1E	0x91	0x05	0xDA			Crystal
AT90S4414	0x1E	0x92	0x01	0xF9	-	-	Crystal
AT90S4433	0x1E	0x92	0x03	0xDA	-	-	Crystal
AT90S4434	0x1E	0x92	0x02	0xF9	-	-	Crystal
AT90S8515	0x1E	0x93	0x01	0xF9	-	-	Crystal
AT90S8535	0x1E	0x93	0x03	0xF9	-	-	Crystal
AT90PWM2 AT90PWM3	0x1E	0x93	0x81	0x62	0xDF	0xF9	8/8MHz RC
AT90PWM2B AT90PWM3B	0x1E	0x93	0x83	0x41	0xDF	0xF9	8/8MHz RC
AT90CAN32	0x1E	0x95	0x81	0x62	0x99	0xFF	8/8MHz RC
AT90CAN64	0x1E	0x96	0x81	0x62	0x99	0xFF	8/8MHz RC
AT90CAN128	0x1E	0x97	0x81	0x62	0x99	0xFF	8/8MHz RC
ATmega8U2	0x1E	0x93	0x89	0x5E	0xD9	0xF4	Crystal
ATmega16U2	0x1E	0x94	0x89	0x5E	0xD9	0xF4	Crystal
ATmega32U2	0x1E	0x95	0x8A	0x5E	0xD9	0xF4	Crystal
ATmega16U4RC	0x1E	0x94	0x88	0x52	0x99	0xFB	8/8MHz RC
ATmega32U4RC	0x1E	0x95	0x87	0x52	0x99	0xFB	8/8MHz RC
ATmega16U4	0x1E	0x94	0x88	0x5E	0x99	0xF3	Crystal
ATmega32U4	0x1E	0x95	0x87	0x5E	0x99	0xF3	Crystal

Table 4.3. Signature Bytes and default fuses of AT90 and mega..U Processors

4.2 Layout of the fuses

Typ	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
m8	BODLEVEL	BODEN	SUT1	SUT0	CKSEL3:0			
m16/32	BODLEVEL	BODEN	SUT1	SUT0	CKSEL3:0			
m64/128	BODLEVEL	BODEN	SUT1	SUT0	CKSEL3:0			
t24/25	CKDIV8	CKOUT	SUT1	SUT0	CKSEL3:0			
t2313/261	CKDIV8	CKOUT	SUT1	SUT0	CKSEL3:0			
t4313/44	CKDIV8	CKOUT	SUT1	SUT0	CKSEL3:0			
t45/461	CKDIV8	CKOUT	SUT1	SUT0	CKSEL3:0			
t84/85	CKDIV8	CKOUT	SUT1	SUT0	CKSEL3:0			
t861/87	CKDIV8	CKOUT	SUT1	SUT0	CKSEL3:0			
t167	CKDIV8	CKOUT	SUT1	SUT0	CKSEL3:0			
m48/88	CKDIV8	CKOUT	SUT1	SUT0	CKSEL3:0			
m165/168/169	CKDIV8	CKOUT	SUT1	SUT0	CKSEL3:0			
m325/328/329	CKDIV8	CKOUT	SUT1	SUT0	CKSEL3:0			
m645/649	CKDIV8	CKOUT	SUT1	SUT0	CKSEL3:0			
90PWM2/3	CKDIV8	CKOUT	SUT1	SUT0	CKSEL3:0			
m8U2/16U2/32U2	CKDIV8	CKOUT	SUT1	SUT0	CKSEL3:0			
m16U4/32U4	CKDIV8	CKOUT	SUT1	SUT0	CKSEL3:0			
t441/841	CKDIV8	CKOUT	-	SUT	CKSEL3:0			
t48/88	CKDIV8	CKOUT	-	SUT	CKSEL3:0			
t1634	CKDIV8	CKOUT	-	SUT	CKSEL3:0			
t26	PLLCK	CKOPT	SUT1	SUT0	CKSEL3:0			

