U-BOOT
ユーザーズマニュアル

対応ボード

TB0287MINI-ITX+MPC5200DIMM

TB0287MINI-ITX+VR4131DIMM

LIMM-MPC5200B

LIMM-VR4131

BLANCA+MPC5200DIMM

BLANCA+VR4131DIMM

2007.11.15

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、弊社で選択したコンパイルオプション（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。

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。

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。

1.3 ハードウェア

1.3.1 MPC5200DIMM(TB0286)

MPC5200DIMM(写真 1)は Freescale Semiconductor 社の MPC5200B(PowerPC 603e コア、400MHz)と、SDRAM 64MBtye,フラッシュ 8M Byte(標準構成)を SO-DIMM 形状に実装したものです。この基板固有の事を記述する場合には、MPC5200DIMM と呼びます。

写真 1:MPC5200DIMM(TB0286)

写真 1 には、ヒートシンクがついておりませんが、実際には写真 4 に実装されている形状になります。

u-boot がサポートするデバイスは、DRAM、フラッシュ及び CPU 内蔵の PCI バスコントローラ、I2C バスコントローラ、RTC、シリアル(2ch のうち 1ch のみ)が利用出来ます。

後述の VR4131DIMM と区別する必要がない場合、両者を併せて DIMM-CPU と記述します。

1.3.2 VR4131DIMM (TB0229)

VR4131DIMM(写真 2)は、NEC エレクトロニクス社の V_R4131(MIPS III、200MHz)と、SDRAM 64M Byte,フラッシュ 4 Mbyte を SO-DIMM 形状に実装したものです。

この基板固有の事を記述する場合には、VR4131DIMM と呼びます。

写真 2:VR4131DIMM(TB0229)

u-boot がサポートするデバイスは、DRAM、フラッシュ及び、CPU 内蔵の PCI バスコントローラ、I2C バスコントローラ、RTC、シリアル(2ch のうち 1ch のみ)が利用出来ます。

前述の MPC5200DIMM と区別する必要がない場合、両者を併せて DIMM-CPU と記述します。

1.3.3 TB0287MINI-ITX(TB0287)

TB0287MINI-ITX 基板(写真 3)は、DIMM-CPU のベースボードです。

MPC5200DIMM、VR4131DIMM のどちらでも搭載できます。

TB0287MINI-ITX 上にあるデバイスで、u-boot から利用可能なデバイスは、

1. Gigabit Ether x2 (PCI 接続)
2. IDE 2ch (PCI 接続、プライマリは CF)
3. RTC(I2C 接続)

となります。

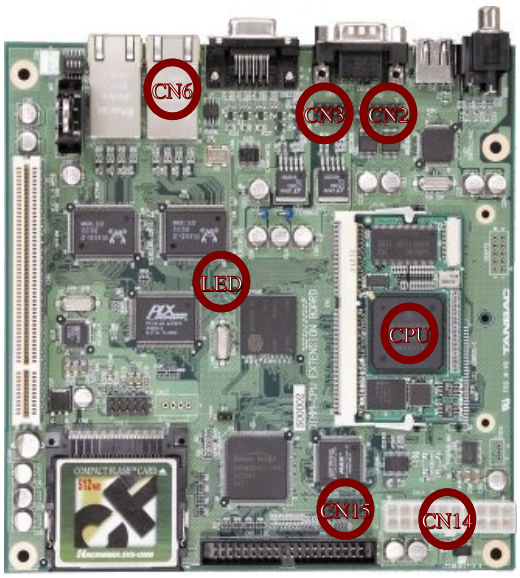

写真 3:TB0287MINI-ITX + MPC5200DIMM

- DIMM-CPU の接続

  DIMM-CPU を CPU スロットに差し込みます。

- シリアルの接続

  DIMM-CPU の CPU 内蔵のシリアルは、写真右上のシリアルコネクタの手前にある 10pin ヘッダ部(CN2 と CN3)につながっております。シリアルコンソールとして利用可能なポートはこの 2 つのポートで、MPC5200DIMM を載せて使う場合 CN2(写真の右側)になり、VR4131DIMM では CN3(写真の左側)を利用します。

  各コネクタは写真左下が 1 番ピンとなり、以下の pin 配置です。

DSR	RTS	CTS	NC	NC
2	4	6	8	10
1	3	5	7	9
NC	RXD	TXD	DTR	GND

  クロスケーブルを使って PC と接続してください。

- 電源の接続

  CN14 (写真右下)に ATX 電源を繋ぎます。

- 電源スイッチ / リセットスイッチの接続

  CN15(写真下、電源コネクタ CN14 の左横の 10pin ヘッダ部)に電源スイッチ等を接続してください。写真左下が 1 番ピンとなり、以下の pin 配置です。

Power LED(-)	Power LED(+)	Power Switch	GND	NC
2	4	6	8	10
1	3	5	7	9
IDE LED(+)	IDE LED(-)	GND	Reset Switch	NC

