E!Kit-1100 のご利用と開発の手引き

（2005 年 3 月 7 日 第 6 版）

２００５年3月7日

株式会社デバイスドライバーズ

ご注意

本書は、株式会社デバイスドライバーズ製組み込み Linux ボード E!Kit-1100 の利用方法に関する解説書です。他の目的には利用できません。内容の全部、一部に関わらず無断複製と無断の引用を禁じます。

- 本書で解説している E!Kit-1100 は実験、研究、教育目的に弊社が開発した評価用ボードです。開発と製造、出荷検査には細心の注意を払っていますが、運用結果につきましては、弊社ではいかなる責任も負いかねます。本製品の安全な運用に関しましては、お客様ご自身で万全を期されますようにご注意をお願い致します。
- E!Kit®と E-Kit ™は慶應義塾大学武藤佳恭氏が所有し、株式会社デバイスドライバーズが使用許可を受けている登録商標、商標です。
- その他の会社名、製品名は各社の商標または登録商標です。

1. 概要

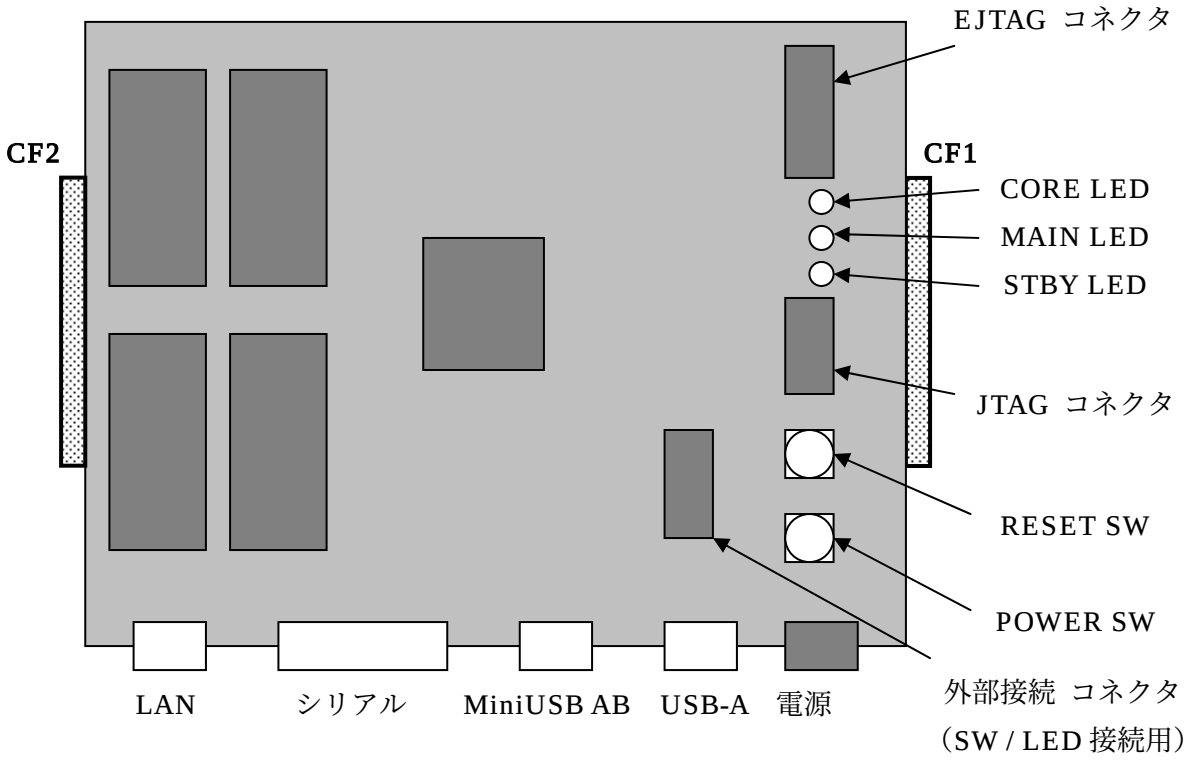
1.1. 目的

この文書は、組み込み Linux ボード E!Kit-1100 の取り扱い説明と、E!Kit-1100 を使用して各種ソフトウェア開発を行う上での必要な情報をまとめたものです。Au1100 CPU、MIPS32 アーキテクチャ、E!Kit-1100 のブート ROM で利用している YAMON モニタに関しては、それぞれのマニュアルやデータシートを参照して下さい。

