U-BOOT
ユーザーズマニュアル

MP201EK 編

2009.06.11

メディアラボ株式会社

目次

1. はじめに

1.1 前提

このマニュアルは、MP201EK ボードで、弊社で選択したコンフィグレーションオプション（設定）の場合での動作についてのみ詳細に記したものです。コンフィグレーション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。
また、弊社で修正を行った部分もありますので、オリジナルの u-boot と動作が異なる部分もあります。あらかじめ御了承ください。
バグ修正以外の変更部分については明記してあります。

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。実際に入力する部分はボールドになっています。

2. u-boot 概略

2.1 u-boot について

u-boot はブートローダです。ブートローダとは、ハードウェアを適切に初期化し、OS をメモリ上に展開し、起動するのがその役目です。
AT 互換機では、BIOS がハードウェアを適切に初期化する部分と、統一的なデバイスのアクセス方法を提供する部分を担い、LILO や grub といったものが、BIOS の機能を利用し OS を起動する為の補足的な作業を担当します。u-boot では BIOS と grub の機能を合わせた部分を全て担当します。
u-boot には rom モニタ的要素もあり、開発をする際のデバイスを理解するための実験から、インストール作業、アップデート作業、出荷検査、不良品発生時の故障解析までさまざまな場面で役立ちます。

2.2 シリアルの接続

2.2.1 概要

u-boot はシリアルからコマンドを入力する事で操作します。
シリアルの初期設定は

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とシリアル通信を行うためのターミナルソフトウェアが必要です。
ターミナルソフトには何も使っても構いませんが代表的なものをあげると、Linux では、C-kermit(お薦めです)、gkermit、minicom、cu 等があり、Windows では TeraTerm(お薦めです)、ハイパーターミナル(OS 付属) 等があります。

ターゲット機器とシリアルクロスケーブルでパソコンに接続し、適切に設定されたターミナルソフトを起動してから、ターゲット機器の電源をいれます。
以下の様に表示され、5 秒のカウントダウンが始まりますので、そこでリターンキー等何か入力すると、u-boot のコマンドプロンプトに入れます。何も入力がないと、

自動で Linux の起動が始まります。

```
U-Boot 2009.01-00320-g6d25633 (Jun 12 2009 - 23:50:07)

CPU:  MP201@250MHz Rev4 BootROM:ver2
HPLL: 250MHz LPLL: 0MHz
FPGA:  ver3101
I2C:   ready
DRAM:  128 MB
Flash: 16 MB
In:    mp201
Out:   mp201
Err:   mp201
Hit any key to stop autoboot:  0
MP201#
```

これで、u-boot を対話的に操作する準備が出来ました。

ターミナルソフトと、u-boot のシリアル経由でのファイルのダウンロードコマンドには相性がありますので、簡単にそれぞれの設定方法と、推奨する u-boot のダウンロードコマンドの説明をしておきます。

2.2.2 Linux から C-kermit を使う場合

C-kermit は ckermit パッケージに含まれています。
以下は ttyS0(com1)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 u-boot の loadb コマンドを利用します。

```
# 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 modem type none
(/usr/src/mldbox/) C-Kermit>set line /dev/ttyS0
(/usr/src/mldbox/) C-Kermit>set flow none
```

```
(/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 prefixing all
(/usr/src/mldbox/) C-Kermit>set parity none
(/usr/src/mldbox/) C-Kermit>set escape 27
(/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 0x80400000 at 115200 bp
s...
                            <=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 scre
  en.
```

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 modem type none
  set line /dev/ttyS1
  set flow none
  set speed 115200
  set serial 8n1
  set carrier-watch off
  set prefixing all
  set parity none
  connect
  #
```

2.2.3 Linux から cu を使う場合

cu コマンドは Debian では cu パッケージ、redhat 系では uucp パッケージに含まれています。また、lrzsz パッケージをいれておくことで、バイナリファイルの送信が出

来ます。
以下は ttyS0 にケーブルを接続したと仮定して説明します。別のシリアルを使う場合には、ttyS0 を ttyS1 や ttyUSB0 等と読みかえてください。
Debian の場合はパッケージを入れるだけで使えます。
Redhat 系では、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 が正しく動く様になれば、以下のコマンドで接続できます。

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

cu では、行頭(改行の次)の位置で ~ (チルダ)を入力すると特別な意味にとられ、その次の文字で処理が決まります。
例えば~? と入力するとヘルプメニューが表示されます。
ファイルを送信するには、loady コマンドを利用します。