Table 4.4. Layout of the AVR Low Fuses

Typ	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
m8	RST-DISBL	WDTON	SPIEN	CKOPT	EESAVE	BOOTSZ1:0		BOOT-RST
m16/32	OCDEN	JTAGEN	SPIEN	CKOPT	EESAVE	BOOTSZ1:0		BOOT-
m64/128	OCDEN	JTAGEN	SPIEN	CKOPT	EESAVE	BOOTSZ1:0		RST
m165/169	OCDEN	JTAGEN	SPIEN	WDTON	EESAVE	BOOTSZ1:0		BOOT-
m325/329	OCDEN	JTAGEN	SPIEN	WDTON	EESAVE	BOOTSZ1:0		RST
m645/649	OCDEN	JTAGEN	SPIEN	WDTON	EESAVE	BOOTSZ1:0		BOOT-
m640	OCDEN	JTAGEN	SPIEN	WDTON	EESAVE	BOOTSZ1:0		RST
m1280	OCDEN	JTAGEN	SPIEN	WDTON	EESAVE	BOOTSZ1:0		BOOT-
m2560	OCDEN	JTAGEN	SPIEN	WDTON	EESAVE	BOOTSZ1:0		RST
m16U4	OCDEN	JTAGEN	SPIEN	WDTON	EESAVE	BOOTSZ1:0		BOOT-
m32U4	OCDEN	JTAGEN	SPIEN	WDTON	EESAVE	BOOTSZ1:0		RST
m328	RST-DISBL	DWEN	SPIEN	WDTON	EESAVE	BOOTSZ1:0		BOOT-RST
m88/48	RST-	DWEN	SPIEN	WDTON	EESAVE	BODLEVEL2:0		
m168	DISBL	DWEN	SPIEN	WDTON	EESAVE	BODLEVEL2:0		
90PWM2	RST-	DWEN	SPIEN	WDTON	EESAVE	BODLEVEL2:0		
90PWM3	DISBL	DWEN	SPIEN	WDTON	EESAVE	BODLEVEL2:0		
t24/25	RST-	DWEN	SPIEN	WDTON	EESAVE	BODLEVEL2:0		
t261	DISBL	DWEN	SPIEN	WDTON	EESAVE	BODLEVEL2:0		
t44/441	RST-	DWEN	SPIEN	WDTON	EESAVE	BODLEVEL2:0		
t461	DISBL	DWEN	SPIEN	WDTON	EESAVE	BODLEVEL2:0		
t45/48	RST-	DWEN	SPIEN	WDTON	EESAVE	BODLEVEL2:0		
t84/841	DISBL	DWEN	SPIEN	WDTON	EESAVE	BODLEVEL2:0		
t85/87	RST-	DWEN	SPIEN	WDTON	EESAVE	BODLEVEL2:0		
t88/861	DISBL	DWEN	SPIEN	WDTON	EESAVE	BODLEVEL2:0		
t167	RST-	DWEN	SPIEN	WDTON	EESAVE	BODLEVEL2:0		
t1634	DISBL	DWEN	SPIEN	WDTON	EESAVE	BODLEVEL2:0		
t4313	DWEN	EESAVE	SPIEN	WDTON	BODLEVEL2:0			RST-
t2313	DWEN	EESAVE	SPIEN	WDTON	BODLEVEL2:0			DISBL
m8U2	DWEN	RSRDISBL	SPIEN	WDTON	EESAVE	BOOTSZ1:0		BOOT-
m16U2	DWEN	RSRDISBL	SPIEN	WDTON	EESAVE	BOOTSZ1:0		RST
m32U2	DWEN	RSRDISBL	SPIEN	WDTON	EESAVE	BOOTSZ1:0		
t26	-	-	-	RST-DISBL	SPIEN	EE-SAVE	BOD-LEVEL	BODEN

Table 4.5. Layout of the AVR High Fuses

Typ	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
m64/128	-	-	-	-	-	-	M103C	WDTON
m165/169	-	-	-	-	BODLEVEL2:0			RST-
m325/329	-	-	-	-	BODLEVEL2:0			DISBL
m645/649	-	-	-	-	BODLEVEL2:0			RST-
m640	-	-	-	-	BODLEVEL2:0			DISBL
m1280	-	-	-	-	BODLEVEL2:0			RST-
m2560	-	-	-	-	BODLEVEL2:0			DISBL
m48	-	-	-	-	-	-	-	SELF-
t24/25	-	-	-	-	-	-	-	PRGEN
t261	-	-	-	-	-	-	-	SELF-
t2313	-	-	-	-	-	-	-	PRGEN
t4313	-	-	-	-	-	-	-	SELF-
t44/45	-	-	-	-	-	-	-	PRGEN
t461/48	-	-	-	-	-	-	-	SELF-
t84/85	-	-	-	-	-	-	-	PRGEN
t861	-	-	-	-	-	-	-	SELF-
t87/88	-	-	-	-	-	-	-	PRGEN
t167	-	-	-	-	-	-	-	
t441/841	ULPOSCSEL2:1			BODPD1	BODPD0	BODACT1:0		SELF-PRGEN
t1634	-	-	-	BODPD1	BODPD0	BODACT1:0		SELF-PRGEN
m168/88	-	-	-	-	-	BOOTSZ1:0		BOOT-
m328	-	-	-	-	-	BOOTSZ1:0		RST
90PWM2	PSC2RB	PSC1RB	PSC0RB	PSCRV	-	BOOTSZ1:0		BOOT-
90PWM3	PSC2RB	PSC1RB	PSC0RB	PSCRV	-	BOOTSZ1:0		RST
m8U2	-	-	-	-	HWBE	BODLEVEL2:0		
16U2	-	-	-	-	HWBE	BODLEVEL2:0		
32U2	-	-	-	-	HWBE	BODLEVEL2:0		
m16U4	-	-	-	-	HWBE	BODLEVEL2:0		
32U4	-	-	-	-	HWBE	BODLEVEL2:0		

