U-BOOT
ユーザーズマニュアル

ARM319(TB0319)編

履歴(詳細はソース管理 git を参照)

2006.07.28	初版作成
2006.08.26	フラッシュのパーティション変更
2007.03.28	例外ハンドラ追加
	tftp,nfs コマンドでフラッシュへの直接書き込み対応
	RTS,DSR,CTS の動作変更
	CFG_XX, Makfile の整理
2007.04.14	リセット時の起動不具合修正
	nfsargs のデフォルト修正

メディアラボ株式会社

目次

1. はじめに

1.1 前堤

このマニュアルは、TB0319 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下さい。

動作確認済みのクロス開発環境：

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

RedHat Fedora Core3

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。

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 の開発は以下で行なわれています。

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

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、参照してみて下さい。
日本語のサイトでは、

http://www.u-boot.jp/pukiwiki/

が参考になります。

このマニュアルでの 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_JFFS2 | \
                             CFG_CMD_I2C | \
                             CFG_CMD_EEPROM | \
                             CFG_CMD_DATE | \
                             CFG_CMD_SNTP | \
                             CFG_CMD_BMP | \
                             CFG_CMD_DHCP | \
                             CFG_CMD_PING | \
                             CFG_CMD_MII )
```

TB0319 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 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 開発環境

u-boot のコンパイルには、以下の組み合わせで、クロス環境で行なっております。

binutils-2.16.1

gcc-3.4.5

http://buildroot.uclibc.org/ で配付されている環境を使えば、簡単に作成出来ます。

2.3 コンパイル

フラッシュにいれる u-boot は、

```
$ make distclean
$ make tb0319_config
$ make
```

で コンパイル出来ます。

シリアル起動に利用するバージョンは、

```
$ make distclean
$ make tb0319_config_serial
$ make
```

WDT 無効版は、

```
$ make distclean
$ make tb0319_config_nowdt
$ make
```

でそれぞれ作成できます。

2.4 ソースコードについて

u-boot のソースには、変更履歴も保存されております。ソースコードのバージョン管理は git で行なわれています。

git の詳細については、http://git.or.cz/ 等を参照して下さい。

TB0319 固有のファイルは、

include/configs/tb0319.h

board/tb0319/*

です。

2.5 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

TB0319 は、標準で 64MB の RAM と、64MB のフラッシュメモリを実装しています。

2.5.1 メモリマップ

u-boot から利用可能なアドレス範囲は、次の通りです。

開始	*終了*	*用途*
00000000	01FFFFFF	SDRAM(32M)
04000000	05FFFFFF	SDRAM(32M)
60000000	63FFFFFF	フラッシュメモリ(64M)
80000000	8008FFFF	AHB レジスタ：詳細は EP9307 のマニュアルを参照して下さい。
80090000	80093FFF	BootROM(16K)：詳細は EP9307 のマニュアルを参照して下さい。
800A0000	900FFFFF	AHB レジスタ：詳細は EP9307 のマニュアルを参照して下さい。
80800000	8094FFFF	APB レジスタ：詳細は EP9307 のマニュアルを参照して下さい。

2.5.2 DRAM エリアの詳細

開始	*終了*	*用途*
00000000	000000FF	例外ハンドラのベクタテーブル
00000100	00007FFF	空き(Linux 起動時 kernel パラメータの引渡し場所)
00008000	01FFFFFF	空き(Linux 起動時 kernel ロードアドレス)
04000000		空き
	05E7FFFF	u-boot スタック
05E80000	05EFFFFF	u-boot malloc 領域(512k)
05F00000	05F2E000	u-boot 本体
05F2E000	05F35000	u-boot データ
05F35000	05FFFFFF	フレームバッファ

u-boot が使っているアドレスは、u-boot 本体の開始アドレスのみが正確なアドレスで、残りは動的に確保していますので、コンパイルしなおしたりすると場所がかわる可能性があります。

2.5.3 フラッシュエリアの詳細

フラッシュエリアは以下の様に分けています。

Linux の mtdblock 番号と名称	*用途*	*開始アドレス*	*サイズ*
0 uboot	u-boot 本体	60000000	256K bytes
1 ubootenv	u-boot 環境変数保存領域	60040000	256K bytes
2 kernel	カーネル保存エリア	60080000	3.5M bytes
3 root	ルートファイルシステム	60400000	28M bytes
4 config	ユーザエリア	62000000	32M bytes

2.6 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専用のコマンドが 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のいくつかのコマンドだけが、書き込み先アドレスがフラッシュの場合に、フラッシュへの書き込みを行なう関数を呼び出すような仕掛になっております。

本機のフラッシュには、NOR 型のフラッシュが搭載されておりますので、消去すると、FF の値で埋められ、書き込みは、1 のビットを 0 に変更するという動作になります。

ですから、フラッシュメモリ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と、期待した値にはならないかもしれませんから注意が必要です。

フラッシュメモリは、1 バイト単位で書き込みを行なえますが、消去はブロック単位でしかおこなえません。一部分だけを変更したい場合には、ブロック全体を一度 DRAM 上にコピーして書き変えたあとに、そのブロックを消去して、書き戻す事になります。

書き込みや、イレースを実行できるのは、プロテクトされていない所だけです。

先に消去が必要なため、一旦 DRAM にロードしてから cp コマンドで書きむのが一番確実な書き込み方法です。

2.7 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。

2.7.1 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。

2.7.2 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、

