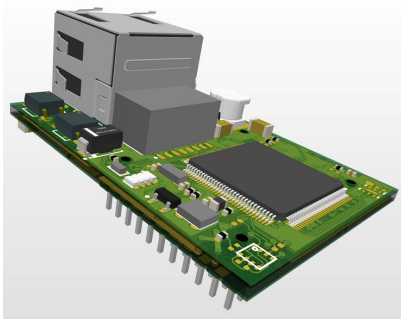
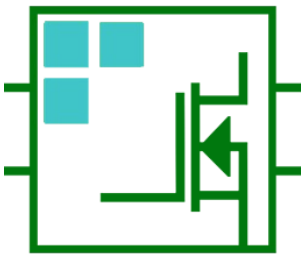
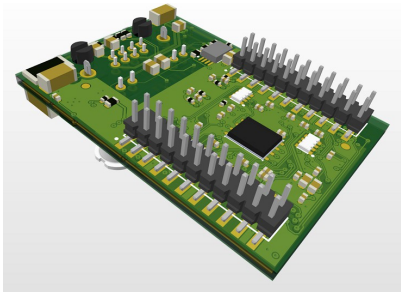
MKC Michels & Kleberhoff Computer GmbH
Vohwinkeler Str. 58, D-42329 Wuppertal
Tel.: ++49 (0)202 27317 0 Fax: ++49 (0)202 27317 49
Internet: http://www.mkc-gmbh.de







Dokumentation Software

MKC-Bootloader

Hinweise:

Die Informationen in diesem Handbuch wurden sorgfältig zusammengestellt und überprüft. Dieses Handbuch wird stetig auf dem aktuellen Zustand gehalten. Jedoch wird von MKC keine Gewähr für fehlerhafte Informationen übernommen.

MKC behält sich das Recht vor, jederzeit ohne weitere Ankündigung technische Änderungen zur Verbesserung der Zuverlässigkeit, der Funktion oder des Designs der Produkte und Überarbeitungen des Handbuchs durchzuführen. Änderungen des Handbuchs zwischen 2 Ausgaben werden im Text nicht markiert.

Das Datum einer Ausgabe bezieht sich auf das Handbuch. Dieses muss nicht mit dem Datum der Änderung der Hardware oder Software übereinstimmen. Bei der Versionsgeschichte wird der Grund für die Handbuch Änderungen genannt.

MKC übernimmt keine Haftung für die Anwendung des hier beschriebenen Produktes. MKC übernimmt weiterhin keine Haftung für Schäden oder Folgeschäden, die durch Verwendung dieses Produktes entstehen. Diese Haftungseinschränkung bezieht sich sowohl auf jeden direkten Abnehmer sowie auf alle seine Kunden und alle Anwender des Produktes.

Es gelten ausschließlich die in diesem Dokument gemachten Zusagen über die Anwendbarkeit des hier beschriebenen Produktes.

Kommentare:

Kommentare oder Korrekturen jedweder Art sind dem Autor jederzeit willkommen. Senden Sie diese bitte an:

MKC Michels & Kleberhoff Computer GmbH
Vohwinkeler Str. 58
42329 Wuppertal

oder

info@mkc-gmbh.de

Handbuch Versionen

Änderungen im Handbuch werden durch eine Erhöhung der Ausgabenummer angezeigt. Handbücher, deren Ausgabe durch einen Buchstaben gekennzeichnet ist, sind vorläufige Handbücher und stimmen möglicherweise noch nicht vollständig mit dem endgültigen Produkt überein. Die erste Ausgabe, die nicht mehr als vorläufig anzusehen ist, beginnt mit der Nummerierung „1“.

Handbuch Versionen			
Ausgabe	Änderungen	Datum	
1	Erste Version (MKC1601, MKC1503)	15.11.2018	

Inhaltsverzeichnis

Liste der Abbildungen

Liste der Tabellen

1 Einleitung

Die eNetMini Module (MKC1601) und die hiervon abgeleiteten IONet Geräte (MKC1503) basieren auf dem Tiva TM4C1294KCPDT Microcontroller des Herstellers Texas Instruments. Dieser beinhaltet einen Bootloader im ROM-Code, der beim Systemstart überprüft, ob im Flash ab der Adresse 0x0000 ein Anwendung abgespeichert ist.
Für den Fall, dass ab dieser Adresse ein ausführbares Programm steht, wird dieses gestartet. In dem Fall, dass diese Prüfung negativ ausfällt, versucht der Controller über seine Netzwerkschnittstelle eine Verbindung zu einem übergeordneten Rechner aufzubauen und von diesem alle notwendigen Netzwerkparameter zu erhalten, um anschließend ein Programm zu laden. Hierzu werden die Protokolle BOOTP und TFTP verwendet.

