

# U-BOOT ユーザーズマニュアル

TB0226 編

Ver.1.0 2004.12.11 第1版

2006.01.30 第2版

メディアラボ株式会社

# 目次

1.はじめに.....	6
1.1 前提.....	6
1.2 本書で使用する規約.....	6
2.u-boot 詳細.....	7
2.1 u-boot について.....	7
2.2 MIPS テクノロジーのクロス開発環境.....	7
2.3 u-boot から見たメモリマップとフラッシュパーティション.....	8
2.3.1 物理空間.....	8
2.3.2 論理空間.....	8
2.3.3 DRAM エリア 0x80000000 から 0x81FFFFFF.....	8
2.3.4 フラッシュエリア 0xBF000000 から 0xBFFFFFFF.....	9
2.4 フラッシュへの書き込みとプロテクト.....	10
2.5 u-boot 用のバイナリイメージの作り方.....	10
2.5.1 mkimage のシンタックス.....	10
2.5.2 Linux カーネル、イニシャルラムディスクの作り方.....	11
2.5.3 スクリプトを作る.....	12
2.5.4 u-boot 用アプリケーションを作る.....	13
2.5.5 fw_printenv と fw_setenv.....	15
2.6 コマンドライン.....	15
2.6.1 変数.....	15
2.6.2 Hush 文法.....	16
2.6.3 クォート処理.....	16
2.6.4 改行だけの入力行の扱い.....	16
2.6.5 TAB による補完とコマンドの省略形.....	16
2.6.6 ^C による中断.....	17
2.7 シリアルの繋げ方.....	17
2.7.1 概要.....	17
2.7.2 Linux から cu を使う場合.....	17
2.7.3 C-kermit を使う場合.....	18
2.8 u-boot コマンド一覧.....	20
2.8.1 ? - 'help'の別名.....	20
2.8.2 askenv -標準入力からの入力で環境変数を設定.....	21
2.8.3 autoscr -メモリ上のスクリプトの実行.....	21

2.8.4 base - メモリ関連のコマンドのベースアドレスの設定と表示.....	22
2.8.5 bdfinfo - ボードの情報を表示.....	22
2.8.6 boot - デフォルトのブートコマンドの実行 .....	22
2.8.7 bootelf - ELF イメージの u-boot 用アプリケーションの実行.....	22
2.8.8 bootm - OS とアプリケーションの起動.....	23
2.8.9 bootp - BOOTP プロトコルで IPv4 アドレスを取得.....	23
2.8.10 bootvx - ELF イメージの vxWorks を起動.....	24
2.8.11 chpart - アクティブパーティションの変更.....	24
2.8.12 cmp - メモリの比較.....	25
2.8.13 coninfo - コンソールデバイスの表示.....	25
2.8.14 cp - メモリ間のコピー.....	25
2.8.15 dhcp - DHCP プロトコルで IPv4 アドレスを取得.....	26
2.8.16 echo - テキストを表示.....	26
2.8.17 erase - フラッシュメモリの消去.....	26
2.8.18 exit - スクリプトの終了 .....	27
2.8.19 flinfo - フラッシュの情報表示.....	27
2.8.20 fsinfo - ファイルシステム情報の表示.....	28
2.8.21 fsload - フラッシュ上のファイルシステムからファイルのロード.....	28
2.8.22 go - u-boot 用アプリケーションの実行.....	28
2.8.23 help - オンラインヘルプ .....	29
2.8.24 iminfo - アプリケーションイメージヘッダの表示.....	29
2.8.25 imls - フラッシュの中にあるイメージを探す .....	30
2.8.26 itest - 整数と文字列の比較テスト.....	30
2.8.27 loadb - シリアル経由でファイルのダウンロード(kermit モード).....	30
2.8.28 loads - シリアル経由で S レコード形式のファイルのダウンロード.....	30
2.8.29 loop - 指定した範囲のアドレスを読み続ける無限ループ.....	31
2.8.30 ls - ファイルシステムの中身を一覧表示.....	31
2.8.31 md - メモリ内容の表示.....	31
2.8.32 mm - 連続するアドレスのメモリ内容を対話的に変更.....	32
2.8.33 mtest - 簡単なメモリのテスト.....	32
2.8.34 mw - メモリ内容を指定した値で埋める.....	33
2.8.35 nfs - NFS プロトコルでファイルをダウンロード.....	33
2.8.36 nm - 同一アドレスのメモリ内容を対話的に変更.....	34
2.8.37 pci - PCI バスの一覧と、PCI コンフィグレーションスペースのアクセス.....	34

2.8.38 ping - ICMP ECHO_REQUEST パケットを指定ホストに送る.....	36
2.8.39 printenv - 環境変数の一覧と内容表示.....	36
2.8.40 protect - フラッシュメモリのプロテクトの設定.....	36
2.8.41 rarpboot- RARP プロトコルで IPv4 アドレスを取得.....	37
2.8.42 reset - CPU のリセット.....	37
2.8.43 run - 環境変数に設定されている文字列の実行.....	38
2.8.44 saveenv- 現在の環境変数の値を全てフラッシュに保存.....	38
2.8.45 setenv - 環境変数の設定と削除.....	38
2.8.46 sleep - 指定した秒数遅延させる.....	38
2.8.47 test - シェルライクな test の最小限の実装.....	38
2.8.48 tftpboot - TFTP プロトコルでファイルをダウンロード.....	38
2.8.49 version - u-boot のバージョンの表示.....	39
2.9 u-boot で特殊な意味を持つ環境変数の一覧.....	39
2.9.1 IFS.....	39
2.9.2 autoload.....	39
2.9.3 autoscript.....	40
2.9.4 autostart.....	40
2.9.5 baudrate.....	40
2.9.6 bootaddr.....	40
2.9.7 bootargs.....	40
2.9.8 bootcmd.....	41
2.9.9 bootdelay.....	41
2.9.10 bootfile.....	41
2.9.11 dnsip.....	41
2.9.12 dnsip2.....	41
2.9.13 domain.....	41
2.9.14 ethact.....	42
2.9.15 ethaddr.....	42
2.9.16 eth1addr.....	42
2.9.17 ethprime.....	42
2.9.18 fileaddr.....	42
2.9.19 filesize.....	43
2.9.20 gatewayip.....	43
2.9.21 hostname.....	43

2.9.22 ipaddr.....	43
2.9.23 loadaddr.....	43
2.9.24 loads_echo.....	43
2.9.25 netmask.....	44
2.9.26 netretry.....	44
2.9.27 nvlan.....	44
2.9.28 preboot.....	44
2.9.29 rootpath.....	44
2.9.30 serial#.....	45
2.9.31 serverip.....	45
2.9.32 stdin.....	45
2.9.33 stdout.....	45
2.9.34 stderr.....	45
2.9.35 verify.....	45
2.9.36 vlan.....	46
3.u-boot 初期設定環境変数の説明.....	47
3.1 概要.....	47
3.1.1 フラッシュ内部の/sbin/init の機能.....	47
3.2 準備.....	48
3.2.1 標準の設定.....	48
3.2.2 固定 IP で運用する場合.....	49
3.2.3 NFS ではなく、tftp を使いたい場合.....	49
3.3 詳細.....	50
3.3.1 カーネルコマンドラインの設定.....	50
3.3.2 linux を起動する.....	50
3.3.3 フラッシュのアップデート.....	51
3.3.4 消去.....	52
3.3.5 環境変数詳細.....	52
3.4 具体的な起動例.....	52
3.4.1 詳細.....	53

# 1. はじめに

---

## 1.1 前提

このマニュアルは、TB0226 での動作についてのみ記したものです。

また、弊社で選択したコンパイルオプションの場合での動作についてのみ詳細に記したものです。コンパイルオプションを変えると、u-boot で使えるコマンドが増減するだけでなく、文法も変わる場合がありますので、ご注意ください。

このマニュアルは、弊社のクロス開発環境のマニュアルから、u-boot に関する部分だけを抜きだしたものです。

クロス開発環境:

Pentium または Pentium 互換以上の CPU を搭載した AT 互換 CPU ボード

ご注意

UNIX は The Open Group の登録商標です。

Linux は、Linus Torvalds の米国およびその他の国における登録商標または商標です。

Windows は米国 Microsoft Corporation の登録商標です。

その他記載されている会社名・製品名等は、各社の登録商標もしくは商標、または弊社の商標です。

本マニュアルおよび本製品の内容は予告なく変更する場合があります。

・免責について

本製品を運用した結果の影響に関して、弊社は一切責任を負いかねますので、ご了承ください。

Copyright 2004-2006 Media Lab. Inc., All Rights Reserved.

## 1.2 本書で使用する規約

・コマンド入力

コンソールや端末エミュレータでのコマンド入力は

```
# ls -l
```

等のように表記します。#は入力のプロンプト(ユーザーの入力を促す記号でその後にコマンド(命令)を入力する)です。入力する部分はボールドになっています。

## 2. u-boot 詳細

---

### 2.1 u-boot について

u-boot のバージョンは TB0226 では、u-boot 1.1.2 と表示されますが、開発時には、まだ 1.1.2 はリリースされておらず、正確には u-boot 1.1.1+CVS(2004/12/1)を TB0226 用に修正したものとお考え下さい。

安定版ではなく CVS 版を利用している理由は、開発元にマージしてもらいたいと考えているからですが、現在の u-boot の mips 版は、big-endian を前提に Makefile 等が作られていて、まだしばらくの間は、ソースツリーに TB0226 用を含めてもらうには時間がかかるでしょう。

u-boot の開発は以下で行なわれています。

<http://sourceforge.net/projects/u-boot/>

また、u-boot に関する詳細なドキュメントは、英語ですが、

<http://www.denx.de/twiki/bin/view/DULG/WebIndex>

からたどれますので、参照してみてください。

1.1.1 からの変更は TB0226 のサポートと、ちょっとしたバグフィックスが主です。

u-boot の主なコンフィグレーションは以下の通りです

```
#define CONFIG_COMMANDS          \
    (CONFIG_CMD_DFL|CFG_CMD_ASKENV|CFG_CMD_JFFS2|\
    CFG_CMD_DHCP|CFG_CMD_PING|CFG_CMD_PCII  \
    CFG_CMD_ELF)
```

TB0226 への移植はメディアラボ( <http://www.mlb.co.jp> )が行ないました。

お問い合わせ、バグレポート等は、[info@mlb.co.jp](mailto:info@mlb.co.jp) 宛に送って下さい。

また、ソースは、

<http://www.mlb.co.jp/u-boot/>

からたどれます。

### 2.2 MIPS テクノロジーのクロス開発環境

u-boot のコンパイルは、MIPS テクノロジーが配布しているクロス開発環境が必要です。これをインストールして下さい。

```
# rpm -Uvh rpms/sdelinux-5.03.06-1.i386.rpm
```

このバイナリについては、MIPS-HOWTO に説明があります。

<http://www.linux.or.jp/JF/JFdocs/MIPS-HOWTO-5.html>

また、最新版に関する情報などが早いのは以下です。

<http://www.linux-mips.org/toolchain.html>

実際の配布元は以下です。

<ftp://ftp.mips.com/pub/tools/software/sde-for-linux/>

## 2.3 u-boot から見たメモリマップとフラッシュパーティション

TB0226 は、32MB の RAM、16MB のフラッシュメモリを実装しています。

### 2.3.1 物理空間

開始	終了	用途
00000000	01FFFFFF	SDRAM(バンク 0)
0F000000	0FFFFFFF	内部 I/O 領域
10000000	15FFFFFF	PCI メモリ空間(10000000 – 15FFFFFF に変換される)
16000000	17FFFFFF	PCI I/O 空間(00000000 – 1FFFFFFF に変換される)
1F000000	1FFFFFFF	フラッシュメモリ(バンク 1)