```
# mkimage  -A arm -O linux -T kernel -C none -a  0x00008000 -e
0x00008000 -n 'Linux-2.6.12.3' -d arch/arm/boot/Image uImage
Image Name:   Linux-2.6.12.3
Created:      Sun Aug  6 15:56:43 2006
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    2318912 Bytes = 2264.56 kB = 2.21 MB
Load Address: 0x00008000
Entry Point:  0x00008000
```

2.7.3 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
$ echo "exit" >> script
$ mkimage -A ARM -O u-boot -T script -C none -n "hello" -a
0x00200000 -d script script.bin
Image Name:   hello
Created:      Sun Aug  6 16:03:28 2006
Image Type:   ARM U-Boot Script (uncompressed)
Data Size:    24 Bytes = 0.02 kB = 0.00 MB
Load Address: 0x00200000
Entry Point:  0x00200000
```

```
Contents:
   Image 0:       16 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。

```
Arm9# nfs 400000 192.168.3.91:/home/arm9/script.bin
File transfer via NFS from server 192.168.3.91; our IP address is
192.168.3.253
Filename '/home/arm9/script.bin'.
Load address: 0x400000
Loading: #
done
Bytes transferred = 88 (58 hex)
Arm9# autoscr 400000
## Executing script at 00400000
hello
Arm9#
```

2.7.4 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、

```
$ mkimage -A ARM -O u-boot -T standalone -C none -n "hello" -a
0x00100000 -d u-boot.git/examples/hello_world.bin hello.img
Image Name:   hello
Created:      Sun Aug  6 16:18:36 2006
Image Type:   ARM U-Boot Standalone Program (uncompressed)
Data Size:    515 Bytes = 0.50 kB = 0.00 MB
Load Address: 0x00100000
Entry Point:  0x00100000
```

mkimage で作ったものは、bootm で実行できます。

```
Arm9# nfs 400000 192.168.3.91:/home/arm9/hello.img
File transfer via NFS from server 192.168.3.91; our IP address is
192.168.3.253
Filename '/home/arm9/hello.img'.
Load address: 0x400000
Loading: #
done
Bytes transferred = 579 (243 hex)
Arm9# bootm
## Booting image at 00400000 ...
   Image Name:   hello
   Created:      2006-08-06   7:18:36 UTC
```