Um dieses prinzipielle Verfahren auch für Systemupdates zu nutzen, liefert MKC die Module/Systeme mit einen eigenen Bootloader aus. Dieser arbeitet sozusagen 'nachrangig' zu dem ROM-Bootloader und kann über global definierte Variablen einfach von der jeweiligen Firmware gesteuert werden. Somit sind Updates über mehrere Stufen bzw. einzelne Komponenten nacheinander möglich.

Der realisierte MKC-Bootloader basiert auf den von Texas Instruments zur Verfügung gestellten Programmquellen und ist von MKC an die jeweilige Hardware angepasst worden. Sämtliche relevanten Definitionen sind in der Quelldatei 'bl_config.h' gesetzt. Der erzeugte ausführbare Programmcode (im folgenden als 'MKC-Bootloader' bezeichnet) wird bei der Produktion der Platinen im Festwertspeicher gespeichert und ist somit stets in dem jeweiligen System vorhanden.
Weiterhin werden in diesem Bereich alle relevanten Produktionsdaten abgelegt. Der hierfür benötigte Speicherbereich (Adresse $00000000 – $00004000) wird schreibgeschützt ausgeliefert.

Der Sonderfall, dass das System gar keine gültige programmierte Firmware enthält, führt der MKC-Bootloader mit einem automatisch gestarteten Systemupdate zu einem laufenden System. Falls die Firmware nicht korrekt arbeitet und somit ein definiertes Anstoßen des Updates nicht möglich ist, kann ein automatisch gestartetes Systemupdate eingeleitet werden indem beim Power-On das externe Signal /FSU_IN aktiviert wird (Forced System Update).

1.1 Hinweise zu Angaben in diesem Handbuch

Zahlenangaben

Hexadezimale Zahlen werden in diesem Handbuch durch ein vorangestelltes Dollarzeichen „$“ gekennzeichnet. Andere geläufige Schreibweisen für Hexadezimale Zahlen sind z.B. durch den Präfix „0x“ oder den Suffix „h“ in der Literatur angegeben. Sie werden hier nur der Vollständigkeit halber erwähnt.

Um die Lesbarkeit von langen hexadezimalen Zahlen zu verbessern, werden diese von rechts durch einen Punkt in 4er Gruppen unterteilt. Eine mathematische Bedeutung liegt diesem Punkt nicht zugrunde.

Vorläufige Angaben

In dieser Handbuchversion sind mehrere Kapitel noch vorläufig, diese Stellen sind mit dem Textzusatz '***TDB: …***' an den entsprechenden Stellen gekennzeichnet.

2 Systemstart

Nach dem Systemstart, (Reset oder PowerOn) prüft der ROM-Code (ROM-Bootloader) der zunächst, ob an der Startadresse des Flashbereiches (Adresse $00000000) ein gültiges ausführbares Programm gespeichert ist. Das Verfahren zeigt Abbildung 1: Funktion nach Reset/PowerOn.

Im Normalfall ist an dieser Adresse der MKC-Bootloader gespeichert. Dieser wird von dem ROM-Code angesprungen, die Arbeitsweise wird in Kapitel 2.2 erläutert.

Für den Sonderfall, dass das System keinerlei Programmcode in dem Flashspeicher abgelegt hat, wird der weitere ROM-Code der CPU ausgeführt. Die Arbeitsweise dieses Programms ist fest kodiert und nicht veränderbar, der Ablauf ist in dem Kapiteln 2.1 dargestellt.