1.2. 基本操作

1.2.1. 主要機能

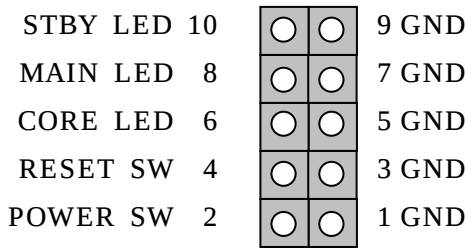
ボードの上面図の部品



- 電源入力時は電源コネクタに AC アダプタ（５Ｖ）を接続後、POWER SW を押します。
- 電源切断操作はソフトウェアからも poweroff コマンド（シェル上で実行）により可能ですが、**POWER SW を５秒間押し続けると、強制的に電源を切ります。**
- CF カードは、ボードの上面から見ると **CF カードが裏面になるように**装着します。
- シリアル・コネクタはクロス・ケーブル（ヌルモデム・ケーブル）を使用してホスト・コンピュータと接続する場合に使用します。(115200bps 8bitData 1Stopbit No Parity)
- **ご購入後最初に電源を入れられる時には、必ず 1.6 項と保証書を参照して、YAMON を起動して MAC アドレスや IP アドレス、ネットマスク等を確認、設定して下さい。**

1.2.2. **外部接続コネクタ**

E!Kit-1100 では、ボードをケースに収納して使用する場合を考慮して、電源、リセットスイッチと各 LED の信号を外部接続コネクタを介して接続できるようにしています。接続ピンの配置は以下のようになっていますので、必要に応じて外部スイッチや外付けの LED と接続して下さい。また、電源ピン(POWER)をジャンパピンでショートする事により、5V の外部電源が供給された後で直ぐに、システムの起動動作に入る事が可能です。



a. STBY LED

点灯時に外部電源が供給され、システムがスタンバイ状態にある事を示します。

b. MAIN LED

点灯時にメイン電源が ON 状態にある事を示します。

c. CORE LED

消燈時にはシステムのスリープ状態、点灯時にはランニング状態を示します。

d. RESET SW

システムをハードウェア・リセットするために使用します。

e. POWER SW

システムのメイン電源を ON/OFF 制御します。

1.3. CF **カードスロット**

1.3.1. **サポート・カード**

CF カードスロットは、標準的な CF メモリカードのほかに以下の種類のカードをサポートしています。また、**標準的な CF メモリカードであっても一部のメモリカードでは、ext2 ファイルシステムのフォーマットができないといった理由により、利用できない製品がある事が判明しています。**

a. Microdrive を始めとする、CF タイプ II 型ハードディスク・ドライブ
b. 各種モデムカード
c. 各種 LAN カード
d. 各種無線 LAN カード
※各製品の動作確認状況につきましては、順次ホームページで公開していきます。

1.3.2. ドライブの認識

CF カードにメモリカードまたは、ハードディスク・ドライブを装着した場合には、Linux で以下のように認識します。

a. メディア／ドライブが片方のスロットに 1 台だけ装着の場合には、どちらのスロットに装着しても常に/dev/hda として認識されます。
b. メディア／ドライブが両スロットに装着された場合には、CF1 スロットが/dev/hda、CF2 スロットが/dev/hdc として認識されます。
c. メディア／ドライブを片方のスロットに 1 台だけ装着して、/dev/hda としてマウントして動作している状況で、2 台目を装着した場合には、2 台目が/dev/hdc として認識されます。

1.4. シリアル・コネクタ

ブート ROM の設定、操作とコンソール端末の接続には EIA-232-E 規格（以前の RS-232C とほぼ同等、以下 RS232C）のシリアル通信(115200bps 8bitData 1Stopbit No Parity)を使用します。ホスト・コンピュータと接続する場合には、DTE 同士の接続なのでクロス・ケーブルを使用します。設定によっては、シリアル通信のパラメータを変更したり、モデム等の DCE を接続したりする事も可能です。

1.5. USB A コネクタ

USB A コネクタは、常時 USB(Ver1.1)のホスト側として動作し、各種 USB ターゲットデバイスを接続することができるコネクタです。キーボード、マウスを始め、各種ストレージ・デバイス等、Linux 標準カーネルがサポートしているほとんどのデバイスをサポートします。