Table 4.6. Layout of the AVR Extended Fuses

4.3 Possible internal clock frequencies

The following table 4.7 shows the possible internal clock frequency selections for different AVR processors.

AVR-Typ	:8 scaler	Calibrated RC generator	128kHz	additional
t24,t44,t84	X	8 MHz	X	
t87,t167	X	8 MHz	X	
t48,t88	X	8 MHz	X	
t25,t45,t85	X	8, 6.4 MHz	X	
t2313,t4313	X	8, 4 MHz	X	
t441,t841	X	8 MHz	-	32-512kHz
t1634	X	8 MHz	-	32kHz
t261,t461,t861	X	8 MHz	X	16MHz PLL
m8,m16,m32,m64,m128	-	8,4,2,1 MHz	-	
m8515,m8535	-	8,4,2,1 MHz	-	
m163	-	1 MHz	-	
m48,m88,m168,m328	X	8 MHz	X	
m164,m324,m644,m1284	X	8 MHz	X	
m165,m325,m645	X	8 MHz	-	
m169,m329,m649	X	8 MHz	-	
m640,m1280,m2560	X	8 MHz	-	
m162	X	8 MHz	-	
at90CAN32,64,128	X	8 MHz	-	
m8U2,m16U2,m32U2	X	8 MHz	-	
m16U4,m32U4	X	8 MHz	-	
at90PWM2,2B,3,3B	X	8 MHz	-	16MHz PLL

Table 4.7. Internal clock frequencies of some AVR processors

Chapter 5

Various USB to serial converter with Linux

The classic serial interface is less and less used today. On older computers this interface with RS232 standard is more common. However, the classic serial interface is not very practical anyway, because the voltage levels used (about -12V and +12V) can not be used directly. This voltage levels must be first converted back to +5V or +3.3V common with microcontrollers. For purpose of programming microcontrollers the USB to serial converter are more practical, because these provide the proper signal level and can also provide the +5V or +3.3V supply for the microcontroller. With the Linux operating system these USB devives are most detected without problems. But problems can occur to access this devices with the normal user access rights. Mostly the access nodes like /dev/ttyUSB1 are assigned to the group dialout. Then you (the user) should belong to this dialout group. You can be member of the group dialout with the command "usermod -a -G dialout %USER".

5.1 The CH340G and the CP2102 converter

Checked have I a board with the label BTE13-005A, at which a CH340G converter from QinHeng Elektronics is mounted. The board has a mini switch to switch the VCC voltage to 3.3V or 5V and a 12 MHz crystal. At one side there is a USB-A plug and on the other side a 6-pin strip with signals GND, CTS, VCC, TXD, RXD and DTR. One LED connected to VCC and another LED is also mounted at the board. You can find a list of supported baud rates in the chinese datasheet of the CH340G chip. The other board with the CP2102 converter from Silicon Laboratories Inc. has no label. At one side of the board you will find also a USB-A plug and at the opposite side there is also a 6-pin strip with the signals 3.3V, GND, +5V, TXD, RXD and DTR. At the two other sides of the board you can add a 4-pin strip with the signals DCD, D3R, RTS and CTS as well as RST, R1, /SUS and SUS. All signals required by the bootloader are present already at the mounted strip. But the sequence of the signals is different for both boards. You can find only a few parts additional to the CP2102 converter chip, but a LED for RXD, TXD and power is present. With my Linux-Mint 17.2 both convertes are detected without further ado. You can see the folowing output of the lsusb command:

```
Bus 002 Device 093: ID 10c4:ea60 Cygnal Integrated Products, Inc. CP210x UART Bridge / myA
Bus 002 Device 076: ID 0403:6001 Future Technology Devices International, Ltd FT232 Serial
```

