PPCBOOT Commands
&
Environment Variables

원 문 : http://www.denx.de 의 PPCBOOT 편

1.1. 명령어 정보

1.1.1. bdinfo – 보드 정보 출력

```
=> help bdinfo
bdinfo - No help available.

=>
```

bdinfo 명령어(줄여서 **bdi**)는 PPCBoot 가 보내는 메모리 주소, 크기, 클럭 주파수, MAC 어드레스 등과 같은 보드에 관란 정보를 출력한다. 이 정보는 주로 리눅스커널에 전달될 때 필요하다.

```
=> bdi
memstart    = 0x00000000
memsize     = 0x04000000
flashstart  = 0x40000000
flashsize   = 0x00800000
flashoffset = 0x00030000
sramstart   = 0x00000000
sramsize    = 0x00000000
immr_base   = 0xFFF00000
bootflags   = 0x00000001
intfreq     =     50 MHz
busfreq     =     50 MHz
ethaddr     = 00:D0:93:00:28:81
IP addr     = 10.0.0.99
baudrate    = 115200 bps
=>
```

1.1.2. coninfo – 콘솔 장치와 정보 출력

```
=> help conin
coninfo
=>
```

coninfo 명령어(줄여서 **conin**)는 이용가능한 콘솔 입/출력 장치에 대한 정보를 출력한다.

```
=> conin
List of available devices:
serial   80000003 SIO stdin stdout stderr
=>
```

출력에는 장치 이름, 플래그값과 현재 사용법이 포함된다. 예를 들어 아래의 출력은

```
Serial   80000003 SIO stdin stdout stderr
```

Input(flag **'I'**)와 output(flag '**O**')를 제공하고, 3 개의 표준 입출력 스트림 **stdin, stdout** 과 **stderr** 를 지정하는 시스템 장치(flag '**S**')인 **serial** 장치를 의미한다.

1.1.3. flinfo – FLASH 메모리 정보 출력

```
=> help flinfo
flinfo
    - print information for all FLASH memory banks
flinfo N
    - print information for FLASH memory bank # N

=>
```

flinfo 명령어(줄여서 **fli**)는 이용가능한 플래쉬 메모리에 대한 정보를 얻는데 사용되어진다.

```
=> fli

Bank # 1: FUJITSU AM29LV160B (16 Mbit, bottom boot sect)
  Size: 4 MB in 35 Sectors
  Sector Start Addresses:
    40000000 (RO) 40008000 (RO) 4000C000 (RO) 40010000 (RO) 40020000
(RO)
    40040000      40060000      40080000      400A0000      400C0000
    400E0000      40100000      40120000      40140000      40160000
```

```
    40180000      401A0000      401C0000      401E0000      40200000
    40220000      40240000      40260000      40280000      402A0000
    402C0000      402E0000      40300000      40320000      40340000
    40360000      40380000      403A0000      403C0000      403E0000

Bank # 2: FUJITSU AM29LV160B (16 Mbit, bottom boot sect)
  Size: 4 MB in 35 Sectors
  Sector Start Addresses:
    40400000      40408000      4040C000      40410000      40420000
    40440000      40460000      40480000      404A0000      404C0000
    404E0000      40500000      40520000      40540000      40560000
    40580000      405A0000      405C0000      405E0000      40600000
    40620000      40640000      40660000      40680000      406A0000
    406C0000      406E0000      40700000      40720000      40740000
    40760000      40780000      407A0000      407C0000      407E0000
=>
```

1.1.4. iminfo – application 이미지의 헤더정보 출력

```
=> help iminfo
iminfo addr [addr ...]
    - print header information for application image starting at
      address 'addr' in memory; this includes verification of the
      image contents (magic number, header and payload checksums)

=>
```

iminfo 명령어(줄여서 **imi**)는 리눅스 커널이나 램디스크 같은 이미지에 대한 헤더 정보를 출력하는데 사용된다. 또한, 이미지 이름, 형태, 크기와 이미지 내에 저장되어 있는 CRC32 체크섬이 안전한지를 나타낸다.

```
=> imi 100000

## Checking Image at 00100000 ...
   Image Name:   Linux-2.4.4
   Created:      2002-04-07  21:31:59 UTC
```

```
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    605429 Bytes = 591 kB = 0 MB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
=>
```

Note: 다른 명령어들처럼, 이 명령어의 정확한 동작은 PPCBoot 환경변수들의 셋팅에 의해서 조절되어진다.(**verify** 변수)

1.1.5. help – 도움말 출력

```
=> help help
help [command ...]
    - show help information (for 'command')
'help' prints online help for the monitor commands.

Without arguments, it prints a short usage message for all commands.

To get detailed help information for specific commands you can type
'help' with one or more command names as arguments.

=>
```

help 명령어(줄여서 **h** or **?**) 는 도움말을 출력한다. 인자가 없으면 PPCBoot 환경에서 이용할 수 있는 모든 PPCBoot 명령어들의 리스트를 출력한다. **help** 명령어의 인자로 특정한 명령어의 이름을 넣으면 자세한 설명을 얻을 수 있다.

```
=> help protect
protect on  start end
    - protect FLASH from addr 'start' to addr 'end'
protect on  N:SF[-SL]
    - protect sectors SF-SL in FLASH bank # N
protect on  bank N
    - protect FLASH bank # N
protect on  all
```

```
    - protect all FLASH banks
protect off start end
    - make FLASH from addr 'start' to addr 'end' writable
protect off N:SF[-SL]
    - make sectors SF-SL writable in FLASH bank # N
protect off bank N
    - make FLASH bank # N writable
protect off all
    - make all FLASH banks writable

=>
```

1.2. 메모리 명령어

1.2.1. base – 주소 offset 값을 출력, 지정

```
=> help base
base
    - print address offset for memory commands
base off
    - set address offset for memory commands to 'off'

=>
```

base 명령어(줄여서 **ba**)는 모든 메모리 명령에 대한 주소 offset 값으로 사용되어지는 “base address”를 출력, 지정하는데 사용한다. Base address 의 디폴트 값은 0 이다. 입력하는 모든 주소는 수정없이 사용되어진다. 그러나, 어떤 메모리 지역(PowerPC 프로세서에서 내부 메모리 같은)을 접근해야 할 때, 그 지역의 시작에 base address 를 지정하고 단지 offset 값을 이용하면 매우 편리하게 이용할 수 있다.

```
=> base
Base Address: 0x00000000
=> md 0 c
00000000: feffffff 00000000 7cbd2b78 7cdc3378    ........|.+x|.3x
```

```
00000010: 3cfb3b78 3b000000 7c0002e4 39000000    <.;x;...|...9...
00000020: 7d1043a6 3d000400 7918c3a6 3d00c000    }.C.=...y...=...
=> base 40000000
Base Address: 0x40000000
=> md 0 c
40000000: 27051956 50504342 6f6f7420 312e312e    '..VPPCBoot 1.1.
40000010: 3520284d 61722032 31203230 3032202d    5 (Mar 21 2002 -
40000020: 2031393a 35353a30 34290000 00000000     19:55:04)......
=>
```

1.2.2. crc32 – 체크섬 계산

```
=> help crc32
crc32 address count [addr]
    - compute CRC32 checksum [save at addr]

=>
```

crc32 명령어(줄여서 **crc**)는 메모리 범위에 CRC32 체크섬을 계산하는데 사용되어진다.

```
=> crc 100004 3FC
CRC32 for 00100004 ... 001003ff ==> d433b05b
=>
```

3 개의 인자를 가지고 사용하면, 주어진 주소에 계산된 체크섬을 저장한다.

```
=> crc 100004 3FC 100000
CRC32 for 00100004 ... 001003ff ==> d433b05b
=> md 100000 4
00100000: d433b05b ec3827e4 3cb0bacf 00093cf5    .3.[.8'.<.....<.
=>
```

보이는 것과 같이, CRC32 체크섬은 출력될 뿐만 아니라, 0x100000 번지에 저장되어진다.