```
# loady
## Ready for binary (ymodem) download to 0x80400000 at 115200 bp
s...
C~+sz u-boot.bin
Sending: u-boot.bin
Bytes Sent: 251008   BPS:7543
Sending:
Ymodem sectors/kbytes sent:   0/ 0k
Transfer complete
N) packets, 4 retries
## Total Size      = 0x0003d434 = 250932 Bytes
```

の様にします。u-boot.bin と入力している所で、C の文字がみえますが、これは u-boot が数秒間隔で送って来ている文字ですので、コマンドラインの見た目は崩れま

すが気にしないで入力します。また最後の 2 行は、sz コマンドの出力と u-boot の出力が混ざる場合がありますので、文字化けを起こす場合がありますが、
cu を終了するには、~. を入力します。ただし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。

2.2.4　Linux から minicom を使う場合

minicom パッケージ以外に、lrzsz パッケージも必要です。
まず、minicom の設定を行います。（以下は Debian Etch で試しました、環境によっては、デフォルト設定が違うかもしれません。）日本語だと、メニューが崩れる場合があるようですので、以下は、設定は英語になるようにして起動しています。
$ **LANG= minicom -s**
とすると、設定メニューに入れます。
Serial port setup を選択し、
　　A を入力して Serial Device を/dev/ttyS0 に変更
　　F を入力して Hardware Flow Control を No に変更
　　リターンキーを入力して元の画面に戻り、
Modem and dialing を選択し、
　　A を入力して、Init string を空にする。
　　R を入力して、Modem has DCD line を No にする。
　　リターンキーを入力して元の画面に戻り、
Save setup as dfl を選択して保存します。
Exit from Minicom を選択して終了します。

接続を行います。
$ minicom
でつながります。
ファイルを送信するには、u-boot で、loady リターンといれてから、^AS で zmodem を選択し(xmodem/ymodem/zmodem のどれでもよい)、次にファイルを一つ選択します。(カーソルをファイル名の位置にあわせて、スペースキーで選択、ディレクトリはスペースキー 2 回で移動、もしくは選択せずにリターンキーでファイル名を直接入力します。)

2.2.5　Windows から Tera Term を使う場合

Windows で利用する場合のお薦めは、Tera Term (http://hp.vector.co.jp/authors/VA002416/) です。ファイルの送信には、u-boot の loady コマンドを使って、Tera Term から Xmodem でファイルを送信します。

Tera Term の kermit モードと zmodem モードは u-boot とは互換性がありません。

2.2.6 Windows からハイパーターミナルを使う場合

Windows95～XP では OS 付属のハイパーターミナルが使えます(Vista にはありません)。バイナリファイルの転送には XMODEM、YMODEM、kermit が利用できます。
u-boot の loady コマンドで待機させ、ハイパーターミナルから、XMODEM　もしくは、YMODEM でファイルを送信します。
もくは、u-boot の loadb コマンドで待機させ、ハイパーターミナルから、kermit でファイルを送信します。

2.3 u-boot を使ってみる

help と入れると、利用可能なコマンドが一覧表示されます。

```
# help
?       - alias for 'help'
askenv  - get environment variables from stdin
autoscr - run script from memory
base    - print or set address offset
.....省略....
```

help の引数にコマンド名を付けると、より詳細なコマンドの使いかたが表示されます。

```
# help run
run var [...]
    - run the commands in the environment variable(s) 'var'
#
```

コマンドラインからは、TAB を使って文字列の補間が行えます。

例えば、以下の例の様に p だけ打って TAB を入力すると、p で始まるコマンドが一覧され pri まで入力して、TAB を入力することで printenv と補間されます。

さらに、load_kernel まで打った所で TAB を入力すると、load_kernel で始まる環境変数の候補が一覧されます。

```
# p
  protect ping printenv
# printenv load_kernel
  load_kernel_cram load_kernel_tftp load_kernel_nfs load_kernel_se
rial
  load_kernel
# printenv load_kernel
load_kernel=run load_kernel_$load_method&& iminfo $loadaddr
#
```

2.4 環境変数

u-boot の環境変数には 3 つの目的があります。

- u-boot の動作に関係する環境変数

  シリアルのボーレートを設定する baudrate、自分の IP アドレスを設定する ipaddr 等たくさんあります。

  例えば、シリアルのボーレートを変更するには、

  ```
  # setenv baudrate 9600
  ## Switch baudrate to 9600 bps and press ENTER ...
  ```

  とします。このあと、ターミナルソフトのボーレートを変更して、再接続すると、継続できます。この設定を保存(saveenv コマンド)しておくと、次回電源投入時は、最初から 9600 ボーになりますが、保存しなければ元のボーレートで起動してきます。

  また、環境変数 ipaadr は、ネットワーク系のコマンドが利用する自ホストの IP アドレスになります。

  これらの環境変数の名称は固定です。

- コマンドの実行結果に基づいて設定される環境変数

  ファイルをロードするコマンドでは、ロードしたサイズを filesize という名前の環境変数に設定してきます。

  dhcp コマンドは、ipaddr 等ネットワーク系のコマンドが利用する環境変数を変更してきます。

  これらの環境変数の名称は固定です。

- 上記以外

  上記以外の名前の環境変数はユーザが自由に追加できます。u-boot では、コマンド列を環境変数にとっておいて、それを run コマンドで実行する機能がありますので、色々な起動方法や設定をとっておけます。

  こんな使いかたが出来ますというサンプルを弊社でコンパイル時のデフォルトとしていれてあります。

  run コマンドの中から run コマンドを使う事も出来ますので、目的別にシンプルなコマンド列を複数定義しておいて、組み合わせて利用する事もできます。

環境変数の一覧は、printenv コマンドで一覧出来ます。また、printenv コマンドに引数を渡せば、個別の環境変数のみが表示されます。

環境変数は saveenv コマンドを実行したときにその時の状態が保存されますので、saveenv を実行しなければ、次回の起動時には元の値に戻ります。

また、いくつかのコマンドは実行すると環境変数を設定してきます。例えば、dhcp コマンドを使うと IP アドレス等を環境変数に設定してきますので、その後に saveenv コマンドを実行すると、次回起動時にも自動取得した設定が残ってしまいます。もちろん dhcp コマンドを実行しなおせば上書きされるので問題ありませんが、dhcp コマンドの実行を忘れると、IP アドレスが重複する等の問題を起こす場合がありますので気を付けてください。

環境変数を削除するには、

