ebFlashSetup

Benutzerhandbuch

Version 1.1

Peer Georgi

5. Mai 2005

Inhaltsverzeichnis

Kapitel 1

Allgemeines

Das Ziel von *ebFlashSetup* ist die Ausrichtung von Dateiinhalten, sodass diese in einen Festwertspeicher (z.B. Flash), eines eingebetteten Computersystems, programmiert werden können. Dabei erzeugt das Programm alle notwendigen Daten, um die Konfiguration des Festwertspeichers vorzunehmen. Der Umstand, dass der Übertragungsablauf koordiniert werden muss, berücksichtigt das Programm ebenfalls, da es Hand in Hand mit dem Kommunikationsprogramm („*ebStartUp*"), zur Übertragung der Daten zusammen arbeitet und erstellt dabei aus der Projektdatei zusätzlich eine Konfigurationsdatei für ebStartUp.

1.1 Funktion und Einsatz

Das Programm ebFlashSetup gliedert sich in eine Gruppe von weiteren Programmen ein, die gemeinsam das Ziel haben, ein eingebettetes Computersystem aus einem Festerwertspeicher (Flash), selbständig zu starten. Dabei sind folgende weiteren Programme beteiligt:

- Der Boot-Loader des eingebetteten Computersystems
  Er lädt Programme und Daten aus dem Festwertspeicher in den Arbeitsspeicher und führt ein Programm aus.
- Das Festwertspeicher Konfigurationsprogramm
  Es läuft auf dem Zielsystem und programmiert den Festwertspeicher.
- Das Übertragungsprogramm „ebStartUp"
  Es überträgt Programme und Daten in den Arbeitsspeicher des Zielsystems und startet ein Programm
- Das Datenaufbereitungsprogramm „ebFlashSetup"

ebFlashSetup hat dabei die Aufgabe, die Belegung des Festwertspeichers und des Arbeitsspeichers des Zielsystems zu verwalten und das Übertragungsprogramm zu konfigurieren. Es verwaltet den Arbeitsspeicher des eingebetteten Zielsystems und zwei Situationen.

1. Während des Startvorgangs
2. Während der Übertragung der Daten mit ebStartUp

Während des Startvorgangs müssen definierte Datenblöcke aus dem Festwertspeicher in den Arbeitsspeicher kopiert werden. Dabei wird die Zieladresse im Arbeitsspeicher in der Konfigurationsdatei spezifiziert.
Bei der Übertragung dient der Arbeitsspeicher des Zielsystems nur als temporärer Zwischenspeicher und ebFlashSetup sorgt dafür, dass es keine Überlappungen und Konflikte zwischen den Daten gibt.

Zusätzlich wird der Festwertspeicher verwaltet. Es wird dafür gesorgt, dass er optimal genutzt wird und keine ungenutzten Bereiche zwischen den abgelegten Daten entstehen. Weiterhin wird eine Alloktionstabelle (ähnlich einer **F**ile **A**llocation **T**able) erzeugt, welche der Boot-Loader zum Einlesen der Daten verwendet.

Nachdem alle Adressen der Daten in jeder Situation feststehen, wird eine Konfigurationsdatei für ebStartUp erstellt. Diese kann für Einrichtungsvorgang verwendet werden, bei dem die Daten zum Zielsystem übertragen und in den Festwertspeicher geschrieben werden. Zur Kommunikation mit dem Programm, welches die Daten in den Festwertspeicher schreibt, wird eine Tabelle benötigt, die dem Programm beschreibt, wo sich die Daten im Arbeitsspeicher befinden und wo sie im Festwertspeicher abgelegt werden. Der Aufbau dieser Tabelle ist in diesem Dokument beschrieben und wird ebenfalls von ebFlashSetup erzeugt.

1.2 Verwendung

Das Program ebFlashSetup ist ein Kommandozeilenprogramm, welches durch eine Konfigurationsdatei (Projektdatei), konfiguriert wird. Das Verhalten des Programms kann durch die Angabe von Kommandozeilenparametern und Umgebungsvariablen gesteuert werden.

1.2.1 Kommandozeilenparameter

Kommandozeilenparameter werden dem Programm bei der Ausführung übergeben. Einige Kommandozeilenparameter sind optional und müssen nicht unbedingt angegeben werden. In diesem Fall werden Voreinstellungen verwendet, die jedoch bei expliziter Angabe eines entsprechenden Kommandozeilenparameters überschrieben werden können. Alle Kommandozeilenparameter haben eine Kurz- und eine Langform. Die Kurzform (z.B. „-p"), spart Schreibarbeit, während die Langform („–project"), einen gewissen Dokumentationswert hat.