1.2.3. cmp – 메모리 비교

```
=> help cmp
cmp [.b, .w, .l] addr1 addr2 count
    - compare memory

=>
```

cmp 명령어는 두 메모리 지역의 내용이 동일한지 아닌지를 테스트하는데 사용된다. 이 명령어는 세번째(길이) 인자에 의해서, 전체 범위를 테스트 할 것인지, 첫번째 차이점이 발견되면 멈출 것인지 결정한다.

```
=> cmp 100000 40000000 400
word at 0x00100004 (0x50ff4342) != word at 0x40000004 (0x50504342)
Total of 1 word were the same
=> md 100000 C
00100000: 27051956 50ff4342 6f6f7420 312e312e    '..VP.CBoot 1.1.
00100010: 3520284d 61722032 31203230 3032202d    5 (Mar 21 2002 -
00100020: 2031393a 35353a30 34290000 00000000     19:55:04)......
=> md 40000000 C
40000000: 27051956 50504342 6f6f7420 312e312e    '..VPPCBoot 1.1.
40000010: 3520284d 61722032 31203230 3032202d    5 (Mar 21 2002 -
40000020: 2031393a 35353a30 34290000 00000000     19:55:04)......
=>
```

다른 메모리 명령어들처럼 **cmp** 는 다른 크기로 메모리를 제어할 수 있다. 32bit(long word), 16bit(word) 또는 8bit(byte) 단위. 만약, 디폴트 사이즈(32bit or long words)가 사용되면 대신 **cmp.l** 을 사용할 수 있다. 만약 16bit 나 word 단위로 접근을 원한다면 **cmp.w** 를 사용할 수 있다. 그리고 8bit 나 byte 단위는 **cmp.b** 를 사용할 수 있다.

Note: *count* 인자는 처리할 데이터 항목의 개수, 즉 비교할 long words 나 words 나 bytes 의 개수를 지정한다.

```
=> cmp.l 100000 40000000 400
word at 0x00100004 (0x50ff4342) != word at 0x40000004 (0x50504342)
Total of 1 word were the same
=> cmp.w 100000 40000000 800
```

```
halfword at 0x00100004 (0x50ff) != halfword at 0x40000004 (0x5050)
Total of 2 halfwords were the same
=> cmp.b 100000 40000000 1000
byte at 0x00100005 (0xff) != byte at 0x40000005 (0x50)
Total of 5 bytes were the same
=>
```

1.2.4. cp – 메모리 복사

```
=> help cp
cp [.b, .w, .l] source target count
    - copy memory

=>
```

cp 는 메모리 구역을 복사는데 사용되어진다.

```
=> cp 40000000 100000 10000
=>
```

cp 는 확장자 **.l**, **.w** 와 **.b** 를 가진다.

```
=> cp.l 40000000 100000 10000
=> cp.w 40000000 100000 20000
=> cp.b 40000000 100000 40000
=>
```

1.2.5. md – 메모리 출력

```
=> help md
md [.b, .w, .l] address [# of objects]
    - memory display

=>
```

md 는 Hex 값과 ASCII 값으로 메모리 내용을 출력하는데 사용되어진다.

```
=> md 100000
00100000: 27051956 50504342 6f6f7420 312e312e    '..VPPCBoot 1.1.
00100010: 3520284d 61722032 31203230 3032202d    5 (Mar 21 2002 -
00100020: 2031393a 35353a30 34290000 00000000     19:55:04).....
00100030: 00000000 00000000 00000000 00000000    ................
00100040: 00000000 00000000 00000000 00000000    ................
00100050: 00000000 00000000 00000000 00000000    ................
00100060: 00000000 00000000 00000000 00000000    ................
00100070: 00000000 00000000 00000000 00000000    ................
00100080: 00000000 00000000 00000000 00000000    ................
00100090: 00000000 00000000 00000000 00000000    ................
001000a0: 00000000 00000000 00000000 00000000    ................
001000b0: 00000000 00000000 00000000 00000000    ................
001000c0: 00000000 00000000 00000000 00000000    ................
001000d0: 00000000 00000000 00000000 00000000    ................
001000e0: 00000000 00000000 00000000 00000000    ................
001000f0: 00000000 00000000 00000000 00000000    ................
=>
00100100: 3c60fff0 7c7e9ba6 3aa00001 4800000c    <`..|~..:...H...
00100110: 3aa00002 48000004 38601002 7c600124    :...H...8`..|`.$
00100120: 7c7b03a6 7c7422a6 7c000278 7c1c23a6    |{..|t".|..x|.#.
00100130: 7c1d23a6 7c1623a6 7c1723a6 7c708aa6    |.#.|.#.|.#.|p..
00100140: 7c788aa6 3c600a00 7c708ba6 7c788ba6    |x..<`..|p..|x..
00100150: 3c600c00 7c708ba6 7c788ba6 3c600400    <`..|p..|x..<`..
00100160: 7c788ba6 3c600200 7c708ba6 7c0002e4    |x..<`..|p..|...
00100170: 4c00012c 3c604000 60630000 38630188    L..,<`@.`c..8c..
00100180: 7c6803a6 4e800020 3c60fff0 60612ec0    |h..N.. <`..`a..
00100190: 9401fffc 9401fffc 38400007 7c5e23a6    ........8@..|^#.
001001a0: 3c400000 60420000 7c5523a6 48000005    <@..`B..|U#.H...
001001b0: 7dc802a6 800e22bc 7dc07214 48019d41    }.....".}.r.H..A
001001c0: 7ea3ab78 4800c05d 00000000 00000000    ~..xH..].........
001001d0: 00000000 00000000 00000000 00000000    ................
001001e0: 00000000 00000000 00000000 00000000    ................
001001f0: 00000000 00000000 00000000 00000000    ................
=>
```

이 명령어 또한 확장자 **.l**, **.w** 와 **.b** 를 가지고 사용된다.

```
=> md.w 100000
00100000: 2705 1956 5050 4342 6f6f 7420 312e 312e    '..VPPCBoot 1.1.
00100010: 3520 284d 6172 2032 3120 3230 3032 202d    5 (Mar 21 2002 -
00100020: 2031 393a 3535 3a30 3429 0000 0000 0000     19:55:04)......
00100030: 0000 0000 0000 0000 0000 0000 0000 0000    ................
00100040: 0000 0000 0000 0000 0000 0000 0000 0000    ................
00100050: 0000 0000 0000 0000 0000 0000 0000 0000    ................
00100060: 0000 0000 0000 0000 0000 0000 0000 0000    ................
00100070: 0000 0000 0000 0000 0000 0000 0000 0000    ................
=> md.b 100000
00100000: 27 05 19 56 50 50 43 42 6f 6f 74 20 31 2e 31 2e
'..VPPCBoot 1.1.
00100010: 35 20 28 4d 61 72 20 32 31 20 32 30 30 32 20 2d    5 (Mar 21
2002 -
00100020: 20 31 39 3a 35 35 3a 30 34 29 00 00 00 00 00 00
19:55:04)......
00100030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00    ................
=>
```