```
# setenv ipaddr
```

の様に、値を指定しないで　setenv コマンドを使うとその環境変数自体が無くなります。

環境変数を作成する際は起動直後に設定し、すぐに保存するという手順にしておいた方が安全です。

サンプルのスクリプトを理解する

- ファイルを読み込む為のスクリプト

  ファイルを読み込む方法として、以下のサンプルを準備しました。

  - フラッシュ上に保存されているルートファイルの中から読み込む。
  - ネットワーク越しに、tftp プロトコロルで読み込む
  - ネットワーク越しに、nfs プロトコロルで読み込む
  - シリアル経由で読み込む

  です。例えばカーネルをメモリに読み込むコマンド例として、上記の順に、load_kernel_cram load_kernel_tftp load_kernel_nfs load_kernel_serial が設定されています。

  また、環境変数 load_kernel には、run load_kernel_$load_method ... と設定されていますので、自分がよく使う方法(nfs 等)を環境変数 load_method に設定しておく事で、run load_kernel (または load_uboot, load_cram)とすると、$load_method の部分が展開され、目的のスクリプトが起動されます。

  カーネル以外にも、u-boot を読み込むための load_uboot_XXX や、ルートファイルシステムイメージを読み込む load_cram_XXX 等が設定されています。

- Linux を起動する為の環境変数

  いくつかの起動方法のサンプルを用意しました。用意した以外にも色々な方法で起動出来ますので、状況にあわせて追加／修正してください。

名称	カーネル取得場所	マウントするルートファイルシステム
flash	フラッシュ	フラッシュ
cram	フラッシュのルートファイルシステム内部	フラッシュ
disk	フラッシュ	IDE の ext2
nfs	NFS	NFS

カーネルパラメータは u-boot 環境変数 bootargs に設定する事で渡せます。

カーネルをメモリ上にロードした後、bootm コマンドで起動しますが、bootm コマンド実行時の bootargs の内容が、Linux に渡ります。

cramargs では、bootargs をフラッシュ起動に合わせたパラメータに変更します。

同様に、nfsargs で NFS ルートの設定、diskargs でディスク起動に合わせた設定を行っています。userargs は、どの起動方法でも実行されますので、コンソールの設定等の共通事項を記述してあります。

例えば、cram は、if run load_kernel_cram; then run cramargs userargs;bootm;fi

となっており、意味は、load_kernel_cram が成功したら、cramargs と userargs を実行し、kernel を起動するです。

例えば、コンパイルしなおした新しいカーネルを使って、フラッシュをルートファイルシステムとして起動したい等のバリエーションも、

```
setenv test 'if run load_kernel_nfs;then run cramargs userar
gs &&bootm;fi'
```

等の様に簡単に記述できます。

環境変数 bootcmd には、電源投入時に自動起動するコマンドをいれておきます。

```
setenv bootcmd “run nfs”
```

とすると、電源投入時、nfs が実行されますし、

```
setenv bootorder ”cram nfs flash”
setenv bootcmd 'for i in $bootorder; do run $i; done'
```

としておくと、cram、nfs、flash の順に成功するまで試すといった設定も可能になります。

- フラッシュを書き換える(更新する)為の環境変数
  フラッシュの書き換えは失敗すると二度と起動できなくなる可能性がありますので注意してください。

  update_ で始まるスクリプトは、フラッシュ内部を書き換える為のスクリプトです。また、verify_で始まるスクリプトは、フラッシュ内部の内容が、同一

であるかを確認するスクリプトです。init_で始まるものはその領域を消すためのスクリプトです。

update_ で始まるスクリプトは、ファイルをダウンロード（load_XX を実行）し、サイズが適切であれば、該当エリアを消去（init_XX を実行）し、書き込みをおこない、再度ファイルをダウンロードして、内容が同一かどうかを確認します（verify_XX を実行）。

これらのスクリプトを正しく動作させるには、以下の設定が必要です。

固定 IP で運用する場合

ipaddr, netmask, serverip,gatewayip を設定します。

gatewayip はなくてもかまいません。

serverip はファイルサーバの IP アドレスを指定します。

例:

```
setenv gatewayip 192.168.3.1
setenv netmask 255.255.255.0
setenv ipaddr 192.168.3.243
setenv serverip 192.168.3.91
```

NFS を使う場合

nfsbase に、u-boot からダウンロードしたいファイルのおいてある場所を指定します。パスの最後は必ず / を付けてください。

nfsroot にカーネルに渡すルートファイルシステムの場所を設定します。

例:

```
setenv nfsbase 192.168.3.91:/home/mld/
setenv nfsroot 192.168.3.91:/home/mld/rom
```

サーバ側の/etc/exports には、

/home/mld *(rw,async,no_root_squash,no_all_squash,insecure,no_subtree_check)

の様に指定します。セキュリティを考慮した書き方ではありません。

カスタマイズ

bootfile	カーネルのファイル名を設定
ubootfile	u-boot のファイル名を設定
cramfile	ルートファイルシステムイメージのファイル名を設定

その他 ethprime、netretry、partition も変更すると便利になるかもしれません。

これらは、特殊な意味をもつ環境変数の説明の章を参照してください。

環境変数のリセット

環境変数のデフォルト値はコンパイル時に組み込まれます。