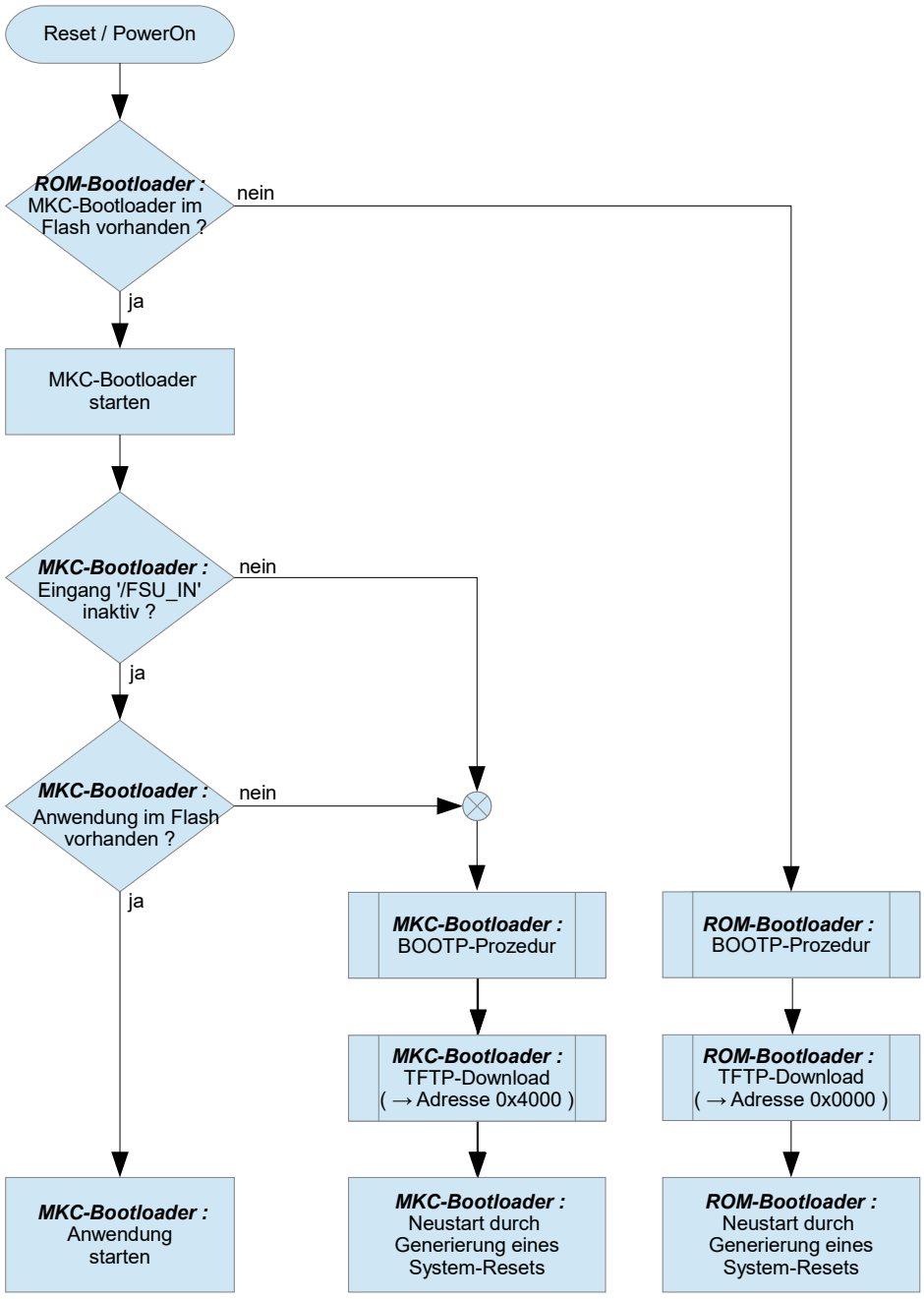


Abbildung 1: Funktion nach Reset/PowerOn

2.1 Arbeitsweise des ROM-Bootloaders

Der in der CPU-Hardware als ROM-Code fest programmierte ROM-Bootloader tritt beim Systemstart immer dann in Aktion, wenn in dem Flashbereich (Adresse $00000000) kein gültiger Programm-Code gespeichert ist. Dieser Zustand ergibt sich nach einem (irrtümlichen) vollständigen Löschen des (normalerweise) schreibgeschützten Flashbereiches mittels Programmieradapter und der entsprechenden Software.