### 2.3.2 論理空間

MIPS の CPU は論理アドレスの上位 3 ビットを使って論理空間を使い分けます。

開始	終了	名称	MMU	キャッシュ
00000000	7FFFFFFF	kuseg (ユーザモード空間)	ON	ON
80000000	9FFFFFFF	kseg0	OFF	ON
A0000000	BFFFFFFF	kseg1	OFF	OFF
C0000000	FFFFFFF	kseg2	ON	ON

kseg0 と kseg1 はどちらも、上位 3 ビットを取り除いた、物理空間へ変換されますので、

U-Boot では、kseg0 もしくは、kseg1 の空間を使って物理空間を利用します。

### 2.3.3 DRAM エリア 0x80000000 から 0x81FFFFFF

開始	終了	用途
80000000		空き
	81F7FFFF	u-boot スタック
81F80000	81F9BFFF	u-boot データ(カーネルパラメータ領域 128K 含む)
81F9C000	81FBFFFF	u-boot malloc 領域(144k)
81FC0000	81FFFFFF	u-boot 本体

u-boot が使っているアドレスの開始、終了は、だいたいその辺という意味で、きっちりそのアドレ

スから始まるわけではなく、メモリの 82000000 から必要な領域を順に確保していきます。

### 2.3.4 フラッシュエリア 0xBF000000 から 0xBFFFFFFF

フラッシュエリアは以下の様に分けています。

u-boot でのパーティション番号	Linux の mtdblock 番号と名称	用途	開始アドレス	サイズ
0	1 root	root ファイルシステム	BF000000	12M bytes
-	2 uboot	u-boot 本体	BFC00000	256K bytes
1	3 opt	ユーザエリア	BFC40000	3712K bytes
2	4 conf	システム設定保存領域	BFFE0000	112K bytes
-	5 ubootenv	u-boot 環境変数保存領域	BFFFC000	16K bytes

この様にパーティションを分けた理由:

フラッシュの中味を消す際には、イレースブロック単位で消す必要があります。

本機に搭載されているフラッシュは、先頭と末尾の 128kbyte はイレースブロックサイズが 16k で残りの部分は 128kbyte です。

- u-boot 本体の位置は、CPU リセット時に開始するアドレスになければいけないので、この位置からは動かさません。
- 一番大きく取れる所を root ファイルシステムとしました。
- u-boot の環境変数保存エリアは、u-boot 本体の末尾の余っている部分にも置けるのですが、書き変えに失敗すると、二度と起動しなくなってしまう可能性があるため、別の領域に分ける事とし、一番小さい末尾から 1 ブロック取りました。
- Linux の MTD ドライバは、異なるイレースブロックサイズが混在しても構いませんが、最大のイレースブロックで割り切れるサイズでないとパーティションを作れませんので、残った部分を二つに分けました。

u-boot でのパーティション番号は、ls や fsload コマンドだけが参照します。パーティションのサイズや位置は、u-boot のコンパイル時の設定ですが、u-boot をコンパイルし直さなくとも、気を付けて使えば変更しても構いません。例えば、現在は root ファイルシステム内にある uImage という名前のファイルをロードしてカーネルを起動する様にデフォルトで設定していますが、システム設定保存領域に直接カーネルを置いても困るわけではありません。

u-boot は、cramfs と jffs2 を自動認識しますので、どちらのフォーマットでも構いませんが、どちらのファイルシステムも、リードオンリーです。書き変えるには、全体を入れ換えるか、Linux から書きこむ必要があります。

また、パーティション範囲をイレースすると、JFFS2 で新規にファイルシステムを作成したのと同等の効果が得られます。

## 2.4 フラッシュへの書き込みとプロテクト

フラッシュへ書き込む為の特殊なコマンドが u-boot にあるわけではありません。メモリ間のコピーや、ファイルをロードするコマンド等一部のコマンドがフラッシュへの書き込みをサポートしています。loadb loads nfs tftpboot のコマンドを使うときにロードアドレスをフラッシュの領域にするか、一旦メモリにロードしてから、cp コマンドで書きこみます。

ただし、フラッシュの領域に対する書き込みは、書きこむ前に書きこもうとするアドレス範囲がイレース済でない、その範囲は書きこみは行なわれず、スキップされます。

このチェックは、u-boot が独自にやっていますので、イレース実行後でも、再起動してしまうと、再度イレースする必要があります。

同様に、イレースを実行するには、その前にプロテクトを外さなくてははいけません、このプロテクト範囲も u-boot 自身で管理しています。(フラッシュチップのプロテクト機能は利用していません。)

先にイレースしなければならないので、一旦 RAM 上にダウンロードしてから cp コマンドで書きこむ様にすることをお勧めします。

## 2.5 u-boot 用のバイナリイメージの作り方

u-boot で実行するファイルは、mkimage というコマンドで u-boot 用のフォーマットに変換する必要があります。

mkimage は u-boot のソースの tools の下にあります。

mkimage の利点は、圧縮をサポートしている点や、チェックサムの確認、ファイルタイプを指定できるので、間違いが起きにくい等です。

### 2.5.1 mkimage のシンタックス

```
mkimage -l image
```

image: mkimage で作られたファイル

image の中のヘッダー情報を表示します。

```
mkimage [-x] -A arch -O os -T type -C comp -a addr -e ep -n name -d data_file[:data_file...]
```

image

arch: ターゲット CPU を指定します。以下のどれかです。

alpha | arm | x86 | ia64 | m68k | microblaze | mips | mips64 | ppc | s390 |  
sh | sparc | sparc64

os: OS を指定します。以下のどれかです。

4\_4bsd | artos | dell | esix | freebsd | irix | linux | lynxos | ncr | netbsd |  
openbsd | psos | qnx | rtems | sco | solaris | svr4 | u-boot | vxworks

u-boot の bootm コマンドは、このタイプよって起動方法を切替えます。

type: タイプを指定します。  
 filesystem | firmware | kernel | multi | ramdisk | script | standalone  
 u-boot の bootm コマンドや, autoscr コマンドはこのタイプをチェックします。

comp: data\_file がどの様に圧縮されているかを指定します。以下のどれかです。  
 none | bzip2 | gzip  
 mkimage が、このオプションに従って圧縮する訳ではありません。u-boot の bootm コマンドはこの圧縮方法をチェックして、自動的に展開してくれます。

addr: ロードアドレスを指定します。

ep: 実行を開始するアドレスを指定します。

name: 名称を指定します(メモです)。

datafile: 変換前のファイル

-x: XIP( execute in place)フラグを設定します。  
 例えば、フラッシュ上でそのまま動かすプログラム等で指定します。

## 2.5.2 Linux カーネル、イニシャルラムディスクの作り方

カーネルや、イニシャルラムディスクを普通に作成したあと、以下の様にします。

カーネルを作るには、

```
# mipsel-linux-objcopy -O binary -R .note -R .comment -S vmlinux \
linux.bin
# gzip -9 linux.bin
# mkimage -A mips -O linux -T kernel -C gzip -a 0x80000000 \
-e `sed -n '/ kernel_entry$/ s/\(^[\ ]*\) .*/\1/p' System.map` \
-n "Linux Kernel Image" -d linux.bin.gz uImage
Image Name:   Linux Kernel Image
Created:      Fri Dec  3 18:57:20 2004
Image Type:   MIPS Linux Kernel Image (gzip compressed)
Data Size:    764307 Bytes = 746.39 kB = 0.73 MB
Load Address: 0x80000000
Entry Point:  0x8019A040
```

mips 用カーネルのエントリーポイントは、コンパイル毎にかわりますので、System.map から取得する必要があります。カーネルの CONFIG\_EMBEDDED\_RAMDISK を利用する場合は、上記だけで、ラムディスクも使えます。

u-boot でイニシャルラムディスクを別途読み込む場合には、kernel 側にも修正が必要です (CONFIG\_EMBEDDED\_RAMDISK を使わないでラムディスクを読ませたい場合)。

arch/mips/kernel/setup.c の

```
init_arch(int argc, char **argv, char **envp, int *prom_vec)
```

の envp に、u-boot が自動生成した変数が渡ります。

具体的には、

```
memsize=32
initrd_start=0xA0500040
initrd_size=0x30FF0
flash_start=0xBF000000
flash_size=0x1000000
```

という様な文字列が NULL ターミネートされ(終わりの印は、2つの NULL)で渡ります。  
(Linux のソースにマージしてもらおうつもりですが、Linux 開発スピードに追いつけず、マージには時間はかかりそうです。)

対応済のカーネルを使う場合には、以下のコマンドで u-boot 用のラムディスクイメージが作成できます。

例えば、ext2 で作ったラムディスクイメージが rammin というファイルであるとして、

```
# gzip -9 rammin
# u-boot/tools/mkimage -A mips -O linux -T ramdisk -C gzip \
-n 'Ramdisk Image' -a 0x80200000 -d rammin.gz ramImage
Image Name:   Ramdisk Image
Created:      Fri Dec  3 19:29:29 2004
Image Type:   MIPS Linux RAMDisk Image (gzip compressed)
Data Size:    198354 Bytes = 193.71 kB = 0.19 MB
Load Address: 0x80200000
Entry Point:  0x80200000
#
```

これで ramImage が出来上がります。

### 2.5.3 スクリプトを作る

u-boot 用のスクリプトも、mkimage で作ることができます。マシンがたくさんあっても、ネットワーク経由でスクリプトをダウンロードする事で、フラッシュのアップデートなどが簡単に出来ます。

```
# echo "echo hello" > script
# u-boot/tools/mkimage -A mips -O u-boot -T script -C none -n
"hello" -a 0x80200000 -d script script.bin
Image Name:   hello
Created:      Sat Dec  4 16:56:14 2004
Image Type:   MIPS U-Boot Script (uncompressed)
Data Size:    19 Bytes = 0.02 kB = 0.00 MB
Load Address: 0x80200000
Entry Point:  0x80200000
Contents:
  Image 0:    11 Bytes =  0 kB = 0 MB
#
```

ここで出来た script.bin を u-boot にダウンロードして、autoscr コマンドを使えば実行できます。

## 2.5.4 u-boot 用アプリケーションを作る

u-boot の内部関数を使ったアプリケーションも作成できます。

u-boot に付いて来たサンプル `hello_world` をイメージにするには、

```
# u-boot/tools/mkimage -A mips -O u-boot -T standalone -C none -n
"hello" -a 0x80200000 -d u-boot/examples/hello_world.bin hello.img
Image Name:   hello
Created:      Sat Dec  4 22:06:09 2004
Image Type:   MIPS U-Boot Standalone Program (uncompressed)
Data Size:    920 Bytes = 0.90 kB = 0.00 MB
Load Address: 0x80200000
Entry Point: 0x80200000
#
```

mkimage で作ったものは、bootm で実行できます。

```
# nfs 80400000 192.168.3.91:/usr/src/mltbox/hello.img
Using i82559#0 device
File transfer via NFS from server 192.168.3.91; our IP address is
192.168.3.202
Filename '/usr/src/mltbox/hello.img'.
Load address: 0x80400000
Loading: #
done
Bytes transferred = 984 (3d8 hex)
# bootm
## Booting image at 80400000 ...
   Image Name:   hello
   Created:      2004-12-04 13:06:09 UTC
   Image Type:   MIPS U-Boot Standalone Program (uncompressed)
   Data Size:    920 Bytes = 0.9 kB
   Load Address: 80200000
   Entry Point:  80200000
OK
Example expects ABI version 2
Actual U-Boot ABI version 2
Hello World
argc = 0
argv[0] = "<NULL>"
Hit any key to exit ...

#
```

u-boot の内部関数は、ジャンプテーブルを介して呼び出されます。

アプリケーションは、`app_startup` を呼び出すことで、u-boot の内部関数が使える様になります。

す。また、DECLARE\_GLOBAL\_DATA\_PTRとする事で、u-bootのグローバルデータに、変数gdを介してアクセス出来ます。また、ジャンプテーブルを書き変えると、u-bootの動作を変える事もできます。

```
int hello_world (int argc, char *argv[])
{
    DECLARE_GLOBAL_DATA_PTR;
    app_startup(argv);
    ...
}
```