※各製品の動作確認状況につきましては、順次ホームページで公開していきます。

1.6. USB ミニ AB コネクタ

USB ミニ AB コネクタは、デバイスドライバで切り替える事により、ミニ A（ホスト側）としても、ミニ B（デバイス側）としても動作させる事が可能です。USB ミニ AB コネクタには、USB Implementers Forum(http://www.usb.org/)規定のタイプミニ A、タイプミニ B のどちらのケーブルも装着可能で、デバイスドライバでどちらのケーブルが装着されたかを認識させる事が可能です。

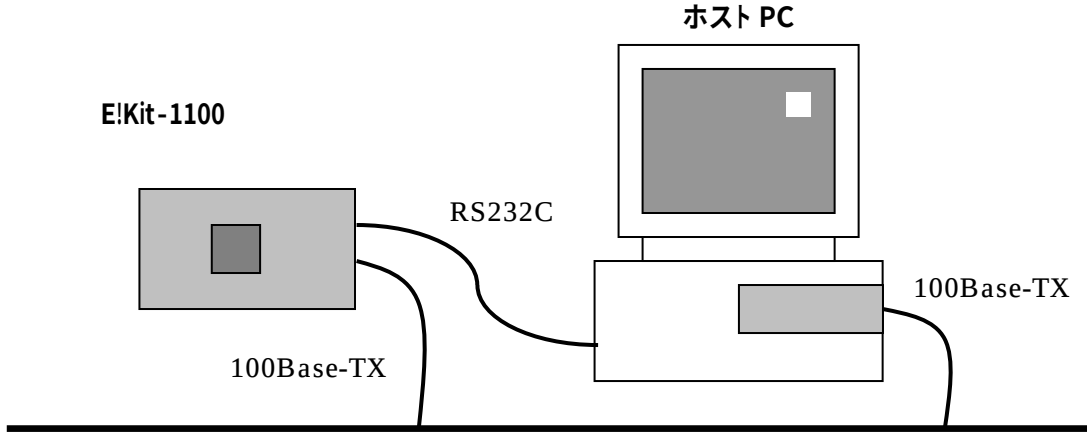
※USB ミニ AB コネクタとデバイスドライバの使用方法に関しては、第５章　補足事項をご参照下さい。

1.7. 接続環境図

1.7.1. ホスト PC

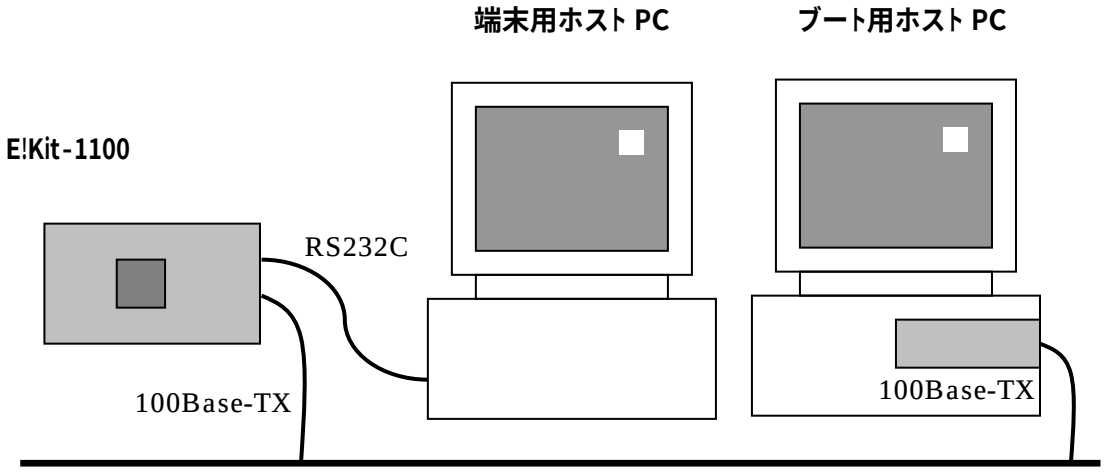
開発に使用する場合には、Red Hat 7.x, 8.0, 9.0 等の 2.4 系カーネルが動作している x86 アーキテクチャで、シリアルポートと 100Base-TX コネクタを持つ Linux PC を開発ホストとして用意して、同じセグメントの LAN と RS232C クロス・ケーブルで接続して下さい。以降は、Red Hat 8.0/9.0 を推奨環境として説明します。

ホスト PC では minicom 等の端末エミュレーション・ソフトを起動して接続します。設定は 115200bps、８ビットデータ、１ストップビット、ノンパリティ、ハードウェアフロー制御有りです。