Der ROM-Bootloader initialisiert das Netzwerk und sendet periodisch Anfragen an einen im Netzwerk erreichbaren BOOTP-Server. Diese Anfragen bauen auf das 'standard bootstrap protocol (BOOTP)' auf und werden als 'broadcast' innerhalb des Netzwerkes gesendet.

Der BOOTP-Server sendet - aufgrund der empfangenen Anfrage des Moduls/Systems - diesem das Antwortpaket mit der zugewiesenen neuen IP-Adresse für das Modul, die IP-Adresse des lokalen TFTP-Servers und einen Dateinamen.
Der ROM-Bootloader empfängt und wertet diese Antwort aus und sendet anschließend ein TFTP-Request für die Datei (mit dem empfangenen Dateinamen) an den TFTP-Server (mit der empfangenen IP-Adresse). Die angeforderte Datei kann jetzt von dem TFTP-Server gesendet werden. Der ROM-Bootloader schreibt die empfangenen Daten ab der Adresse $00000000 in den Flashspeicher der CPU. Ist der Transfer erfolgreich beendet worden, erfolgt ein Systemneustart.

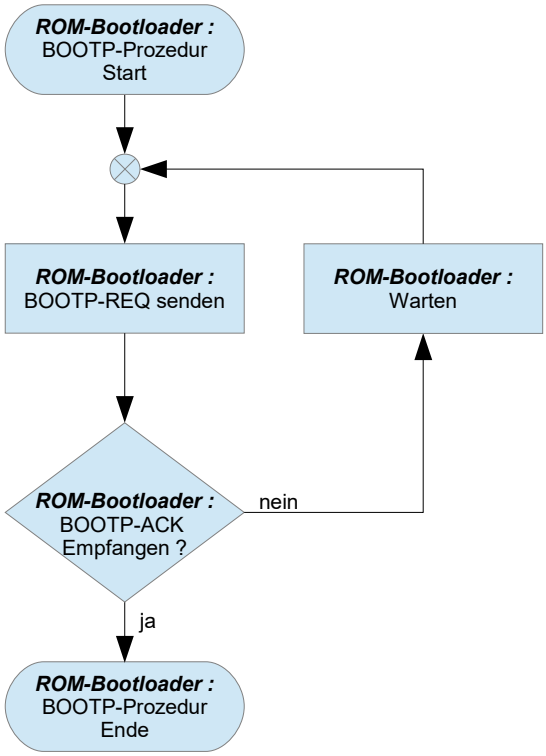


Abbildung 2: ROM-Bootloaders, BOOTP-Prozedur

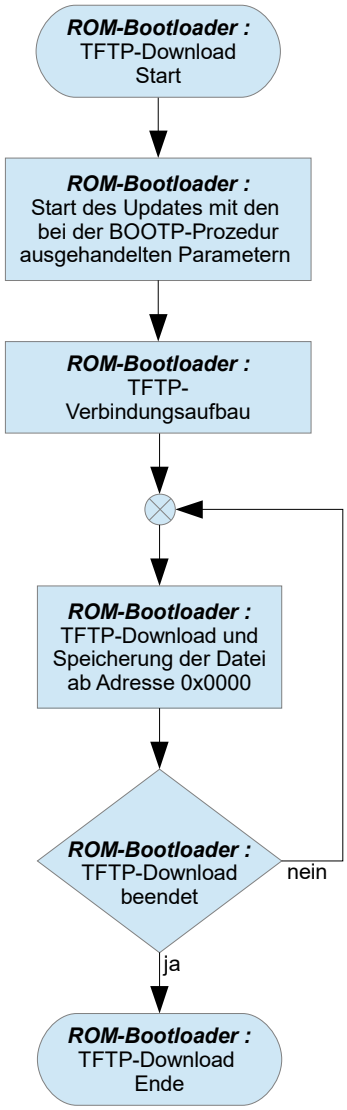


Abbildung 3: ROM-Bootloaders, TFTP-Download

2.2 Arbeitsweise des MKC-Bootloaders

Um eine vom Anwender konfigurierbare Steuerung des Bootloaders zu ermöglichen, werden alle notwendigen Parameter für den Netzwerk-Transfer in einem speziellen Speicherbereich des internen RAMs abgelegt (am obersten Ende des physikalischen Speichers, dieser Bereich ist reserviert für die globale Variable `gbl_vars`). Zunächst wird mittels des BOOTP-Protokolls eine gültige Netzwerkkonfiguration von einem BOOTP-Server im Netzwerk angefordert. Aus dem empfangenen Antwortpaket extrahiert der MKC-Bootloader die zugewiesene IP, die mitgelieferte Gateway-IP und die Adresse des TFTP-Servers. Diese Werte werden in der globalen Variable `gbl_vars` gespeichert und können somit von der Firmware genutzt werden.