使える関数は、

void app_startup(char **);	
unsigned long get_version(void);	ABIバージョン2が返ります
int getc(void);	標準入力から一文字取得(ブロックします)
int tputc(void);	標準入力に文字が来ているか調べる
void putchar(const char);	一文字標準出力に送る
void puts(const char*);	文字列を標準出力に送る
void printf(const char* fmt, ...);	標準出力に、整形した文字列を出力
void install_hdlr(int, interrupt_handler_t*, void*);	割り込みハンドラの登録
void free_hdlr(int);	割り込みハンドラの削除
void *malloc(size_t);	メモリの取得
void free(void*);	メモリの解放
void udelay(unsigned long);	マイクロ秒単位でのウエイト
unsigned long get_timer(unsigned long);	現在のtick値を取得
void vprintf(const char *, va_list);	標準出力に、整形した文字列を出力
void do_reset (void);	再起動する

グローバルデータは、以下の構造です。

```
typedef struct bd_info {
    int          bi_baudrate;      /* serial console baudrate */
    unsigned long bi_ip_addr;      /* IP Address */
    unsigned char bi_enetaddr[6];  /* Ethernet address */
    unsigned long bi_arch_number;  /* unique id for this board */
    unsigned long bi_boot_params;  /* where this board expects params */
    unsigned long bi_memstart;    /* start of DRAM memory */
    unsigned long bi_memsizes;    /* size of DRAM memory in bytes */
};
```

```

        unsigned long  bi_flashstart;      /* start of FLASH memory */
        unsigned long  bi_flashsize;      /* size of FLASH memory */
        unsigned long  bi_flashoffset;    /* reserved area for startup monitor */
    } bd_t;
typedef struct global_data {
    bd_t      *bd;
    unsigned long  flags;
    unsigned long  baudrate;
    unsigned long  have_console;          /* serial_init() was called */
    unsigned long  ram_size;              /* RAM size */
    unsigned long  reloc_off;             /* Relocation Offset */
    unsigned long  env_addr;              /* Address of Environment struct */
    unsigned long  env_valid;             /* Checksum of Environment valid? */
    void    **jt;                          /* jump table */
} gd_t;
gd_t *gd;

```

### 2.5.5 fw\_printenv と fw\_setenv

Linux から u-boot の環境変数エリアを書き替えることも出来ます。u-boot/tools/env/fw\_env.c をコンパイルすれば、u-boot の printenv setenv と同じ使い方で、利用できます。

## 2.6 コマンドライン

u-boot のコマンドラインは、BusyBox の Hush を u-boot 用にしたものです。スクリプトを作成する事もできます。簡単なスクリプトは環境変数に設定して保存しておき、設定した文字列を run コマンドで実行します。

### 2.6.1 変数

シェル変数と環境変数があります。

環境変数は、saveenv コマンドでフラッシュに保存可能ですが、シェル変数は保存されません。

環境変数の設定は setenv コマンドを使います。

```
setenv name value
```

シェル変数への設定は、

```
name=value
```

とします。

変数を参照するには、\${name} もしくは、\$name とします。

環境変数は、シェル変数より優先され、同じ名称のシェル変数は使えなくなります。環境変数

に設定したコマンドは run コマンドで実行できますが、シェル変数は run コマンドに渡せません。環境変数には u-boot のコマンドが利用する予約された変数がたくさんあります。シェル変数 \$? には、最後に実行したコマンドの終了ステータスが入ります。

## 2.6.2 Hush 文法

リスト: コマンドを ;、&&、|| のどれかで区切って並べたもの

リスト自体の終了ステータスは最後に実行したコマンドの結果になります。

コマンド 1 ; コマンド 2

コマンド 1 を実行してからコマンド 2 を実行します。

コマンド 1 && コマンド 2

コマンド 1 の終了ステータスが 0 の場合のみコマンド 2 が実行されます。

コマンド 1 || コマンド 2

コマンド 1 の終了ステータスが 0 以外の場合のみコマンド 2 が実行されます。

for name in 単語のリスト; do リスト; done

単語のリスト一つずつについて、その単語をシェル変数 name に設定してからリストを実行します。

if リスト; then リスト; [ elif リスト; then リスト; ] ... [ else リスト; ] fi

if のリストと elif のリスト部を順番に実行し、終了ステータスが 0 のリストに出会ったら、対応する then のリスト部を実行して終了します。出会わなかった場合には else 部のリストが実行されます。

while リスト; do リスト; done

while のリスト部が終了ステータス 0 以外を返すまで、do のリストを繰り返し実行します。

until リスト; do リスト; done

until のリスト部が終了ステータス 0 を返すまで、do のリストを繰り返し実行します。

## 2.6.3 クォート処理

\$ や ; をエスケープするには、\ を使います。

" で括った文字列内に \${name} 等変数の参照がある場合、変数は展開されますが、' で括った場合、展開されずにそのままの文字列になります。

## 2.6.4 改行だけの入力行の扱い

改行だけの入力は、最後のコマンドの再実行です。注意して下さい。

## 2.6.5 TAB による補完とコマンドの省略形

コマンドの最初の数文字入力したあと、TAB キーを入力する事により、補完可能な範囲で補

完され、候補が複数あれば、候補の一覧が表示されます。  
コマンドは最初の数文字だけでも、それがどのコマンドであるか判別可能な範囲であれば、残りは省略できます。

例えば、tftpboot コマンドは、tftp や tf でも tftpboot と解釈されます。

## 2.6.6 ^Cによる中断

Ctrl-C で処理の中断が出来ますが、コマンドによっては中断できないので、中断されるまでに時間がかかります。例えば、erase コマンドは、処理中には止められません。

## 2.7 シリアルの繋げ方

### 2.7.1 概要

初期設定は

ボーレート	:115200
データ長	:8bit
パリティ	:なし
ストップビット	:1
フロー制御	:なし

となっています。

パソコンと接続するにはクロスケーブルが必要です。

### 2.7.2 Linux から cu を使う場合

cu コマンドは uucp パッケージに含まれています。

以下は ttyS0 にケーブルを接続したと仮定して説明します。

cu は普通 suid/sgid されていて、ルートユーザで起動したとしても、uucp というユーザ ID/グループ ID で実行されますので、uucp ユーザが /dev/ttyS0 に対して読み書き出来ないと実行できません。また、cu は、/var/lock のディレクトリにロックファイルを作成しますので、ここにも読み書き出来る権限が必要です。

uucp の流儀では普通、/dev/ttyS0 のオーナーグループを uucp にして、グループのパーミッションを rw に設定します。

例えば以下の設定が普通です。

```
$ ls -l /dev/ttyS0
crw-rw---- 1 root uucp 4, 64 12月 7 06:29 /dev/ttyS0
$
```

グループのパーミッションとオーナーグループの設定が上記と同一であることを確認して下さい。

なっていないければ、ルートユーザで、

```
# chgrp uucp /dev/ttyS0
# chmod g+rw /dev/ttyS0
```

として下さい。

cu が正しく使える様になれば、以下のコマンドで接続できます。

```
# cu -s 115200 -nostop -l ttyS0
```

Sレコードファイルをダウンロードするには、以下の様にします。

```
# setenv loads_echo 1
# loads 80400000
## Ready for S-Record download ...
~> uImage.srec
1 2 3 4 5 6 7 8 9 10
--省略--
47772 47773 47774 47775 47776
[file transfer complete]
[connected]

## First Load Addr = 0x80400000
## Last Load Addr = 0x804BA9D2
## Total Size      = 0x000BA9D3 = 764371 Bytes
## Start Addr     = 0x00000000
#
```

cu を終了するには、~. を入力します。~. は改行を入力した直後のタイミングでないと、エスケープシーケンスとしては認識されません。また、バッファをフラッシュしてから終了するため、終了するのに時間がかかる場合があります。

### 2.7.3 C-kernel を使う場合

kermit は ckermit パッケージに含まれています。

以下は ttyS0 にケーブルを接続したと仮定して説明します。

デスクトップ環境によっては、kermit 標準のエスケープキャラクタ(接続先へのコマンド入力から、kermit 自体へのコマンド入力モードへ移行する文字) Ctrl-\ がアプリケーションに渡らない場合があります。Ctrl-\ がうまく働かない場合は、エスケープキャラクタを変更するか、別のターミナルエミュレータを使って見て下さい。以下では、エスケープキャラクタを Esc に変更しています。

ファイルをダウンロードするには、以下の様にします。

```
# kermit
C-Kermit 8.0.209, 17 Mar 2003, for Red Hat Linux 8.0
Copyright (C) 1985, 2003,
Trustees of Columbia University in the City of New York.
Type ? or HELP for help.
(/usr/src/mltbox/) C-Kermit>set line /dev/ttyS0
(/usr/src/mltbox/) C-Kermit>set flow auto
(/usr/src/mltbox/) C-Kermit>set speed 115200
/dev/ttyS0, 115200 bps
```

```
(/usr/src/mlinbox/) C-Kermit>set serial 8n1
(/usr/src/mlinbox/) C-Kermit>set carrier-watch off
(/usr/src/mlinbox/) C-Kermit>set escape 27
(/usr/src/mlinbox/) C-Kermit>set prefixing all
(/usr/src/mlinbox/) C-Kermit>connect
Connecting to /dev/ttyS0, speed 115200
Escape character: Ctrl-[ (ASCII 27, ESC): enabled
Type the escape character followed by C to get back,
or followed by ? to see other options.
```

-----

```
# loadb
```

```
## Ready for binary (kermit) download to 0x80400000 at 115200
bps...
```

<=ここで、Esc キーを押してCを入力します

```
(Back at titan.mlb.co.jp)
```

-----

```
(/usr/src/mlinbox/) C-Kermit>send uImage
```

ファイル転送中は以下の様な画面になり、進捗状況がみえます。

```
C-Kermit 8.0.209, 17 Mar 2003, titan.mlb.co.jp [192.168.3.91]
```

```
Current Directory: /usr/src/mlinbox
```

```
Communication Device: /dev/ttyS0
```

```
Communication Speed: 115200
```

```
Parity: none
```

```
RTT/Timeout: 01 / 03
```

```
SENDING: uImage => UIMAGE
```

```
File Type: BINARY
```

```
File Size: 764371
```

```
Percent Done: 6   ///
```

```
...
```

```
10...20...30...40...50...60...70...80...90..100
```

```
Estimated Time Left: 00:01:16
```

```
Transfer Rate, CPS: 9362
```

```
Window Slots: 1 of 1
```

```
Packet Type: D
```

```
Packet Count: 11
```

```
Packet Length: 9024
```

```
Error Count: 0
```

```
Last Error:
```

```
Last Message:
```

X to cancel file, Z to cancel group, <CR> to resend last packet,  
E to send Error packet, ^C to quit immediately, ^L to refresh

screen.

