ebStartUp
User Manual

Version 1.2

Peer Georgi

March 1, 2006

Contents

Chapter 1

Introduction

„ebStartUp" allows the transfer of files to an embedded target system via USB port. The protocols used depend on which protocol is available on the target system at system start. Normally, at start no operating system is running on the target. Currently the following protocols are supported:

- USB-DFU (**D**evice **F**irmware **U**pgrade)
- USB-LDBA (**L**oa**d** **B**ig **A**pplications)

The USB-DFU protocol is specified within the USB specs and is used by many embedded systems. The USB-LDBA protocol was designed for transferring huge data files to different target memory areas. *ebStartUp* can perform several transfers with different protocols. This way, complex operating system setups can be easily started from the PC side.

1.1 Program structure

The file transfer by *„ebStartUp”* must be supported by the target MCU or by target programs that open communication channels. An ARM9 based target system already supports the USB-DFU protocol in the MCU firmware, so this protocol is immediately available after system start. The LDBA protocol however has to be installed explicitely by a program on the target system. The following figure shows an example on an AT91RM9200 system.

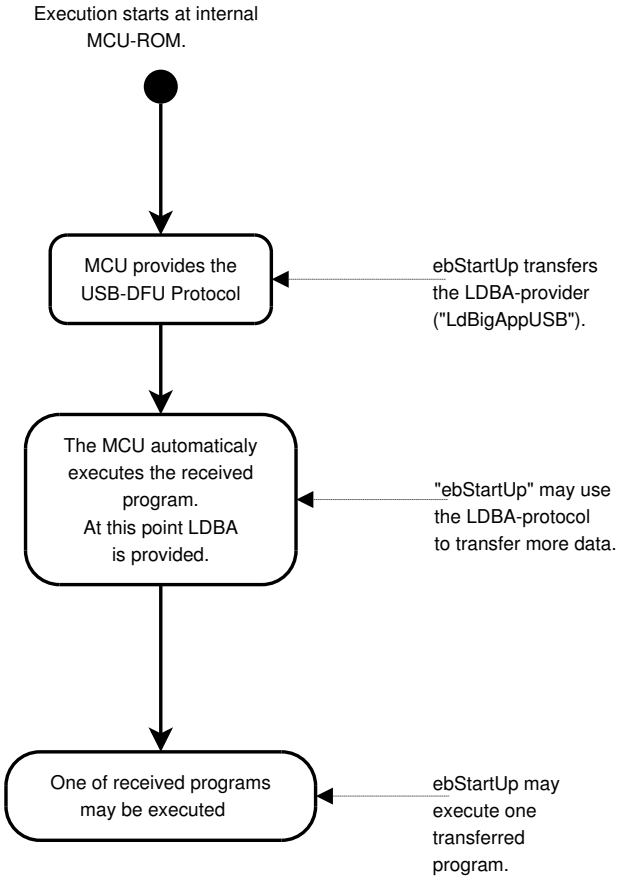

Figure 1.1: Possible system start with „ebStartUp”.

1.2 Useage

„ebStartUp” is a console program, set up by a configuration file. It’s possible and intended to run „ebStartUp” automatically within an integrated development environement. The program behavior depends on command line parameters.

1.2.1 Command line parameters

Command line parameters are passed to the program at execution. Some parameters are optional. If such a parameter is missing, default settings are used. Parameters have a short and long notation. The short notation („-p”) is for lazy typists, while the long notation („–project”) offers better documentation.

1.2.1.1 Configuration

Configuration file name
At program start the configuration file „**setup.ebs**” is read from the current directory. A different file name can be given with the command line option „-p filename” or „–project filename”.

Output
By default, only errors, warnings or essential messages are printed to the standard output device. The parameter „-q” resp. „–quiet” restricts the output to errors only. The parameter „–verbose” causes printout of all sorts of information.

1.2.1.2 Help

The parameter „–help” prints a list of command line parameters with brief descriptions. This list may be more up to date than this manual.

