Arbeitsweise und Umfeld des AVR Bootloaders optiboot

Karl-Heinz Kübbeler
kh_kuebbeler@web.de

4. Dezember 2025

Inhaltsverzeichnis

Vorwort

Etwas intensiver habe ich mich mit den AVR-Bootloadern beschäftigt, als der Wunsch von Nutzern aufkam, die Transistortester-Software auf einigen Platinen der Arduino Familie zum Laufen zu bringen. Natürlich läuft die Software nicht als Arduino Sketch. Die Arduino Entwicklungsumgebung wird lediglich zur Darstellung von Ausgaben über die serielle Schnittstelle benutzt. Die Transistortester-Software benutzt auch nicht die Arduino Bibliothek. Das ist auch gar nicht notwendig, um den Bootloader zu benutzen.

Der Bootloader ist ein kleines Programm, welches Programm-Daten über eine serielle Kommunikation von einem Host (PC) entgegennehmen kann und in den Arbeitsspeicher des Mikrocontrollers laden kann. Da die Transistortester-Software ziemlich viel Programmspeicher braucht, sollte der Bootloader nur wenig vom Programmspeicher für sich selbst belegen. Außer dem Programmspeicher sollte der Bootloader auch den anderen nicht flüchtigen Speicher des AVR beschreiben können, das EEprom. Damit war die Zielsetzung klar. Es sollte ein Bootloader her, der auch das Beschreiben des EEproms unterstützt, aber wenig Flash-Speicherplatz benötigt.

Kapitel 1

Prinzipielle Arbeitsweise eines Bootloaders

Ein Bootloader ist ein kleines Programm, welches neue Programm-Daten für einen Prozessor über eine Datenschnittstelle in Empfang nehmen kann und in den Arbeitsspeicher dieses Prozessors ablegen kann. Üblicherweise wird dieses über die Datenschnittstelle empfangene Programm nach Beenden der Übertragung vom Bootloader gestartet. Damit ist ein Rechner mit einem beschreibbaren Arbeitsspeicher dann in der Lage, beliebige Anwenderprogramme aus dem Arbeitsspeicher auszuführen.

Im Prinzip ist damit das BIOS eines PC's auch ein Bootloader, allerdings erweitert um Möglichkeiten, die Schnittstelle für den ersten Zugriff auf Programm-Daten einzustellen. Beim PC läßt sich so oft eine ganze Kette von Peripheriegeräten einstellen, die auf das Vorhandensein von Programm-Daten getestet werden. Beim PC wird dann oft eine zweite Stufe gestartet, die weitere Einstellungen (Auswahl von Betriebssystemen oder abgesicherter Modus) ermöglicht.

Bei Mikrocontrollern ist der Bootloader meistens einfacher gestaltet. Hier wird nur eine Schnittstelle untersucht und es gibt auch keine weitere Einstellmöglichkeit im Betrieb. Ein Merkmal für die Betriebsweise des Bootloaders besteht übrigens im Typ des Arbeitsspeichers. Wenn der Arbeitsspeicher des Rechners aus flüchtigem Speicher besteht (RAM = Random Access Memory), muß der Bootloader vor einem Anwender-Programmstart sicher sein, daß er gerade vorher ein Programm selbst geladen hatte.

Bei einem Mikrocontroller mit nicht flüchtigem Arbeitsspeicher (Flash) darf der Bootloader annehmen, daß bereits irgendwann einmal vorher ein Anwender-Programm in den Speicher geladen wurde. Deshalb wird nach einer angemessenen Wartezeit auf neue Programm-Daten versucht, das Anwenderprogramm zu starten. Dabei ist es egal, ob gerade vorhin ein neues Anwenderprogramm geladen wurde oder nicht. Selbst wenn noch nie ein Anwenderprogramm geladen wurde oder ein fehlerhaftes, passiert nichts Schlimmes. Die Möglichkeit, ein neues Anwenderprogramm zu laden, bleibt ja weiterhin erhalten. Es ist eher das Gegenteil der Fall. Durch das fehlenden Anwenderprogramm versucht der Bootloader immer wieder, die Kommunikation über die serielle Schnittstelle aufzubauen.

Die Abbildung 1.1 zeigt die prinzipielle Arbeitsweise von Bootloadern, die ihre Daten über eine serielle Schnittstelle empfangen.

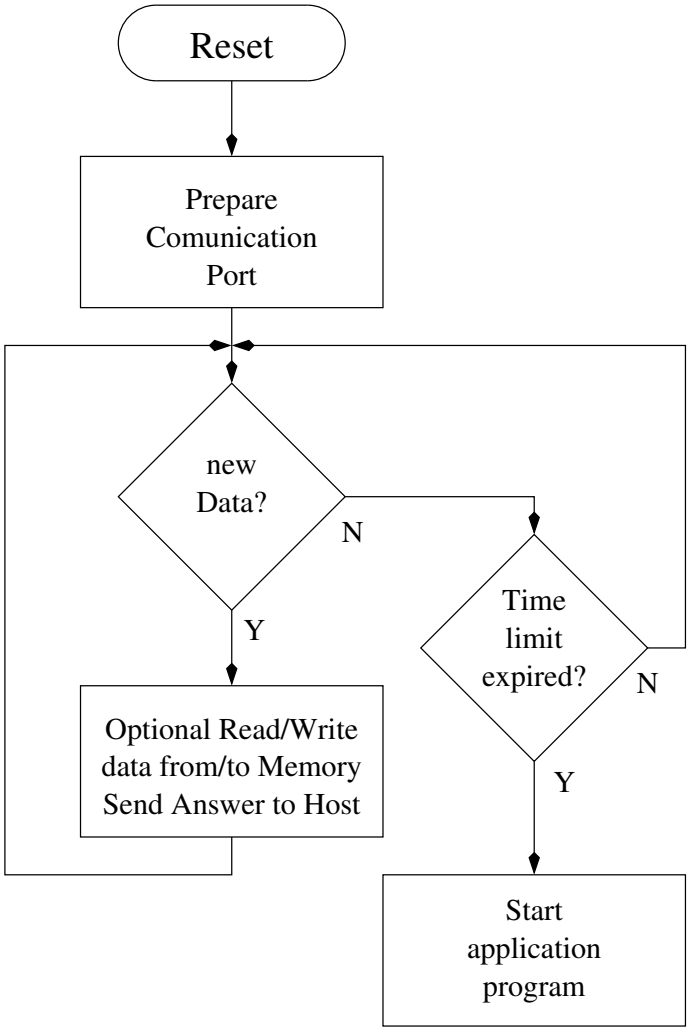


Abbildung 1.1. Prinzipielle Arbeitsweise eines Bootloaders

Der AVR-Zielprozessor wird beim Start des Daten-Sendeprozess auf dem PC zurückgesetzt. Wenn dies nicht automatisch erfolgt, muß der AVR-Prozessor von Hand zurückgesetzt werden. Der PC versucht die Kommunikation mit dem AVR-Prozessor aufzubauen, indem er ein Datenbyte über die serielle Schnittstelle schickt und auf die Antwort des AVR-Prozessors wartet. Wenn die Antwort nicht in angemessener Zeit erfolgt, wird dieser Vorgang wiederholt. Das Bootloader Programm auf dem AVR Prozessor wartet nur eine begrenzte Zeit auf Daten (Time limit). Beim Überschreiten der Wartezeit wird versucht, das Anwender-Programm im Flash-Speicher zu starten.

Kapitel 2

Die Hardware der AVR 8-Bit Mikrocontroller

2.1 CPU und Speicherzugriff

Auf dem Chip der AVR 8-Bit Mikrocontroller ist alles vereinigt, was ein digitaler Minirechner zum Arbeiten braucht. Es ist ein Taktgenerator, Register, Arbeitsspeicher (RAM), Programmspeicher (Flash), Eingaberegister und Ausgaberegister vorhanden. Der Inhalt von Registern und Arbeitsspeicher geht bei jedem Neustart verloren. Der Inhalt des Programmspeicher (Flash) und auch des meistens zusätzlich vorhandenem Datenspeicher (EEprom) bleiben aber erhalten. Die Abbildung 2.1 zeigt ein vereinfachtes Blockdiagramm eines 8-Bit AVR Mikrocontrollers.

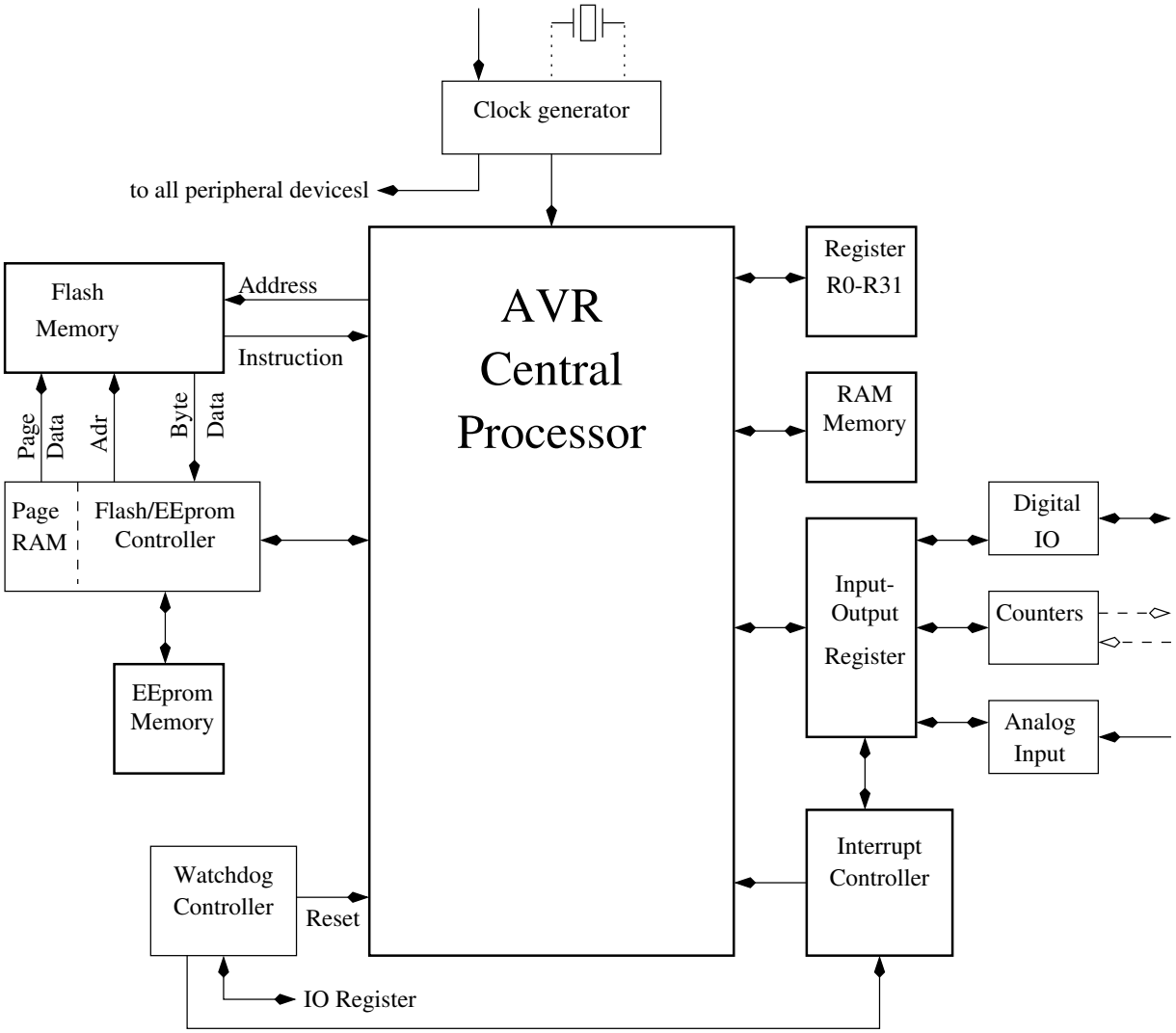


Abbildung 2.1. Vereinfachtes Blockschaltbild eines AVR-Mikrocontrollers

Aus dem Diagramm ist zu entnehmen, daß die CPU (Central Processor Unit) sehr leicht auf die Register R0-R31 und auf den RAM-Speicher zugreifen kann. Auch der Zugriff auf die Eingabe (Input) oder Ausgabe (Output) Register ist leicht möglich. Der Zugriff auf den Programmspeicher (Flash) ist dagegen nur über den zugehörigen Controller möglich und deutlich komplizierter.

Nur die eigentliche Befehls-Ausführungseinheit kann einfach auf die Daten des Flash Speichers für die eingestellte Programm-Adresse zugreifen. Mit dem Load Immediate Befehl (LDI) können dann auch Bestandteile des 16-Bit Befehls-Wortes zu den oberen Registern (R16-R31) transferiert werden. Auch bei den Befehlen ADIW, ANDI, CPI, ORI, SBCI, SBIW und SUBI werden Teile des 16-Bit Befehls-Wortes verarbeitet.

Normalerweise wird bei der Programmausführung die Flash-Adresse je nach Befehls-Größe um ein Wort oder zwei Worte erhöht. Eine Ausnahme hiervon sind bei der normalen Programm-Abarbeitung nur die bedingten oder unbedingten (RJMP, JMP, IJMP, RCALL, CALL, ICALL, RET, RETI) Sprung-Anweisungen.

Außerdem kann ein Reset Ereignis oder ein Unterbrechungs-Signal (Interrupt) für eine Abweichung von der Regel führen. Beim Reset wird der Prozessor zurückgesetzt und der Programm-Zähler auf eine vorher eingestellte Adresse gesetzt. Bei einem Interrupt wird der Programm-Zähler auf die dem Ereignis zugehörende Adresse einer Sprungtabelle gesetzt. Normalerweise ist die Start-Adresse für das Reset Ereignis die Adresse 0. Für Bootloader-Zwecke kann dies aber bei vielen AVR-Prozessoren mit speziellen Konfigurations-Bits (Fuse) anders eingestellt werden.

Ein wahlfreier Lesezugriff zum Programmspeicher ist nur über den Flash-Controller möglich.

Dabei muß dem Controller erst die gewünschte Byte-Adresse mitgeteilt werden. Erst dann ist ein Lesen des Datenbytes mit einem Spezial-Befehl möglich.

Noch komplizierter wird es beim Schreibzugriff. Der Schreibzugriff des Programmspeichers ist nur seitenweise möglich. Die Flash-Speicherseite muß vor jedem Beschreiben vorher gelöscht sein. Die neuen Daten für eine Flash-Seite müssen vor dem Beschreiben komplett in den Zwischenspeicher des Flash-Controllers geladen werden. Erst dann kann der Controller mit dem Neubeschreiben der Flash-Seite beauftragt werden. Dieses Verfahren läßt sich anschaulich mit dem Bedrucken einer Seite mit einem Stempel vergleichen. Der Stempel kann mit austauschbaren Zeichen bestückt werden, so daß jeder Text möglich ist. Für das Stempeln der Information muß aber eine leere Papierseite vorhanden sein und der Stempel muß zuerst mit den Zeichen für den Text bestückt sein. Erst dann kann mit einem Stempeldruck die leere Seite neu beschrieben werden.

Auch der EEprom Speicher (nicht flüchtiger Speicher für Daten) wird über einen speziellen Controller zugegriffen. Hier ist das Beschreiben des Speichers aber etwas einfacher, ist aber auch nur über einen Controller möglich. Der EEprom-Controller kann nicht zusammen mit dem Flash-Controller benutzt werden, da die Controller wohl gemeinsame Teile haben.

2.2 Eingabe und Ausgabefunktion

Über die IO-Register kann die CPU auf die externen Pins zugreifen. Die Register sind Byte weise (8-Bit) organisiert, so daß mit einem Register bis zu 8 Pins angesprochen werden können. Die Abbildung 2.2 zeigt den Aufbau einer Port Pin Beschaltung.

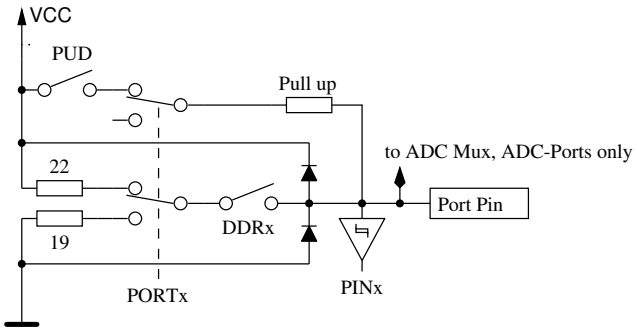


Abbildung 2.2. Vereinfachtes Schaltbild jedes AVR-Portpins

Jeder Pin ist mit einem Port Bit fest verknüpft und kann normalerweise sowohl als Ausgangs-Pin (PORT) als auch als Eingangs-Pin (PIN) verschaltet werden. Zur Umschaltung der Funktion der 8 Bits eines Ports dienen die 8 Bits des Data Direction Registers DDR. Es gibt also für jeden Pin eine zugeordnete Bitnummer in drei verschiedenen Registern. Mit dem DDR Register kann die Datenrichtung für jedes des zugeordneten Pins umgeschaltet werden. Das PIN Register gibt den Zustand der Spannung an den 8 Eingangspins an. Unterhalb etwa der halben Betriebsspannung wird eine 0 angezeigt, darüber eine 1. Wenn das dem Pin zugehörige DDR Bit auf Ausgang (1) geschaltet ist, wird mit dem zugehörigen PORT Bit der Ausgang auf 0 V oder auf Betriebsspannung (VCC) geschaltet. Bei mit dem auf 0 gesetzten DDR Bit inaktiviertem Ausgang wird je nach Zustand des zugehörigen PORT Bits ein Pull-Up Widerstand für diesen Pin zugeschaltet (PORT Bit = 1) oder nicht (PORT Bit = 0). Wenn allerdings das PUD Bit im MCU Konfigurations-Register MCUCR gesetzt ist, bleiben alle internen Pull-Up Widerstände inaktiv. Alle zugehörigen Register und Bits einer Gruppe (8-Bit) benutzen immer den gleichen Kennbuchstaben. Also für die zweite Gruppe wäre der Buchstabe ein B. Der Ausgabe Port würde also PORTB heißen, der Eingangs Port würde PINB heißen und das Register für die Datenrichtung DDRB. Für die Bezeichnung der einzelnen Bits wird jeweils eine Ziffer 0-7 angehängt. Zum Beispiel würde das Bit 0 des Eingangs Ports dann PINB0

heißen. Diese Bezeichnungsweise wird so in der Beschreibung der Hardware von Atmel verwendet und wird üblicherweise auch innerhalb von Programmiersprachen so verwendet.

2.3 Das Starten des AVR Mikrocontrollers

In der vom Werk ausgelieferten Konfiguration des Mikrocontrollers löst das erste Erreichen der erforderlichen Betriebsspannung einen Reset des Prozessors aus. Dabei werden die IO-Register auf vorbestimmte Werte gesetzt, noch etwas Zeit zur Stabilisierung der Betriebsspannung gewartet und dann der Befehl ausgeführt, der auf der Adresse 0 des Flash-Speichers steht. In aller Regel sind die Ausgabe-Funktionen aller Pins in diesem Zustand abgeschaltet. Außer diesem Einschalt-Reset gibt es noch drei weitere Gründe für einen Reset des Prozessors. Der Grund für den Reset wird in einem MCU Status-Register (MCUSR) mit 4 Bits festgehalten.

Name des Flags	Grund für den Reset
PORF	Power-on Reset Dieser Reset wird beim ersten Erreichen der Betriebsspannung ausgelöst. Dieser Grund kann nicht abgeschaltet werden.
BORF	Brown-out Reset Der Reset wegen Spannungseinbruch kann nur dann vorkommen, wenn die Funktion mit den BODLEVEL Bits in dem Fuse-Bit freigeschaltet ist und kein Brown-out Interrupt benutzt wird.
EXTRF	External Reset Wird ausgelöst durch Pegel 0 am Reset Pin, wenn RSTDISBL nicht konfiguriert wurde.
WDRF	Watchdog Reset Wird nur ausgelöst, wenn der zugehörige Interrupt nicht freigeschaltet wurde.

Tabelle 2.1. Die verschiedenen Reset Gründe im MCUSR Register

Durch Setzen der entsprechenden Konfigurations-Bits in den Fuses des AVR-Mikrocontrollers kann die Start-Adresse abweichend von 0 eingestellt werden. Die Abbildung 2.3 zeigt die Möglichkeiten für den ATmega168. Dieser Prozessor hat insgesamt 16384 Byte Arbeitsspeicher (Flash). Der Befehls-Interpreter des Mikrocontrollers greift übrigens auf einen 16-Bit breiten Befehlscode des Flash Speicher zu. Deshalb ist der höchste Programm-Zähler Wert nur 8190!

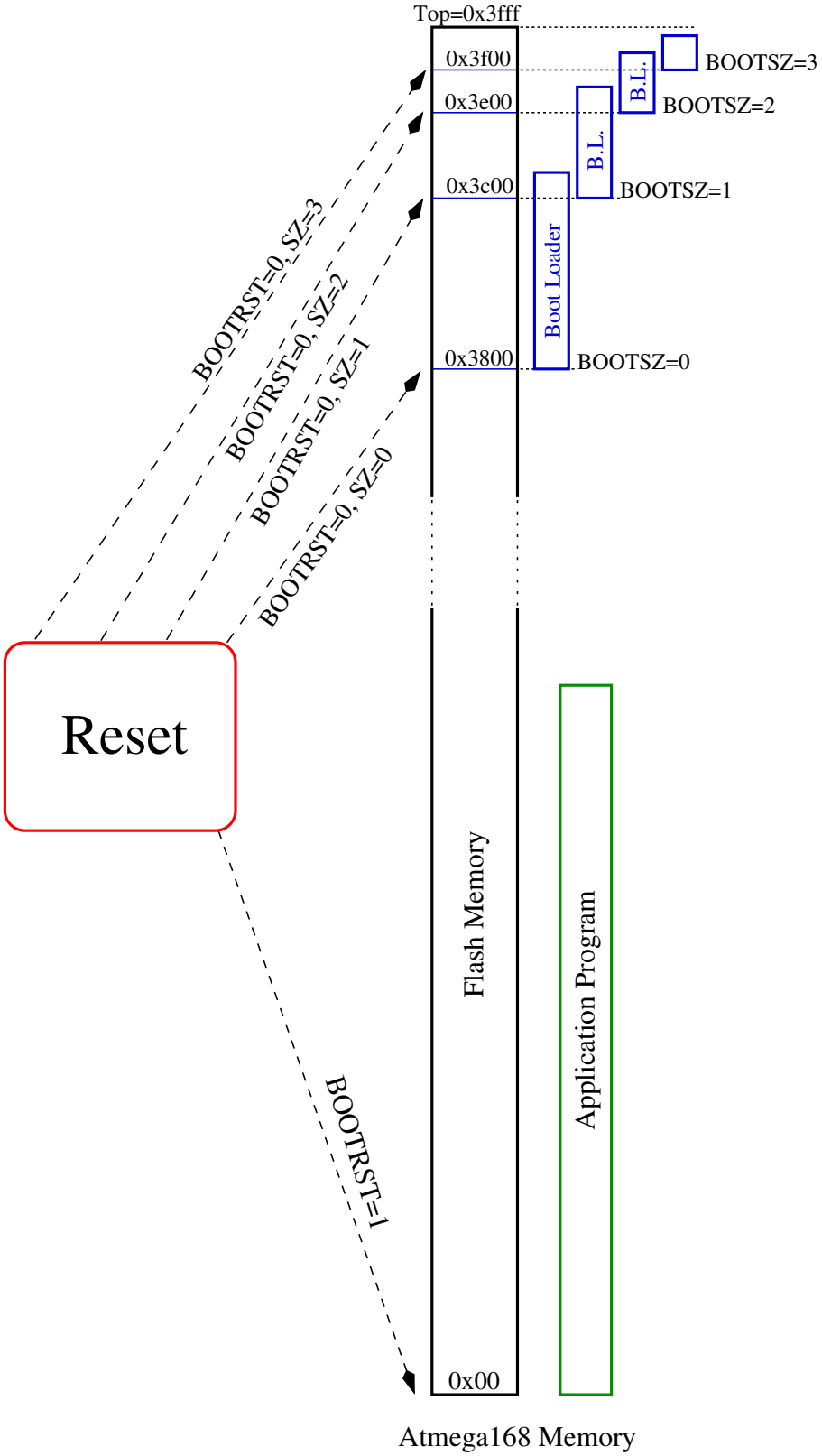


Abbildung 2.3. Verschiedene Start-Möglichkeiten für den ATmega168

Als mögliche Bootloader-Größen unterstützt der ATmega168 256 Bytes (BOOTSZ=3), 512 Bytes (BOOTSZ=2), 1024 Bytes (BOOTSZ=1) und 2048 Bytes (BOOTSZ=0). Für das Anwenderprogramm ist natürlich interessant, den Bootloader-Bereich möglichst klein zu wählen, damit möglichst viel Speicherplatz für das Anwenderprogramm zur Verfügung steht. Der Bootloader wird ja immer auf die höchstmögliche Start-Adresse platziert. Durch Programmieren der Lock-Bits des AVR-Mikrocontrollers läßt sich der Bootloader-Bereich gegen Überschreiben sichern. Die einmal installierte

Sicherung des Flash-Speichers läßt sich dann aber nur durch komplettes Löschen (erase) des AVR Speichers wieder zurücksetzen.

Für diesen Prozessor, wie auch für den Mega48 und Mega88, befinden sich die Steuerbits für den Bootloader Start in der extended Fuse. Dies gilt auch für die BOOTRST Fuse, mit der die Startadresse von 0 auf den Bootloader Start umgeschaltet wird. Für viele andere AVR Mikrocontroller, auch dem ATmega328, befinden sich die gleichen Steuerbits in der high Fuse. Die Tabelle 2.2 zeigt die Speichergrößen für verschiedene AVR-Mikrocontroller sowie die Möglichkeiten für den Bootloader-Bereich. Die verwendeten Bit-Nummern für die Bootloader Einstellung sind übrigens die gleichen, egal ob high oder extended Fuse Register.

Processor Typ	Flash Größe	EEprom Größe	RAM Größe	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

Tabelle 2.2. Bootloader Konfigurationen für verschiedene Mikrocontroller

Übrigens funktioniert ein Bootloader selbst dann beim ersten Mal, wenn das BOOTRST Fuse Bit nicht gelöscht wurde. Dann steht der Reset-Vektor immer noch auf Adresse 0, auf den normalerweise das Anwenderprogramm steht. Da aber noch kein Anwenderprogramm geladen wurde, läuft die CPU durch den Flash Speicher, bis er auf den Bootloader-Code trifft. Bei diesem Prozessor sind das weniger als 8000 Befehle, die die CPU bei 8 MHz Taktrate in weniger als 1 ms durchläuft. Sobald aber ein Anwenderprogramm geladen wurde, startet mit einem Reset immer das Anwenderprogramm, solange das BOOTRST Bit gesetzt bleibt. Der Bootloader würde mit gesetztem BOOTRST Bit nicht mehr funktionieren, weil er gar nicht mehr angesprochen wird.

2.4 Das Beschreiben des AVR Speichers

Die AVR Mikrocontroller kennen 3 verschiedene nicht flüchtige Speicher. Der wichtigste ist der Programmspeicher, der sogenannte Flash-Speicher.

Daneben gibt es noch einige Konfigurationsbits, mit denen die Eigenschaften des Prozessors eingestellt werden können. Diese sind Konfigurationsbits sind aufgeteilt in mehrere Bytes, die lfuse (low Fuse), hfuse (high Fuse), efuse (extended Fuse), das lock-Byte und das Kalibrations-Byte. Das Kalibrations-Byte dient dem Frequenz-Abgleich des internen RC-Oszillators. Mit dem lock-Byte können Zugriffe auf die Speicher gesperrt werden. Beim lock-Byte lassen sich einmal aktivierte Bits nicht wieder zurücksetzen. Der einzige Weg, einmal gesetzte lock-Bits wieder zurückzusetzen, ist ein komplettes Löschen aller AVR-Speicher. Beim lock-Byte werden die Schutzfunktionen durch Löschen von entsprechenden Bits auf 0 aktiviert. Im gelöschten Zustand des AVR-Mikrocontrollers sind alle wirksamen Bits des lock-Bytes auf 1 gesetzt. Die Aufteilung des Konfigurationsbits auf die verschiedenen Fuse-Bytes unterscheidet sich für die verschiedenen Prozessor-Modelle und sollte im jeweiligen Datenblatt nachgelesen werden. Mit den Konfigurationsbits lassen sich Art der Taktgewinnung für den Prozessor, eine Startverzögerung und eine Betriebsspannungsüberwachung einstellen.

Die meisten AVR Mikrocontroller verfügen auch über einen nicht flüchtigen Datenspeicher, das EEprom. Dieser Speicher hat keine spezielle Aufgabe für den Prozessor. Es ist lediglich eine Möglichkeit, Daten für den nächsten Programmstart festzuhalten.

2.4.1 parallele Programmierung

Alle drei Speicherarten können mit verschiedenen Methoden beschrieben und auch gelesen werden. Normalerweise wird die parallele Methode zum Beschreiben der nicht flüchtigen Speicher eher selten verwendet. Manchmal hilft diese Methode aber Prozessoren wieder zu beleben, die mit anderen Methoden nicht mehr ansprechbar sind. Beispielsweise funktioniert die serielle Programmierung dann nicht mehr, wenn mit den Fuse-Bits die Funktion des Reset-Pins abgeschaltet wurde. Diese Methode ist daran zu erkennen, daß bei der Programmierung der Reset Eingangspin auf höhere Spannungswerte (12V) gesetzt wird. Deshalb heißt diese Methode auch HV-Programmierung. Einzelheiten zu dieser Programmierung sollten dem jeweiligen Datenblatt entnommen werden.

2.4.2 serieller Download mit ISP

Der normale Weg zum Programmieren des nicht flüchtigen Speichers ist die serielle Programmierung. Die Atmel Dokumentation spricht auch von Serial Downloading. Dabei wird eine SPI (Serial Parallel Interface) Schnittstelle verwendet. Die SPI Schnittstelle besteht aus drei Signalen, MOSI, MISO und SCK. Damit der Prozessor überhaupt diese spezielle Art der Programmierung benutzt, ist außerdem das Reset Signal erforderlich, das auf 0 gehalten werden muß. Zusammen mit den Betriebsspannungs Signalen GND und VCC (etwa 2.7V bis 5V) ergibt sich eine Schnittstelle, die oft bereits auf einer Platine integriert ist, die ISP (In System Programming) Schnittstelle. Die Abbildung 2.4 zeigt zwei gebräuchliche Stecker, die auf Platinen mit AVR Mikrocontrollern integriert sind.