Um sicher zu stellen, dass die gesamten notwendigen Programmdaten über das Netzwerk transferiert wurden und erfolgreich im Flash gespeichert worden sind, werden die ersten 4 Byte des 'neuen' Programms erst am Ende des gesamten Update-Prozesses in das Flash an die Programmstartadresse geschrieben. Erst hiermit wird für den MKC-Bootloader das neue Programm gültig. Tritt ein Fehler während des laufenden Updates auf, startet der MKC-Bootloader beim Systemstart das bisherige (noch nicht überschriebene und immer noch gültige) Programm. Für den Fall, dass das Anwenderprogramm teilweise gelöscht oder durch das neue Programm teilweise überschrieben wurde, wird der Update-Prozess mit den gültigen Werten der Variablen `gbl_vars` bei einem Systemneustart wiederholt. Sind auch diese Werte ungültig, so erfolgt das Update in Form des Forced System Update.

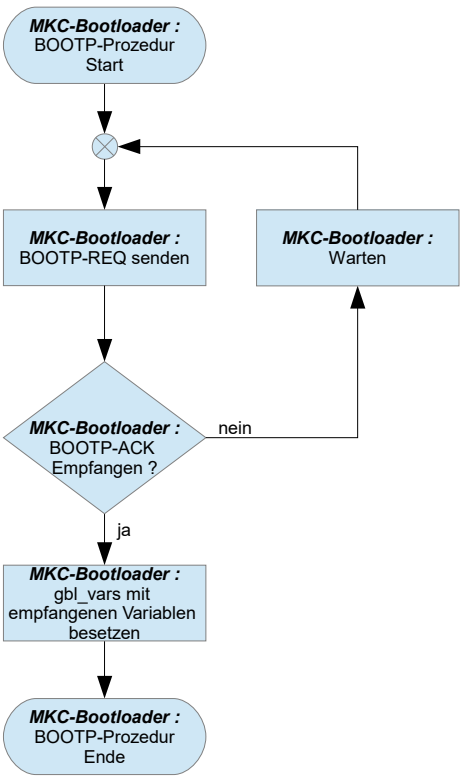


Abbildung 4: MKC-Bootloaders, BOOTP-Prozedur

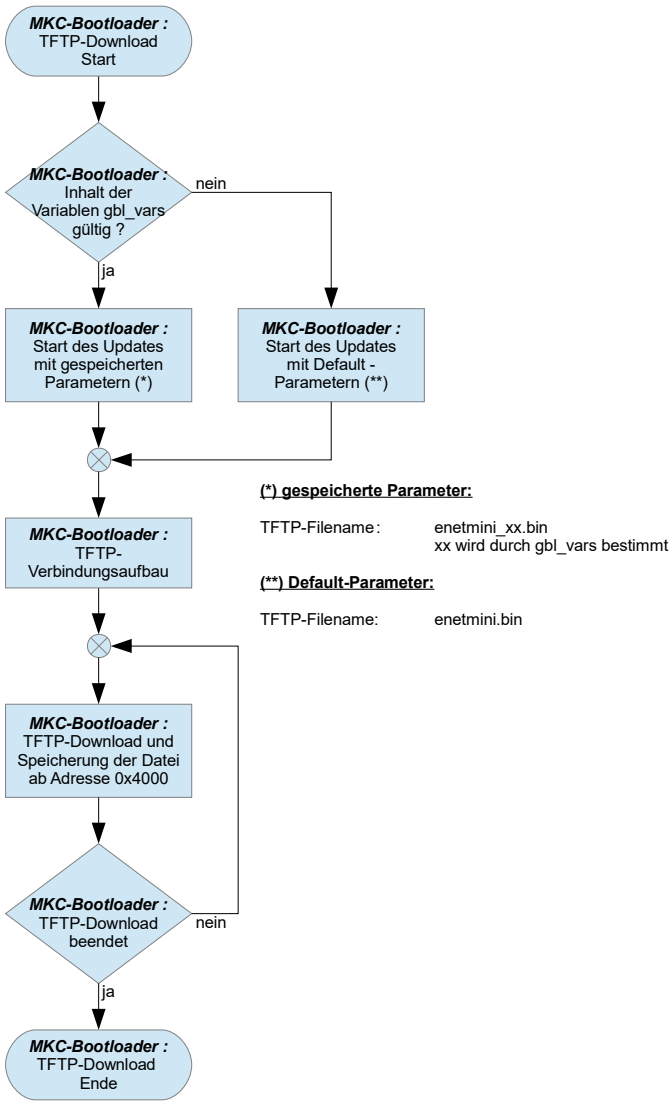


Abbildung 5: MKC-Bootloaders, TFTP-Download

Hinweis: Die Struktur `decl_bl_vars` wird in Abbildung 7 (Kapitel 3.1) gelistet.

2.3 Sytem Update

In dem Fall, dass nach dem Systemstart keine Bedingung 'Forced System Update' anliegt, prüft der MKC-Bootloader, ob ein gültiges Programm ab der Programm-Startadresse (Adresse $00004000) gespeichert ist. Hierzu werden die ersten 2 Werte (jeweils 32 Bit) dieses Speicherbereiches auf Plausibilität getestet.