1.2.1.1 Voreinstellungen

Name der Projektdatei
Nach dem Start des Programms wird im aktuellen Verzeichnis nach einer Projektdatei gesucht, welche den Namen „setup.prj" trägt. Diese Voreinstellung kann mit Hilfe der Kommandozeilenoption „-p Dateiname" oder „–project Dateiname" geändert werden.

Name der ebStartUp-Konfigurationsdatei
Weiterhin wird der Name „setup.ebs" als Dateiname für die, zu generierende, ebStartUp-Projektdatei voreingestellt. Dies kann mit der Option „-o Dateiname" oder „–output Dateiname" geändert werden.

Ausgaben
Per Voreinstellungen werden nur Fehler, Warnungen und wichtige Informationen auf der Standartausgabe ausgegeben. Dies kann mit den Parametern „-q" bzw. „–quiet" unterbunden werden. In diesem Fall werden nur noch Fehler ausgegeben.
Der Parameter „–verbose" hingegen sorgt für die Ausgabe aller möglichen Informationen auf der Standartausgabe.

1.2.1.2 Hilfe

Der Parameter „–help" gibt eine Liste möglicher Kommandozeilenparameter aus. Dabei wird jeder Eintrag kurz beschrieben. Im Zweifelsfalle sei auf diese Möglichkeit verwiesen, da sie unter Umständen aktueller ist, als dieses Handbuch.

1.2.1.3 Dokumentation

Der Parameter „-d" führt zur Erstellung einer Dokumentation im Latex-Format. Näheres dazu im Kapitel „Extras".

1.2.2 Umgebungsvariablen

Umgebungsvariablen können in der Projektdatei verwendet werden. Sie werden wie normale Variablen behandelt und können dazu dienen eine Projektdatei konfigurierbar zu machen. Siehe dazu 2.3.5.

1.3 Unterstützte Zielsystem-Platformen

Das Programm wurde ursprünglich für die Datenaufbereitung und Konfiguration eine ARM9-basierenden, eingebetteten Computersystems entwickelt. Es ist jedoch für alle eingebetteten Computersystem geeignet, die einen entsprechenden Boot-Loader und Übertragungsprogramme zur Verfügung haben.

1.4 Unterstützte Platformen

- Linux

Derzeit wird nur Linux unterstützt.

Kapitel 2

Die Projektdatei

Die Projektdatei ist eine Textdatei, welche mit einem gewöhnlichen Texteditor editiert werden kann. Dabei werden die derzeit üblichen Zeilenendungen der Betriebssysteme „MacOS", „Windows" und Unix unterstützt. Sie ist streng hirarisch aufgebaut und verwendet Schlüsselworte, Variablen und Blöcke zur Strukturierung.

2.1 Syntax

Die Syntax erinnert an C bzw. C++. Das Ende einer Zuweisung wird durch ein Semikolon besiegelt. Kommentare sind mit „// Kommentar" möglich und gelten bis zum Ende der Zeile. Ein Block wird mit „{" geöffnet und mit „}" geschlossen.
Gross und Kleinschreibung wird **unterschieden**!

2.2 Aufbau

Die Projektdatei besteht aus zwei Blöcken. Der Konfigurationsblock und der Anweisungssektion. Die Anweisungssektion enthält die Anweisungsblöcke. Ein Anweisungsblöck spezifiziert die Datei, deren Übertragungsweg, den Typ und die Adresseigenschaften der Datei, die Einfluss auf die Adressvergabe haben.

2.2.1 Konfigurationsblock

Dieser Block wird mit dem Schlüsselwort „GlobalSettings" eingeleitet und hat die Aufgabe, einige Variablen zu definieren, die für den späteren Verlauf notwendig sind. Es können auch eigene Variablen definiert und verwendet werden. Diese sind jedoch nur im Konfigurationsblock gültig.

2.2.2 Anweisungssektion

Die Anweisungssektion wird mit dem Schlüsselwort „TransferSetup" eingeleitet und enthält die Anweisungsblöcke.

2.3 Variablen

In der Projektdatei wird massiver Gebrauch von Variablen gemacht. Es wird dabei zwischen verschiedenen Arten von Variablen unterschieden.

- Einfache Variablen
- Automatische Variablen
- Variablen zur Konfiguration

2.3.1 Einfache Variablen

Es handelt sich hierbei um Variablen im eigentlichen Sinne. Sie werden durch ihre Verwendung definiert und einfache Zuweisungen sehen folgendermassen aus:

```
// Eine einfache Zuweisung
Filename = „datei.bin”;
// Zuweisung mit dem Inhalt einer Variable
Imagename = $(Filename);
// Zuweisung mit dem Inhalt einer Umgebungsvariable
UserName = $$(USER);
```

Sollte eine Variable nicht aufgelöst werden können, da beispielsweise ein Umgebungsvariable nicht definiert ist, so wird ein entsprechender Fehler ausgegeben. Finite rekursive Zuweisungen sind erlaubt.