1.3 Supported target platforms

The program was designed for data conversion and configuration for ARM9 based embedded systems. It is suited for all embedded systems that provide a boot loader and transfer programs.

1.4 Supported operating systems

- Linux with kernel version 2.6.x.

At the moment only Linux is supported.

1.5 Requirements

The program accesses the USB virtual file system directly and thus must be executed with administrative rights („root” priority).

Chapter 2

The Configuration file

The configuration file is a plain text file that can be edited with any text editor. The line break codes of „MacOS", „Windows" and „Unix" are supported. The file is hierarchically organized and uses keywords and blocks for structuring.

2.1 Syntax

The syntax is similar to C or C++. Statements end with a semicolon „;". Comments start with „//" and end with the end of the line. Blocks are opened with „{" and closed with „}". Keywords are **case sensitive**.

2.1.1 Reading the configuration file

The configuration file is read in like a script. It consists of one or more instruction blocks. The order of blocks is relevant. The statements within a block can be in any order.

2.1.2 Instruction blocks

An instruction block defines one or several data transfers. The block contains instructions for the data handling and the transfer protocol. Protocol parameters can be set through configuration variables. Defining several transfers within a single block is currently supported by the LDBA protocol only. Otherwise it is possible to specify several instruction blocks.

2.1.2.1 The USB-DFU instruction block

```
setup "USB-DFU"
  {
    image
      {
        Filename = "LdBigAppUSB.bin";
      }
  }
```

The program „*LdBigAppUSB.bin*” is transferred to the target system, using the USB-DFU protocol. The protocol does not require any additional parameters. The program is executed immediately after the transfer. The transfer is considered terminated when the USB-DFU device disconnects.

2.1.2.2 The USB-LDBA instruction block

```
setup "USB-LDBA"
  {
    image
      {
        LoadAddress = 0x20007fc0;
        Filename =  "uImage";
        Execute = no;
        Comment = "Transfer the Linux kernel\\n";
      }
    image
      {
         // Further transfers may follow...
      }
  }
```

In an USB-LDBA instruction block, every transfer process transfers one file and is initiated with the keyword „image”. For every transfer the target memory address is specified by the configuration variable „*LoadAddress*”. The address is given in hex format with leading „0x”. The keyword „Filename” specifies the file to be transferred. The „*Execute*” configuration variable specifies by the „yes” or „no” parameter whether to execute the program after transfer or not.
The „Comment” keyword prints a message on the standard output device of the target system. If „Comment” is not specified, the file name is output instead for observing the transfer process.

Several file transfers can be specified. Please take care to execute only one of the transferred files.

Chapter 3

Examples

The examples can be uses as templates for individual configuration files.

3.1 Starting a test program

The target system is a MCU module by Conitec Datasystems Corp. It's an ARM9 based embedded system comprising the AT91RM9200 microcontroler. The system contains 128 MByte RAM and 8 MByte ROM.

Using „ebStartUp", a test program is to be transferred and started without further setup of the target system. The target ROM does not contain a valid program, thus the MCU attempts to receive a program via the USB interface. For details see the MCU data sheet.

The following programs are available:

- LdBigAppUSB
  The transfer program that provides the LDBA protocol on the MCU.
- The test program to be executed at system start.
  Call it „**pong.bin**".

3.1.1 Prerequisites

At first it has to be determined at which RAM address the program is to be executed. The address is defined by the link process of the test program. An exception is a program that does not contain absolute jump instructions and uses relative memory addresses only. This can be enforced with some compilers. If the program is written in assembler, the developer can enforce this manually, of course.

In the following the program shall be executed at address **0x20000000**.

3.1.2 Writing the configuration file

```
setup "USB-DFU"
  {
    image
      {
       // this way the LDBA protocol is made available
       Filename = "LdBigAppUSB.bin";
      }
  }
setup "USB-LDBA"
  {
    image
      {
        LoadAddress = 0x20000000;
        Filename =  "pong.bin";
        Execute = yes;
        Comment = "Transfer the test program...\\n";
      }
  }
```

This configuration file named „**start-pong.ebs**” is contained in the „examples” folder.