```
   Image Type:   ARM U-Boot Standalone Program (uncompressed)
   Data Size:    515 Bytes =  0.5 kB
   Load Address: 00100000
   Entry Point:  00100000
   Loading Standalone Application ... 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。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  tstc(void);                                    標準入力に文字が来ているか調べる
void putc(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

int i2c_write(uchar chip, uint addr, int alen, uchar *buffer, int len); I2C バスの書込

int i2c_read(uchar chip, uint addr, int alen, uchar *buffer, int len); I2C バスの読込

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 adress */
    struct environment_s            *bi_env;
    ulong               bi_arch_number;         /* unique id for this board */
    ulong               bi_boot_params;         /* where this board expects params */
    struct                                      /* RAM configuration */
    {
        ulong start;
        ulong size;
    }                   bi_dram[CONFIG_NR_DRAM_BANKS];
} 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 reloc_off;        /* Relocation Offset */
        unsigned long env_addr;         /* Address  of Environment struct */
        unsigned long env_valid;        /* Checksum of Environment valid? */
        unsigned long fb_base;          /* base address of frame buffer */
        void            **jt;           /* jump table */
}gd_t *gd;
```

2.7.5　fw_printenv と fw_setenv

Linux から u-boot の環境変数エリアを書き変えることも出来ます。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。

2.8 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。

2.8.1 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。

2.8.2 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。

2.8.3 クオート処理

$ 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。

2.8.4 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。

2.8.5 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。

2.8.6 ^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。

2.9 シリアルの繋げ方

2.9.1 概要

初期設定は

ボーレート　：115200
データ長　：8bit
パリティー　：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。

2.9.2 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。

まず、取りこぼしがないように、ボーレートを下げます。

```
Arm9# setenv baudrate 38400
## Switch baudrate to 38400 bps and press ENTER ...
~.
Disconnected.
$ cu -s 38400 --nostop -l ttyS0
Connected.                  <=ここで、Enter キーを押します
Arm9# setenv loads_echo 1
Arm9# loads 00400000
## 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
Arm9#
```

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。

2.9.3 Linux から C-kermit 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/mldbox/) C-Kermit>set line /dev/ttyS0
(/usr/src/mldbox/) C-Kermit>set flow auto
(/usr/src/mldbox/) C-Kermit>set speed 115200
/dev/ttyS0, 115200 bps
(/usr/src/mldbox/) C-Kermit>set serial 8n1
(/usr/src/mldbox/) C-Kermit>set carrier-watch off
(/usr/src/mldbox/) C-Kermit>set escape 27
(/usr/src/mldbox/) C-Kermit>set prefixing all
(/usr/src/mldbox/) 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 0x00400000 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/mldbox/) 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/mldbox
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/mldbox/) 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/mldbox/) C-Kermit>q
Closing /dev/ttyS0...OK
```

```
#
```

~/.kermrc を以下の様に設定しておくと、kermit 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
#
```

2.9.4 Windows から利用する場合

Windows で利用する場合のお薦めは、Tera Term (http://hp.vector.co.jp/authors/VA002416/) ですが、Tera Term の kermit モードは、u-boot とは互換性がありません。kermit モードを使いたいときは、Windows 標準のハイパーターミナルをご利用下さい。

2.10 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。

2.10.1 ? - '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

2.10.2 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、00400000 です

スクリプトは、mkimage で作成されたものでなくてはいけません。

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。)

例:

```
Arm9# iminfo 00400000

## Checking Image at 00400000 ...
   Image Name:   hello
   Created:      2006-08-06   7:04:52 UTC
   Image Type:   ARM U-Boot Script (uncompressed)
   Data Size:    24 Bytes =  0 kB
   Load Address: 00200000
   Entry Point:  00200000
   Verifying Checksum ... OK
Arm9# autoscr 400000
## Executing script at 00400000
hello
Arm9#
```

関連: run

2.10.3 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

2.10.4 bdinfo - ボードの情報を表示

文法: bdinfo

RAM のサイズや、MAC アドレスなどのボードに関する情報を表示します。

例：

```
Arm9# bdinfo
arch_number = 0x00000447
env_t       = 0x00000000
boot_params = 0x00000100
DRAM bank   = 0x00000000
-> start    = 0x00000000
-> size     = 0x02000000
DRAM bank   = 0x00000001
-> start    = 0x04000000
-> size     = 0x02000000
ethaddr     = 00:20:0F:12:00:01
ip_addr     = 192.168.3.237
baudrate    = 115200 bps
```

2.10.5 bmp - 画像の表示

文法：bmp info addr

addr: メモリ上のアドレス

bmp ファイルの詳細を表示します

例：