  5 番と 6 番をショートすると電源が入ります。5 番と 6 番を 5 秒以上ショートすると、電源が強制的に落ちます。(電源制御は PIC マイコンでおこなっていますので、CPU が暴走した状態でも制御できます)

- イーサネットは 2ch 共使えますが、写真右側(CN6)の方が先に認識される方になります。

電源投入時の確認事項

写真中央にある 4 つの LED で状態が確認できます。写真で LED とかかれた部分の上から順に D15,D16,D1,D14 があります。

D15	本体の電源が入り、電源から 5V が供給されると点灯します
D16	本体の電源が入り、電源から 3.3V が供給されると点灯します
D1	ATX 電源からスタンバイ電源が供給されると点灯します
D14	スタンバイ電源を使った電源制御用 PIC マイコンが電源投入待ちの間 1 秒周期で点滅します。本体の電源が入ると消灯します。

注意：

市販の ATX 電源の中には、負荷が小さい場合に出力電圧が安定しないものがあります。その様な場合には、ハードディスクを電源に繋げる等負荷を増やす事で安定させられる場合があります。

電源の管理の為に PIC マイコンをスタンバイ電源を使って動かしています。古い ATX 電源ではスタンバイ電源容量が(0.5W)程度のものが普通であり、PCI スロットに電源の供給能力以上のスタンバイ電源を利用するカード（例えば、WOL 対応ネットワークワークカード等）を差した場合に、本体の電源が入らなくなる場合があります。この時 D1 は点灯するが D14 が点滅を始めない状態になります。

1.3.4 TB0311(LIMM-MPC5200B、LIMM-VR4131)

TB0311 (写真 4)は、DIMM-CPU のベースボードの一つです。

MPC5200DIMM、VR4131DIMM のどちらでも搭載できます。

TB0311 は、DIMM-CPU の評価ボードとしての利用と、DIMM-CPU のコネクタを CQ 出版社の BLANCA ボードのオプション CPU スロットに繋げる際の変換基板としても利用されています。（利用目的に応じて、実装部品が異なりますので共用は出来ません。）

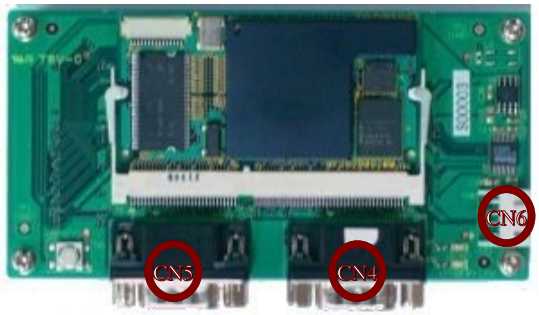

写真 4:TB0311+MPC5200DIMM

DIMM-CPU の評価ボードとして DIMM-CPU とセットで販売されているものは、それぞれ LIMM-MPC5200B、LIMM-VR4131 と呼びますが、これらのにはフラッシュには、BLANCA のオプションとして販売されているものと同一のバイナリを載せて出荷されています。

- DIMM-CPU の接続

DIMM-CPU を CPU スロットに差し込みます。

- シリアルの接続
  DIMM-CPU の CPU 内蔵のシリアルが写真下部のシリアルコネクタに引き出されています。シリアルコンソールとして利用可能なポートは、この 2 つのポートで、MPC5200DIMM の場合写真下の右側(CN4)、R4131DIMM では写真下の左側(CN5)になります。
  (写真では、右側のシリアルコネクタに白いシールが張られておりますが、実際にはありません。)
  クロスケーブルを使って PC と接続してください。
- 電源の接続
  写真右下の CN6 に 3.3V(±5%)を供給します。最大でも 0.2A 未満です。
  写真下側が 1 番ピンで、以下の様になります。

1	GND
2	GND
3	+3.3V 電源
3	+3.3V 電源

DIMM-CPU のフラッシュには、次章の BLANCA 用と同一のバイナリを載せて出荷されています。

1.3.5 BLANCA

BLANCA とは、CQ 出版社より「組込みシステム開発評価キット」として販売されている FPGA 搭載マザーボードです。BLANCA のオプション CPU カードコネクタに、TB0311 を接続する事で、MPC5200DIMM、VR4131DIMM のどちらも搭載できる様になります。DIMM-CPU から BLANCA 上のデバイスを利用する場合には、PCI 接続となります。BLANCA 上の FPGA には、PCI デバイスとして見えるコアが入っていなくてはなりません。

BLANCA 上にあるデバイスで、u-boot から利用可能なデバイスは、

1. 10/100 Ether
2. IDE

です。

シリアルコンソールは TB0311 に載っているコネクタを利用します。

電源は BLANCA のオプション CPU カードコネクタを経由して供給されますので、TB0311 の電源コネクタ(CN6)は使わないでください。

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、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 1.2.0-mlb-g89471a9c (Oct  2 2007 - 16:12:07)

Board: TANBAC TB0229 on TB0287 (CPU Speed 200 MHz)
DRAM:  64 MB
Flash:  4 MB
IDE0 @0xb60013c0
In:    serial
Out:   serial
Err:   serial
Net:   RTL8169#0, RTL8169#1