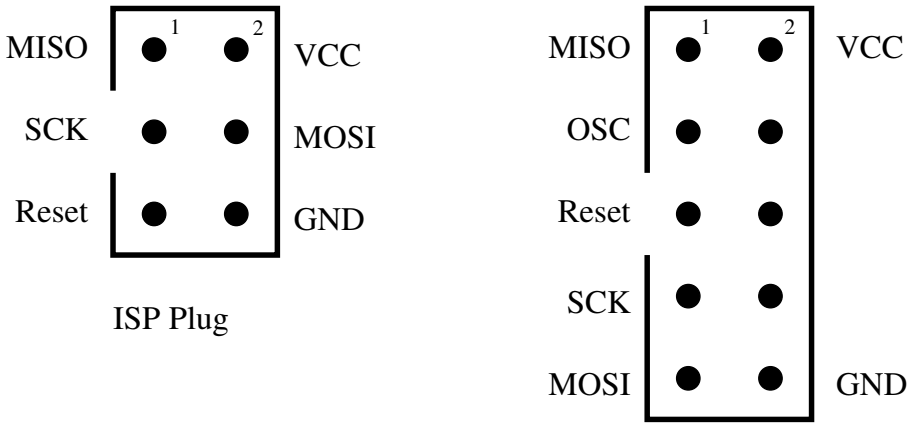


Abbildung 2.4. Zwei verschiedene Arten von ISP Steckern

Bei der 10-poligen Variante des ISP-Steckers kann zusätzlich das Takt Signal OSC für die Versorgung des AVR mit einem Takt aufgelegt sein. Einer dieser beiden Stecker ist also in der Regel erforderlich, um einen neuen Bootloader in den Flash-Speicher eines AVR-Mikrocontrollers zu schreiben. Die Daten für den Flash-Speicher werden im allgemeinen auf einem PC erzeugt. Für das Transferieren der Daten zum AVR-Mikrocontroller ist ein ISP-Programmer erforderlich, der meistens zur Rechnerseite hin über einen USB-Anschluss verfügt. Es ist aber auch ein Anschluss des ISP-Programmers über eine serielle oder parallele Schnittstelle möglich. Die USB-Schnittstelle hat aber den Vorteil, daß auch die Stromversorgung (5V oder 3.3V) für dem Mikrocontroller direkt über die Schnittstelle zur Verfügung gestellt wird. Es gibt auf dem Markt eine Auswahl an solchen ISP-Programmern, von Atmel wird beispielsweise der AT AVR-ISP MK2 Programmer angeboten. Auch die Verwendung eines Arduino UNO oder eines anderen Arduino ist mit Spezialprogramm für eine angeschlossene ISP-Schnittstelle möglich. Ich selbst verwende einem DIAMEX ALL-AVR, der über beide ISP-Steckervarianten verfügt und einige Zusatzfunktionen hat.

2.4.3 Selbstprogrammierung mit serieller Schnittstelle

Da der AVR-Prozessor den Flash und EEprom Speicher auch selbst beschreiben kann, kann man mit einer der beiden anderen Methoden ein kleines Programm in den Flash-Speicher laden, der Daten über eine serielle Schnittstelle empfängt und diese Daten in den Flash- oder auch EEprom-Speicher schreibt. Genau das macht der Bootloader Optiboot. Das Setzen der Fuses oder des Lock-Bytes ist mit dieser Methode oft nicht möglich und wird auch vom Bootloader nicht unterstützt. Die Fuses und das Lock-Byte müssen also immer mit einer der anderen Methoden geschrieben werden. Für die Übermittlung der Daten werden Teile des STK500 Communication Protocol von Atmel verwendet. Da moderne Rechner meist nicht mehr über serielle Schnittstellen verfügen, wird heute meistens ein USB - Seriell UART Wandler wie beispielsweise das FTDI Chip von Future Technology Devices International Ltd verwendet. Ein Modul mit diesem Wandler-Chip ist z.B. das UM232R. Die Chips PL2303 von Prolific Technology Inc. und CP2102 von Silicon Laboratories Inc. erfüllen wohl den gleichen Zweck. Von Atmel kann ein entsprechend programmierter ATmega16U2 auch diese Aufgabe übernehmen. Alle Chips haben eine einstellbare Baud-Rate und haben TTL-Pegel für die Signale. Für echte RS232 Signale müßten noch Pegelwandler verwendet werden. Die sind aber für die AVR Mikrocontroller nicht notwendig. Einer dieser Chips ist auf den Arduino Platinen mit USB-Schnittstelle integriert. Für eine schnelle Reaktion auf die Programm-Datenübertragung sollte das DTR-Signal des Schnittstellenwandlers über einen 100nF Kondensator mit dem Reset Eingang des AVR-Prozessors verbunden sein. Die Abbildung 2.5 zeigt eine typische Anschlussart.

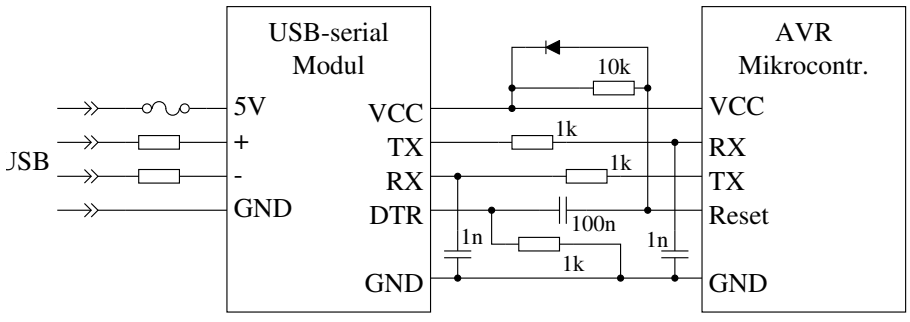


Abbildung 2.5. Anschluss eines USB-seriell Wandlers an den Mikrocontroller

Bei der Verwendung von USB-seriell Modulen sollte auf die richtige Wahl der Betriebsspannung geachtet werden. Viele Module können auf 3.3V oder 5V Signalpegel gestellt werden (Jumper). Wenn man einen Arduino UNO mit gesockeltem ATmega328p besitzt, kann man den ATmega328p entfernen und das Board als USB-seriell Wandler benutzen. So kann man dann auch die Programmdaten zu

einem anderen AVR-Prozessor übertragen. Bei häufigem Gebrauch ist natürlich ein separates USB-seriell Modul sinnvoll. Die beiden 1 nF Kondensatoren an den RX-Eingängen filtern Impulsspitzen weg. Der Test des Software-UART Programms war nur erfolgreich mit einem kleinen Kondensator am RX Eingang des AVR Prozessors. Die Tastspitze eines Oszilloskops war aber als „Filter" schon ausreichend. Die Hardware-UART Schnittstelle filtert wohl besser und läuft mit und ohne Kondensator einwandfrei.

Bei der Arduino Entwicklungsumgebung Sketch ist unter „Tools - serieller Port" die Möglichkeit vorgesehen, eine serielle Schnittstelle auszuwählen. Weiter kann dann unter „Tools - Serial Monitor" ein Kontrollfenster geöffnet werden, welches die seriellen Ausgaben des AVR Mikrocontrollers auf dem Bildschirm darstellen kann. Die Baud-Rate der seriellen Schnittstelle kann in diesem Fenster eingestellt werden. Der laufende Serial Monitor des Arduino Sketch stört aber das Laden von Programmen über die serielle Schnittstelle, wenn das gleiche USB-Seriell Wandlermodul verwendet wird. Es gibt aber die Möglichkeit, ein weiteres USB-Seriell Modul in den PC einzustecken und bei diesem nur das RX-Signal an den TX-Port des AVR-Mikrocontrollers anzuschließen. Dieser zweite serielle Eingang horcht so die Ausgaben des AVR mit und stört dabei die serielle Kommunikation beim Programm-laden nicht, wenn diese Schnittstelle für den Serial Monitor gewählt wird.

2.4.4 Diagnose Werkzeuge

Unter dem Linux-Betriebssystem gibt es ein Werkzeug (Tool) mit dem seltsamen Namen **jpnevulator**, mit dem zwei serielle Eingänge zur selben Zeit überwacht werden können. Die übertragenen Daten werden im sedezimal (hex) Format dargestellt und mit der Option -a auch als ASCII-Zeichen. Mit der Option –timing-print werden die System-Zeiten der empfangenen seriellen Datenpakete gezeigt. Um eine Beeinflussung des Datenverkehrs beim Laden von Programmdaten auszuschließen, sollten zwei separate USB-Seriell Module für die Überwachung benutzt werden. Der serielle Eingang eines Moduls wird an das TX-Signal und der serielle Eingang des anderen Moduls an das RX-Signal des AVR-Mikrocontrollers angeschlossen. Zusammen mit dem Modul für das Programmladen sind dann drei solche Module an den PC angeschlossen. Natürlich müssen alle drei Module auf die gleiche Baudrate eingestellt sein (stty ... -F /dev/ttyUSB1). Ohne den speed Parameter gibt das stty Programm die Parameter der angegebenen Schnittstelle aus (stty -F /dev/ttyUSB1). Einige Anwendungen wie **avrdude** können die Baudrate selber einstellen. Der komplette Aufruf für das Werkzeug mit dem Beginn der Ausgaben könnte beispielsweise so aussehen:

```
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                                             ..
...
```

Daneben hat dat Programm **avrdude** die Möglichkeit, die Aktivitäten zu protokollieren. Eingeschaltet wird diese Funktion mit der Option -v . Durch mehrere v wird das Protokoll ausführlicher. Mit -vvvv wird nahezu jedes Detail protokolliert.

Kapitel 3

Der optiboot Bootloader für AVR Mikrocontroller

Der **optiboot** Bootloader wurde in der Sprache C von Peter Knight und Bill Westfield entwickelt. Die Version 6.2 habe ich als Basis für die hier beschriebene überarbeitete Assembler Version benutzt. Dabei möchte ich betonen, daß ich den **optiboot** Bootloader nicht neu erfunden habe, sondern lediglich weiter optimiert. Viele Anpassungen an verschiedene Zielprozessoren und spezielle Platinenentwürfe waren bereits in der Version 6.2 vorhanden. Es werden Teile des STK500 Kommunikations-Protokols benutzt, die in der AVR061 [21] von Atmel veröffentlicht wurde.

3.1 Änderungen und Weiterentwicklung von Version 6.2

Im wesentlichen habe ich das komplette Programm in die Assembler- Sprache umgeschrieben und mit einem bash Shell Script die Erzeugung der .hex file so umgebaut, daß die Programmlänge automatisch weiterverarbeitet wird und damit die Startadresse des Bootloaders sowie die Fuses des ATmega richtig eingestellt werden. Die eingeschlagene Lösung erzeugt während der Abarbeitung der Einzelschritte für die Erzeugung des Programmcodes für den Bootloader Variablen, welche in den nachfolgenden Schritten für die Anpassung der Start-Adresse und der Fuses erforderlich sind. Die Startadresse für den jeweiligen Zielprozessor ist abhängig von der vorhandenen Flash-Speichergröße, dem Speicherbedarf für den aktuellen Bootloader-Code und der Kachelgröße, die für Bootloader beim Zielprozessor zur Verfügung steht. Als Kachelgröße bezeichne ich die kleinste Speichergröße für Bootloader, die der jeweilige Prozessor zur Verfügung stellen kann.

Bei Prozessoren wie der ATtiny84, die keine Bootloader Startadresse einstellen können, wird die Seitengröße des Flash-Speichers für die Berechnung benutzt. Beim ATtiny84 sind das 64 Bytes. Damit liegt die Startadresse des Bootloaders immer am Anfang einer Flash Speicherseite.

Bei allen anderen Zielprozessoren kann der Bootloader-Bereich mit den Fuse-Bits BOOTSZ1 und BOOTSZ0 eingestellt werden (jeweils mit den Werten 0 und 1). Wenn man die beiden Bits zusammensetzt, ergibt sich daraus die Bootloader-Größe BOOTSZ mit Werten zwischen 0 und 3. Dabei bedeutet 3 immer den kleinsten mögliche Bootloader Speicherbereich. Der Wert 2 gibt den doppelten, der Wert 1 den vierfachen und der Wert 0 den acht-fachen Speicherbereich vor. Die Tabelle 2.2 auf Seite 11 gibt einen Überblick für verschiedene Zielprozessoren. Das Bild 3.1 zeigt fast alle Testplatinen und Prozessoren, die für die **optiboot** Tests verwendet wurden.

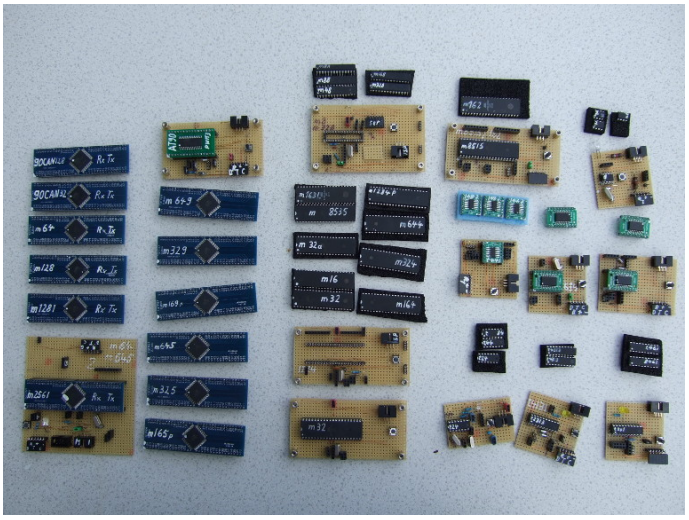


Abbildung 3.1. Einige Testplatinen und AVRs für den **optiboot** Test

3.2 Eigenschaften der Assemblerversion von optiboot

Um viele Fallstricke bei der Erzeugung eines lauffähigen Bootloaders zu umgehen, wurde die Erzeugung der Bootloader Datei weitgehend automatisiert. Zusätzlich werden Einstellungen auch geprüft und die Erzeugung mit einer entsprechenden Fehlermeldung abgebrochen, wenn beispielweise die gewählte Betriebsfrequenz (AVR_FREQ) nicht zu der mit den Fuses (CKSEL,CKDIV8) eingestellten Möglichkeiten für den Takt paßt. Ein Empfang von Programmdaten für **optiboot** ist derzeit nur über eine serielle Schnittstelle möglich.

Hier sind einige der Fähigkeiten des **optiboot** Programms aufgelistet:

- Unterstützt eine große Zahl von AVRs, wobei fast alle auf Funktion getestet wurden.
- Unterstützt das Laden von Daten in den EEprom Speicher.
- Passt in den meisten Fällen in 512 Byte Flash-Speicher.
- Die Anpassung der Programmgröße und die Änderung der notwendigen Fuses wird automatisch vorgenommen.
- Die von der Programmgröße und dem AVR Modell abhängige Startaddresse wird automatisch berechnet und auf dem Bildschirm angezeigt.
- Alle beim jeweiligen AVR-Prozessor vorhandenen seriellen Schnittstellen können frei gewählt werden. Als Standard wird immer das erste UART benutzt.
- Kann die serielle Schnittstelle über Software emulieren. Damit werden auch ATtiny Prozessoren ohne UART unterstützt. Außerdem können die TX und RX Pins bei SOFT_UART völlig frei gewählt werden.
- Bei der SOFT_UART Lösung kann die serielle Kommunikation auch über nur einen AVR Pin durchgeführt werden. Bei Benutzung des Hardware UARTs ist das nicht sinnvoll, da RXD und TXD Pin hier fest vergeben sind.

- Kann eine automatische Baudraten-Anpassung vornehmen. Dabei wird die Baudzeit aus dem ersten empfangenen Datenbyte gemessen. Verschiedene Methoden der Baudzeit-Messung können dabei ausgewählt werden. Wenn der so konfigurierte **optiboot** noch in 512 Byte Flash passen soll, muß die LED Blink Funktion beim Programmstart abgewählt werden.

- Kann auch für AVRs ohne Bootloader Unterstützung benutzt werden.
  Diese VIRTUAL_BOOT_PARTITION Fähigkeit kann natürlich auch für AVRs benutzt werden, die die Bootloader Unterstützung besitzen. Das **optiboot** Programm wird dabei größer, kann aber dann auf jeder Flash-Speicher Seite starten.

- Läßt eine angeschlossene LED standardmäßig drei mal blinken (LED_START_FLASHES=3). Bei eintreffenden seriellen Daten wird das Blinken aber sofort abgebrochen.

- Anstelle des LED-Blinkens beim Start kann man auch die LED aufleuchten lassen, wenn auf serielle Daten gewartet wird (LED_DATA_FLASH=1). Auch ein Dauerleuchten der LED kann mit LED_DATA_FLASH=4 gewählt werden. Dabei geht die LED erst durch den Watchdog-Reset wieder aus.

- Die Länge des erzeugten Programms ist nur abhängig vom gewählten AVR-Prozessor und den ausgesuchten Optionen. Die Version des avr-gcc Kompilers spielt keine Rolle wenn die Assembler Quellen benutzt werden. Dies macht eine nachträgliche Kontrolle (Verify) des installierten Bootloaders einfacher, selbst mit einem anderen PC.

- Unterstützt die Anpassung der Oszillator-Frequenz für den AVR internen RC-Oszillator. Dadurch ist auch in Problemfällen die Benutzung eines Bootloaders mit fester Baudrate möglich.

- Das Laden des **optiboot** Programms in den jeweiligen AVR Prozessor kann automatisch mit dem Programm avrdude und einem angeschlossenem ISP-Programmer erfolgen. Dazu ist nur die zusätzliche Eingabe von ISP=1 beim **make** Aufruf erforderlich. Die zusätzliche Eingabe ISP=2 würde eine Verifikation mit avrdude bewirken. Mit ISP=3 wird der gesamte Programmspeicher (Flash) des angeschlossenen AVR ausgelesen. Mit ISP=4 wird der gesamte EEprom-Speicher des angeschlossenen AVR ausgelesen.

- Unterstützt die App do_spm, welche mit der Original optiboot Version 8 eingebaut wurde. Die befindet sich 2 Byte hinter der Bootloader Startaddresse.

Die nachfolgenden Fähigkeiten beziehen sich nur auf den Herstellungsprozess des Bootloaders:

- Für die Erzeugung des **optiboot** Bootloaders können wahlweise auch angepaßte C-Quellen benutzt werden. Die meisten Funktionen sind auch hier möglich. Natürlich wird das erzeugte **optiboot** Programm größer!

- Der Umfang der Bildschirmausgabe bei der Erzeugung von **optiboot** kann mit der Systemvariablen VerboseLev verändert werden. Der Wert kann zwischen 1 und 4 liegen, wobei 2 der Standardwert ist.

- Die Bildschirmausgabe kann mit der Systemvariablen WITH_COLORS=1 bunter gestaltet werden. Mit WITH_COLORS=2 wird reiner Text ausgegeben

- Die Erzeugung des **optiboot** Programms wird in einer Bash Script Datei gesteuert, wobei die Text-Datei avr_params.def die notwendigen Daten für alle unterstützten AVR-Prozessoren liefert. In dieser avr_params.def Datei sind auch für jeden unterstützten Prozessor Standard-Werte für die Fuses, Arbeitsfrequenz und Baudrate festgelegt.

- Die für die jeweiligen Prozessoren verfügbaren Pinbelegungen findet die Script Datei in dem avr_pins Verzeichnis in einer passenden Datei. Hier werden bei Bedarf auch Standardwerte für die von SOFT_UART benutzten Pins und für die LED gefunden.
- Die Erzeugung des **optiboot** Programms läuft unter Linux mit installierten avr Paketen. Eine Erzeugung unter Windows10 ist möglich, wenn zusätzlich zum Arduino Paket Programmpakete installiert sind, die die bei der Erzeugung erforderlichen Programme wie bash, bc, echo und andere zur Verfügung stellt. Geprüft habe ich das Cygwin64 Paket auf einen Windows10 Laptop mit vorher installierten Arduino Paket.
- Die bei der Programmerzeugung gewählten Parameter werden sowohl am Ende einer .lst Datei als auch in einer .log Datei festgehalten.

3.3 Automatische Größenanpassung in der optiboot Makefile

Die Bootloader Startadresse und die benötigte Bootloadergröße wird automatisch in dem von der Makefile aufgerufenem bash script build_hex.sh angepasst. Für die Berechnungen werden einige Variable erzeugt, was nur zusammen mit den folgenden Linux Werkzeugen funktioniert:

bash ein leistungsfähiger Kommandoprozessor für die Abarbeitung des Scripts.

bc ein einfacher Rechner, der die Eingabe- und Ausgabe-Werte sowohl dezimal als auch sedezimal (hex) verarbeiten kann.

cat gibt den Inhalt von Dateien auf der Standard-Ausgabe aus.

cut kann Teile von Zeilen einer Textdatei ausschneiden.

echo gibt den angegebenen Text auf der Standard-Ausgabe aus.

grep gibt nur Zeilen einer Textdatei mit dem angegebenen Suchtext aus.

tr kann Text-Zeichen ersetzen oder löschen.

Bisher ist die Funktion des bash Scipts nur mit einem Linux-System und unter Windows10 mit installiertem Cygwin Paket zusammen mit den Arduino Paket getestet.

Hier sind die Namen und die Bedeutung der Script Dateien aufgeführt.

build_hex.sh erhält die Variablen von der Makefile, mit der dann ein passender **optiboot** Hex-Datei in Intel Format erzeugt wird. Der build_hex.sh Script ruft die weitere Shell Scripte get_avr_params.sh, avr_family.sh, show_led_pin.sh, show_rx_pin.sh und show_tx_pin.sh auf. Wenn die Variable ISP gesetzt ist, wird auch der Script program_target.sh aufgerufen.

program_target.sh wird von build_hex.sh aufgerufen, um die hfuse und efuse Einstellung zu überprüfen und zu korrigieren. Bei geänderter Anzahl der benutzten Bootpages müssen die BOOTSZ Bits des ATmega angepaßt werden. Die Bits liegen je nach AVR-Modell in dem hfuse oder efuse Byte, was die Korrektur nicht einfacher macht. Nach erfolgten Korrekturen wird das Programm avrdude mit dem Script only_avrdude.sh aufgerufen, um den AVR mit dem **optiboot** Bootloader zu beschreiben und alle Fuses zu setzen.

only_avrdude.sh enthält nur einen avrdude Aufruf mit der in der Variablen DUDE_PARAMS Parameterliste. Zur Kontrolle wird der Aufruf auf dem Terminal protokolliert. Wenn avrdude bei der Rückkehr einen Fehler meldet, gibt der Script allgemeine Hinweise zur Fehlersuche aus. Bei einem Linux System wird auch nach eventuell passenden seriellen Schnittstellen gesucht.

find_serials.sh Sucht in einem Linux System nach möglichen Namen für USB-Seriell Wandlern. Für diese Aufgabe gibt es passende Chips, die auf Linux Systemen meist mit dems Namen /dev/ttyUSB beginnen. Es gibt auch Software-Emulationen auf Mikrokontrollern, die eine USB-Schnittstelle integriert haben. Diese sind meist über einen Namen zugreifbar, der mit /dev/ttyACM beginnt. Es wird auch für jede gefundene Schnittstelle überprüft, ob ein Zugriff möglich ist. Dazu muß man als Benutzer in der Gruppe Mitglied sein, der diesen Gerät den Zugriff erlaubt (meist dialout). Dieser Script wird von only_avrdude bei Fehler Rückkehr aufgerufen, man kann den Script auch mit bash ./find_serials.sh aufrufen. Wenn man mit einer bash arbeitet, kann man sie ausführbar machen und direkt aufrufen mit ./find_serials.sh. Bei Windows ist dieser Script nutzlos. Bei Windows muß man den Geräte Manager benutzen.

get_avr_params.sh findet für den mit MCU_TARGET spezifizierten AVR Prozessor alle benötigten Parameter wie Flashgröße, die Größe der Flash-Seiten und die Größe einer Bootloader Seite. Zusätzlich werden auch einige Parameter wie Betriebsfrequenz, Baudrate und die Belegung der Fuses besetzt, wenn nicht anders vom Nutzer gewünscht. Die Quelle für diese Vorbesetzungen ist die Datei avr_params.def .

avr_family.sh buildet einen Gruppennamen für AVR-Prozessoren mit gleicher Pinbelegung. Normalerweise ist der Gruppenname identisch mit dem AVR Prozessorname mit dem größten Speicher (atmega328 für atmega168 oder atmega88).

show_led_pin.sh findet den Pin für die LED, der für diesen Prozessor oder die Gruppe als Standard vorgegeben ist und protokolliert diese Wahl auf dem Terminal. Wenn der Bediener einen anderen Pin gewählt hat, wird dieser Pin protokolliert. Ohne die Benutzerwahl wird der Pin aus einer avr_pins/<group>.pins Datei entnommen.

show_rx_pin.sh protokolliert den gewählten RX Pin für die eingehenden seriellen Daten. Ohne eine spezielle Benutzerwahl bei SOFT_UART wird der Pin aus einer avr_pins/<group>.pins Datei entnommen.

show_tx_pin.sh protokolliert den gewählten TX Pin für die Ausgabe der seriellen Daten. Ohne eine spezielle Benutzerwahl bei SOFT_UART wird der Pin aus einer avr_pins/<group>.pins Datei entnommen.

baudcheck.tmp.sh wird während der Abarbeitung von build_hex.sh durch einen C-Preprozessor Aufruf aus der Datei baudcheck.S erzeugt und ausgeführt, um Informationen über die gewählte Baudraten Einstellung zu geben.

def_colors.sh wird von verschiedenen Scripts eingefügt, um die Terminalausgabe mit Farben zu versehen.

show_help2.sh gibt Hilfetext für optiboot aus.

show_help3.sh gibt weiteren Hilfetext für optiboot aus.

3.4 Zielvorgaben für optiboot Makefile

Es können verschiedene Konfigurationen auch für einen Prozessortyp festgelegt werden. In der Tabelle 3.1 sind die einige der derzeit vordefinierten Konfigurationen für AVR Prozessoren angegeben. Die einstellbaren Parameter sind aber auch in der Aufrufzeile des **make** Programms als Parameter oder auch als Umgebungsvariable der Shell einstellbar und haben Vorrang zu den Grundeinstellungen. Es gibt zwei Haupt Variabalennamen, die dem bash Script build_hex.sh übergeben werden, TARGET und MCU_TARGET . Die Variable MCU_TARGET muß dabei einen legalen AVR Prozessor Name enthalten. Die Variable TARGET kann mit einem frei wählbaren Namen belegt werden, ist aber üblicherweise identisch dem MCU_TARGET Namen oder ist der Name einer Platine mit diesem Prozessor.

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

Tabelle 3.1. einige Prozessor targets für optiboot Makefile

Alle Angaben für Größen sind in Bytes angegeben, die Werte für die Fuses sind die sedezimalen Werte (hex). Die Frequenz-werte werden in Hz angegeben, 16M entspricht also 16000000 Hz. Die Standard Baud-Rate für die serielle Schnittstelle beträgt bei $16MHz$ Betrieb meistens 115200.

Neben den universellen Konfigurationen gibt es auch Konfigurationen für bestimmte Platinen oder Arbeitsumgebungen. Die Tabelle 3.2 zeigt die unterschiedlichen Einstellungen.

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	-		-

Tabelle 3.2. vorkonfigurierte targets für optiboot Makefile

3.5 Die Optionen für die optiboot Makefile

Mit den Optionen wird die Eigenschaft des **optiboot** Bootloaders eingestellt. Beispielsweise kann mit der Option SOFT_UART veranlasst werden, daß ein Softwareprogramm für die serielle Kommunikation verwendet werden soll. Sonst wird, wenn vorhanden, die auf dem Chip integrierte serielle Schnittstelle mit den Pins TX (Transmit = senden) und RX (Receive = empfangen) benutzt. Bei mehreren integrierten seriellen Schnittstellen wird normalerweise die erste Schnittstelle mit den Nummer 0 verwendet. Es kann aber auch jede andere vorhandene Schnittstelle mit der Option UART vorgegeben werden (UART=1 für die zweite Schnittstelle). Bei der Hardware UART Schnittstelle sind die Pins für Empfangen (RX) und Senden (TX) fest verknüpft. Bei der Software-Lösung für die serielle Schnittstelle sind alle Pins des AVR Prozessors frei für die serielle Kommunikation wählbar. Die einzige Bedingung ist, daß die Pins für digitale Eingabe (RX) beziehungsweise Ausgabe (TX) geeignet sind. Näheres zu den möglichen Optionen findet man in der Übersicht 3.3, 3.4 und 3.5

Name der Option	Beispiel	Funktion
F_CPU	F_CPU=8000000	Teilt dem Programm die Taktrate des Prozessors mit. Die Angabe erfolgt in Hz (Schwingungen pro Sekunde. Das Beispiel gibt eine Frequenz von 8 MHz an.
BAUD_RATE	BAUD_ RATE=9600	Gibt die Baud-Rate für die serielle Kommunikation an. Es werden immer 8 Datenbits ohne Parity verwendet. Werte < 100 aktivieren eine Messung and Anpassung der Baudrate mit verschiedenen Verfahren. Der Vorteil besteht darin, daß man problemlos auf eine niedrigere Baudrate ausweichen kann, wenn Probleme mit einer hohen Baudrate auftauchen.
SOFT_UART	SOFT_UART=1	Wählt Software-Lösung für die serielle Kommunikation.
UART_RX	UART_RX=D0	Gibt den Port und die Bitnummer für die seriellen Empfangsdaten an. Das Beispiel nimmt Bit 0 des D Ports für den seriellen Eingang.
UART_TX	UART_TX=D1	Gibt den Port und die Bitnummer für die seriellen Sendedaten an. Das Beispiel nimmt Bit 1 des D Ports für den seriellen Ausgang.
UART	UART=1	Wählt für die serielle Schnittstelle des Chips. Eine Auswahl setzt das Vorhandensein mehrerer Schnittstellen voraus, funktioniert dann aber auch mit SOFT_UART.
INVERSE_UART	INVERSE_UART=1	Invertiert die Pegel von RX und TX Signalen. Die Option kann nur bei Software UART benutzt werden.
LED_START_ FLASHES	LED_START_ FLASHES=3	Wählt für die Anzahl der Blink-Zyklen für die Kontroll-LED. Bei 1 oder -1 wird nur einmal geblinkt ohne Wiederholung. Negative Vorgaben bedeuten, daß in der Programmschleife das RX Bit der seriellen Schnittstelle nicht überwacht wird. Bei positiven Werten wird die Binkschleife sofort abgebrochen, sobald eingehende RX-Daten festgestellt werden. Bitte beachten Sie,daß das Blinken den Start des Anwenderprogramms verzögert.
LED	LED=B3	Wählt das Port-Bit für die Kontroll-LED. Beim Beispiel würde eine an das Bit 3 des Port B angeschlossene LED blinken. Bei der LED_START_FLASHES Option blinkt die LED die angegebene Anzahl vor dem Kommunikations-Start.
LED_DATA_ FLASH	LED_DATA_ FLASH=1	Die Kontroll-LED leuchtet während des Wartens auf Empfangsdaten der seriellen Kommunikation. Ein Wert von 4 bewirkt, daß die LED nur beim Start einmalig eingeschaltet wird. Hiermit kann man auch erkennen, daß der Bootloader gestartet wurde und es wird viel weniger Platz im Programmspeicher gebraucht.

Tabelle 3.3. Wichtige Optionen für die optiboot Makefile

Bei Betrieb mit internem RC-Generator ist es durchaus möglich, daß ein serieller Datentransfer nicht auf Anhieb gelingt. Das ist prinzipiell auch unabhängig davon, ob die Hardware UART Schnittstelle oder eine Softwarelösung (SOFT_UART) benutzt wird. Ohne zusätzliche Messungen ist man dann auf Probieren mit der OSCCAL_CORR angewiesen. Einen Hinweis kann das Datenblatt des Prozessors liefern. Hier ist beschrieben, bei welcher Betriebsspannung und bei welcher Temperatur der RC-Oszillator kalibriert wurde. Außerdem ist der prinzipielle Verlauf der Frequenzänderung mit der Betriebsspannung, mit der Temperatur und mit OSCCAL-Änderung beschrieben.

Weitere Optionen sind in den Tabellen 3.4 und 3.5 aufgezählt. Diese Optionen sind meist nur für Software-Untersuchungen, die Frequenzkalibration des RC-Generators und für Prozessoren ohne Bootloader-Bereich interessant.