ファイル転送が終了すると、コマンドラインに戻りますので、connect で再接続します。

```
(/usr/src/mltbox/) C-Kermit>connect
Connecting to /dev/ttyS0, speed 115200
Escape character: Ctrl-[ (ASCII 27, ESC): enabled
Type the escape character followed by C to get back,
or followed by ? to see other options.
```

```
-----
## Total Size      = 0x000ba9d3 = 764371 Bytes
## Start Addr     = 0x80400000
```

終了するには、

```
#                               <=ここで、Esc キーを押して C を入力します
(Back at titan.mlb.co.jp)
```

```
-----
(/usr/src/mltbox/) C-Kermit>q
Closing /dev/ttyS0...OK
#
```

~/kermrc を以下の様に設定しておく、kermrc を起動したときに、すぐに接続することができます。

```
# cat ~/.kermrc
set line /dev/ttyS0
set flow auto
set speed 115200
set serial 8n1
set carrier-watch off
set escape 27
set prefixing all
connect
#
```

## 2.8 u-boot コマンド一覧

u-boot での数値の入力は基本的に 16 進数として解釈されます。

引数省略時のデフォルトや、省略可能な引数が複数ある場合に一つだけ引数を与えた場合の動作(解釈方法)はコマンドによって異なりますし、コンパイル時の機能の選択(特に、CFG\_HUSH\_PARSER)によって、書き方が変わったりしますので、出荷時の状態では、こういう使い方になるはずであるという説明です。また、全てのコマンドの確認をしておりませんが、未確認な部分は、未確認と明記してあります。

### 2.8.1 ? - 'help'の別名

文法: ? [command...]

command:コマンド名

command が省略された場合、全てのコマンドとその概略の一覧を表示します。

command が指定された場合、そのコマンドの説明と使い方を表示します。

関連: help

## 2.8.2 askenv -標準入力からの入力で環境変数を設定

文法: askenv name [size]

askenv name [message] size

name: 環境変数の名称

size: 環境変数を読みこむ際の最大文字数の指定(10進数で指定します)

1以上255以下が可能です。

ユーザは255文字まで入力可能ですが、入力された文字列は指定したサイズに切り詰められます。

message: 入力をうながすプロンプト文字列を指定します。

環境変数を設定するには、setenv もありますが、askenv の利点は、特殊な文字列をクオートする必要がなく、入力した文字列がそのまま設定できる所にあります。

関連: setenv printenv

## 2.8.3 autoscr -メモリ上のスクリプトの実行

文法: autoscr [addr]

addr: スクリプトの置いてあるアドレスです。省略した場合は、80400000です。スクリプトは、mkimage で作成されたものでなくてはなりません。

環境変数 verify が n で始まらない場合、チェックサムを検査し、正しいときのみ実行します。(デフォルトでは verify=n に設定しています。)

```
# iminfo 80400000
```

```
## Checking Image at 80400000 ...
```

```
Image Name: hello
```

```
Created: 2004-12-04 9:24:20 UTC
```

```
Image Type: MIPS U-Boot Script (uncompressed)
```

```
Data Size: 24 Bytes = 0 kB
```

```
Load Address: 80200000
```

```
Entry Point: 80200000
```

```
Verifying Checksum ... OK
```

```
# autoscr
```

```
## Executing script at 80400000
```

```
hello
```

```
#
```

関連: run

#### 2.8.4 base - メモリ関連のコマンドのベースアドレスの設定と表示

文法: base [offset]

offset: 設定したいオフセット

offset が省略されると、現在の設定値を表示します。

md mm nm mw cmp cp crc32 コマンドの引数にアドレスを指定したときに、base コマンドで設定された値が自動的に加算されます。

再起動すると0に初期化されます。

デフォルトで環境変数に設定してあるスクリプトは、base コマンドで値を設定すると、動かなくなりますので、気を付けて下さい。

関連: md mm nm mw cmp cp crc32

#### 2.8.5 bdfinfo - ボードの情報を表示

文法: bdfinfo

RAM やフラッシュのアドレス、サイズなどのボードに関する情報を表示します。

#### 2.8.6 boot - デフォルトのブートコマンドの実行

文法: boot

bootd

環境変数 bootcmd は、電源投入時に自動で実行すべきコマンドが保存されています。

boot コマンドはこれを実行するコマンドです。

bootd というコマンドもありますが、bootd は過去互換の為の名前であり、boot とまったく同じコマンドです。

#### 2.8.7 bootelf - ELF イメージの u-boot 用アプリケーションの実行

文法: bootelf [addr]

addr: ELF ファイルが置いてあるアドレス。

省略した場合は、最後にファイルをメモリ上にロードしたアドレス

例:

```
# nfs \"192.168.3.91:/usr/src/mlinbox/u-boot/examples/hello_world\"
Using i82559#0 device
File transfer via NFS from server 192.168.3.91; our IP address is
192.168.3.202
Filename '/usr/src/mlinbox/u-boot/examples/hello_world'.
Load address: 0x80400000
Loading: ##
```

```

done
Bytes transferred = 6798 (1a8e hex)
# bootelf 80400000
Loading .text @ 0x80200000 (672 bytes)
Loading .rodata @ 0x802002a0 (160 bytes)
Loading .reginfo @ 0x80200340 (24 bytes)
Loading .got @ 0x80200360 (56 bytes)
## Starting application at 0x80200000 ...
Example expects ABI version 2
Actual U-Boot ABI version 2
Hello World
argc = 1
argv[0] = "80400000"
argv[1] = "<NULL>"
Hit any key to exit ...

## Application terminated, rc = 0x0
#
    関連: bootm go

```

### 2.8.8 bootm - OS とアプリケーションの起動

文法: bootm [addr [initrdaddr]]

addr: カーネルが置いてあるアドレスです。

省略した場合は、最後にファイルをメモリ上にロードしたアドレスになります。

initrdaddr: イニシャルラムディスクが置いてあるアドレスです。

省略した場合は、イニシャルラムディスクに関するオプションはカーネルに渡されません。Linux 以外の OS を起動する際は、initrdaddr を指定してはいけません。

mkimage で作られたファイルには、OS の種類等が埋め込まれています。その種類に応じたアプリケーションの起動を行います。

環境変数 verify が n で始まらない場合、チェックサムを検査し、正しいときのみ実行します。(デフォルトでは verify=n に設定しています。)

イメージタイプが Linux カーネルイメージの場合、カーネル引数は環境変数 bootargs に設定しておく、自動的に渡されます。

Linux カーネルと u-boot 用アプリケーション以外での動作は未確認です。

関連: bootelf bootvx iminfo

### 2.8.9 bootp - BOOTP プロトコルで IPv4 アドレスを取得

文法: bootp [addr] [filename]

addr: ロードするアドレスです。

省略した場合、環境変数 loadaddr の値、loadaddr も設定されていなければ、80400000 です。

filename: ロードするファイルです。

省略した場合は、bootp サーバの応答によって指定されたもの。サーバから指定も無かった場合は、環境変数 bootfile の値が使われます。

addr を省略して、filename を指定する場合、filename は " で括られていなくてははいけません、パーサが " をはずしてしまうため、bootp コマンドに " が渡るように、エスケープする必要があります。

最初に Bootp プロトコルで IP アドレス、サーバアドレス、ダウンロードするファイル等を取得し、以下の環境変数を更新します。

serverip: bootp サーバの IP アドレス

ipaddr: 割り当てられた IP アドレス

bootfile: 読みこむべきファイル

(filename が指定されている場合、引数で渡したものがコピーされます)

gatewayip, netmask, hostname, rootpath, dnsip, dnsip2, domain:

サーバの応答で指定があった場合に更新され、取得できなかったものは以前の値が保持されます。

次に、環境変数の autoload をチェックして、

autoload の値が n で始まっていたら終了します。

autoload の値が NFS の場合、nfs コマンドを同じ引数で呼び出します。

autoload が上記以外の場合は tftpboot コマンドを同じ引数で呼び出します。

関連: dhcp nfs rarpboot tftpboot

### 2.8.10 bootvx - ELF イメージの vxWorks を起動

文法: bootvx [addr]

addr: vxWorks の ELF イメージが置いてあるアドレスです。

bootm でも vxWorks は起動できますが、こちらは ELF ファイル版です。このコマンドの動作は未確認です。

関連: bootm

### 2.8.11 chpart - アクティブパーティションの変更

文法: chpart num

num:パーティション番号(0-2)

ls,fsload,fsinfo コマンドの対称にするパーティション番号を指定します  
フラッシュ上のパーティションです。

再起動すると、0に戻ります。

関連: ls fsload fsinfo

## 2.8.12 cmp - メモリの比較

文法: cmp[.b, .w, .l] addr1 addr2 count

[.b, .w, .l]: バス幅

演算するときに、バイト単位(.b)、ハーフワード(2バイト)単位(.w)か、ワード(4バイト)単位(.l)かを指定します。省略した場合、.lになります。

addr1: 比較するアドレス

addr2: 比較するアドレス

count: 比較する数

[.b, .w, .l]の単位で数えます。また、16進で指定します。

addr1 から count 個のメモリ範囲と、addr2 から count 個のメモリ範囲の内容が同じかどうか検査します。count はバイト数ではない事に注意して下さい。

例:

```
# nfs 80400000 192.168.3.91:/usr/src/mlinbox/u-boot.bin
Using i82559#0 device
File transfer via NFS from server 192.168.3.91; our IP address is
192.168.3.202
Filename '/usr/src/mlinbox/u-boot.bin'.
Load address: 0x80400000
Loading: #####
done
Bytes transferred = 199948 (30d0c hex)
# cmp.b BFC00000 80400000 $filesize
Total of 199948 bytes were the same
#
```

関連: base md mm nm mw cp crc32

## 2.8.13 coninfo - コンソールデバイスの表示

文法: coninfo

コンソールデバイスの一覧を表示します。

## 2.8.14 cp - メモリ間のコピー

文法: cp[.b, .w, .l] source target count

[.b, .w, .l]: バス幅

演算するとき、バイト単位(.b)、ハーフワード(2 バイト)単位(.w)か、ワード(4 バイト)単位(.l)かを指定します。省略した場合、.lになります。

source: コピー元アドレス

target: コピー先アドレス

count: コピーする数

[.b, .w, .l]の単位で数えます。また、16 進で指定します。

使い方の詳細は、`cmp` を参照して下さい。

関連: `base md mm nm mw cmp crc32`

### 2.8.15 `dhcp` - DHCP プロトコルで IPv4 アドレスを取得

文法: `dhcp`

DHCP プロトコルでアドレスを取得し、以下の環境変数を更新します。

`serverip`: DHCP サーバの IP アドレス

`ipaddr`: 割り当てられた IP アドレス

`bootfile`: 読みこむファイル

`gatewayip, netmask, hostname, rootpath, dnsip, dnsip2, domain`:

サーバの応答で指定があった場合に更新され、取得できなかったものは以前の値が保持されます。

次に、環境変数の `autoload` をチェックして、

`autoload` の値が `n` で始まっていたら終了します。

`autoload` の値が `NFS` の場合、`nfs` コマンドを引数なしで呼び出します。

`autoload` が上記以外の場合は `tftpboot` コマンドを引数なしで呼び出します。

関連: `bootp nfs rarpboot tftpboot`

### 2.8.16 `echo` - テキストを表示

文法: `echo [args ...]`

`args`: 出力する文字列

`args` を標準出力に出力します。出力する文字列に `\c` が含まれていると、改行は出力されません。

### 2.8.17 `erase` - フラッシュメモリの消去

文法: `erase start end`

`erase N:SF[-SL]`

`erase bank N`

erase all

start: 開始アドレス(最初のイレースブロックの先頭アドレス)

end: 終了アドレス(最後のイレースブロックの最終アドレス)