Note: 마지막 출력된 주소와 카운트 인자값은 저장되기 때문에 인자없이 md 명령어를 입력하면 자동적으로 다음 주소에서 계속되며, 같은 카운트를 계속 사용한다.

```
=> md.b 100000 20
00100000: 27 05 19 56 50 50 43 42 6f 6f 74 20 31 2e 31 2e
'..VPPCBoot 1.1.
00100010: 35 20 28 4d 61 72 20 32 31 20 32 30 30 32 20 2d    5 (Mar 21
2002 -
=> md.w 100000
00100000: 2705 1956 5050 4342 6f6f 7420 312e 312e    '..VPPCBoot 1.1.
00100010: 3520 284d 6172 2032 3120 3230 3032 202d    5 (Mar 21 2002 -
00100020: 2031 393a 3535 3a30 3429 0000 0000 0000     19:55:04)......
```

```
00100030: 0000 0000 0000 0000 0000 0000 0000 0000    ................
=> md 100000
00100000: 27051956 50504342 6f6f7420 312e312e    '..VPPCBoot 1.1.
00100010: 3520284d 61722032 31203230 3032202d    5 (Mar 21 2002 -
00100020: 2031393a 35353a30 34290000 00000000     19:55:04)......
00100030: 00000000 00000000 00000000 00000000    ................
00100040: 00000000 00000000 00000000 00000000    ................
00100050: 00000000 00000000 00000000 00000000    ................
00100060: 00000000 00000000 00000000 00000000    ................
00100070: 00000000 00000000 00000000 00000000    ................
=>
```

1.2.6. mm – 메모리 수정 (자동증가)

```
=> help md
md [.b, .w, .l] address [# of objects]
    - memory display

=>
```

mm 은 메모리 내용을 수정하는 방법이다. 주소와 현재의 내용을 표시하고, 사용자 입력을 위하여 대기한다. 적당한 Hex 값을 입력하면 그 주소에 쓰여질 것이다. 그리고 다음 주소로 넘어갈 것이다. 만일, 아무 값없이 ENTER 만 누를경우, 그 주소의 내용은 변함없이 남게된다. 명령은 Hex 값이 아닌(. 과 같은) 자료를 입력하면 멈추게 된다.

```
=> mm 100000
00100000: 27051956 ? 0
00100004: 50504342 ? AABBCCDD
00100008: 6f6f7420 ? 01234567
0010000c: 312e312e ? .
=> md 100000 10
00100000: 00000000 aabbccdd 01234567 312e312e    .........#Eg1.1.
00100010: 3520284d 61722032 31203230 3032202d    5 (Mar 21 2002 -
00100020: 2031393a 35353a30 34290000 00000000     19:55:04)......
00100030: 00000000 00000000 00000000 00000000    ................
```

```
=>
```

이 명령어 또한 확장자 .l, .w 와 .b 를 가지고 사용된다.

```
=> mm.w 100000
00100000: 0000 ? 0101
00100002: 0000 ? 0202
00100004: aabb ? 4321
00100006: ccdd ? 8765
00100008: 0123 ? .
=> md 100000 10
00100000: 01010202 43218765 01234567 312e312e    ....C!.e.#Eg1.1.
00100010: 3520284d 61722032 31203230 3032202d    5 (Mar 21 2002 -
00100020: 2031393a 35353a30 34290000 00000000     19:55:04)......
00100030: 00000000 00000000 00000000 00000000    ................
=>
=> mm.b 100000
00100000: 01 ? 48
00100001: 01 ? 61
00100002: 02 ? 6c
00100003: 02 ? 6c
00100004: 43 ? 6f
00100005: 21 ? 20
00100006: 87 ? 20
00100007: 65 ? 20
00100008: 01 ? .
=> md 100000 10
00100000: 48616c6c 6f202020 01234567 312e312e    Hallo   .#Eg1.1.
00100010: 3520284d 61722032 31203230 3032202d    5 (Mar 21 2002 -
00100020: 2031393a 35353a30 34290000 00000000     19:55:04)......
00100030: 00000000 00000000 00000000 00000000    ................
=>
```

1.2.7. mtest – 단순 RAM 테스트

```
=> help mtest
```

```
mtest [start [end [pattern]]]
    - simple RAM read/write test

=>
```

mtest 는 단순 메모리 테스트를 제공한다.

```
=> mtest 100000 200000
Testing 00100000 ... 00200000:
Pattern 0000000F  Writing...  Reading...
=>
```

Note: 이 테스트는 메모리 내용을 수정하면서 저장한다. ROM 이나 플래쉬 메모리에 적용할 경우 실패하게 된다.

주 의
이 명령어는 테스트하는 메모리 범위가 PPCBoot 펌웨어의 작동에 필요한 구역(Exception vector code 나 PPCBoot 의 내부 프로그램 코드, 스택 또는 힙 메모리 구역와 같은) 을 포함할 경우, 시스템을 파괴할 수도 있다.

1.2.8. mw – 메모리 쓰기

```
=> help mw
mw [.b, .w, .l] address value [count]
    - write memory

=>
```

mw 는 어떤 값으로 메모리를 초기화하는 방법이다. 카운트 인자없이 사용될 때, 이 값들은 특정 주소에만 쓰여진다. 카운트를 가지고 사용할 경우, 전체 메모리 지역은 이 값으로 초기화되어진다.

```
=> md 100000 10
00100000: 0000000f 00000010 00000011 00000012    ................
00100010: 00000013 00000014 00000015 00000016    ................
```

```
00100020: 00000017 00000018 00000019 0000001a    ................
00100030: 0000001b 0000001c 0000001d 0000001e    ................
=> mw 100000 aabbccdd
=> md 100000 10
00100000: aabbccdd 00000010 00000011 00000012    ................
00100010: 00000013 00000014 00000015 00000016    ................
00100020: 00000017 00000018 00000019 0000001a    ................
00100030: 0000001b 0000001c 0000001d 0000001e    ................
=> mw 100000 0 6
=> md 100000 10
00100000: 00000000 00000000 00000000 00000000    ................
00100010: 00000000 00000000 00000015 00000016    ................
00100020: 00000017 00000018 00000019 0000001a    ................
00100030: 0000001b 0000001c 0000001d 0000001e    ................
=>
```

이 명령어 또한 확장자 .l, .w 와 .b 를 가지고 사용된다.

```
=> mw.w 100004 1155 6
=> md 100000 10
00100000: 00000000 11551155 11551155 11551155    .....U.U.U.U.U.U
00100010: 00000000 00000000 00000015 00000016    ................
00100020: 00000017 00000018 00000019 0000001a    ................
00100030: 0000001b 0000001c 0000001d 0000001e    ................
=> mw.b 100007 ff 7
=> md 100000 10
00100000: 00000000 115511ff ffffffff ffff1155    .....U.........U
00100010: 00000000 00000000 00000015 00000016    ................
00100020: 00000017 00000018 00000019 0000001a    ................
00100030: 0000001b 0000001c 0000001d 0000001e    ................
=>
```

1.2.9. nm – 메모리 수정

```
=> help nm
nm [.b, .w, .l] address
```

```
    - memory modify, read and keep address

=>
```

nm 명령어(자동증가하지 않는 memory modify) 는 같은 주소에 몇번의 다른 데이터를 기록할 때 사용된다. 장치 레지스터에 접근, 수정에 편리하게 사용된다.

```
=> nm.b 100000
00100000: 00 ? 48
00100000: 48 ? 61
00100000: 61 ? 6c
00100000: 6c ? 6c
00100000: 6c ? 6f
00100000: 6f ? .
=> md 100000 8
00100000: 6f000000 115511ff ffffffff ffff1155    o....U.........U
00100010: 00000000 00000000 00000015 00000016    ................
=>
```

이 명령어 또한 확장자 **.l**, **.w** 와 **.b** 를 가지고 사용된다.

1.2.10. loop – 주소범위에 무한루프

```
=> help loop
loop [.b, .w, .l] address number_of_objects
    - loop on a set of addresses

=>
```

loop 명령어는 메모리 범위의 루프안에서 읽는다. 가능한 빨리 메모리로부터 읽어내려고 하기 때문에, 메모리 테스트의 특별한 형태로 나타난다.