Type "boot"             for default way
Type "run cram"         to boot from flash
Type "run disk"         to boot from CF
Type "run nfs"          to boot from NFS
Type "run initrd"       to boot with initrd

Hit any key to stop autoboot:  0
#
```

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 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 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 line /dev/ttyS0
  set flow auto
  set speed 115200
  set serial 8n1
  set carrier-watch off
  set escape 27
  set prefixing all
  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 pci
# printenv load_kernel_
  load_kernel_cram load_kernel_ext2 load_kernel_fat load_kernel_tf
tp
  load_kernel_nfs load_kernel_serial
# printenv load_kernel_fat
load_kernel_fat=fatload ide $fatdev 80400000 $bootfile
```

#

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定義して、組み合わせて利用する事ができます。

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。
環境変数を保存する際は起動直後に設定し、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。
  - IDE 接続のデバイスの FAT ファイルシステムから読み込む
  - IDE 接続のデバイスの EX2 ファイルシステムから読み込む
  - 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_ext2　load_kernel_fat

  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_$load_method ... と設定されていますので、自分がよく使う方法(cram,ext2,fat,tftp,nfs のどれか)を環境変数 load_method に設定しておく事で、run load_kernel (または load_uboot, load_cram)とすると、$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。(VR4131DIMM では、uboot と、cram の時は、環境変数 update_method を利用)。

  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	IDE の ext2	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 ”disk nfs cram”
setenv bootcmd 'for i in $bootorder; do run $i; done'
```

  としておくと、disk、nfs、cram 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

ext2dev　　IDE ディスク番号とパーティションを設定

fatdev　　IDE ディスク番号とパーティションを設定

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。

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。

PowerPC では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が、MIPS では、例外ハンドラは無限ループになっており、見た目固まった様に見えます。

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。

3.1 MPC5200DIMM

MPC5200DIMM には、64MB の DRAM、8MB のフラッシュメモリを実装しています。

開始	終了	用途
00000000	03FFFFFF	DDR-SDRAM
40000000	4fffffff	PCI メモリ空間(PCI バス上では 40000000-4fffffff に変換)
50000000	50ffffff	PCI　I/O 空間（PCI バス上では 50000000-50ffffff に変換)
F0000000	F000FFFF	内部 I/O 領域
FF800000	FFFFFFFF	フラッシュメモリ

PCI バス上でのアドレス範囲 00000000 ～ 3FFFFFFF はホスト側が PCI ターゲットとして動作する際の空間に割り当てられていますので、DMA 等を行う際には実アドレスをそのまま設定します。

3.1.1 DRAM エリア　　0x00000000　　から 0x03FFFFF

開始	終了	用途
00000000	00002FFF	例外ベクタ
00003000		空き
	03F42FFF	u-boot スタック
03F43000	03F82FFF	u-boot malloc 領域(256k)
03F83000	03FFFFFF	u-boot 本体

u-boot が使っているアドレスの開始、終了は、だいたいその辺という意味で、きっちりそのアドレスから始まるわけではなく、メモリの最後(04000000)から必要な領域を順に確保していきます。正確に知りたい場合には、lib_ppc/board.c の先頭に、

#define DEBUG 1

を挿入し、作成しなおすと表示されます。

3.1.2 フラッシュエリア　　0xFF800000　　から 0xFFFFFFFF

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	kernel	カーネル保存領域	FF800000	2M bytes
1	user	ルートファイルシステム	FFA00000	5M bytes

番号	名称	用途	開始アドレス	サイズ
2	uboot	u-boot 本体	FFF00000	256K bytes
3	ubootenv	u-boot 環境変数保存領域	FFF40000	128K bytes
4	conf	設定保存領域	FFF60000	640K 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 64 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。

3.2 VR4131DIMM

VR4131DIMM には、64MB の DRAM、4MB のフラッシュメモリを実装しています。

3.2.1 物理空間

開始	終了	用途
00000000	03FFFFFF	SDRAM(バンク 0)
0A000000	0A03FFFF	システムバス(IOCS0)
0C000000	0C03FFFF	システムバス(IOCS1)
0F000000	0FFFFFFF	内部 I/O 領域
10000000	15FFFFFF	PCI メモリ空間(PCI バス上では 10000000-15FFFFFF に変換)
16000000	17FFFFFF	PCI　I/O 空間（PCI バス上では 00000000-01FFFFFF に変換)
1FC00000	1FFFFFFF	フラッシュメモリ(バンク 1)

PCI I/O 空間は注意が必要です。CPU からみた物理アドレス 16000000 にアクセスすると、PCI バス上の I/O アドレス 0 番地に変換されます。実際には次章で説明する kseg1 の空間を使ってアクセスしますので、B6000000 が PCI バス上の I/O アクセスの際の 0 番地になります。

PCI バス上のメモリ空間でのアドレス 0 番地から 03FFFFFF までは SDRAM がマップされます。DMA を利用する場合（ホストが PCI ターゲットとして動作する場合）には、転送先アドレスに物理アドレスを指定すれば動作します。

3.2.2 論理空間

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

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

u-boot では MMU は使いません。kseg0 と kseg1 はどちらも上位 3 ビットを取り除いた物理空間へ変換されますので、u-boot では DRAM 等は kseg0 を使い、デバイスのレジスタの操作や DMA を行う時には kseg1 の空間を使います。

3.2.3 DRAM エリア　　0x80000000　　から 0x83FFFFFF

開始	終了	用途
80000000		空き
	83F27FFF	u-boot スタック
83F28000	83F3FFFF	u-boot データ(カーネルパラメータ領域 128K 含む)
83F48000	83F87FFF	u-boot malloc 領域(256k)
83F88000	83FFFFFF	u-boot 本体

u-boot が使っているアドレスの開始、終了は、だいたいその辺という意味で、きっちりそのアドレスから始まるわけではなく、メモリの最後(84000000)から必要な領域を順に確保していきます。

現在は利用しておりませんが、将来的には DRAM の先頭に例外ベクタが並ぶ可能性があります。u-boot から DRAM を利用する際には、先頭数 K バイトと、最後は使わない様にしておきます。

正確に知りたい場合には、lib_mips/board.c の先頭に、

#define DEBUG 1

を挿入し、作成しなおすと表示されます。

3.2.4 フラッシュエリア　　0xBFC00000　　から 0xBFFFFFFF

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	BFC00000	256K bytes
1	ubootenv	u-boot 環境変数保存領域	BFC40000	128K bytes
2	root	ルートファイルシステム	BFC60000	3M bytes
3	conf	設定保存領域	BFF60000	640K 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は 35 個のイレースブロックがあり、先頭の 4 ブロックは

イレースブロックサイズが順に 32k, 16k, 16k, 64k バイトとなっており、残りの部分は全て 128k バイトです。

- u-boot 本体の位置は、CPU リセット時に開始するアドレスになければいけないので、この位置からは動かせません。
- 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。
  フラッシュ容量が小さいので、デフォルトではカーネルを入れる場所を作らず、ルートファイルシステム内にカーネルをいれております。

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

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。

4.2 u-boot 用の Linux カーネルの作り方

4.2.1 PowerPC の場合

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。

4.2.2 MIPS の場合

カーネルを普通に作成したあと、以下の様にします。

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

mips 用カーネルのエントリーポイントは、コンパイル毎にかわりますので、
System.map から取得する必要があります。

4.2.3 スクリプトを作る

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