N: バンク番号(常に 1 を指定します)

SF: イレースブロック番号

SL: フラッシュ内のイレースブロックを先頭から数えた番号(0 から数え始めます)  
省略した場合、SF と同じ値になります。

指定された範囲内で、プロテクトされていないイレースブロックを消去します。  
イレースブロック番号は、フラッシュ内のイレースブロックを先頭から数えた番号です。  
(0 から数え始めます)

本機は 1 バンク構成ですから、erase bank 1 と erase all は同じ意味になります

本機に搭載されているフラッシュは先頭と末尾の 128kbyte はイレースブロックサイズ  
が 16kbyte で残りの部分は 128kbyte です。場所によってサイズが変わりますので、イ  
レースブロック番号の計算には注意が必要です。アドレス指定での使い方をお勧めしま  
す。

例

```
# erase BFC40000 BFFDFFFF
```

```
..... done
```

```
Erased 29 sectors
```

```
#
```

関連: protect

## 2.8.18 exit - スクリプトの終了

文法: exit

スクリプトを終了します

mkimage で作るスクリプトは、最後を NULL ターミネートしづらいので、最後の行は  
exit で終了する様にしておくと安全です。

## 2.8.19 flinfo - フラッシュの情報表示

文法: flinfo [N]

N: バンク番号(省略時は全バンク)

本機は 1 バンク構成ですから、バンク番号を指定しても何もかわりません。

フラッシュチップの概略、各イレースブロックの開始アドレス、プロテクトの状態が一覧  
出来ます。

関連: protect

## 2.8.20 fsinfo - ファイルシステム情報の表示

文法: fsinfo

アクティブパーティションのファイルシステム情報を表示します。

ファイルシステムは自動で認識されます

サポートされているファイルシステムは、cramfs と jffs2 です。

アクティブパーティションを変更するには、chpart を使います。

関連: chpart ls fsload

## 2.8.21 fsload - フラッシュ上のファイルシステムからファイルのロード

文法: fsload [addr] [filename]

addr: ロードするアドレス

省略した場合、最後に loadb,fsload,nfs,tftpboot 等のコマンドでファイルを読み込んだアドレス。起動後始めてファイルを読み込む場合、80400000 になります。

filename:ロードするファイル名

省略した場合、環境変数 bootfile の値。

bootfile が設定されていないならば、"uImage"

アクティブパーティションのファイルシステムからメモリ上にファイルを読み込みます。

ファイルシステムは自動で認識されます。

環境変数 filesize に読みこんだファイルサイズが設定されます。

サポートされているファイルシステムは、cramfs と jffs2 です。

関連: chpart ls fsinfo loadb loads nfs tftpboot

## 2.8.22 go - u-boot 用アプリケーションの実行

文法: go addr [arg ...]

add: 実行するアドレス

arg: プログラムに渡す引数

u-boot の内部関数を利用したアプリケーションを実行する場合に使います

以下に実行例を載せます

```
# nfs 80200000 $nfsbase/u-boot/examples/hello_world.bin
Using i82559#0 device
File transfer via NFS from server 192.168.3.91; our IP address is
192.168.3.202
Filename '/usr/src/mldbox//u-boot/examples/hello_world.bin'.
Load address: 0x80200000
Loading: #
done
```

```
Bytes transferred = 920 (398 hex)
# go 80200000 myip $serverip
## Starting application at 0x80200000 ...
Example expects ABI version 2
Actual U-Boot ABI version 2
Hello World
argc = 3
argv[0] = "80200000"
argv[1] = "myip"
argv[2] = "192.168.3.29"
argv[3] = "<NULL>"
Hit any key to exit ...

## Application terminated, rc = 0x0
#
```

関連: bootelf

### 2.8.23 help - オンラインヘルプ

文法: help [command...]

command: コマンド名

command が省略された場合、全てのコマンドとその概略の一覧を表示します。

command が指定された場合、そのコマンドの説明と使い方を表示します。

### 2.8.24 iminfo - アプリケーションイメージヘッダの表示

文法: iminfo [addr...]

addr: アドレス

省略した場合、最後にファイルを読み込んだアドレス。

ヘッダ情報表示と、チェックサム確認を行います。

例:

```
# iminfo

## Checking Image at 80400000 ...
Image Name:   Linux Kernel Image
Created:      2004-12-02  7:52:37 UTC
Image Type:   MIPS Linux Kernel Image (gzip compressed)
Data Size:    764307 Bytes = 746.4 kB
Load Address: 80000000
Entry Point:  8019a040
Verifying Checksum ... OK

#
```

### 2.8.25 imls - フラッシュの中にあるイメージを探す

文法: imls

フラッシュの全てのイレースブロックの先頭を調べて、アプリケーションイメージヘッダが見付かる毎に、見付けた場所を表示し、iminfo を実行します。

### 2.8.26 itest - 整数と文字列の比較テスト

文法: itest[.b, .w, .l, .s] [\*]value1 <op> [\*]value2

[.b, .w, .l, .s]:バイトで比較するか、ハーフワード(2 バイト)か、ワード(4 バイト)か、文字列として比較するかを指定します。省略された場合、.lと同じです。

[\*]value1:\*がある場合、value1 はアドレスとして解釈され、比較対象はそのアドレスにある値になります。\*がない場合比較対象は value1 そのものとなります。

op :<, >, =, <=, >=, <, >, <=, >= のどれか

[\*]value2:\*がある場合、value2 はアドレスとして解釈され、比較対象はそのアドレスにある値になります。\*がない場合比較対象は value2 そのものとなります。

比較を行い結果を返します。結果は表示されません。if や &&, || とともに用います。

関連: test

### 2.8.27 loadb - シリアル経由でファイルのダウンロード(kermit モード)

文法: loadb [addr] [baudrate]

addr: ダウンロードするアドレス

省略した場合、環境変数 loadaddr の値、loadaddr が設定されていない場合は、80400000 です。

baudrate: ダウンロードするボーレート

省略した場合現在のボーレートが使われます。

バイナリファイルをダウンロードします。C-kermit を使ってダウンロードする場合に使います。ダウンロードの間だけボーレートを変更する事もできます。

正常にダウンロードされると、ダウンロードされたファイルのサイズが環境変数 filesize に設定されます。その後、環境変数 autoscript が yes に設定されていると、autoscr を呼び出します。

関連: fsload loads nfs tftpboot

### 2.8.28 loads - シリアル経由でSレコード形式のファイルのダウンロード

文法: loads [offset]

offset: Sレコーで示されているアドレスに加算する値

省略した場合 0 です。

Sレコード形式のファイルをダウンロードします。

環境変数 `loads_echo` を 1 に設定すると、100 行読み込む毎に `!!` が表示されます。

正常にダウンロードされると、ダウンロードされたファイルのサイズが環境変数 `filesize` に設定されます。

関連: `fsload`, `loadb nfs tftpboot`

### 2.8.29 `loop` - 指定した範囲のアドレスを読み続ける無限ループ

文法: `loop[.b, .w, .l] addr count`

[.b, .w, .l]: バス幅

アクセスするときに、バイト単位か(.b)、ハーフワード(2バイト)単位(.w)か、ワード(4バイト)単位(.l)かを指定します。省略した場合、.lになります。

addr: 開始アドレス

count: 指定した単位での数を 16 進で指定します

開始アドレスから、count 個までの範囲を読み続けます。

動作確認はしていません。

### 2.8.30 `ls` - ファイルシステムの中身を一覧表示

文法: `ls [name]`

name: 表示したいディレクトリもしくはファイル名

省略時は / になります。

アクティブパーティションのファイルシステム内のリストを表示します。

ファイルシステムは自動で認識されます

サポートされているファイルシステムは、`cramfs` と `jffs2` です。

アクティブパーティションを変更するには、`chpart` を使います。

関連: `chpart` `fsinfo` `fsload`

### 2.8.31 `md` - メモリ内容の表示

文法: `md[.b, .w, .l] addr [count]`

[.b, .w, .l]: バス幅

アクセスするときに、バイト単位か(.b)、ハーフワード(2バイト)単位(.w)か、ワード(4バイト)単位(.l)かを指定します。省略した場合、.lになります。

addr: 表示するアドレス

count: 指定した単位での数を 16 進で指定します。

コマンド実行後に、改行だけを入力すると、自動的にアドレスがインクリメントされて実行されます。

例:

```

# md.b 80400000 8
80400000: 27 05 19 56 fc 21 36 93    '..V.!6.
#
80400008: 41 ae c9 c5 00 0b a9 93    A.....
#
80400010: 80 00 00 00 80 19 a0 40    .....@
#
80400018: 69 9f c3 da 05 05 02 01    i.....
#
80400020: 4c 69 6e 75 78 20 4b 65    Linux Ke
#

```

関連: base mm nm mw cmp cp crc32

### 2.8.32 mm - 連続するアドレスのメモリ内容を対話的に変更

文法: mm[.b, .w, .l] addr

[.b, .w, .l]: バス幅

アクセスするときに、バイト単位(.b)、ハーフワード(2バイト)単位(.w)か、ワード(4バイト)単位(.l)かを指定します。省略した場合、.lになります。

addr: 変更するアドレス

現在の値を表示し、新しい値を入力する様に促されますので、変更したい場合新しい値を16進で入力します。そのまま良ければそのまま改行を入力します。

'-'を入力すると、一つ前のアドレスに戻ります。

アドレスを自動的に指定単位分増やして再度聞いて来ます。

^C (Ctrl キーを押しながらC)で終了します。

例:

```

# md.w 80400020 4
80400020: 694c 756e 2078 654b    Linux Ke
# mm.w 80400020
80400020: 694c ? 494c
80400022: 756e ?
80400024: 2078 ? -
80400022: 756e ? # <INTERRUPT>
# md.w 80400020 4
80400020: 494c 756e 2078 654b    LInux Ke
#

```

関連: base md nm mw cmp cp crc32

### 2.8.33 mtest - 簡単なメモリのテスト

文法: mtest [start [end [pattern]]]

start: 開始アドレス

省略時 80000000 になります。

end: 終了アドレス

省略時 81F70000 になります。

pattern:0

パターンの値を書いて確認したあと、パターンの値をビット反転したもので確認します。

パターンの値を1増やしながらか、上記を永久に繰り返します。

^C (Ctrl キーを押しながら C)で終了します。

例:

```
# mtest 80000000 81F70000 55555555
Pattern 55555556 Writing... Reading...
#
```

### 2.8.34 mw - メモリ内容を指定した値で埋める

文法: mw[.b, .w, .l] addr val [count]

[.b, .w, .l]: バス幅

アクセスするときに、バイト単位か(.b)、ハーフワード(2バイト)単位(.w)か、ワード(4バイト)単位(.l)かを指定します。省略した場合、.lになります。

addr: 変更するアドレス

val: 値

count: 指定した単位での数を16進で指定します。

省略した場合、1とみなされます。

関連: base md nm mw cmp cp crc32

### 2.8.35 nfs - NFS プロトコルでファイルをダウンロード

文法: nfs [addr] [[ip:]filename]

addr: ロードするアドレスです。

省略した場合、環境変数 loadaddr の値、loadaddr も設定されていない場合は、80400000 です。

ip: nfs サーバの IP アドレスです。

省略した場合、環境変数 serverip の値が使われます。

filename: ロードするファイルです。

省略した場合は、環境変数 bootfile の値、環境変数 bootfile が設定されていない場合は、/nfsroot/<自分の IP アドレス>.img です。

