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 für den Attiny2313? 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.

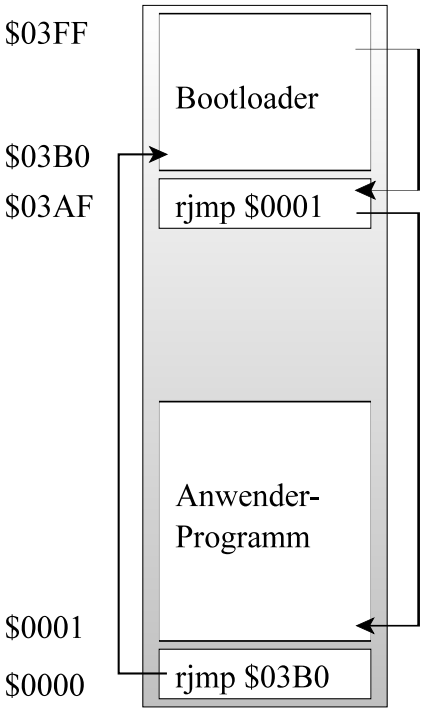

Abbildung 1

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.

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 Block des Anwender-Programms von

16 Wörtern (Doppelbytes) übertragen werden soll[1]. Der Attiny quittiert diese Ankündigung, indem er den 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 allerdings 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 über die DTR-Leitung an PortD.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 PortD.3 = 1 ist und springt damit 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 nun 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 hier 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”.

---

[1] In Wirklichkeit muss hier noch zusätzlich die Adresse übertragen werden, bei der der Block im Flash-Speicher des Attiny abgelegt werden soll.

Beim Attiny4313 ist die Vorgehensweise fast identisch; da er doppelt so viel Flash-Speicher besitzt und die einzelnen Blöcke nun 32 Wörter besitzen, muss nun statt Abb. 1 die Abb. 2 benutzt werden. Das Bootloader-Programm für den Attiny4313 sieht dann so aus:

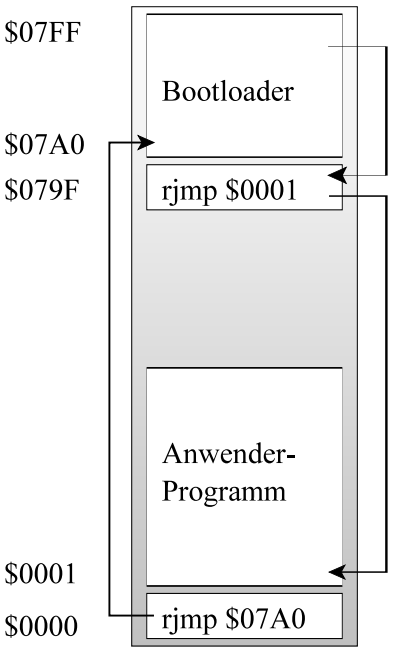

Abbildung 2

```
; 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 Ta1 beim Einschalten
; gedrückt bzw. das DTR-Signal durch das
; Uploader-Programm auf 0 gesetzt ist
; ansonsten wird Kalibrierprogramm gestartet

; greift auf Ideen von Burkhard Kainka
; (Lernpaket Mikrocontroller) zurück

; Version 1.2.0
; 19.02.21

.include "tn4313def.inc"

.def   param           = r16
.def   uartparam       = r18
.def   command         = r19
.def   temp            = r20
.def   RWCount         = r23
.def   spmcsrval       = r22
.def   epromadresse    = r21
.equ   DTR             = 3
.equ   LED             = 6
.equ   TAKT            = 4000000 ; HZ
.equ   PageSize4313    = 32

; r26 bis r31 reserviert für Pointer


; Verzweigung zum Bootloader
.org $0000
       rjmp  $07A0


; Kalibrierprogramm
.org $0001

; hier folgt das Kalibrierprogramm...


; letzte Adresse vor Bootloader
.org $079F
        rjmp $0001                  ; wird vom Uploader überschrieben


; Es folgt der eigentliche Bootloader...

.org $07A0
        sbi  portd, DTR             ; für Taster Ta1 bzw. DTR-Signal vorbereiten

; zuerst OSCCAL aus Eprom lesen...
        ldi   epromadresse, 255
        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               ; DTR = 0 bzw. Taster Ta1 geschlossen -> BOOTLOADER,
                                     ; sonst Anwender-Programm
       rjmp  Start
       rjmp  $079F                   ; Sprung zum Anwender-Programm

Start:
       sbi   ddrd, LED               ; für Anzeige LED
       sbi   portD, LED              ; Anzeige: LED an D.5 an, wenn Bootloading
       sbis  pinD, DTR               ; warten bis DTR = 1 bzw. Ta1 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, 4000000/(9600*16)-1  ;Baudrate 9600 einstellen
       out   UBRRL, uartparam
       rcall rdcom                   ; warten auf Startbyte
       cbi   portd, LED              ; LED an D.5 aus, wenn Startbyte erhalten
       ldi   param, 105              ; ACK senden
       rcall wrcom

MainSchlaufe:
       rcall rdcom
       mov   command, param

B201:                                ; Block (Page) schreiben
       cpi   command, 201
       brne  B203
       rcall Schreib_Block
       ldi   param, 1                ; ACK Block_Schreiben
       rcall wrcom

B203:                                ; Anwender-Programm starten
       cpi   command, 203
       brne  zurueck                 ; Wenn keine Zahl zutrifft zum Anfang
       ldi   param, 11               ; ACK Ende des Uploads
       rcall wrcom
       rcall warte1ms                ; 1 ms warten
       cbi   UCSRB, 4                ; RX deaktivieren
       cbi   UCSRB, 3                ; TX deaktivieren
       rjmp  $03AF

zurueck:
       rjmp  MainSchlaufe


Schreib_Block:
       ldi   RWCount, PageSize4313
       rcall rdcom                   ; Block-Adresse vom Master
       mov   ZH, param
       rcall rdcom
       mov   ZL, param
Loeschen:                            ; Block löschen
       ldi   spmcsrval, 3
       out   spmcsr, spmcsrval
       spm
Puffer_Schreib_Schleife:             ; alle Wörter des Blocks einzeln in den Puffer
                                     ; r1:r0 übertragen
       rcall rdcom
       mov   r0, param
       rcall rdcom
       mov   r1, param
       ldi   spmcsrval, 1            ; Puffer schreiben
       out   spmcsr, spmcsrval
       spm

       adiw  ZL, 2                   ; Flash-Adresse um 1 Word erhöhen
       mov   param, r0               ; ACK für die Übernahme eines Wortes
       rcall wrcom
       mov   param, r1
       rcall wrcom
```

```
        dec   RWCount
        brne  Puffer_Schreib_Schleife ; Ende der Puffer_Schreib_Schleife

        subi  ZL, PageSize4313        ; Startwert für Page in Z-Register restaurieren, um...
        subi  ZL, PageSize4313
        ldi   spmcsrval, 5            ; ... gesamten Puffer in FLASH schreiben
        out   spmcsr, spmcsrval
        spm
        ret

rdcom:
        sbis    UCSRA, 7              ; USR=UCSRA=0x0B RXC=Bit7
        rjmp    rdcom                 ; 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

warte1ms:
; Vorbereitung Zähler für Warteschleife
        ldi     r24, low(TAKT/4000)
        ldi     r25, high(TAKT/4000)
warteschleife:
        sbiw    r24, 1
        brne    warteschleife
        ret
```