2.3.2 Automatische Variablen

Es handelt sich hierbei um eingebaute Variablen, deren Inhalt von der Verwendungssituation abhängt. Sie werden verwendet, um eine automatische Wertzuweiseung vorzunehmen, wobei der zuzuweisende Wert durch ebFlashSetup berechnet wird.

```
// Die nächste freie Adresse im Transferspeicher
TransferLoadAddress = $(@FIRST_POSSIBLE_TRANSFER_SDRAM_ADDRESS);
```

Bei dieser Zuweisung wird der Variable „TransferLoadAddress”, die nächst mögliche freie Adresse im Arbeitsspeicher des Zielsystems zugewiesen. Der tatsächliche Wert wird beim Zeitpunkt der Zuweisung errechnet und durch die Vorgschichte beeinflusst. Folgende Variablen gelten als automatisch. Dennoch haben sie nur in den Transfer-Blöcken Gültigkeit.

- ***@FIRST_POSSIBLE_TRANSFER_SDRAM_ADDRESS***
  Der Inhalt beschreibt die nächste, unverwendete Adresse im Arbeitsspeicher des Zielssystems, währen der Übertragung
- ***@FIRST_POSSIBLE_FLASH_ADDRESS***
  Inhalt ist die nächste, freie Adresse im Festwertspeicher
- ***@FIRST_POSSIBLE_BOOTLOAD_SDRAM_ADDRESS***
  Inhalt ist die nächste, freie Adresse im Arbeitsspeicher des Zielsystems, während des Startvorgangs

- ***@FLASH_FILE_MAP_IMAGE_NAME***
  Der Name der zu generierenden Datei, welche die Akationstabelle beinhaltet.
- ***@FLASH_SETUP_MAP_IMAGE_NAME***
  Der Name der zu generierenden Date Datei, welche die Tabelle für den Einrichtungsvorgang enthält
- ***@IMAGE_FILE_SIZE***
  Die grösse der aktuellen Image-Datei.

Die Verwendung automatischer Variablen vereinfacht die Erstellung der Projektdatei erheblich, da die Berechnung der Startadressen häufig von ebStartUp selbst durchgeführt werden kann.

2.3.3 Variablen zur Konfiguration

Damit Speicheradressen berechnet werden können, muss das Programm einige Informationen bezüglich der verwendeten Speicher bekommen. Diese Informationen werden mit Hilfe eingebauter Variablen festgelegt, die in der Projekt-Datei definiert sein müssen. Der Ort für diese Zuweisungen ist der Konfigurationsblock. Folgende Variablen müssen definiert werden:

- ***SDRAM_SIZE***
  Gibt die grösse des Arbeitsspeichers, des Zielsystems, in Bytes, an.
- ***FLASH_SIZE***
  Gibt die grösse des Festwertspeichers in Bytes an.
- ***FLASH_FILE_MAP_IMAGE_NAME***
  Hier muss ein Dateiname angegeben werden, wie die erzeugte Allokationstabelle für den Boot-Loader heissen soll.
- ***FLASH_FILE_MAP_ITEMS***
  Hier wird die Anzahl der maximal möglichen Einträge in der Allokationstabelle vorgegeben. Sollte sich der Wert als zu klein herausstellen, wird ein Fehler ausgegeben.
- ***FLASH_SETUP_MAP_IMAGE_NAME***
  Der Dateiname für die Tabelle für das Flash-Einrichtungsprogramm.
- ***FLASH_SETUP_MAP_ITEMS***
  Die Anzahl der Einträge in der Tabelle für das Flash-Einrichtungsprogramm. Sollte sich der Wert als zu klein herausstellen, wird ein Fehler ausgegeben.
- ***SDRAM_BOOTLOAD_START_ADDRESS***
  Der Startwert für die erste, freie Adresse im Arbeitsspeicher des Zielsystems beim Systemstart.
- ***SDRAM_TRANSFER_START_ADDRESS***
  Der Startwert für die erste, freie Adresse im Arbeitsspeicher des Zielsystems bei der Übertragung.
- ***FLASH_START_ADDRESS***
  Ab dieser Adresse wird der Festwertspeicher verwendet.

Alle Ganzzahligen Werte, müssen als Hexadezimalzahl mit vorangestelltem „0x" übergeben werden. Zeichenketten, wie Dateinamen in Anführungszeichen. Überlappungen werden dabei automatisch vermieden.