100BaseT Ethernet 回線

また、以下のようにホスト PC をブート用と端末用に 2 台用意した場合には、ブート ROM の設定、操作とコンソール端末用に、ハイパーターミナル等の端末エミュレーション・ソフト Windows マシンを使用する事も可能です。



100BaseT Ethernet 回線

1.7.2. IP アドレス

以降の説明では、以下のように E!Kit-1100 の IP アドレスを固定して使用する事を想定して説明します。

システム	IP アドレス	ネットマスク	備考
E!Kit-1100	192.168.1.150	255.255.255.0	CF ルートファイルシステム用 Tiny システム
E!Kit-1100	192.168.1.150	255.255.255.0	NFS ルートファイルシステム用 Large システム
ホスト PC	192.168.1.201	255.255.255.0	TFTP サーバ、DHCP サーバ、NFS サーバ、NTP サーバ

これらの E!Kit-1100 で使用する IP アドレスは、変更して利用することも可能ですが、無用なトラブルを避けるためにも、システム立ち上げ時には一度、上記と同じ IP アドレス、ネットワーク・アドレスの環境を構築して通信動作の確認をされる事を、推奨致します。

E!Kit-1100 の IP アドレスは以下のファイルを編集して設定します。

ファイル名：**/etc/sysconfig/network-scripts/ifcfg-eth0**

以下に IP アドレス設定をする ifcfg-eth0 ファイルの設定内容例を示します。

```
DEVICE=eth0
ONBOOT=yes
BOOTPROTO=static
IPADDR=192.168.1.150
NETMASK=255.255.255.0
```

1.7.3. ユーザ名

初期設定では、以下のユーザ名を用意し、事前に設定しています。

ユーザ名	User ID	Group ID	パスワード
root	0	0	なし
aurum	1100	1100	なし
aurum1	1101	1100	aurum1

初期設定では、root アカウントにパスワードを設定していませんので、セキュリティ上の問題がある場合には、passwd コマンド等でパスワードを設定して下さい。

1.7.4. サービス

E!Kit-1100 では、ルートファイルシステムを小容量 CF メモリ用の Tiny システムと、NFS / Microdrive 等の大容量 CF メモリカード用の Large システムで、以下のサービスを用意していますので、RS232C での接続以外にこれらのプロトコルを使って LAN 経由で接続し、操作する事が可能です。

サービス名	アプリケーション	Tiny システム	Large システム
telnet	Busybox-1.00pre3	○	―
ftp	proftpd-1.2.8	○	○
ssh	openssh-3.7.1p2	○	○
httpd	apache-1.3.27	○	○

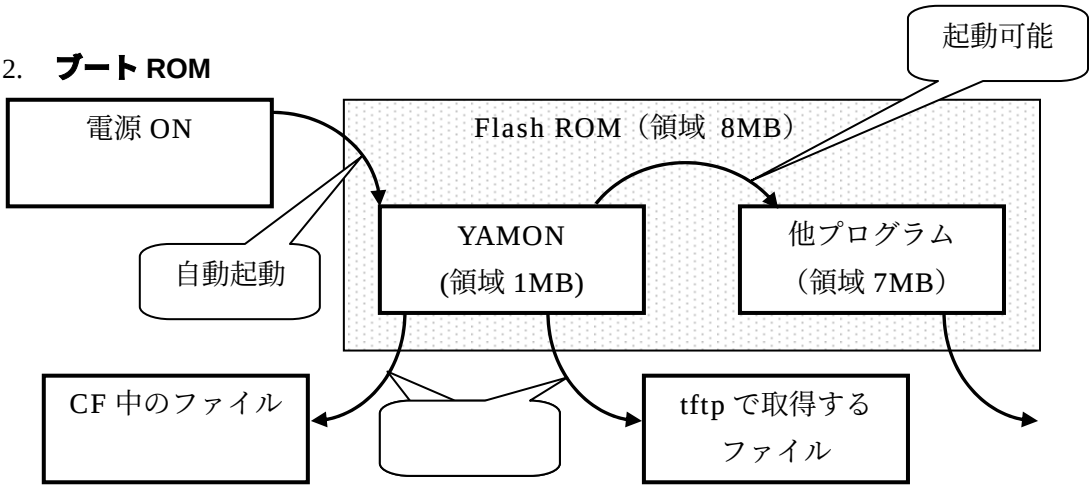
1.8. 起動設定

1.8.1. ブートローダ

E!Kit-1100 ではシステムの起動をサポートするために、オンボード Flash ROM に、YAMON というブートローダを搭載しています。YAMON は MIPS テクノロジー社から無償で公開されている ROM モニタ・プログラムで、システムの起動のほか、外部プログラムのロード、実行と、簡単なデバッグ機能を備えます。YAMON が起動時に使用する各種パラメータ情報は、オンボード Flash ROM 内の YAMON 管理領域に保持するために、設定内容は電源を切っても維持されます。YAMON については、次の MIPS テクノロジー社のサイトに詳しい情報が載っています。

