

Der Bootloader

Bei unserer Attiny-Platine werden die Programme über die serielle Schnittstelle übertragen. Grundsätzlich wäre es möglich, hierzu auf die im Attiny-Mikrocontroller fest eingebauten SPI-Funktionen zurückzugreifen. Allerdings würden die Ladevorgänge dabei sehr lange dauern - insbesondere dann, wenn ein USB-COM-Konverter zum Einsatz kommt.

Deswegen haben wir auf jedem Attiny-Mikrocontroller einen so genannten Bootloader installiert; dabei handelt es sich um ein kleines Programm, welches Daten über die serielle Schnittstelle vom Uploader-Programm des PCs entgegennimmt und im Programmspeicher des Attiny ablegt. Auch wenn der Attiny nicht an eine elektrische Quelle angeschlossen ist, geht der Inhalt des Programmspeichers nicht verloren. Da das Uploader-Programm und auch der Bootloader so konzipiert sind, dass der Bootloader niemals überschrieben wird, bleibt der Bootloader im Attiny, solange man keine andere Programmier-Software als das Uploader-Programm benutzt.

Wie funktionieren nun Bootloader und Uploader-Programm? Der Bootloader befindet sich im oberen Bereich des Speicherplatzes; er beginnt bei der Adresse \$03B0 und erstreckt sich bis zum Ende des Speicherbereichs bei \$03FF. Auf der Adresse \$0000, also ganz am Anfang des Speicherbereichs steht ein Sprungbefehl zu diesem Bootloader.

Sobald der Mikrocontroller an eine elektrische Quelle angeschlossen wird (oder wenn ein Reset ausgeführt wird), springt der Attiny an den Anfang seines Programmspeichers und beginnt mit der Bearbeitung der Befehle. In unserem Fall trifft er zunächst auf den Sprung-Befehl `rjmp $03B0`. Er springt daher zur Adresse \$03B0. Mit seinen ersten Befehlen kontrolliert der Bootloader, ob der Taster T1 gedrückt ist. Ist dies der Fall, wartet er, bis der Taster losgelassen wurde, initialisiert dann die serielle Schnittstelle des Attiny, und wartet schließlich auf den Empfang eines ersten Bytes über die serielle Schnittstelle.

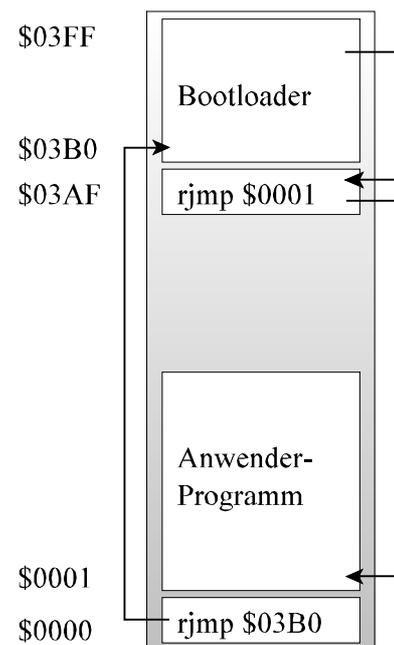


Abbildung 1

Das Uploader-Programm sendet als Erstes ein Startbyte (0); dieses wird von dem Attiny quittiert, indem dieser seinerseits ein Byte an den Uploader zurückschickt (105). Dieser Code 105 signalisiert dem Uploader-Programm, dass die serielle Kommunikation mit dem Attiny funktioniert. Nun sendet der Uploader den Code 201 an den Bootloader des Attiny; damit zeigt er ihm an, dass nun ein Programmdatenblock von 16 Wörtern (Doppelbytes) übertragen werden soll¹. Der Attiny quittiert diese Ankündigung, indem er den

¹In Wirklichkeit muss hier noch zusätzlich die Adresse übertragen werden, bei der der Block im Attiny abgelegt werden soll.