```
$ echo "echo hello" > script
$ u-boot/tools/mkimage -A mips -O u-boot -T script -C none -n
```

```
"hello" -d script script.img
Image Name:   hello
Created:      Sat Dec  4 16:56:14 2004
Image Type:   MIPS U-Boot Script (uncompressed)
Data Size:    19 Bytes = 0.02 kB = 0.00 MB
Load Address: 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 mips 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。

上記は MIPS での例です。PowerPC では、00400000 にダウンロードして実行します。

実行するには、autoscr を使います。

```
# nfs 80400000 192.168.3.91:/home/mld/script.img
Using ifnic#0 device
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: 0x80400000
Loading: #
done
Bytes transferred = 83 (53 hex)
# autoscr
## Executing script at 80400000
hello
#
```

上記は MIPS での例です。PowerPC では、00400000 にダウンロードして実行します。

4.2.4 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 80400000 \ 192.168.3.91:/home/mld/u-boot/examples/hello_worl
d
Using RTL8169#0 device
File transfer via NFS from server 192.168.3.91; our IP address is
192.168.3.227
Filename '/home/mld/u-boot/examples/hello_world'.
Load address: 0x80400000
```

```
Loading: ##
done
Bytes transferred = 6544 (1990 hex)
# bootelf
Loading .text @ 0x80200000 (720 bytes)
Loading .reginfo @ 0x802002d0 (24 bytes)
Loading .rodata.str1.4 @ 0x802002e8 (152 bytes)
Loading .got @ 0x80200380 (56 bytes)
## Starting application at 0x80200000 ...
Example expects ABI version 3
Actual U-Boot ABI version 3
Hello World
argc = 0
argv[0] = "<NULL>"
Hit any key to exit ...

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

上記はMIPSでの例です。PowerPCでは、00400000にダウンロードして実行します。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、メディアラボでの拡張です。

u-boot用のイメージに変換するには、最初にバイナリに変換します。

MIPSでは、

```
$ mipsel-linux-objcopy -O binary -R .note -R .comment -S hello_world hello_world.bin
$ gzip hello_world.bin
$ mkimage -A mips -O u-boot -T standalone -n "hello world" -C gzip -a 80200000 -d hello_world.bin.gz hello_world.img
Image Name:   hello world
Created:      Thu Oct 25 18:01:22 2007
Image Type:   MIPS U-Boot Standalone Program (gzip compressed)
Data Size:    528 Bytes = 0.52 kB = 0.00 MB
Load Address: 0x80200000
Entry Point:  0x80200000
$
```