Of course, the information on the bus depends on the computer and the USB port used. The automatically created device name after the connection can be find out by the command "dmesg | tail -20". In the following example I have summarized both results.

```
usb 2-4.2: new full-speed USB device number 94 using ohci-pci
usb 2-4.2: New USB device found, idVendor=1a86, idProduct=7523
usb 2-4.2: New USB device strings: Mfr=0, Product=2, SerialNumber=0
usb 2-4.2: Product: USB2.0-Serial
ch341 2-4.2:1.0: ch341-uart converter detected
usb 2-4.2: ch341-uart converter now attached to ttyUSB1
usb 2-4.5: new full-speed USB device number 93 using ohci-pci
usb 2-4.5: New USB device found, idVendor=10c4, idProduct=ea60
usb 2-4.5: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 2-4.5: Product: CP2102 USB to UART Bridge Controller
usb 2-4.5: Manufacturer: Silicon Labs
usb 2-4.5: SerialNumber: 0001
cp210x 2-4.5:1.0: cp210x converter detected
usb 2-4.5: reset full-speed USB device number 93 using ohci-pci
usb 2-4.5: cp210x converter now attached to ttyUSB2
```

With this information and my own experiments I have created the tables 5.1 and 5.2. Because the newer operating system Linux Mint 18.3 has given partially better results, I have measured the results with this system.

CH340G supported BaudRate	CH340G stty speed	CH340G measured BaudRate	CP2102 supported BaudRate	CP2102 stty speed	CP2102 measured BaudRate	AVR UBBR @16MHz
50	50	50.00	-	Error		
75	75	75.18	-	Error		
100	Error	-	-	Error		
110	110	109.3	-	Error		
134	134	133.4	-	Error		
150	150	150.4	-	Error		
300	300	300.7	300	300	300.7	
600	600	602.4	600	600	598.8	3332
900	Error	-	(900)	Error	-	2221
1200	1200	1204.8	1200	1200	1198	832
1800	1800	1801.6	1800	1800	1802	555
2400	2400	2409.6	2400	2400	2410	416
3600	Error	-	(3600)	Error	-	277
(4000)	Error	-	4000	Error	-	249
4800	4800	4808	4800	4800	4808	207
(7200)	Error	-	7200	Error	-	138
9600	9600	9616	9600	9600	9616	207
14400	Error	-	14400	Error	-	138
-	Error	-	16000	Error	-	124
19200	19200	19232	19200	19200	19232	103

Table 5.1. Tested baud rates of the CH340 and CP2102 chips at lower baud speed

CH340G supported BaudRate	CH340G stty speed	CH340G measured BaudRate	CP2102 supported BaudRate	CP2102 stty speed	CP2102 measured BaudRate	AVR UBBR @16MHz
28800	Error	-	28800	Error	-	68
33600	Error	-	(33600)	Error	-	59
38400	38400	38.464k	38400	38400	38.464k	51
(51200)	Error	-	51200	Error	-	38, 0.16%
56000	Error	-	56000	Error	-	35, -0.79%
57600	57600	57.8k	57600	57600	57.472k	34, -0.79%
(64000)	Error	-	64000	Error	-	30, 0.80%
76800	Error	-	76800	Error	-	25, 0.16%
115200	115200	115.6k	115200	115200	114.96k	16, 2.12%
128000	Error	-	128000	Error	-	15, -2.34%
153600	Error	-	153600	Error	-	12, 0.16%
230400	230400	229.9k	230400	230400	229.9k	8, -3.54%
(250000)	Error	-	250000	Error	-	7, 0.00%
(256000)	Error	-	256000	Error	-	7, -2.34%
460800	460800	460.8k	460800	460800	458.7k	-, >5%
(500000)	500000	500.0k	500000	500000	500.0k	3, 0.00%
(576000)	576000	**543.4k**	576000	576000	571.4k	-, >5%
921600	921600	**851.2k**	921600	921600	921.6k	-, >5%
(1000000)	1000000	1000k	(1000000)	1000000	**921.6k**	1, 0.00%
(1200000)	Error	-	(1200000)	Error	-	-, >5%
1500000	1500000	1498k	(1500000)	1500000	1498k	-, >5%
2000000	2000000	2000k	(2000000)	2000000	**1504k**	0, 0.00%
(3000000)	3000000	3007k	(3000000)	2000000	-	-, >5%