Code 1 zurücksendet. Nun erfolgt die eigentliche Datenübertragung. Jedes empfangene Doppelbyte wird vom Attiny als Echo wieder zurückgeschickt; so kann das Uploader-Programm kontrollieren, ob alle Daten korrekt übertragen worden sind. Gleichzeitig legt der Bootloader den Block im Anwenderprogrammbereich ab. Er beginnt bei der Adresse \$0000. Dabei hat der Uploader zuvor den Sprungbefehl "rjmp 03B0" automatisch in den ersten Block eingefügt; so wird garantiert, dass bei einer späteren Benutzung der Bootloader wieder angesprungen werden kann.

Auf die gleiche Weise werden die nächsten Blöcke übertragen. (Der Fachausdruck für einen solchen Block ist übrigens "page".) Hat das Uploader-Programm den letzten Block übertragen, so sendet es den Code 203 an den Bootloader; der weiß nun, dass die Datenübertragung abgeschlossen ist, quittiert dies durch das Senden des Codes 11 und springt zur Adresse \$03AF. Von dort geht es gleich weiter zur Adresse \$0001, dem Anfang des Anwenderprogramms. Nach dem Laden des Programms wird also das Anwenderprogramm sofort gestartet.

Was geschieht nun, wenn der Attiny gestartet wird, ohne dass Ta1 betätigt wird oder ein entsprechendes Signal von RXD an PortB.3 gelangt? In diesem Fall wird zunächst ein Sprung zur Adresse \$03B0 durchgeführt, d. h. der Bootloader wird wieder gestartet. Dieser stellt nun fest, dass T1 nicht betätigt wird und springt sogleich zur Adresse \$03AF; von dort geht es zur Adresse \$0001 und die Befehle des Anwenderprogramms werden ausgeführt. Da dies in Bruchteilen von Millisekunden geschieht, hat der Anwender den Eindruck, dass nach dem Einschalten sofort das Anwenderprogramm startet.

Der aufmerksame Leser wird sich längst gefragt haben, warum der Bootloader nicht direkt zum Anwenderprogramm, sondern erst über einen Umweg, nämlich den Sprungbefehl in Adresse \$03AF, dorthin springt. Der Grund liegt darin, dass ein Anwenderprogramm manchmal bei einer höheren Adresse als \$0001 beginnen muss. Insbesondere beim Einsatz von Interrupts ist dies unumgänglich. Gehen wir also einmal davon aus, dass das Anwenderprogramm erst bei der Adresse \$0005 beginnen soll. Compiler wie BASCOM erzeugen dann in der Adresse \$0000 automatisch den Sprungbefehl rjmp \$0005. Unser Uploader-Programm merkt sich diesen Sprungbefehl und ersetzt ihn durch einen Sprung zum Bootloader. Zusätzlich wird in der Adresse \$03AF der Sprungbefehl rjmp \$0001 durch den Sprungbefehl rjmp \$0005 ersetzt. Diese Anpassung bedeutet keinen Eingriff in den Bootloader selbst; dieser beginnt ja erst in der nächsten Adresse. Springt der Bootloader nun zum Abschluss seiner Arbeit an die Adresse \$03AF, so wird er an den korrekten Beginn des Anwenderprogramms weitergeleitet.

Der Bootloader kann natürlich nicht mithilfe des Bootloaders selbst auf den Attiny transportiert werden. Hier greift man auf die ISP-Programmierung zurück. Da der Bootloader nur wenige Bytes umfasst und in der Regel auch nur ein einziges Mal auf den Attiny übertragen werden muss, wird man die längere Brennzeit wohl gut verkraften. Näheres dazu erfahren Sie im Kapitel "ISP-Programmierung".