2.3.4 Gültigkeitsbereiche

Eingebaute Variablen, die zur Konfiguration verwendet werden haben sowohl im Konfigurationsblock, alsauch in den Anweisungsblöcken Gültigkeit. Benutzerdefinierte Variablen sind nur lokal in dem Block gültig, wo sie definiert wurden. Die automatischen Variablen haben spezielle Anwendungen und sind nur in den Anweisungsblöcken sinnvoll einsetzbar.

2.3.5 Umgebungsvariablen

Umgebungsvariablen können an jedem Ort eingesetzt werden, wo auch eine Zuweisung mit einer Konstante oder anderen Variable möglich ist. Sollte sie bei der Anwendung der Projektdatei, nicht definiert sein, wird ein Fehler ausgegeben. Sie können dafür verwendet werden, ein Projekt konfigurierbar zu machen.

2.4 Die Anweisungsblöcke

Ein jeder Anweisungsblock enthält folgende Informationen.

- „***Image***" = [Dateiname]
  Der Dateiname.
- „***BootLoadAddress***" = [Ganzzahl]
  Die Startadresse beim Systemstart.
- „***TransferLoadAddress***" = [Ganzzahl]
  Die Startadresse bei der Übertragung, während des Einrichtungsprozesses.
- „***StoreToFlash***" = [yes | no]
  Die Information, ob die Datei in den Festwertspeicher geschrieben werden soll.
  - „***FlashAddress***" = [Ganzzahl | ignore]
    Die Startadresse innerhalb des Festwertspeichers.
  - „***Execute***" = [yes | alternate | no]
    Gibt an, ob die Datei während des Systemstarts ausgeführt werden soll.
- „***Protocol***" = [USB_DFU | USB_LDBA | ...]
  Gibt das Protokoll an, mit dem die Datei während des Einrichtungsvorgangs übertragen werden soll.
- „***ProtocolParam***"
  Dies leitet einen Block ein, dessen Inhalt direkt in die Konfigurationsdatei von ebStartUp durchgereicht wird. Der Inhalt hat auf ebFlashSetup selbst, keinen Einfluss.

Die Reihenfolge dieser Angaben, innerhalb eines Anweisungsblocks, ist unerheblich.

Ein typischer Anweisungsblock könnte zum Beispiel so aussehen:

```
Transfer
  {
    StoreToFlash = yes;
    Image = "LdDataFlash";
    FlashAddress = 0x0;
    BootLoadAddress = ignore; // the cpu will load this automaticaly //
    TransferLoadAddress = $(@FIRST_POSSIBLE_TRANSFER_SDRAM_ADDRESS);
    Execute = no;
    Protocol = "USB-LDBA";
    ProtocolParam
      {
        Comment = "Loading LdDataFlash...\\\\n  ";
         Filename = "$(Image)";
         LoadAddress = $(TransferLoadAddress);
      }
  }
```

Dieser Block beschreibt eine Datei („LdDataFlash”), die in den Festwertspeicher geschrieben werden soll. Die Startadresse im Flash wird explizit mit Adresse 0 vorgegeben. Die Startadresse beim Systemstart wird hingegen ignoriert, da die MCU selbstständig den ersten Block aus dem Festwertspeicher lädt, wird dieser Eintrag vom Bootprogramm ignoriert. (Es handelt sich übrigens um das Boot-Programm selbst.)
Die Startadresse bei der Übertragung ist uninteressant und deren Berechnung wird ebFlashSetup überlassen. Das Feld Execute = no besagt, dass das BootProgramm diesen Eintrag nicht ausführen soll. (Dies hat die MCU bereits automatisch getan).

Als Übertagungsprotokoll wird das LDBA-Protokol verwendet dessen spezielle Parameter im ProtocolParam-Block hinterlegt sind.

Kapitel 3

Beispiele

Die Beispiele können als Ausgangspunkt für eigene Projekte gelten. Dennoch ist die Anpassung der variablen Parameter in den meisten Fällen notwendig.