주 의
이 명령어는 결코 멈출 수 없다. 보드를 리셋하는 방법 외에 다른 방법은 없다.

```
=> loop 100000 8
```

1.3. 플래쉬 메모리 명령어

1.3.1. cp – 메모리 복사

```
=> help cp
cp [.b, .w, .l] source target count
    - copy memory

=>
```

cp 는 플래쉬 메모리 구역에 대하여 알고 있으며, 목표 구역이 플래쉬 메모리 안에 있을 경우, 자동적으로 필요한 플래쉬 프로그래밍 알고리즘을 포함한다.

```
=> cp 100000 40000000 10000
Copy to Flash··· done
=>
```

Note: 플래쉬 메모리에 쓰는 것은 목표 구역이 지워지지 않았거나, 쓰기방지가 되어있을 경우 실패한다.

```
=> cp 100000 40000000 10000
Copy to Flash··· Can't write to protected Flash sectors
=>
```

주 의
카운트 인자가 복사할 항목의 개수를 지정한다는 것을 기억하라. 만일, 대신 길이만 알고 있을 경우 **cp.b** 를 사용하거나 정확한 항목의 개수를 계산해야한다.

1.3.2. flinfo – 플래쉬 메모리 정보 출력

flinfo(줄여서 **fli**) 명령어는 이용가능한 플래쉬 메모리에 대한 정보를 얻는데 사용된다. 플래쉬 bank 의 개수는 크기, 플래쉬 섹터 조직, 삭제한 정보와 함께 출력된다. 모든 섹터에서 시작주소는 출력되어지며, 어떤 섹터는 쓰기방지가 되어있거나, 읽기전용("**(RO)**")로 나타난다.

```
⇨ fli
```

```
   ⇨
Bank # 1: FUJITSU AM29LV160B (16 Mbit, bottom boot sect)
  Size: 4 MB in 35 Sectors
  Sector Start Addresses:
    40000000 (RO) 40008000 (RO) 4000C000 (RO) 40010000 (RO) 40020000
(RO)
    40040000      40060000      40080000      400A0000      400C0000
    400E0000      40100000      40120000      40140000      40160000
    40180000      401A0000      401C0000      401E0000      40200000
    40220000      40240000      40260000      40280000      402A0000
    402C0000      402E0000      40300000      40320000      40340000
    40360000      40380000      403A0000      403C0000      403E0000

Bank # 2: FUJITSU AM29LV160B (16 Mbit, bottom boot sect)
  Size: 4 MB in 35 Sectors
  Sector Start Addresses:
    40400000      40408000      4040C000      40410000      40420000
    40440000      40460000      40480000      404A0000      404C0000
    404E0000      40500000      40520000      40540000      40560000
    40580000      405A0000      405C0000      405E0000      40600000
    40620000      40640000      40660000      40680000      406A0000
    406C0000      406E0000      40700000      40720000      40740000
    40760000      40780000      407A0000      407C0000      407E0000
=>
```

1.3.3. erase – 플래쉬 메모리 삭제

```
=> help era
erase start end
    - erase FLASH from addr ‘start’ to addr ‘end’
erase N:SF[-SL]
    - erase sectors SF-SL in FLASH bank # N
erase bank N
    - erase FLASH bank # N
erase all
    - erase all FLASH banks
```

```
=>
```

Erase 명령어는(줄여서 **era**) 는 한 개나 그 이상의 플래쉬 메모리의 섹터 내용을 지우는데 사용된다. 복잡한 명령어 중의 하나이다.

아마 이 명령어에서 가장 자주 쓰여지는 문법은 삭제할 주소의 처음과 끝을 나타내는 문법이다.

```
   ⇨ era 40040000 402FFFFF
   ⇨ Erase Flash from 0x40040000 to 0x402fffff
…… done
Erased 22 sectors
=>
```

Note: 이 명령어에서 시작과 끝 주소는 플래쉬섹터의 시작과 끝을 정확하게 가리켜야 한다. 그렇지 않으면 작동하지 않을 것이다.

삭제 명령에서 플래쉬 메모리의 확실한 구역을 선택하는 다른 방법은 플래쉬 bank 와 섹터의 표기를 이용한 방법이다.

기술적으로, bank 는 CPU 의 동일한 chip select 신호에 연결된 하나 또는 이상의 메모리 칩에 의해서 실행되는 메모리의 구역이다. 플래쉬 섹터나 erase unit 은 한 동작에서 삭제되는 가장 작은 구역이다.

실용적인 목적에서, 한번의 동작에서 결국에 어떤 플래쉬 bank 가 지워지는지는 충분히 기억할만하다. 이것은 아마 섹터로 같은 구역 섹터를 지우는 것보다 더 빠르고 효과적일 것이다.

PPCBoot 에서, 플래쉬 섹터가 0 부터 시작하는 반면에, 플래쉬 bank 는 1 부터 시작한다.

위의 예제에서 시작과 끝 주소로 사용된 동일한 플래쉬 구역을 지우는 방법은 아래와 같이도 사용된다.

```
   ⇨ era 1:6-8
   ⇨ Erase Flash Sectors 6-8 in Bank # 1
.. done
```

```
=>
```

전체 플래쉬 메모리를 지우기 위해서 다음과 같이 사용한다.

```
   ⇨ era bank 1
   ⇨ Erase Flash Bank # 1 ? Warning: 5 protected sectors will not be
     erased!
……..... done
=>
```

Note: 플래쉬 bank 안에 지울 수 없는 write protected(쓰기방지) 섹터가 존재하기 때문에 경고 메시지가 나타났다.

명령어 :

```
   ⇨ era all
   ⇨ Erase Flash Bank # 1 ? Warning: 5 protected sectors will not be
     erased!
……. Done
Erase Flash Bank # 2
……. Done
=>
```

전체 플래쉬 메모리(쓰기방지 섹터를 제외한)가 삭제된다.

1.3.4. protect – 쓰기가능, 방지 플래쉬

```
=> help protect
protect on  start end
    - protect FLASH from addr ‘start’ to addr ‘end’
protect on  N:SF[-SL]
    - protect sectors SF-SL in FLASH bank # N
protect on  bank N
    - protect FLASH bank # N
protect on  all
    - protect all FLASH banks
```

```
protect off start end
    - make FLASH from addr ‘start’ to addr ‘end’ writable
protect off N:SF[-SL]
    - make sectors SF-SL writable in FLASH bank # N
protect off bank N
    - make FLASH bank # N writable
protect off all
    - make all FLASH banks writable

=>
```