```
Arm9# bmp info 400000
Image size    : 250 x 50
Bits per pixel: 8
Compression   : 0
```

文法：bmp display addr [x y]

addr: メモリ上のアドレス

x：LCD 上の X 座標

y: LCD 上の Y 座標

BMP ファイルを表示します。

関連：cls

2.10.6 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。

2.10.7 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 に設定しておくと、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： go, iminfo

2.10.8 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いけませんが、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,domain,:ntpserverip

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

2.10.9 chpart - アクティブパーティションの変更

文法:chpart part-id

part-id:パーティション ID(詳細は mtdparts の項を参照して下さい)

ls,fsload,fsinfo コマンドの対称にするパーティションを指定します

フラッシュ上のパーティションです。

環境変数 partition、mtddevnum、mtddevname が更新されます。

関連: ls fsload fsinfo

2.10.10 cls - 画面のクリア

文法: cls

LCD 画面をクリアします。

関連: bmp

2.10.11 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。

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：

```
Arm9# nfs 400000 192.168.3.91:/home/arm9/u-boot.bin
File transfer via NFS from server 192.168.3.91; our IP address is
192.168.3.242
Filename '/home/arm9/u-boot.bin'.
Load address: 0x400000
Loading: #################################
done
Bytes transferred = 188464 (2e030 hex)
Arm9# cmp.b 60000000 400000 $filesize
Total of 188464 bytes were the same
Arm9#
```

関連：base md mm nm mw cp crc32

2.10.12 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と、現在の設定を表示します。

例：

Arm9# **coninfo**

List of available devices:

lcd 00000002 ..O

serial 80000003 SIO stdin stdout stderr

2.10.13 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。

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

2.10.14 crc32 - チェックサムの計算

文法：crc32 address count [addr]

address：チェックサム開始位置のメモリ上のアドレス(16 進)

count: チェックサムを取る範囲(16 進)

addr: 結果を保存する DRAM 上のアドレス

関連：base md mm nm mw cmp crc32

2.10.15 date - RTC クロックの表示と設定

文法：date

現在の設定内容を表示します。

文法：date MMDDhhmm[[CC]YY][.ss]

MM：月

DD: 日

hh: 時

mm: 分

CC: 年の上位 2 桁

YY: 年の下位 2 桁

ss: 秒

例：

```
Arm9# date 080711392006.10
Date: 2006-08-07 (Monday)    Time: 11:39:10
```

文法：date reset

RTC の初期化を行ないます。(RTC が止まっている場合に動かします)

関連：sntp

2.10.16 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,domain,ntpserverip:

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

2.10.17 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。

HASH パーサが \ を処理するので、クオートのしかたによっては注意が必要です

例:

```
Arm9# echo 'a\\c'
aArm9# echo "a\c"
aArm9# echo a\\c
aArm9#
```

2.10.18 eeprom - eeprom の操作

文法: eeprom read addr off cnt

addr: DRAM 上のアドレス(16 進)

off: eeprom 上でのアドレス(16 進)

cnt: 読みこむバイト数(16 進)

eeprom の内容を DRAM にコピーします

文法: eeprom write addr off cnt

addr: DRAM 上のアドレス(16 進)

off: eeprom 上でのアドレス(16 進)

cnt: 書き込むバイト数(16 進)
DRAM の内容を eeprom に書き込みます

文法： eeprom erase off
off: eeprom 上でのアドレス(16 進)
eeprom の off 番値の属するセクタを消去します。
eeprom には、MAC アドレス等が保存されています。
利用するには注意して下さい。

文法： eeprom info
認識した chip の型番を表示します。

2.10.19 erase - フラッシュメモリの消去

文法： erase start end
erase start +len
erase N:SF[-SL]
erase bank N
erase part-id
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)
len: start +len-1(16 進)の位置が含まれるイレースブロックまで
という意味になります。
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。
part-id: パーティション ID(詳細は mtdparts の項を参照して下さい)

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
プロテクションのかかっているイレースブロックは消去できません。先に protect コマンド

でプロテクトを外してから、消去します。

例