3.1.3 Starting the transfer

The program „ebStartUp” must be executed with root rights because it accesses the USB file system directly.

```
ebStartUp -p start-pong.ebs
```

There are two possibilities for starting the transfer. The transfer can be started by connecting the USB ports[1]. If the USB ports are already connected, the target system can be reset manually to initiate the transfer. Once the transfer is finished, the program is started on the target system.

[1] Dies ist bei MCUs der Fall, die USB-DFU per Firmware unterstützen. Bei dem AT91RM9200 ist das der Fall.

3.2 Starting „u-boot”

„u-boot” is a Linux loader that support a variety of platforms and target systems. It is an open domain program that can start up the Linux kernel and transfer data via Ethernet.

```
setup "USB-DFU"
  {
    image
      {
        Filename = "LdBigAppUSB";
      }
  }
setup "USB-LDBA"
  {
    image
      {
        Filename = "u-boot.bin";
        LoadAddress = 0x21f00000;
        Execute = yes;
      }
  }
```

Please take care to configure „*u-boot*” correctly and adapt it to the target system. The target address for „u-boot” is displayed by the link process. This address can also be found in the file „***board/at91rm9200dk*/config.mk**” in the „u-boot” source directory.

ebStartUp -p start-uboot.ebs

After transfer start the following output is printed to the standard output device of the target system (RS232 console):

```
USB Big Application Loader v0.9a (c) 2004 Conitec Datensystem GmbH.
Setting up USB...
UDP-Warning: - write timeout stage 1
loading LdLinux++ to address 0x20780000...
  ********** - ok
loading u-boot.bin to address 0x21f00000...
  ************************ - ok
Shutting down...
Executing application at 0x21f00000
U-Boot 1.0.0 (Apr 25 2005 - 17:57:17)
U-Boot code: 21F00000 -> 21F14CE0  BSS: -> 21F186B8
DRAM Configuration:
Bank #0: 20000000 32 MB
Atmel: AT49BV1614 (16Mbit)
Flash:  2 MB
DataFlash:AT45DB642
Nb pages:   8192
Page Size:   1056
Size= 8650752 bytes
Logical address: 0xC0000000
Area 0: C0000000 to C0007FFF (RO)
Area 1: C0008000 to C001FFFF (RO)
Area 2: C0020000 to C0027FFF
Area 3: C0028000 to C083FFFF
In:    serial
Out:   serial
Err:   serial
Uboot>
```

The upper part is printed by the transfer program of the MCU.

3.3 Linux Startup

„ebStartUp" can initiate a complete Linux system startup without using „u-boot". This has advantages when other tasks must be performed at startup which are not offered by „u-boot". For this, a program has to be provided that replaces „u-boot" and starts the Linux kernel. On the AT91RM9200 for instance, LdLinux++ can perform this task. Details shall not be discussed here.

In this context, here's a slightly more complex ebStartUp script that starts a Linux kernel on an AT91RM9200 based system:

```
setup "USB-DFU"
  {
    image
      {
        Filename = "LdBigAppUSB.bin";
      }
  }
setup "USB-LDBA"
  {
    image
      {
        LoadAddress = 0x20007fc0;
        Filename =  "uImage";
        Execute = no;
        Comment = "Transfer Linux Kernel\\n";
      }
    image
      {
        LoadAddress = 0x20000100;
        Filename =  "cmd.line";
        Execute = no;
        Comment = "Linux Command Line...\\n\\n  ";
      }

    image
      {
        LoadAddress = 0x20780000;
        Filename = "LdLinux++";
        Execute = yes;
        Comment = "This is the tiny Linux loader!\\n  ";
      }
  }
```

Chapter 4

The LDBA protocol

The LDBA protocol is described in a separate document (work in progress).