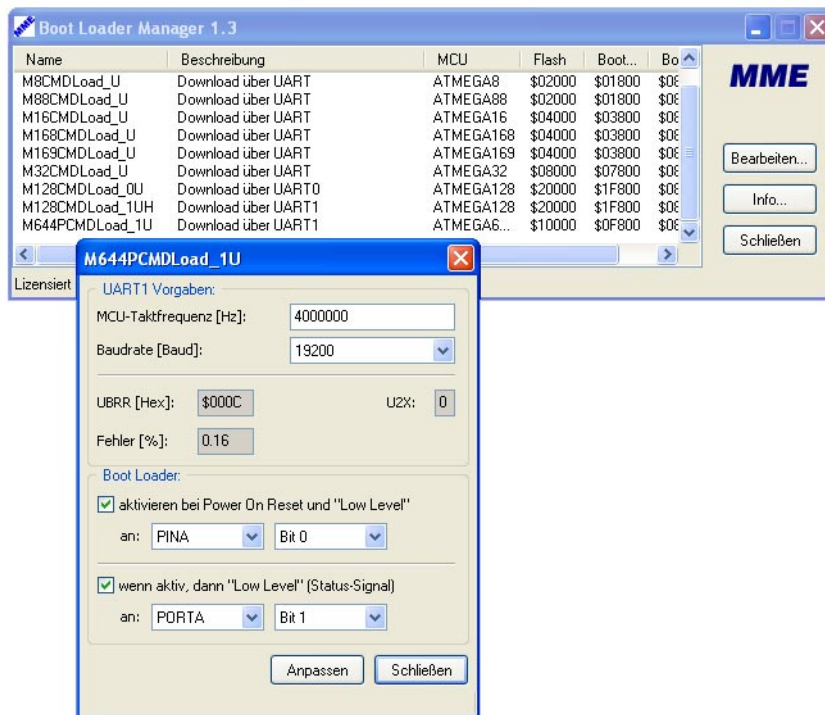


Boot Loader für AVR ATmega Controller

Copyright © 2003 - 2011 MME Berlin
Alle Rechte vorbehalten
Dokumentation: blm, Revision 1.23



Einschränkung der Gewährleistung. Es wird keine Garantie für die Richtigkeit des Inhaltes dieses Datenblattes übernommen. Da sich Fehler, trotz aller Bemühungen, nicht immer vollständig vermeiden lassen, sind wir für Hinweise jederzeit dankbar.

Die im Datenblatt verwendeten Soft- und Hardwarebezeichnungen und Markennamen der jeweiligen Firmen unterliegen im allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz.

MME Müller Mikroelektronik
Hauptstraße 8, Gewerbehau II, 10827 Berlin (Schöneberg)
Tel.: +49-30-787.09.450, FAX: +49-30-787.09.451
E-Mail: info@mme-berlin.de, Internet: http://www.mme-berlin.de

Einführung/Leistungsmerkmale

Ein Leistungsmerkmal der AVR ATmega Controller ist das sogenannte „self programming“. Ein ATmega Controller ist in der Lage, seinen eigenen Programmcode zu überschreiben. Hiermit können sehr komfortabel Firmware-Updates (im System), ohne spezielle Programmier-Hardware (ISP, JTAG), vorgenommen werden.

Mit Hilfe des *MME* Boot Loaders ist es möglich, ROM- oder HEX-Intel-Dateien über die serielle Schnittstelle in den AVR Controller zu laden. Der Boot Loader ist in der Lage, Anwender-Programme über den UART seriell zu empfangen und diese in den internen FLASH-Speicher zu programmieren. Auf der PC-Seite wird das Download-Programm *CMDLoad32* zur Verfügung gestellt.