<自分の IP アドレス>の部分は、IP アドレスを 16 進で表記して、並べたものです。例えば、  
192.168.3.202 を 16 進で表記すると、C0.A8.03.CA となりますので、並べると、  
CA03A8C0 となりますので、"/nfsroot/C0A803CA.img"を探しに行きます。

addr を省略して、filename を指定する場合、filename は " で括られていなくては  
いけません。パーサが " をはずしてしまうため、コマンドに " が渡るように、  
エスケープする必要があります。

ファイルのダウンロードが正常に終了すると、環境変数 fileaddr にロードしたアドレスが  
設定され、環境変数 filesize にダウンロードしたファイルのサイズが設定されます。その  
後、環境変数 autostart が yes に設定されていると、続けて bootm を呼び出し、環境変  
数 autoscript が yes に設定されていると、さらに続けて autoscr を呼び出します。

関連: fsload loadb loads tftpboot

### 2.8.36 nm - 同一アドレスのメモリ内容を対話的に変更

文法: nm[.b, .w, .l] addr

[.b, .w, .l]: バス幅

アクセスするときに、バイト単位(.b)、ハーフワード(2 バイト)単位(.w)か、ワード(4  
バイト)単位(.l)かを指定します。省略した場合は、.l になります。

addr: 変更するアドレス

現在の値を表示し、新しい値を入力する様に促されますので、新しい値を入力します。  
リターンだけの入力、書き込みしません。

^C (Ctrl キーを押しながら C)で終了します。

GPIO のレジスタや IO ポートを操作する際には便利です。

### 2.8.37 pci - PCI バスの一覧と、PCI コンフィグレーションスペースのアクセス

文法: pci [bus] [long]

bus: バス番号を指定します。省略時は 0

long: 詳細を出したいときには、long を指定します。

指定されなかった場合は、簡単な表示になります。

PCI バスをリストします。

文法: pci header <b.d.f>

b.d.f: PCI のデバイスを指定します。

b,d,f はそれぞれ、バス.デバイス.ファンクションの番号を指定します。

指定されたデバイスの詳細(PCI コンフィグレーションスペースのヘッダ情報)が出力されます。

文法: pci display[.b, .w, .l] b.d.f [addr] [count]

[.b, .w, .l]: バス幅

アクセスするときに、バイト単位か(.b)、ハーフワード(2 バイト)単位(.w)か、ワード(4 バイト)単位(.l)かを指定します。省略した場合、.l になります。

b.d.f: PCI のデバイスを指定します。

b,d,f はそれぞれ、バス.デバイス.ファンクションの番号を指定します。

addr: 表示するアドレス

count: 指定した単位での数を 16 進で指定します

PCI コンフィグレーションスペースの内容を表示します。

文法: pci modify[.b, .w, .l] b.d.f addr

[.b, .w, .l]: バス幅

アクセスするときに、バイト単位か(.b)、ハーフワード(2 バイト)単位(.w)か、ワード(4 バイト)単位(.l)かを指定します。省略した場合、.l になります。

addr: 変更するアドレス

mm コマンドの PCI 用です。現在の値を表示し、新しい値を入力する様に促されますので、変更したい場合新しい値を 16 進で入力します。そのまま良ければそのまま改行を入力します。

↑ を入力すると、一つ前のアドレスに戻ります。

アドレスを自動的に指定単位分増やして再度聞いて来ます。

^C (Ctrl キーを押しながら C)で終了します。

文法: pci next[.b, .w, .l] b.d.f addr

[.b, .w, .l]: バス幅

アクセスするときに、バイト単位か(.b)、ハーフワード(2 バイト)単位(.w)か、ワード(4 バイト)単位(.l)かを指定します。省略した場合、.l になります。

b.d.f: PCI のデバイスを指定します。

b,d,f はそれぞれ、バス.デバイス.ファンクションの番号を指定します。

addr: 変更するアドレス

nm コマンドの PCI 用です。現在の値を表示し、新しい値を入力する様に促されますので、新しい値を入力します。リターンだけの入力の場合は、書き込みしません。

^C (Ctrl キーを押しながら C)で終了します。

文法: pci write[.b, .w, .l] b.d.f addr value

[.b, .w, .l]: バス幅

アクセスするときに、バイト単位か(.b)、ハーフワード(2バイト)単位(.w)か、ワード(4バイト)単位(.l)かを指定します。省略した場合、.lになります。

b.d.f: PCIのデバイスを指定します。

b,d,fはそれぞれ、バス.デバイス.ファンクションの番号を指定します。

addr: 変更するアドレス

value: 書き込む値

long header display modify next write はそれぞれ、l h d m n w と略する事も可能です。

### 2.8.38 ping - ICMP ECHO\_REQUEST パケットを指定ホストに送る

文法: ping ip

ip: IPアドレス。

指定したアドレスに1回 ping パケットを送出し、返事を待ちます。

### 2.8.39 printenv - 環境変数の一覧と内容表示

文法: printenv [name...]

name: 環境変数の名前

name が指定されなかった場合、全ての環境変数を表示します。

name が指定された場合その環境変数だけを表示します。

### 2.8.40 protect - フラッシュメモリのプロテクトの設定

文法: protect op start end

protect op N:SF[-SL]

protect op bank N

protect op all

op: on もしくは off

start: 開始アドレス(最初のイレースブロックの先頭アドレス)

end: 終了アドレス(最後のイレースブロックの最終アドレス)

N: バンク番号(常に1を指定します)

SF: イレースブロック番号

SL: フラッシュ内のイレースブロックを先頭から数えた番号(0 から数え始めます)  
省略した場合、SFと同じ値になります。

指定された範囲内で、プロテクトの状態を変更します。  
イレースブロック番号は、フラッシュ内のイレースブロックを先頭から数えた番号です。  
(0 から数え始めます)

本機は 1 バンク構成ですから、protect bank 1 と protect all は同じ意味になります  
本機に搭載されているフラッシュは先頭と末尾の 128kbyte はイレースブロックサイズ  
が 16kbyte で残りの部分は 128kbyte です。場所によってサイズが変わりますので、イ  
レースブロック番号の計算には注意が必要です。アドレス指定での使い方をお勧めしま  
す。

関連: erase

#### 2.8.41 rarpboot- RARP プロトコルで IPv4 アドレスを取得

文法: rarpboot [addr] [filename]

addr: ロードするアドレスです。

省略された場合、環境変数 loadaddr の値、loadaddr も設定されていなければ、  
80400000 です。

filename: ロードするファイルです。

addr を省略して、filename を指定する場合、filename は " で括られていなくて  
はいけません、パーサが " をはずしてしまうため、rarpboot コマンドに " が渡る  
ように、エスケープする必要があります。

RARP プロトコルで IPv4 アドレスを取得し、環境変数 ipaddr を更新します。

また、環境変数 serverip が設定されていない場合、serverip を RARP サーバの IP ア  
ドレスで更新します。

次に、環境変数の autoload をチェックして、

autoload の値が n で始まっていたら終了します。

autoload の値が NFS の場合、nfs コマンドを同じ引数で呼び出します。

autoload が上記以外の場合は tftpboot コマンドを同じ引数で呼び出します。

このコマンドの動作は未確認です。

関連: bootp dhcp nfs tftpboot

#### 2.8.42 reset - CPU のリセット

文法: reset

再起動します。

#### 2.8.43 run - 環境変数に設定されている文字列の実行

文法: run name [...]

name: 環境変数の名称

環境変数に設定されている文字列を実行します

run a b は、run a && run b と同等です。

#### 2.8.44 saveenv- 現在の環境変数の値を全てフラッシュに保存

文法: saveenv

現在の環境変数の値を全てフラッシュに保存します。

自動的に生成される環境変数も保存されてしまいます。特に dhcp 等をお使いの際は割り振られたアドレスもセーブしてしまいますので注意して下さい。

#### 2.8.45 setenv - 環境変数の設定と削除

文法: setenv name value

name: 環境変数の名称

value: 設定する文字列

value を指定しなかった場合、その環境変数を削除します。

value が指定された場合、その文字列に設定します。

#### 2.8.46 sleep - 指定した秒数遅延させる

文法: sleep N

N: 遅延させる秒数(10進で指定します)

#### 2.8.47 test - シェルライクな test の最小限の実装

-o, -a, -z, -n, =, !=, >, <, -eq, -ne, -lt, -le, -gt, -ge が使えます。

数値は、10進数として扱われます。

if && || 等とともに利用します。

補足:

Hush パーサでは -n -z は事実上使えないので、

if test x\$env = x; then echo env not set; fi の様に使って下さい。

#### 2.8.48 tftpboot - TFTP プロトコルでファイルをダウンロード

文法: tftpboot [addr] [filename]

addr: ロードするアドレスです。

省略された場合、環境変数 loadaddr の値、loadaddr も設定されていなければ、80400000 です。

filename:ロードするファイルです。

省略された場合は、環境変数 bootfile の値、環境変数 bootfile が設定されていない場合、<自分の IP アドレス>.img を使います。

<自分の IP アドレス>の部分は、IP アドレスを 16 進で表記して、並べたものです。例えば、192.168.3.202 を 16 進で表記すると、C0.A8.03.CA となりますので、"CA03A8C0.img"を探しに行きます。

addr を省略して、filename を指定する場合、filename は " で括られていないはいけませんが、パーサが " をはずしてしまうため、コマンドに " が渡るように、エスケープする必要があります。

ファイルのダウンロードが正常に終了すると、環境変数 fileaddr にロードしたアドレスが設定され、環境変数 filesize にダウンロードしたファイルのサイズが設定されます。その後、環境変数 autostart が yes に設定されていると、続けて bootm を呼び出し、環境変数 autoscript が yes に設定されていると、さらに続けて autoscr を呼び出します。

関連: fsload loadb loads nfs

## 2.8.49 version - u-boot のバージョンの表示

文法: version

バージョンと、ビルドした日付と時刻が表示されます。

## 2.9 u-boot で特殊な意味を持つ環境変数の一覧

### 2.9.1 IFS

値: 文字列

初期値: なし

Hush パーサのトークンのセパレータです。

設定していなければ、スペース、タブ、改行になります。

設定しないで下さい。

### 2.9.2 autoload

値: n | NFS

初期値: no

bootp, dhcp, rarpboot コマンドで IP アドレスを取得したあと、自動的にファイルをダウンロードするかどうかを決めます。

n で始まる文字列であれば何もしません。

NFS なら nfs コマンドを実行します。

上記以外もしくは、設定されていないければ、tftpboot コマンドを実行します。

### 2.9.3 autoscript

値: yes

初期値: なし

loadb, nfs, tftpboot でファイルをダウンロードしたあと、自動的に autoscr を呼び出すかどうかを決めます。

yes 以外、もしくは設定されていないければ、呼び出されません。

### 2.9.4 autostart

値: yes

初期値: なし

loadb, nfs, tftpboot でファイルをダウンロードしたあと、自動的に bootm を呼び出すかどうかを決めます。

yes 以外、もしくは設定されていないければ、呼び出されません。

### 2.9.5 baudrate

値: 9600 | 19200 | 38400 | 57600 | 115200

初期値: 15200

コンソールのボーレートを設定します。

変更するとすぐに反映されます。

### 2.9.6 bootaddr

値: 16 進の値

初期値: なし

VxWorks を起動する際に使う様ですが、詳細は不明です。

### 2.9.7 bootargs

値: 任意の文字列

初期値: root=/dev/mtdblock1

OS に渡す起動パラメータを設定します。

### 2.9.8 bootcmd

値: 任意の文字列

初期値: run tryboot

デフォルトのカーネル起動方法を設定します。

(boot コマンドや、自動起動に設定されている時に参照されます)