Name der Option	Beispiel	Funktion
TIMEOUT_MS	TIMEOUT_ MS=2000	Diese Option gibt eine Zeitschranke in Millisekunden vor für den Empfang von Boot-Daten. Nach dieser Zeit wird der Bootvorgang abgebrochen und versucht, das Anwenderprogramm zu starten. Mögliche Werte für TIMEOUT_MS sind 500, 1000, 2000, 4000 und 8000. Der tatsächlich mögliche Wert kann abhängig vom Prozessor auf 2 Sekunden begrenzt sein. Wenn kein TIMEOUT_MS angegeben wird, wird die Zeitschranke auf 1 Sekunde gesetzt.
SUPPORT_ EEPROM	SUPPORT_ EEPROM=1	Wählt für das Bootloader-Programm die Lese- und Schreib-Funktion für EEproms. Wenn als Quelle das Assembler-Programm gewählt wurde, ist die EEprom Unterstützung ohne gesetzte Option eingeschaltet, kann aber abgeschaltet werden, wenn die SUPPORT_EEPROM Option auf 0 gesetzt wird.
LFUSE	LFUSE=EF	Gibt einen Wunschwert für das Setzen der Low Fuse des AVRs vor. Es dürfen nur zwei Hex Zeichen angegeben werden.
HFUSE	HFUSE=D9	Gibt einen Wunschwert für das Setzen der High Fuse des AVRs vor. Es dürfen nur zwei Hex Zeichen angegeben werden.
EFUSE	EFUSE=FC	Gibt einen Wunschwert für das Setzen der Extended Fuse des AVRs vor. Es dürfen nur zwei Hex Zeichen angegeben werden.
NO_APP_SPM	NO_APP_SPM=1	Setzt die Nutzerapplikation do_spm außer Funktion wenn 1. NO_APP_SPM=2 bewirkt ein völliges Entfernen der Zusatzfunktion und spart 4 Byte Flash.

Tabelle 3.4. Weitere Optionen für die optiboot Makefile

Name der Option	Beispiel	Funktion
C_SOURCE	C_SOURCE=1	Wählt als Programmquelle das C-Programm anstelle des Assembler-Programms (0 = Assembler). Die Assembler Version benötigt weniger Speicherplatz. Bei der C-Quelle muß die Funktion SUPPORT_EEPROM extra eingeschaltet werden (Standard = aus).
BIGBOOT	BIGBOOT=512	Wählt zusätzlichen Speicherverbrauch für das Bootloader-Programm. Das dient nur zum Test der automatischen Anpassung an die Programmgröße in der Makefile.
VIRTUAL_ BOOT_ PARTITION	VIRTUAL_ BOOT_ PARTITION	Ändert die Programmdaten eines Anwenderprogramms so ab, daß der Bootloader beim Reset angesprochen wird. Für den Start des Anwenderprogramms wird ein anderer Interrupt-Vektor benutzt.
save_vect_ num	save_vect_ num=4	Wählt eine Interrupt-Vektornummer für die VIRTUAL_BOOT_PARTITION Methode aus.
OSCCAL_ CORR	OSCCAL_ CORR=5	Mit der Option OSCCAL_CORR kann der interne 8 MHz RC-Generator des AVR abgeglichen werden. Ist bei Quarz-Betrieb oder externem Takt unwirksam! Der Korrekturwert wird vom voreingestellten OSCCAL Byte abgezogen. Bei positivem Korrekturwert wird die Frequenz normalerweise niedriger. Da die erzeugte Baud-Rate direkt vom Prozessortakt abhängt, ist ein richtig eingestellter Prozessortakt für eine erfolgreiche serielle Kommunikation wichtig. Der Wert sollte zwischen -20 und +20 liegen.
NO_EARLY_ PAGE_ERASE	NO_EARLY_ PAGE_ERASE=1	Verhindert das Löschen der Flash Seite bevor die Daten über die serielle Schnittstelle empfangen sind. Da das Löschen sonst parallel zum Datenempfang abläuft, ist das Programmieren des Flashs mit dieser Option um etwa 30% langsamer. Da der normalerweise auch durchgeführte Datenvergleich (verify) etwa genau so viel Zeit braucht, ist der Zeitverlust nicht so erheblich. Dafür spart man beim **optiboot** Bootloader etwa 14 Bytes Platz in der Bootloader-Seite, was wegen der AVR-Technik in der Praxis auch eine Halbierung des Platzbedarfs bedeuten kann.

Tabelle 3.5. Weitere Optionen für die optiboot Makefile

3.6 Benutzung von optiboot ohne Bootloader-Bereich

Für Prozessoren ohne speziellen Bootloader-Bereich im Flash-Speicher wie dem ATtiny84 ist eine Möglichkeit vorgesehen, trotzdem **optiboot** zu benutzen. Diese Funktion wird mit der Option VIRTUAL_BOOT_PARTITION gewählt. Dabei wird im Anwenderprogramm auf der Reset-Vektoradresse die Start-Adresse des Bootloaders eingetragen damit bei einem Reset immer der Bootloader zuerst angesprochen wird. Die Start-Adresse des Anwender-Programms wird dabei auf die Adresse eines anderen Interrupt-Vektors verlegt. Dieser Interrupt-Vektor sollte vom Anwenderprogramm nicht benutzt werden. Wenn der Bootloader in angemessener Zeit keine Daten von der seriellen

Schnittstelle empfängt, springt er zu dem Sprungbefehl, der auf der „Ersatz"-Vektoradresse steht und startet damit das Anwenderprogramm. Die Abbildung 3.2 soll die Veränderung verdeutlichen.

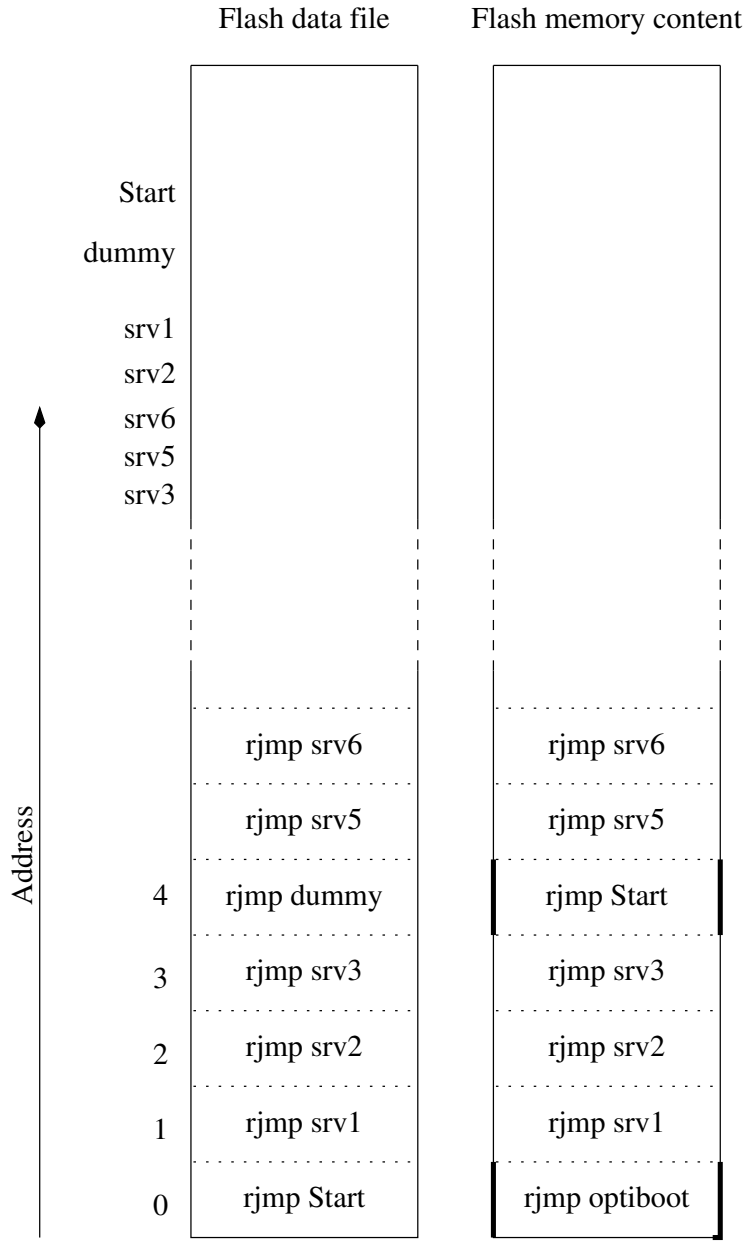


Abbildung 3.2. Veränderung der Programmdaten durch optiboot

Auf der linken Seite ist der Inhalt der Datei dargestellt, welche die Programmdaten (.hex) enthält. Rechts daneben ist der Inhalt des Flash-Speichers dargestellt, wie er vom Optiboot Bootloader geschrieben wird. An zwei Interruptvektor-Adressen wurde der Inhalt verändert. Einmal wurde auf dem Reset-Vektor 0 der Optiboot Bootloader eingetragen und zum anderen auf der „Ersatz"-Vektor Adresse 4 ein Sprung zum Start des Applikations-Programms eingetragen. Eine Schwierigkeit bei der Methode entsteht aber dadurch, daß die Programmdaten nach der Programmierung meistens zur Kontrolle zurückgelesen werden. Damit bei der Kontrolle keine Fehler gemeldet werden, gibt Optiboot nicht den wirklichen Inhalt der Interrupt-Vektortabelle zurück, sondern den Zustand vor seiner Manipulation. Die Sprung-Adresse im Reset-Vektor kann dafür aus den Daten des „Ersatz"-Interruptvektors rekonstruiert werden. Aber die ursprünglichen Daten des „Ersatz"-Interruptvektors wären verloren, da sie an keiner Stelle der Vektortabelle wiederzufinden sind. Optiboot benutzt zum Sichern des Original-Inhaltes des „Ersatz"-Vektors deshalb die beiden letzten Bytes des EEprom-Speichers. Damit ist eine Kontrolle der Programmdaten solange möglich, wie die beiden letzten

Bytes des EEproms nicht überschrieben werden. Selbst wenn die EEprom Daten überschrieben werden, bleibt der Bootloader funktionsfähig. Nur die Kontrolle (verify) der Programmdaten ist dann nicht mehr möglich. Bei der Adresse des „Ersatz"-Interruptvektors wird dann ein Fehler gemeldet.

Bei Prozessoren mit mehr als 8 kByte Flash Speicher werden zwei Befehlsworte für jeden Interrupt-Vektor vorgesehen. Normalerweise stehen auf diesen Doppelworten jmp Befehle mit den jeweiligen Sprungzielen. Auch diese Art der Vektortabelle wird von Optiboot berücksichtigt. Wenn aber für das Bindeprogramm (Linker avr-ld) die Option –relax verwendet wird, werden alle jmp Befehle durch die kürzeren rjmp Befehle ersetzt, wenn dies bei der jeweiligen Sprungadresse möglich ist. Dies wird derzeit nicht von **optiboot** berücksichtigt. Das Optiboot Programm geht fest davon aus, daß in der Vektortabelle jmp Befehle stehen, wenn mehr als 8 kByte Flash-Speicher vorhanden sind. Deshalb wird die VIRTUAL_BOOT_PARTITION Methode meistens nicht funktionieren, wenn die –relax Option beim Programmbinden benutzt wurde!

Weiter ist zu beachten, daß bei Benutzung der VIRTUAL_BOOT_PARTITION Option für Prozessoren, die auch die normale Bootloader Unterstützung bieten, das BOOTRST Fuse Bit nicht aktiviert wird. Der Grund hierfür ist, daß bei Benutzung der VIRTUAL_BOOT_PARTITION die Start-Adresse des Bootloaders auf einer anderen Adresse liegen kann wie bei der normalen Bootloader Unterstützung. Bei Benutzung der Option VIRTUAL_BOOT_PARTITION kann die Startadresse auf jedem Anfang einer Flashspeicher-Seite liegen. Bei der normalen Bootloader Unterstützung kann immer nur das einfache, doppelte, vierfache oder achtfache einer Mindest-Bootloadergröße berücksichtigt werden (BOOT_SZ Fuse-Bits), wie es in Abbildung 2.3 auf Seite 10 dargestellt wird.

3.7 Die Möglichkeiten der seriellen Schnittstelle mit der verwendeten Software

Das Programm für die Erzeugung und Verarbeitung der elektrischen Signale ist in AVR-Assembler geschrieben. Die Arbeitsweise ist von der Veröffentlichung von Atmel übernommen. Allerdings sind einige Besonderheiten eingebaut. So wird beispielsweise berücksichtigt, daß nicht für alle Port Adressen die speziellen Bitbefehle SBI, CBI bzw SBIC benutzt werden können. Das ist nur bis zur Adresse 31 (0x1f) möglich. Für etwas größere Port Adressen von 63 (0x3f) ist noch die Verwendung spezieller Einlese (IN) und Ausgabe (OUT) Befehle möglich. Darüber liegende Adressen können nur mit den LDS und STS Befehlen erreicht werden. Diese Befehle benötigen sowohl zwei Takte für die Ausführung und belegen auch den doppelten Speicherplatz im Flash (2 Worte oder 4 Byte). Die veränderte Taktzahl für einen Schleifendurchlauf ohne eine zusätzliche Verzögerung wird vom Programm automatisch ermittelt. Diese Taktzahl wird dann für die Berechnung der Verzögerungsschleife berücksichtigt, um eine korrekte Zeit für die Übertragung eines Bits zu erreichen. Die Diagramme 3.3 und 3.4 sollen die Erzeugung der Schleife mit dem C-Preprozessor verdeutlichen.

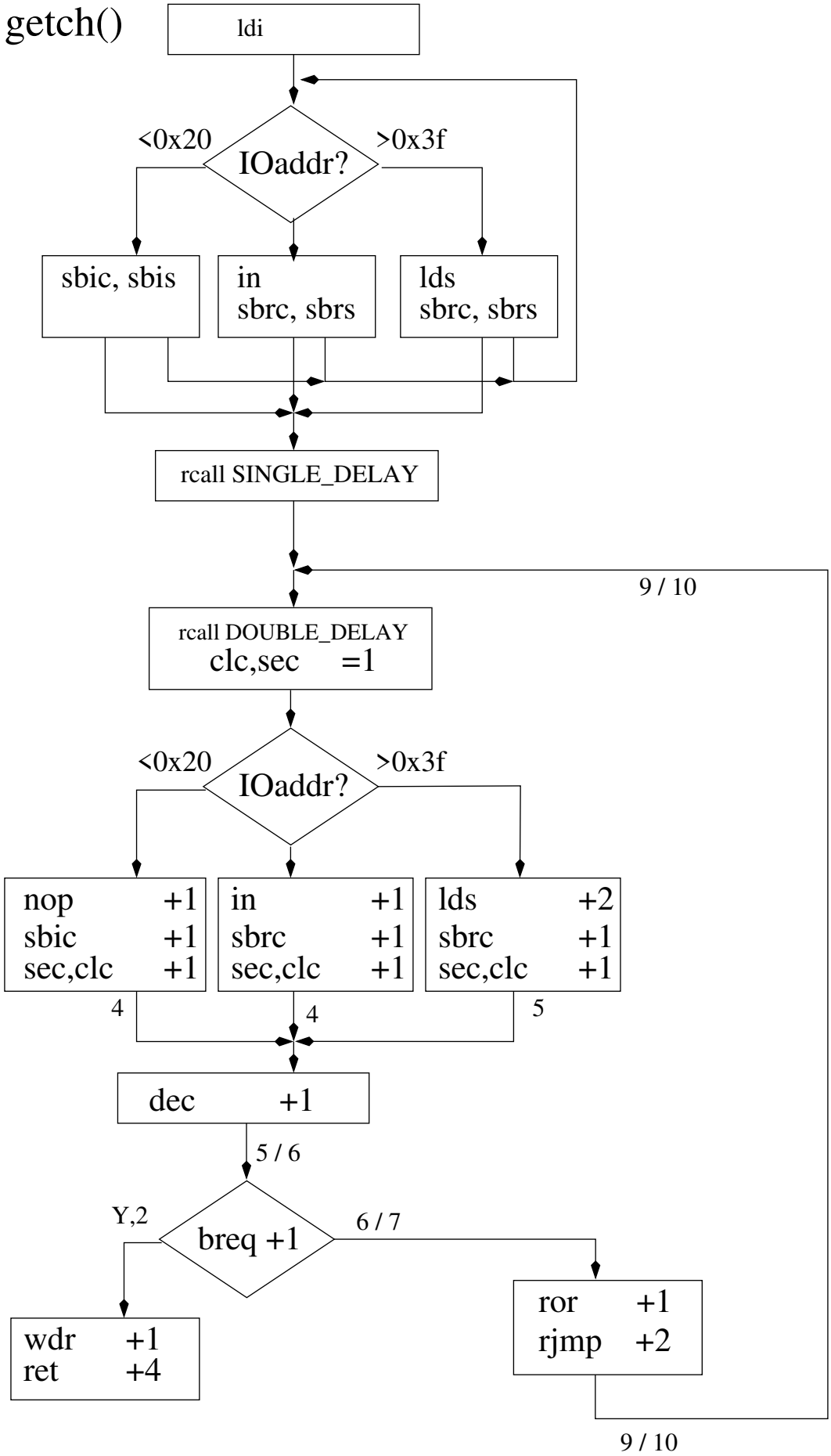


Abbildung 3.3. Mögliche Varianten der getch Funktion

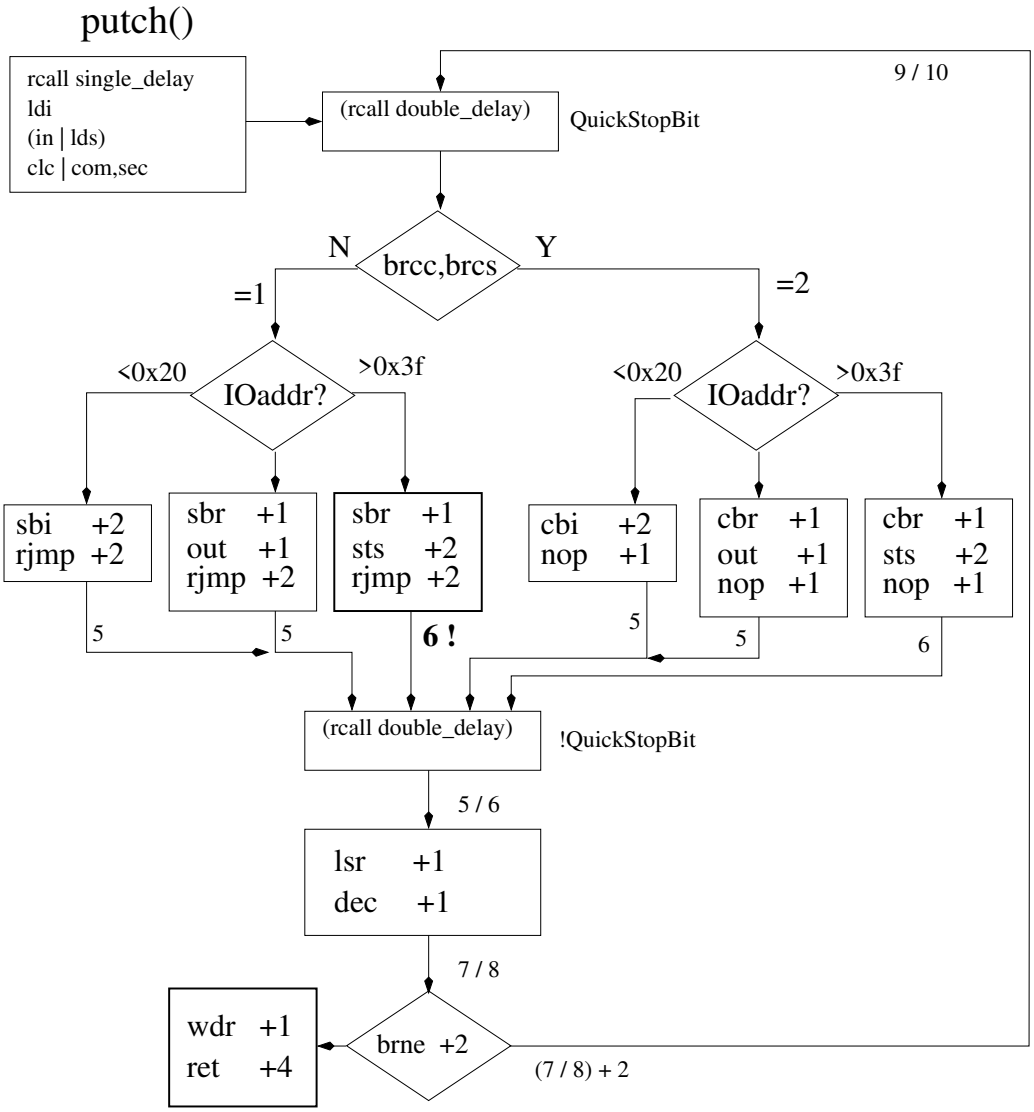


Abbildung 3.4. Mögliche Varianten der putch Funktion

Beide Schleifen werden so erzeugt, daß für die gleichen Bedingungen von INVERSE_UART und Adresse des verwendeten Ports die gleiche Taktzahl gebraucht wird. Damit können dann beide Funktionen die gleiche Funktion für die notwendige Verzögerung benutzen.

3.7.1 Berechnung der Verzögerungszeit

Für die Einlesefunktion getch() wird auch die halbe Baudzeit gebraucht. Von der Erkennung des Startbits wird 1.5 Mal die Baud-Zeit gewartet, um das erste Datenbit einzulesen. Deshalb ist die Basisschleife auf die halbe Zeitdauer einer Bitübertragung ausgelegt. Für die ganze Baudzeit wird diese Basisschleife in einer speziellen Weise zwei mal aufgerufen, damit exakt die doppelte Zeit eingehalten wird. Sollte durch die Bildung der Hälfte der Zeit und der nachfolgenden Verdopplung ein Takt verloren gegangen sein, wird dieser Takt durch einen zusätzlichen NOP Befehl ausgeglichen damit die Gesamtzeit für die Übertragung eines Bits auf einen Takt genau stimmt. Wenn bei der Schleifenzeit ein oder zwei Takte wegen der Zeitauflösung (3 Takte) fehlen, wird das automatisch durch einen zusätzlichen Befehl ausgeglichen. Natürlich geschieht das alles automatisch, da für die Berechnung nur die Schleifenzeit der Einlesefunktion (getch) bzw. Ausgabefunktion (putch), die Taktfrequenz des Prozessors und die gewünschte Baudrate bekannt sein muß. Auch die übrigen Parameter wie die Anzahl der Takte für einen Unterprogramm-Aufruf (RCALL, RET) sind für den Zielprozessor bekannt. Ein Nachteil der Basis-Verzögerungsschleife ist die begrenzte Anzahl Takte für die Verzöge-

rung. Es sind wegen des verwendeten 8-Bit Zählers maximal 256*3 Takte für die Scheifenverzögerung möglich. Dazu kommt noch der Unterprogrammaufruf von 7 Takten, wodurch sich für die halbe Verzögrungszeit dann 775 Takte ergeben. Dieser Wert muß verdoppelt werden (Verzögerung für eine ganze Bitzeit) und dazu noch die Schleifenzeit der Ausgabe- bzw. Eingabe dazugezählt werden. Somit ist die höchste erzielbare Verzögerungszeit 1559 Takte. Bei 16 MHz Taktfrequenz sind so mit maximal $97.4\mu s$ noch nicht einmal 9600 Baud ($104.17\mu s$) einstellbar. Würde man den Zähler der Zählschleife für die Verzögerungszeit von 8 Bit auf 16 Bit erweitern, hätte man eine noch schlechtere Auflösung als die 3 Takte der 8-Bit Variante. Außerdem käme die Schleife wahrscheinlich nicht wie die 8-Bit Variante ohne Veränderung des Übertragssignals (carry) aus. Dieses Problem habe ich durch die schrittweise Verdopplung der Verzögerungszeiten durch Verdopplung der Schleifenfunktion gelöst. Mit dem C-Preprozessor wird geprüft, ob der Anfangswert für die 8-Bit Zählschleife bei der gewählten Taktfrequenz des Prozessors und der Baudrate größer als 255 sein würde. Für diesen Fall wird die Berechnung für einen Doppelaufruf der Schleifenfunktion wiederholt. Wenn dann der Anfangswert für die Zählschleife immer noch zu hoch wäre, wird der Doppelaufruf noch einmal verdoppelt. Diese Prüfung wird derzeit bis zum Faktor 64 des Basiszeit fortgesetzt. Bei einer Taktfrequenz von 16 MHz oder 20 MHz sind so auf jeden Fall noch 300 Baud einstellbar. Für jede Verdopplung der Verzögerungszeit wird ein zusätzlicher Befehl (2 Byte) benötigt. Bei der maximal möglichen Zahl der Verdopplung werden dann 6 zusätzliche Befehle (12 Byte) im Flash belegt. Ein Ausgleich der durch die Vorteilung möglicherweise fehlenden Takte wird hier nicht durchgeführt, um nicht unnötig Platz im Flash zu belegen. Da die zusätzliche Vorteilung ja nur bei Bedarf erfolgt, bleibt der Fehler der Baudzeit deutlich unter 1% , weil die Zeit für eine Basisverzögerung auf einen Prozessor-Takt genau eingehalten wird. Die Basisschleife hat mindestens 127 Durchläufe mit etwa 381 Takten. Die doppelte Verzögerungszeit macht auch noch keinen Fehler wegen des „NOP" Ausgleichs. Somit bleibt der Fehler unter 1:762, also 0.13%. Probleme mit der Einhaltung der Baudrate bestehen also eher für hohe Baudraten, weil die Übertragungszeit für ein Bit nicht zum vorgegebenen Zeitraster durch den CPU-Takt paßt. Das gleiche Problem hat dann aber auch die Hardware UART-Schnittstelle. Ein Beispiel erläutert das Problem. Wenn man mit $16MHz$ CPU-Taktrate eine Baudrate von 230400 erzeugen möchte, stehen für das UART im besten Fall $2MHz$ Takt zur Verfügung. kann man entweder 8 Takte mit einer Taktzeit von $4\mu s$ oder 9 Takte (UBRR=8) mit einer Baud-Zeit von $4.5\mu s$ verwenden. Im ersten Fall ist die Baud-Zeit um 7.84% zu kurz, im zweiten Fall ist die Baud-Zeit um 3.68% zu lang.

3.7.2 Benutzung von mehr als einer seriellen Schnittstelle

Die Assembler Datei soft_uart.S ist dafür ausgelegt, von einer anderen Datei eingefügt (#include) zu werden, welche ein normales Assembler-Programm für die AVR-Familie beinhaltet. Für die **optiboot** Applikation wird das von der Datei optiboot.S gemacht. Die eingefügte Datei soft_uart.S benutzt viele Anweisungen für den GNU C-Preprocessor und fügt eine weitere Datei uart_delay.S ein, um eine Verzögerungsschleife für die gewünschte Baud-Rate zu generieren. Weil dieses Verfahren für uart_delay.S mit anderen Parametern mehrfach wiederholt werden kann, kann man bis zu 4 verschiedene Verzögerungsschleifen erzeugen lassen. Diese Vorgehensweise nutzt die Datei soft_uart.S für die Erzeugung der getch und putch Funktion. Für beide Funktionen wird die Datei uart_delay.S eingefügt. Aber für den zweiten #include wird meistens keine neue Verzögerungsschleife produziert, weil die Parameter beim zweiten Aufruf gleich sind. Nur bei verschiedenen Parametern wird eine neue Verzögerungsschleife generiert. Bitte beachten Sie, daß die Aufrufe der Verzögerungsschleifen mit C-Preprozessor Makro Namen versehen sind. Diese Makro Namen DOUBLE_DELAY_CALL und SINGLE_DELAY_CALL werden durch die richtigen Funktionsnamen ersetzt, wenn die #include Anweisung vor der Befehlsfolge für die serielle Eingabe- oder Ausgabe-Funktion steht.

Drei Konstanten müssen vor jeder Einfügung von uart_delay.S gesetzt sein, BAUD_RATE, F_CPU und LOOP_TICS. Die Konstante LOOP_TICS muß auf die Anzahl der Takte gesetzt

sein, die von einer seriellen Bit-Ausgabe in der Schleife gebraucht wird (normalerweise 9 Takte). Für jede erzeugte Verzögerungsfunktion wird die Anzahl der Verzögerungstakte in einer von vier verschiedenen Konstanten-Namen des C-Prozessors gespeichert, BIT_CLOCKS_0, BIT_CLOCKS_1, BIT_CLOCKS_2 und BIT_CLOCKS_3. Bevor eine neue Verzögerungsschleife erzeugt wird, prüft der C-Preprozessor, ob eine schon erzeugte Schleife zu den geforderten Parametern paßt. Weil die Datei soft_uart.S auch mit #include eingefügt werden muß, kann man diesen include auch mit anderen Parametern für eine andere serielle Schnittstelle wiederholen. Aber man muß eine weitere Konstante vorbesetzen, um die Namen der erzeugten Funktionen zu unterscheiden. Wenn Sie die Konstante SOFT_UART_NUMBER vor dem #include auf 1 setzen (#define SOFT_UART_NUMBER 1), wird die Funktion für die serielle Eingabe getch_1 und die Funktion für die serielle Ausgabe putch_1 heißen. Wenn man die Konstante NO_SOFT_UART_TX vor dem #include gesetzt hat, wird keine serielle Ausgabefunktion generiert. Gleiches gilt auch für die serielle Eingabe, wenn die Konstante NO_SOFT_UART_RX vor dem #include gesetzt ist.

3.7.3 Serielle Eingabe und Ausgabe über nur einen AVR Pin

Manchmal ist es sinnvoll, die serielle Datenkommunikation nur über einen Pin zu betreiben, um einen der wenigen IO-Pins bei kleinen AVRs freizuschalten. Mit einer speziellen Schaltungstechnik kann erreicht werden, daß in den Ausgabe-Pausen der seriellen Ausgabe ein Einlesen von Daten möglich ist. Bei der hier bei **optiboot** gewählten Software-Lösung ist ohnehin nur ein Halb-Duplex Betrieb möglich. Es kann also zu einer Zeit nur entweder Daten gesendet oder Daten empfangen werden. Normalerweise wird der Ausgabepin mit der TX-Funktion in den Sendepausen auf High Pegel geschaltet, was ein Einlesen von Daten auf dem gleichen Pin verhindert. Wenn aber der TX Ausgabepin statt auf den High Pegel in den Sendepausen auf den Eingabemodus zurückgeschaltet wird, kann ein externer Pull-Up Widerstand den erforderlichen High Pegel erzeugen. Im Gegensatz zum festen High Pegel kann jetzt aber ein extern angeschlossenes TX-Signal den Pegel für die serielle Einlesefunktion auf Low Pegel ziehen. Ein serieller Widerstand in der Verbindung von dem gemeinsamen TX/RX Pin des AVR mit dem TX Ausgang kann die Funktion des Pull-Up Widerstandes übernehmen, da der Ruhezustand dieser Schnittstelle auf High Pegel liegt. Außerdem sorgt dieser serielle Widerstand für eine Strombegrenzung, falls doch einmal beide TX Schnittstellen gleichzeitig senden. Damit die TX-Ausgabe des AVR von der externen seriellen Schnittstelle gelesen werden kann, muß der RX-Eingang direkt mit dem gemeinsamen TX/RX verbunden werden. Die Abbildung 3.5 soll die einfachste Verbindung veranschaulichen.