Protect 명령어는 다른 복잡한 명령어 중의 하나이다. 이것은 플래쉬 메모리의 어느 부분을 읽기전용 또는 다시 쓰기가능하게 지정하는데 사용되어진다.
“쓰기방지”(=읽기전용)인 플래쉬 메모리는 쓰거나(**cp**) 지울 수 없다.(**erase**)
쓰기방지로 보호되어지는 구역은 **flinfo** 명령어의 출력에서 (**RO**)(읽기전용) 로 나타난다.

```
   ⇨ fli
   ⇨
Bank # 1: FUJITSU AM29LV160B (16 Mbit, bottom boot sect)
  Size: 4 MB in 35 Sectors
  Sector Start Addresses:
    40000000 (RO) 40008000 (RO) 4000C000 (RO) 40010000 (RO) 40020000
(RO)
    40040000      40060000      40080000      400A0000      400C0000
    400E0000      40100000      40120000      40140000      40160000
    40180000      401A0000      401C0000      401E0000      40200000
    40220000      40240000      40260000      40280000      402A0000
    402C0000      402E0000      40300000      40320000      40340000
    40360000      40380000      403A0000      403C0000      403E0000

Bank # 2: FUJITSU AM29LV160B (16 Mbit, bottom boot sect)
  Size: 4 MB in 35 Sectors
  Sector Start Addresses:
    40400000      40408000      4040C000      40410000      40420000
    40440000      40460000      40480000      404A0000      404C0000
```

```
    404E0000      40500000      40520000      40540000      40560000
    40580000      405A0000      405C0000      405E0000      40600000
    40620000      40640000      40660000      40680000      406A0000
    406C0000      406E0000      40700000      40720000      40740000
    40760000      40780000      407A0000      407C0000      407E0000
=> protect on 40100000 401FFFFF
Protected 8 sectors
   ⇨ fli
   ⇨
Bank # 1: FUJITSU AM29LV160B (16 Mbit, bottom boot sect)
  Size: 4 MB in 35 Sectors
  Sector Start Addresses:
    40000000 (RO) 40008000 (RO) 4000C000 (RO) 40010000 (RO) 40020000
(RO)
    40040000      40060000      40080000      400A0000      400C0000
    400E0000      40100000 (RO) 40120000 (RO) 40140000 (RO) 40160000
(RO)
    40180000 (RO) 401A0000 (RO) 401C0000 (RO) 401E0000 (RO) 40200000
    40220000      40240000      40260000      40280000      402A0000
    402C0000      402E0000      40300000      40320000      40340000
    40360000      40380000      403A0000      403C0000      403E0000

Bank # 2: FUJITSU AM29LV160B (16 Mbit, bottom boot sect)
  Size: 4 MB in 35 Sectors
  Sector Start Addresses:
    40400000      40408000      4040C000      40410000      40420000
    40440000      40460000      40480000      404A0000      404C0000
    404E0000      40500000      40520000      40540000      40560000
    40580000      405A0000      405C0000      405E0000      40600000
    40620000      40640000      40660000      40680000      406A0000
    406C0000      406E0000      40700000      40720000      40740000
    40760000      40780000      407A0000      407C0000      407E0000
   ⇨ era 40100000 401FFFFF
   ⇨ Erase Flash from 0x40100000 to 0x401fffff ? Warning: 8 protected
     sectors will not be erased!
 Done
```

```
Erased 8 sectors
=> protect off 1:11
Un-Protect Flash Sectors 11-11 in Bank # 1
   ⇨ fli
   ⇨
Bank # 1: FUJITSU AM29LV160B (16 Mbit, bottom boot sect)
  Size: 4 MB in 35 Sectors
  Sector Start Addresses:
    40000000 (RO) 40008000 (RO) 4000C000 (RO) 40010000 (RO) 40020000
(RO)
    40040000      40060000      40080000      400A0000      400C0000
    400E0000      40100000      40120000 (RO) 40140000 (RO) 40160000
(RO)
    40180000 (RO) 401A0000 (RO) 401C0000 (RO) 401E0000 (RO) 40200000
    40220000      40240000      40260000      40280000      402A0000
    402C0000      402E0000      40300000      40320000      40340000
    40360000      40380000      403A0000      403C0000      403E0000

Bank # 2: FUJITSU AM29LV160B (16 Mbit, bottom boot sect)
  Size: 4 MB in 35 Sectors
  Sector Start Addresses:
    40400000      40408000      4040C000      40410000      40420000
    40440000      40460000      40480000      404A0000      404C0000
    404E0000      40500000      40520000      40540000      40560000
    40580000      405A0000      405C0000      405E0000      40600000
    40620000      40640000      40660000      40680000      406A0000
    406C0000      406E0000      40700000      40720000      40740000
    40760000      40780000      407A0000      407C0000      407E0000
   ⇨ era 1:11
   ⇨ Erase Flash Sectors 11-11 in Bank # 1
. done
=>
```

Note: 쓰기방지의 실제 수준는 사용자의 하드웨어에 사용된 플래쉬칩들과 보드의 플래쉬 장치드라이버의 수행에 의존한다. 대부분의 경우, PPCBoot 는 단지 간단한

소프트웨어 보호 즉, 실수로 인하여 중요한 부분(PPCBoot 자체 또는 환경변수) 이 지워지거나 덮어씌워지는 것을 방지할 뿐이다. 그러나, 그 자신의 플래쉬 드라이버 코드를 읽고 가동시키려는 악의의 사용자를 이 메커니즘으로부터 보호하지는 못한다. 또한, 대부분의 경우 이 보호는 PPCBoot 가 동작하는 동안에만 효과적이다. 즉, 다른 운영체제는 쓰기방지 플래쉬 구역에 대해서 알지 못하며, 요청이 있으면 지워버릴 것이다.

1.4. 제어 실행 명령어

1.4.1. autoscr – 메모리로부터 스크립트 동작

```
=> help autoscr
autoscr [addr] ? run script starting at addr. A valid autoscr header
must be present

=>
```

Autoscr 명령어를 가지고 PPCBoot 에서 쉘 스크립트를 실행할 수 있다.
실행하려는 명령어를 텍스트 파일에 단지 적는 것만으로 PPCBoot 스크립트를 만들 수 있다. 텍스트 파일을 PPCBoot 이미지(“**script**”형태의 이미지를 사용하여) 로 바꾸는 도구인 **mkimage** 를 사용해야한다. 이 이미지는 다른 이미지 파일처럼 읽어들일 수 있다. 그리고, **autoscr** 명령어로 이미지를 실행할 수 있다. 다음의 텍스트 파일을 예를 들어본다.

```
Bash$ cat /tftpboot/TQM860L/example.script
Echo
Echo Network Configuration:
Echo ----------------------
Echo Target:
Printenv ipaddr hostname
Echo
Echo Server:
Printenv serverip rootpath
Echo
```

이처럼 **mkimage** 명령어를 이용하여 PPCBoot 스크립트 이미지로 변환할 수 있다.

```
Bash$ mkimage ?A ppc ?O linux ?T script ?C none ?a 0 ?e 0 \
> -n “autoscr example script” \
> -d /tftpboot/TQM860L/example.script /tftpboot/TQM860L/example.img
Image Name:   autoscr example script
Created:      Mon 멕  8 01:15:02 2002
Image Type:   PowerPC Linux Script (uncompressed)
Data Size:    157 Bytes = 0.15 kB = 0.00 MB
Load Address: 0x00000000
Entry Point:  0x00000000
Contents:
   Image 0:      149 Bytes =    0 kB = 0 MB
```

PPCBoot 에서 스크립트 이미지를 읽어들여 실행할 수 있다.

```
   ⇨ tftp 100000 /tftpboot/TQM860L/example.img
   ⇨ ARP broadcast 1
TFTP from server 10.0.0.2; our IP address is 10.0.0.99
Filename ‘/tftpboot/TQM860L/example.img’.
Load address: 0x100000
Loading: #
Done
Bytes transferred = 221 (dd hex)
   ⇨ autoscr 100000
   ⇨ ## Executing script at 00100000

Network Configuration:
--------------------------------------------------------------------------------
Target:
Ipaddr=10.0.0.99
Hostname=tqm

Server:
Serverip=10.0.0.2
Rootpath=/opt/hardhat/devkit/ppc/8xx/target

=>
```

1.4.2. bootm – 메모리로부터 application 이미지 부팅

```
=> help bootm
bootm [addr [arg …]]
    -  boot application image stored in memory
    -          passing arguments ‘arg …’; when booting a Linux
       kernel,
        ‘arg’ can be the address of an initrd image

=>
```

Bootm 은 운영체제 이미지를 시작하는데 사용된다. 이미지 헤더로부터 운영체제의 형태, 사용된 파일 압축방법, 포인트 주소를 읽고 들어가는 등의 정보을 얻는다. 명령어는 요구된 메모리 주소로부터, 필요하다면 압축되지 않은 이미지를 읽어들인다. 운영체제에 따라서, 요구된 부트 인자들이 읽어지고, 도입 부분에서 운영체제를 시작한다.