PowerPCでは、

```
$ powerpc-linux-objcopy -O binary -R .note -R .comment -S hello_world hello_world.bin
$ gzip -9 hello_world.bin
```

```
$ mkimage -A ppc -O u-boot -T standalone -n "hello world" -C gzip
-a 00040000 -e 00040004 -d hello_world.bin.gz hello_world.img
Image Name:   hello world
Created:      Thu Oct 25 18:19:42 2007
Image Type:   PowerPC U-Boot Standalone Program (gzip compressed)
Data Size:    651 Bytes = 0.64 kB = 0.00 MB
Load Address: 0x00040000
Entry Point:  0x00040004
$
```

とします。

mkimage で作ったバイナリは、bootm コマンドで実行できます。

```
# nfs 80400000 192.168.3.91:/usr/src/mldbox/hello.img
Using RTL8169#0 device
File transfer via NFS from server 192.168.3.91; our IP address is
192.168.3.202
Filename '/usr/src/mldbox/hello.img'.
Load address: 0x80400000
Loading: #
done
Bytes transferred = 592 (250 hex)
# bootm
## Booting image at 80400000 ...
   Image Name:   hello world
   Created:      2007-10-25   9:01:22 UTC
   Image Type:   MIPS U-Boot Standalone Program (gzip compressed)
   Data Size:    528 Bytes =  0.5 kB
   Load Address: 80200000
   Entry Point:  80200000
   Uncompressing Standalone Application ... OK
Example expects ABI version 3
Actual U-Boot ABI version 3
Hello World
argc = 0
argv[0] = "<NULL>"
Hit any key to exit ...

#
```

引数も渡せますが、第一引数はプログラムの置いてある場所として、第二引数は、展開する場所として解釈されますので、3 っ目から自由に使えます。

```
# bootm 80400000 80200000 a b c d
## Booting image at 80400000 ...
   Image Name:   hello world
   Created:      2007-10-25   9:01:22 UTC
```

```
   Image Type:   MIPS U-Boot Standalone Program (gzip compressed)
   Data Size:    528 Bytes =  0.5 kB
   Load Address: 80200000
   Entry Point:  80200000
   Uncompressing Standalone Application ... OK
Example expects ABI version 3
Actual U-Boot ABI version 3
Hello World
argc = 6
argv[0] = "80400000"
argv[1] = "80200000"
argv[2] = "a"
argv[3] = "b"
argv[4] = "c"
argv[5] = "d"
argv[6] = "<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 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 3 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 と同じ

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

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/u-boot.h と include/asm/global_data.h にありますので、ソースを参照してください。

MIPS であれば、include/asm-mips/u-boot.h、include/asm-mips/global_data.h

PowerPC であれば、include/asm-ppc/u-boot.h、include/asm-ppc/global_data.h

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。

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 80000000 4
80000000: 00000000 00000000 00000000 00000000    ................
#
80000010: 00000000 00000000 00000000 00000000    ................
#
```

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 した後等に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　^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。

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 <ボード名>_config
$ make
```

とします。

u-boot の機能の設定は、u-boot ソースの include/configs/<ボード名>.h で行います。
ボード名の部分はそれぞれ、

TB0287MINI-ITX+MPC5200DIMM	tb0287_tb0286
TB0287MINI-ITX+VR4131DIMM	tb0287_tb0229
LIMM-MPC5200B	blanca_tb0286
LIMM-VR4131	blanca_tb0229
BLANCA+MPC5200DIMM	blanca_tb0286
BLANCA+VR4131DIMM	blanca_tb0229

となります。

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、1.2 に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、MIPS では 80400000 で、PowerPC では 00400000 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。(デフォルトでは verify=n に設定しています。)

例：

```
# iminfo 80400000

## Checking Image at 80400000 ...
   Image Name:   hello
   Created:      2004-12-04   9:24:20 UTC
   Image Type:   MIPS U-Boot Script (uncompressed)
   Data Size:    24 Bytes =  0 kB
   Load Address: 80200000
   Entry Point:  80200000
   Verifying Checksum ... OK
# autoscr
## Executing script at 80400000
hello
```

#

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。

例：

```
# nfs \"192.168.3.91:/usr/src/mldbox/u-boot/examples/hello_world\"
Using RTL8169#0 device
File transfer via NFS from server 192.168.3.91; our IP address is
192.168.3.202
Filename '/usr/src/mldbox/u-boot/examples/hello_world'.
Load address: 0x80400000
Loading: ##
done
Bytes transferred = 6798 (1a8e hex)
# bootelf 80400000
Loading .text @ 0x80200000 (672 bytes)
Loading .rodata @ 0x802002a0 (160 bytes)
Loading .reginfo @ 0x80200340 (24 bytes)
Loading .got @ 0x80200360 (56 bytes)
## Starting application at 0x80200000 ...
Example expects ABI version 2
Actual U-Boot ABI version 2
Hello World
argc = 1
argv[0] = "80400000"
argv[1] = "<NULL>"
Hit any key to exit ...

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

関連： bootm go

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 ...]]]