Table 5.2. Tested baud rates of the CH340 and CP2102 chips at higher baud speed

In both tables you can see, that not all baud rates specified by the manufactorers are adjustable with the Linux stty command. In most cases an error is reported, but unfortunately not always. With the CP2102 converter the error limit is exceeded at the baud rates 1 MBaud and 2 MBaud. Even at the 576 kBaud rate the deviation is higher than expected. With the CH340G converter there are only two baud rate noticeable, the setting 576000 and 921600. All other baud selections have a uncritical tolerance. I have not searched for the reason of this abnormalities. But something is dubious with the documentation of the CH340G chip. We can select the baud rate 500 kBaud and 1 MBaud, which should be impossible according to the documentation.

5.2 The PL-2303 and the FT232R converter

This time I have tested a board with the label ”SBT5329” and the PL-2303RX converter from Profilic Technology and a board ”FTDI Basic 1” with a FT232RL converter from Future Technology Devices.

The SBT5329 board has a USB-A plug at one side and a 5-pin strip with the signals +5V, GND, RXD, TXD, and 3.3V at the opposite side. Additionally you can find a 12 MHz crystal and three LEDs Tx, Rx and power. The control signals of the serial interface are not routed to the strip at this board. But the PL-2303 chip provide the handshake signals of the serial interface.

The FTDI Basis 1 board has a USB-B plug at one side and at the opposite side a 6-pin strip with the signals GND, CTS, 5V, TXD, RXD and DTR. Only a few parts can be found at the boardadditional to the FT232RL chip. But two LEDs for Tx and Rx are also present.

As with the other two boards, the devices are properly recognized by Linux Mint 17.1:

```
Bus 002 Device 095: ID 067b:2303 Prolific Technology, Inc. PL2303 Serial Port
Bus 002 Device 076: ID 0403:6001 Future Technology Devices International, Ltd FT232 Serial
```

So the devices are shown with the command lsusb. With the command ”dmesg | tail -20” you can get direcly after plug in a simular result as:

```
usb 2-4.5: new full-speed USB device number 95 using ohci-pci
usb 2-4.5: New USB device found, idVendor=067b, idProduct=2303
usb 2-4.5: New USB device strings: Mfr=1, Product=2, SerialNumber=0
usb 2-4.5: Product: USB-Serial Controller
usb 2-4.5: Manufacturer: Prolific Technology Inc.
pl2303 2-4.5:1.0: pl2303 converter detected
usb 2-4.5: pl2303 converter now attached to ttyUSB1
```

or rather

```
usb 2-4.3: new full-speed USB device number 96 using ohci-pci
usb 2-4.3: New USB device found, idVendor=0403, idProduct=6001
usb 2-4.3: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 2-4.3: Product: FT232R USB UART
usb 2-4.3: Manufacturer: FTDI
usb 2-4.3: SerialNumber: A50285BI
ftdi_sio 2-4.3:1.0: FTDI USB Serial Device converter detected
usb 2-4.3: Detected FT232RL
usb 2-4.3: Number of endpoints 2
usb 2-4.3: Endpoint 1 MaxPacketSize 64
usb 2-4.3: Endpoint 2 MaxPacketSize 64
usb 2-4.3: Setting MaxPacketSize 64
usb 2-4.3: FTDI USB Serial Device converter now attached to ttyUSB0
```

In the table 5.3 the baud setting for the FT232R converter is tested below 300 baud, which can not be solved by the chip. But the stty command accept these settings without an error message. Of course the resulting baud rate is incorrect for these settings.

PL2303 supported BaudRate	PL2303 stty speed	PL2303 measured BaudRate	FT232R supported BaudRate	FT232R stty speed	FT232R measured BaudRate	AVR UBBR @16MHz
75	75	75.18	(75)	75	**415**	
(110)	110	109.9	(110)	110	**278**	
(134)	134	135.1	(134)	134	**502**	
150	150	149.8	(150)	150	**832**	
300	300	300.7	300	300	300.72	
600	600	602.4	600	600	602.4	3332
(900)	Error	-	900	Error	-	2221
1200	1200	1204.8	1200	1200	1212	832
1800	1800	1801.6	1800	1800	1809.6	555
2400	2400	2409.6	2400	2400	2424	416
3600	Error	-	3600	Error	-	277
4800	4800	4808	4800	4800	4831	207
7200	Error	-	7200	Error	-	138
9600	9600	9616	9600	9600	9664	207
14400	Error	-	14400	Error	-	138
19200	19200	19232	19200	19200	19320	103

Table 5.3. Tested baud rates of the PL-2303 and FT232R chips at lower baud speed

In the table 5.4 the baud setting for the FT232R converter are done correctly, when the stty command report no error. I don't know the reason, why some baud rates are not accepted by the stty command for the FT232R chip. Only the 576000 setting can be solved more exactly by the chip than done here.