Und so sieht das Bootloader-Programm aus:

```
; Bootloader für Attiny 2313-Experimentierplatine
; von E. Eube, G. Heinrichs und U. Ihlefeldt

; arbeitet zusammen mit Uploader.exe

; liest OSCCAL-Wert aus Eeprom-Adresse 127
; springt in den Bootloader, wenn T1 bei Einschalten gedrückt
; ansonsten wird Kalibrierprogramm gestartet

; Version 1.0.1
; 17.11.12

.include "tn2313def.inc"

.def param = r16
.def uartparam = r18
.def Kom = r19
.def temp = r20
.def RWCount = r23
.def spmcsrval = r22
.def epromadresse = r21
.equ DTR = 3
.equ LED = 6
.equ TAKT = 4000000 ; HZ

; r26 bis r31 reserviert für pointer

; Verzweigung zum Bootloader
.org $0000
    rjmp $03B0

; Kalibrierprogramm
.org $0001

; hier folgt das Kalibrierprogramm...

; letzte Adresse vor Bootloader
.org $03AF
    rjmp $0001 ; wird vom Uploader überschrieben

; Es folgt der eigentliche Bootloader...

.org $03B0

    sbi portd, DTR ; für Taster Tal vorbereiten

; zuerst OSCCAL aus Eprom lesen...
reprom:
    ; sbic eecr, eewe ; am Anfang nicht nötig!
    ; rjmp reprom
    ldi epromadresse, 127
    out eear, epromadresse
    sbi eecr, eere
    in param, eedr
    out osccal, param

; nun Stackpointer initialisieren
    ldi temp, LOW(RAMEND) ;Stackpointer auf RAMEND setzen
    out SPL, temp

GuckObStarten:
    sbis pinD, DTR ;Taster Tal=0 (geschlossen) -> BOOTLOADER sonst normales
                    ;Programm
    rjmp Start
    rjmp $03AF

Start:
    sbi ddrD, LED ; für Anzeige LED
    sbi portD, LED ; Anzeige: D.5 = an, wenn Start
    sbis pinD, DTR ; warten bis Tal=1 (offen)
    rjmp Start
```

```
; UART initialisieren:
sbi UCSRB, 4 ; UCR=UCSRB=0x0B RXEN=Bit4 RX aktivieren
sbi UCSRB, 3 ; UCR=UCSRB=0x0B UDRE=Bit3 TX aktiv
ldi uartparam, 400000/(9600*16)-1 ;Baudrate 9600 einstellen
out UBRRRL, uartparam
rcall rdc com ; warten auf Startbyte
cbi portd, LED ; LED aus, wenn Startbyte erhalten
ldi param, 105 ; ACK senden
rcall wrcom

MainSchlaufe:
rcall rdc com
mov kom, param

B201: ; Block Schreiben
cpi kom, 201
brne B203
rcall Schreib_Block
ldi param, 1
rcall wrcom

B203: ; Anwenderprogramm Start
cpi kom, 203
brne zurück ; Wenn keine Zahl zutrifft zum Anfang
ldi param, 11 ; ACK
rcall wrcom
rcall wartelms ; 1 ms warten
cbi UCSRB, 4 ; RX deaktivieren
cbi UCSRB, 3 ; TX deaktivieren
rjmp $03AF

Zurück:
rjmp MainSchlaufe

Schreib_Block:
ldi RWCount, 16
rcall rdc com ; Block-Adresse vom Master
mov ZH, param
rcall rdc com
mov ZL, param

Loeschen:
ldi spmcsrval, 3
out spmcsr, spmcsrval
spm

Schreib_schleife:
rcall rdc com
mov r0, param
rcall rdc com
mov r1, param

ldi spmcsrval, 1 ; Puffer schreiben
out spmcsr, spmcsrval
spm
inc ZL ; Adresse um 1 Word erhöhen
inc ZL
mov param, r0 ; Bestätigung
rcall wrcom

mov param, r1
rcall wrcom

dec RWCount
brne Schreib_schleife
subi ZL, 32
ldi spmcsrval, 5 ; Puffer in FLASH schreiben
out spmcsr, spmcsrval
spm
ret

rdcom:
sbis UCSRA, 7 ; USR=UCSRA=0x0B RXC=Bit7
rjmp rdc com ; warten, bis UDR bereit
in param, UDR
```

```
ret

wrcom:
    sbis    UCSRA, 5           ; USR Bit5 = UDRC
    rjmp   wrcom             ; warten, bis UDR bereit
    out    UDR, param
    ret

wartelms:
; Vorbereitung Zähler für Warteschleife
    ldi    r24, low(TAKT/4000)
    ldi    r25, high(TAKT/4000)
warteschleife:
    sbiw   r24, 1
    brne   warteschleife
    ret
```