Leistungsmerkmale des *MME* Boot Loaders:

- Hohe Downloadgeschwindigkeit: Bis zu 7,3 KB/Sekunde bei 115200 Baud und 11,0592 MHz.
- Boot Loader belegt lediglich 2 KB im AVR.
- Kurzer Execution-Overhead (Einsprung Boot Loader bis zur Programmausführung): Ca. 10 µs bei 11,0592 MHz.
- Keine Compilierung des Boot Loaders erforderlich. Der Boot Loader ist völlig unabhängig vom Anwenderprogramm.
- Definiertes Verhalten bei Abbruch eines Downloads.
- Programm- und hardwaregesteuerte Aktivierung.
- Konfigurierbares Status-Signal.
- Einfache Konfigurierbarkeit über Windows-Software *Boot Loader Manager*.
- Windows-Software *CMDLoad32* zum Download des Anwenderprogrammes.
- Firmware-Update über Internet.
- Windows-DLL *DLLLoad* zur Entwicklung von eigenen Download-Programmen.
- Download über Pocket PC in Vorbereitung.
- Zur Zeit unterstützte Controller: ATmega128, ATmega32, ATmega16, ATmega8, ATmega88, ATmega168, ATmega169, ATmega8535, ATmega644, ATmega644P und ATmega1284P.

Der Boot Loader zeichnet sich durch eine hohe Datenübertragungsgeschwindigkeit aus. So werden z. B. bei einer Baudrate von 115,2 kBaud und einem 11,0592 MHz AVR ca. 7,3 KB/Sekunde erreicht. Ein Download von 30 Kilobyte dauert also lediglich ca. 4 Sekunden. Der Execution-Overhead (Einsprung in den Boot Loader bis zur Ausführung des Anwenderprogrammes) liegt bei ca. 10 µs (@ 11,0592 MHz AVR).

Die Konfiguration des Boot Loaders erfolgt mit der Windows-Software *Boot Loader Manager*, kurz *BLM*. Je nach AVR-Taktfrequenz lassen sich folgende Baudraten einstellen: 300 Baud, 600 Baud, 900 Baud, 1200 Baud, 2400 Baud, 4800 Baud, 9600 Baud, 19200 Baud, 38400 Baud, 57600 Baud, 115200 Baud, 128000 Baud und 230400 Baud.

Weiterhin kann konfiguriert werden, ob der Boot Loader nach einem Reset über eine Port-Leitung des AVR aktiviert werden soll. Port und Bit können eingestellt werden. Der softwaremäßige Einsprung in den Boot Loader, aus dem Anwenderprogramm heraus, ist ebenfalls möglich.

Die Aktivität des Boot Loaders kann über eine konfigurierbare Port-Leitung des AVR signalisiert werden.

Für die Entwicklung eigener Download-Programme steht die Windows-DLL *DLLLoad* zur Verfügung. Der Download wird vollständig von der DLL abgewickelt. Der Anwender legt lediglich die Konfigurationsparameter fest und ruft dann die Funktion zum Download auf.

Funktionsweise

Nach einem Reset wird der Boot Loader „angesprungen“. Dieser untersucht jetzt, ob ein externes Ereignis vorliegt (falls konfiguriert), um in den Download-Modus zu schalten. Ist dies nicht der Fall, wird anhand des Reset-Vektors überprüft, ob ein Anwenderprogramm vorhanden ist. Ist dies der Fall, wird das Anwenderprogramm „angesprungen“ und kommt zur Ausführung. Andernfalls wird ein Download über die serielle Schnittstelle erwartet (Download-Modus).

Ein AVR Anwenderprogramm kann auch „programmgesteuert“ den Boot Loader aufrufen, um so einen Download einzuleiten. Die Einsprungadresse des Boot Loaders liegt immer 2 KB vor dem Ende des FLASH-Speichers. Die Einsprungadresse variiert also bei unterschiedlichen AVR-Typen. Nachfolgend einige Beispiele, wie dies aus einer Hochsprache heraus realisiert werden kann:

Boot Loader-Aufruf mit Embedded Pascal (<http://users.iafrica.com/r/ra/rainier/>):

```
procedure EnterBootLoaderFromApplication;
{
  Programmgesteuerter Aufruf des Boot Loaders bei einem
  ATmega128 oder ATmega1284P-Controller (128 KB Flash).
}
begin
  DI;           { Interrupts sperren }
  _MCUCSR:= 0; { Boot Loader soll Download-Modus aktivieren }
  asm
    jmp $1F800 { Sprung in den Boot Loader }
  end;
  { Hier kommen wir schon nicht mehr hin }
end;

procedure EnterBootLoaderFromApplication;
{
  Programmgesteuerter Aufruf des Boot Loaders bei einem
  ATmega16-, ATmega168- oder ATmega169-Controller (16 KB Flash).
}
begin
  DI;           { Interrupts sperren }
  _MCUSR:= 0;  { Boot Loader soll Download-Modus aktivieren }
  asm
    jmp $3800  { Sprung in den Boot Loader }
  end;
  { Hier kommen wir schon nicht mehr hin }
end;

procedure EnterBootLoaderFromApplication;
{
  Programmgesteuerter Aufruf des Boot Loaders bei einem
  ATmega644 oder ATmega644P-Controller (64 KB Flash).
}
begin
  DI;           { Interrupts sperren }
  _MCUSR:= 0;  { Boot Loader soll Download-Modus aktivieren }
  asm
    jmp $F800  { Sprung in den Boot Loader }
  end;
  { Hier kommen wir schon nicht mehr hin }
end;
```

Funktionsweise

Boot Loader-Aufruf mit Codevision-C (<http://www.hpinfotech.ro>):

```
void EnterBootLoaderFromApplication(void)
/*
  Programmgesteuerter Aufruf des Boot Loaders bei einem
  ATmega128 oder ATmega1284P-Controller (128 KB Flash).
*/
{
  #asm("cli")      /* Interrupts sperren                */
  MCUCSR= 0;      /* Boot Loader soll Download-Modus aktivieren        */
  #asm
    jmp $0FC00    /* Sprung in den Boot Loader                        */
  #endasm;
  /* Hier kommen wir schon nicht mehr hin */
}

void EnterBootLoaderFromApplication(void)
/*
  Programmgesteuerter Aufruf des Boot Loaders bei einem
  ATmega16-, ATmega168- oder ATmega169-Controller (16 KB Flash).
*/
{
  #asm("cli")      /* Interrupts sperren                */
  MCUSR= 0;       /* Boot Loader soll Download-Modus aktivieren        */
  #asm
    jmp $01C00    /* Sprung in den Boot Loader                        */
  #endasm;
  /* Hier kommen wir schon nicht mehr hin */
}

void EnterBootLoaderFromApplication(void)
/*
  Programmgesteuerter Aufruf des Boot Loaders bei einem
  ATmega644 oder ATmega644P-Controller (64 KB Flash).
*/
{
  #asm("cli")      /* Interrupts sperren                */
  MCUSR= 0;       /* Boot Loader soll Download-Modus aktivieren        */
  #asm
    jmp $07C00    /* Sprung in den Boot Loader                        */
  #endasm;
  /* Hier kommen wir schon nicht mehr hin */
}
```

Boot Loader-Aufruf mit AVR-GCC:

```
void EnterBootLoaderFromApplication(void)
/*
  Programmgesteuerter Aufruf des Boot Loaders bei einem
  ATmega128 oder ATmega1284P-Controller (128 KB Flash).
*/
{
  cli();          /* Interrupts sperren                */
  MCUCSR= 0;     /* Boot Loader soll Download-Modus aktivieren        */
  asm volatile("jmp 0xFC00"); /* Sprung in den Boot Loader        */
  /* Hier kommen wir schon nicht mehr hin */
}

void EnterBootLoaderFromApplication(void)
/*
  Programmgesteuerter Aufruf des Boot Loaders bei einem
  ATmega16-, ATmega168- oder ATmega169-Controller (16 KB Flash).
*/
{
  cli();          /* Interrupts sperren                */
  MCUSR= 0;      /* Boot Loader soll Download-Modus aktivieren        */
  asm volatile("jmp 0x1C00"); /* Sprung in den Boot Loader        */
  /* Hier kommen wir schon nicht mehr hin */
}
```