```
# erase BFC40000 BFFDFFFF

.......................................................... done
Erased 29 sectors
#
```

関連：flinfo protect

2.10.20 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。

2.10.21 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

2.10.22 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

2.10.23 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 が設定されていなければ、”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

2.10.24 go - u-boot 用アプリケーションの実行

文法：go addr [arg ...]

addr: 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

```
Arm9# nfs 100000 $nfsbase/u-boot.git/examples/hello_world.bin
File transfer via NFS from server 192.168.3.91; our IP address is
192.168.3.205
Filename '/home/arm9/u-boot.git/examples/hello_world.bin'.
Load address: 0x100000
Loading: #
done
Bytes transferred = 515 (203 hex)
Arm9# go 100000 myip $serverip
## Starting application at 0x00100000 ...
Example expects ABI version 2
Actual U-Boot ABI version 2
Hello World
argc = 3
argv[0] = "100000"
argv[1] = "myip"
argv[2] = "192.168.3.29"
argv[3] = "<NULL>"
Hit any key to exit ...

## Application terminated, rc = 0x0
Arm9#
```

関連： bootm

2.10.25 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。

関連： ?

2.10.26 icrc32 - I2C バスに繋がったメモリのチェックサムを計算

文法： icrc32 chip address[.0, .1, .2] count

chip:I2C バス上のアドレス(16 進)

address:チップ内での開始アドレス(16 進)

[.0, .1, .2] :address のサイズ、.0 はアドレスがない場合、.1 は 1byte アドレス、
.2 は 2 バイトアドレスとしてアドレスを出力する。省略時は .1

connt:CRC32 を計算する範囲(16 進)

関連： crc32, iprobe, iloop, imd, imm, imw, inm

2.10.27 iloop - I2C のリード無限ループ

文法： iloop chip address[.0, .1, .2] count

chip: I2C バス上のアドレス(16 進)

address: チップ内での開始アドレス(16 進)

[.0, .1, .2] :address のサイズ、.0 はアドレスがない場合、.1 は 1byte アドレス、
.2 は 2 バイトアドレスとしてアドレスを出力する。省略時は .1

connt: 読みだす範囲(16 進)

指定したアドレスから count 個分データを読みつづけます。

関連： loop, icrc32, iprobe, imd, imm, imw, inm

2.10.28 imd - I2C デバイスのレジスタのダンプ

文法： imd chip address[.0, .1, .2] count

chip: I2C バス上のアドレス(16 進)

address: チップ内での開始アドレス(16 進)

[.0, .1, .2] :address のサイズ、.0 はアドレスがない場合、.1 は 1byte アドレス、
.2 は 2 バイトアドレスとしてアドレスを出力する。省略時は .1

connt: 読みだす範囲(16 進)

指定したアドレスから count 個分データを読み表示します。

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 count 個分インクリメントされて実行されます。

例：RTC クロックのレジスタのダンプをとるには、

```
Arm9# imd 68 0 10
0000: 20 06 05 03 08 88 06 00 10 00 05 00 00 1f 18
01     ...............
Arm9# imd 68 0 1
0000: 24    $
Arm9#
0001: 35    5
Arm9#
```

関連：md, icrc32, iprobe, iloop, imm, imw, inm

2.10.29 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：