3.1 Konfigurationsblock

Dieser Konfigurationsblock gilt für ein Zielsystem mit 16MByte Arbeitsspeicher und einem Festwertspeicher mit einer Grösse von 8MByte. Der Arbeitsspeicher beginnt ab der physikalischen Adresse 0x0x20000000.
Die erste gültige Startadresse beim Systemstart liegt bei 0x20000000, wobei die automatische Variable „***FIRST_POSSIBLE_BOOTLOAD_SDRAM_ADDRESS"*** in den seltensten Fällen sinnvoll ist, da für die Startadressen der Daten, während des Systemstarts, ja meist Vorgaben gelten.
Die Einstellung der Variable „***SDRAM_TRANSFER_START_ADDRESS***" lässt ein halbes Megabyte Platz für ein mögliches Programm, dass sich dort befindet. (z.B. das Übertragungsprogramm und das Festwertspeicher-Einrichtungsprogramm).

```
GlobalSettings
  {
    SDRAM_SIZE = 0x1000000; // 16MB
    FLASH_SIZE = 0x800000;  // 8MB
    FLASH_FILE_MAP_IMAGE_NAME = "filemap.img"; // the name of filemap-image
    FLASH_FILE_MAP_ITEMS = 0x10; // number of entries in file_map
    FLASH_SETUP_MAP_ITEMS = 0x10; // number of entries in setup-map
    FLASH_SETUP_MAP_IMAGE_NAME = "setupmap.img";
    // start values //
    SDRAM_BOOTLOAD_START_ADDRESS = 0x20000000;
    // 512kB reserved for ldba-bootloader and dataflash setup tool //
    SDRAM_TRANSFER_START_ADDRESS = 0x20000000;
    FLASH_START_ADDRESS = 0x0;    // start with this address //
  }
```

3.2 Ein konkretes Beispiel

Das Zielsystem soll das MCU-Modul der Firma Conitec Datensysteme GmbH, sein. Es handelt sich um ARM9-basierendes, eingebettetes Computersystem, welches den AT91RM9200 Mikrokontroller einsetzt. Das Computersystem verfügt über 128MByte Arbeitsspeicher und einen 8MByte grossen Festwertspeicher.
Nach dem hier beschriebenen Einrichtungsvorgang, soll auf dem Computersystem ein Programm automatisch, nach dem Reset aus dem Festwertspeicher laden und zur Ausführung bringen. Es wird davon ausgegangen, dass der Festwertspeicher kein gültiges Startprogramm enthält.
In diesem Fall wartet die MCU nach dem Reset auf eine XMODEM oder USB-DFU Verbindung, mit der sie ein Programm empfängt und zur Ausführung bringt. Für die genauen technischen Hintergründe sei auf das Datenblatt der MCU verwiesen.

Es liegen folgende Programme vor:

- LdDataFlash
  Der Boot-Loader.
- StDataFlash
  Das Festwertspeicher-Einrichtungsprogramm.
- LdBigAppUSB
  Das Übertragungsprogramm.
- Das Testprogramm, dass bei dem Systemstart ausgeführt werden soll
  Nennen wir es „ping.bin”.

3.2.1 Vorbereitungen

Zunächst müssen Informationen zusammengetragen werden, an welchen Adressen die Programme zur Ausführung kommen und welche Voraussetzungen sie sonst mitbringen.

LdDataFlash

Dieses Programm wird von der automatisch MCU in ihren Cache geladen. Die Voraussetzung ist, dass sich das Programm an Adresse 0 im Festwertspeicher befindet. Weiterhin erwartet Das Programm die Allokationstabelle im Flash, die den weiteren Inhalt des Festwertspeichers beschreibt. Diese Adresse ist im Programm vorgegeben mit 0x4000 im Flash. Das ist eine sinnvolle Vorgabe, da die MCU genau 16kBytes aus dem Flash lädt und der Bootloader diese grösse nicht überschreiten darf. Die Allokationstabelle schliesst sich also unmittelbar nach dem Bootloader im Festwertspeicher an.
Es fehlt noch die Information, an welcher Startadresse das Programm zur Ausführung kommt. Diese wird beim Linken des Programms vorgegeben und entspricht der Adresse 0x200000. Dies ist die Adresse des MCU-Cache, wohin das Programm geladen wird
Damit sind, unter anderem, folgende Informationen gewonnen:

- **Startadresse bei der Ausführung: *0x200000***
  Da das Programm wird jedoch durch die MCU ausgeführt wird und nicht durch BootLoader, kann sie im ebFlashSetup-Projekt ignoriert werden.
- **Startadresse im Festwertspeicher: *0x0***
- **Startadresse der Allokationstabelle im Festwertspeicher: *0x4000***

StDataFlash

Das Programm soll die Konfiguration des Festwertspeichers vornehmen und wird nach Abschluss der Konfiguration automatisch gestartet. Es erwartet eine Einrichtungstabelle an einer, im Programm spezifizierten, Adresse im Arbeitsspeicher. Die Struktur dieser Tabelle entspricht der in Kapitel 5.2 vorgestellten Struktur.

