



Virtex-II Pro PowerPC405

BootLoader Descriptions

Rev1.0

Tokyo Electron Device Limited



Table of Contents

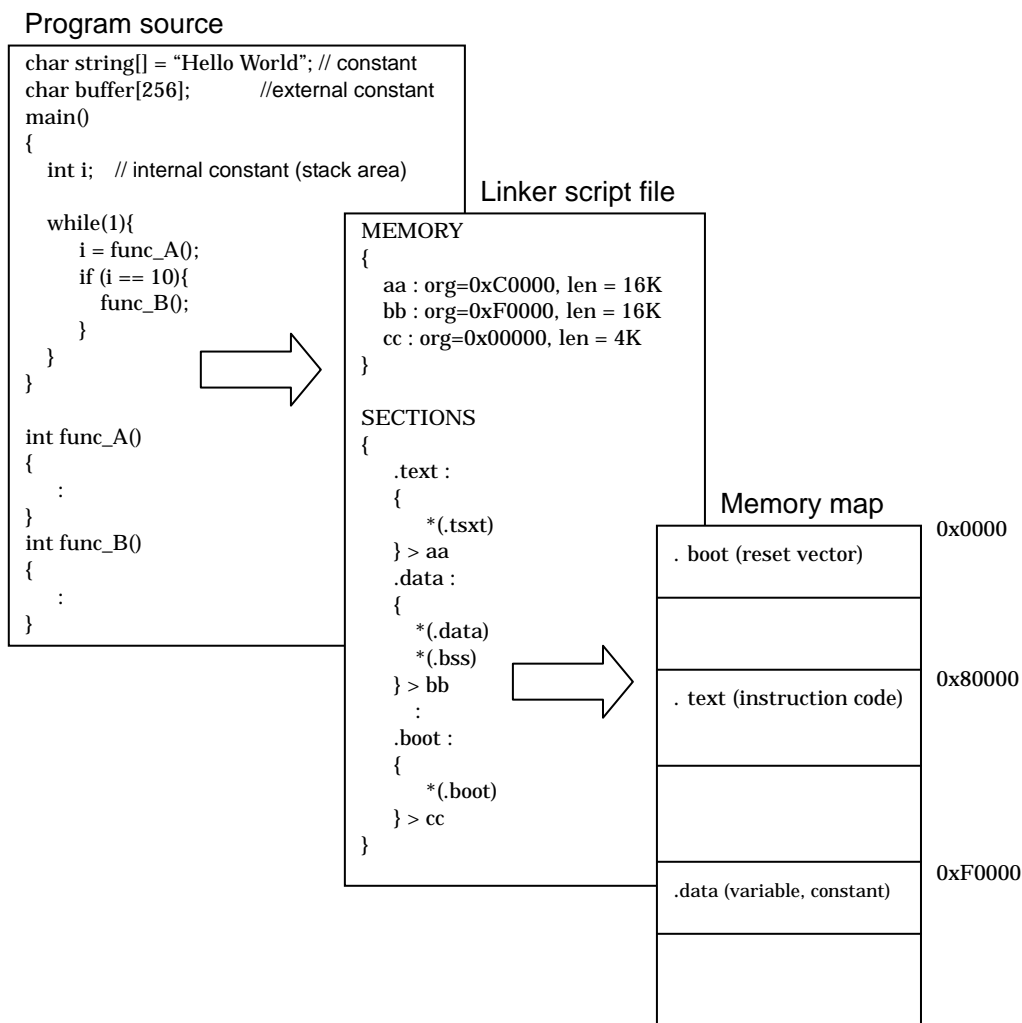
I. Overview	3
1. Program Memory Assignment.....	3
2. Structure of Executable File.....	4
3. How to Download Program onto TB-V2P-XSP-016.....	5
4. BootLoader Functions.....	6
II. elf2mcs	7
1. Specifications of the ELF File.....	7
2. MCS File.....	7
3. USAGE.....	7
4. Output File.....	7
III. BootLoader	8
1. Method of Use.....	8
2. Memory Assignment of the Boot Program and Application.....	8

I. Overview

This document describes the method of downloading the Flash Memory software with TE7720 as well as the BootLoader which, at the boot-up of PowerPC405, loads the target program from the Flash Memory and expands it inside the RAM for startup.

1. Program Memory Assignment

Instruction codes, constants, and variables of the program can be assigned to the desired addresses by means of a linker script file during the link process.



The user can also make this assignment so that the program storage area is divided into the internal RAM area and external RAM area. However, if you want store such the program in ROM, it is difficult to prepare a ROM that maintains the memory image as-is.