http://www.mips.jp/02products/DevSystem/yamon.html

1.8.2. ブート ROM



この図に示すように、YAMON は電源 ON 時に自動的に実行されるプログラムです。弊社では、YAMON のプログラムを E!Kit-1100 に移植する時に機能拡張して、CF スロット

に装着した CF メモリカード中の指定された S レコード形式のファイルを読み込んで起動できるようにしています。また Flash ROM 中には、他プログラム用に 7MB の空き領域が確保してありますので、この領域に小さなシステムや他のブートロードを組み込んで利用する事が可能です。

この弊社の機能拡張と、YAMON 自身が備える機能により、E!Kit-1100 では様々なプログラムの様々な起動動作が可能で、Linux 以外のオペレーティングシステムの動作も可能にしています。また YAMON を使用して、YAMON 自身を書き換えたり、他のブートローダに更新したりすることも可能です。

1.8.3. YAMON の設定

RS232C ケーブルで E!Kit-1100 とホスト PC を接続して E!Kit-1100 を起動すると、端末エミュレーション・ソフト上に次のようなメッセージが出力されます。この**メッセージが表示されている２秒間の間に、Control-C を入力する(Control キーを押しながら c キーを押す)**事で、ブート環境等を設定変更できる、YAMON モニタに入ることが可能です。

```
YAMON ROM Monitor, Revision 02.19AURUM-120103.
Copyright (c) 1999-2000 MIPS Technologies, Inc. - All Rights Reserved.

For a list of available commands, type 'help'.

Compilation time =          Dec  1 2003  16:18:08
MAC address =               00.0e.6c.00.00.09
Processor Company ID =      0x03
Processor ID/revision =     0x02 / 0x04
Endianness =                Little
CPU =                       396 MHz
Flash memory size =         4 MByte
SDRAM size =                128 MByte
First free SDRAM address =  0x80090f0c

Environment variable 'start' exists. After 2 seconds
it will be interpreted as a YAMON command and executed.
Press Ctrl-C to bypass this.
```

YAMON モニタで使用できる主なコマンドは以下の通りです。

コマンド	内容
help	ヘルプメッセージの表示
setenv	環境変数の設定、または表示
unsetenv	ユーザ定義環境変数の削除
load	TFTP サーバ等からのプログラムのロード
go	ロードしたプログラムの実行
copy	メモリ間のデータ・コピー（Flash ROM への焼きこみも可能です）
erase	Flash ROM 領域のデータ削除（焼きこみ前に必要です）

現在の YAMON の環境変数を表示させるためには、setenv コマンドを使用します。CD-ROM 中に含まれる 64MB CF 用の Tiny システムを使用して、CF メモリカードに Linux カーネルとルートファイルシステムを置いて、ブートする場合の環境変数の設定例を以下に示します。

```
MAC         (R/W)  0
bootfile    (R/W)  boot/ekit1100.srec
bootprot    (R/W)  file
bootserport (R/W)  tty1
bootserver  (R/W)  192.168.1.201
ethaddr     (R/W)  00.0e.6c.00.00.09
gateway     (R/W)  0.0.0.0
ipaddr      (R/W)  192.168.1.150
memsize     (RO)   0x08000000
modetty0    (R/W)  115200,n,8,1,none
modetty1    (R/W)  115200,n,8,1,none
prompt      (R/W)  YAMON
start       (R/W)  load //hda1; go . root=/dev/hda1
subnetmask  (R/W)  255.255.255.0
```