- **Startadresse bei der Ausführung: *0x20080000***
  Diese Angabe resultiert aus dem Linker-Skript
- **Startadresse der Einrichtungstabelle: *0x20070000***
  Diese Angabe wurde im Programm selbst festgelegt
- Das Programm wird ***nicht*** in den Festwertspeicher geladen, da es diesen ja konfigurieren soll.

ping.bin

Es handelt sich um das Programm, welches automatisch zur Ausführung gebracht werden soll. Die Startadresse beim Ausführungszeitpunkt bestimmt der jenige, der das Programm entwickelt und muss das Linkevorgang angeben. Die Position im Festwertspeicher interessiert nicht und soll von ebFlashSetup automatisch so festgelegt werden, dass möglichst wenig Lücken entstehen. Besondere Anforderungen an Datenbereiche sind nicht vorhanden.

LdBigAppUSB

Dieses Programm stellt das Gegenstück zu „ebStartUp” dar und bietet eine schnelle Kommunikation mit Hilfe der USB-Schnittstelle an. Es bietet die Voraussetzung, dass die Daten zum Zielsystem übertragen werden und initiiert den Start von StDataFlash, damit der Einrichtungsvorgang ablaufen kann.

Da die MCU noch einen leeren Festwertspeicher hat, wird das Programm mit Hilfe des USB-DFU-Protokolls übertragen und die MCU wird es sofort nach dem Empfang starten. Nach dem Start stellt *LdBigAppUSB* selbst einen Übertragungskanal zur Verfügung, der es erlaubt die anderen Daten in den Arbeitsspeicher der MCU zu übertragen. Das zur Verfügung gestellte Protokoll ist das **LDBA** (**L**oa**d** **B**ig **A**pplications) Protokoll. Um die Startaddresse muss man sich an dieser Stelle nicht kümmern, da das Programm mit Hilfe des DFU-Protokolls übertragen wird, ist sie von der MCU vogegeben und beim Linkvorgang des Programms bereits berücksichtigt.

Die Vorbereitungen sind jetzt abgeschlossen und die Projektdatei für ebFlashSetup kann erstellt werden.

3.2.2 Erstellen der Projektdatei

Zunächst müssen die globalen Parameter definiert werden. Es unterscheidet sich nicht wesentlich vom Beispiel in 3.1.

GlobalSettings

```
{
  SDRAM_SIZE = 0x10000000; // 256MB
  FLASH_SIZE = 0x800000;  // 8MB
  FLASH_FILE_MAP_IMAGE_NAME = "filemap.img";
  FLASH_FILE_MAP_ITEMS = 0x4;
  FLASH_SETUP_MAP_ITEMS = 0x10;
  FLASH_SETUP_MAP_IMAGE_NAME = "setupmap.img";
  // start values //
  SDRAM_BOOTLOAD_START_ADDRESS = 0x20000000;
  SDRAM_TRANSFER_START_ADDRESS = 0x20000000;
  FLASH_START_ADDRESS = 0x0;
}
```

Es folgt die Anweisungssektion:

```
TransferSetup
  {
    // alle Anweisungsblöcke befinden sind innerhalb
    // dieses Blockes...
```

Der nächste Schritt ist die Initiierung des Verbindungsaufbaus. Somit muss LdBigAppUSB als erstes Programm übertragen werden. Es wird von der MCU sofort zur Ausführung gebracht.

```
  Transfer
    {
      TransferLoadAddress = 0x200000; // internal SRAM (CPU)
      Image = "LdBigAppUSB";
      StoreToFlash = no;
      Protocol = "USB-DFU";
      ProtocolParam
        {
          Filename = "$(Image)";
        }
    }
```

Es folgt die Definition für das Startprogramm (Bootloader). Hier wird die Adresse im Flash spezifiziert, die Adresse während der Übertragung interessiert hingegen nicht, da das Programm im Arbeitsspeicher nur zwischengespeichert wird. Das Execute - Feld ist eine Information für den BootLoader. Da dies der BootLoader ist, und direkt durch die MCU ausgeführt wird, ist das Feld hier auf no gesetzt.

```
Transfer
  {
    TransferLoadAddress = $(@FIRST_POSSIBLE_TRANSFER_SDRAM_ADDRESS);
    Image = "LdDataFlash";
    StoreToFlash = yes;
    FlashAddress = 0x0;
    BootLoadAddress = ignore;
    Execute = no;
    Protocol = "USB-LDBA";
    ProtocolParam
      {
        Filename = "$(Image)";
        LoadAddress = $(TransferLoadAddress);
      }
  }
```