```
Arm9# iminfo 60080000

## Checking Image at 60080000 ...
   Image Name:   Linux-2.6.12.3
   Created:      2006-08-02  15:47:41 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    2318912 Bytes =  2.2 MB
   Load Address: 00008000
   Entry Point:  00008000
   Verifying Checksum ... OK
Arm9#
```

2.10.30 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。

2.10.31 imm - I2C デバイスのレジスタ値を対話的に変更

文法： imm chip address[.0, .1, .2]

chip: I2C バス上のアドレス(16 進)

address: チップ内での開始アドレス(16 進)

[.0, .1, .2] :address のサイズ、.0 はアドレスがない場合、.1 は 1byte アドレス、
.2 は 2 バイトアドレスとしてアドレスを出力する。省略時は .1

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。そのままで良ければそのまま改行を入力します。
アドレスを自動的にインクリメント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。

関連： mm, icrc32, iprobe, iloop, imd, imw, inm

2.10.32 imw - I2C デバイスのレジスタを指定した値で埋める

文法： imw chip address[.0, .1, .2] value [count]

chip: I2C バス上のアドレス(16 進)

address: チップ内での開始アドレス(16 進)

[.0, .1, .2] :address のサイズ、.0 はアドレスがない場合、.1 は 1byte アドレス、
.2 は 2 バイトアドレスとしてアドレスを出力する。省略時は .1

value: 書き込む値(16 進)

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： mw, icrc32, iprobe, iloop, imd, imm, inm

2.10.33 inm - I2C デバイスのレジスタを対話的に何度も書き変える

文法： inm chip address[.0, .1, .2]

chip: I2C バス上のアドレス(16 進)

address: チップ内での開始アドレス(16 進)

[.0, .1, .2] :address のサイズ、.0 はアドレスがない場合、.1 は 1byte アドレス、
.2 は 2 バイトアドレスとしてアドレスを出力する。省略時は .1

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。

関連：nm, icrc32, iprobe, iloop, imd, imm, imw

2.10.34 iprobe - I2C バスのスキャン

文法：iprobe

I2C バスをスキャンして、みつかったチップアドレスを表示します。

関連：icrc32, iprobe, iloop, imd, imm, imw, inm

2.10.35 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 :-lt,<,-gt,>,-eq,==,-ne,!=,<>,-ge,>=,-le,<= 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

2.10.36 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 が設定されていなければ、00400000 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。ダウンロードする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 loady nfs tftpboot

2.10.37 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 loady nfs tftpboot

2.10.38 loady - シリアル経由でファイルのダウンロード(ymodem モード)

文法： loady [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 が設定されていなければ、00400000 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。ymodem プロトコルを使ってダウンロードする場合に使います。ダウンロードする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 loadb nfs tftpboot

2.10.39 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。
動作確認はしておりません。

2.10.40 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

2.10.41 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 count 個分アドレスがインクリメントされて実行されます。

例：

```
# Arm9# md.b 60000000 8
60000000: 43 52 55 53 11 00 00 ea    CRUS....
Arm9#
60000008: 14 f0 9f e5 14 f0 9f e5    ........
Arm9#
60000010: 14 f0 9f e5 14 f0 9f e5    ........
Arm9#
```

関連： base mm nm mw cmp cp crc32

2.10.42 mtdparts - フラッシュのパーティションの設定

パーティションの設定には、3 つの環境変数を使います。

partition： カレントパーティションのパーティション ID が設定されます。

パーティション ID は次のフォーマットです

<part-id> := <dev-id>,part_num

<dev-id> := <type><dev-num>

<type> := 'nand'|'nor'

<dev-num> := デバイス番号

本機では、<dev-num> デバイス番号は 0 のみが有効です。
本機では,<type>は nor のみが有効です
ですから、<dev-id> はいつも nor0 となります。
結果としてパーティション ID は、nor0,0 の様になります。

mtdids：Linux の MTD デバイス名と、u-boot のデバイス ID の対応付けを設定します。

mtdids=<idmap>[,<idmap>,...]
<idmap> := <dev-id>=<mtd-id>
<mtd-id> := Linux の MTD デバイス名称(tb0319)
本機では、mtdids は nor0=tb0319 のみが有効な設定です。

mtdparts：パーティションのリスト

LinuxMTD デバイスのコマンドラインパーティションの設定をする時と同じ内容です。

文法：mtdparts
現在の設定を表示する

文法：mtdparts delall
設定を削除する

文法：mtdparts del part-id
part-id: パーティション ID
指定したパーティションを削除する

文法： mtdparts add mtd-dev size[@offset] [name] [ro]
mtd-dev: nor0 | nand0
size: パーティションサイズ
offset:フラッシュの先頭からのオフセット
name:パーティション名称
ro:リードオンリーフラグ
指定したパーティションを追加する

文法：mtdparts default
出荷時の状態に戻す

これらのコマンドを利用すると、環境変数 partition、mtdparts,mtddevnum,mtddevname が更新されます。

2.10.43 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

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。そのままで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：