```
# run init_env
```

とすると、環境変数を保存してあるエリアがクリアされますので、再度 u-boot を起動しなおす（電源を投入しなおすが、reset コマンドを入力）と、デフォルトの値に戻ります。保存してある環境変数が読み込めずに、デフォルトに戻った場合には、起動時に

*** Warning - bad CRC, using default environment

という出力があるはずです。

コンパイル時のデフォルトは、出荷時のデフォルトとは多少違う場合があります。

2.5 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別なコマンドが u-boot にあるわけではありません。cp コマンド（メモリ間のデータ転送）や、ファイルをロードするコマンド等、一部のコマンドが書き込み先のアドレスを元にフラッシュへの書き込みかどうかを検査する事で対応しています。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 DRAM 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といけませんし、イレースを実行する前にプロテクトを外さなくてはいけません。

先にイレースしなければならない以上、安全に書き換えるには書き換えたいイメージを一旦 DRAM 上にダウンロードして、正しくダウンロードできたのを確認してからフラッシュを消去し、cp コマンドで書きこむ様にすることをお薦めします。

弊社では、cp コマンド以外でのフラッシュへの書き込みの動作検証はしておりません。

> 本ボードでの u-boot のプロテクトコマンドは、実際にはフラッシュのプロテクト機能を利用していません。間違えて書いたりしないように、ソフトウエアで管理しているだけです。

2.6 例外ハンドラ

u-boot では、割り込みや例外を利用しません。

不正なアドレス範囲へのアクセスやアライメントのとれていないアクセスを行うと、例外ハンドラに飛んでしまいます。

例外ハンドラに飛んでくると、CPU のレジスタダンプと共にその旨表示されます。

2.7 コンソールの変更

シリアルに別の機器をつなげて使いたい場合等には、コンソールを別のシリアルに出したり、使えなくする事も出来ます。

利用できるデバイスの一覧が coninfo コマンドでみれます。

```
# coninfo
List of available devices:
serial   80000003 SIO
nulldev  80000003 SIO
mp201    00000003 .IO stdin stdout stderr
eserial1 00000003 .IO
eserial0 00000003 .IO
#
```

変更する場合には、環境変数 stdin, stdout, stderr に、デバイス名を設定します。設定すると同時に切り替わりますので、変更先にもシリアルケーブルを繋げて行うのが安全です。

serial は設定が壊れたときの為のデフォルトデバイスを意味し、mp201 と同じです。

nulldev はダミーのデバイスで、コンソールを殺したい時に使います。

環境変数を初期化した直後等では、標準入出力が serial になっています。その場合、一回 stdin stdout stderr を mp201 に設定してから、別のデバイスに変更する様にしてください。（多分 u-boot のバグですが、切り替えられない場合があります。）

入力だけを無効にして、出力は残しておく設定も可能です。

3. メモリマップ

u-boot を使いこなすには、メモリマップを正しく理解しておく必要があります。

128MB の SDRAM、16MB のフラッシュメモリを実装しています。

開始	終了	用途
00000000	00ffffff	フラッシュメモリ
30000000	37ffffff	SDRAM

3.1.1 DRAM エリア　　0x30000000　　から 0x37FFFFFF

開始	終了	用途
30000000		空き
	37f4ff80	u-boot スタック
37f50000	37f7ffff	u-boot malloc 領域(192k)
37f80000	3fffffff	u-boot 本体

3.1.2 フラッシュエリア　　0x00000000　　から 0x00FFFFFF

フラッシュエリアは以下の様に分けています。

番号	名称	用途	開始アドレス	サイズ
0	uboot	u-boot 本体	00000000	256K bytes
1	ubootenv	u-boot 環境変数保存領域	00040000	128K bytes
2	kernel	カーネル保存領域	00060000	1.6M bytes
3	root	ルートファイルシステム	00200000	14M bytes

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。

搭載されているフラッシュには 128k バイトのイレースブロックが 128 個あります。

- 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せません。
- u-boot の環境変数保存エリアは、u-boot 本体の末尾の未使用な領域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でるため、別の領域に分ける事とし、u-boot 本体の次の 1 ブロックを割り当てました。
- 残った部分は利用方法にあわせて自由に変更できます。

パーティション範囲全体をイレースすると、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。パーティションはイレースブロック単位で作成します。フラッシュのイレースブロックは、flinfo コマンドで確認できます。また、環境変数 mtdparts を直接変更、もしくは、mtdparts コマンドを利用する事で、領域を変更でき、その変更はカーネルのコマンドラインパラメータを通して、Linux にも反映されます。

4. 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(例えば Linux カーネルや、ラムディスクイメージ等)は、mkimage というコマンドで u-boot 用のフォーマットに変換して利用します。
このフォーマットに変換しておくことで、ロードアドレス、開始アドレスを埋め込めます。また、名前をつけたり作成日の確認ができますので、ファイルシステムを利用せずにフラッシュ内に置いておく場合に特に役にたちます。
他にも、圧縮をサポートしている、チェックサムの確認ができる、ファイルの種類を指定できるので、間違いが起きにくい等の利点があります。
mkimage は u-boot のソースの tools の下にあります。
Debian では、uboot-mkimage　というパッケージになっています。

4.1　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 | nios | nios2 | ppc | s390 | sh | sparc | sparc64 | blackfin | avr32

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 | flat_dt
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。
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: 展開先アドレスを指定します。
bootm コマンドは、このアドレスに中身を展開します。

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に指定します。

注意：この部分少し古いです。ソースの中の方にはオプションが追加されているようです。

4.2 u-boot 用の Linux カーネルの作り方

mkimage がパスの通った所においてあれば、

Linux 側のソースツリーで、make uImage とする事で u-boot 用のバイナリが作成されます。

4.3 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 arm -O u-boot -T script -C none -n
"hello" -d script script.img
Image Name:   hello
Created:      Sat Dec  4 16:56:14 2004
Image Type:   ARM U-Boot Script (uncompressed)
Data Size:    19 Bytes = 0.02 kB = 0.00 MB
Load Address: 0x00000000
Entry Point:  0x00000000
Contents:
   Image 0:       11 Bytes =    0 kB = 0 MB
$
```

ここで出来た script.img 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。

スクリプトに対する圧縮はサポートされておりません。また、-A arm と-C none はメモとして入れてあるだけで、autoscr コマンドはこれらをチェックしません。