Bootm 에서 첫번째 인자는 운영체제에 의존하는 임의의 인자에 뒤따르는 이미지가 저장되어 있는 메모리 주소이다.(RAM, ROM 또는 플래쉬 메모리에서)

리눅스에서, 정확히 하나의 임의의 인자는 통과할 수 있다. 만일 인자가 존재한다면 램디스크 이미지(RAM, ROM 또는 플래시 메모리에서) 를 **initrd** 하는 시작 주소로 해석된다. **Bootm** 명령어가 세 단계로 구성된다면 첫번째 리눅스 커널 이미지는 압축되지 않고 RAM 으로 복사된다. 그 때, 램디스크 이미지는 RAM 에 읽어진다. 마지막으로 제어권은 램디스크의 위치와 크기에 대한 정보를 통해 리눅스 커널로 넘어가게 된다.

initrd 램디스크 이미지 없이 리눅스 커널을 부팅하려면, 다음 명령어가 사용된다.

```
*** MISSING ***
```

램디스크 이미지가 사용되어진다면, 다음과 같다.

```
*** MISSING ***
```

주 의
램으로부터 부팅 이미지를 읽어올 때(TFTP 를 사용한다고 가정), 압축되지 않은

커널로 메모리가 덮어씌워지지 않게, (압축된) 이미지가 저장된 위치에 주의하여야 한다. 예를들어, low 메모리에서 램디스크 이미지를 읽는다면, 리눅스 커널이 읽어지면서 덮어씌워진다. 이것은 막연한 시스템 파괴를 일으킨다.

1.4.3. go – 'addr' 주소에서 시작

```
=> help go
go addr [arg …]
    - start application at address 'addr'
      passing 'arg' as arguments

=>
```

PPCBoot 는 소위말하는 **standalone applications** 을 지원한다. 이것은 복잡한 환경의 운영체제를 요구하지 않는 것을 말한다. 대신에 PPCBoot 의 콘솔 입출력 또는 malloc() 과 free() 같은 서비스 기능을 이용하여 PPCBoot 에 의해서 읽어지고 실행될 수 있다.

이것은 PPCBoot 에 동적으로 특별한 하드웨어 테스트 루틴이나 파일 시스템 으로부터 OS 이미지를 읽어오는 bootstrap 코드 같은 특별한 확장을 읽어들 이고 실행시키는데 사용되어 질수 있다.

Go 는 standalone 응용 프로그램을 시작할 때 사용되어진다. 임의의 인자들 은 별도의 수정없이 응용프로그램에 적용된다.

1.5. Download 명령어

1.5.1. bootp – BOOTP/TFTP 프로토콜을 통한 boot 이미지

```
=> help bootp
bootp [loadAddress] [bootfilename]

=>
```

1.5.2. dhcp – Dynamic Host Configuration Protocol

```
=> help dhcp
dhcp

=>
```

1.5.3. loadb – 시리얼을 통한 바이너리 파일 load (kermit mode)

```
=> help loadb
loadb [ off ] [ baud ]
    - load binary file over serial line with offset ‘off’ and
baudrate ‘baud’

=>
```

시리얼을 통해서 kermit 으로 바이너리 파일을 다운로드 할 수 있다. 리눅스 커널 이미지인 pImage 를 다운로드 하는 것을 보여준다. 이 때, **kermit** 에 대한 셋팅이 되어 있어야 한다.

```
=> loadb 100000
## Ready for binary (kermit) download ...
Ctrl-\c
(Back at denx.denx.de)
--------------------------------------------------
C-Kermit 7.0.197, 8 Feb 2000, for Linux
 Copyright (C) 1985, 2000,
  Trustees of Columbia University in the City of New York.
Type ? or HELP for help.
Kermit> send /tftpboot/pImage
...
Kermit> connect
Connecting to /dev/ttyS0, speed 115200.
The escape character is Ctrl-\ (ASCII 28, FS)
```

```
Type the escape character followed by C to get back,
or followed by ? to see other options.
-----------------------------------------------
= 550260 Bytes
## Start Addr      = 0x00100000
=> iminfo 100000

## Checking Image at 00100000 ...
   Image Name:   Linux-2.4.4
   Created:      2002-07-02  22:10:11 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    550196 Bytes = 537 kB = 0 MB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
```

1.5.4. loads – 시리얼을 통한 S-Record 파일 load

```
=> help loads
loads [ off ]
    - load S-Record file over serial line with offset 'off'

=>
```

1.5.5. rarpboot - RARP 를 통한 이미지 부팅

```
=> help rarp
rarpboot [loadAddress] [bootfilename]

=>
```

1.5.6. tftpboot- TFTP 를 통한 이미지 부팅

```
=> help tftp
tftpboot [loadAddress] [bootfilename]
```

```
=>
```

1.6. 환경 변수 명령어

1.6.1. printenv- 환경 변수 출력

```
=> help printenv
printenv
    - print values of all environment variables
printenv name ...
    - print value of environment variable 'name'

=>
```

printenv 는 한 개 이상의 PPCBoot 환경 변수를 출력한다. 인자가 주어질 경우, 값들이 인쇄될 환경 변수의 이름으로 해석된다.

```
=> printenv ipaddr hostname netmask
ipaddr=10.0.0.99
hostname=tqm
netmask=255.0.0.0
=>
```

인자가 없다면, **printenv** 는 환경 변수와 환경 변수 값들의 목록, 추가로 이용가능한 메모리의 전체 크기와 현재 사용중인 통계량을 출력한다.

```
=> printenv
baudrate=115200
serial#=TQM860LDDBA3-P50.203 10226122 4
ethaddr=00:D0:93:00:28:81
bootdelay=5
loads_echo=1
clocks_in_mhz=1
load=tftp 100000 /tftpboot/ppcboot.bin
update=protect off all;era 1:0-4;cp.b 100000 40000000 $(filesize);setenv
filesize;saveenv
```

```
rtai=tftp 100000 /tftpboot/pImage.rtai;run nfsargs;run addip;bootm
preboot=echo;echo Type "run flash_nfs" to mount root filesystem over
NFS;echo
nfsargs=setenv bootargs root=/dev/nfs rw nfsroot=$(serverip):$(rootpath)
addip=setenv bootargs $(bootargs)
ip=$(ipaddr):$(serverip):$(gatewayip):$(netmask):$(hostname):$(netdev):off
panic=1
flash_nfs=run nfsargs;run addip;bootm $(kernel_addr)
kernel_addr=40040000
netdev=eth0
hostname=tqm
rootpath=/opt/hardhat/devkit/ppc/8xx/target
ramargs=setenv bootargs root=/dev/ram rw
flash_self=run ramargs;run addip;bootm $(kernel_addr) $(ramdisk_addr)
ramdisk_addr=40100000
bootcmd=run flash_self
stdin=serial
stderr=serial
stdout=serial
filesize=dd
netmask=255.0.0.0
ipaddr=10.0.0.99
serverip=10.0.0.2

Environment size: 992/16380 bytes
=>
```

1.6.2. saveenv – 저장장치에 환경 변수 저장

```
=> help saveenv
saveenv - No help available.

=>
```

PPCBoot 환경을 만든 모든 변수값은 단지 RAM 에만 저장된다. 그 변수값은 시스템을 부팅하면 잃어버리게 된다. 바꾼 변수값을 지속적으로 저장하려면,