```
Arm9# md.w 00400020 4
00400020: 694c 756e 2d78 2e32    Linux-2.
Arm9# mm.w 00400020
00400020: 694c ? 594c
00400022: 756e ?
00400024: 2d78 ? -
00400022: 756e ? Arm9# <INTERRUPT>
Arm9# md.w 00400020 4
00400020: 594c 756e 2d78 2e32    LYnux-2.
Arm9#
```

関連： base md nm mw cmp cp crc32

2.10.44 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 00000000 になります。

end: 終了アドレス

省略時 01F00000 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、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 00000000 01F00000 55555555
Pattern 55555556  Writing...  Reading...
#
```

2.10.45 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

2.10.46 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 も設定されていなければ、00400000 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、/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 loady tftpboot

2.10.47 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

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。

GPIO のレジスタや IO ポートを操作する際には便利です。

2.10.48 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。

2.10.49 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。

2.10.50 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 start +size

protect op N:SF[-SL]

protect op bank N

protect op part-id

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)

size: start +size-1(16 進)の位置が含まれるイレースブロックまでという意味に
なります。

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。

part-id: パーティション ID(詳細は mtdparts の項を参照して下さい)

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

関連: erase

2.10.51 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、
00400000 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
はいけませんが、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

2.10.52 reset - CPU のリセット

文法：reset

再起動します。

2.10.53 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。

2.10.54 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。

2.10.55 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。

2.10.56 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)

2.10.57 sntp - SNTP プロトコルで RTC をあわせます

文法：sntp [ipaddr]

ipaddr: NTP サーバの IP アドレス

ipaddr が省略された場合、環境変数 ntpserverip で設定されたアドレスを参照します。
環境変数 timeoffset が設定されていると、設定した値分足されて設定されます。
日本時間を RTC に書き込む場合には、timeoffset に 32400 を設定します。

2.10.58 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。

2.10.59 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、00400000 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 " で括られていなくてはいけませんが、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

2.10.60 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。

2.11 特殊な意味を持つ環境変数の一覧

2.11.1 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。

2.11.2 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、設定されていなければ、tftpboot コマンドを実行します。

2.11.3 autoscript

値: yes

初期値: なし

loadb, loady, 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ければ、呼び出されません。

2.11.4 autostart

値: yes

初期値: なし

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ければ、呼び出されません。

2.11.5 baudrate

値: 9600 | 19200 | 38400 | 57600 | 115200

初期値: 115200

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。

2.11.6 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。

2.11.7 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)

2.11.8 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。

0 を指定すると即座に起動します。

0 を指定しても、シリアルにデータが来ていれば、コマンドラインに入れます。

2.11.9 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。

2.11.10 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。

2.11.11 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。

2.11.12 ethaddr

値: MAC アドレス

初期値: なし

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。

2.11.13 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。

2.11.14 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。

2.11.15 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。

2.11.16 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。

2.11.17 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。

2.11.18 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。

2.11.19 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。

2.11.20 mtddevname

値: 文字列

初期値: なし

カレントパーティションの名称が設定されます

2.11.21 mtddevnum

値: 数値

初期値: なし

カレントパーティションの番号が設定されます

2.11.22 mtdids

値: 文字列

初期値: nor0=tb0319

Linux の MTD デバイス名と、u-boot のデバイス ID の対応付けを設定します。

2.11.23 mtdparts

値: 文字列

初期値: tb0319:0x40000(uboot),0x40000(ubootenv),0x380000(kernel),
0x01c00000(root),-(config)

パーティションの設定を記述します。

2.11.24 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。

2.11.25 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。

2.11.26 ntpserverip

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。

sntp コマンドが参照するデフォルトのサーバの IP アドレスに利用されます。

2.11.27 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。

2.11.28 partition

値: パーティション ID

初期値: なし

カレントパーティションが設定されます。

2.11.29 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 mtdblock1" \

"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。

2.11.30 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。

2.11.31 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, bootp, dhcp で上書きされたりします。