Funktionsweise

```
void EnterBootLoaderFromApplication(void)
/*
  Programmgesteuerter Aufruf des Boot Loaders bei einem
  ATmega644 oder ATmega644P-Controller (64 KB Flash).
*/
{
  cli();          /* Interrupts sperren */
  MCUSR= 0;      /* Boot Loader soll Download-Modus aktivieren */
  asm volatile("jmp 0x7C00"); /* Sprung in den Boot Loader */
  /* Hier kommen wir schon nicht mehr hin */
}
```

In dem MCUCSR/MCUSR-Register wird u. A. die Ursache für einen Reset festgehalten. Wird dieses Register auf 0 gesetzt und anschließend in den Boot Loader gesprungen, so schaltet sich dieser in den Download-Modus. Bei einem „echten“ Reset würde das MCUCSR/MCUSR-Register einen Wert ungleich von Null enthalten. Der Boot Loader würde dann das Anwenderprogramm starten.

Boot Loader Manager

Der *Boot Loader Manager (BLM)* ist ein 32-Bit Windows-Programm, welches unter den Windows Versionen 9X/ME/NT/2000/XP/Vista/Windows 7 ablauffähig ist. Mit dem *BLM* werden AVR Boot Loader konfiguriert und als ROM- und HEX-Intel-Datei bereitgestellt.

Installation

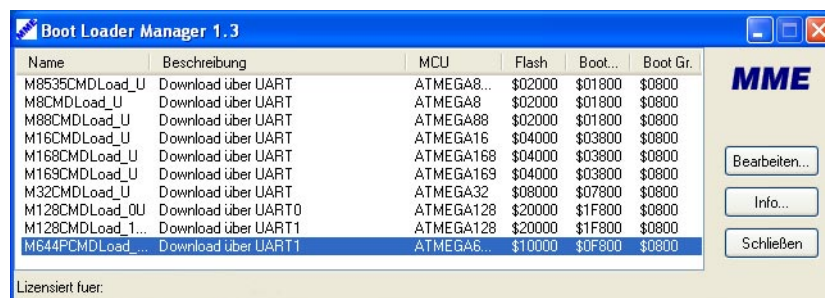
Das Programm kann in ein beliebiges Verzeichnis auf der Festplatte kopiert werden und von dort gestartet werden. Zu dem *BLM* gehört mindestens ein AVR Boot Loader. Dieser muß sich im gleichen Verzeichnis befinden wie *BLM.EXE* und hat die Dateierweiterung *.ROM*.

Mitgelieferte Dateien

BLM.EXE	- Boot Loader Manager
CMDLoad32.EXE	- Download-Programm
MxxCMDLoad_U.ROM	- AVR Boot Loader für den jeweiligen ATmega-Controller
BLM.TXT	- Textdatei mit Erläuterungen
DEMO_PC\DLLLoad.dll	- DLL zur Einbindung in eigene Download-Programme (optional)
DEMO_PC\DemoDLLLoad1.DPR	- Delphi Projekt für erstes Demo
DEMO_PC\DemoDLLLoad1.EXE	- Erstes Demo als ausführbares Programm
DEMO_PC\Demo1.PAS	- Delphi Source für erstes Demo
DEMO_PC\Demo1.DFM	- Delphi Formular-Datei für erstes Demo
DEMO_PC\DemoDLLLoad2.DPR	- Delphi Projekt für zweites Demo
DEMO_PC\DemoDLLLoad2.EXE	- Zweites Demo als ausführbares Programm
DEMO_PC\Demo2.PAS	- Delphi Source für zweites Demo
DEMO_PC\Demo2.DFM	- Delphi Formular-Datei für zweites Demo

Hinweis: Die Download-DLL *DLLLoad* gehört nicht zum Standardlieferungsumfang und muß gesondert geordert werden.