### 2.9.9 bootdelay

値: -1 もしくは、負でない整数(10 進数)

初期値: 5

電源投入時にデフォルトのコマンドを実行するまでの待ち時間を設定します。

-1 を指定すると、自動起動しません。

### 2.9.10 bootfile

値: 任意の文字列

初期値: なし

起動に使うデフォルトのファイルを指定します。自動的に更新される場合がありますので、フラッシュへ保存する際には注意が必要です。

### 2.9.11 dnsip

値: IP アドレス

初期値: なし

bootp,dhcp で DNS サーバの IP アドレスがもらえた場合に設定されます。

この変数を参照するコマンドはありません。

### 2.9.12 dnsip2

値: IP アドレス

初期値: なし

bootp,dhcp で DNS サーバの二つ目の IP アドレスがもらえた場合に設定されます。

この変数を参照するコマンドはありません。

### 2.9.13 domain

値: 文字列

初期値: なし

bootp,dhcp でドメイン名がもらえた場合に設定されます。

この変数を参照するコマンドはありません。

### 2.9.14 ethact

値: i82559#0 | i82559#1

初期値: i82559#0

ネットワークを利用するコマンドが使用するデバイスを指定します。

i82559#0 は本体に 1 と書かれた方を使い、i82559#1 は本体に 2 と書かれた方を使います。

環境変数 `netretry` が `once` に設定されている場合、この変数に指定されたデバイスでアクセスに失敗するとこの変数は次のデバイスに書き換えられ、再試行されます。成功した場合はこの変数には成功したデバイス名が保持されています。

### 2.9.15 ethaddr

値: MAC アドレス

初期値: なし

本機 1 番のポートのイーサネットデバイスを初期化する際に指定された MAC アドレスを使うようにします。

設定した場合の動作は未確認です。

### 2.9.16 eth1addr

値: MAC アドレス

初期値: なし

本機 2 番のポートのイーサネットデバイスを初期化する際に指定された MAC アドレスを使うようにします。

設定した場合の動作は未確認です。

### 2.9.17 ethprime

値: i82559#0 | i82559#1

初期値: なし

ネットワークを利用するコマンドが優先的に使用するデバイスを指定します。

電源投入時、環境変数 `ethact` の値を `ethprime` の値で初期化します。

### 2.9.18 fileaddr

値: 16 進の値

初期値: なし

`nfs`, `tftpboot` コマンドが、ファイルをダウンロードしたメモリ上のアドレスを設定します。この変数を参照するコマンドはありません。

### 2.9.19 filesize

値: 16 進の値

初期値: なし

fsload,loadb,loads,nfs,tftpboot コマンドが、ファイルをダウンロードした時にダウンロードしたファイルサイズを設定します。

この変数を参照するコマンドはありません。

### 2.9.20 gatewayip

値: IP アドレス

初期値: なし

デフォルトゲートウェイの IP アドレスを設定します。

bootp,dhcp でゲートウェイの IP アドレスがもらえた場合は上書きされます。

### 2.9.21 hostname

値: 文字列

初期値: なし

dhcp で IP アドレスを要求する時に、現在の値を送信し、DHCP の応答で得られた値で更新されます。

### 2.9.22 ipaddr

値: IP アドレス

初期値: なし

本機の IP アドレスを設定します。

bootp,dhcp,rarpboot コマンドで IP アドレスが取得できた場合は上書きされます。

### 2.9.23 loadaddr

値: 16 進の値

初期値: なし

ファイルをダウンロードする際のデフォルトのアドレスを指定します。

### 2.9.24 loads\_echo

値: 1

初期値: なし

1 に設定すると、loads コマンドで、100 行読み込む毎に '!' が表示されます。

cu コマンドで S レコードファイルをダウンロードするには、1 に設定する必要があるでしょう。

### 2.9.25 netmask

値: IPアドレス

初期値: なし

bootp,dhcp でネットマスクがもらえた場合に設定されます。  
この変数を参照するコマンドはありません。

### 2.9.26 netretry

値: no | once

初期値: once

no が設定されていると、bootp,dhcp が失敗したときに、デバイスを切り替えて再試行しません。

once が設定されていると、bootp,dhcp が失敗したときに、1回だけ試します。

### 2.9.27 nvlan

native VLAN の略です。

VLAN の動作確認が弊社で出来ないため、この環境変数を設定したときの動作は未確認です。

### 2.9.28 preboot

値: 任意の文字

初期値: "echo;" \  
"echo Type \"boot\" for boot normal way;" \  
"echo Type \"run cram\" to boot with root filesystem on mtblock1" \  
"echo Type \"run usb\" to boot with root filesystem on USB;" \  
"echo Type \"run nfs\" to boot with NFS root filesystem;" \  
"echo"

自動起動の前に実行するコマンドを設定します。

### 2.9.29 rootpath

値: 任意の文字

初期値: なし

bootp,dhcp で rootpath がもらえた場合には上書きされます。  
この変数を参照するコマンドはありません。

### 2.9.30 serial#

値: 任意の文字

初期値: なし

製品のシリアルナンバーを設定します。  
一度設定すると、変更できなくなります。  
この変数を参照するコマンドはありません。

### 2.9.31 serverip

値: IP アドレス

初期値: なし

tftp サーバや、nfs のデフォルトサーバを設定します。  
rarpboot で設定されたり、bootp,dhcp で上書きされたりします。

### 2.9.32 stdin

値: デバイス名

初期値: serial

標準入力に使うデバイスを設定します。  
serial 以外の値の動作は未確認です。

### 2.9.33 stdout

値: デバイス名

初期値: serial

標準出力に使うデバイスを設定します。  
serial 以外の値の動作は未確認です。

### 2.9.34 stderr

値: デバイス名

初期値: serial

標準エラー出力に使うデバイスを設定します。  
serial 以外の値の動作は未確認です。

### 2.9.35 verify

値: n

初期値: no

autoscr, bootm を実行する時、チェックサムを検査するかどうかを指定します。

### 2.9.36 vlan

値: 4095 未満の正の整数

初期値: 未設定

802.1q の VLAN タグを設定します。  
設定したときの動作は未確認です。

## 3. u-boot 初期設定環境変数の説明

---

### 3.1 概要

大きく分けて、3つの目的で、より使いやすくなるように環境変数をいくつか初期設定してあります。できるだけ機能毎にスクリプトを分けて、組み合わせる事で自由度が増すように考えて設定してあります。

以下は環境変数に設定したスクリプトですので、実行するには、

```
run <変数名>
```

とします。

これらのいくつかは、フラッシュ内部のルートイメージにある /sbin/init と関係して成り立っている機能もあります。

- 内蔵フラッシュの書き換えを容易にする
- 内蔵フラッシュだけを使って Linux を起動する  
ネットワークは使わずに単体で起動します。
- ネットワーク経由で Linux を起動する  
内蔵フラッシュを使わずにネットワークだけで起動します  
主にデバッグ目的です。

#### 3.1.1 フラッシュ内部の/sbin/init の機能

フラッシュ内部の /sbin/init では、次の処理を行ないます。

カーネルコマンドラインオプション `usbroot` で指定されたデバイスを認識し、マウントする。

マウントできたら、

1. マウントしたファイルシステム内に、/sbin/init があれば、ルートファイルシステムをそのデバイスに変更して、/sbin/init を起動する
2. マウントしたファイルシステム内に、sbin/init が無いが、/rom.img があれば、/rom.img をループデバイス経由でマウントし、ルートファイルシステムをそのファイルに変更して、rom.img の中の sbin/init を起動する。

いずれかに失敗したら、フラッシュをルートファイルシステムとして起動する

フラッシュ内部の /sbin/init では次のカーネルコマンドラインオプションを解析します

- `usbroot`

ルートファイルシステムとして認識したいデバイスを設定します

指定がない、既にマウントされている、もしくは、このデバイスが正しくマウント出来なかった場合には、何もせずに、通常起動します。

- usbttimeout

ドライバをロードしたあと、デバイスが存在するのを確認するまでの、最大の待ち時間を指定します。

ドライバをロードしてから、実際に使える様になるまで、時間のかかる場合に、最大秒数を指定します。指定がない場合 5 秒になります。

1 秒毎に、usbroot で指定されたデバイスが存在するかどうかを確認し、最大 usbttimeout 回再試行するという意味です。

- mlddebug

デバッグ用です。mlddebug=y 等、何らかの値を設定すると、/sbin/init スクリプトの開始時や、終了時にシェルを立ち上げますので、状態を確認出来ます。exit することで、処理を続行します。

例 1: 最初の USB ディスク のパーティション 2 から起動したい場合

```
setenv usbroot 0/part2
```

## 3.2 準備

普通は NFS で開発するのが一番便利ですので、NFS の利用を前提に設定してあります。

例えば、192.168.3.91 のマシンで、/usr/src/mldbox/ というディレクトリを作成し、その下に

rom/	ルートファイルシステム
rom.img	ルートファイルシステムを cramfs でファイルに変換したもの
uImage	カーネルのイメージ
u-boot.bin	u-boot のイメージ
ramImage	イニシャルラムディスクのイメージ

があると前提して説明します。

**注意:**

u-boot の環境変数の保存は、起動直後に設定してから行なって下さい。DHCP でアドレスを取得した後に保存(saveenv)を行なうと、DHCP で取得したアドレスも保存されてしまいます。

### 3.2.1 標準の設定

サーバ側では、/etc/exports に

```
/usr/src/mlinbox 192.168.3.0/255.255.255.0(rw,async,no_root_squash,no_all_squash)
```

という行を追加して、nfs サーバを起動して下さい。

u-boot では、以下の2つの環境変数を設定すれば、DHCP 環境下で、NFS を利用可能であれば、準備は終了です。

nfsbase

ルートファイルシステムのイメージ、カーネルのイメージ、u-boot のイメージ等を置いたディレクトリを指定します。

例:

```
setenv nfsbase 192.168.3.91:/usr/src/mlinbox/
```

NFS サーバ IP:ディレクトリ/ (最後の / を忘れないで下さい)をご利用の環境に合わせて設定して下さい。

nfsroot

NFS 経由でルートファイルシステムをマウントして Linux を起動する際のルートファイルシステムのパスを設定します。

例:

```
setenv nfsbase 192.168.3.91:/usr/src/mlinbox/rom
```

NFS サーバ IP:ルートディレクトリ をご利用の環境に合わせて設定して下さい。

最後に、saveenv を実行して、フラッシュに保存して下さい。

### 3.2.2 固定 IP で運用する場合

必要に応じて DHCP でアドレスを取得に行くようになっていますが、以下の設定をする事で、固定 IP の環境に変更できます。

```
setenv gatewayip 192.168.3.1
setenv netmask 255.255.255.0
setenv ipaddr 192.168.3.243
setenv serverip 192.168.3.91
setenv nfsargs 'setenv bootargs root=/dev/nfs
ip=$ipaddr:$serverip:$gatewayip:$netmask::eth0:
nfsroot=$nfsroot ro'
```

とします。(nfsargs の行は、実際は一行です)

最後に、saveenv を実行して、フラッシュに保存して下さい。

### 3.2.3 NFS ではなく、tftp を使いたい場合

ネットワーク経由でファイルを取って来る設定である、load\_cram、load\_uboot、load\_kernel、load\_initrd の各変数の、nfs コマンドを tftp を使う様に直して下さい。また、サーバ側で配付するファイルを tftp のパスを nfsbase に設定します。