PL2303 supported BaudRate	PL2303 stty speed	PL2303 measured BaudRate	FT232R supported BaudRate	FT232R stty speed	FT232R measured BaudRate	AVR UBBR @16MHz
28800	Error	-	28800	Error	-	68
(33600)	Error	-	33600	Error	-	59
38400	38400	38.464k	38400	38400	38.6k	51
(51200)	Error	-	51200	Error	-	38, 0.16%
(56000)	Error	-	56000	Error	-	35, -0.79%
57600	57600	57.8k	57600	57600	57.8k	34, -0.79%
(64000)	Error	-	64000	Error	-	30, 0.80%
(76800)	Error	-	76800	Error	-	25, 0.16%
115200	115200	115.6k	115200	115200	115.6k	16, 2.12%
(128000)	Error	-	128000	Error	-	15, -2.34%
(153600)	Error	-	153600	Error	-	12, 0.16%
230400	230400	231.2k	230400	230400	231.2k	8, -3.54%
(250000)	Error	-	250000	Error	-	7, 0.00%
(256000)	Error	-	256000	Error	-	7, -2.34%
460800	460800	460.8k	460800	460800	465.1k	-, >5%
(500000)	500000	500.0k	500000	500000	500.0k	3, 0.00%
(576000)	Error	-	576000	576000	**588.24k**	-, >5%
921600	921600	925.6k	921600	921600	930.4k	-, >5%
(1000000)	1000000	1000k	1000000	1000000	1005k	1, 0.00%
(1200000)	Error	-	1200000	Error	-	-, >5%
(1500000)	Error	1482k	1500000	1500000	1509k	-, >5%
(2000000)	2000000	2010k	2000000	2000000	2020k	0, 0.00%
(3000000)	3000000	3007k	3000000	3000000	3007k	-, >5%

Table 5.4. Tested baud rates of the PL-2303 and FT232R chips at higher baud speed

5.3 The USB-serial converter with the ATmega16X2 software

At some Arduino UNO boards a ATmega16X2 is used as USB-serial converter. Therefore I would like to test this board for selectable baud rates. At the first table 5.5 for the lower baud speeds is remarkable, that baud rates below 600 are accepted by the stty command without error messages. But all baud rates above 600 are implemented correctly.

Mega16X2 supported BaudRate	Mega16X2 stty speed	Mega16X2 measured BaudRate	AVR UBBR @16MHz
75	75	**956**	
110	110	**1120**	
134	134	**757.6**	
150	150	**1914**	
300	300	**778**	
600	600	599	3332
900	Error	-	2221
1200	1200	1198	832
1800	1800	1802	555
2400	2400	2395	416
3600	Error	-	277
4800	4800	4808	207
7200	Error	-	138
9600	9600	9616	207
14400	Error	-	138
19200	19200	19232	103

Table 5.5. Tested baud rates of the ATmega16X2 at lower baud speed

At the table 5.6 for the upper baud speeds you can see, that not all settings without error message are implemented correctly. Of course a better feedback of the stty command would be desirable to let us know, if a baud rate is selectable or not.

Mega16X2 supported BaudRate	Mega16X2 stty speed	Mega16X2 measured BaudRate	AVR UBBR @16MHz
28800	Error	-	68
33600	Error	-	59
38400	38400	38.321k	51
51200	Error	-	38, 0.16%
56000	Error	-	35, -0.79%
57600	57600	58.82k	34, -0.79%
64000	Error	-	30, 0.80%
76800	Error	-	25, 0.16%
115200	115200	116.9k	16, 2.12%
128000	Error	-	15, -2.34%
153600	Error	-	12, 0.16%
230400	230400	221.0k	8, -3.54%
250000	Error	-	7, 0.00%
256000	Error	-	7, -2.34%
(460800)	460800	**500.0k**	-, >5%
500000	500000	500.0k	3, 0.00%
(576000)	576000	**667k**	-, >5%
(921600)	921600	**995.0k**	-, >5%
(1000000)	1000000	1000k	1, 0.00%
(1200000)	Error	-	-, >5%
(1500000)	Error	**2000k**	-, >5%
2000000	2000000	2000k	0, 0.00%
(3000000)	3000000	**2000k**	-, >5%