시작하는 동안 자동적으로 읽어지는 저장장치에 환경변수를 저장하는 **saveenv** 명령어를 사용해야한다.

```
=> saveenv
Saving Enviroment to Flash...
Un-Protected 1 sectors
Erasing Flash...
. done
Erased 1 sectors
Writing to Flash... done
Protected 1 sectors
=>
```

1.6.3. setenv – 환경 변수 지정

```
=> help setenv
setenv name value ...
    - set environment variable 'name' to 'value ...'
setenv name
    - delete environment variable 'name'

=>
```

PPCBoot 환경을 수정하기 위해서 **setenv** 명령어를 사용해야 한다. 한 개의 인자만 주어질 경우, PPCBoot 환경에서 변수값이 존재한다면 변수값을 삭제할 것이다.

```
=> printenv foo
foo=This is an example value.
=> setenv foo
=> printenv foo
## Error: "foo" not defined
=>
```

인자를 가지고 사용할 경우, 첫번째 인자는 변수의 이름이고, 두번째 인자는 (한 칸의 스페이스로 연결된) 이 변수에 저장될 값을 나타낸다. 새로운 변수는 자동적으로 생성되며, 존재할 경우는 덮어쓰게 된다.

```
=> printenv bar
## Error: "bar" not defined
=> setenv bar This is a new example.
=> printenv bar
bar=This is a new example.
=>
```

변수의 값이 특별한 의미의 특수문자(변수 대체용으로 사용되는 '$" 또는 명령어를 구분하는 세미콜론 같은) 를 가지는 규칙을 기억해야 한다. 백슬러시('\") 는 그러한 특수문자를 빠져나올 때 사용한다.

```
=> setenv cons_opts console=tty0 console=ttyS0,\$(baudrate)
=> printenv cons_opts
cons_opts=console=tty0 console=ttyS0,$(baudrate)
=>
```

Note: 변수 이름으로 사용되어지는 특수문자에는 명령어 라인에서 쓰여지는 제약(백슬러시를 빠져나올 때 사용하거나, 스페이스나 탭을 인자구분으로, 또는 세미콜론과 뉴라인을 명령어 구분으로 사용하는 것 같은) 을 제외 하고는 다른 제약이 없다. 심지어 이상한 입력인 "**-/|()+**"도 PPCBoot 에서는 변수 이름으로 사용되어질 수 있다.

주 의

흔한 실수로는 아래와 같은 경우가 있다.
 	
`setenv name=value`
 	
대신에 아래와 같이 사용한다.
 	
`setenv name value`
 	
제대로 작동된 것처럼 에러 메시지가 뜨지 않는다. 그러나 작동하지 않는다. *'value'*라는 값을 *'name'*이라는 변수에 저장하는 것 대신에, *'name=value'*라는

변수를 지우려 할 것이다. ? 이것은 원하던 바가 아니다. 항상 이름과 값 사이에는 스페이스 그리고/또는 탭으로 분리되어야 한다는 것을 기억하라.

1.6.4. run – 환경 변수를 실행하는 명령어

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

=>
```

PPCBoot 환경 변수는 명령어를 저장하고 명령어를 연속적으로 수행하는데 사용된다. 명령어를 수행하기 위해서는 run 명령어를 사용한다.

```
=> setenv test echo This is a test\;printenv ipaddr\;echo Done.
=> printenv test
test=echo This is a test;printenv ipaddr;echo Done.
=> run test
This is a test
ipaddr=10.0.0.99
Done.
=>
```

연속적으로 수행될 명령어들을 인자로 하여 run 명령어를 수행할 수 있다.

```
=> setenv test2 echo This is another Test\;printenv serial#\;echo
Done.
=> printenv test test2
test=echo This is a test;printenv ipaddr;echo Done.
test2=echo This is another Test;printenv serial#;echo Done.
=> run test test2
This is a test
ipaddr=10.0.0.99
Done.
This is another Test
serial#=TQM860LDDBA3-P50.203 10226122 4
```

```
Done.
=>
```

Note: 몇 개의 명령어들(세미콜론으로 구분된) 을 가진 PPCBoot 변수를 "**run**"을 했는데, 그 중 하나가 실패할 경우, 나머지 명령어들은 **어쨌든 실행이 된다**.

만일, 한 번의 **run** 으로 몇 개의 변수를 실행할 때, 실패한 명령어는 "run"을 끝낸다. 즉, 남아 있는 변수는 **실행되지 않는다**.

1.6.5. bootd - default 부팅, run 'bootcmd'

```
=> help boot
bootd - No help available.

=>
```

bootd (줄여서 **boot**) 는 default 부트 명령어를 실행한다. 즉, 초기 카운트 다운이 아무 간섭을 받지 않았을 때 일어난다. 이것은 "**run bootcmd**" 명령어와 같은 말이다.

1.7. 특수 명령어

1.7.1. i2c - I2C 부시스템

```
=> help i2c
Unknown command 'i2c' - try 'help' without arguments for list of all
known commands

=>
```

1.7.2. ide - IDE 부시스템

```
=> help ide
ide reset - reset IDE controller
ide info  - show available IDE devices
```

```
ide device [dev] - show or set current device
ide part [dev] - print partition table of one or all IDE devices
ide read  addr blk# cnt
ide write addr blk# cnt - read/write `cnt' blocks starting at block
`blk#'
    to/from memory address `addr'

=>
```

1.7.3. diskboot- IDE 장치로부터의 부팅

```
=> help disk
diskboot loadAddr dev:part

=>
```

1.7.4. doc - Disk-On-Chip 부시스템

```
=> help doc
Unknown command 'doc' - try 'help' without arguments for list of all
known commands

=>
```

Disk-On-Chip 플래쉬 디스크를 지원하는 시스템에서, **doc** 명령어는 PPCBoot 로부터 이 장치들을 제어하는 것을 허락한다. help 메시지에서 나타나듯이 이 명령어는 플래쉬 디스크를 가지고 정보를 출력하고, 입/출력을 수행하게 사용되는 부명령어들을 수행한다.

1.7.5. docboot - DOC 장치로부터 부팅

```
=> help doc
Unknown command 'doc' - try 'help' without arguments for list of all
known commands

=>
```

docboot 명령어(줄여서 **docb**)는 Disk-On-Chip 플래쉬 디스크로부터 이미지 (예를들어 리눅스커널)를 읽어오는데 사용된다. **autostart** 환경변수가 "yes"로 지정되있다면, 이미지는 자동으로 시작된다.

1.8. 기타 명령어

1.8.1. date - get/set/reset 날짜 & 시간

```
=> help date
date [MMDDhhmm[[CC]YY][.ss]]
date reset
  - without arguments: print date & time
  - with numeric argument: set the system date & time
  - with 'reset' argument: reset the RTC

=>
```

date 명령어는 표준형식으로 현재 시간을 나타내거나 시스템에 날짜를 지정한다. 시스템에 따라 시스템 클럭을 리셋(초기화)하는데 사용되기도 한다.

```
=> date
Date: 1970-01-01 (Thursday)    Time:  0:-1:-18
=> date 040723152002.35
Date: 2002-04-07 (Sunday)    Time: 23:15:35
=> date reset
Reset RTC...
Date: 2002-04-07 (Sunday)    Time: 23:15:36
=>
```

1.8.2. echo – 콘솔에 인자 출력

```
=> help echo
echo [args..]
    - echo args to console; \c suppresses newline

=>
```

echo 는 콘솔에 인자들을 출력한다.

```
=> echo The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.
=>
```

1.8.3. reset – CPU 의 Reset 수행

```
=> help reset
reset - No help available.

=>
```

reset 은 시스템을 재부팅한다.

```
*** MISSING ***
```

1.8.4. sleep – 일정 시간 실행을 지연

```
=> help sleep
sleep N
    - delay execution for N seconds (N is _decimal_ !!!)