Die Allokationstabelle für den Bootloader folgt. Sie wird an die spezifizierte Adresse 0x4000 im Festwertspeicher abgelegt.

```
Transfer
  {
    TransferLoadAddress = $(@FIRST_POSSIBLE_TRANSFER_SDRAM_ADDRESS);
    Image = $(FLASH_FILE_MAP_IMAGE_NAME);
    StoreToFlash = yes;
    FlashAddress = 0x4000;
    BootLoadAddress = ignore;
    Execute = no;
    Protocol = "USB-LDBA";
    ProtocolParam
      {
        Filename = "$(FLASH_FILE_MAP_IMAGE_NAME)";
        LoadAddress = $(TransferLoadAddress);
      }
  }
```

Das Programm, welches zur Ausführung gebracht werden soll.

```
Transfer
 {
   TransferLoadAddress = $(@FIRST_POSSIBLE_TRANSFER_SDRAM_ADDRESS);
   Image = "ping.bin";
   StoreToFlash = yes;
   FlashAddress = $(@FIRST_POSSIBLE_FLASH_ADDRESS);
   BootLoadAddress = 0x20780000;
   Execute = yes;
   Protocol = "USB-LDBA";
   ProtocolParam
     {
       Filename = "$(Image)";
       LoadAddress = $(TransferLoadAddress);
     }
 }
```

Alle Datenübertragungen, die den Festwertspeicher selbst betreffen, sind jetzt abgeschlossen. Es folgt die Vorbereitung für das Programm zur Speicherprogrammierung (StDataFlash).

Zunächst soll die Einrichtungstabelle übertragen werden. Diese Tabelle wird von *ebFlashSetup* erstellt. Die Zieladresse ist durch das Programm StDataFlash festgelegt. (Siehe „Vorbereitungen”)

```
// Setup-List for StDataFlash
Transfer
  {
    TransferLoadAddress = 0x20070000;
    Image = $(FLASH_SETUP_MAP_IMAGE_NAME);
    StoreToFlash = no;
    Protocol = "USB-LDBA";
    ProtocolParam
      {
        Filename = "$(FLASH_SETUP_MAP_IMAGE_NAME)";
        LoadAddress = $(TransferLoadAddress);
      }
  }
```

Nachdem all diese Übertragungen definiert wurden, wird jetzt der Einrichtungsvorgang spezifiziert. Das Programm „StDataFlash” soll übertragen und zur Ausführung gebracht werden.

```
Transfer
  {
    TransferLoadAddress = 0x20080000;
    Image = "StDataFlash";
    StoreToFlash = no;
    Protocol = "USB-LDBA";
    ProtocolParam
      {
        Filename = "$(Image)";
        LoadAddress = $(TransferLoadAddress);
        Execute = yes;
      }
  }
} // Diese Klammer schliesst den noch offenen Block „TransferSetup”
```

Diese Datei liegt als Beispielkonfiguration (example1.prj), vor. Folgende Eingabe führt zum Erfolg:

ebFlashSetup -p example1.prj

Nach dem Start von ebFlashSetup befinden sich folgende, generierte, Dateien im aktuellen Verzeichnis:

- filemap.bin - die generierte Allokationstabelle
- setupmap.bin - die generierte Einrichtungstabelle
- ***setup.ebs - die Projektdatei für ebStartUp***

Alles was jetzt noch notwendig ist, um das Zielsystem zu konfigurieren ist folgende Eingabe:

ebStartUp

Wenn das Zielsystem mit USB verbunden ist und ein Reset durchgeführt wird, startet die Konfiguration automatisch. Nach Beendigung des Vorgangs ist das Zielsystem selbst startfähig.

Kapitel 4

Extras

Ein besonderes Merkmal von „*ebFlashSetup*" seine die Fähigkeit, eine Latex-Dokumentation über die erzeugten Speicherbelegungen zu erzeugen. Die Ausgabe kann dabei wahlweise in Deutsch oder Englisch generiert werden. Dokumentiert werden dabei die drei Speicherbelegungstabellen. Hierfür muss bei der Ausführung des Programms der Kommandozeilenparameter „-d [de|en]" angegeben werden. Im aktuellen Verzeichnis befindet sich dann eine Datei mit dem Namen „*memorymap.latex*". Für das obige Beispiel führt folgende Eingabe zur Erzeugung der Dokumentation in englischer Sprache:

```
ebFlashSetup -p example1.prj -d en
latex memorymap.latex
dvips memorymap.dvi
ps2pdf memorymap.ps
```

Nach diesen Eingaben liegt im aktuellen Verzeichnis eine Datei mit dem Namen „***memorymap.pdf***". Für das obige Beispiel ist diese Dokumentation in den nächsten Seiten eingefügt.