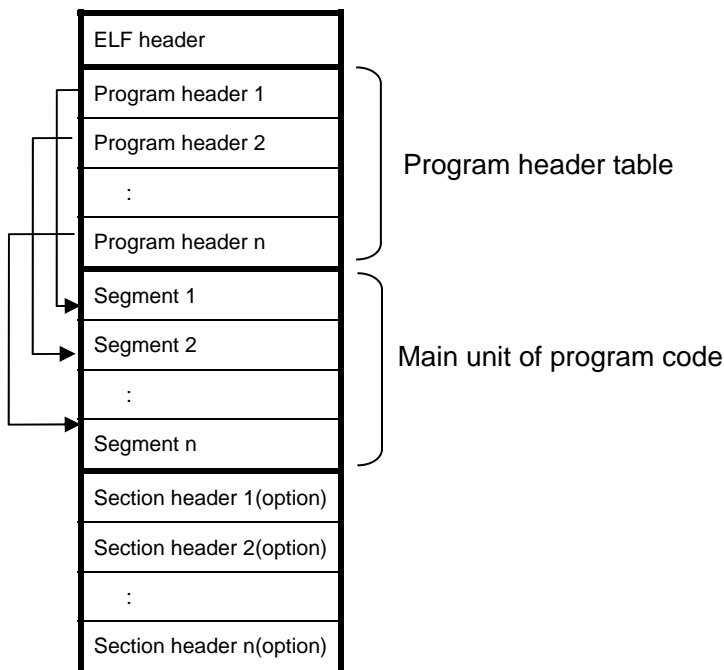
Therefore, in the case of large-size applications, the user will store the program while the address spaces are properly compressed, and then use BootLoader to expand it over a wide area of RAM.



2. Structure of Executable File

Most of the compilers for built-in software employ the executable file called ELF (*1).

The file format of ELF is shown below:



4Byte	
File ID	
File ID	
File ID	
File ID	
Type	Machine
Version	Entry
Entry	
Ph-offset	
Sh-offset	
Flags	
Eh-size	ph-entsize
Ph-num	Sh-entsize
Sh_num	Shstrndx

Structure of ELF header

4Byte	
p_type Shstrndx	
p_offset	
p_vaddr	
p_paddr	
p_filesz	
p_memsz	
p_flags	
p_align	

Structure of Program header

As described above, the ELF header refers the Program header table so that the Segments filed for each Program header can be copied into the RAM area as pointed by p_vaddr. When this process is completed, execution of the application will start with an unconditional jump to the address specified by ELF header → Entry.



3. How to Download Program onto TB-V2P-XSP-016

This section describes the procedure to write the ELF file to the Flash Memory without modifying the data structure.

- (1) Use the Win32 console program, lbplay2.exe, to write in the Flash Memory via TE7720.

```

USAGE: lbplay2 [-nv] [-nt] [-deb] [-disp] [-cf] [-sk(n)] [-chk] [-id] [-wv] [-v]
           [file1] [file2] [file3] [file4] [file5] [file6] [file7] [file8]
-nv           : no verify
-nt           : WinNT mode
-deb          : debug file output
-disp         : display more infomation
-cf           : assert XCF
-sk(n)        : skip 'n' device
-chk          : calculate check sum and save flash data
-id           : display JTAG chain
-wv           : write with verify (make ***.wbf)
-v           : verify only (use ***.wbf)
file1 - file8 : the mcs file to execute

```

File to be passed to "lbplay2.exe" must be of the MCS-format (I-HEX).

- (2) Convert the binary-formatted ELF file into the MCS format.

Conversion program: elf2mcs.exe

```

>elf2mcs executable.elf
INFO : Data encoding is 2's big endian
INFO : OS is UNIX System V
INFO : TYPE is Executable file
Type   offset  vaddr  paddr  size    msize   flag    alignment
PT_LOAD 00010000 00000000 00000000 000020C4 000020C4 00000005 00010000
PT_LOAD 00020000 FFFF0000 FFFF0000 00008E34 00008E88 00000007 00010000
PT_LOAD 00028E88 FFFF8E88 FFFF8E88 00000000 00001AD8 00000006 00010000
PT_LOAD 0002FFFC FFFFFFFC FFFFFFFC 00000004 00000004 00000005 00010000
SIZE : 196608

```

executable_0.mcs and executable_1.mcs will be created.

- (3) Writing in the Flash Memory

```
> lbplay2 executable_0.mcs executable_1.mcs
```

* For detail refer to the TB-V2P-XSP-016 Hardware User Guide.



4. BootLoader Functions

- This is a program for PowerPC405.
- This analyzes the ELF data stored in the Flash Memory, expands it in the RAM area, then unconditionally jumps to the start address.
- Since this program is supplied in a set of library functions, any of which can be called optionally in the user application.

◆ Link method ◆

Library name: libbl.a

After setting up as follows, proceed to compile → link of the program.

Set the compiler options.