実行するには、autoscr を使います。

```
# nfs 192.168.3.91:/home/mld/script.img
smc911x: initializing
smc911x: detected LAN9218 controller
smc911x: phy initialized
smc911x: MAC 00:0a:a3:02:0b:0e
File transfer via NFS from server 192.168.3.91; our IP address is
192.168.3.234
Filename '/home/mld/script.img'.
Load address: 00x31100000
Loading: #
done
Bytes transferred = 83 (53 hex)
# autoscr
## Executing script at 31100000
hello
#
```

4.3.1 u-boot用アプリケーションを作る

いくつかの 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 examples/hello_world を実行するには、

```
# nfs 192.168.3.91:/home/ito/MP201/build/u-boot.git/examples/hello
_world
smc911x: initializing
smc911x: detected LAN9218 controller
smc911x: phy initialized
smc911x: MAC 00:0a:a3:02:0b:0e
File transfer via NFS from server 192.168.3.91; our IP address is
192.168.3.250
Filename '/home/ito/MP201/build/u-boot.git/examples/hello_world'.
Load address: 0x31100000
Loading: ##
done
Bytes transferred = 9364 (2494 hex)
MP201# bootelf
Loading .text @ 0x30001000 (400 bytes)
Loading .rodata @ 0x30001190 (138 bytes)
## Starting application at 0x30001000 ...
Example expects ABI version 4
Actual U-Boot ABI version 4
Hello World
argc = 0
argv[0] = "<NULL>"
```

```
  Hit any key to exit ...

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

引数も渡せますが、最初の引数は ELF ファイルの場所を示しますので、ダウンロードしたアドレスを指定しなければいけなく、2 つ目以降の引数はアプリケーションで利用できます。

> 注意：2 つ以上の引数が渡せる部分は、メディアラボの拡張です。

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 を利用する事で、u-boot のグローバルデータ(struct global_data)に、変数 gd を介してアクセス出来ます。

```
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 4 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
                                            i386 もしくは PowerPC でのみ利用可
void free_hdlr(int);                        割り込みハンドラの削除
                                            i386 もしくは PowerPC でのみ利用可
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, uint, int , uchar* , int); i2C バスにつながっている、EEPROM と RTC への書き込み

int i2c_read (uchar, uint, int , uchar* , int); i2C バスにつながっている、EEPROM と RTC の読み込み

ABI 3 で追加された関数

unsigned long simple_strtoul(const char *cp,char **endp,unsigned int base);

libc での strtoul とほぼ同じ（先頭の空白は許さない）

char *getenv (char *name); 環境変数の取得

void setenv (char *varname, char *varvalue); 環境変数の設定

ABI 4 で追加された関数

long simple_strtol(const char *cp,char **endp,unsigned int base);

libc での strtol とほぼ同じ（先頭の空白は許さない）

int strcmp(const char * cs,const char * ct); libc と同じ

int ustrtoul(const char *cp, char **endp, unsigned int base);

strtoul とほぼ同じですが、G,M,K,k が最後についた場合にそれぞれ、2^30,2^20,2^10 をかけた値が返ります

グローバルデータの構造は、アーキテクチャによって変わります。

struct bd_info と、struct global_data の内容が参照できます。それぞれの定義が、u-boot ソースの include/asm-arm/u-boot.h と include/asm-arm/global_data.h にありますので、ソースを参照してください。

5. u-boot 詳細

5.1 コマンドライン(Hush パーサ)

弊社で作成した u-boot は全て、

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。

5.1.1 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。

5.1.2 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

単語のリストー 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。

5.1.3 クオート処理とエスケープ文字

$; ” '等の特殊な文字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。

5.1.4 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。

いくつかのコマンドは、再実行の際に引数が自動インクリメントされます。

例えば、メモリの内容をダンプするコマンド md では、

```
# md.l 0 4
00000000: ea000012 e59ff014 e59ff014 e59ff014    ................
#
00000010: e59ff014 e59ff014 e59ff014 e59ff014    ................
#
00000020: 37f80120 37f80180 37f801e0 37f80240     ..7...7...7@..7
```

の様に、改行を入力するたびに、ダンプするアドレスがインクリメントされます。

^C した後でも、改行を入力してしまいますと、最後に実行したコマンドが再実行されますので注意して下さい。

5.1.5 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を入力したあと 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。
printenv 等、環境変数がくるはずの場所では、引数も補間できます。

5.1.6 ヒストリーと、コマンドライン編集機能

ヒストリーやコマンドライン文字列中を移動し、行の編集が出来ます。
^P という表記はコントロールキーを押しながら p を入力という意味です。

- ^P もしくは上矢印キー
  コマンドヒストリの前の行を表示します。
- ^N もしくは下矢印キー
  コマンドヒストリの次の行を表示します。
- ^B もしくは左矢印キー
  カーソル位置を一文字前に移動します
- ^F もしくは右矢印キー
  カーソル位置を一文字後ろ移動します
- ^A もしくは Home キー
  カーソル位置を行の先頭に移動します
- ^E
  カーソル位置を行末に移動します
- ^H もしくは バックスペースキー
  カーソル位置の一つ前の文字を削除します
- ^D もしくは DEL キー
  カーソル位置の文字を一つ消します
- ^K
  カーソル位置から行末までを削除します
- ^O
  上書きモード、挿入モードをトグルします
- ^X もしくは ^U
  現在の入力行を全削除します

5.1.7 ^C による中断

Ctrl-C で処理の中断が出来ますが、これは、各コマンドが中断すべきかどうかをチェックしているタイミングで行われますので、停止までに時間がかかったり、中断できないコマンドもあります。例えば、erase コマンドは処理中には止められません。スクリプトは中断できますので、スクリプトから erase コマンドを実行している途中で^C した場合、erase コマンドが終了してから処理を停止します。