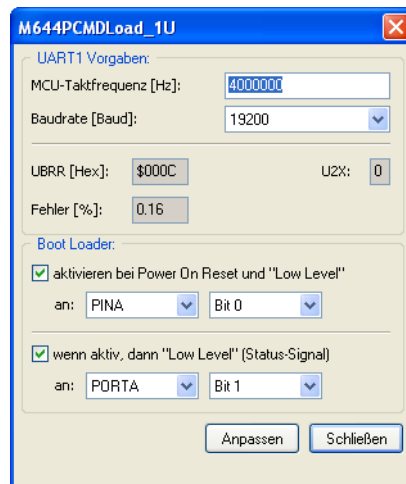
Programmablauf



Hauptfenster des *BLM*.

Nach dem Start von *BLM.EXE* werden alle vorhandenen AVR Boot Loader aufgelistet. Durch Doppelclick auf einen Eintrag (oder betätigen des Buttons „Bearbeiten“) wird ein Dialog-Fenster geöffnet. In diesem Dialog können Anpassungen an den Boot Loader vorgenommen werden. Nachdem die Anpassungen vorgenommen worden sind, kann der modifizierte Boot Loader auf die Festplatte geschrieben werden. Hierzu ist der Button „Anpassen“ zu betätigen. Der Boot Loader wird als binäres Image (ROM-Datei) sowie als HEX-Intel-Datei erzeugt. Der Boot Loader wird dann mit einem AVR Programmiergerät (z. B. ATMEL STK500) in einen AVR Controller geflasht.

Anpassungen des Boot Loaders



Dialog zur Anpassung des Boot Loaders.

Die Download-Baudrate kann konfiguriert werden. Folgende Baudraten sind einstellbar: 300 Baud, 600 Baud, 900 Baud, 1200 Baud, 2400 Baud, 4800 Baud, 9600 Baud, 19200 Baud, 38400 Baud, 57600 Baud, 115200 Baud, 128000 Baud sowie 230400 Baud.

Nach Vorgabe der AVR Taktfrequenz werden die AVR Register „UBRR“ sowie „U2X“ errechnet. Zusätzlich wird noch der „Baudratenfehler“ angegeben. Bei einem Fehler von größer 1 % wird eine Warnung ausgegeben. Eine serielle Kommunikation ist dann eventuell nicht fehlerfrei möglich. Hinweis: 230400 Baud lassen sich nicht bei allen PCs einstellen.

Weiterhin kann konfiguriert werden, ob der Bootloader nach einem Reset eine bestimmte Port-Leitung überwachen soll, um dann gegebenenfalls in den Download-Modus zu schalten. Wenn dies gewünscht ist, dann sind das Port sowie das entsprechende Port-Bit festzulegen. Wird die Port-Leitung vom JTAG-Interface belegt, dann muß dieses abgeschaltet werden. Hinweis: Ein externer Pullup-Widerstand von ca. 10 Kiloohm ist erforderlich, damit nach dem Einschalten ein sicherer High-Pegel erkannt wird.

Die Aktivität des Boot Loaders kann über eine zu konfigurierende Port-Leitung signalisiert werden. Wird diese Leitung vom JTAG-Interface belegt, dann muß dieses abgeschaltet werden.

Programmieren des Boot Loaders in den AVR

Der Boot Loader wird mit einer geeigneten Programmier Hard- und Software in den AVR geladen. Z. B. mit dem ATMEL STK500 Board. Der Boot Loader liegt als binäres Image vor (z. B. M128CMDLoad_0U.rom) sowie in Form von einer HEX-Intel-Datei (z. B. M128CMDLoad-0U.hex). Nachfolgende Anweisungen **müssen unbedingt** berücksichtigt werden, da sonst der Boot Loader im AVR nicht (zuverlässig) arbeitet.

Ladeadresse

Bei der Verwendung der HEX-Intel-Datei wird die korrekte Ladeadresse automatisch berücksichtigt. Der gesamte AVR (128 Kilobyte) wird geflasht. Wenn die ROM-Datei verwendet wird, dann ist die Position des Boot Loaders im AVR zu berücksichtigen. Die Startadresse liegt immer 2048 Byte vor dem Ende des FLASH-Speichers. Sie beträgt z. B. beim ATmega128 \$1F800 und beim ATmega168 \$3800.