Ist dieser Test erfolgreich, so wird das Programm angesprungen und ausgeführt. Falls dieser Test negativ ausfällt (dieses bedeutet, dass kein gültiges Programm an der Startadresse gespeichert ist), baut der MKC-Bootloader eine Netzwerkverbindung zu einem BOOTP-Server auf, fordert gültige Netzwerkkonfigurationsdaten an. Anschließend verbindet sich der Bootloader mit dem TFTP-Server und lädt von diesem Server Daten. Diese Daten werden ab der Programm-Startadresse ($00004000) in das interne Flash geschrieben. Ist die Programmierung aller Daten erfolgreich beendet worden, wird dieses Programm ausgeführt.

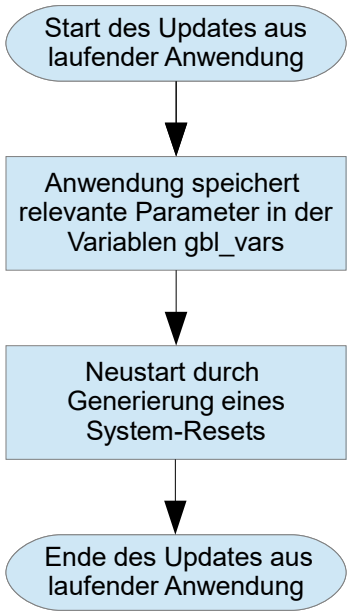


Abbildung 6: Funktion des MKC-Bootloaders nach Aufruf aus der Anwendung

2.4 Forced Sytem Update

Der von MKC implementierte und mitgelieferte MKC-Bootloader prüft, ob die Bedingung 'Forced System Update' anliegt. Diese Bedingung wird durch die Aktivierung des entsprechenden Eingangs /FSU_IN zum Zeitpunkt des Systemstarts generiert. Ist dieses der Fall, wird ein Systemupdate durchgeführt.

Der sich anschließende Datentransfer fordert von dem TFTP-Server die Datei 'enetmini.bin' an, diese übertragenen Daten werden geladen und im internen Flash (Adresse $00004000) abgespeichert und danach ausgeführt.

2.5 Hinweise zur Implementierung und zu der Variablen gbl_vars

Alle notwendigen Parameter für den Netzwerk-Transfer werden in einem speziellen Speicherbereich des internen RAMs abgelegt (am obersten Ende des physikalischen Speichers, dieser Bereich ist reserviert für die globale Variable `gbl_vars`). Weiterhin werden in der Datei 'bl_vars_h.h' alle notwendigen Definitionen für die von MKC erfolgte Anpassung des allgemeinen mitgelieferten Bootloader-Codes von TI gesetzt. Diese ist im Kapitel 3.1 veröffentlicht.