=>
```

sleep 은 인자에서 지정한 초만큼 실행을 정지시킨다.

```
=> date ; sleep 5 ; date
Date: 2002-04-07 (Sunday)    Time: 23:15:40
Date: 2002-04-07 (Sunday)    Time: 23:15:45
=>
```

1.8.5. version – 버전 출력

```
=> help version
version - No help available.
```

```
=>
```

version 명령어(줄여서 **vers**)를 사용해서 현재 시스템에 돌아가는 PPCBoot 이미지의 버전과 제작날짜를 출력할 수 있다.

```
=> version

PPCBoot 1.1.5 (Mar 21 2002 - 19:55:04)
=>
```

1.8.6. ? - 'help'의 다른 명칭

? 를 사용해서 짧은 형태로 help 명령어를 사용할 수 있다.

2.1. PPCBoot 환경 변수

PPCBoot 환경은 PPCBoot가 시작될 때, 저장장치에 보관되었다가 RAM 으로 복사되는 메모리의 블록이다. 시스템 구성에 사용되는 환경 변수를 저장하는데 사용된다. 환경은 CRC32 체크섬에 의해서 보호된다.

이 페이지에는 PPCBoot에서 특별한 의미를 가진 가장 중요한 환경 변수를 다룬다. 환경 변수들은 PPCBoot를 구성하는데 사용되어진다.

autoload

> "no" 로 지정되어 있을 경우('n"으로 시작하는 어뜬 단어), **rarpb**, **bootp** 또는 **dhcp** 명령어는 BOOTP/DHCP 서버로부터 수행된다. 그러나 그렇지 않을 경우, TFTP를 이용한 다른 이미지를 읽어올 것이다.

autostart

> "yes"로 지정되어 있을 경우, **rarpb**, **bootp**, **dhcp**, **tftp**, **disk**, 또는 **docb** 명령어를 사용하여 읽어진 이미지는 자동적으로 시작된다. (**bootm** 명령어의 내부 호출에 의해서)

baudrate

콘솔 baudrate(in bps)를 결정하는 십진수. 미리 정의된 baudrate 의 목록만이 사용될 수 있다.

baudrate 를 바꿀 경우(“setenv baudrate …”를 사용해서), PPCBoot 는 콘솔 터미널의 baudrate 를 바꾸고, 새로운 속도로 들어오는 뉴라인을 기다릴 것이다. 이것은 실제로 새로운 속도로 입력하는 것이 확실하다. 만일 실패한다면, 보드를 리셋해야 한다. (“saveenv”를 할 수 없었기 때문에 예전 속도로 작동할 것이다)

“baudrate” 변수가 정의되지 않으면, default baudrate 가 사용된다.

default baudrate 는 115200 bps 이다.

bootargs

이 변수의 내용은 부트 인자로 리눅스 커널에 전달된다.

bootcmd

이 변수는 초기 카운트다운이 진행된 후에 자동적으로 실행될 명령어를 정의한다.

이 명령어는 변수 **bootdelay** 가 정의되어 있을때만 실행된다.

bootdelay

리셋후에, PPCBoot 는 **bootcmd** 변수의 내용을 실행하기 전에 이 변수에 정의된 초 동안 기다린다. 카운트다운이 출력되는 시간 동안 아무키나 누를 경우 카운트다운은 중단된다.

이 변수를 딜레이없이 0 으로 지정하자. **bootcmd** 변수의 내용에 따라서 다시는 절대 명령라인으로 들어갈 수 없을수도 있으므로 주의해야한다.

autoboot 가 불가능하게 **-1** 로 변수를 지정해보자.

bootfile

TFTP 에서 읽어들일 default 이미지의 이름

ethaddr

첫번째 이더넷 인터페이스에 대한 MAC 어드레스(=**eth0** 리눅스에서)

이 변수는 단지 한 번만 지정될 수 있다.(보통 보드 제조 과정에서) PPCBoot 는 한 번 지정된 이 변수를 지우거나 덮어쓰는 것을 막는다.

eth1addr

두번째 이더넷 인터페이스에 대한 MAC 어드레스(=**eth1** 리눅스에서)

eth2addr

세번째 이더넷 인터페이스에 대한 MAC 어드레스(=**eth2** 리눅스에서)

initrd_high

램디스크 이미지의 위치를 제한하는데 사용된다.

만일 이 변수가 지정되지 않는다면, 램디스크 이미지는 RAM 에서 가능한 가장 높은 주소로 복사될 것이다. 이것은 보통 램디스크 이미지의 최대 크기를 허용할 때 사용되어진다. 만일 다른 이유로 CFG_BOOTMAPSZ 아래에 이미지를 올려고 싶다면, 이 변수의 값을 “no” 또는 “off” 또는 “0”으로 정할 수 있다. 그렇지 않으면, 사용할 최상위 주소를 지정할 수 있다. (PPCBoot 는 여전히 PPCBoot 스택과 데이터에 덮어씌우는지 확인할 것이다.)

예를들어, 16MB RAM 을 가진 시스템으로 리눅스에서 4MB 를 사용하려고 할 경우, “bootargs” 변수의 값으로 “mem=12M”를 추가해주면 된다. 그러나 램디스크 이미지는 처음 12MB 에 위치하는 것을 기억해야한다.

```
setenv initrd_high 00c00000
```

ipaddr

IP 주소: **tftp** 명령어에 필요하다.

loadaddr

tftp 나 **loads** 같은 명령어를 위한 디폴트 로드 어드레스

loads_echo

1 로 정의된다면, 시리얼 다운로드(**loads** 명령어를 사용한)를 하는 동안 전송된 모든 문자는 출력된다. 터미널 에뮬레이션("**cu**"같은)에서 필요할 것이다. 반대일 경우는 시간이 어느 정도 걸릴 것이다.

pram

보드 환경설정에서 "Protected RAM"이 가능하다면, 이 변수는 "protected RAM"같은 제한-PPCBoot 에 의해서 RAM 이 덮어씌워지지 않는-이 가능하게 정의한다. 이 변수를 pRAM 으로 잡길 원하는만큼 kB 숫자를 정의해보자. 보드 정보에 대한 구조는 아직 RAM 의 전체 크기가 나타나는 것을 기억하자. 만일, pRAM 이 잡혀진다면, 새로운 환경 변수 "mem"은 아래와 같이 리눅스에 부트 인자를 전달하는 형태로 자동적으로 남아있는 RAM 의 전체 크기를 나타낼 것이다.

```
=> setenv bootargs $(bootargs) mem=\$(mem)
=> saveenv
```

이 방법은 리눅스에게 이 메모리를 사용하지 않으며, 재부팅에 의해서도 영향받지 않는다는 말이다.

serverip

TFTP 서버 IP 주소: **tftp** 명령어에서 필요하다.

serial#

type string 이나 시리얼 넘버 같은 하드웨어 식별 정보를 포함한다.

이 정보는 단지 한 번만(보드 제조 과정 중) 지정할 수 있다. PPCBoot 는 한 번 정의된 이 변수를 지우거나 덮어씌우지 못하게 한다.

다음의 환경 변수들은 부트 서버에서 제공되는 정보에 따라서 네트워크 부팅 명령어(**bootp**, **dhcp**, or **tftp**)에 의해 사용되어지며, 자동적으로 업데이트된다.

bootfile

위에 참조

dnsip

도메인 네임 서버의 IP 주소

gatewayip

사용할 게이트웨이(라우터) IP 주소

hostname

타겟 호스트명

netmask

서브넷 마스크

rootpath

NFS 서버에서의 루트 파일시스템의 path 명

filesize

bootp, **dhcp**, 또는 **tftp** 명령어를 사용하여 다운로드된 파일의 크기