Table 5.6. Tested baud rates of the ATmega16X2 at higher baud speed

5.4 Der Pololu USB AVR Programmer v2.1

My Linux system find out two serial interfaces with name /dev/ttyACM0 and /dev/ttyACM1, if the Polulo programmer is plugged in. The first serial interface is used to control the ISP interface. The second serial interface /dev/ttyACM1 is freely availabe. The signals of this serial interface are available at a additional female header (TX, RX, B=DTR, A=free).

The results of my tests for the additional serial interface is shown in the table 5.7. The pololu shows a good result, all tested baud rates above 299 have little deviation, if stty accept them.

tested Baud Rate	stty speed	measured Baud Rate	AVR UBBR @16MHz
75	75	**415**	
110	110	**275**	
134	134	**500**	
150	150	**830**	
300	300	300	
600	600	600	3332
900	Error	-	2221
1200	1200	1200	832
1800	1800	1800	555
2400	2400	2400	416
3600	Error	-	277
4800	4800	4800	207
7200	Error	-	138
9600	9600	9600	207
14400	Error	-	138
19200	19200	19200	103

Table 5.7. tested Baud rates of the Pololu serial Interface at lower Baud speed

Also the results for the higher baud rated in table 5.8 shows a good result. From all by stty accepted baud rates is only 576000 unusable because of the high deviation.

getestete BaudRate	stty speed	gemessene BaudRate	AVR UBBR @16MHz
28800	Error	-	68
33600	Error	-	59
38400	38400	38.32k	51
51200	Error	-	38, 0.16%
56000	Error	-	35, -0.79%
57600	57600	58.86k	34, -0.79%
64000	Error	-	30, 0.80%
76800	Error	-	25, 0.16%
115200	115200	115.28k	16, 2.12%
128000	Error	-	15, -2.34%
153600	Error	-	12, 0.16%
230400	230400	230.56k	8, -3.54%
250000	Error	-	7, 0.00%
256000	Error	-	7, -2.34%
(460800)	460800	461.6k	-, >5%
500000	500000	500.0k	3, 0.00%
(576000)	576000	**541.8k**	-, >5%
(921600)	921600	923.9k	-, >5%
(1000000)	1000000	998.8k	1, 0.00%
(1200000)	Error	-	-, >5%
(1500000)	1500000	1501.2k	-, >5%
2000000	2000000	2000k	0, 0.00%
(3000000)	3000000	3000k	-, >5%

Table 5.8. tested Baud rates of the Pololu serial interface at higher Baud speed

Bibliography

[1] Atmel Corporation *8-bit AVR with 8KBytes In-System Programmable Flash - ATmega8(L),*. Manual, 2486AA-AVR-02/13, 2013

[2] Atmel Corporation *8-bit AVR with 8KBytes In-System Programmable Flash - ATmega8515(L),*. Manual, 2512K-AVR-01/10, 2010

[3] Atmel Corporation *8-bit AVR with 8KBytes In-System Programmable Flash - ATmega8535(L),*. Manual, 2502K-AVR-10/06, 2006

[4] Atmel Corporation *8-bit AVR with 16KBytes In-System Programmable Flash - ATmega16A,*. Manual, 8154C-AVR-07/14, 2014

[5] Atmel Corporation *8-bit AVR with 32KBytes In-System Programmable Flash - ATmega32(L),*. Manual, doc2503-AVR-07/11, 2011

[6] Atmel Corporation *8-bit AVR with 16KBytes In-System Programmable Flash - ATmega163, ATmega163L,*. Manual, 1142E-AVR-02/03, 2003

[7] Atmel Corporation *8-bit AVR with 4/8/16/32KBytes In-System Programmable Flash - ATmega48 - ATmega328,*. Manual, 8271J-AVR-11/15, 2015

[8] Atmel Corporation *8-bit AVR with 64KBytes In-System Programmable Flash - ATmega64, ATmega64L,*. Manual, 2490R-AVR-02/2013, 2013

[9] Atmel Corporation *8-bit AVR with 16/32/64/128KBytes In-System Programmable Flash - ATmega164 - ATmega1284,*. Manual, 8272G-AVR-01/15, 2015