AVR Fuses beim ATmega128 und ATmega1284P

BOOTSZ1 = 1 und BOOTSZ0 = 0 (Boot Size 2048 Byte = 1024 Word)

BOOTRST = 0: Ein Reset führt direkt zur Ausführung des Boot Loaders.

Anpassungen des Boot Loaders

AVR Fuses beim ATmega644 und ATmega644P

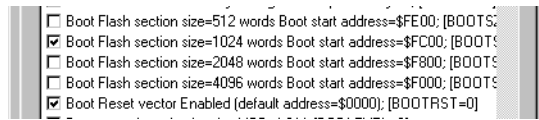
BOOTSZ1 = 1 und BOOTSZ0 = 0 (Boot Size 2048 Byte = 1024 Word)

BOOTRST = 0: Ein Reset führt direkt zur Ausführung des Boot Loaders.

AVR Fuses beim ATmega16, ATmega168 und ATmega169

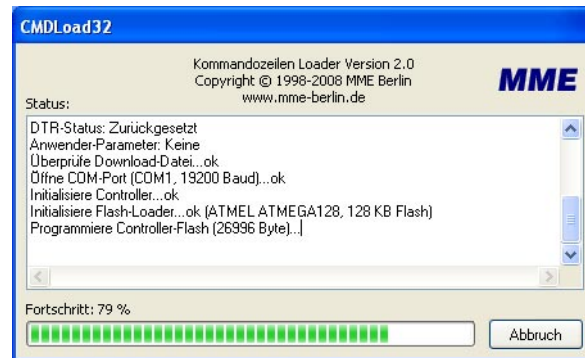
BOOTSZ1 = 0 und BOOTSZ0 = 0 (Boot Size 2048 Byte = 1024 Word)

BOOTRST = 0: Ein Reset führt direkt zur Ausführung des Boot Loaders.



Einstellungen der Fuse-Bits für den ATmega128 in AVRStudio/STK500

Download-Programm CMDLoad32



CMDLoad32 während des Downloads.

Mit dem Download-Programm *CMDLoad32* werden AVR-Anwendungen vom PC über die serielle Schnittstelle in den AVR geladen. Das Programm kann sowohl binäre Images (ROM-Dateien) als auch HEX-Intel-Dateien verarbeiten. *CMDLoad32* wird über die Kommandozeile gestartet. Der Aufruf lautet:

```
CMDLoad32 <romfile.rom|hexfile.hex> [/B<baudrate>] [/COM<n>] [Anwenderparameter]
```

Beispiel:

```
CMDLoad32 test.hex /B19200 /COM2
```

würde die HEX-Datei *test.hex* mit einer Baudrate von 19200 Baud über die serielle Schnittstelle COM2 übertragen.

```
CMDLoad32 test.rom /B38400 /COM3
```

würde die ROM-Datei *test.rom* mit einer Baudrate von 38400 Baud über die serielle Schnittstelle COM3 übertragen.

Ab der Version 1.9 von *CMDLoad32* können auch ROM-Images über das Internet geladen werden. Hierzu ist es erforderlich, daß der PC mit dem Internet verbunden ist. Es wird das HTTP-Protokoll verwendet. Gegebenenfalls ist eine vorhandene Firewall zu konfigurieren. Soll z. B. das ROM-Image *myimage.rom* von dem Webserver *www.myserver.de* geladen werden, so lautet der Aufruf:

```
CMDLoad32 http://www.myserver.de/myimage.rom
```

Hinweis: Zur Zeit können noch keine HEX-Intel-Dateien übertragen werden.