5.2 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。u-boot ソースの 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。

> 応用:次回リブート時に、1 回だけ普段とは別の方法で起動する例
>
> 例えば、bootcmd（起動時に自動実行される内容を記述する環境変数です）が、run disk だとして、次回起動時だけ run cram で起動してみて、仮りに起動に失敗しても、電源を入れ直せば、元に戻る様にしておきたい場合の例です。
>
> Linux で、
>
> # fw_setenv bootcmd 'setenv bootcmd run disk;saveenv;run cram
>
> として再起動すると bootcmd を元に戻して保存してから実行しますので、次の一回だけ起動方法を変えるという事が実現できます。フラッシュの更新を u-boot から行いたい場合等にも応用できるかと思います。

5.3 u-boot のカスタマイズ

u-boot はブートローダですが、ブートローダに期待される項目には、一般に

- 小さくて、高速なものが望ましい
- 柔軟性もあるに越した事はない。

というある意味矛盾した要求があります。
u-boot は必要に応じて機能を増減できます。必要最小限の機能のみを組み込む事で、より小さくもできますし、高機能にする事もできます。

コンパイルし直すには、クロスコンパイラにパスを通して、

```
$ make mp201ek_config
$ make
```

とします。

u-boot の機能の設定は、u-boot ソースの include/configs/mp201ek.h で行います。

5.4 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。

> この章は、古いソース(1.1)で、便利と思われるコマンドの動作について、ソースコードを読んで、詳細な動作を解析したものがベースになっております。解析をしていないコマンドも残っていますし、バージョンが新しくなって、変わった事に気づいた部分と、旧バージョンのマニュアルでの間違いは更新してありますが、全てのコマンドの再確認をおこなったわけではありません。あらかじめ御了承ください。

5.4.1 ? - '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

5.4.2 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

5.4.3 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、31100000 です。
スクリプトは、mkimage でタイプを script に指定して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。
スクリプトの内容は、一度 u-boot の malloc 領域にコピーされてから実行されます。スクリプト内部で DRAM を利用する場合に、スクリプトがダウンロードされた場所を意識して作成する必要はありません。

関連：　　run

5.4.4　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

5.4.5　bdinfo　- ボードの情報を表示

文法：　　bdinfo

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。

5.4.6　boot、bootd　- 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が、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。

5.4.7 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 [args...]]

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

args: プログラムに渡す追加引数(メディアラボの拡張仕様)

ELF ファイルを起動します。

関連： bootm go

5.4.8 bootm - OS とアプリケーションの展開と起動

文法： bootm [addr [arg1 [arg2 ...]]]
bootm start [addr [arg1 [arg2 ...]]]
bootm loados
bootm go

addr: mkimage でタイプに kernel、multi、standalone を指定して作成した u-boot 用のバイナリファイ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。

arg1: 起動するバイナリファイルによって解釈が異なります。
タイプが standalone の場合、mkimage で指定した展開先アドレスをここで指定したアドレスに変更します。
タイプが kernel の場合、指定した os タイプによって、さらに解釈が異なります。例えばタイプが kernel で、os が linux の場合、mkimage で作成したイニシャルラムディスクイメージがおいてある場所のアドレスという意味になり arg2 以降は無視されます。

arg2: 起動するバイナリファイルによって解釈が異なります。
タイプが standalone の場合、アプリケーションに渡されます。

'start' | 'loados' | 'go'
最初の起動方法（例えば引数なしの bootm コマンド）が実行する一連の内容をより細かなステージに分割して、各ステージのみを実行します。

mkimage でタイプに kernel、multi、standalone を指定して作成した u-boot 用のバイナリファイルを指定した圧縮方法に従って DRAM に展開し実行します。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。
タイプが standalone で、環境変数 autostart が no の場合、展開までを行い、環境変数 filesize を展開後のサイズで更新し終了します。
mkimage で作られたファイルには、CPU タイプが必ず指定されますが、それがあわないと何もしません。
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。

各ステージに分割した、2 番目以降の起動方法は、各ステージのみを実行します。ステージは以下の様に分割されています。

- start
  分割ステージの開始、引数の解釈のみ行います。
  過去に行った分割ステージの状態をリセットします
- loados
  uImage をメモリに展開する部分を行います。
- bdt
  ボード固有な部分の処理を行いますが、Linux-arm 版にこの処理はありませんので、エラーになります。
- cmdline
  OS に渡すコマンドライン関連の処理を行いますが、Linux-arm 版にはこの処理はありませんので、エラーになります。
- prep
  OS 起動の前準備の処理を行いますが、Linux-arm 版にはこの処理はありませんので、エラーになります。
- go
  Os 起動

例：

```
# bootm start 60000
## Booting kernel from Legacy Image at 00060000 ...
   Image Name:   Linux-2.6.29-rc8-mlb-00213-gc784
   Created:      2009-05-11  12:54:06 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1417936 Bytes =  1.4 MB
   Load Address: 30008000
   Entry Point:  30008000
   Verifying Checksum ... OK
# bootm loados
```

```
   Loading Kernel Image ... OK
OK
# bootm go

Starting kernel ...
```

関連：　bootelf bootvx iminfo

注意：現在の版では、u-boot 用アプリケーション（standalone）が認識しなくなっています。いずれ直します。

5.4.9 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

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。

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

5.4.10 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

5.4.11　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 mtdparts

5.4.12　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。

5.4.13　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。
先頭から、デバイス名、フラグの値、フラグを文字で表記したもの、現在の
標準入出力に設定されているデバイスの場合、stdin、stdout,stderr がそれぞれ
表示されます。