[10] Atmel Corporation *8-bit AVR with 16/32/64KBytes In-System Programmable Flash - ATmega165A/PA - ATmega6450A/P,*. Manual, 8285F-AVR-ATmega-08/2014, 2013-2014

[11] Atmel Corporation *8-bit AVR with 128KBytes In-System Programmable Flash - ATmega128, ATmega128L,*. Manual, 2467X-AVR-ATmega-06/11, 2011

[12] Atmel Corporation *8-bit AVR with 2/4KBytes In-System Programmable Flash - ATtiny2313A-ATtiny4313 ,*. Manual, 8246B-AVR-09/11, 2011

[13] Atmel Corporation *8-bit AVR with 2/4/8KBytes In-System Programmable Flash - ATtiny24-ATtiny44-ATtiny84 ,*. Manual, doc8006-AVR-10/10, 2010

[14] Atmel Corporation *8-bit AVR with 4/8KBytes In-System Programmable Flash - ATtiny441-ATtiny841 ,*. Manual, 8495H-AVR-05/14, 2014

[15] Atmel Corporation *8-bit AVR with 2/4/8KBytes In-System Programmable Flash - ATtiny25-ATtiny45-ATtiny85 ,*. Manual, 2586N-AVR-04/11, 2011

[16] Atmel Corporation *8-bit AVR with 2/4/8KBytes In-System Programmable Flash - ATtiny261A-ATtiny461A-ATtiny861A ,*. Manual, 8197C-AVR-05/11, 2011

[17] Atmel Corporation *8-bit AVR with 4/8KBytes In-System Programmable Flash - ATtiny48-ATtiny88 ,*. Manual, 8008H-AVR-04/11, 2011

[18] Atmel Corporation *8-bit AVR with 16KBytes In-System Programmable Flash - ATtiny1634 ,*. Manual, Atmel-8303H-AVR-ATtiny1634-Datasheet_02/2014, 2014

[19] Atmel Corporation *8-bit AVR with 32/64/128KBytes of ISP Flash and CAN Controller - AT90CAN32-AT90CAN64-AT90CAN128 ,*. Manual, 7682C-AUTO-04/08, 2008

[20] Atmel Corporation *8-bit AVR with 8KBytes of In-System Programmable Flash - AT90PWM2B-AT90PWM3B ,*. Manual, 4317K-AVR-03/2013, 2013

[21] Atmel Corporation *STK500 Communication Protocol ,*. Application Note, AVR061-04/03, 2003

[22] Atmel Corporation *Calibration of the internal RC oscillator ,*. Application Note, AVR053-12/03, 2003

[23] Atmel Corporation *Half Duplex Compact Software UART ,*. Application Note, 0952C-AVR-0905, 2005

[24] `http://en.wikibooks.org/wiki/LaTeX` *LaTeX documentation,*. Guide to the LaTeX markup language, 2012

[25] `http://www.xfig.org/userman` *Xfig documentation,*. Documentation of the interactive drawing tool xfig, 2009

[26] `http://www.cs.ou.edu/~fagg/classes/general/atmel/avrdude.pdf` *AVRDUDE User-guide,*. A program for download/uploading AVR microcontroller flash and eeprom, by Brian S. Dean 2006

[27] `http://www.mikrocontroller.net/articles/AVRDUDE` *Online Dokumentation des avrdude programmer interface,* Online Article, 2004-2011

[28] `http://wch.cn` *USB to serial chip CH340,* English DataSheet

[29] `http://www.silabs.com` *CP2102N Data Sheet,* USBXpress Family, 2016

[30] `http://www.silabs.com` *CP210x Baud Rate Support,* AN205 Rev 0.4, 2007

[31] `http://www.ftdichip.com/FTProducts.htm` *FT232R USB UART IC Datasheet,* Version 2.13, 2015

[32] `http://www.ftdichip.com` *Configuring FT232R, FT2232 and FT232B Baud Rates,* AN232B-05, 2004-2006

[33] `http://www.prolific.com.tw` *PL-2303 Edition USB to Serial Bridge Controller,* Product Datasheet, Rev. 1.6, April26, 2006

[34] `https://www.pololu.com/product/3172` *description of the dual use programmer Pololu USB AVR v2.1,* Online description and offer, 2020

[35] `https://www.feeltech.net` *FY6900 Series Function Waveform Generator,* Host Computer Communication Protocol Specification, Rev 1.8