Anwenderparameter

Mit den sogenannten Anwenderparametern ist es möglich, vor dem eigentlichen Download eine bestimmte Zeichenkette zu senden, so daß eine AVR-Anwendung hierauf einen Download einleiten kann.

```
/AZ<text>
```

text ist die Zeichenkette, die vom Download-Programm gesendet wird, um den AVR in den Download-Modus zu schalten. Steuerzeichen werden mit # und dem dreistelligen ASCII-Code gekennzeichnet. Wenn z. B. die Zeichenkette FEUER! mit anschließendem Carriage Return und Line Feed gesendet werden soll, dann lautet der Parameter:

Download-Programm CMDLoad32

/AZFEUER!#013#010

/AB<baudrate>

Hiermit wird die Baudrate angegeben, mit der die Anwenderzeichenkette gesendet wird. Um also z. B. die Anwenderzeichenkette mit 300 Baud zu senden, würde der Parameter lauten:

/AB300

/AD<##>

Hiermit wird eine Verzögerungszeit in Sekunden angegeben, die das Download-Programm nach dem Senden der Anwenderzeichenkette wartet, bis der eigentliche Download gestartet wird.

Download-DLL DLLLoad.dll

Mit dieser DLL können eigene Windows Download-Programme geschrieben werden. Die beigefügten Delphi-Demos zeigen, wie die DLL in eigene Programme eingebunden wird. Ab der Version 1.3 der DLL können auch ROM-Images aus dem Internet geladen werden. Siehe dazu auch die Hinweise auf Seite 9.

Die DLL exportiert folgende Funktionen:

```
function LOAD_GetVersion(var pcVersion: PChar): Word; stdcall;
function LOAD_GetLicense(var pcLicense: PChar): Word; stdcall;
function LOAD_GetVersionAsNumber: Word; stdcall;
function LOAD_IntoController(pcLoadParameter: PChar): Word; stdcall;
function LOAD_IntoControllerExtended(pcLoadParameter: PChar;
                                     DLLLoadCallback: TDLLLoadCallback): Word; stdcall;
function LOAD_Cancel: Word; stdcall;
```

Beschreibung der exportierten DLL-Funktionen:

```
function LOAD_GetVersion(var pcVersion: PChar): Word; stdcall;
```

Diese Funktion liefert die Versionskennung der DLL als nullterminierten String (PChar) zurück. Also z. B.: „DLLLoad Version 1.6 Copyright (C) 2006-2011 MME Berlin“.

```
function LOAD_GetLicense(var pcLicense: PChar): Word; stdcall;
```

Diese Funktion liefert den Lizenznehmer der DLL als nullterminierten String zurück. Also z. B.: „Lizenziert für Megasoft GmbH, SN: 1234“.

```
function LOAD_GetVersionAsNumber: Word; stdcall;
```

Diese Funktion liefert die Versionsnummer der DLL als Word-Variable zurück. Also z. B. 100 für die Version 1.00, 110 für 1.10, 112 für 1.12 usw.

```
function LOAD_IntoController(pcLoadParameter: PChar): Word; stdcall;
```

Diese Funktion wickelt den vollständigen Download in den AVR ab. In `pcLoadParameter` werden die Parameter als nullterminierten String übergeben. Die Parameter sind kompatibel zum Download-Programm *CMDLoad32*. Also:

```
<romfile.rom|hexfile.hex> [/B<baudrate>] [/COM<n>] [Anwenderparameter]
```

Beispiele siehe Seite 9, Download-Programm *CMDLoad32*. Siehe auch beigefügte Demo-Programme. Der Rückgabewert der Funktion hat folgende Bedeutung:

- 0: Download erfolgreich.
- 1: Es sind keine Parameter angegeben worden.
- 2: Ungültiger Parameter.
- 3: Ungültiger Dateiname.
- 4: Ungültiger COM-Port.
- 5: Ungültige Baudrate.
- 10: COM-Port lässt sich nicht öffnen.
- 11: Datei nicht gefunden.