Specify the relative path to Libbl.a.

For specifying Libbl.a, describe only "bl".

```
// This is all for the program !!
main()
{
    bootloader((unsigned long *)0x31000000);
}
```

Pass the leading address of the ELF data.



II. elf2mcs

1. Specifications of the ELF File

- (1) Make sure that the input file is and ELF.
- (2) Limited to ELFCLASS32. (Not compatible to ELFCLASS64(64-bit).)
- (3) Limited to executable type. Check whether ELF Header → type is EXEC (0002).
- (4) Compatible to little endians and big endians.
- (5) Creates an MCS file only from the Program Header and its Segments that are required to execute the program, and ignores the Section information.

2. MCS File

“lbplay2.exe” is made to handle such the data of a size larger than 4-Mbit and smaller than 8-Mbit. So, if the ELF data is less than 4-Mbit (less than 512 Kbytes), add the dummy data to make a total of 640 Kbytes for creating two MCS files.

3. USAGE

elf2mcs ELF_file

<< Error Message>>

<i>ERR : Can't open file [%s]</i>	: ELF file can not be opened.
<i>ERR : Can't read file header</i>	: ELF Header read error.
<i>ERR: Magic number is invalid</i>	: Not an ELF file.
<i>ERR: Not support non-CLASS32</i>	: Not the 32 bit type.
<i>ERR : [%s] is not executable file</i>	: Not the executable type.
<i>ERR : Can't read program header</i>	: Program Header read error.
<i>ERR : Can't Create MCS file[%s]</i>	: Can not create an MCS file.
<i>ERR: ELF file read</i>	: Can not read the Segment of ELF file.

4. Output File

Delete the extension of the input file, then add “_n.mcs”. [n:0-1]

Example) In the case of elf2mcs ../elfsample.elf,
the output file name will be as follows.
../elfsample_0.mcs, ../elfsample_1.mcs



III. BootLoader

1. Method of Use

This is a function specific for PowerPC405 applications.

Specification: int bootloader (unsigned long *elfaddress);

elfaddress: Top address of ELF data

Return value: At a normal termination, the program flow will not return from this function, since it branches to the application.

Error code (decimal)

101: Magic number is invalid (can not recognize it as ELF data).

102: Not support non-CLASS32

103: has not Program Header

104: Verify check error

Branching to application

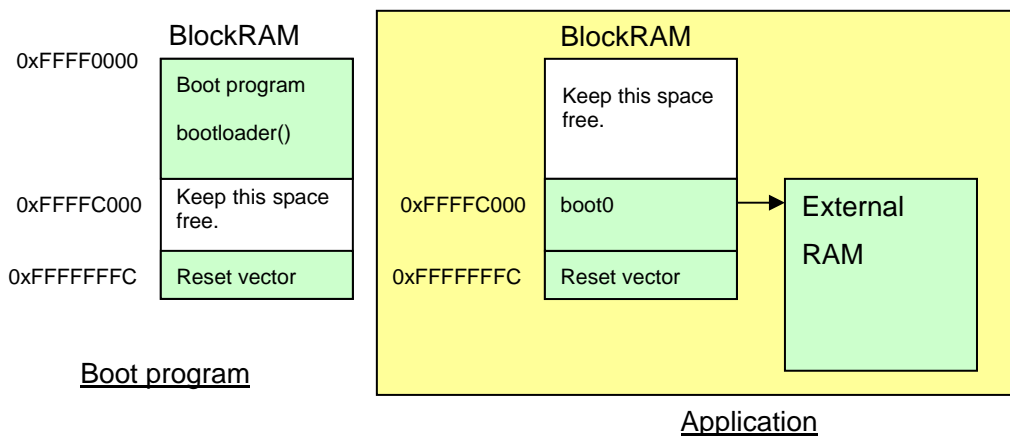
Branches to the address described in ELF Header → Entry.

When the program has been created using EDK, Entry is set to 0xFFFFFFFFC (reset vector).

2. Memory Assignment of the Boot Program and Application

Execution of the BootLoader must require its own memory space. If the address space developed by bootloader() overlaps the address space that is required for executing the bootloader itself, the bootloader's own program area will be crashed, resulting in the abnormal operation.

Therefore, exercise a sufficient care when assigning the addresses for both user application and Boot program.



Example memory assignment



Record of Revisions
Oct. 10, 2003 Rev1.0 Formal Release

Any corporation name or product name herein described is the trade mark or registered trade mark of each corporation.

Contents of this document may be subject to change without notice.

Tokyo Electron Device shall not be liable to any extra, associated or consequential damages arising either from description errors of this document or from use or abuse of this document and/or product.

Neither whole nor part of this document shall be duplicated or reproduced without permission.

Tokyo Electron Device (Ltd.)
PLD Solution Product Group