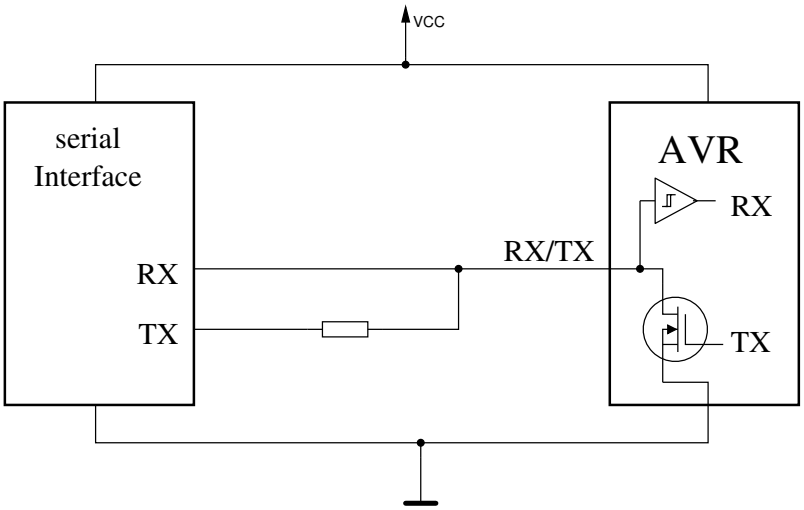
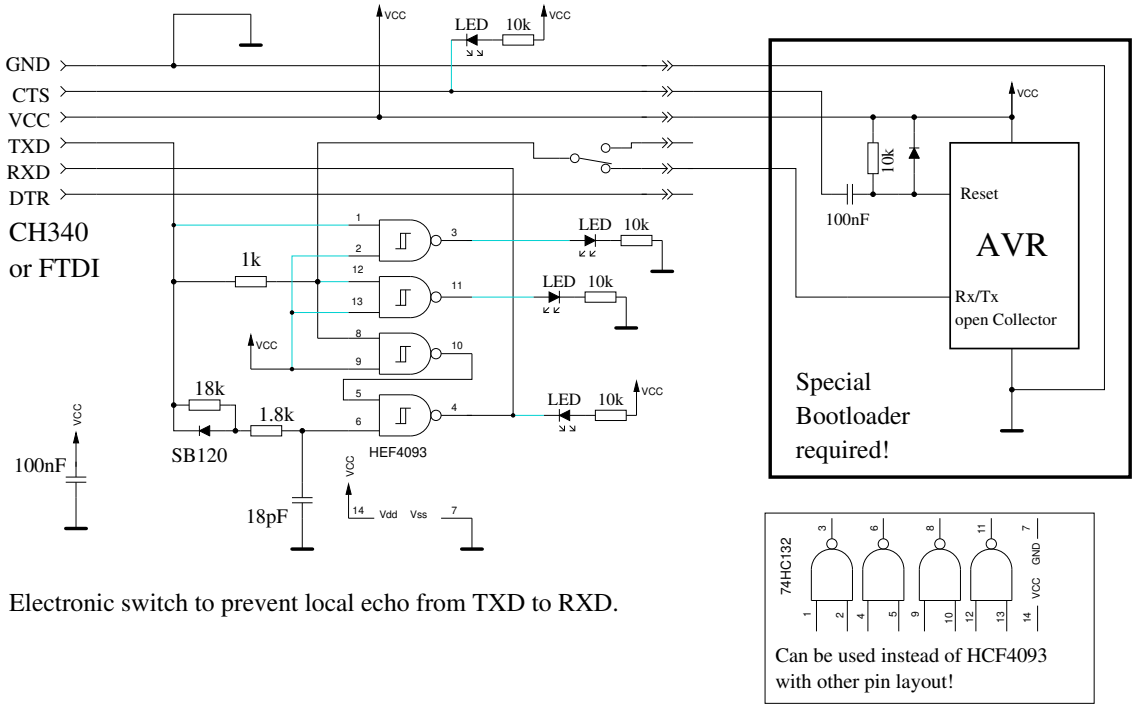


Abbildung 3.5. Mögliche serielle Verbindung zum AVR mit einem Pin

Das Problem besteht nun darin, daß die meistens full-duplex fähige externe serielle Schnittstelle

die eigene Ausgabe zurückliest. Darauf ist ein Kommunikationsprogramm wie avrdude auf der externen Seite im allgemeinen nicht vorbereitet.

Es gibt nun die Möglichkeit, das Programm auf der externen Seite entsprechend anzupassen. Ich halte das für keine gute Idee, da die angepasste Version dann entweder keine Wartung mehr erhält oder es müßte die Änderung für neue Versionen ständig wiederholt werden. Aus diesem Grund bevorzuge ich eine Hardware-Lösung. Eine elektronische Schaltung, die zwischen die beiden Endpunkte der seriellen Schnittstelle geschaltet wird, muß unterscheiden, ob die Daten von der externen Schnittstelle oder von der AVR Seite gesendet werden. Ein entsprechender Schaltungsvorschlag zeigt die Abbildung 3.6.

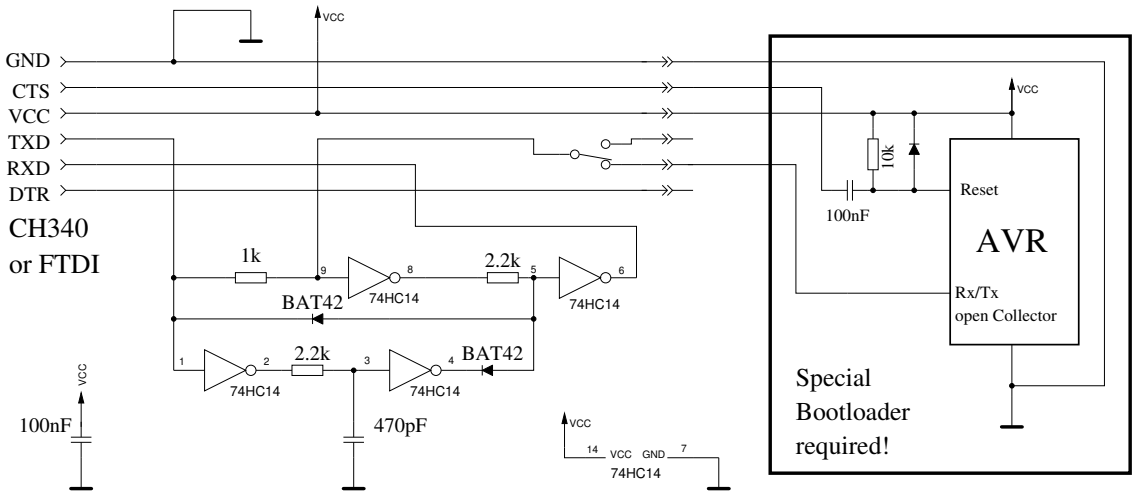


Electronic switch to prevent local echo from TXD to RXD.

Abbildung 3.6. Schaltung zur Unterdrückung des Echos

Die Tx Daten des PC werden über den $1k\Omega$ Widerstand an den Empfänger des AVR weitergereicht. Wenn der AVR Daten zurücksendet, kann dieser den Pegel nahe 0V schalten, weil der $1k\Omega$ Widerstand das nicht verhindern kann, obwohl der TX Pegel des PC's auf VCC geschaltet bleibt. Über die beiden Nand-Gatter des HEF4093 werden die Daten etwas verzögert an den PC RX Eingang weitergereicht. Über den 0V Pegel der PC-TX Seite wird genau dieses Weiterreichen der Daten verhindert und der RX Pegel bleibt auf VCC, also auf dem Ruhesignal der seriellen Übertragung. Probleme machen eigentlich nur die Flankenwechsel des TX-Signals. Deshalb wird mit der Schaltung aus Diode, zwei Widerständen und einem $18pF$ Kondensator die runtergehende Flanke schneller zum NAND-Eingang 6 weitergeleitet als die raufgehende Flanke. Nur zwei NAND Gatter sind für die Funktion erforderlich, die anderen beiden werden nur zur Anzeige der RX/TX Aktivität benutzt. Mit dieser Schaltung kann man ein unverändertes Übertragungsprogramm wie avrdude benutzen. Die Schaltung ist weitgehend unabhängig von der benutzen Baudrate. Lediglich bei sehr hohen Baud-Raten erwarte ich Probleme. Bei 115200 Baud läuft die Schaltung aber einwandfrei. Sehr viel höhere Baudraten sind bei der Softwarelösung sowieso oft nicht möglich.

Die Abbildung 3.7 zeigt eine alternative Schaltung mit einem 74HC14 Hex Inverter mit Schmitt-Trigger Eingängen.



Electronic switch to prevent local echo from TXD to RXD.

Abbildung 3.7. Alternative Schaltung zur Unterdrückung des Echos

3.7.4 Benutzung der automatischen Baudraten-Bestimmung

Wenn für die Erzeugung des Bootloaders eine Baudrate unter $100Baud$ angegeben wird, wird die Baudrate aus dem ersten empfangenen Zeichen automatisch ermittelt. Zum Beginn einer Übertragung mit dem STK500 Protokoll sendet der Rechner das Kommando STK_GET_SYNC (0x30) gefolgt vom Steuerzeichen CRC_EOP (0x20). Bei der seriellen Übertragung ist als Ruhezustand der High-Pegel (1) verabredet. Die Übertragung beginnt dann mit einen Start-Bit auf Low-Pegel (0). Unmittelbar darauf wird die verabredete Zahl der Datenbits (8) beginnend mit dem niederwertigsten Bit übertragen und dann die Übertragung mit einem oder mehreren Stoppbit (1) abgeschlossen. Der Pegel jedes einzelnen Bits wird für die verabredete Baud-Zeit beibehalten. Unmittelbar nach dem Ende der Stoppbit Übertragung kann das nächste Start-Bit folgen. Es kann aber auch eine Pause mit nicht definierter Zeit bis zur Übertragung der nächsten Bitfolge auf das Stoppbit folgen. In der Abbildung 3.8 ist das erwartete Ergebnis für die möglichen Zeitmessungen dargestellt. Auf der Zeitachse „t" sind vier mögliche Startpositionen für eine vollständige Zeitmessung dargestellt. Der Counter wird jeweils bei einer erkannten 1 - 0 Flanke gestartet. Die richtige Startposition für das STK_GET_SYNC Zeichen ist mit der „1" markiert. Die Marken „2", „3" und „4" zeigen die Situation für drei falsche Startpositionen. Bei der Marke „2" und „4" wird fälschlicherweise eine 1-0 Flanke in der Datenübertragung als Start-Bit angenommen. Bei der Marke „3" wird zwar ein Start-Bit richtig erkannt, es ist aber das falsche Zeichen (CRC_EOP). Es können immer dann die falschen Start-Bits erkannt werden, wenn der Bootloader erst gestartet wird, wenn die Datenübertragung schon läuft. In der Abbildung 3.8 sind die erwarteten Zählerstände für die 4 möglichen Startfälle angegeben. Dabei bedeutet das „b" den Zählerstand für ein Bit (Baud-Dauer), das „d" steht für eine mögliche Zeitverzögerung zwischen dem Ende der Stoppbit-Übertragung bis zum Beginn eines neuen Startbits. Das „D" steht für eine zu erwartende lange Zeitverzögerung bis zum nächsten Startbit. Die Sendesequenz ist hier zu Ende und der Rechner wartet auf eine Antwort des AVRs. In der Rx-Reihe sind die Datenbits von „0" (kleinstes Bit) bis „7" (höchstes Bit) nummeriert. Das Start-Bit ist mit „A" und das Stoppbit mit „E" gekennzeichnet. Das erste Byte ist die Codierung des STK_GET_SYNC Kommandos und das zweite Byte die Kodierung des Steuerzeichen CRC_EOP.

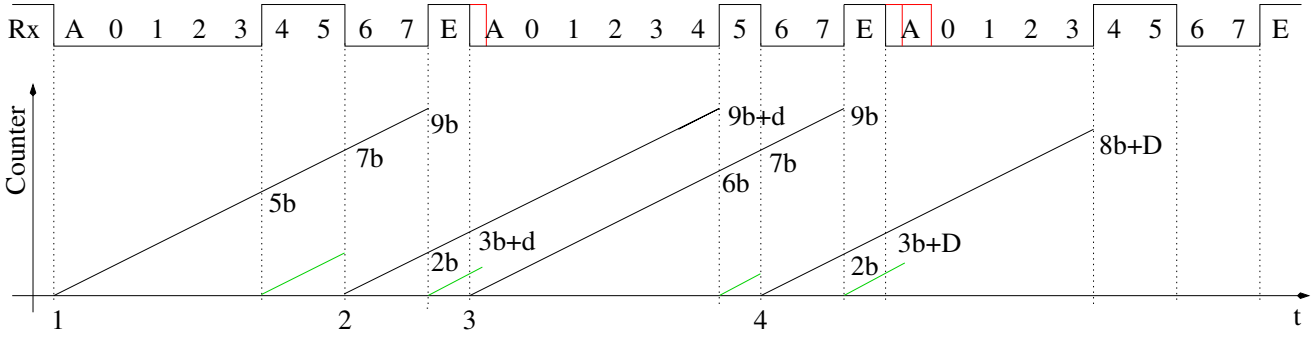


Abbildung 3.8. Mögliche Zeitmessungen für die STK_GET_SYNC Bitfolge

- Einfachste Form der Baudraten Messung, BAUD_RATE 20-29

  Die einfachste Form der Baudraten-Messung beachtet die Möglichkeit der falschen Startbit Wahl überhaupt nicht. Es wird fest davon ausgegangen, daß das erkannt Startbit zum STK_-GET_SYNC Zeichen gehört. Nach der Startbit Erkennung wird bis zur nächsten 0-1 Flanke gewartet und dann ein 16-Bit Zähler mit der Frequenz F_CPU/8 bei 0 gestartet. Wenn die Einerstelle der Baudrate größer als 4 angegeben wird, wird der 16-Bit Zähler jedoch mit der F_CPU Frequenz gestartet. Bei der darauf folgenden 1-0 Flanke wird der Zählerstand abgelesen und halbiert als Frequenzteiler des UARTs genommen. Die Halbierung ist notwendig, weil die Zeit von 2 Datenbits (4-5) gemessen wurde. Eigentlich muß noch 1 vom Ergebnis für den Frequenzteiler abgezogen werden. Aber bei der Halbierung sollte gerundet werden, also 1 auf den Zählerstand addiert werden. Anstelle der Subtraktion der 1 vom Divisions-Ergebnis kann man auch 2 vom Zählerstand subtrahieren. Die Methode funktioniert ausreichend gut, wenn automatisch kurz vor dem Übertragungs-Start ein Reset des AVRs und damit ein Neustart des Bootloaders ausgelöst wird. Dies ist dann der Fall, wenn das DTR (Data Terminal Ready) Signal für die Erzeugung eines Reset-Signals benutzt wird. Man kann auch ohne das automatisch generierte Reset-Signal erfolgreich sein, wenn eine Reset-Taste am AVR angeschlossen ist, die solange gedrückt gehalten wird, bis das Übertragungsprogramm gestartet ist. Hier ist aber etwas Gefühl für den richtigen Loslass-Zeitpunkt des Tasters erforderlich. Das Bootloader-Programm wartet nur maximal die Zeit auf den Beginn der Übertragung, die über der Watchdog-Timer vorgegeben ist. Wenn der Watchdog-Timer einen Reset auslöst, wird normalerweise das Anwenderprogramm gestartet. Nur wenn noch kein Anwenderprogramm geladen wurde, startet der Bootloader erneut und versucht es noch einmal.

- Verbesserte Form der Baudraten Messung, BAUD_RATE 30-39

  Wie bei der einfachsten Form der Baudraten-Messung wird hier auf eine Prüfung der Bit-Wechselzeiten verzichet. Es wird nur weiter auf die nächste 0-1 Flanke des RX-Signals gewartet und erst dann der Zähler gelesen. Dadurch wird die Zeit von vier Datenbits (4-7) gemessen, die dann für die Berechnung der Baudrate verwendet wird. Da die Zeiterfassung durch die Abfrage-Schleife (polling) unpräzise ist, wirkt dieser mögliche Zeitfehler weniger, wenn er durch vier statt durch zwei geteilt wird. Daher sollte diese Form immer dann gewählt werden, wenn es auf wenig Speicherbelegung ankommt. Gegenüber der einfachsten Form werden nur 4 Byte mehr belegt.

- Einfachste Form der Baudraten Messung mit Zeitüberwachung, BAUD_RATE 40-49

  Hier wird die gleiche Methode der Baudraten-Bestimmung angewendet, wie bei der einfachsten Form. Nur wird das mögliche Überlauf-Ereignis des Zählers zusätzlich überwacht, was die Abfrage-Schleife zusätzlich verlangsamt. Durch den längeren Schleifen-Zyklus wird die Zeitmessung weniger exakt als bei der Methode ohne Zeitüberwachung zu erwarten ist. Daher ist diese Version nicht bei der Verwendung hoher Baudraten zu empfehlen. Bei dem Überlauf-Ereignis wird sofort mit der Suche nach einem neuen RX-Start-Bit begonnen. Ohne diese Begrenzung

würde das Programm in der Schleife verweilen, bis entweder ein Start-Bit entdeckt wird oder die Schleife durch einen Reset vom Watchdog-Timer abgebrochen wird. Wahrscheinlich wird dadurch das Problem mit der Start-Position „4" im Diagramm 3.8 etwas verbessert. Aber auch der Neustart der Start-Bit Suche würde einen Reset vom Watchdog-Timer nur geringfügig verzögern. Bei der Suche nach dem Start-Bit darf der Watchdog-Timer nicht zurückgesetzt werden, weil sonst kein Anwenderprogramm mehr gestartet würde. Deshalb muß die Zeit für den Watchdog-Timer größer sein als die Zeit bis zur Wiederholung der STK_GET_SYNC Sequenz ohne eine Antwort auf die vorausgehende STK_GET_SYNC Sequenz.

- Verbesserte Form der Baudraten Messung mit Zeitüberwachung, BAUD_RATE 50-59

  Die Messung der Baudrate erfolgt genau so wie bei der verbesserten Methode 30-39). Nur wird wie bei der Methode 40-49 beschrieben zusätzlich eine Begrenzung der Wartezeit durchgeführt. Weil der Zählerstand erst bei der 0-1 Flanke gelesen wird und da der Zählerüberlauf nur bei der 1-0 Flanke geprüft wird, ist die ermittelte Baud-Zeit nicht besser wie bei der verbesserten Methode ohne die Zeitüberwachung, die auch die Zeit von vier Datenbits (4-7) als Basis verwendet.

- Aufwendige Form der Baudraten Messung, BAUD_RATE 60-69

  Bei dieser Methode wird der Zähler mit der Erkennung eines Start-Bits sofort bei 0 gestartet und bei den folgenden 3 Bitwechseln der Zählerstand abgelesen. Dadurch ist die Zeitfolge der Bitwechsel für das STK_GET_SYNC Zeichen vollständig bekannt. Vom Programm wird die Zeitfolge auf Plausibilität geprüft. Als erster Test wird geprüft, ob die Differenz zwischen der 3. Zählerlesung und der ersten Zählerlesung kleiner ist als die erste Zählerlesung. Für den richtigen Startpunkt „1" ist das der Fall ((9b - 5b) < 5b). Für den falschen Startpunkt „3" ist das aber leider auch der Fall ((9b - 6b) < 6b). Mit großer Wahrscheinlichkeit ist das für den Startpunkt „2" aber nicht der Fall ((9b -3b) < (3b+d)), wenn die Zeitverzögerung des zweiten Startbits „d" hinreichend klein ist. Für den falschen Startpunkt „4" hilft ohnehin nur eine Zeitüberwachung beim Warten auf den 2. Bitwechsel, da die Wiederholung der STK_GET_SYNC Sequenz erst nach längerer Wartezeit („D") auf die Antwort erfolgt. Mit einem zweiten Test wird geprüft, ob die Differenz zwischen der 3. Zählerlesung und der 2. Zählerlesung nicht deutlich größer ist als die Differenz zwischen der 2. Zählerlesung und der ersten. Für den richtigen Startpunkt „1" ergibt sich folgender Vergleich: ((9b-7b) < (7b-5b+4)) oder (2b < 2b+4). Für den falschen Startpunkt „2" sieht der Vergleich so aus: ((9b-3b) < (3b+d-2b+4)) oder (6b < (b+d+4)). Für den falschen Startpunkt „3" ergibt sich der Vergleich so: ((9b-7b) < (7b-6b+4)) oder (2b < (b+4)). Diese Prüfungen sind relativ sicher, erfordern aber auch relativ viel zusätzlichen Speicherplatz für das Programm. Diese Methode ist besonders dann zu empfehlen, wenn ohnehin mindestens 1024 Byte für den Bootloader reserviert sind (Bootloader Page Size). Für die Berechnung des UART Frequenzteilers wird die Differenz der 3. Zählerlesung und der 2. Zählerlesung halbiert (9b-7b-1)/2 = (2b+1)/2-1.

- Aufwendige Form der Baudraten Messung, BAUD_RATE 70-79

  Alle Zeitlesungen und Kontrollen werden wie bei der Methode mit Baud_RATE 60-69 durchgeführt. Lediglich für die Berechnung des UART Frequenzteilers wird die Zeit von 4 Bits benutzt. Somit ergibt sich für den richtigen Startpunkt „1" die folgende Berechnung (9b-5b-2)/4 = (4b+2)/4-1.

- Aufwendige Form der Baudraten Messung, BAUD_RATE 80-89 Alle Zeitlesungen und Kontrollen werden wie bei der Methode mit Baud_RATE 60-69 durchgeführt. Lediglich für die Berechnung des UART Frequenzteilers wird die Zeit von 9 Bits benutzt. Somit ergibt sich für den richtigen Startpunkt „1" die folgende Berechnung (9b-4)/9 = (9b+5)/9-1.

- Einfachste Form der Baudraten Messung, BAUD_RATE 10-19 Die Zeitmessung von 9 seriell übertragenen Bits ist auch ohne Kontrolle der Daten möglich. Eigentlich sollte aber die Zeitmessung von 4 Bits ausreichend sein.

Für alle Methoden der Baudraten-Bestimmung würde ich die Zusatzfunktion des LED-Blinkens weglassen, um die Erkennung des Start-Bits nicht zu verzögern. Leider braucht auch die simpelste Form der Baudraten-Messung so viel Programmspeicher, daß der Bootloader nicht in 512 Byte Flash passt, wenn zusammen mit der EEprom-Unterstützung (SUPPORT_EEPROM=1) auch das LED-Blinken gewählt wird. Bei einigen Prozessoren kann als Zusatzfunktion noch die LED_DATA_FLASH gewählt werden ohne die 512 Byte Grenze zu überschreiten. Wenn die Grenze von 512 Byte wegen benötigter Funktionen ohnehin überschritten wird, sind auch mit der Baudratenmessung alle Zusatzfunktionen wieder problemlos wählbar, da dann der doppeltre Platz von 1024 Byte zur Verfügung steht. Zusammen mit der SOFT_UART Funktion kann die Baudratenbestimmung nur mit der Assembler-Version des Optiboot Programms benutzt werden. Die C-Variante unterstützt die Baudratenmessung mit der automatischen Einstellung der Baudrate nur für einen Hardware-UART.

Die nachfolgende Tabelle 3.6 faßt die verschiedenen Möglichkeiten noch einmal zusammen. Die angegebenen Programmgrößen in Bytes beziehen sich auf einen ATmega328 ohne LED Blinkfunktion, aber mit EEprom Unterstützung. Die Programmgrößen in Klammern ergeben sich beim Betrieb der seriellen Schnittstelle mit Software.

BAUD_RATE	Bit Anzahl	Zeit Basis	Time Limit	Kontrolle	mega328 Größe (HW)
10-14	9	clk/8	-	simpel	502
15-19	9	clk	-	simpel	502
20-24	2	clk/8	-	simpel	490
25-29	2	clk	-	simpel	494
30-34	4	clk/8	-	simpel	494
35-39	4	clk	-	simpel	494
40-44	2	clk/8	Ja	simpel	496
45-49	2	clk	Ja	simpel	500
50-54	4	clk/8	Ja	simpel	500
55-59	4	clk	Ja	simpel	500
60-64	2	clk/8	Ja	Komplex	550
65-69	2	clk	Ja	Komplex	554
70-74	4	clk/8	Ja	Komplex	554
75-79	4	clk	Ja	Komplex	554
80-84 90-94	9	clk/8	Ja	Komplex	564
85-89 95-99	9	clk	Ja	Komplex	564

Tabelle 3.6. Einstellmöglichkeit für die Baudratenbestimmung.

Bei der Baudratenmessung für die Software UART Funktion bestehen einige Unterschiede und Besonderheiten gegenüber der Lösungen für die Hardware UART Schnittstelle. Normalerweise für wird für die halbe Baud-Zeit ein 8-Bit Zähler in einer Verzögerungsschleife benutzt. Mit diesem Zähler kann die Anzahl der CPU-Takte für die Verzögerung wegen der Schleifen-Durchlaufzeit nur auf 3 Takte genau eingestellt werden. Wegen des doppelten Aufrufs der Schleife für die volle Baud-Zeit muß man die Auflösung ebenfalls verdoppeln, also 6 Takte. Bei einer fest gewählten Baudrate wird der

dadurch entstehende Fehler im Programm ausgeglichen. Das ist hier nicht möglich, da die Baudrate ja nicht im voraus bekannt ist. Um auf jeden Fall die bestmögliche Rundung des eingestellten Wertes zu ermöglichen, wird für die Zeitmessung der Zähler besser bei der vollen CPU-Taktrate (F_CPU) betrieben. Für den Hardware-UART reicht für den Zähler ein Betrieb mit F_CPU/8, da die gleiche Frequenz auch vom UART benutzt wird und die Zeit von 2 oder 4 Takten gemessen wird.

Bedingt durch den 8-Bit Zähler ergibt sich abhängig von der CPU-Taktrate eine maximale Verzögerungszeit und damit eine Mindest-Baudrate, die deutlich höher ist als die Mindest-Baudrate für den Hardware-UART. Durch die beschränkte Auflösung der Perioden-Einstellung für die Baudrate ergibt sich auch eine obere Baudrate, für die ein Fehler von kleiner als 4% garantiert werden kann. Bei Betrieb des AVR mit einem RC-Generator ist oft die Taktrate selbst schon fehlerbehaftet. Bei der Messung der Baudrate ist dieser Fehler dann schon vom Zähler berücksichtigt, der Fehler durch die beschränkte Einstellmöglichkeit ist dann aber nicht im voraus berechenbar. An einem Beispiel für den Hardware-UART möchte ich zeigen, was ich damit meine. Die Baudrate von $250kHz$ ist bei einer Taktrate von genau $8MHz$ benutzbar. Dazu wird der Teiler für die Baudraten-Erzeugung auf 32=4*8 eingestellt, wobei sich dann kein Fehler ergibt. Wenn ich nun eine abweichende Taktrate von $7.6MHz$ annehme, ist der bestmögliche Teiler ebenfalls 32. Dadurch ergibt sich die Baudrate mit dem gleichen Fehler wie die Taktrate von $237.5kHz$, in beiden Fällen sind es -5%. Um wenigstens die untere Baud-Grenze mit der Software-UART Lösung überwinden zu können, wird bei allen vorgegebenen Baud-Werten unter 100, die ungerade sind, wird eine Verzögerungsschleife mit 15-Bit benutzt. Leider beträgt dann die Durchlaufzeit der Schleife dann 5 Takte. Durch den doppelten Aufruf der Verzögerungsschleife beträgt die Auflösung nur noch 10 Takte und damit etwas schlechter als die Hardware-UART Lösung durch den 8:1 Vorteiler. Daher sollte man diese Option nur dann benutzen, wenn man keine sehr hohen Baudraten benutzen möchte. Die folgende Tabelle 3.7 soll den Einsatzbereich der verschiedenen Optionen bei der Betriebsfrequenz $8MHz$ zeigen. In der Tabelle ist der Zeit-Fehler nicht berücksichtigt, der bei der Messung der Baudrate durch die Abfrageschleife des RX-Signals entstehen kann.

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

Tabelle 3.7. Grenzen für die automatische Baudrate bei 8MHz Takt.

Für die Fehlergrenze von 2%, die normalerweise als Grenze für das Gelingen angenommen wird, ist angenommen worden, daß beide Gegenstellen einen Fehler in unterschiedlicher Richtung haben können. Da bei der Autobaud-Funktion der Fehler der Gegenstelle mit gemessen wird, kann man hier eine doppelte Obergrenze für den Fehler von 4% annehmen. Bei Benutzung der Hardware-UART

Schnittstelle sind unabhängig der gewählten Option für die Messung die Standard Baudraten 1200, 2400, 4800, 9600, 19200, 38400 und 57600 benutzbar. Höhere Baudraten sind nicht mehr sicher bei der Benutzung, obwohl auch $115.2kBaud$ bei Tests oft funktionierte. Ebenso war ein Test mit 115.2 kBaud bei $16MHz$ Quarz-Betrieb erfolgreich, selbst bei Benutzung der Software UART Lösung mit der 15-Bit Verzögerungsschleife. Wenn für die Software-UART Lösung die 8-Bit Verzögerungsschleife benutzt wird, kann die Baudrate 1200, 2400 und 4800 nicht benutzt werden. Erst bei Benutzung der 15-Bit Verzögerungsschleife sind die unteren Baudraten mit Software UART benutzbar. Der Unterschied zwischen der 47 und der 67 bzw. 77 Einstellung ergibt sich durch den verwendeten 16-Bit Zähler. Bei der Einstellung 49 wird nur die Zeit von 2 Datenbits gemessen, bei den Einstellungen über 59 wird die komplette Byte-Sequenz vom Start-Bit bis zum Stoppbit mit dem 16-Bit Zähler gemessen. Baudraten unter 9600 Baud werden wahrscheinlich aber ohnehin eher selten benutzt. Die Bereiche für die Baudrate verschieben sich natürlich mit der verwendeten Taktrate für den Prozessor.

Für die Software UART Schnittstelle ergeben sich zahlreiche Einstellmöglichkeiten, die einerseits die Messung der Baudzeit betreffen, aber auch die Art der Verzögerungsberechnung betreffen. In der Tabelle 3.8 sind die resultierenden Programmlängen des Bootloaders für einen ATmega328 gegenüber gestellt. Die Programme in den Spalten _0 und _5, also alle mit den Ziffern 0 oder 5 endenden Baudraten, sind leider in der Praxis nicht benutzbar. Der Grund ist, daß die Division durch fortlaufende Subtraktion zu langsam ist, um sicher in der Zeit des Stoppbits fertig zu werden. Dann wird der Anfang des nächsten Start-Bits möglicherweise verpasst. Abhilfe würde hier schaffen, wenn der Sender der Daten (avrdude) die Daten mit 2 Stopp-Bits senden würde. Weil das so von der avrdude Version 6.3 nicht vorgesehen ist, benutzen die anderen Spalten eine beschleunigte Subtraktions-Schleife, die dann aber ein längeres Programm bewirkt. Wenn möglich wird statt der Subtraktions-Schleife die schnellere Schiebe-Operationen für die Division durch 2-er Potenzen benutzt. Um das zu ermöglichen, sind in den Spalten _3, _4, _8 und _9 die Anzahl der Takte für einen Durchlauf der Verzögerungszeit-Schleife auf 2-er Potenzen verlängert (8T oder 16T). Leider ergibt sich nur dann ein kürzeres Programm, wenn die Anzahl der Schiebe-Operationen kleiner als 3 ist und damit auf eine Scheife verzichtet werden kann. Man kann aber mit der Option NO_EARLY_PAGE_ERASE 14 bytes zusätzlich einzusparen und damit etwa 30 Varianten mehr in 512 Byte Flash unterbringen.

BAUD_RATE ,Meßbasis	_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

Tabelle 3.8. Bootloader-Programmlängen mit automatischer Baudratenwahl für Software UART

Abschließend möchte ich die Ergebnisse von Tests mit einen ATmega1281 bei etwa $8MHz$ Frequenz mit externem Takt vorstellen, die ich bei und oberhalb der garantierten Baudrate durchgeführt habe. Die Frequenz des externen Takts habe ich dabei in Schritten von $100kHz$ von $7MHz$ bis $9MHz$ verstellt und jeweils ein kleines Testprogramm geladen. Mit der Software UART Lösung gelangen alle Tests, ein kleines Anwenderprogramm zu laden. Weil die Ergebnisse bei der Baudrate 57600 so ermutigend waren, habe ich zusätzlich die Tabelle um Tests bei 115200 Baud ergänzt.