Download-DLL *DLLLoad.dll*

- 12: Datei läßt sich nicht öffnen.
- 13: Leseversuch fehlgeschlagen.
- 14: Datei läßt sich nicht aus dem Internet laden.
- 15: Nicht genügend freier Speicher vorhanden.
- 16: Controller meldet sich nicht. Download abgebrochen.
- 17: Controller sendet nicht. Download abgebrochen.
- 18: Controller Firmware korrupt. Download abgebrochen.
- 19: Zeitüberschreitung. Download abgebrochen.
- 20: Checksumme falsch. Download abgebrochen.
- 31: ROM-Datei zu groß. Die maximale Größe für den ATMEGA32 beträgt 30720 Byte.
- 32: ROM-Datei zu groß. Die maximale Größe für den ATMEGA128 beträgt 129024 Byte.
- 33: ROM-Datei zu groß. Die maximale Größe beträgt 129024 Byte.
- 34: ROM-Datei zu groß. Die maximale Größe für den ATMEGA8 beträgt 6144 Byte.
- 35: ROM-Datei zu groß. Die maximale Größe für den ATMEGA8535 beträgt 6144 Byte.
- 36: ROM-Datei zu groß. Die maximale Größe für den ATMEGA88 beträgt 6144 Byte.
- 37: ROM-Datei zu groß. Die maximale Größe für den ATMEGA168 beträgt 14336 Byte.
- 38: ROM-Datei zu groß. Die maximale Größe für den ATMEGA169 beträgt 14336 Byte.
- 39: ROM-Datei zu groß. Die maximale Größe für den ATMEGA644P beträgt 63488 Byte.
- 40: ROM-Datei zu groß. Die maximale Größe für den ATMEGA1284P beträgt 129024 Byte.
- 41: HEX-INTEL Datei enthält ungültige Zeichen.
- 42: HEX-INTEL Datei enthält nicht unterstütztes Feld: RECTYPE.
- 43: HEX-INTEL Datei kann nicht aus dem Internet geladen werden.
- 44: Aus dem Internet geladenes ROM-Image ist korrupt.

```
function LOAD_IntoControllerExtended(pcLoadParameter: PChar;  
                                     DLLLoadCallback: TDLLLoadCallback): Word; stdcall;
```

Diese Funktion wickelt den vollständigen Download in den AVR ab. In `pcLoadParameter` werden die Parameter als nullterminierten String übergeben. Die Parameter sind kompatibel zum Download-Programm *CMDLoad32*. Der Rückgabewert der Funktion hat die gleiche Bedeutung wie bei der Funktion `LOAD_IntoController`. Zusätzlich kann der Funktion noch eine sogenannte Callback-Funktion übergeben werden, welche von der DLL während des Downloads aufgerufen wird, um über den Zustand des Downloads zu informieren. Die Callback-Funktion hat folgendes Format:

```
TDLLLoadCallback = Function(wAction: Word; pcParameter: PChar): Word;
```

Mit der Word-Variablen `wAction` informiert die DLL den Aufrufer darüber, welche Funktion sie gerade ausführt. In dem nullterminierten String `pcParameter` werden zusätzliche Informationen übergeben. `wAction` kann folgende Werte annehmen:

- 0: Serielle Schnittstelle wird geöffnet.
- 1: Es wird überprüft, ob die zu ladende Datei vorhanden ist.
- 2: AVR wird initialisiert.
- 7: Fortschritt beim Download. Die Datenmenge wird in `pcParameter` mitgeteilt.
- 8: Ende des Downloads. Die Übertragungszeit (in Sekunden) und Datenrate (in Byte/s) wird in `pcParameter` angegeben. Die Angabe 00:01.882;7294 würde z. B. bedeuten, daß die Übertragung 1.882 Sekunden gedauert hat. Die Datenrate betrug dabei 7294 Byte pro Sekunde.
- 11: Download wird gestartet. In `pcParameter` wird die Dateilänge in Byte angegeben.
- 17: Datei wird aus dem Internet geladen.
- 18: Der Ladevorgang aus dem Internet war erfolgreich.

Download-DLL DLLLoad.dll

```
function LOAD_Cancel: Word; stdcall;
```

Mit dieser Funktion kann ein Download abgebrochen werden.