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。

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。

関連： bootelf bootvx iminfo

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、loadaddr も設定されていなければ、MIPS では 80400000 で、PowerPC では 00400000 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,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。

例：

```
# nfs 80400000 192.168.3.91:/usr/src/mldbox/u-boot.bin
Using RTL8169#0 device
File transfer via NFS from server 192.168.3.91; our IP address is
192.168.3.202
Filename '/usr/src/mldbox/u-boot.bin'.
Load address: 0x80400000
Loading: #########################################
done
Bytes transferred = 199948 (30d0c hex)
# cmp.b BFC00000 80400000 $filesize
Total of 199948 bytes were the same
#
```

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。

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：

```
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)

> TB0287MINI-ITX では、バッテリーバックアップされた、ベースボード上の RTC チップに対して読み書きを行います。
>
> TB0311 や BLANCA では、CPU 内蔵の RTC に対して読み書きを行いますので、電源を切ると日時はリセットされます。

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。(デフォルト)
off であることが前提の部分がありますので、on にしたらハングするでしょう。

MIPS でのキャッシュの扱いは、アドレスの上位 3 ビットで区別しますので、このコマンドはありません。

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 diskboot - IDE デバイスからのブート

文法： diskboot [addr] [dev[:part]]

addr: ロードするアドレスです。
省略した場合、MIPS では 80400000 で、
PowerPC では 00400000 です。

dev[:part]: デバイス番号(とパーティション番号)を指定します。省略すると環境変数 bootdevice の文字列が使われます。
パーティション番号を省略した場合は 1 になります。

指定した IDE デバイスからブートイメージをロードします。その後、環境変数 autostart が yes に設定されていると、引数なしで bootm を呼び出します。

5.4.20 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\#
```

5.4.21 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)。
どのボードも 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。
VR4131DIMM 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。
例

```
# erase BFC40000 BFFDFFFF

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

関連： protect

5.4.22 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。

5.4.23 ext2load - ext2 からのファイルのロード

文法： ext2load interface dev[:part] [addr] [filename] [bytes]

interface: インタフェースを指定します。
ide、scsi、usb、mmc、ace が指定可能ですが、このマニュアルが対象としているボードでは、ide のみが有効なインタフェースです。

dev[:part]: デバイス番号(とパーティション番号)を指定します。
パーティション番号を省略した場合は 1 になります。
パーティションがないデバイスではパーティション番号として 0 を指定します。

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 が設定されていなければ MIPS では 80400000 で PowerPC では 00400000 です。

filename: ロードするファイルです。
省略すると環境変数 bootfile の値が使われます。

bytes: ロードするバイト数です。
省略した場合または 0 を指定した場合はファイルサイズになり

ます。

指定したファイルのデータをメモリにロードします。
ファイルのロードが正常に終了すると、環境変数 filesize にロードしたバイト数が設定されます。

関連：　ext2ls

5.4.24　ext2ls　- ext2 のファイルリスト

文法：　ext2ls interface dev[:part] [directory]

interface:　インタフェースを指定します。
ide、scsi、usb、mmc、ace が指定可能ですが、このマニュアルが対象としているボードでは、ide のみが有効なインタフェースです。

dev[:part]:　デバイス番号(とパーティション番号)を指定します。
パーティション番号を省略した場合は 1 になります。
パーティションがないデバイスではパーティション番号として 0 を指定します。

directory:　リストするディレクトリを指定します。
省略すると / です。

指定した ext2 ファイルシステムのファイルリストを表示します。

関連：　ext2load

5.4.25　fatinfo　- FAT ファイルシステムの情報表示

文法：　fatinfo interface dev[:part]

interface:　インタフェースを指定します。