Die Variablen `ui32YIAddr`, `ui32GIAddr`, `ui32SIAddr`, `ui32SIAddr` und `ui32FileMode` legen die Werte fest, die der Bootloader beim TFTP-Request nutzt. Der angeforderte Dateiname wird aus dem aktuellen Zustand der Variablen `ui32FileMode` zusammengesetzt. Hierbei wird 'enetmini.bin' um ein Zählindex erweitert; so wird aus 'enetmini.bin' für den Netzwerkaufruf der Name 'enetmini_xy.bin' erzeugt. 'xy' wird aus dem höchsten Bit mit dem Wert '0' der Variablen `ui32FileMode` generiert. Dabei gilt folgendes:

höchste '0'-Bit ist das Bit0	'enetmini_00.bin'
höchste '0'-Bit ist das Bit1	'enetmini_01.bin'
höchste '0'-Bit ist das Bit2	'enetmini_02.bin'
...	
höchste '0'-Bit ist das Bit30	'enetmini_1e.bin'
höchste '0'-Bit ist das Bit31	'enetmini_1f.bin'
kein Bit ist '0'	'enetmini .bin'

Durch dieses Verfahren kann ein sequenzielles Systemupdate mit mehreren unabhängigen Programmteilen einfach durch Rücksetzen (1 → 0) der entsprechenden Bits in der Variablen `ui32FileMode` implementiert werden. Jedes (Teil-)Programm setzt die `ui32FileMode` am Programmende passend für den Bootloader, dieser fordert die nächste Datei an. Sollte ein Fehler während der Ausführung des (Teil-)Programms auftreten, so wird beim Systemstart dieses Programm nochmal aufgerufen oder per TFTP angefordert und der Schritt somit wiederholt.

Hinweis: Die Struktur `decl_bl_vars` wird in Kapitel 3.1 gelistet.

3 Anhang

3.1 Struktur `_decl_bl_vars`, statische Variable `gbl_vars`

Folgende Strukturdefinition ist in der Quelldatei bl_vars_h.h von MKC veröffentlicht.

```
/**
 * Copyright Â© 2016
 * MKC Michels & Kleberhoff Computer GmbH
 *
 * 1.1.2.14 eNetMiniDefCon
 * 1.1.2.9 MkcHelperLib
 * 1.1.2.4 MkcDeviceLib
 *
 * */

#ifndef _BL_VARS_H
#define _BL_VARS_H

/**
 * Test mode active
 * */
#define TESTMODE_ACTIVE 0x80000000L

/**
 * Structure definition used to store boot loader specific vars.
 * */
typedef struct _decl_bl_vars
{
  /**
   * Boot loader major version.
   * */
  uint8_t ui8MajorVersion;
  /**
   * Boot loader minor version.
   * */
  uint8_t ui8MinorVersion;
  /**
   * Boot loader mkc version.
   * */
  uint8_t ui8MkcVersion;
  /**
   * Boot loader internal version.
   * */
  uint8_t ui8InternalVersion;
  /**
   * Extension Bits for the TFTP request file.
   * */
  uint32_t ui32FileMode;
  /**
   * Network client (your) IPv4 Address, assigned by the BOOTP server.
   * */
  uint32_t ui32YIAddr;
  /**
   * Network gateway IPv4 Address, assigned by the BOOTP server.
   * */
  uint32_t ui32GIAddr;
  /**
   * TFTP server IPv4 Address, assigned by the BOOTP server.
   * */
  uint32_t ui32SIAddr;
```

```
  /**
   * Pointer to the test mode flag.
   * */
  uint32_t *pui32TestMode;
  /**
   * Pointer to the assembled board information.
   * */
  char *pcAssembledBoard;
  /**
   * Pointer to the ethernet mac address.
   * */
  uint8_t *pui8MacAddress;
  /**
   * Maximum assembled board information length.
   * */
  uint16_t ui16AssembledBoardLength;
  /**
   * Maximum mac address length.
   * */
  uint16_t ui16MacAddressLength;
} decl_bl_vars;

/**
 * Member to store boot loader specific data
 * */
extern decl_bl_vars gbl_vars;

extern void bl_clearFileMode();
extern void bl_setFileMode(uint8_t ui8Step);

#endif /* _BL_VARS_H */
```

Abbildung 7: Struktur `decl_bl_vars`, statische Variable `gbl_vars`.