4.1 Dokumentation der Speicherbelegungen

4.1.1 Der Arbeitsspeicher während des Startvorgangs

Memory region	Name	Position	Size
0x20780000 | 0x20784133	ping.bin	fixed	16692 Bytes

4.1.2 Die Speicherbelegung des Festwertspeichers

Memory region	Name	Position	Size
0x00000000 | 0x00002b6b	LdDataFlash	fixed	11116 Bytes
0x00004000 | 0x0000421f	filemap.img	fixed	544 Bytes
0x00004224 | 0x00008357	ping.bin	variable	16692 Bytes

4.1.3 Der Arbeitsspeicher während der Konfiguration

Memory region	Name	Position	Size
0x00200000 | 0x00202a57	LdBigAppUSB	fixed	10840 Bytes
0x20000000 | 0x20002b6b	LdDataFlash	variable	11116 Bytes
0x20004138 | 0x20004357	filemap.img	variable	544 Bytes
0x2000435c | 0x2000848f	ping.bin	variable	16692 Bytes
0x20070000 | 0x200701db	setupmap.img	fixed	476 Bytes
0x20080000 | 0x20083427	StDataFlash	fixed	13352 Bytes

Kapitel 5

Interne Strukturen

Die Erzeugung der Allokationstabelle für den Bootloader und die Einrichtungs-Tabelle für das Festwertspeicher-Einrichtungsprogramm findet in Form von Dateien statt, die zusätzlich zum Zielsystem übertragen werden.
Der Aufbau dieser Dateien soll hier offengelegt werden.

5.1 Die Allokationstabelle für den Systemstart

In dieser Struktur befinden sich alle, für den BootLoader notwendigen Informationen über die im Festwertspeicher abgelegten Daten. Es handelt sich um eine statische Struktur, deren Erweiterung zur Laufzeit nicht vorgesehen ist. Die Anzahl der möglichen Einträge kann jedoch in der Projektdatei mit der Konfigurationsvariable „***FLASH_FILE_MAP_ITEMS***" beeinflusst werden. Die Tabelle besteht aus Einträgen der folgenden Struktur:

```
#define DF_BTENTRY_FLAG_USE             (1 << 0) // use the item //
#define DF_BTENTRY_FLAG_COPY            (1 << 1) // copy this item to
sdram //
#define DF_BTENTRY_FLAG_EXEC            (1 << 2) // execute this item by
default //
#define DF_BTENTRY_FLAG_EXEC_ALTERNATE  (1 << 3) // execute this item
alternate //

// size = 16+4+4+4+4 = 32 bytes.
struct tDFBlockTableEntry
  {
    char msItem[16];                // name of image inclusive trailing zero
//
    unsigned int mDfBTEntryFlags;   // load / execution flags //
    unsigned int mDfBTEntryDFStart; // start in dataflash //
    unsigned int mDfBTEntryDFSize;  // size of image //
    unsigned int mDfBTEntryMemStart; // location in sdram where the image to
be load //
  }  __attribute__((packed));
```

Den letzten Eintrag erkennt der Boot-Loader an dessen Initialisierung mit 0. Alle Felder des letzten Eintrags sind mit dem Wert 0 gefüllt.

5.2 Die Einrichtungstabelle für das Festwertspeicher-Einrichtungsprogramm

Diese Struktur wird, während der Einrichtung, in den Arbeitsspeicher des Zielsystems übertragen und dient dem Festwertspeicher-Einrichtungsprogramm als Hinweis, wo sich die Daten im Arbeitsspeicher befinden, die übertragen wurden. Diese Information nutzt das Konfigurationsprogramm für den Festwertspeicher und kopiert die Daten daraufhin in den Flash. Der Aufbau ist ähnlich der Allokationstabelle und ebenfalls statischer Natur. Die Anzahl möglicher Einträge ist mit der Konfigurationsvariable „***FLASH_SETUP_MAP_ITEMS***" möglich und notwendig. Aufgrund der Tatsache, dass diese Datei selbst nicht in den Festwertspeicher kopiert wird, kann die Anzahl der Einträge hoch gewählt werden. Maximale Anzahl möglicher Einträge ist bedingt durch das Konfigurationsprogramm und dessen Vorgaben über die Zieladresse bei der Übertragung.

```
struct cSetupItem
  {
    char msItemName[16];              // zero terminated msItemName[0] == 0
means end of list //
    unsigned int miDFStartAddress;    // start address in dataflash //
    unsigned int miLoadAddress;       // address where the data copied from //
    unsigned int miSize;              // size of data object //
  };
```

Die Anzahl der Einträge erkennt das Konfigurationsprogramm am letzten Eintrag, welcher in allen Feldern den Wert 0 enthält.