Freq. MHz	HW-UART 57600	SW-UART 57600		HW-UART 115200	SW-UART 115200	
	Mode 72	Mode 72	Mode 77	Mode 56	Mode 52	Mode 57
7.0	OK	OK	OK	10/10	0/1	10/10
7.1	OK	OK	OK	10/10	2/3	10/10
7.2	OK	OK	OK	10/5	10/2	10/3
7.3	OK	OK	OK	0/0	10/0	10/0
7.4	OK	OK	OK	0/0	2/2	0/0
7.5	OK	OK	OK	0/0	0/0	0/2
7.6	OK	OK	OK	1/0	0/0	0/2
7.7	OK	OK	OK	1/0	2/0	1/0
7.8	OK	OK	OK	10/10	4/6	0/0
7.9	OK	OK	OK	10/10	10/3	10/10
8.0	OK	OK	OK	10/10	5/0	10/10
8.1	OK	OK	OK	10/10	1/0	10/10
8.2	OK	OK	OK	1/0	2/0	10/6
8.3	OK	OK	OK	0/0	0/1	10/1
8.4	OK	OK	OK	0/0	0/2	10/0
8.5	OK	OK	OK	0/0	6/4	0/0
8.6	OK	OK	OK	1/0	10/0	0/0
8.7	OK	OK	OK	0/0	0/0	0/0
8.8	OK	OK	OK	10/10	3/0	0/0
8.9	OK	OK	OK	10/10	0/1	0/0
9.0	OK	OK	OK	10/8	1/1	0/1

Tabelle 3.9. Test für die automatische Baudrate bei 8MHz Takt.

Für den Betriebsmodus 82 habe ich mit dem Software UART auch die ungeraden OSCCAL_CORR Einstellungen ohne erkennbare Schwierigkeiten geprüft. Auch mit dem einfachsten Betriebsmodus 42 waren bei den geraden OSCCAL_CORR Einstellungen der Tabelle keine Ausfälle zu verzeichnen. Erst bei der für diese CPU-Frequenz extrem hohen Baudrate von $115.2k$ waren für den Betriebsmodus 52 vier Ausfälle und mit dem Betriebsmodus 57 neun Ausfälle zu beobachten. Die Zunahme der Ausfälle bei Betriebsart 57 mit der 15-Bit Zeitschleife war wegen der groberen Rasterung der einstellbaren Zeiten zu erwarten.

Ein Test mit einer chinesischen Arduino UNO Platine, die als USB-seriell Wandler einen CH340G Chip verwendet, lief übrigens nur bis 38400 Baud einwandfrei. Bei höheren Baudraten machte das Zurücklesen der Programmdaten Probleme. Wahrscheinlich wird das letzte Byte eines Datenpakets nicht zum Rechner übertragen und die Kommunikation bleibt hängen. Bei einer Arduino UNO Platine mit Mega16U2 Controller als USB-seriell Wandler trat das Problem nicht auf. Mit dieser Platine war sogar ein Betrieb der seriellen Schnittstelle bei $230.4kBaud$ möglich, wahrscheinlich weil sowohl der ATmega328p als auch der Mega16U2 die gleiche tatsächliche Baudrate von $250kBaud$ benutzen.

3.7.5 Besonderheiten der seriellen Schnittstelle

Normalerweise wird in **optiboot** bei der seriellen Ausgabe ein verlängertes Stoppbit ausgegeben. Die Hardware UART Schnittstelle schaltet ein zweites Stoppbit ein, bei der Software wird nur ein halbes zusätzliches Stoppbit berücksichtigt. Dies scheint bei geringfügig unterschiedlichen Baudraten sinnvoll zu sein, da so den Folgezeichen mehr Zeit zur Verfügung steht, neu mit dem nächsten Startbit zu synchronisieren. Die PC-Seite mit der avrdude Version 6.3 hat leider die Möglichkeit nicht,

bei der seriellen Ausgabe ein zweites Stoppbit auszugeben. Für meine Tests habe ich die Quellen von avrdude so verändert, daß diese Funktion mit einem zusätzlichen Parameter -S angewähltt werden kann. Abschalten läßt sich die zwei Stoppbit Ausgabe bei **optiboot** übrigens mit der Option TWO_STOP_BITS=0. Der Unterschied bei der Geschwindigkeit ist allerdings unbedeutend, so daß ich empfehle, bei der Benutzung des längeren Stoppbits zu bleiben.

Bei der Software für die serielle Schnittstelle (SOFT_UART=1) wird übrigens die Wartezeit für das Stoppbits am Anfang einer seriellen Übertragung berücksichtigt. Dadurch wird dem Programm mehr Zeit gegeben, sich gegebenenfalls am Ende der seriellen Datenübertragung auf den Empfang von seriellen Daten vorzubereiten. Auch diese Funktion ist mit einer zusätzlichen Option QuickStopBit=0 abschaltbar. Ich habe aber bisher keine Probleme oder Nachteile dieser Methode beobachtet, auch nicht beim Betrieb mit nur einem Pin.

3.8 Einige Beispiele für die Erzeugung eines optiboot Bootloaders

Das erste Beispiel ist die Bildung eines Bootloaders für den beliebten ATmega328P:

```
make atmega328p
```

```
Optiboot for 16000000 Hz (16.00 Mhz) operation with Baudrate 115200 and EEprom support
 configured.
 >>> Start building optiboot 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
```

Wenn keine zusätzliche Option angegeben wird, wird eine Baudrate von 115200 mit einer Taktfrequenz von $16MHz$ gewählt. Für die serielle Ausgabe wird die vorhandene Hardware-Schnittstelle benutzt. Man beachte, daß der systematische Fehler der Baudrate über 2% mit dem Hardware UART beträgt. Das zweite Beispiel zeigt die Erzeugung von **optiboot** für den gleichen Prozessor mit einer Software-Lösung für die serielle Schnittstelle.

```
make atmega328p SOFT_UART=1
```

```
Optiboot for 16000000 Hz (16.00 Mhz) operation with Baudrate 115200 and EEprom support
 configured.
 >>> Start building optiboot 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
```

Beachten Sie bitte, daß die Software-Lösung für die serielle Schnittstelle etwas mehr flash-Speicher braucht, aber es wird auch nur eine „boot page" gebraucht. Der systematische Fehler der Baudrate ist hier deutlich kleiner wie bei dem Hardware-UART. Aber der Hardware-UART hat den Vorteil, daß Eingang und Ausgang gleichzeitig bearbeitet werden kann (full duplex) und außerdem ist die Hardware-Lösung fehlertoleranter gegenüber kurzen Störungen des Eingangssignals. Für die serielle Schnittstelle mit Software kann jeder digitale IO-Pin als Eingang (UART_RX) und als Ausgang (UART_TX) gewählt werden. In diesem Beispiel wird die Fähigkeit benutzt, automatisch die IO-Pins des Hardware-UART's zu wählen. Die automatische Wahl der IO-Pins hängt vom gewählten Prozessor-Typ und von der gewählten UART Nummer ab, wenn mehr als ein UART zur Verfügung steht.

Die letzten beiden Beispiele zeigen jeweils eine Konfiguration mit der neuen automatischen Anpassung der Baudrate. Um Platz im Speicher zu sparen, wurde beim ersten Beispiel die LED-Blinkfunktion abgewählt.

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

```
Optiboot for 16000000 Hz (16.00 Mhz) operation with Auto-Baudrate and EEprom support
 configured.
 >>> Start building optiboot 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 488 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
```