デバイス名

- serial
  仮想的なデバイスです。過去互換の為であり、デフォルトを意味します。
- Nulldev
  仮想的なデバイスです。ここへの出力は捨てられますし、ここからの入力は何も来ません。
- Mp201
  CPU 内臓シリアルデバイス（DSUB-9)
- eserial0
  外付けのシリアルデバイス(UART1)
- eserial1
  外付けのシリアルデバイス(UART2)

フラグ

- S　　システムデバイス
- I　　入力に設定する事が出来るデバイス
- O　　出力に設定する事が出来るデバイス

例：

```
# coninfo
List of available devices:
serial   80000003 SIO stdin stdout stderr
nulldev  80000003 SIO
mp201    00000003 .IO
eserial1 00000003 .IO
eserial0 00000003 .IO
#
```

5.4.14 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。

5.4.15 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

CRC32 を計算し、表示します。

5.4.16 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

指定した時刻を RTC に書き込みます。

例：

```
# 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

5.4.17 dcache - CPU データキャッシュの操作

文法： dcache [on|off]

引数を与えないと現在の状態を表示します。

on を指定すると、ライトバック動作になります。
off を指定すると、ライトスルーになります。
変更した時に、正しくキャッシュフラッシュが出来るかどうかまでは検証しておりませんので、使う人の責任でご利用ください。

関連：　icache

5.4.18 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

5.4.19 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 \ を処理してから echo コマンドが評価するのですが、echo コマンド処理部でも \ の処理をもう一度するので、\c をコマンド処理部に正しく評価させる方法はかなり独特です。
例：

```
# echo 'a\c'
ac
```

```
# echo 'a\\c'
a# echo 'a\\\c'
a# echo 'a\\\\c'
a\# echo "a\c"
a# echo "a\\c"
a\# echo a\c
ac
# echo a\\c
a# echo a\\\c
a# echo a\\\\c
a\#
```

5.4.20 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: 開始イレースブロック番号
SL: 終了イレースブロック番号
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)

指定された範囲内の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)。
本ボードは 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。

関連： protect

5.4.21 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、最後の行は exit で終了する様にしておくと安全です。
古い u-boot ではスクリプトの最後に NUL 文字(0x0)が必要でした。今はスクリプトの最後に自動的に終わりの印の NUL を付加して評価していますが、'exit\n'でスクリプトを終了させる事で、終わりの印が無くても正しく動作します。

5.4.22 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)

本ボードでは 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, erase

5.4.23 fsinfo - フラッシュ上の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。

関連：ls fsload

5.4.24 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、31100000 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：ls fsinfo loadb loads nfs tftpboot

5.4.25 go - 指定アドレスから実行を始める

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

例：

```
# nfs 30001000 192.168.3.91:/usr/src/mldbox/u-boot.git/examples/he
llo_world.bin
Using RTL8169#0 device
File transfer via NFS from server 192.168.3.91; our IP address is
192.168.3.202
Load address: 0x30001000
Loading: #
done
Bytes transferred = 538 (21a hex)
# go 30001000 a b c d
## Starting application at 0x30001000 ...
Example expects ABI version 4
Actual U-Boot ABI version 4
Hello World
argc = 5
argv[0] = "0x30001000"
argv[1] = "a"
argv[2] = "b"
argv[3] = "c"
argv[4] = "d"
argv[5] = "<NULL>"
Hit any key to exit ...

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

関連：　　bootelf

5.4.26　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。

5.4.27　icache　- CPU インストラクションキャッシュの操作

文法：　　icache [on|off]