ide、scsi、usb、mmc、ace が指定可能ですが、このマニュアルが対象としているボードでは、ide のみが有効なインタフェースです。

dev[:part]:　デバイス番号(とパーティション番号)を指定します。
パーティション番号を省略した場合は 1 になります。

FAT ファイルシステムの情報(デバイス情報、ボリュームラベル等)を表示し

ます。

関連：　fatload fatls

5.4.26 fatload - FAT からのファイルのロード

文法：　fatload interface dev[:part] addr filename [bytes]

interface:　インタフェースを指定します。
ide、scsi、usb、mmc、ace が指定可能ですが、このマニュアルが対象としているボードでは、ide のみが有効なインタフェースです。。

dev[:part]:　デバイス番号(とパーティション番号)を指定します。
パーティション番号を省略した場合は 1 になります。

addr:　ロードするアドレスです。

filename:　ロードするファイル名です。

bytes:　ロードするバイト数です。
省略した場合または 0 の場合はファイルサイズになります。

指定したファイルのデータをメモリにロードします。
ファイルのロードが正常に終了すると、環境変数 filesize にロードしたバイト数が設定されます。

関連：　fatinfo fatls

5.4.27 fatls - FAT のファイルリスト

文法：　fatls interface dev[:part] [directory]

interface:　インタフェースを指定します。
ide、scsi、usb、mmc、ace が指定可能ですが、このマニュアルが対象としているボードでは、ide のみが有効なインタフェースです。。

dev[:part]:　デバイス番号(とパーティション番号)を指定します。
パーティション番号を省略した場合は 1 になります。

directory:　リストするディレクトリを指定します。省略すると / です。

指定した FAT ファイルシステムのファイルリストを表示します。

関連：　　fatinfo fatload

5.4.28　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)

このマニュアルが対象としているボードでは どれも 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

5.4.29　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

5.4.30　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：ls fsinfo loadb loads nfs tftpboot

5.4.31 go - 指定アドレスから実行を始め

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 80200000 $nfsbase/u-boot/examples/hello_world.bin
Using RTL8169#0 device
File transfer via NFS from server 192.168.3.91; our IP address is
192.168.3.202
Filename '/usr/src/mldbox//u-boot/examples/hello_world.bin'.
Load address: 0x80200000
Loading: #
done
Bytes transferred = 920 (398 hex)
# go 80200000 myip $serverip
## Starting application at 0x80200000 ...
Example expects ABI version 2
Actual U-Boot ABI version 2
Hello World
argc = 3
argv[0] = "80200000"
argv[1] = "myip"
argv[2] = "192.168.3.29"
argv[3] = "<NULL>"
Hit any key to exit ...

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

関連： bootelf

5.4.32 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。

5.4.33 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。(デフォルト)
off を指定すると、キャッシュが無効になります。
MIPS でのキャッシュの扱いは、アドレスの上位 3 ビットで区別しますので、このコマンドはありません。

関連： dcache

5.4.34 ide - IDE サブシステム

文法： ide info

認識された IDE デバイスの一覧を表示します。

文法： ide device [dev]

dev: デバイス番号を指定します。
指定したデバイスが current デバイスになります。
省略した場合 current デバイスを指定したことになります。

IDE デバイスの情報を表示します。

文法： ide part [dev]

dev: デバイス番号を指定します。
省略するとパーティションを持つ全デバイスが対象になります。

IDE デバイスのパーティション情報を表示します。

文法： ide read addr blk# cnt

addr: 読み込み先の先頭アドレス

blk#: 読み込み開始のブロック番号

cnt: 読みこむブロック数

current デバイスのブロック blk#から cnt ブロックをメモリの addr に読み込みます。

文法： ide write addr blk# cnt

addr: 書き込むメモリの先頭アドレス

blk#: 書き込み開始のブロック番号

cnt: 書き込むブロック数

current デバイスのブロック blk#から cnt ブロックにメモリの addr から書き込みます。

文法： ide reset

IDE デバイスの認識をやり直します。最初に認識されたデバイスが current デバイスになります。

オリジナルの u-boot では、起動時に IDE の reset が自動的に行われますが、何も接続されていない場合に時間がかかるので、明示的に ide reset コマンドを行わないと、IDE の認識はしない様に変更してあります。

5.4.35 iminfo - アプリケーションイメージヘッダの表示

文法： iminfo [addr...]

addr: アドレス

省略した場合、最後にファイルを読み込んだアドレス。

ヘッダ情報表示と、チェックサム確認を行います。

例：