Beim letzten Beispiel wurde die aufwendigste Methode der Baudraten-Messung gewählt, da mit der LED-Blink Funktion ohnehin die Grenze von 512 Byte überschritten würde (534 Bytes).

```
make atmega328p BAUD_RATE=76
```

```
Optiboot for 16000000 Hz (16.00 Mhz) operation with Auto-Baudrate and EEprom support
 configured.
 >>> Start building optiboot 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 Anpassung der Taktfrequenz bei internem RC-Generator

Die Benutzung der seriellen Schnittstelle ist nur möglich, wenn die eingestellte Baudrate hinreichend genau eingehalten wird. Die tatsächliche Baudrate hängt unmittelbar von dem tatsächlichen Prozessortakt und von dem berücksichtigtem Teilerverhältnis für die Takt der seriellen Ein-Ausgabe ab. Die Hardware UART Schnittstelle teilt den Prozessortakt generell durch 8 oder durch 16 und kann dann für den bereits vorgeteilten Takt einen weiteren einstellbaren Teiler zwischen 1:1 und 1:4096 zur Erzeugung des Taktes für die serielle Schnittstelle benutzen. Für niedrige Baudraten kann der Prozessortakt bei Bedarf noch um zusätzliche 2er Potenzen geteilt werden. Wenn das Verhältnis zwischen dem Prozessortakt zu dem Baudraten-Takt ausreichend hoch ist, kann damit die Baudrate ausreichend genau eingestellt werden. Bei der Erzeugung des Optiboot Bootloaders wird der systematische Fehler dieser Erzeugung protokolliert. Normalerweise sind Fehler von weniger als 2% als unkritisch anzusehen. Für die Implementation der seriellen Schnittstelle mit Software (SOFT_UART) werden meistens geringere systematische Fehler der Baudrate erreicht. Hier liegen die Schwierigkeiten bei der

fehlenden Fehlerunterdrückung der Einlese-Funktion und bei der mangelden Fähigkeit, die Schnittstelle „voll duplex" zu betreiben. Von der Ausgabe des letzten Bits bis zur Empfangsbereitschaft vergeht immer eine gewisse Zeit. Daher ist hier bei geringeren Baudraten mit weniger Schwierigkeiten zu rechnen.

Alle diese Betrachtungen setzen aber voraus, daß der Prozessortakt selbst hinreichend genau eingehalten wird. Bei Quarz oder Keramik-Resonator Betrieb ist das meistens ohne weitere Maßnahmen der Fall. Anders sieht das aber bei der Benutzung des internen RC-Generators der AVR Prozessoren aus. Hier kann die Prozessorfrequenz deutlich vom Wunschwert abweichen. Die Prozessoren sind zwar ab Werk vorkalibriert. Dies gilt aber oft nur für eine Frequenz und für eine Temperatur und eine Betriebsspannung. Je nach Prozessortyp ist die Frequenz des RC-Generators mehr oder weniger abhängig von Temperatur und Betriebsspannung.

Um vorhandene Fehler der Frequenz mit dem internen RC-Generators ausgleichen zu können, wird der Kalibrationswert des Herstellers beim Start des Prozessors in das IO-Register OSCCAL kopiert. Der Optiboot Bootloader kann die Option OSCCAL_CORR benutzen, um den vorhandenen Restfehler auszugleichen.

Wenn die aktuelle Taktrate für Ihre Applikation unwichtig ist und Sie mehr Speicherplatz für den Bootloader zur Verfügung stellen können, können Sie auch die automatische Baudraten-Anpassung von Optiboot benutzen. Die automatische Baudraten-Anpassung wird immer dann eingebaut, wenn Sie bei der Erzeugung eine Baudrate unter 100 angeben. Näheres zur automatischen Baudraten-Anpassung finden Sie im Unterkapitel 3.7.4 auf Seite 33. Aber Sie sollten im Gedächtnis behalten, daß die Abweichung der Takt-Frequenz auch das Applikationsprogramm beeinflußt (wenn eine serielle Schnittstelle benutzt werden soll).

In den folgenden Unterkapiteln werde ich einige AVR-Typen exemplarisch näher untersuchen.

3.9.1 Untersuchung der RC-Generatoren des ATmega8

Der ATmega8 kann 4 verschiedene Frequenzen mit dem internen RC-Generator über die Low-Fuse einstellen, $1MHz$, $2MHz$, $4MHz$ und $8MHz$. In der Tabelle 3.10 habe ich alle 4 Frequenzeinstellungen untersucht.

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

Tabelle 3.10. Mögliche OSCCAL_CORR Einstellungen für die RC-Frequenzen des ATmega8

Die Tabelle 3.10 zeigt, daß für den 1Mhz und 2MHz Betrieb eine Korrektur des OSCCAL Registers nicht erforderlich ist. Dieser ATmega8 ist ab Werk für diese Frequenzen sehr gut kalibriert. Bei $4MHz$ kommt man noch ohne Korrektur aus, erreicht aber bei OSCCAL_CORR=7 die richtige Schwingfrequenz. Bei $8MHz$ ist der Betrieb der seriellen Schnittstelle so gerade eben noch möglich, sicherer läuft die serielle Schnittstelle aber mit OSCCAL_CORR=6.

3.9.2 Untersuchung der RC-Generatoren des ATmega8535

Der ATmega8535 kann wie der ATmega8 4 verschiedene Frequenzen mit dem internen RC-Generator über die Low-Fuse einstellen, $1MHz$, $2MHz$, $4MHz$ und $8MHz$. In der Tabelle 3.11 sind die Ergebnisse bei einem Exemplar für alle 4 Frequenzeinstellungen festgehalten.

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

Tabelle 3.11. Mögliche OSCCAL_CORR Einstellungen für die RC-Frequenzen des ATmega8535

3.9.3 Untersuchung der RC-Generatoren des ATmega8515 und des ATmega162

Der ATmega8515 kann wie der ATmega8 4 verschiedene Frequenzen mit dem internen RC-Generator über die Low-Fuse einstellen, $1MHz$, $2MHz$, $4MHz$ und $8MHz$. In der Tabelle 3.12 sind die Ergebnisse bei einem Exemplar für alle 4 Frequenzeinstellungen festgehalten.

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

Tabelle 3.12. Mögliche OSCCAL_CORR Einstellungen für die RC-Frequenzen des ATmega8515

Mit einer ähnlichen Pinbelegung aber nur einem $8MHz$ RC-Generator liefert ein ATmega162 die Ergebnisse von Tabelle 3.13.

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

Tabelle 3.13. Mögliche OSCCAL_CORR Einstellungen für den RC-Oszillator des ATmega162

3.9.4 Untersuchung der RC-Generatoren der ATmega328 Familie

Bei der ATmega328 Familie kann nur eine RC-Oszillatorfrequenz von $8MHz$ gewählt werden. Die Frequenz kann aber mit einem Fuse-Bit mit Faktor 8 vorgeteilt werden, so daß auch ein $1MHz$ Betrieb eingestellt werden kann. Die Tabelle 3.14 zeigt die Resultate der untersuchten Prozessoren.

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

Tabelle 3.14. Mögliche OSCCAL_CORR Einstellungen für die ATmega328 Familie

Bei allen untersuchten Prozessoren ist ein Betrieb der seriellen Schnittstelle auch mit dem internen RC-Oszillator möglich. Nur bei dem untersuchten ATmega88 würde sich bei OSCCAL_CORR=4 eine Korrektur überhaupt lohnen.

3.9.5 Untersuchung der RC-Generatoren des ATmega32 / 16

Der ATmega32 und der ATmega16 können 4 verschiedene Frequenzen mit dem internen RC-Generator über die Low-Fuse einstellen, $1MHz$, $2MHz$, $4MHz$ und $8MHz$. In den Tabellen 3.15 und 3.16 habe ich alle 4 Frequenzeinstellungen untersucht.

Typ	AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
				Corr	Freq	Corr	Freq	Corr	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

Tabelle 3.15. Mögliche OSCCAL_CORR Einstellungen für die RC-Frequenzen des ATmega32

AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
			Corr	Freq	Corr	Freq	Corr	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

Tabelle 3.16. Mögliche OSCCAL_CORR Einstellungen für die RC-Frequenzen des ATmega16A

Immer wenn in der MinCorr Spalte positive Werte oder in der MaxCorr Spalte negative Werte auftauchen, ist bei dieser Frequenz und diesem Prozessor-Exemplar der Betrieb der seriellen Schnittstelle ohne Korrektur nicht möglich. Wenn eine 0 in den Spalten auftaucht, ist der Betrieb der seriellen Schnittstelle gerade eben noch möglich.

3.9.6 Untersuchung des RC-Generators des ATmega163L

Der ATmega163L besitzt nur einen $1MHz$ RC-Generator, der mit dem OSCCAL Register abgestimmt werden kann. Mein Exemplar hatte keine Voreinstellung des OSCCAL Wertes. Daher sind hier außergewöhnlich hohe Korrekturwerte erforderlich, um die Frequenz auf etwa $1MHz$ einstellen zu können.

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

Tabelle 3.17. Mögliche OSCCAL_CORR Einstellungen für die RC-Frequenz des ATmega163L

3.9.7 Untersuchung der RC-Generatoren des ATmega64 / 128

Der ATmega64 und der ATmega128 können 4 verschiedene Frequenzen mit dem internen RC-Generator über die Low-Fuse einstellen, $1MHz$, $2MHz$, $4MHz$ und $8MHz$. In den Tabellen 3.18 und 3.19 habe ich alle 4 Frequenzeinstellungen untersucht. An dieser Stelle sei auch ein Hinweis erlaubt, das das Laden der Programmdaten über die ISP-Schnittstelle nicht mit den Signalen MISO und MOSI, sondern über die Signale TXD (PE1) und RXD (PE0) erfolgt. Dies muß natürlich beim Anschluß an den Programmer berücksichtigt werden.

AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
			Corr	Freq	Corr	Freq	Corr	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

Tabelle 3.18. Mögliche OSCCAL_CORR Einstellungen für die RC-Frequenzen des ATmega64

AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
			Corr	Freq	Corr	Freq	Corr	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

Tabelle 3.19. Mögliche OSCCAL_CORR Einstellungen für die RC-Frequenzen des ATmega128

Die Tabellen zeigen, daß für den $1Mhz$ und $2MHz$ Betrieb eine Korrektur des OSCCAL Registers nicht erforderlich ist. Bei $4MHz$ und $8MHz$ Betrieb kann die serielle Schnittstelle aber beim untersuchten ATmega64 nicht ohne Korrektur betrieben werden. Ohne Korrektur wäre die 4MHz Frequenz um etwa 4% zu hoch und die $8MHz$ Frequenz um etwa 4.3% zu hoch. In der Dokumentation von Atmel wird übrigens abgegeben, daß der RC-Generator des ATmega64 und ATmega128 bei $1MHz$ abgeglichen wird. Es sei noch einmal darauf hingewiesen, daß es sich bei den Tabellendaten um die Untersuchung eines einzelnen Exemplars des ATmega handelt. Außerhalb der angegebenen

Min- bzw. Max-Werte der OSCCAL Korrekturen war ein Betrieb der seriellen Schnittstelle bei der angegebenen Baud-Rate nicht möglich.

3.9.8 Untersuchung der RC-Generatoren der ATmega644 Familie

Bei der ATmega644 Familie kann eine RC-Oszillatorfrequenz von 8MHz gewählt werden. Daneben kann auch noch ein $128kHz$ Gererator als Takt gewählt werden, der sonst den Watchdog-Timer versorgt. Die gewählte Frequenz kann aber mit einem Fuse-Bit mit Faktor 8 vorgeteilt werden, so daß auch ein $1MHz$ Betrieb eingestellt werden kann. Die Tabelle 3.20 zeigt die Resultate der untersuchten Prozessoren.

Typ	AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
				Corr	Freq	Corr	Freq	Corr	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

Tabelle 3.20. Mögliche OSCCAL_CORR Einstellungen für die ATmega644 Familie

3.9.9 Untersuchung der RC-Generatoren der ATmega645 Familie

Bei der ATmega645 Familie kann nur eine RC-Oszillatorfrequenz von $8MHz$ gewählt werden. Die Frequenz kann aber mit einem Fuse-Bit mit Faktor 8 vorgeteilt werden, so daß auch ein $1MHz$ Betrieb eingestellt werden kann. Die Tabelle 3.21 zeigt die Resultate der untersuchten Prozessoren.

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

Tabelle 3.21. Mögliche OSCCAL_CORR Einstellungen für die ATmega645 Familie

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 Untersuchung der RC-Generatoren der ATmega649 Familie

Bei der ATmega649 Familie kann nur eine RC-Oszillatorfrequenz von $8MHz$ gewählt werden. Die Frequenz kann aber mit einem Fuse-Bit mit Faktor 8 vorgeteilt werden, so daß auch ein $1MHz$ Betrieb eingestellt werden kann. Die Tabelle 3.22 zeigt die Resultate der untersuchten Prozessoren.

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

Tabelle 3.22. Mögliche OSCCAL_CORR Einstellungen für die ATmega649 Familie

3.9.11 Untersuchung des RC-Generators der ATmega2560 Familie

Bei der ATmega2560 Familie kann eine RC-Oszillatorfrequenz von 8MHz gewählt werden. Daneben kann auch noch ein $128kHz$ Gererator als Takt gewählt werden, der sonst den Watchdog-Timer versorgt. Die gewählte Frequenz kann aber mit einem Fuse-Bit mit Faktor 8 vorgeteilt werden, so daß auch ein $1MHz$ Betrieb eingestellt werden kann. Die Tabelle 3.23 zeigt die Resultate der untersuchten Prozessoren.

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

Tabelle 3.23. Mögliche OSCCAL_CORR Einstellungen für die ATmega2560 Familie

Das Laden von mehr als 128KByte Daten wurde mit dem ATmega2561 erfolgreich getestet. Normalerweise beginnen die Anwenderdaten für den Flash-Speicher bei der Addresse 0. Das ist für den Daten-Download über die serielle Schnittstelle nicht unbedingt erforderlich. Die Anfangs-Addresse muß aber auf jeden Fall unter 128K (0x20000) liegen, damit das Laden von Daten in die obere Speicherhälfte funktioniert. Die Option VIRTUAL_BOOT_PARTITION kann bei Prozessoren mit mehr als 128Kbyte Flash Speicher nicht benutzt werden.

3.9.12 Untersuchung der RC-Generatoren der ATtiny4313 Familie

Beim ATtiny4313 und ATtiny2313 kann der interne RC-Generator bei $8MHz$ oder bei $4MHz$ betrieben werden. Außerdem ist auch noch ein interner $128kHz$ Generator wählbar, der aber nicht kalibriert werden kann. Die Tabelle 3.24 zeigt die Ergebnisse der Frequenzmessung beim $8MHz$ und $4MHz$ Betrieb

Typ	AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
				Corr	Freq	Corr	Freq	Corr	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

Tabelle 3.24. Mögliche OSCCAL_CORR Einstellungen für die ATtiny4313 Familie

Bei allen drei vorhandenen ATtinys dieser Serie hat sich das Einstellen der Frequenz als schwierig erwiesen weil sich die Frequenz bei kleiner OSCCAL Korrektur schon relativ stark ändert.

3.9.13 Untersuchung der RC-Generatoren der ATtiny84 Familie

Bei der ATtiny84 Familie kann außer dem $8MHz$ RC-Oszillator auch noch der $128kHz$ Takt der Watchdog-Schaltung benutzt werden. Der $128kHz$ Takt kann nicht kalibriert werden. Wenn man diesen Takt benutzen möchte, kann man die erzeugte Baudrate nur über den vorgegebenen Wert korrigieren oder die automatische Baudraten-Anpassung benutzen. Bei einem ATtiny24a habe ich die erzeugte Baudrate kontrolliert. Statt der eingestellten 2400 Baud habe ich nur 2170 Baud gemessen. Das ergibt einen Frequenzfehler von etwa 9.6% und damit deutlich zu viel, um ohne Korrektur benutzt werden zu können. Bei eingestellten 2640 Baud funktionierte der Download mit 2400 Baud. Die gemessene Taktrate betrug dann nur $115.2kHz$ anstelle der $128kHz$. Die Tabelle 3.25 stellt für die untersuchten Exemplare die möglichen OSCCAL Änderungen für den $8MHz$ Betrieb dar.

Typ	AVR_ FREQ	LFUSE	Baud-rate	Minimum		Maximum		Best	
				Corr	Freq	Corr	Freq	Corr	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

Tabelle 3.25. Mögliche OSCCAL_CORR Einstellungen für die ATtiny84 Familie

3.9.14 Untersuchung der RC-Generatoren der ATtiny85 Familie

Bei der ATtiny84 Familie kann außer dem $8MHz$ RC-Oszillator auch noch der ein $6.4MHz$ RC-Oszillator und der $128kHz$ Takt der Watchdog-Schaltung benutzt werden. Der $6.4MHz$ RC-Oszillator wird aber immer mit einem Frequenzteiler auf $1.6MHz$ heruntergeteilt. Der $128kHz$ Takt kann nicht kalibriert werden. Wenn man diesen Takt benutzen möchte, kann man die erzeugte Baudrate nur über den vorgegebenen Wert korrigieren oder die automatische Baudraten-Anpassung benutzen. Die erste Tabelle 3.26 stellt für die untersuchten Exemplare die möglichen OSCCAL Änderungen für den $8MHz$ Betrieb dar.

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

Tabelle 3.26. Mögliche OSCCAL_CORR Einstellungen für die ATtiny85 Familie bei 8MHz

Die Einstellwerte bei einem ATtiny25 sehen merkwürdig aus, aber bei der Korrektur 3 hat der OSCCAL Wert die Zahl 128 unterschritten und ist damit in einem anderen Stellbereich. Erst bei einem Korrekturwert von etwa 34 wurde dann wieder eine Frequenz von $8364kHz$ erreicht, bei der die serielle Schnittstelle wieder betrieben werden kann. Eine ähnliche Frequenz wurde im anderen Stellbereich beim Korrekturwert -6 erreicht. Die nächste Tabelle 3.27 stellt für die untersuchten Exemplare die möglichen OSCCAL Änderungen für den $1.6MHz$ Betrieb dar. Der RC-Generator läuft dabei bei $6.4MHz$, diese Frequenz wird aber immer durch Faktor 4 geteilt.

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

Tabelle 3.27. Mögliche OSCCAL_CORR Einstellungen für die ATtiny85 Familie bei $1.6MHz$

Bei dem Betrieb mit $1.6MHz$ hat sich die Besonderheit mit der OSCCAL Korrektur-Einstellung nicht gezeigt. Alle untersuchten Exemplare würden auch ohne Korrektur der Frequenz mit der seriellen Schnittstelle arbeiten können. Die ATtiny84 Prozessor Familie kann den Prozessor auch mit einem PLL-Oszillator betreiben, der vom $8MHz$ RC-Oszillator kontrolliert wird. Der PLL-Oszillator kann entweder $64MHz$ oder $32MHz$ erzeugen, die normalerweise für den T1 Zähler benutzt werden können. Wenn der PLL-Takt als Prozessor-Takt verwendet werden soll, kann der PLL-Oszillator nur bei $64MHz$ arbeiten und wird immer durch Faktor 4 geteilt. Somit ergeben sich 16MHz für den Prozessortakt. Die Tabelle 3.28 zeigt die gemessenen Resultate. Erwartungsgemäß weichen diese Ergebnisse nicht wesentlich von den $8MHz$ Ergebnissen ab.

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

Tabelle 3.28. Mögliche OSCCAL_CORR Einstellungen für die ATtiny85 Familie bei 16MHz

3.9.15 Untersuchung der RC-Generatoren der ATtiny841 Familie

Der ATtiny841 und der ATtiny441 haben ebenfalls einen $8MHz$ RC-Generator, der abgestimmt werden kann. Für die Unterstützung dieser Familie waren einige Anpassungen beim Optiboot Bootloader erforderlich. In der Tabelle 3.29 sind die Ergebnisse der untersuchten Exemplare dargestellt.

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

Tabelle 3.29. Mögliche OSCCAL_CORR Einstellungen für die ATtiny841 Familie bei 8MHz

Bei allen untersuchten Exemplaren kann die serielle Schnittstelle auch ohne die OSCCAL-Korrektur benutzt werden.

3.9.16 Untersuchung der RC-Generatoren der ATtiny861 Familie

Bei der ATtiny861 Familie kann außer dem $8MHz$ RC-Oszillator noch ein PLL-Oszillator und der $128kHz$ Takt der Watchdog-Schaltungs als Prozessor-Takt benutzt werden. Der $128kHz$ Takt der Watchdog-Schaltung kann aber nicht kalibriert werden und kommt daher für die Bootloader-Anwendung sinnvoll nur mit der automatischen Baudraten-Anpassung in Frage. Der PLL-Oszillator hat für den Prozessor eine Frequenz von $16MHz$, kann aber leider nur vom internen RC-Oszillator synchronisiert werden. Daher kann mit dem PLL-Oszillator keine genaue Zeitmessung durchgeführt werden. Die erste Tabelle 3.30 stellt für die untersuchten Exemplare die möglichen OSCCAL Änderungen für den $8MHz$ Betrieb dar.

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

Tabelle 3.30. Mögliche OSCCAL_CORR Einstellungen für die ATtiny861 Familie bei $8MHz$

Beim ATtiny261 habe ich auf das optionale LED-Blinken beim Start von Optiboot verzichtet, damit genug Platz für das Testprogramm zur Verfügung steht (Option LED_START_FLASHES=0).

3.9.17 Untersuchung des RC-Generators der ATtiny87 Familie

Die ATtiny87 Familie kann außer einem $8MHz$ internem RC-Generator nur noch einen nicht kalibrierbaren $128kHz$ Generator oder einen externen Takt als Taktgeber für den Prozessor wählen. Ein Frequenzteiler mit Faktor 8 ist für den Prozessortakt bei der LFUSE auch einstellbar. In der Tabelle 3.31 sind die Ergebnisse für die Kalibration des internen $8MHz$ Generators bei zwei Exemplaren der Familie dargestellt.

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

Tabelle 3.31. Mögliche OSCCAL_CORR Einstellungen für die ATtiny87 Familie

3.9.18 Untersuchung des RC-Generators der ATtiny88 Familie

Die ATtiny88 Familie kann außer einem $8MHz$ internem RC-Generator nur noch einen nicht kalibrierbaren $128kHz$ Generator oder einen externen Takt als Taktgeber für den Prozessor wählen. Ein Frequenzteiler mit Faktor 8 ist für den Prozessortakt bei der LFUSE auch einstellbar. In der Tabelle 3.32 sind die Ergebnisse für die Kalibration des internen $8MHz$ Generators bei zwei Exemplaren der Familie dargestellt.

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

Tabelle 3.32. Mögliche OSCCAL_CORR Einstellungen für die ATtiny88 Familie

Bei beiden Exemplaren läuft die serielle Schnittstelle ohne Frequenz-Korrektur. Die Takt-Frequenz wird aber bei einer Korrektur von 5 (6) besser eingehalten.

3.9.19 Untersuchung der RC-Generatoren des ATtiny1634

Für den ATtiny1634 habe ich zwei Exemplare mit dem $8MHz$ internem RC-Generator untersucht. Neben der Frequenz kann auch noch der Frequenz-Temperaturgang des RC-Generators mit zwei zusätzlichen Kalibrations-Registern eingestellt werden. Den Abgleich des Temperaturgangs habe ich in der Tabelle 3.33 nicht untersucht. Neben dem $8MHz$ RC-Generator kann bei diesem Prozessor auch die Frequenz des internen $32kHz$ Generators mit einem zusätzlichen Kalibrations-Register (OSCCAL1) abgeglichen werden. Das wird derzeit aber noch nicht vom Optiboot Bootloader unterstützt.

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

Tabelle 3.33. Mögliche OSCCAL_CORR Einstellungen für den ATtinye1634

3.9.20 Untersuchung des RC-Generators der AT90PWM Familie

Neben dem üblichen $8MHz$ RC-Generator hat die AT90PWM Familie noch einen PLL-Oszillator, der vom $8MHz$ Generator synchronisiert wird. Mit dem PLL-Oszillator kann der Prozessor mit $16MHz$ betrieben werden.

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	5	15.79M	2	15.98M

Tabelle 3.34. Mögliche OSCCAL_CORR Einstellungen für die AT90PWM Familie

Beim untersuchten AT90PWM2B wäre der Betrieb der seriellen Schnittstelle ohne die OSCCAL-Korrektur nicht möglich gewesen.

3.9.21 Untersuchung des RC-Generators der AT90CAN Familie

Untersucht habe ich nur jeweils ein Exemplar AT90CAN32 und AT90CAN128. Beide Exemplare können nur einen internen 8 MHz RC-Oszillator wählen. Alle anderen Wahlmöglichkeiten erfordern

einen externen Quarz oder einen externen Taktgenerator. Die Taktfrequenz kann optional auch mit dem Faktor 8 geteilt werden, so daß auch 1 MHz Betrieb mit dem internen Takt möglich ist.

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

Tabelle 3.35. Mögliche OSCCAL_CORR Einstellungen für die AT90CAN Familie

3.10 Prüfung der Zusammenarbeit mit USB-Seriell Wandlern

3.10.1 Vorwort zum Testumfeld

Für eine Serienuntersuchung des Verhaltens der automatischen Baudratenanpassung sind die Bedingungen erst einmal schlecht. Die Baudraten der AVR-UART Schnittstelle können für höhere Baudraten nur in groben Schritten eingestellt werden. Auch die Baudraten der USB-Seriell Schnittstellen lassen sich nur relativ grob einstellen und die Wahlmöglichkeit ist vom jeweiligen Chip-Typ und den Fähigkeiten des Treibers beschränkt. Es besteht zwar bei den AVR Prozessoren die Möglichkeit, die Taktfrequenz des lokalen RC-Generators mit OSCCAL zu verstimmen, aber das Resultat der Verstimmung ist vom jeweiligen Exemplar abhängig. Die jeweilige Schwingfrequenz des lokalen Oszillators könnte dann zwar gemessen werden, aber die Schrittweite der Verstimmung kann nicht geändert werden um eine bestimmte Frequenz genau zu erreichen. Der einzige mir bekannte Ausweg ist die Benutzung einer externen Taktquelle für den AVR-Prozessor. Dafür habe ich mir den relativ preiswerten Kkmoon FY6900 Funktionsgenerator beschafft, der nahezu beliebige Wellenformen mit Frequenzen bis zu 60 MHz erzeugen kann. Natürlich kann dieser Funktionsgenerator auch Rechteck-Signale erzeugen, die dann als Takt für den AVR dienen können. Der Funktionsgenerator kann sowohl über das Frontpaneel und auch über ein USB-Interface bedient werden. Das USB-Interface besteht aus einem internen CH341 USB-Seriell Wandler, so daß kein spezieller USB Treiber erforderlich ist. Die Steuerung kann mit gut dokumentierten ASCII Zeichenfolgen erfolgen. Damit sind wesentliche Voraussetzungen für eine systematische Untersuchung des Zusammenspiels von avrdude, den USB-Seriell Wandlern und des Optiboot Bootloaders erfüllt. Speziell das Verhalten der automatischen Baudratenanpassung sollte intensiver untersucht werden. Dafür sollten Downloads mit avrdude mit verschiedenen Taktfrequenzen entsprechend oft wiederholt werden, damit Probleme erkannt werden. Eine Anzahl von 40 Wiederholungen des Downloads für jede Frequenz erschien mir ein vertretbarer Kompromiss zwischen Zeitaufwand und statistischer Aussagekraft zu sein. Da bei der automatischen Baudratenanpassung auch 41 verschiedene Frequenzen untersucht werden sollen, ergeben sich für eine Variante schon 1640 Schreibvorgänge. Bei über 10 dieser Varianten würde damit die garantierte Zyklenzahl des AVR Flashspeichers überschritten. Da ich nicht die Zyklen-Festigkeit des AVR-Flashspeichers untersuchen wollte, habe ich dafür einen Schreibschutz-Pin in der **optiboot** Software vorgesehen. Diese Zusatzfunktion von **optiboot** ist nur für diese Untersuchungen sinnvoll, deshalb ist diese Option (WRITE_PROTECT_PIN) nicht weiter dokumentiert.

3.10.2 Autobaud Messungen mit verschiedenen USB-Seriell Chips

Die Daten für die hier vorgestellten Ergebnisse wurden mit der Bash-Script Datei test_download im Unterverzeichnis ee_test erzeugt und dann in .tab Dateien im Verzeichnis Doku/results mit

einem Editor zusammengefaßt. Im Laufe der Untersuchungen konnte eine Verbesserung der Baudratenmessung mit der vorgeteilten Taktfrequenz in **optiboot** eingebaut werden. Die Verbesserung besteht aus einer Zurücksetzung des Vorteilers für die Zähler des AVR. Die andere Fehlerquelle für die Zeitmessung entsteht durch die Abfrageschleife für den RX Datenpin und beträgt mindestens 3 Takte für jeden Meßpunkt. Doch hier erst einmal als Ergebnis die Fehlerrate mit verschiedenen USB-Seriell Chips bei jeweils 40 Downloads für jede Taktfrequenz zwischen 14 MHz und 18 MHz mit einer Schrittweite von 100 kHz und einer Baudrate von 115200. Insgesamt wurden für jeden Tabelleneintrag 1640 Downloads von 10 kByte mit avrdude durchgeführt. Es wurde mit einer speziell angepassten avrdude Version gearbeitet, bei der eine serielle Ausgabe mit zwei Stoppbits mit dem Parameter -S angewählt werden kann. Die Differenzen zur avrdude Version 6.3 sind in der Datei ee_test/avrdude.diff festgehalten.

Baudrate	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

Tabelle 3.36. Fehlerrate bei 1640 Downloads mit je 10 kByte mit HW-UART

Zur Erinnerung sei hier der Schlüssel für die Optiboot Auto-Baudraten Einstellung erwähnt. Die Einstellung 22, 26 und 62 wählen alle eine 2-Bit Zeitmessung an. Bei den Einstellungen 22 und 62 arbeitet der für die Zeitmessung verwendete Zähler mit /8 Frequenz-Vorteiler, bei Einstellung 26 läuft der Zähler mit voller Taktrate. Entsprechend wählen die Einstellungen 32, 36, 72 und 76 eine 4-Bit Zeitmessung an. Bei den Einstellungen 32 und 72 wird ein Zähler mit /8 Frequenz-Vorteiler benutzt, die Einstellungen 36 und 76 benutzen für die Zeitmessung einen Zähler ohne Vorteiler. Die beiden übrigen Einstellungen 12 und 82 benutzen für die Zeitmessung 9 Bits, beide mit dem /8 Vorteiler für den Zähler.

Der USB-Seriell Wandler FT232 taucht in Tabelle 3.36 zweimal auf, einmal als FT232RL, dem Original Chip von Future Technology Devices International Limited (FTDI) und einmal als FT232clone, einem Chinesischen Nachbau unbekannter Herkunft mit der Seriennummer A50285BI. Die Nachbauten fallen meist durch gleiche Seriennummer auf. Die Original Chips haben alle eine unterschiedliche Seriennummer. Äußerlich sind die nachgebauten Chips fast nicht zu erkennen, da auch das Hersteller-Logo FTDI kopiert wurde. Der untersuchte chinesische Nachbau erreicht aber ähnlich gute Testergebnisse wie die Original-Chips.

In der Tabelle 3.36 fällt auf, daß die Zahl der Fehler bei den Autobaud Einstellungen 62, 72 und 82 auffällig hoch sind. Der Grund liegt daran, daß der Zähler mit dem /8 Vorteiler für die zusätzlichen Kontrollen schon beim Startbit der seriellen Empfangsdaten gestartet wird und dann für die Zeitmessung nur abgelesen wird. Dadurch ist der Zustand des Vorteilers bei der eigentlichen Zeitmessung undefiniert. Bei den Einstellungen 22 und 32 wird der Zähler mit zurückgesetztem Teiler erst mit dem ersten Datenbit gestartet, das für die Messung benutzt wird. Dadurch gewinnt die Zeitmessung

an Meßsicherheit. Die Einstellungen 26, 36 und 76 benutzen den Zähler ohne Vorteiler und haben daher das Problem mit dem undefinierten Vorteiler gar nicht. Erwähnen möchte ich hier, daß frühere Versionen der **optiboot** Autobaud Messung mit der Einstellung 22 und 32 sicherlich ähnlich hohe Fehlerraten wie bei den Einstellungen 62 und 72 ergeben hätten, da der Reset des Vorteilers beim Meßstart erst jetzt eingebaut wurde. Bei normaler Benutzung tauchen üblicherweise weniger Probleme mit Autobaud auf, da die TaktFrequenz des AVR besser zur gewünschten Baudrate passt. Mit dieser Untersuchung sollten ja die Grenzen der automatischen Baudratenanpassung getestet werden.

Zwei USB-Seriell Wandler fallen in der Tabelle 3.36 mit höherer Fehlerrate auf, der CH340 und der Pololu AVR. Um die Ursache einzugrenzen, wurden weitere Messungen mit fest eingestelltem UBBR Teiler gemacht, die im nächsten Unterkapitel dargestellt werden.

3.10.3 Messung von USB-Seriell Wandlern mit fester optiboot Baudrate

Für diese Untersuchung wurde die **optiboot** Baudrate fest auf 115200 bei einer Taktrate von 16 MHz eingestellt. Die USB-Seriell Wandler werden auch bei 115200 Baud betrieben. Die Taktfrequenz des ATmega wurde statt der festen 16 MHz von 14.7 Mhz in 100 kHz Schritten auf 17.1 MHz erhöht. Mit der Frequenzänderung ändert sich dann die **optiboot** Baudrate proportional mit.

Für die Tests wurden jeweils 40 Downloads von 10 kByte mit avrdude durchgeführt. In einer Tabelle wurde die Zahl der Fehlversuche für jede Frequenz von der Script-Datei test_download festgehalten. Aus den so gewonnenen Tabellen für die verschiedenen USB-Seriell Wandler wurde mit einem Editor jeweils eine Gesamttabelle für die Testreihe mit 1 Stoppbit (Doku/results/HWfix_1_115200 _USBserial_0.tab) und für die Testreihe mit 2 Stoppbits (Doku/results/HWfix_1_115200 _USBserial_1.tab) erstellt. Aus diesen Tabellen wurde dann die Tabelle 3.37 erstellt.

	115200 Baud, 2 Stoppbits			115200 Baud, 1 Stoppbit		
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%

Tabelle 3.37. Nutzbarer Taktfrequenzbereich mit Hardware-UART

Erwartet wurde bei dieser Untersuchung eine zulässige Abweichung der Baudrate von ±4%, also ein Gesamtbereich von 8%. Dieser 8% Bereich wird von den untersuchten USB-Seriell Wandlern bei einem Stoppbit nur vom CP2104 und dem PL2303TA erreicht. Mit der auf 2 Stoppbits eingestellten seriellen Schnittstelle erreichen die FT232 Wandler, die CP2102, CP2104 und PL2303TA Wandler sowie der Arduino UNO den erwarteten 8% Bereich. Der CH340G Wandler verbessert den nutzbaren Bereich auf 6.8% deutlich, bleibt aber unter 8%. Der PololuAVR und der PL2303TA zeigen keine Reaktion auf die Erhöhung der Stopp-Bitzahl von 1 auf 2. Eine Zeitmessung mit dem Linux Kommando „time" der Ausgabe von über 10000 Zeichen ergab aber beim PL2303TA einen etwa 10% höheren Zeitbedarf bei der 2 Stoppbit Einstellung. Der PololuAVR zeigt bei diesem Test keine Änderung der Zeit. Es ist wahrscheinlich, daß das Zusammenspiel von Treiber und Chip beim PololuAVR nicht

funktioniert.

Jedenfalls ist hiermit die mögliche Ursache für das schlechte Abschneiden des CH340G Chips und des PololuAVR in Tabelle 3.36 auf Seite 54 gefunden. Beim PololuAVR sollte man berücksichtigen, das dies von Hause aus ein ISP-Programmer ist und der USB-Seriell Wandler nur eine Zusatzfunktion darstellt.
Eindeutige Spitzenreiter bei diesem speziellen Vergleichstest ist der CP2104 der Silicon Laboratories Inc. und der PL2303TA der Prolific Incorporated Inc. , der in diesem Test auch bessere Werte erreicht als der FT232RL der Future Technology Devices International. Nach Datenblatt unterstützt der FT232RL aber eine maximale Baudrate von 3 MBaud, der PL2303TA sogar 6 Mbaud, die nach Datenblatt weder vom CP2102 (1 MBaud) noch vom Testsieger CP2104 (2 MBaud) erreicht werden.

Zusätzlich habe ich auch den Arbeitsbereich der USB-Seriell Wandler im Zusammenspiel mit der Software-UART Lösung von **optiboot** untersucht. Die Software benutzt beim Senden von Daten etwa 1,5 Stoppbits, die USB-Seriell Wandler benutzen entweder 1 (Doku/results/SWfix_1_115200 _USBserial_0.tab) oder 2 Stoppbits (Doku/results/SWfix_1_115200_USBserial_0.tab). In der Tabelle 3.38 sind die Ergebnisse zusammengefaßt.

	115200 Baud, 2 Stoppbits			115200 Baud, 1 Stoppbit		
USB-Seriell Typ	Mindest-Frequenz	Maximal-Frequenz	Frequenz-Bereich	Mindest-Frequenz	Maximal-Frequenz	Frequenz-Bereich
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%

Tabelle 3.38. Nutzbarer Taktfrequenzbereich mit Software-UART

Bei allen Meßserien mit der Software UART Lösung ergeben sich etwas größere Frequenzbereiche ohne Fehler im Vergleich zum Hardware UART Betrieb aus der Tabelle 3.37. Das liegt wohl daran, daß die Softwarelösung nur eine Abtastung des Eingangssignals je Bit macht und versucht diese in die Mitte der mit der Baudrate erwarteten Übertragungszeit zu legen. Die Hardware UART Lösung der AVR Prozessoren machen 3 Abtastungen je Datenbit. beim hier benutzten „Double Speed Mode" werden die Abtastnummern 4, 5 und 6 der 8 Abtastungen je Datenbit oder Stoppbit benutzt. Die drei Abtastungen vorher (1-3) und die beiden hinterher (7-8) werden nicht benutzt. Mindestens 2 Abtastungen der 3 benutzten müssen bei der Stoppbit Abtastung eine 1 ergeben, sonst wird ein „Framing" Fehler erzeugt. Dieses Verfahren der Überabtastung kommt zwar mit gestörter Übetragung besser zurecht, lässt aber weniger Toleranz bei der Baudrate zu.

Über die Art der Abtastung der seriellen Daten der USB-Seriell Wandler habe ich keine nähere Informationen gefunden. Ich gehe aber davon aus, daß alle eine Form der Überabtastung ähnlich wie bei der AVR-UART Schnittstelle benutzen. Allerdings muß sich das Abtastverfahren der einzelnen USB-Seriell Wandler doch unterscheiden, anders sind die unterschiedlichen Testergebnisse nicht zu erklären, weil die **optiboot** Seite für alle Testkandidaten gleich war.

3.10.4 Extremtest mit 115200 Baud bei 8 MHz Takt

Der weite Baudratenbereich für eine funktionierende serielle Kommunikation einiger USB-Seriell Wandler hat mich ermuntert, das Verhalten der Autobaud Funktion in der Praxis für eine Baudrate von 115200 Baud bei einer Taktrate von 8 MHz zu untersuchen. Dafür habe ich den USB-Seriell Wandler mit dem größten Baudratentoleranz ausgewählt, den CP2104. Den für die Tests verwendeten ATmega1281 habe ich mit einem **optiboot** Bootloader mit dem Autobaud Modus 32 ohne LED-Blinken versehen. In der Autobaud Gruppe 30-39 werden 4 Datenbits zur Messung der Baudrate benutzt. Mit dieser Einstellung wurden jeweils 40 Downloads von 10 kByte mit avdrude für 41 Frequenzen von 7 MHz bis 9 MHz in 50 kHz Schritten durchgeführt. Um das Ergebnis bewerten zu können, wurde die gleiche Meßreihe auch mit fest eingestellter Baudrate von **optiboot** wiederholt. Dabei erzeugt die Einstellung 111111 Baud eine UBRR Einstellung des Hardware UARTs von 8, bei der Einstellung 125 kBaud wird eine UBRR Einstellung von 7 benutzt. Durch die Änderung der Taktfreuenz ändert sich die tatsächliche Baudrate natürlich proportional mit.

Typ		CP2104					CH340G
	Modus	HW32	HW111111	HW125000	SW32	SW34	HW32
Frequenz	Periode	auto	UBRR=8	UBRR=7	auto	auto	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

Tabelle 3.39. Fehlerrate beim Extremtest mit 115200 Baud bei 8 MHz Takt

Alle Testergebnisse in der Tabelle 3.39 stellen die Anzahl der Fehlversuche bei der jeweils eingestellten Taktrate (Spalte 1) von 40 Download Versuchen je 10 kByte mit avrdude dar. Die USB-Seriell

Wandler arbeiten immer mit 2 Stoppbits für diese Tests. In der ersten Datenspalte sind die Fehlversuche für einen CP2104 USB-Seriell Wandler festgehalten. Der PL2303TA hat übrigens ein gleiches Ergebnis gezeigt. Wie man sieht gibt es zwischen 7850 kHz und 7950 kHz drei Totalausfälle, keiner der 40 Downloads hat funktioniert. Die Erklärung liefern die nächsten beiden Spalten. Hier wurde mit konstanter UART-Teilereinstellung 9 (UBRR=8) und 8 (UBRR=7) für den gesamten Frequenzbereich die 40 Downloads versucht. Für die Frequenzen 7850 kHz bis 7950 kHz funktioniert keiner der beiden Einstellungen. Zwischen UBRR=7 und UBRR=8 ist aber keine weitere Einstellung wählbar. Damit ist klar, warum hier keine Autobaud Variante funktionieren kann. Es ist mit diesen Vorbedingungen schlichtweg nicht möglich die gewünschte Baudrate von 115200 mit diesen Takfrequenzen hinreichend genau einzustellen. Der teilweise Ausfall des Downloads bei 7800 kHz mit der Autobaud-Variante 32 liegt wohl an der Unsicherheit der Baudratenmessung.

In der vierten Datenspalte wurde der Download bei gleichen Einstellungen mit der Software-UART Lösung von **optiboot** versucht. Bei Software-UART zeigte sich schon in der Tabelle 3.38 auf Seite 56 ein größerer Frequenzarbeitsbereich mit dem CP2104 gegenüber den Hardware-UART Tests in Tabelle 3.37 auf Seite 55. Außerdem wird die Verzögerungszeit des Software-UART mit Warteschleifen gebildet, die eine Einstellung von Vielfachen von 6 Takten erlauben. Durch den Vorteiler des Hardware-UART ergibt sich hier zum Vergleich nur eine Einstellmöglichkeit von Vielfachen von 8 Takten (UBRR Einstellung). Beide Umstände führen wohl dazu, daß beim CP2104 die Software Lösung im gesamten Frequenzbereich nur 4 Fehlversuche bei der Taktfrequenz 7150 kHz zeigt, ein wirklich erstaunliches Ergebnis! Ab 7200 kHz Taktrate ist die Baudrate 115200 mit diesem CP2104 USB-seriell Wandler bei 2 Stoppbit Betrieb und Software-UART des **optiboot** mit BAUD_RATE=32 uneingeschränkt nutzbar.

Als weiteren Test mit der Software UART Lösung von **optiboot** habe ich die BAUD_RATE=34 Einstellung getestet. Bei dieser Einstellung wird die Verzögerungs-Schleife um einen Takt verlängert, so daß sich hier nur ein Vielfaches von 8 Takten wählen läßt, genau wie bei dem Hardware-UART. Der Hintergrund für diese Wahlmöglichkeit ist lediglich ein geringfügig kürzeres **optiboot** Programm mit dieser Einstellung. Obwohl bei dieser Software-UART Einstellung genau so Vielfache von 8 Takten für die Periode wie bei dem Hardware-UART, ist das Ergebnis hier leicht besser. Das liegt wahrscheinlich am schon erwähnten größeren Frequenz-Arbeitsbereich des CP2104 Wandlers mit **optiboot** bei Software-UART Betrieb. Außerden liegt der kritische Bereich hier nicht so dicht bei der gewählten Arbeitsfrequenz von 8000 kHz wie beim Hardware UART.

In der 6. und letzten Datenspalte habe ich die gleichen Messung wie in der ersten Datenspalte mit dem CH340G getestet. Hier zeigen deutlich mehr Frequenzen mit Totalausfällen der Downloadversuche, was auch schon die Untersuchung in Tabelle 3.37 auf Seite 55 erwarten ließ.

3.10.5 Extremtest mit 230400 Baud bei 16 MHz Takt

Nach den Voruntersuchungen bei 8 MHz Takt und 115200 Baud habe ich zusätzlich noch alle verfügbaren USB-Seriell Wandler mit 230400 Baud und einer Autobaud Einstellung 32 des AVRs untersucht. Hierbei wird die Baudrate aus der Übertragungszeit für 4 Datenbits berechnet. Für die Zeitmessung wird ein Zähler mit /8 Vorteiler benutzt. Der Takt des AVR wurde in Schritten von 100 kHz beginnend bei 14 MHz auf 18 MHz erhöht. Für jede Frequenz 40 Downloads von je 10 kByte mit dem modifizierten avrdude (2 Stoppbits) versucht. Die Tabelle 3.40 zeigt die Resulte für die vorhandenen USB-Seriell Wandler.

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

Tabelle 3.40. Fehlerrate beim Extremtest mit 230400 Baud bei 16 MHz Takt

Wieder zeigen der CP2104 und der PL2303 in der Tabelle 3.40 die wenigsten Ausfälle über den gesamten Frequenzbereich. Die serielle Ausgabe der USB-Seriell Wandler wurden bei dieser Unter-

suchung mit zwei Stoppbits betrieben und der **optiboot** Bootloader arbeitete mit den Hardware UART. Bei Verwendung des Software-UARTs von **optiboot** wäre das Ergebnis noch besser ausgefallen. Dummerweise zeigen außer dem Arduino UNO mit dem ATmega16U2 als USB-Seriell Wandler alle anderen USB-Seriell Wandler knapp unter der Taktrate 16 MHz Ausfälle. Der FT232clone hat sogar bei 16 MHz selber Probleme mit dem Download, beim Pololu ist ein ganzer Frequenzbereich um die 16 MHz nicht benutzbar. Fast alle USB-Seriell Wandler versuchen, die Baudrate von 230400 möglichst genau einzuhalten, was wegen der höheren internen Taktrate von etwa 48 MHz bis 96 MHz auch ganz gut gelingt. Dagegen benutzt der ATmega16U2 eine Baudrate von 222222, welche auch vom für die Tests verwendeten ATmega1281 bei 16 MHz erzeugt wird. Dies ist der Grund, weshalb der auf dem Arduino UNO verbaute Mega16U2 keine Probleme mit dem Download zum ATmega1281 im Taktfrequenzbereich um die 16 MHz zeigt.

Wenn der vorhandene USB-Seriell Wandler den Betrieb mit 250000 Baud unterstützt, ist diese Einstellung wahrscheinlich bei einer AVR Taktrate von 16 MHz günstiger, da diese Baudrate ohne Abweichung erzeugt werden kann. Bei einem **optiboot** mit der Autobaud Funktion kann man ja einfach ausprobieren, ob eine höhere Baudrate möglich ist.

3.10.6 Zusammenfassung und Empfehlungen

- USB-Seriell Wandler
  Die USB-Seriell Wandler FT232RL, CP2102, CP2104, PL2303TA und auch der von einer Arduino UNO Platinen Variante verwendete ATMega16U2 haben sich bei den Tests bewährt. Für eine Neuanschaffung würde ich den CP2104 empfehlen, der bei den Tests mit dem größten Baudratenbereich bei der 115200 Baud Einstellung zurecht kam. Mit dem PL2303 Modul hatte ich bei den Tests Schwierigkeiten, der Linux Treiber meldete Fehler und die Ergebnisse bei den Tests waren nicht stimmig. Außerdem mußte ich einen DTR Ausgangs-Pin nachrüsten, da dieser Pin bei meinem Modul fehlte. Erst nachdem ich den PL2303HX Chip gegen einen PL2303TA getauscht hatte, wurden die Ergebnisse besser. Dabei mußte zusätzlich der GND (Pin 7) des PL2303TA Chips auf der Platine nachverdrahtet werden. Nach der Umrüstung des PL2303 Moduls zeigte der PL2303TA gleich gute Ergebnisse wie der CP2104. Jetzt habe ich auch schon chinesische Angebote von Arduino Nano clones mit dem PL2303TA Chip statt dem sonst üblichen CH340G gesehen.

- 2 Stoppbits
  Bei den aufwendigen Tests, die in den Tabellen 3.37 auf Seite 55 und 3.38 auf Seite 56 zusammengefasst sind, hat sich gezeigt, daß die Benutzung eines zweiten Stoppbits für die gesendeten Daten die nutzbare Baudratendifferenz deutlich erhöht. Damit wurde gezeigt, daß sich das Auftreten ungünstiger Baudraten-Kombinationen mit 2 Stoppbits verringert. Leider läßt der Standard avrdude mit Version 6.3 die Verwendung eines zweiten Stoppbits nicht zu. Ich hatte dafür die avrdude Quellen von Version 6.3 so verändert, daß diese Einstellung mit einem Parameter -S wählbar ist. Die Änderungen sind in der Datei ee_test/avrdude.diff festgehalten. Den Geschwindigkeitsverlust durch das zweite Stoppbit halte ich für unerheblich, mit einer höheren Baudrate, die sonst nicht möglich wäre, gewinnt man mehr.

- Wahl des Autobaud Modus
  Mit der letzten Verbesserung der Autobaud Funktion kann auch die einfachste Messung der Baudrate mit der Einstellung BAUD_RATE=22 benutzt werden. Dies ist deswegen interessant, weil diese Version den wenigsten Speicher braucht. Leider wird aber für diese sinnvolle Verbesserung (Vorteiler Reset) ein Assemblerbefehl mehr benötigt, so daß 2 Byte Flash zusätzlich benutzt werden für alle Autobaud Modi mit Takt-Vorteiler für den Zähler (Einer-Ziffer der Baudrate <5). Der Speicherverbrauch ist aber trotzdem geringer im Vergleich zur Benutzung des

Zählers bei voller Taktrate (Einer-Ziffer der Baudrate >4). **Empfohlen wird aber die Baudratenmessung über 4 Datenbits (Modus 32).** Auf die zusätzliche Kontrolle bei Modus 52 kann verzichtet werden. Die Benutzung der vollen Kontrolle des Datenbytes bei Baudrate >59 wird nicht empfohlen, speziell bei Benutzung des Zählers mit Vorteiler ergeben sich deutlich schlechtere Baudratenmessungen. Die Messung der Baudrate über 9 Bits (BAUD_RATE=12) kann zwar uneingeschränkt benutzt werden, die Messung über 4 Bits reicht aber völlig aus und braucht weniger Speicher.

- Benutzung Software UART
  Es gibt mehrere Gründe, die den Einsatz der Software UART Lösung notwendig machen wie die Verwendung anderer Pins für die serielle Kommunikation oder auch die Ein-Pin Kommunikation (RX und TX-Pin gleich). Auch für diesen Fall ist die Benutzung der Autobaud Funktion möglich. Da sowohl die Software-UART Lösung als auch die Autobaud Funktion mehr Flash Speicher benötigen, wird es immer schwieriger, den **optiboot** in 512 Byte unterzubringen. Wenn der Zielprozessor eine größere Bootloader-Seite wie 1024 Byte hat, braucht man sich natürlich um die 512 Byte Grenze nicht zu kümmern. Die Tests, ob die **optiboot** Version in 512 Byte passt, sollte man jedenfalls mit **make** Aufrufen ohne die ISP Option oder mit ISP=0 machen. So kann man mit den Optionen so lange ausprobieren, bis eine geeignete Konfiguration gefunden ist, die in 512 Byte passt. Die Größe der Bootloader-Seite des Zielprozessors wird übrigens bei jedem **make** Aufruf mit der Angabe eines Zielprozessors im Terminal-Protokoll angezeigt.
  Als erstes sollte man die Blink-Funktion der LED mit LED_START_FLASHES=0 abschalten, um Platz zu sparen. Wer unbedingt etwas zum Leuchten braucht, kann die Funktion LED_DATA_FLASH=4 benutzen, denn die braucht nur 4 Byte Flash statt der 56 Byte der Blinkfunktion. Die Tabelle 3.8 auf Seite 38 gibt eine Vorinformation über den benötigten Speicher für alle Einstellmöglichkeiten.
  Mit der Option NO_EARLY_PAGE_ERASE=1 werden zusätzlich 14 Byte eingespart, wobei dadurch lediglich das Laden des Anwenderprogramms etwas langsamer wird.
  Auf die Möglichkeit, das EEprom zu beschreiben, sollte man nach Möglichkeit nicht verzichten. Die Abschaltung der Funktion ist zwar mit SUPPORT_EEPROM=0 auch möglich, spart aber nur etwa 24 Byte ein.

  Wenn mit mehr Störungen der Übertragung zu rechnen ist, weil beispielsweise längere Kabel verwender werden, sollte man nach Möglichkeit aber das Hardware-UART verwenden und höhere Baudraten vermeiden.

3.11 Schlußbemerkung

Als ich die Arbeit mit der **optiboot** Umstellung auf Assembler und Einbau der EEprom-Unterstützung begonnen hatte, hatte ich keine Ahnung, wie viel hier untersucht und erweitert werden könnte. Die erste Aufgabe bestand ja darin, einen **optiboot** zu bauen, der das Laden von EEprom Daten unterstützt und nicht mehr wie 512 Byte Flash braucht. Das erschien mir ein überschaubarer Aufwand zu sein.

Da war aber zusätzlich noch die Unterstützung vieler AVR Prozessoren einzubauen, die dann beschafft, die verschiedenen Bootloader-Seitengrößen berücksichtigt und natürlich auch getestet werden mußten. Die vielen Optionen und die sich daraus ergebenden Programmvarianten erzeugen unterschiedliche Programmlängen, die jetzt immer automatisch berücksichtigt werden. Damit werden Falschsetzungen der AVR-Fuses für alle unterstützten Prozessoren vermieden.

Die Software-UART Lösung wurde optimiert und so ist damit jetzt auch ein Betrieb mit einem IO-Pin möglich. Es wurde die automatische Baudraten-Anpassung (Autobaud) eingebaut und getestet. Dabei können jetzt viele Autobaud-Varianten ausgewählt werden, wobei die Autobaud Funktion sowohl mit dem Hardware-UART als auch mit der Software-UART Lösung von **optiboot** funktioniert.

Die Einstellmöglichkeit des internen RC-Oszillators als Taktgeber für die unterstützten AVR's wurde untersucht und die Einstellmöglichkeit OSCCAL_CORR in **optiboot** integriert. Speziell bei der Kombination Autobaud und Software-UART ist die 512 Byte Grenze für den Flash eine Herausforderung, selbst wenn man die LED-Blinkfunktion abwählt. Natürlich läuft **optiboot** auch dann, wenn mehr als 512 Byte gebraucht werden. Die Grenze war nur meine Zielvorstellung, ich wollte möglichst alle Funktionen in die 512 Byte integrieren. So wurde mit der Option NO_EARLY_PAGE_ERASE die Möglichkeit gefunden, gegebenenfalls noch ein paar Byte einzusparen. Dadurch wird das Flash-Schreiben etwas langsamer, aber wenn beispielsweise die Baudrate im Autobaud Modus höher gewählt werden kann, wird das mehr als ausgeglichen. Der Vorteil bei der Autobaud-Funktion ist ja, daß man die höhere Baudrate wie beispielsweise 115200 Baud bei 8 MHz Takt ausprobieren kann. Wenn das dann nicht funktioniert, kann man auf eine niedrigere Baudrate ausweichen, ohne den Bootloader zu verändern.

So konnte ich in aufwendigen Testreihen nachweisen, daß mit einem zweiten Stoppbit der seriellen Schnittstelle eine größere Abweichung der Baudrate von avrdude toleriert wird. Es wundert ein wenig, daß das bisher nicht aufgefallen ist. Sonst dürfte das zweite Stoppbit wenigstens als Option von avrdude wählbar sein.

Sogar, als ich glaubte, es gäbe nichts mehr zu verbessern, kam bei den abschließenden Tests der Autobaud-Funktion mit verschiedenen USB-Seriell-Wandlern die Idee, den Frequenz-Vorteiler für die Zähler bei der Zeitmessung für Autobaud zurückzusetzen. Das wird selten benutzt, da der Vorteiler von allen integrierten Zählern gemeinsam benutzt wird. Aber hier in **optiboot** kann diese Funktion benutzt werden und führt zu einer Verringerung der Fehlers bei der Autobaud Zeitmessung, wenn der Zähler mit /8 Vorteiler benutzt wird.

Es tut mir leid, daß die vielen Einstell-Möglichkeiten für **optiboot** für einen Benutzer verwirrend sind. Soweit es möglich war, habe ich die Optionen der C-Version von Peter Knight und Bill Westfield übernommen, damit es Benutzer der Ursprungsversion von **optiboot** leichter haben. Sonst hoffe ich aber, daß die Beispiele aus Kapitel 3.8 auf Seite 40 den ersten Einstieg erleichtern. Außerdem reicht immer ein **make** Aufruf mit dem gewünschten Zielprozessor als Parameter aus, um einen lauffähigen **optiboot** als .hex Datei zu erzeugen. Aus dem Terminal-Protokoll kann man entnehmen, welche Einstellung gewählt wurde. Das ist immer eine vorgewählte Taktfrequenz und eine vorgewählte (feste) Baudrate für diesen speziellen Prozessor. Im Bedarfsfall können die Voreinstellungen natürlich geändert werden.

Kapitel 4

Daten der AVR 8-Bit Mikrocontroller

4.1 Signatur Bytes und Standard Fuse Einstellung

Die folgenden Tabellen zeigen die Signatur-Bytes und die Standard-Werte für die Fuses verschiedener AVR-Familien.

	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

Tabelle 4.1. Signatur Bytes der ATtiny Prozessoren und Standard Fuses

Typ	Signature			Default Fuses			Default
	1	2	3	Low	High	Extended	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

Tabelle 4.2. Signatur Bytes der ATmega Prozessoren und Standard Fuses

Typ	Signature			Default Fuses			Default
	1	2	3	Low	High	Extended	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

Tabelle 4.3. Signatur Bytes und Standard Fuses der AT90 und mega..U Prozessoren

4.2 Belegung der 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			

Tabelle 4.4. Belegung der Low Fuse von AVR Prozessoren

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	RST-	SPIEN	WDTON	EESAVE	BOOTSZ1:0		BOOT-
m16U2	DWEN	DISBL	SPIEN	WDTON	EESAVE	BOOTSZ1:0		RST
m32U2	DWEN		SPIEN	WDTON	EESAVE	BOOTSZ1:0		
t26	-	-	-	RST-DISBL	SPIEN	EE-SAVE	BOD-LEVEL	BODEN

Tabelle 4.5. Belegung der High Fuse von AVR Prozessoren

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		

Tabelle 4.6. Belegung der Extended Fuse von AVR Prozessoren

4.3 Mögliche Interne Takt-Frequenzen

Die folgende Tabelle 4.7 zeigt die möglichen Einstellungen für eine interne Taktfrequenz-Erzeugung für verschiedene AVR-Prozessoren.

AVR-Typ	:8 Teiler	Calibrated RC Generator	128kHz	zusätzlich
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

Tabelle 4.7. Interne Taktfrequenzen von AVR Prozessoren

Kapitel 5

Verschiedene USB zu Seriell Wandler mit Linux

Die klassische serielle Schnittstelle wird heute immer weniger verwendet. Bei älteren Rechnern findet man diese Schnittstelle nach RS232 Standard häufiger. Die serielle Schnittstelle ist aber für die Programmierung eines Mikrocontrollers ohnehin nicht sehr praktisch, da die verwendeten Spannungspegel (etwa -12V und +12V) nicht direkt verwendet werden können. Diese Spannungspegel müßten erst wieder mit Pegelwandlern auf die bei Mikrocontrollern gebräuchlichen +5V oder 3.3V zurückgewandelt werden. Praktischer sind für den Zweck der Programmierung die USB zu Seriell Wandler, da diese neben dem benötigten Spannungspegel auch noch die +5V oder +3.3V für die Versorgung des Mikrocontrollers zur Verfügung stellen. Unter Linux werden diese USB-Geräte meistens problemlos erkannt. Es kann aber bei den Zugriffsrechten hapern. Meistens werden die Zugangsknoten wie /dev/ttyUSB1 der Gruppe dialout zugewiesen. Dann sollte man selbst (user) dieser Gruppe angehören. Mit dem Kommando „usermod -a -G dialout $USER" sollte man der Gruppe dialout angehören.

5.1 Der CH340G und der CP2102 Wandler

Untersucht habe ich eine Platine mit der Aufschrift BTE13-005A, auf der ein CH340G Wandler von QinHeng Elektronics untergebracht ist. Die Platine hat einen Minischalter zum Umschalten der VCC Spannung auf 3.3V oder 5V und einen 12 MHz Quarz. Auf der einen Seite befindet sich ein USB-A Stecker und auf der anderen Seite eine 6-polige Pin-Leiste mit den Signalen GND, CTS, VCC, TXD, RXD und DTR. In einem chinesischen Datenblatt zum Chip kann man Angaben zu den unterstützten Baudraten finden. Auf der Platine befindet sich eine mit VCC verbundene LED und eine weitere LED.

Die andere Platine mit dem CP2102 Wandler von Silicon Laboratories Inc. hat keine Aufschrift auf der Platine. Wie bei der CH340G Platine befindet sich auf der einen Seite der USB-A Stecker und auf der gegenüberliegenden Seite eine 6-polige Stiftleiste mit den Signalen 3.3V, GND, +5V, TXD, RXD und DTR. Auf den beiden anderen Seiten läßt sich jeweils eine 4-polige Stiftleiste nachrüsten mit den Signalen DCD, D3R, RTS und CTS sowie RST, R1, /SUS und SUS. Alle für den Bootloader benötigten Signale befinden sich aber auch bei dieser Platine auf der bereits bestückten Stiftleiste. Die Reihenfolge der Belegung ist aber unterschiedlich. Auf der Platine mit dem CP2102 Wandler befinden sich außer dem Wandlerchip sehr wenige Bauteile, eine LED für RXD, TXD und Power ist aber vorhanden.

Mit meinem Linux-Mint 17.2 wurden beide Wandler ohne weiteres erkannt. Mit dem Kommando

lsusb sieht man in der Ausgabe:

```
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
```

Natürlich sind die Angaben zum Bus abhängig vom Rechner und dem verwendeten USB-Port.

Die beim Einstecken automatisch erzeugten Device Namen kann man sofort nach dem Einstecken durch das Kommando „dmesg | tail -20" herausfinden. Beim folgenden Beispiel habe ich beide Ausgaben zusammengefaßt.

```
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
```

Nach diesen Angaben und eigenen Experimenten habe ich die Tabellen 5.1 und 5.2erstellt. Weil sich mit dem neueren Betriebssystem Linux Mint 18.3 teilweise bessere Ergebnisse ergeben haben, habe ich die Meßergebnisse mit diesem System gemessen.

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	(50)	Error		
75	75	75.18	(75)	Error		
100	Error	-	(100)	Error		
110	110	109.3	(120)	Error		
134	134	133.4	(134)	Error		
150	150	150.4	(150)	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
(16000)	Error	-	16000	Error	-	124
19200	19200	19232	19200	19200	19232	103

Tabelle 5.1. geprüfte Baudraten des CH340 und CP2102 im unteren Baud-Bereich

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%

Tabelle 5.2. geprüfte Baudraten des CH340 und CP2102 im oberen Baud-Bereich

Bei beiden Tabellen fällt auf, daß nicht alle von den Herstellern angegebenen Baudraten mit dem Linux stty Kommando einstellbar sind. In den meisten Fällen wird ein Fehler gemeldet, aber leider nicht immer. Beim CP2102 Kontroller wird die Fehlergrenze bei den Baudraten 1 MBaud und 2 MBaud deutlich überschritten. Auch bei den 576 kBaud ist die Abweichung etwas höher als erwartet. Bei der Messung der eingestellten Baudrate fällt nur die Einstellung 576000 und 921600 des CH340G Kontrollers negativ auf. Die anderen Einstellungen bleiben bei der Toleranz unkritisch. Wo die Ursachen für diese Auffälligkeiten liegen, habe ich nicht erörtert. Bei der Dokumentation zum CH340G scheint aber nicht alles mit der gelieferten Version übereinzustimmen. Die Einstellungen 500 kBaud und 1 MBaud sind zwar möglich, aber nicht in der Dokumentation erwähnt.

5.2 Der PL-2303 und der FT232R Wandler

Diesmal habe ich eine Platine mit der Aufschrift „SBT5329" und dem PL-2303RX Wandler von Profilic Technology und eine Platine „FTDI Basic 1" mit einem FT232RL Wandler von Future Technology Devices untersucht.

Die SBT5329 Platine hat auf der einen Seite einen USB-A Stecker und auf der anderen Seite eine 5-polige Stiftleiste mit den Signalen +5V, GND, RXD, TXD und 3.3V. Außerdem findet man auf der Platine noch einen 12 MHz Quarz und drei Leuchtdioden Tx, Rx und Power. Die Steuersignale der seriellen Schnittstelle sind bei dieser Platine nicht auf Stiftleisten herausgeführt. Der PL-2303

Chip stellt aber die Handshake-Signale der seriellen Schnittstelle bereit.

Die FTDI Basic 1 Platine hat auf der einen Seite eine USB-B-Buchse und auf der anderen Seite eine 6-polige Steckerleiste mit den Signalen GND, CTS, 5V, TXD, RXD und DTR. Außer dem FT232RL Chip sind nur wenige weitere Bauteile auf der Platine zu finden. Zwei LED's für Tx und Rx sind jedenfalls auch vorhanden.

Wie auch bei den beiden anderen Platinen werden die Geräte unter Linux Mint 17.1 einwandfrei erkannt.

```
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 zeigen sich die Geräte mit dem Kommando lsusb. Mit dem Kommando „dmesg | tail -20" sieht man dann unmittelbar nach dem Einstecken die Meldungen:

```
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
```

beziehungsweise

```
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 der Tabelle 5.3 wird die Baudrate für den FT232R Kontroller auch unter 300 Baud eingestellt, obwohl der Chip das nicht kann. Aber das stty Kommando akzeptiert diese Einstellungen ohne eine Fehlermeldung. Natürlich ist die resultierende Baudrate für diese Einstellungen falsch.

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

Tabelle 5.3. geprüfte Baudraten des PL-2303 und FT232R im unteren Baud-Bereich

In der Tabelle 5.4 wird die Baudrate für den FT232R Konverter richtig gesetzt, wenn das stty Kommando keinen Fehler meldet. Ich kenne den Grund nicht, warum einige Baudraten für den FT232R Chip nicht vom stty Kommando akzeptiert werden. Nur die Baudrate 576000 könnte vom Chip besser eingestellt werden, als hier gemessen.

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%

Tabelle 5.4. geprüfte Baudraten des PL-2303 und FT232R im oberen Baud-Bereich

5.3 Der USB-serial Wandler mit der ATmega16X2 Software

Auf manchen Arduino UNO Platinen wird ein ATmega16X2 als USB-seriell Wandler benutzt. Deswegen möchte ich die Untersuchungen der einstellbaren Baudraten auch für diese Lösung durchführen. In der ersten Tabelle 5.5 für den unteren Baudraten-Bereich fällt nur negativ auf, daß die Baudraten unter 600 Baud von stty ohne Fehlermeldung akzeptiert werden. Ab 600 Baud aufwärts werden die Einstellungen ohne Fehlermeldung brauchbar eingehalten.

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

Tabelle 5.5. geprüfte Baudraten des ATmega16X2 im unteren Baud-Bereich

Bei der Tablelle 5.6 für den oberen Baudbereich kann man erkennen, daß auch nicht alle Zahlenwerte ohne Fehlermeldung richtig eingestellt werden. Wünschenswert wäre natürlich eine bessere Rückmeldung von stty, welche Baudraten für das angewählte Interface eingestellt werden können.

Mega16X2 supported BaudRate	Mega16X2 stty speed	Mega16X2 measured BaudRate	AVR UBBR @16MHz
28800	Error	-	68
33600	Error	-	59
38400	38400	38.312k	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%

Tabelle 5.6. geprüfte Baudraten des ATmega16X2 im oberen Baud-Bereich

5.4 Der Pololu USB AVR Programmer v2.1

Bei meinem Linux System werden bei eingestecktem Pololu Programmer zwei serielle Schnittstellen erkannt, /dev/ttyACM0 und /dev/ttyACM1 . Mit der ersten serielle Schnittstelle wird die ISP-Schnittstelle angesteuert. Die zweite Schnittstelle /dev/ttyACM1 steht zur freien Verfügung. Die Signale der seriellen Schnittstelle sind auf einer zusätzlichen Buchsenleiste verfügbar (TX, RX, B=DTR, A=frei).

Die Ergebnisse meiner Untersuchungen der zusätzlichen seriellen Schnittstelle habe ich für den unteren Baud-Bereich in der Tabelle 5.7 festgehalten. Ab 300 Baud macht der Pololu ein gutes Bild, alle von stty akzeptierten Baudraten werden sehr gut eingehalten.

getestete BaudRate	stty speed	gemessene BaudRate	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

Tabelle 5.7. geprüfte Baudraten der Pololu seriellen Schnittstelle im unteren Baud-Bereich

Bei den in der Tabelle 5.8 gezeigten Ergebnissen ist bei allen von stty akzeptierten Baudraten nur 576000 unbrauchbar wegen zu hoher Abweichung.

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%

Tabelle 5.8. geprüfte Baudraten der Pololu seriellen Schnittstelle im oberen Baud-Bereich

Kapitel 6

Erstellen des optiboot Bootloaders

Für die Erstellung wird hier ein Linux-System vorausgesetzt, bei dem die folgenden Pakete installiert sind:

- avr-gcc
- binutils-avr
- avrdude
- bash
- bash-completion
- bc
- grep

Einige der Pakete können schon standardmäßig installiert sein. Sie sollten auf der bash Kommandoebene arbeiten. Das kann entweder ein Terminalfenster oder eine virtuelle Konsole sein. Normalerweise kann Linux zwischen 6 virtuellen Konsolen umschalten mit den Tastenkombination **<Strg> + <Alt> + <F1>** bis **<Strg> + <Alt> + <F6>** . In den Konsolen müssen Sie sich mit ihrem Benutzernamen und dem Kennwort anmelden. Sie können aber genau so gut in einem Terminalfenster in der grafischen Oberfläche **(<Strg> + <Alt> + <F7>)** arbeiten. Wenn sie eine Befehlszeile mit geringen Änderungen wiederholen möchten, können Sie die **Cursor Aufwärts** Taste ↑ auf ihrer Tastatur betätigen um einen alten Befehl zurückzuholen. Wenn sie nur einen Parameter ergänzen wollen, können sie den einfach neu dazu schreiben und die Return Taste drücken. Mit der Taste **Cursor links** ← können sie auch an eine zu ändernde Stelle des alten Befehls positionieren. Diese paar einleitenden Sätze sollen Linux Neulingen den Einstieg erleichtern, für alle sind aber die folgenden Hinweise gedacht.

Die Erstellung des optiboot Bootloaders sollte in zwei Hauptschritten erfolgen. Zuerst sollte die Erstellung des Programms auf dem PC erfolgen ohne das Programm auf den Zielprozessor (Atmel Mikrocontroller) zu schreiben. Erst im zweiten Hauptschritt sollte das Laden des Programms auf den Zielprozessor versucht werden. Diese Aufteilung hat den Grund, daß viele Einstellungen schon vorab geprüft werden und die Erstellung schon mit einer Fehlermeldung abgebrochen werden kann. Außerdem sollten Sie das Bildschirm-Protokoll prüfen und somit feststellen, ob die eingegebenen Parameter richtig verarbeitet sind, also ohne Tippfehler eingegeben sind. Für jeden der unterstützten Prozessoren gibt es Voreinstellungen, die aber durch Eingabe von Parametern beim **make** Aufruf geändert werden können. Die Fähigkeiten des optiboot Bootloaders sind im Kapitel 3 ab Seite 16

beschrieben. Die wichtigsten Fähigkeiten sind in einer Liste auf Seite 18 aufgeführt. Die zur Verfügung stehenden Parameter sind in Tabelle 3.3 auf Seite 23 und in Tabelle 3.4 auf Seite 24 sowie in Tabelle 3.5 auf Seite 25 aufgeführt. Wenn sich einige Parameter während den Probeläufen nicht ändern sollen, können diese auch als Umgebungsvariable definiert werden. Ein gutes Beispiel dafür ist der verwendete ISP-Programmertyp. Die entsprechende Variable heißt ISPTOOL. Wenn diese Variable weder als Umgebungsvariable noch als Parameter beim **make** Aufruf angegeben ist, wird **avrisp2** als Typ angenommen. Wenn der Typ für Sie nicht richtig ist, können sie das beim **make** Aufruf ändern wie:

```
make atmega328p ISP=1 ISPTOOL=stk500
```

Alternativ können sie ISPTOOL auch als Umgebungsvariable setzen und **make** ohne diesen Parameter aufrufen wie:

```
export ISPTOOL=stk500
make atmega328p ISP=1
```

Bis zum Ende Ihrer Terminal-Sitzung brauchen sie bei allen nachfolgenden **make** Aufrufen den Parameter ISPTOOL nicht mehr angeben. Dann ist in diesem Terminal der ISP-Programmertyp auf stk500 eingestellt. Wenn Sie nach längerer Arbeitspause nicht mehr wissen, ob und welcher Programmertyp eingestellt ist, gibt das Kommando

```
printenv ISPTOOL
```

eine Auskunft darüber. Wenn die Vorbesetzung für ISPTOOL auf stk500 zukünftig für alle ihre Terminalfenster gelten soll, kann das **export ISPTOOL=stk500** an die versteckte Textdatei ~/.bashrc angehängt werden. Besonders wenn man beabsichtigt, verschiedene AVR Mikrocontroller mit einem neuen Bootloader zu versehen, sollte man vom Prozessortyp abhängige Parameter **nicht** mit der Umgebungsvariable fest einstellen. Beispiele hierfür sind die Parameter LED, LFUSE, HFUSE, EFUSE, BAUD_RATE und AVR_FREQ.

6.1 Ein einfaches Beispiel für die optiboot Erstellung

Um viele Fallstricke bei der Erzeugung eines lauffähigen Bootloaders zu umgehen, wurde die Erzeugung der Bootloader Datei weitgehend automatisiert. Zusätzlich werden Einstellungen auch geprüft und die Erzeugung mit einer entsprechenden Fehlermeldung abgebrochen, wenn beispielweise die gewählte Betriebsfrequenz (AVR_FREQ) nicht zu der mit den Fuses (CKSEL,CKDIV8) eingestellten Möglichkeiten für den Takt paßt. Es muß zwingend ein Zielprozessor für den optiboot Bootloader als erster Parameter beim **make** Aufruf angegeben werden. Sie sollten die Beispiele nach Möglichkeit auch an ihrem PC ausführen. Wenn sie in einen Terminal-Fenster arbeiten, dann wechseln sie dort mit dem **cd** Kommando in ihr optiboot verzeichnis und tippen sie die **make** Kommandos ab. Wenn sie tippfaul sind, können sie auch die PDF Dokumenation öffnen und die **make** Befehle mit der linken Maustaste vollständig markieren und mit der mittleren Maustate in das Terminal-Fenster kopieren. Zum Markieren fahren sie mit festgehaltener linken Maustaste über den vollständigen Befehl oder sie können wahrscheinlich auch mit dem Mauszeiger über der Zeile die linke Maustaste drei Mal tippen.

Beginnen wir mit einem einfachen Beispiel:

```
make atmega328
```

```
Optiboot für 16000000 Hz (16.00 Mhz) Betrieb mit Baudrate 115200 und EEprom Unterstützung
 konfiguriert.
 >>> Starte:: optiboot für AVR atmega328 erstellen
```

```
LED-Pin PB5 benutzt Pin 19-PDIP28 17-TQFP32, mit Spezialfunktionen: SCK PCINT5.
 RX-Pin PD0 benutzt Pin  2-PDIP28 30-TQFP32, mit Spezialfunktionen: PCINT16 RXD.
 TX-Pin PD1 benutzt Pin  3-PDIP28 31-TQFP32, mit Spezialfunktionen: PCINT17 TXD.
avr-gcc -g -Wall -Os -fno-split-wide-types -mrelax -mmcu=atmega328  -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%
--------------------------------------------------------------------------------
# # # # # # # # # # # # # # # # # # # # # #
Bootloader Startadresse: 0x7E00 = 32256
# # # # # # # # # # # # # # # # # # # # # #

   text    data     bss     dec     hex filename
    488       0       0     488     1e8 optiboot.elf
Benötigt 1 Boot Seite mit 512 Bytes, das ist 1.5% des Flash Speichers
BOOTSZ=3, das bedeutet 1 Boot Seite
```

Wenn wir uns das Bildschirm-Protokoll ansehen, zeigen die ersten Zeilen für welchen Mikrocontroller mit welcher Taktrate und welcher Baudrate der seriellen Schittstelle der Bootloader konfiguriert ist. In den nächsten Zeilen werden die benutzten Pins angegeben und welche Sonderfunktionen diese Pins bei diesem Mikrocontroller haben können. Diese Anzeige ist besonders nützlich, wenn eine vom Standard abweichende Konfiguration genutzt wird. Damit kann beispielsweise eine ungünstige Belegung der LED Funktion vermieden werden. Hinter dem Kompileraufruf wird die angestrebte und die tatsächliche Baudrate mit einem prozentualen Fehler angegeben. Danach wird auch die Start-Adresse des Bootloaders und die Zahl der benutzten Boot-Speicherseiten sowie die Größe des Bootloaders angegeben. Trotz des einfachen Aufrufes würde diese Konfiguration für gängige Platinen wir **Arduino UNO** oder **Arduino Nano** passen. Die von der Größe des Bootloaders abhängige Zahl der Boot-Speicherseiten wird übrigens beim Programmieren des ATmega328 automatisch berücksichtigt. So entsteht auch bei deutlich größeren Bootloader-Versionen ein lauffähiger Bootloader. Aus reiner Neugier können wir doch mal testen, wie das Ergebnis wäre, wenn man statt der Assembler-Version die C-Quelle verwenden würde, also:

```
make atmega328 C_SOURCE=1
```

```
Optiboot für 16000000 Hz (16.00 Mhz) Betrieb mit Baudrate 115200 konfiguriert.
 >>> Starte: optiboot für AVR atmega328 erstellen
LED-Pin PB5 benutzt Pin 19-PDIP28 17-TQFP32, mit Spezialfunktionen: SCK PCINT5.
 RX-Pin PD0 benutzt Pin  2-PDIP28 30-TQFP32, mit Spezialfunktionen: PCINT16 RXD.
 TX-Pin PD1 benutzt Pin  3-PDIP28 31-TQFP32, mit Spezialfunktionen: PCINT17 TXD.
avr-gcc -g -Wall -Os -fno-split-wide-types -mrelax -mmcu=atmega328  -fno-diagnostics-show-caret
 -DBAUD_RATE=115200 -DLED_START_FLASHES=3 -DSUPPORT_EEPROM=0 -DLED=pB5 -DUART=00 -DSOFT_UART=0
 -DUART_-DSUPPORT_EEPROM=0RX=pD0 -DUART_TX=pD1 -DF_CPU=16000000 -DHFUSE=hexDE -DLFUSE=hexFF
 -DBOOT_PAGE_LEN=512 -DVerboseLev=2 -c -o optiboot.o optiboot.c
optiboot.c: In function 'main':
optiboot.c:628:7: warning: #warning "BAUD_RATE error greater than 2%" [-Wcpp]
optiboot.c: In function 't1_delay':
optiboot.c:1538:1: warning: control reaches end of non-void function [-Wreturn-type]

--------------------------------------------------------------------------------
BAUD RATE CHECK: Desired: 115200,  Real: 117647, UBRR = 16, Difference=2.12%
--------------------------------------------------------------------------------
# # # # # # # # # # # # # # # # # # # # # #
```

```
Bootloader Startadresse: 0x7E00 = 32256
# # # # # # # # # # # # # # # # # # # # # # #

   text    data     bss     dec     hex filename
    488       0       0     488     1e8 optiboot.elf
Benötigt 1 Boot Seite mit 512 Bytes, das ist 1.5% des Flash Speichers
BOOTSZ=3, das bedeutet 1 Boot Seite
```

Das ist ja eine Überraschung, die Programmlänge ist gleich wie bei der Assembler-Version. Noch einmal prüfen, ob da was schief gelaufen ist. Beim **avr-gcc** Aufruf ist aber an letzter Stelle das Programm optiboot.c und nicht optiboot.S angegeben. Also wird der Parameter „C_SOURCE=1“ richtig verarbeitet. Wenn man genauer hinschaut, fehlt aber in der ersten Zeile des Bildschirm-Protokolls der Hinweis „ und EEprom Unterstützung“. Richtig, in der Parameterliste beim **avr-gcc** Aufruf ist mit „-DSUPPORT_EEPROM=0“ die EEProm Unterstützung tatsächlich abgeschaltet. Die EEProm Unterstützung läßt sich aber sicher wieder einschalten:

```
make atmega328 C_SOURCE=1 SUPPORT_EEPROM=1
```

```
Optiboot für 16000000 Hz (16.00 Mhz) Betrieb mit Baudrate 115200 und EEprom Unterstützung
 konfiguriert.
 >>> Starte: optiboot für AVR atmega328 erstellen
LED-Pin PB5 benutzt Pin 19-PDIP28 17-TQFP32, mit Spezialfunktionen: SCK PCINT5.
 RX-Pin PD0 benutzt Pin  2-PDIP28 30-TQFP32, mit Spezialfunktionen: PCINT16 RXD.
 TX-Pin PD1 benutzt Pin  3-PDIP28 31-TQFP32, mit Spezialfunktionen: PCINT17 TXD.
avr-gcc -g -Wall -Os -fno-split-wide-types -mrelax -mmcu=atmega328  -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.c
optiboot.c: In function 'main':
optiboot.c:628:7: warning: #warning "BAUD_RATE error greater than 2%" [-Wcpp]
optiboot.c: In function 't1_delay':
optiboot.c:1538:1: warning: control reaches end of non-void function [-Wreturn-type]

--------------------------------------------------------------------------------
BAUD RATE CHECK: Desired: 115200,  Real: 117647, UBRR = 16, Difference=2.12%
--------------------------------------------------------------------------------
# # # # # # # # # # # # # # # # # # # # # # #
Bootloader Startadresse: 0x7C00 = 31744
# # # # # # # # # # # # # # # # # # # # # # #

   text    data     bss     dec     hex filename
    592       0       0     592     250 optiboot.elf
Benötigt 2 Boot Seiten, je 512 Bytes, das ist 3.1% des Flash Speichers
BOOTSZ=2, das bedeutet 2 Boot Seiten
```

Jetzt sieht das Ergebnis doch anders aus. Das Programm mit den gleichen Fähigkeiten wie die Assembler-Version ist 104 Byte größer und verbraucht jetzt mit 2 Boot-Speicherseiten doppelt so viel Platz im Flash-Speicher des ATmega328. Das ist also der Grund für die Umstellung von der Programmiersprache C auf Assembler gewesen. Es hat sich gelohnt, um jedes Byte zu kämpfen! Wenn jetzt versucht würde, mit dem zusätzlichen Parameter „ISP=1“ diesen Bootloader auf den ATmega328 zu laden, wäre dieser Bootloader lauffähig, da die Fuses des ATmega und auch die Start-Adresse automatisch an die größere Bootloader-Version angepasst würden. Es ist aber jetzt auch verständlich, warum bei der Standard-Konfiguration für die C-Quelle die EEProm Unterstützung weggelassen wird. Der Bootloader soll standardmäßig in eine Bootseite von 512 Byte passen.

6.2 Exotische Beispiele für die optiboot Erstellung

Wenn wir nun den Standard-Aufruf mit einem anderen Zielprozessor wiederholen, sieht das Ergebnis völlig anders aus:

```
make attiny88
```

```
Optiboot für 8000000 Hz (8.00 Mhz) Betrieb mit Baudrate 115200 und EEprom Unterstützung
 konfiguriert.
 >>> Starte: optiboot für AVR attiny88 erstellen
LED-Pin PB5 benutzt Pin 19-PDIP28 17-TQFP32, mit Spezialfunktionen: SCK PCINT5.
 RX-Pin PD0 benutzt Pin  2-PDIP28 30-TQFP32, mit Spezialfunktionen: PCINT16 .
 TX-Pin PD1 benutzt Pin  3-PDIP28 31-TQFP32, mit Spezialfunktionen: PCINT17 .
avr-gcc -g -Wall -Os -fno-split-wide-types -mrelax -mmcu=attiny88  -fno-diagnostics-show-caret
 -DBAUD_RATE=115200 -DLED_START_FLASHES=3 -DSUPPORT_EEPROM=1 -DVIRTUAL_BOOT_PARTITION=1
 -DLED=pB5 -DUART=00 -DSOFT_UART=01 -DUART_RX=pD0 -DUART_TX=pD1 -DF_CPU=8000000 -DHFUSE=hexDD
 -DLFUSE=hexEE -DBOOT_PAGE_LEN=64 -DVerboseLev=2 -c -o optiboot.o optiboot.S

--------------------------------------------------------------------------------
BAUD RATE CHECK: Desired: 115200,  SoftUART_Real: 115942, Delay: 46*1, Difference=.64%
--------------------------------------------------------------------------------
# # # # # # # # # # # # # # # # # # # # # # #
Bootloader Startadresse: 0x1D80 = 7552
# # # # # # # # # # # # # # # # # # # # # # #

   text    data     bss     dec     hex filename
    588       0       0     588     24c optiboot.elf
Benötigt 10 Flash Seiten, je 64 Bytes, das ist 7.8% des Flash Speichers
Keine Boot Seiten vorhanden!
```

Zunächst ist die Betriebsfrequenz jetzt 8 MHz und es wird eine Software-Emulation der seriellen Schnittstelle auf Pin PD0 und PD1 für den Bootloader benutzt. Das Fehlen einer seriellen Schnittstelle kann man in der Optionsliste für den Kompiler **avr-gcc** aus der automatisch gesetzten Option „-DSOFT_UART=1“ erkennen sowie beim „BAUD RATE CHECK:“ aus dem Wort **SoftUART_Real**. Die angegebene Differenz zur gewünschten Baudrate von 115000 sollte einen aber nicht täuschen. Wenn der ATtiny88 mit dem internen Taktgenerator betrieben wird, hält er wahrscheinlich die Taktfrequenz nicht genau ein und damit weicht auch die errechnete Baudrate in Wirklichkeit anders ab. Außerdem unterstützt der ATtiny88 keine Bootloader-Funktion. Stattdessen wird hier eine Nachbildung per Software benutzt. Der Bootloader wird aber wie bei einer echten Bootloaderfunktion in den oberen Speicherbereich geladen (Bootloader Startadresse:). Deswegen ist das Bootloaderprogramm deutlich größer und benutzt 10 Flashspeicher-Seiten mit jeweils 64 Bytes.

Jetzt noch ein Beispiel mit einem anderen Mikrocontroller:

```
make attiny88
```

Auch hier sieht das Protokoll auf dem Bildschirm wieder anders aus:

```
Optiboot für 8000000 Hz (8.00 Mhz) Betrieb mit Baudrate 115200 und EEprom Unterstützung
 konfiguriert.
 >>> Starte: optiboot für AVR attiny87 erstellen
LED-Pin PA2 benutzt Pin  3-SOIC20 31-MLF32, mit Spezialfunktionen: MISO D0 OC0A ADC2 PCINT2.
 RX-Pin PA0 benutzt Pin  1-SOIC20 29-MLF32, mit Spezialfunktionen: RXLIN RXD ADC0 PCINT0.
 TX-Pin PA1 benutzt Pin  2-SOIC20 30-MLF32, mit Spezialfunktionen: TXLIN TXD ADC1 PCINT1.
avr-gcc -g -Wall -Os -fno-split-wide-types -mrelax -mmcu=attiny87  -fno-diagnostics-show-caret
```

```
 -DBAUD_RATE=115200 -DLED_START_FLASHES=3 -DSUPPORT_EEPROM=1 -DVIRTUAL_BOOT_PARTITION=1
 -DLED=pA2 -DUART=00 -DSOFT_UART=0 -DUART_RX=pA0 -DUART_TX=pA1 -DF_CPU=8000000 -DHFUSE=hexDD
 -DLFUSE=hexE2 -DBOOT_PAGE_LEN=128 -DVerboseLev=2 -c -o optiboot.o optiboot.S

--------------------------------------------------------------------------------
BAUD RATE CHECK: Desired: 115200,  Real: 111111, UBRR = 8, Difference=-3.54%
--------------------------------------------------------------------------------
# # # # # # # # # # # # # # # # # # # # # #
Bootloader Startadresse: 0x1D80 = 7552
# # # # # # # # # # # # # # # # # # # # # #

   text    data     bss     dec     hex filename
    572       0       0     572     23c optiboot.elf
Benötigt 5 Flash Seiten, je 128 Bytes, das ist 7.8% des Flash Speichers
Keine Boot Seiten vorhanden!
```

Dieser Mikrocontroller hat ebenfalls keine Bootloader Unterstützung und besitzt aber eine serielle Schnittstelle, die standardmäßig vom Bootloader unterstüzt wird. Das Bootloader Programm belegt hier 5 Flashspeicher Seiten a 128 Bytes. Diese Beispiele sollen zeigen, was beim **make** Aufruf alles über den Ziel-Prozessor bekannt ist. Wenn man über die Eingabe von Parametern die Standard Einstellung des Bootloaders verändern will, sollte man immer aufmerksam das Bildschirm-Protokoll lesen, ob das Ergebnis so gewünscht wird.

6.3 Beispiel für die optiboot Erstellung mit Sonderfunktion

Doch nun zurück zum weit verbreiteten Mikrocontroller ATmega328. Jetzt wollen wir versuchen, statt einer festen Baudrate eine automatisch ermittelte Baudrate zu benutzen. Das wird mit einer angegebenen Baudrate unter 100 angewählt. Hier werden durch die angegebene Zahl verschiedene Verfahren ausgewählt, die im Unterkapitel 3.7.4 auf Seite 33 beschieben sind. Aus den Untersuchungen ist die Angabe einer 32 sinnvoll:

```
make atmega328 BAUD_RATE=32
```

Das Bildschirm Protokoll sieht jetzt so aus:

```
Optiboot für 16000000 Hz (16.00 Mhz) Betrieb mit automatischer Baudrate und EEprom Unterstützung
 konfiguriert.
 >>> Starte: optiboot für AVR atmega328 erstellen
LED-Pin PB5 benutzt Pin 19-PDIP28 17-TQFP32, mit Spezialfunktionen: SCK PCINT5.
 RX-Pin PD0 benutzt Pin  2-PDIP28 30-TQFP32, mit Spezialfunktionen: PCINT16 RXD.
 TX-Pin PD1 benutzt Pin  3-PDIP28 31-TQFP32, mit Spezialfunktionen: PCINT17 TXD.
avr-gcc -g -Wall -Os -fno-split-wide-types -mrelax -mmcu=atmega328  -fno-diagnostics-show-caret
 -DBAUD_RATE=32 -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

--------------------------------------------------------------------------------
Simple Baudrate measurement implemented in optiboot! (4-bit, Clk/8)
UART Minimum 488 Baud, Difference surely less than 4% up to 160.0 kBaud
--------------------------------------------------------------------------------
# # # # # # # # # # # # # # # # # # # # # #
Bootloader Startadresse: 0x7C00 = 31744
# # # # # # # # # # # # # # # # # # # # # #

   text    data     bss     dec     hex filename
```

```
    552       0       0     552     228 optiboot.elf
Benötigt 2 Boot Seiten, je 512 Bytes, das ist 3.1% des Flash Speichers
BOOTSZ=2, das bedeutet 2 Boot Seiten
```

Hier fällt auf, daß der Bootloader jetzt mit 552 Bytes etwas mehr als eine Bootseite (512 Bytes) braucht. Dann werden aber zwei Boot-Seiten benutzt, also 1024 Bytes. Man könnte diese Konfiguration in den Mikrocontroller laden. Es würden alle Fuses dazu richtig eingestellt. Aber es ist doch schade für den verschenkten Platz im Flash Speicher. Vielleicht hilft es ja, wenn man Funktionen abschaltet, die nicht unbedingt gebraucht werden. Versuchen wir es mal mit der LED, die der Bootloader zu Beginn drei mal blinken läßt. Der Bootloader sollte doch auch ohne blinkende LED laufen, also:

```
make atmega328 BAUD_RATE=32 LED_START_FLASHES=0
```

Jetzt sieht das Bildschirm Protokoll schon viel besser aus:

```
Optiboot für 16000000 Hz (16.00 Mhz) Betrieb mit automatischer Baudrate und EEprom Unterstützung
 konfiguriert.
 >>> Starte: optiboot für AVR atmega328 erstellen
 RX-Pin PD0 benutzt Pin  2-PDIP28 30-TQFP32, mit Spezialfunktionen: PCINT16 RXD.
 TX-Pin PD1 benutzt Pin  3-PDIP28 31-TQFP32, mit Spezialfunktionen: PCINT17 TXD.
avr-gcc -g -Wall -Os -fno-split-wide-types -mrelax -mmcu=atmega328  -fno-diagnostics-show-caret
 -DBAUD_RATE=32 -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 implemented in optiboot! (4-bit, Clk/8)
UART Minimum 488 Baud, Difference surely less than 4% up to 160.0 kBaud
--------------------------------------------------------------------------------
# # # # # # # # # # # # # # # # # # # # # # #
Bootloader Startadresse: 0x7E00 = 32256
# # # # # # # # # # # # # # # # # # # # # # #

   text    data     bss     dec     hex filename
    494       0       0     494     1ee optiboot.elf
Benötigt 1 Boot Seite mit 512 Bytes, das ist 1.5% des Flash Speichers
BOOTSZ=3, das bedeutet 1 Boot Seite
```

Der Bootloader kommt wieder mit einer Bootseite aus. Aber es wird keine LED mehr benutzt, die den Start des Bootloaders anzeigt. Da gibt es doch noch die Option LED_DATA_FLASH. Diese Option schaltet die LED normalerweise ein, wenn die serielle Schnittstelle auf Daten wartet. Mit der speziellen Einstellung 4 wird die LED nur beim Start des Bootloaders eingeschaltet. Das sollte doch reichen um den gestarteten Bootloader zu erkennen und braucht sicher nicht viel Programmplatz, also:

```
make atmega328 BAUD_RATE=32 LED_START_FLASHES=0 LED_DATA_FLASH=4
```

Jetzt wird der LED-Port wieder benutzt, wie das Protokoll zeigt:

```
Optiboot für 16000000 Hz (16.00 Mhz) Betrieb mit automatischer Baudrate und EEprom Unterstützung
 konfiguriert.
 >>> Starte: optiboot für AVR atmega328 erstellen
LED-Pin PB5 benutzt Pin 19-PDIP28 17-TQFP32, mit Spezialfunktionen: SCK PCINT5.
```

```
 RX-Pin PD0 benutzt Pin  2-PDIP28 30-TQFP32, mit Spezialfunktionen: PCINT16 RXD.
 TX-Pin PD1 benutzt Pin  3-PDIP28 31-TQFP32, mit Spezialfunktionen: PCINT17 TXD.
avr-gcc -g -Wall -Os -fno-split-wide-types -mrelax -mmcu=atmega328  -fno-diagnostics-show-caret
 -DBAUD_RATE=32 -DLED_START_FLASHES=0 -DLED_DATA_FLASH=4 -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

--------------------------------------------------------------------------------
Simple Baudrate measurement implemented in optiboot! (4-bit, Clk/8)
UART Minimum 488 Baud, Difference surely less than 4% up to 160.0 kBaud
--------------------------------------------------------------------------------
# # # # # # # # # # # # # # # # # # # # # # #
Bootloader Startadresse: 0x7E00 = 32256
# # # # # # # # # # # # # # # # # # # # # # #

   text    data     bss     dec     hex filename
    498       0       0     498     1f2 optiboot.elf
Benötigt 1 Boot Seite mit 512 Bytes, das ist 1.5% des Flash Speichers
BOOTSZ=3, das bedeutet 1 Boot Seite
```

Das wäre doch eine Alternative zum Standard Bootlader mit fester Baudrate. Bei diesem Bootloader kann man zuerst die Standard-Baudrate von 115200 testen. Wenn es Schwierigkeiten gibt, kann man das Hochladen eines Programms mit einer niedrigeren Baudrate wie 57600 durchführen ohne vorher den Bootloader neu programmieren zu müssen. Auch ein Wechsel der Taktrate mit einem anderen Quarz macht so weniger Probleme.

6.4 Optiboot in den Zielprozessor laden

6.4.1 Voraussetzungen zum Programmieren

Nachdem ein geeigneter Bootlader für einen Mikrocontroller erstellt ist, sollte man sich um das Hochladen des Bootloaders in den Mikrocontroller kümmern. Auf der Platine mit dem Mikrocontroller sollte es eine ISP-Schnittstelle geben. Das ist idealerweise ein 6-poliger oder ein 10-poliger Wannenstecker wie sie auf Abbildung 2.4 auf Seite 12 dargestellt sind. Wir benötigen ebenfalls einen ISP-Programmer, der idealerweise einen gleichen Wannenstecker haben sollte. Einen Flachbandkabel mit passenden Steckbuchsen wird auch benötigt. Das Flachbandkabel wird oft mit dem ISP-Programmer mitgeliefert. Die meisten käuflich erwerbbare ISP-Programmer benutzen zum Verbinden mit dem PC eine USB-Schnittstelle. Die USB-Schnittstelle bietet gegenüber einer seriellen Schnittstelle den Vorteil, daß sie sowohl der Programmer als auch die Mikrocontroller-Platine mit Spannung (5V oder 3,3V) versorgen kann. Damit ist beim Programmieren dann keine weitere Spannungsversorgung nötig. Sie brauchen nur die Zielplatine und den ISP-Programmer mit dem Flachbandkabel zu verbinden und dann den ISP-Programmer mit einer freien USB-Schnittstelle ihres PC zu verbinden. Die Reihenfolge des Zusammensteckens ist beliebig.

Besondere Vorsicht ist geboten, wenn der Atmel Prozessor auf einem separaten Sockel programmiert werden soll. Alle mir bekannten Atmel Prozessoren laufen im Auslieferungszustand mit einem internen Takt, der oft auch geteilt wird. Wenn der Prozessor auf der Platine mit einem Quarz laufen soll, werden als erster Programmier-Schritt die Fuses programmiert. Weil man davon ausgehen muß, daß bei diesem Schritt der Takt noch deutlich geringer ist als beim **make** Aufruf für den Bootloader spezifiziert, wird dieser Schritt generell bei langsamer Bitrate (**avrdude -B 500**) durchgeführt. Dieser Schritt läuft also bei jedem neuen Atmel Prozessor. Wenn der Atmel Prozessor aber schon auf den Quarzbetrieb eingestellt ist, läuft auch schon der erste Programmier-Schritt nicht, da jeglicher Takt für den Prozessor ohne den Quarz fehlt. Aus dem gleichen Grund können die weiteren Programmier-Schritte für den Atmel Prozessor nicht ohne Quarz laufen. Abhilfe für dieses Problem

schafft ein zusätzlich am Programmier-Sockel angebrachter Quarz. Außer dem ersten Programmierschritt (setzen der Fuses) wird die Bitrate (**avrdude -B**) bei der Programmierung automatisch an die gewählte Taktrate (AVR_FREQ) des Atmel Prozessors angepaßt.

6.4.2 Programmieren mit ISP Standardeinstellungen

Nach dem Einstecken des ISP-Programmers in eine USB-Schnittstelle sollte bei einem Linux Sytem das neu eingesteckte USB-Gerät einen Eintrag im System-Protokoll generiert haben. die können sie mit folgendem Kommando prüfen:

```
dmesg -T | tail -20
```

```
...
usb 1-6: new full-speed USB device number 17 using xhci_hcd
usb 1-6: New USB device found, idVendor=03eb, idProduct=2104, bcdDevice= 2.00
usb 1-6: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 1-6: Product: ERFOS AVRISP MkII Clone
usb 1-6: Manufacturer: ERFOS
usb 1-6: SerialNumber: 0000A00128255
```

Dies Beispiel-Protokoll stammt von einem älteren **Diamex AVR-Prog** Programmer und ist mit dmesg -t (ohne Zeit) erzeugt. Sie sollten aber die Option -T verwenden, um die Zeit des Einträge kontrollieren zu können. Dieser Programmer ist zum **Atmel AVRISP mkII** kompatibel und ist als Programmer voreingestellt (ISPPORT=usb , ISPTOOL=avrisp2). Für diesen Programmer braucht an den **make** Aufruf nur der Parameter ISP=1 angehängt zu werden. Für das letzte Beispiel mit der automatischen Baudraten-Erkennung also:

```
make atmega328 BAUD_RATE=32 LED_START_FLASHES=0 LED_DATA_FLASH=4 ISP=1
```

```
Optiboot für 16000000 Hz (16.00 Mhz) Betrieb mit automatischer Baudrate und EEprom Unterstützung
 konfiguriert.
 >>> Starte: optiboot für AVR atmega328 erstellen
LED-Pin PB5 benutzt Pin 19-PDIP28 17-TQFP32, mit Spezialfunktionen: SCK PCINT5.
 RX-Pin PD0 benutzt Pin  2-PDIP28 30-TQFP32, mit Spezialfunktionen: PCINT16 RXD.
 TX-Pin PD1 benutzt Pin  3-PDIP28 31-TQFP32, mit Spezialfunktionen: PCINT17 TXD.
avr-gcc -g -Wall -Os -fno-split-wide-types -mrelax -mmcu=atmega328  -fno-diagnostics-show-caret
 -DBAUD_RATE=32 -DLED_START_FLASHES=0 -DLED_DATA_FLASH=4 -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

--------------------------------------------------------------------------------
Simple Baudrate measurement implemented in optiboot! (4-bit, Clk/8)
UART Minimum 488 Baud, Difference surely less than 4% up to 160.0 kBaud
--------------------------------------------------------------------------------
# # # # # # # # # # # # # # # # # # # # # #
Bootloader Startadresse: 0x7E00 = 32256
# # # # # # # # # # # # # # # # # # # # # #

   text    data     bss     dec     hex filename
    498       0       0     498     1f2 optiboot.elf
Benötigt 1 Boot Seite mit 512 Bytes, das ist 1.5% des Flash Speichers
BOOTSZ=3, das bedeutet 1 Boot Seite

####### Start von program_target.sh für atmega328 #############
   Die Fuses in program_target.sh sind gesetzt auf lfuse=0xFF, hfuse=0xDE, efuse=0xFD
Bootloader HFUSE wird auf 0xDE gesetzt, OK!
BootLoader Startadresse ist gesetzt auf 0x7E00, 32256
```

```
##### Lösche den atmega328 und setze die Fuses
avrdude  -c avrisp2 -B 200 -p atmega328 -P usb -b 115200 -q -q -e  -U efuse:w:0xFD:m
 -U hfuse:w:0xDE:m -U lfuse:w:0xFF:m
avrdude error: expected signature for ATmega328 is 1E 95 14
        double check chip or use -F to override this check
#-#-#-#-#-#-#-#
*** avrdude *** meldet Fehler mit Rückgabewert 1!
#-#-#-#-#-#-#-#
Wenn avrdude keine Kommunikation mit dem Programmer gestartet hat,
kann das an einer falschen ISPPORT [jetzt: usb] liegen!
Suche vorhandene serielle Schnittstellen auf Ihrem Linux-System:


Kein Gerät mit Namen /dev/ttyUSBx oder /dev/ttyACMx gefunden! x steht für eine vergebene Nummer!

Prüfen Sie die Verbindung ihres ISP-Programmers zu einer USB-Schnittstelle ihres PCs.
Wenn sie unsicher sind, welcher Programmer eingesteckt ist, kann das Kommando 'lsusb' helfen.
Das Kommando 'lsusb' zeigt alle angeschlossenen USB-Geräte ihres Systems.

Sie können mehr über den gerade angeschlossene Programmer erfahren mit einem Kommando
wie 'dmesg | tail -20'. 'dmesg' gibt die gesamte Logdatei ihres Linux-Systems aus und
'| tail -20' zeigt nur die letzten 20 Zeilen davon.

Wenn avrdude die Kommunikation zu derm Ziel-Prozessor mit dem ISP-Programmer gestartet hat,
sollte der gesetzte ISPPORT richtig sein, Deshalb sollten Sie die Kabelverbindung
zwischen dem ISP-Programmer und Ihrem AVR Prozessor prüfen
Sie sollten USB-Hubs für den Anschluß des ISP-Programmers vermeiden.
Diese können Zeitprobleme mit dem halb-duplex Protokoll von avrdude verursachen.
```

Da ist doch etwas schiefgelaufen! Das Programm **avrdude** hat mit einem Fehler abgebrochen, der Grund ist an der **avrdude** Fehlermeldung zu erkennen:

```
avrdude error: expected signature for ATmega328 is 1E 95 14
        double check chip or use -F to override this check
```

Das Programm **avrdude** erwartet eine andere Signatur des Mikrocontrollers. Die nachfolgenden Ratschläge sind uninteressant. Das Programm **avrdude** kehrt bei jedem Fehler mit Rückgabewert 1 zurück, der aufrufende Script geht aber von einem Kommunikationsproblem des Programm **avrdude** aus. Die Ursache für den Fehler ist aber anders, auf der Arduino Nano Platine ist ein ATmega328P und nicht ein ATmega328 verbaut. Die beiden Prozessoren haben aber eine unterschiedliche Signature. Das hat **avrdude** festgestellt, also läuft die Kommunikation zwischen avrdude und dem ATmega328P! Das ist jedoch schnell korrigiert:

```
make atmega328p BAUD_RATE=32 LED_START_FLASHES=0 LED_DATA_FLASH=4 ISP=1
```

```
Optiboot für 16000000 Hz (16.00 Mhz) Betrieb mit automatischer Baudrate und EEprom Unterstützung
 konfiguriert.
 >>> Starte: optiboot für AVR atmega328p erstellen
LED-Pin PB5 benutzt Pin 19-PDIP28 17-TQFP32, mit Spezialfunktionen: SCK PCINT5.
 RX-Pin PD0 benutzt Pin  2-PDIP28 30-TQFP32, mit Spezialfunktionen: PCINT16 RXD.
 TX-Pin PD1 benutzt Pin  3-PDIP28 31-TQFP32, mit Spezialfunktionen: PCINT17 TXD.
avr-gcc -g -Wall -Os -fno-split-wide-types -mrelax -mmcu=atmega328p  -fno-diagnostics-show-caret
 -DBAUD_RATE=32 -DLED_START_FLASHES=0 -DLED_DATA_FLASH=4 -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
```

```
--------------------------------------------------------------------------
Simple Baudrate measurement implemented in optiboot! (4-bit, Clk/8)
UART Minimum 488 Baud, Difference surely less than 4% up to 160.0 kBaud
--------------------------------------------------------------------------
# # # # # # # # # # # # # # # # # # # # # #
Bootloader Startadresse: 0x7E00 = 32256
# # # # # # # # # # # # # # # # # # # # # #

   text    data     bss     dec     hex filename
    498       0       0     498     1f2 optiboot.elf
Benötigt 1 Boot Seite mit 512 Bytes, das ist 1.5% des Flash Speichers
BOOTSZ=3, das bedeutet 1 Boot Seite

####### Start von program_target.sh für atmega328p #############
   Die Fuses in program_target.sh sind gesetzt auf lfuse=0xFF, hfuse=0xDE, efuse=0xFD
Bootloader HFUSE wird auf 0xDE gesetzt, OK!
BootLoader Startadresse ist gesetzt auf 0x7E00, 32256
##### Lösche den atmega328p und setze die Fuses
avrdude  -c avrisp2 -B 200 -p atmega328p -P usb -b 115200 -q -q -e  -U efuse:w:0xFD:m
 -U hfuse:w:0xDE:m -U lfuse:w:0xFF:m
##### Schreibe den optiboot Bootloader auf atmega328p und setze die Lock Bits
avrdude  -c avrisp2 -B 1.50 -p atmega328p -P usb -b 115200 -D
 -U flash:w:optiboot_atmega328p.hex:i -U lock:w:0xef:m

avrdude: AVR device initialized and ready to accept instructions
avrdude: device signature = 0x1e950f (probably m328p)
avrdude: reading input file optiboot_atmega328p.hex for flash
         with 500 bytes in 2 sections within [0x7e00, 0x7fff]
         using 4 pages and 12 pad bytes
avrdude: writing 500 bytes flash ...

Writing | ################################################## | 100% 0.01 s

avrdude: 500 bytes of flash written
avrdude: verifying flash memory against optiboot_atmega328p.hex

Reading | ################################################## | 100% 0.00 s

avrdude: 500 bytes of flash verified
avrdude: reading input file 0xef for lock
         with 1 byte in 1 section within [0, 0]
avrdude: writing 1 byte lock ...
avrdude: 1 byte of lock written
avrdude: verifying lock memory against 0xef
avrdude: 1 byte of lock verified

avrdude done.  Thank you.
```

Jetzt hat alles geklappt, der Bootloader ist auf der Arduino Nano Platine installiert. Der Vorteil dieser Art von ISP-Programmern ist, daß sie vom Programm **avrdude** gefunden werden, wenn als Schnittstelle (-P) „usb“ angegeben wird. Wenn sie die Zielplatine mit dem frisch installierten Bootloader unter Spannung setzen, blinkt die LED langsam. Das kommt daher, daß zuerst der Bootloader gestartet wird. Der schaltet die LED ein. Nach dem eingestellten Time-out wird versucht, das Anwenderprogramm zu starten. Dabei geht die LED wieder aus. Da kein Anwenderprogramm installiert ist, läuft der Programmzähler von 0 immer weiter hoch bis wieder der Bootloader erreicht wird und die LED wieder einschaltet. Sobald ein Anwender-Programm über die serielle Schnittstelle geladen ist, verschwindet dieser Effekt und die LED bleibt nach dem Time-out dunkel sofern sie nicht vom Anwenderprogramm anders benutzt wird.

6.4.3 ISP-Programmieren mit USB-seriell Schnittstelle

Viele andere Programmern benutzen nicht eine direkte USB-Schnittstelle. Die benutzen stattdessen einen USB-seriell Wandler, wobei der ISP-Programmer an die serielle Schnittstelle angeschlossen ist. Das Betriebssystem erkennt den USB-seriell Wandler und weist der Seriell-Schnittstelle einen Eintrag im Dateisystem für den Zugriff zu. Bei Linux Systemen ist das normalerweise ein Eintrag in der Form /dev/ttyACMx, wobei das x für eine fortlaufende Nummer ist. Wenn der USB-seriell Wandler aus einem Spezialchip wie FT232 oder CH341 besteht, vergibt ein Linux System einen Eintrag in der Form /dec/ttyUSBx, wobei x wieder für eine fortlaufende Nummer steht. Da die Spezialchips relativ teuer waren, wird aber oft eine Emulation mit einem Mikrocontroller verbaut. Sowohl bei der Form /dev/ttyACMx als auch bei der Form /dev/ttyUSBx ist aber aber so, das sich das x in dem Namen aus der Reihenfolge des Einsteckens in die USB-Schnittstellen des PCs ergibt. Ein einmal vergebener Name wird aber bis zum Ausstecken beibehalten. Wenn der erste eingesteckte USB-seriell Wandler den Namen /dev/ttyACM0 erhält, erhält der zweite eingesteckte USB-seriell Wandler den Namen /dev/ttyACM1 . Diesen Namen behält der zweite USB-seriell Wandler auch dann, wenn der erste USB-seriell Wandler wieder ausgesteckt wird. Neu eingesteckte USB-Geräte erzeugen aber einen Eintrag im System-Ringpuffer. Der Inhalt des System-Ringpuffers wird mit dem Kommando dmesg ausgegeben. Da der System-Ringpuffer aber aus vielen Zeilen besteht, braucht man sich normalerweise nur die letzten Zeilen anzusehen. Ein geeignetes Kommando zum Ansehen der letzten 20 Zeilen des System-Ringpuffers ist:

```
dmesg -t | tail -20
```

Hier wurde statt **-T** ein **-t** verwendet, da die Zeit beim gedruckten Beispiel uninteressant ist. Sie sollten aber **dmesg -T** verwenden um die Zeit des Eintrags kontrollieren zu können. Kurz nach dem Einstecken eines **Diamex EXA-PROG** Programmers sieht man so folgendes:

```
...
usb 1-3: new full-speed USB device number 24 using xhci_hcd
usb 1-3: New USB device found, idVendor=16c0, idProduct=2a9b, bcdDevice=45.80
usb 1-3: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 1-3: Product: EXA-PROG
usb 1-3: Manufacturer: ERFOS
usb 1-3: SerialNumber: 58491-80751-303
cdc_acm 1-3:1.0: ttyACM0: USB ACM device
```

Wenn vorher schon eine andere USB-seriell Schnittstelle eingesteckt wurde, meldet sich der gleiche Programmer im System-Ringpuffer so:

```
...
usb 1-4: new full-speed USB device number 29 using xhci_hcd
usb 1-4: New USB device found, idVendor=16c0, idProduct=2a9b, bcdDevice=45.80
usb 1-4: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 1-4: Product: EXA-PROG
usb 1-4: Manufacturer: ERFOS
usb 1-4: SerialNumber: 58491-80751-303
cdc_acm 1-4:1.0: ttyACM1: USB ACM device
```

Man sieht in der letzten Zeile, daß sich der Name von ttyACM0 auf ttyACM1 geändert hat. Somit hat man für diesen Typ von ISP-Programmern eine Möglichkeit, den Namen der Schnittstelle für **avrdude** richtig einzustellen. Der Name der Schnittstelle wird beim **avrdude** Programm mit dem Parameter -P angegeben. Beim **make** Aufruf wird die Schnittstelle mit dem Parameter ISPPORT angegeben. Das Protokoll für den ISP-Programmer wird beim Programm **avrdude** mit dem Parameter

-c angegeben, beim **make** Aufruf heißt der entsprechende Parameter ISPTOOL. Bei entsprechender Einstellung der DIP-Schalter unterstützt der Diamex EXA-PROG das stk500 Protokoll von Atmel unterstützt. Die Baudrate der seriellen Schnittstelle (arduino Parameter -b) ist auf 115000 eingestellt und braucht in der Regel nicht mit dem Parameter ISPSPEED geändert werden. Jetzt klappt das Laden eines Standard optiboot-Bootloaders mit der Angabe von zwei zusätzlichen Parametern:

```
make atmega328p ISP=1 ISPPORT=/dev/ttyACM0 ISPTOOL=stk500
```

```
Optiboot für 16000000 Hz (16.00 Mhz) Betrieb mit Baudrate 115200 und EEprom Unterstützung
 konfiguriert.
 >>> Starte: optiboot für AVR atmega328p erstellen
LED-Pin PB5 benutzt Pin 19-PDIP28 17-TQFP32, mit Spezialfunktionen: SCK PCINT5.
 RX-Pin PD0 benutzt Pin  2-PDIP28 30-TQFP32, mit Spezialfunktionen: PCINT16 RXD.
 TX-Pin PD1 benutzt Pin  3-PDIP28 31-TQFP32, mit Spezialfunktionen: 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%
--------------------------------------------------------------------------------
# # # # # # # # # # # # # # # # # # # # # # #
Bootloader Startadresse: 0x7E00 = 32256
# # # # # # # # # # # # # # # # # # # # # # #

   text    data     bss     dec     hex filename
    488       0       0     488     1e8 optiboot.elf
Benötigt 1 Boot Seite mit 512 Bytes, das ist 1.5% des Flash Speichers
BOOTSZ=3, das bedeutet 1 Boot Seite

####### Start von program_target.sh für atmega328p #############
   Die Fuses in program_target.sh sind gesetzt auf lfuse=0xFF, hfuse=0xDE, efuse=0xFD
Bootloader HFUSE wird auf 0xDE gesetzt, OK!
BootLoader Startadresse ist gesetzt auf 0x7E00, 32256
##### Lösche den atmega328p und setze die Fuses
avrdude  -c stk500 -B 200 -p atmega328p -P /dev/ttyACM0 -b 115200 -q -q -e  -U efuse:w:0xFD:m
 -U hfuse:w:0xDE:m -U lfuse:w:0xFF:m
##### Schreibe den optiboot Bootloader auf atmega328p und setze die Lock Bits
avrdude  -c stk500 -B 1.50 -p atmega328p -P /dev/ttyACM0 -b 115200 -D i
 -U flash:w:optiboot_atmega328p.hex:i -U lock:w:0xef:m

avrdude: AVR device initialized and ready to accept instructions
avrdude: device signature = 0x1e950f (probably m328p)
avrdude: reading input file optiboot_atmega328p.hex for flash
         with 490 bytes in 2 sections within [0x7e00, 0x7fff]
         using 4 pages and 22 pad bytes
avrdude: writing 490 bytes flash ...

Writing | ################################################## | 100% 0.02 s

avrdude: 490 bytes of flash written
avrdude: verifying flash memory against optiboot_atmega328p.hex

Reading | ################################################## | 100% 0.00 s

avrdude: 490 bytes of flash verified
avrdude: reading input file 0xef for lock
         with 1 byte in 1 section within [0, 0]
```

```
avrdude: writing 1 byte lock ...
avrdude: 1 byte of lock written
avrdude: verifying lock memory against 0xef
avrdude: 1 byte of lock verified

avrdude done.  Thank you.
```

Dummerweise weiß das Betriebssystem nichts über den an die serielle Schnittstelle angeschlossenen ISP-Programmer. Für die Einstellung des Parameters ISPTOOL (Option -c für **avrdude**) sollte das Handbuch des Programmers oder der Hersteller Auskunft erteilen können. Wenn die Entwicklung des ISP-Programmers aus einem Projekt stammt, gibt die Dokumentation des Projektes hoffentlich Hinweise. Das Programm **avrdude** unterstützt viele Programmer. wenn Sie **avrdude** nur mit der Spezifikation des Zielprozessors „-p atmega328p“ aufrufen, werden Ihnen 127 verschiedene Möglichkeiten eines Programmers aufgelistet. Wenn man die Einstellungen ausschließt, die kein ISP-Protokoll verwenden, verbleiben immer noch 83 Möglichkeiten. Deswegen sollten Sie jede Möglichkeit nutzen, eine geeignete Einstellung für Ihren ISP-Programmer zu finden. Manchmal gibt der Name der von **avrdude** angezeigten Programmer Einstellungen einen Hinweis auf die Eignung dieser Einstellmöglichkeit wie die folgenden Beispiele:

- arduinoisp = Arduino ISP Programmer
- usbasp = USBasp, http://www.fischl.de/usbasp/
- usbtiny = USBtiny simple USB programmer, https://learn.adafruit.com/usbtinyisp
- dragon_isp = Atmel AVR Dragon in ISP mode

Literaturverzeichnis

[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 Userguide,*. 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