引数を与えないと現在の状態を表示します。

on を指定すると、キャッシュが有効になります。

off を指定すると、キャッシュが無効になります。

変更した時、正しくキャッシュフラッシュが出来るかどうかまでは検証しておりませんので、使う人の責任でご利用ください。

関連：　　dcache

5.4.28　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 60000

## Checking Image at 00060000 ...
   Legacy image found
   Image Name:   Linux-2.6.29-rc8-mlb-00213-gc784
   Created:      2009-05-11  12:54:06 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1417936 Bytes =  1.4 MB
   Load Address: 30008000
   Entry Point:  30008000
   Verifying Checksum ... OK
#
```

関連： imls imxtract

5.4.29 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。

関連： imxtract iminfo

5.4.30 imxtract - マルチイメージの一部を展開

文法： imxtract addr part [dest]

addr: イメージが置いてあるアドレス

part: 展開したいパート

addr: 展開先のアドレス

マルチイメージの part 番目を dest に展開します。

関連： imls iminfo

5.4.31 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

5.4.32 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、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 nfs tftpboot

5.4.33 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

5.4.34 loady - シリアル経由でファイルのダウンロード

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。xmodem、ymodem、zmodem 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

5.4.35 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。

5.4.36 loopw - 指定した範囲のアドレスを書き続ける無限ループ

文法： loopw[.b, .w, .l] addr count data

[.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

data: 書き込むデータ

開始アドレスから、count 個までの範囲に data を書き続けます。

動作確認はしておりません。

5.4.37 ls - フラッシュ上の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。

関連：chpart fsinfo fsload mtdparts

5.4.38 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。

関連： base mm nm mw cmp cp crc32

5.4.39 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。

関連：　　base md nm mw cmp cp crc32

5.4.40　mp201_clocks　　- 現在のクロック供給の状態を表示

文法：　　mp201_clocks

clk1:クロックが供給されているライン
クロックが供給されていないライン
というフォーマットで、出力されます。
RTC、OSC、HPLL、LPLL、CPU のクロックも表示します

5.4.41　mp201_gpiopin　　- 現在の GPIO の状態を表示

文法：　　mp201_gpiopin

gpio0～63,gpo10～gpo8 の状態を表示します。
pin が別機能に割り当てられている場合は、そのピンの名称を表示し、
gpio に割り当てられている場合には、方向、状態、割込みの種別、状態、トリガの状態、種別が表示されます。

5.4.42　mp201_resets　　- 現在のリセットラインの状態を表示

文法：　　mp201_resets
　　各領域（L0,1A,1B,DSP)のリセットラインの状態を表示します。
　領域名：リセット解除されているライン
　＃リセットされているライン
の順に表示されます。

5.4.43　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

part_num:= パーティション番号

本ボードでは、<dev-num> デバイス番号は 0 のみが有効です。
本ボードでは、<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。

mtdids：u-boot のデバイス ID と Linux の MTD デバイス名と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

本ボードでは、mtdids は nor0=tphysmap-flash.0 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。

これら 3 つの環境変数の設定、表示を行うのが、mtdparts コマンドです。直接環境変数を設定しても構いません。

文法：mtdparts

現在の設定を表示する

文法：mtdparts delall

設定を全て削除する

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
パーティション ID には、パーティション名称も利用できます。

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

5.4.44 mtest - 簡単なメモリのテスト

文法： mtest [start [end [pattern [iterations]]]]
start: 開始アドレス
省略時 30000000
end: 終了アドレス
省略時 37F40000 になります。
pattern: 書き込む値の初期値
iterations: 繰り返す回数
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もう一度行います。
パターンの値を 1 増やしながら、上記を iterations 回繰り返します。
^C (Ctrl キーを押しながら C)で終了します。

5.4.45 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

5.4.46 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、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、/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 tftpboot

5.4.47 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。

5.4.48 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。

5.4.49 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。

5.4.50 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 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の 4 ブロックはサイズが
32k,16k,16k,64k バイトで、残りの部分は 128k バイトです。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

5.4.51　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、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

5.4.52　reset　- CPU のリセット

文法：　　reset

再起動します。

5.4.53　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。

5.4.54 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。

5.4.55 saves メモリの内容を S レコード形式で吸いだし

文法： saves addr size
addr: 開始アドレス
size: バイト数
メモリの内容を S レコード形式でシリアルに出力します。

5.4.56 showvar - 環境変数の設定と削除

文法： showvar [name]
name: シェル変数の名称

シェル変数と、設定されている内容の一覧を表示します。
name が指定された場合、その変数名の行のみ表示します。

5.4.57 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。

5.4.58 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)

5.4.59 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。

関連: date

5.4.60 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。

5.4.61 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

5.4.62　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。

5.5　u-boot で特殊な意味を持つ環境変数の一覧

5.5.1　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。

5.5.2　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。

5.5.3　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ければ、呼び出されません。

5.5.4　autostart

値：　yes|no

初期値：　なし

diskboot, 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ければ、呼び出されません。
bootm で standalone アプリを起動する時は、no と設定されていると展開まで行い終了します。

5.5.5 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。

5.5.6 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。

5.5.7 bootargs

値： 任意の文字列
初期値： ソースの CONFIG_BOOTARGS の定義を参照して下さい。
OS に渡すデフォルトの起動パラメータを設定します。

5.5.8 bootcmd

値： 任意の文字列
初期値： CONFIG_BOOTCOMMAND の定義を参照して下さい。
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、自動起動する際に参照されます)

5.5.9 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。

5.5.10 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。

5.5.11 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。

5.5.12 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。

5.5.13 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。

5.5.14 ethact

値： 文字列
初期値： 最初に見付けたイーサネットカードの名称
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。
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変えられ、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。

5.5.15 ethaddr

値： MAC アドレス
初期値： なし
1 番目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。

5.5.16 eth1addr

値： MAC アドレス

初期値： なし

2 番目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。

5.5.17 ethprime

値： 文字列

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。

5.5.18 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。

5.5.19 filesize

値： 16 進の値

初期値： なし

ext2load,fatload,fsload,loadb,loads,nfs,reiserload,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。

5.5.20 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。

5.5.21 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。

5.5.22 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。

5.5.23 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。

5.5.24 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。

5.5.25 mtddevname

値： 文字列
初期値： なし
フラッシュのカレントパーティションの名称が設定されます

5.5.26 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

5.5.27 mtdids

値： 文字列
初期値： MTDIDS_DEFAULT を参照
Linux の MTD デバイス名と、u-boot のデバイス名の対応付けを設定します。

5.5.28 mtdparts

値： 文字列
初期値： MTDIDS_DEFAULT を参照
パーティションの設定を記述します。

5.5.29 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。

5.5.30 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。

5.5.31 ntpserverip

値： IP アドレス
初期値： なし
bootp,dhcp で ntp-servers がもらえた場合に上書き設定されます。
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。

5.5.32 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。

5.5.33 partition

値： パーティション ID
初期値： なし
フラッシュのカレントパーティションが設定されます。

5.5.34 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。

5.5.35 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。

5.5.36 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。

5.5.37 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。

5.5.38 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。

5.5.39 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。

5.5.40 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。
bootp,dhcp サーバから time-offset が取得できた場合に上書きする場合があります。

sntp コマンドが使用します。

5.5.41　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。

5.5.42　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。

5.6　u-boot についてもっと知る

u-boot の開発は以下で行なわれています。

http://www.denx.de/wiki/U-Boot

また、u-boot に関するより詳細なドキュメントは、英語ですが、

http://www.denx.de/wiki/view/DULG/UBoot

からたどれますので、参照して下さい。

日本語のサイトでは、

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

が参考になります。

移植はメディアラボ(http://www.mlb.co.jp)が行ないましたので、
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、アップデート等は、

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

からたどれます。