2.11.32 stdin

値: lcd | serial

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。

2.11.33 stdout

値: lcd | serial

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。

2.11.34 stderr

値: lcd | serial

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。

2.11.35 timeoffset

値: 数値

初期値: 無し

RTC にローカルタイムで設定したい場合、UTC 時刻との差を秒で設定します。

sntp コマンドが使用します。

2.11.36 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。

2.11.37 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. 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 と連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変えを容易にする
- 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 ローカルなディスクから起動するには

現在の TB0319 用の u-boot では、USB のディスク等に直接アクセスする機能はもっていません。出来ることは、Linux カーネルを、内蔵フラッシュ、もしくはネットワークから読みこむ事だけです。ローカルなディスクをルートファイルシステムとして利用するには、3 つの方法があります。

1. 利用したいデバイスをカーネル組みこみにする。
   ハードウェア構成が決まっている場合には、これが一番簡単です。
2. イニシャルラムディスクを使う

   Linux のイニシャルラムディスクの機能は、ルートファイルシステムを認識するのに必要なドライバをロードする為にあるます。イニシャルラムディスクで使うルートファイルシステムを別途用意する必要があり、余分にフラッシュの容量が必要です。
3. /sbin/init で処理する

   最近の Linux は、イニシャルラムディスクを使わなくても、普通に起動した後で、ルートファイルシステムを変更する機能を持っています。

しかし、元のファイルシステムを解放するには、元のファイルシステムを使っている全てのプロセスを再起動しなくてはいけません。/sbin/init をシェルスクリプトに置き換えて、本当にマウントしたいファイルシステムをマウントし、その中にある、/sbin/init を起動するスクリプトを用意することで、これが実現できます。

フラッシュ内部の/sbin/init では、この方法で実現してあります。

3.1.2 サンプルの/sbin/init の機能

フラッシュ内の /sbn/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 rootdev 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、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、フラッシュから起動する

/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

- rootdev

  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

  ", " 区切りで複数指定でき、先に認識できたものが使われます。

  指定がない場合や、既にマウントされている、もしくは、先に認識できたデバイスが正しくマウント出来なかった場合には、現在のルートデバイスから起動します。

  デバイスを認識するのに必要なドライバは、後述の devorder で指定されるか、カーネル組みこみにする必要があります。

- fstype

  ルートファイルシステムで使われているファイルシステムのタイプを指定します

  ", "区切りで複数指定でき、指定された順に成功するまで試します。

  指定がない場合、ext3,ext2,vfat を指定したのと同等の処理をします。

- devorder

  ロードするデバイスドライバと、ロードする順番を決定します

  ", "区切りで複数指定でき、指定された順に読みこみます。モジュール名も指定可能ですので、他に必要なものがあれば、追加しておけば、自動的にロードされます。

  指定がない場合、ide,cf,usb,1394 を指定したのと同等の処理をします。

- timeout

  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。

  USB ディスク等、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、rootdev 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 timeout 回再試行するという意味ですので、長めに設定する事をお薦めします。

- 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 3：最初の USB ディスク のパーティション 2 に ext2 で用意したルートファイルシステムから起動したい場合

```
setenv rootdev sda1
setenv devorder usb
setenv fstype ext2
```

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、/home/arm9/ 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

があると前堤して説明します。

> 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

/home/arm9 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:/home/arm9/
```

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 nfsroot 192.168.3.91:/home/arm9/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 $mtdparts'
```

とします。

nfsargs の行は、実際は一行で、各行の間にはスペースが一つはいります。

> 注意：
>
> 2007/04/14 日版より、 nfsargs はこの設定をデフォルトに変更しましたが、それ以前の版から新しい u-boot にアップデートを行なっても、以前の値が残ります。

最後に、saveenv を実行して、変更を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

setenv load_method tftp

saveenv

とします。

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ないます。
   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ないます。
   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。

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 2 つです。

1. run cram
   フラッシュから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。
   環境変数の設定には、cramargs、userargs が実行されます
2. 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

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 cram'
```

とすると、run nfs 起動を試して、失敗したら、run cram 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:/home/arm9/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:/home/arm9/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 uboot_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_kernel

  kernel 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:/home/arm9/uImage 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 kernel_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_conf

  config 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。
- 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一か検査します。