```
# iminfo

## Checking Image at 80400000 ...
   Image Name:   Linux Kernel Image
   Created:      2004-12-02   7:52:37 UTC
   Image Type:   MIPS Linux Kernel Image (gzip compressed)
```

```
    Data Size:    764307 Bytes = 746.4 kB
    Load Address: 80000000
    Entry Point:  8019a040
    Verifying Checksum ... OK
  #
```

5.4.36 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。

5.4.37 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

5.4.38 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

5.4.39 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

5.4.40 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

5.4.41 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。

5.4.42 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

5.4.43 md - メモリ内容の表示

文法： md[.b, .w, .l] addr [count]

[.b, .w, .l]: バス幅

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

addr: 表示するアドレス

count: 指定した単位での数を 16 進で指定します。

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

例：

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

関連： base mm nm mw cmp cp crc32

5.4.44 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：

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

```
# md.w 80400020 4
80400020: 494c 756e 2078 654b    LInux Ke
#
```

関連：　　base md nm mw cmp cp crc32

5.4.45　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

DIMM-CPU では、<dev-num> デバイス番号は 0 のみが有効です。

DIMM-CPU 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

DIMM-CPU 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

5.4.46 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 VR4131DIMM では 80000000、MPC5200DIMM では 00100000 になります。

end: 終了アドレス

省略時　VR4131DIMM では 8080000、MPC5200DIMM では 00f00000 になります。

pattern:任意の値

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

5.4.47　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

5.4.48　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

5.4.49 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。

5.4.50 pci - PCI バス一覧と、PCI コンフィグレーションアクセス

文法： pci [bus] [long]

bus: バス番号を指定します。省略時は 0

long: 詳細を出したいときには、long を指定します。
指定されなかった場合は、簡単な表示になります。

PCI バス上のデバイスをリストします。

文法：pci header <b.d.f>

b.d.f: PCI のデバイスを指定します。
b,d,f はそれぞれ、バス.デバイス.ファンクションの番号を指定します。

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

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

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

b.d.f: PCI のデバイスを指定します。
b,d,f はそれぞれ、バス.デバイス.ファンクションの番号を指定します。

addr: 表示するアドレス

count: 指定した単位での数を 16 進で指定します

PCI コンフィグレーションスペースの内容を表示します。

文法：pci modify[.b, .w, .l] b.d.f addr

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

addr: 変更するアドレス

mm コマンドの PCI 用です。現在の値を表示し、新しい値を入力する様に促

されますので、変更したい場合新しい値を 16 進で入力します。そのまま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。

文法：pci next[.b, .w, .l] b.d.f addr

[.b, .w, .l]: バス幅

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

b.d.f: PCI のデバイスを指定します。

b,d,f はそれぞれ、バス.デバイス.ファンクションの番号を指定します。

addr: 変更するアドレス

nm コマンドの PCI 用です。現在の値を表示し、新しい値を入力する様に促されますので、新しい値を入力します。リターンだけの入力の場合は、書き込みしません。
^C (Ctrl キーを押しながら C)で終了します。

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

[.b, .w, .l]: バス幅

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

b.d.f: PCI のデバイスを指定します。

b,d,f はそれぞれ、バス.デバイス.ファンクションの番号を指定します。

addr: 変更するアドレス

value: 書き込む値

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

5.4.51 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。

5.4.52 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。

5.4.53 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

5.4.54　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

5.4.55　reiserload　　- reiserfs からのファイルのロード

文法：　　reiserload interface dev[:part] [addr] [filename] [bytes]

interface:　　インタフェースを指定します。

ide、scsi、usb、mmc、ace が指定可能ですが、このマニュアルが対象としているボードでは、ide のみが有効なインタフェースです。

dev[:part]:　　デバイス番号(とパーティション番号)を指定します。

パーティション番号を省略した場合は 1 になります。

パーティションがないデバイスではパーティション番号として 0 を指定します。

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。
省略すると環境変数 bootfile の値が使われます。
bytes: ロードするバイト数です。
省略した場合または 0 の場合はファイルサイズになります。

指定したファイルのデータをメモリにロードします。
ファイルのロードが正常に終了すると、環境変数 filesize にロードしたバイト数が設定されます。

注意：このコマンドは、以前は有効にしていましたが、コードサイズが大きくなるので、デフォルトで有効にしてありません。コンパイルできる事のみ確認してあります。
関連： reiserls

5.4.56 reiserls - reiser のファイルリスト

文法： reiserls interface dev[:part] [directory]
interface: インタフェースを指定します。有効なインタフェースは ide だけです。
dev[:part]: デバイス番号(とパーティション番号)を指定します。
パーティション番号を省略した場合は 1 になります。
パーティションがないデバイスではパーティション番号として 0 を
指定します。
directory: リストするディレクトリを指定します。省略すると / です。

指定した reiser ファイルシステムのファイルリストを表示します。

関連： reiserload
注意：このコマンドは、以前は有効にしていましたが、コードサイズが大きくなるので、デフォルトで有効にしてありません。コンパイルできる事のみ確認してあります。

5.4.57 reset - CPU のリセット

文法： reset

再起動します。

5.4.58 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。

5.4.59 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。

5.4.60 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。

5.4.61 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)

5.4.62 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

5.4.63 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。

5.4.64 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

5.4.65　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 bootdevice

値： 任意の文字列

初期値：　なし

diskboot コマンドで dev を省略した場合のデフォルトのデバイスを指定します。

5.5.11　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。

5.5.12　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。

5.5.13　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。

5.5.14　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。

5.5.15　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。

5.5.16 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。

5.5.17 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。

5.5.18 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。

5.5.19 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。

5.5.20 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。

5.5.21 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。

5.5.22 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。

5.5.23 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。

5.5.24 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。

5.5.25 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。

5.5.26 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

5.5.27 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

5.5.28 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。

5.5.29 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。

5.5.30 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。

5.5.31 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。

5.5.32 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。

5.5.33 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。

5.5.34 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。

5.5.35 preboot

値： 任意の文字

初期値： "echo \"\n" \
"Type \"boot\" for default way\n" \
"Type \"run nfs\" to boot from NFS\n" \
"Type \"run nfsinitrd\" to boot with nfs initrd\n\""

自動起動の前に実行するコマンドを設定します。

5.5.36 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。

5.5.37 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。

5.5.38 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。

5.5.39 stdin

値： デバイス名

初期値： serial

標準入力に使うデバイスを設定します。

serial 以外の値の動作は未確認です。

5.5.40 stdout

値： デバイス名

初期値： serial

標準出力に使うデバイスを設定します。

serial 以外の値の動作は未確認です。

5.5.41 stderr

値： デバイス名

初期値： serial

標準エラー出力に使うデバイスを設定します。

serial 以外の値の動作は未確認です。

5.5.42 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。

5.5.43 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。

5.5.44 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://sourceforge.net/projects/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。