例えば、tftp の直下に置く場合は / に設定します。

```
setenv load_cram 'run checknfsbase checkipaddr&&tftp 80400000
  $nfsbase$cram_file'
setenv load_uboot 'run checknfsbase checkipaddr&&tftp 80400000
  $nfsbase$uboot_file'
setenv load_kernel 'run checknfsbase checkipaddr&&tftp 80400000
  $nfsbase$kernel_file&&iminfo 80400000'
setenv load_initrd 'run checknfsbase checkipaddr&&tftp 80600000
  $nfsbase$initrd_file&&iminfo 80600000'
setenv setenv nfsbase /
```

## 3.3 詳細

### 3.3.1 カーネルコマンドラインの設定

カーネルのコマンドラインを設定するスクリプトです。

#### 1. cramargs

内蔵フラッシュをルートファイルシステムとする最小限の設定を行いません。  
カーネルコマンドラインとして渡される環境変数 bootargs を上書きします。

#### 2. nfsargs

NFS ルートとして起動する最小限の設定を行いません。  
カーネルコマンドラインとして渡される環境変数 bootargs を上書きします。

#### 3. userargs

シリアルコンソールの設定等、どの起動方法でも共通の設定を書きます。  
bootargs に追加する様に記述します。

#### 4. nfsargs

サンプルの/sbin/init で解釈される引数を bootargs に追加します。  
NFS 起動であっても、NFS マウントされるルートファイルシステムの/sbin/init がサンプルの  
ものと同じであれば、この設定をすることで NFS ルートからローカルディスクへ移行できま  
す。

### 3.3.2 linux を起動する

初期設定で用意してある起動方法は以下の 5 つです。

#### 1. run cram

フラッシュのカレントパーティションから Linux カーネルを読みこみ、フラッシュをルートにマウ  
ントして起動します。

jffs もしくは、cramfs のファイルシステム内にある、/uImage を読みこみます。

ロードするファイル名は、環境変数 kernel\_file で変更できます。

環境変数の設定には、cramargs、userargs が実行されます

## 2. run usb

cram とほぼ同じですが、サンプルの/sbin/init が認識するオプションを追加でカーネルコマンドラインに追加するスクリプト、nfsargs が追加で実行されます。

環境変数の設定には、cramargs、usbargs、userargs が実行されます

## 3. run initrd

フラッシュのカレントパーティションから Linux カーネルとイニシャルラムディスクを読みこみ、フラッシュをルートにマウントして起動します。

jffs もしくは、cramfs のファイルシステム内にある、/uImage と/ramImage を読みこみます。

ロードするファイル名は、環境変数 kernel\_file と、initrd\_file で変更できます。

環境変数の設定には、cramargs、nfsargs、userargs が実行されます

## 4. run nfs

ネットワーク経由で Linux カーネルを読みこみ、NFS ルートで起動します。

ロードするカーネルは環境変数 kernel\_file で変更可能

ロードするカーネルの場所は、環境変数 nfsbase で変更可能

ルートファイルシステムとして、マウントする場所は、環境変数 nfsroot で変更可能

環境変数の設定には、nfsargs、userargs が実行されます

## 5. run nfsinitrd

ネットワーク経由で Linux カーネルとイニシャルラムディスクを読みこみ、NFS ルートで起動します。

ロードするファイルは環境変数 kernel\_file と、initrd\_file で変更可能

ロードするカーネルの場所は、環境変数 nfsbase で変更可能

ルートファイルシステムとして、マウントする場所は、環境変数 nfsroot で変更可能

環境変数の設定には、nfsargs、usbargs、userargs が実行されます

## デフォルトの起動方法

u-boot 起動時のデフォルトの起動方法は環境変数 bootorder で指定します

例えば、

```
setenv bootorder 'nfs usb'
```

とすると、run nfs 起動を試して、失敗したら、run usb を実行します。これにより、ネットワークケーブルが差さっていればネットワークから、差さっていなければ、ローカルなディスクからといった運用が可能になります。

### 3.3.3 フラッシュのアップデート

#### ● update\_cram

root ファイルシステムをネットワーク経由で取って来て、フラッシュに書きこみます。

上記の設定の場合、92.168.3.91:/usr/src/mlinbox/rom.img を取ってきて書きこみます。

ロードするファイルは環境変数 `cram_file` で変更可能

ロードするファイルの場所は、環境変数 `nfsbase` で変更可能

- `update_uboot`

`u-boot` をネットワーク経由で取って来て、フラッシュに書きこみます。

上記の設定の場合、`92.168.3.91:/usr/src/mltbox/u-boot.bin` を取ってきて書きこみます。

ロードするファイルは環境変数 `fuboot_file` で変更可能

ロードするファイルの場所は、環境変数 `nfsbase` で変更可能

### 3.3.4 消去

- `init_env`

`u-boot` の環境変数エリアをクリアします。

再起動すると、全ての環境変数がデフォルトの状態にもどります。

- `init_opt`

`opt` パーティションをクリアします。

- `init_conf`

`conf` パーティションをクリアします。

### 3.3.5 環境変数詳細

- `load_` で始まる変数は、ファイルのダウンロードの仕方を設定しています。

- `init_` で始まる変数は、フラッシュの各領域の削除の仕方を設定しています。

`init_env` で `u-boot` の環境変数エリアをクリアしますので、再起動するとデフォルトの状態にもどります。

`init_opt` は `opt` 領域をクリアします。クリアする事で、JFFS2 のファイルシステムとしては初期化された状態になります。

- `verify_` で始まる変数は、フラッシュの各領域の内容がネットワーク上のものと同じかチェックします。

## 3.4 具体的な起動例

- `run usb` と同じ事を、NFS 上にあるカーネルと、ルートイメージで行なうには、

```
setenv test 'if run load_kernel;then run cramargs usbargs
userargs;bootm 80400000;fi'
```

(NFS 上にあるルートファイルシステムの `/sbin/init` がフラッシュ内部のものと同じである場合)

- 最初の USB ディスクのパーティション6から起動

```
setenv redhat 'setenv rootdev 0/part6;run usb'
```

### 3.4.1 詳細

以下、`printenv` の出力を元に、簡単なコメントを挿入しています

```
# printenv
bootcmd=run tryboot
bootdelay=5
baudrate=115200
preboot=echo `
Type "boot"          for default way
Type "run cram"      to boot from flash
Type "run usb"       to boot from USB-disk
Type "run nfs"       to boot from NFS
Type "run initrd"    to boot with initrd
Type "run nfsinitrd" to boot with nfs initrd
`
bootargs=root=/dev/mtdblock1
```

`run userargs` とすると、カーネル起動パラメータにシリアルコンソールを使う設定が追加されます。

```
userargs=setenv bootargs $bootargs console=ttyS0,$baudrate
cramargs=setenv bootargs root=/dev/mtdblock1
```

`run usbargs` とすると、カーネル起動パラメータに `usb` から起動するための引数が追加されます。`usb` から起動するための引数の解析は、実際には、起動してから、シェルスクリプトで解析して、`pivot_root`しています。

```
usbargs=setenv bootargs $bootargs usbroot=$usbroot
usbtimeout=$usbtimeout ro
```

`run nfsargs` とすると、カーネル起動パラメータに `nfs` ルートを使う設定が追加されます。

`ip` アドレス固定の方や、`bootp` で、ルートパスを指定している方はここを書き換える必要があるでしょう。

```
nfsargs=setenv bootargs root=/dev/nfs ip=dhcp nfsroot=$nfsroot ro
```

`run cram` とすると、フラッシュから起動します

```
cram=run cramargs userargs;fsload uImage&&bootm
```

`run usb` とすると、`usb` ディスクから起動します

```
usb=run cramargs usbargs userargs;fsload uImage&&bootm
```

`run nfs` とするとカーネルをネットワーク越しにダウンロードして、ルートファイルシステムを NFS でマウントした Linux が起動されます。

```
nfs=if run load_kernel;then run nfsargs userargs;bootm 80400000;fi
```

`run initrd` とすると イニシャルラムディスクを使ったロードを行いません。

```
initrd=run cramargs usbargs userargs;fsload 80400000 ulmage&&fsload
80500000 ramImage&&bootm 80400000 80500000
```

`run nfsinitrd` とするとカーネルとイニシャルラムディスクをネットワーク越しにダウンロードして起動を行いません。

```
nfsinitrd=if run load_kernel load_initrd; then run nfsargs usbargs userargs;
bootm 80400000 80500000;fi
```

ルートにマウントする usb ディスクの番号とパーティション番号の設定です。

```
usbroot=0/part2
```

usb ディスクの認識までのタイムアウトです。

```
usbtimeout=5
autoload=no
netretry=once
verify=no
```

パーティションの設定です。

```
cram_start=BF000000
cram_end=BFBFFFFF
uboot_start=BFC00000
uboot_end=BFC3FFFF
opt_start=BFC40000
opt_end=BFFDFFFF
conf_start=BFFE0000
conf_end=BFFFBBBB
env_start=BFFFC000
env_end=BFFFFFFF
```

ダウンロードするファイル名の設定です。

```
cram_file=rom.img
uboot_file=u-boot.bin
kernel_file=uImage
initrd_file=ramImage
```

```
checknfsbase=if test x$nfsbase = x;then echo "*** ERROR: 'nfsbase'
not set";test a = b;else test 1 = 1; fi
```

```
checkipaddr=if test x$ipaddr = x;then dhcp;else test 1 = 1;fi
```

ファイルのダウンロード方法の指定です。nfs でなく、tftp 等を使いたい方はここを変更する必要があります。

```
load_cram=run checknfsbase checkipaddr&&nfs 80400000
$nfsbase$cram_file
load_uboot=run checknfsbase checkipaddr&&nfs 80400000
$nfsbase$uboot_file
load_kernel=run checknfsbase checkipaddr&&nfs 80400000
$nfsbase$kernel_file&&iminfo 80400000
load_initrd=run checknfsbase checkipaddr&&nfs 80500000
$nfsbase$initrd_file&&iminfo 80500000
```

消去方法の指定です。

```
init_cram=protect on all;protect off $cram_start $cram_end;echo
    erasing cram area;erase $cram_start $cram_end
init_uboot=protect on all;protect off $uboot_start $uboot_end;echo
    erasing uboot area;erase $uboot_start $uboot_end;
init_opt=protect on all;protect off $opt_start $opt_end;echo
    erasing opt area;erase $opt_start $opt_end
init_conf=protect on all;protect off $conf_start $conf_end;echo
    erasing conf area;erase $conf_start $conf_end
init_env=protect on all;protect off $env_start $env_end;echo
    erasing env area;erase $env_start $env_end
```

ベリファイの指定です。

```
verify_cram=if run load_cram; then echo verifying image;cmp.b
    80400000 $cram_start $filesize&&fsload uImage&&imininfo;fi
verify_uboot=if run load_uboot; then echo verifying image;cmp.b
    80400000 $uboot_start $filesize;else test a = b;fi
update_cram=if run load_cram; then run init_cram;cp.b 80400000
    $cram_start $filesize;run verify_cram;fi
```

フラッシュの更新

```
update_cram=if run load_cram; then run init_cram;cp.b 80400000
    $cram_start $filesize;run verify_cram;fi
update_uboot=if run load_uboot; then run init_uboot;cp.b 80400000
    $uboot_start $filesize;run verify_uboot && reset;fi
```

起動の優先順位の設定

```
bootorder=cram usb nfs
tryboot=for i in $bootorder; do run $i; done
```

起動時のデフォルト

```
stdin=serial
stdout=serial
stderr=serial
ethact=i82559#0
```

Environment size: 2664/16380 bytes

#