環境変数を変更する場合には、以下の例のように setenv コマンドを、パラメータ付きで実行します。

```
YAMON> setenv bootfile vmlinux.srec
```

start 環境変数は、電源 ON 時に自動実行をするための、特別な環境変数です。**start** 環境変数がセットされていない場合には、起動後、**YAMON>** のプロンプトを表示して停止します。

bootprot 環境変数は YAMON 標準の tftp, asc のほかに、**file** オプションが選択可能です。**file** オプションでは、E!Kit-1100 の独自拡張機能で、CF スロットにマウントしたデバイス中にある、指定された S レコード形式のファイルを読み込んで起動します。

load コマンドでは **bootprot** が **file** の場合に、E!Kit-1100 の独自拡張機能として、CF スロット番号(hda, hdb)と 4 つのプライマリパーティションの何れか（1～4）を指定して、読み込むファイルのパーティションを指定する必要があります。

setenv コマンドで、新たにユーザ定義環境変数を設定して使用して利用する事も可能です。万が一、setenv コマンドで誤ったユーザ定義環境変数を設定してしまった場合には、unsetenv コマンドでユーザ定義環境変数を削除することが可能です。

1.9. MAC アドレスの設定

本製品では出荷前に動作試験を行ってから出荷していますが、お客様が本製品を始めてお使いになる際には、Ethernet MAC アドレスが正しく設定されていない場合があります。そのためにお手数ですが、ご購入後最初に電源を入れる時には、必ず保証書を参照して、MAC アドレス（Ethernet 物理アドレス）を以下の手順で設定するか、または設定内容を確認して下さい。MAC アドレスは一度設定すれば本体 Flash ROM に記録されますので、電源を切っても再設定の必要はありません。

a. 電源 ON

本体に LAN ケーブルを接続せずに、シリアル・クロス・ケーブルで端末エミュレーション・ソフトを起動してあるホスト PC に接続した後で、起動して下さい。

b. YAMON の呼び出し

ホスト PC の端末エミュレーション・ソフト上に「**Press Ctrl-C to bypass this.**」のメッセージが現れるので、直ぐに「コントロール・キー」と「**C** キー」を同時に押して下さい。

c. MAC アドレスの設定

YAMON> のプロンプトが表示されますので、各製品に割り当てられている、保証書の「シリアル番号（Mac アドレス）」の欄に載っている 48 ビット分の 16 進数表示の値（Mac アドレス）を設定します。

設定例）

```
YAMON> setenv ethaddr 00.0e.6c.00.00.09
```

※この設定例は Mac アドレスを **00.0e.6c.00.00.09** に設定する場合のものです。実際にアドレスを入力する場合には各製品に割り当てられている値を設定して下さい。設定終了後は一旦電源を切るか、リセット・スイッチを押すことで、新しい Mac アドレスで再起動します。

※保証書には、**00-0e-6c-00-00-09** と、各 **16** 進数を「**-**」で区切って表示していますが、設定の際には「**.**」で区切って入力して下さい。

1.10. Linux の立ち上げ

1.10.1. サポート・システム

E!Kit-1100 では使用目的に合わせて、システムの起動に必要な Linux カーネルと、運用に必要なルートファイルシステムを以下の場所に配置して、利用することができます。

a. オンボード Flash ROM
b. CF カードスロットに装着したメモリカード（Microdrive を含みます）
c. USB コネクタに接続した外部ストレージ（USB Flash メモリ、USB 外部ディスク）
d. 他のホストがサービスする NFS 上のファイル

これらのファイル配置方法の典型的な組み合わせ例を以下に示します。Kernel はブートに必要な initrd とスタティックにリンクした Linux カーネルを示し、Root FS は Linux の運用に必要な全てのドライバ、コマンド、環境等を示します。

名称・サポート	配置場所	容量	主な用途
Flash システム（*未サポート）	Kernel, Root FS ともに Flash ROM に配置	YAMON / Root FS 併せて 8MB 以内	CF 2 スロットを利用するシステム
Micro システム（*未サポート）	Kernel=Flash ROM Root FS=CF、または USB 外部記憶装置	Flash ROM 部 8MB 以内 Root FS は外部記憶容量に依存	YAMON がサポートしていないデバイスからのブート
Tiny システム	Kernel, Root FS ともに CF に配置	約 64MB	組み込み用途向け標準システム
Small システム（開発中）	Kernel, Root FS ともに CF に配置	約 256MB	最小のセルフコンパイル環境
Large システム (CF)	Kernel, Root FS ともに CF に配置	約 640MB	スタンドアロン、セルフコンパイル環境
Large システム（NFS）	Kernel, Root FS ともに NFS に配置	約 640MB	開発用セルフコンパイル環境

* Flash システムと Micro システムは弊社で動作確認していますが、これらのルートファイルシステムは弊社からは提供しませんので、「未サポート」扱いとさせて頂いています。また、Flash ROM の書き換え作業は、**メモリアドレスの書き間違いや、Flash メモリの内容破壊によるトラブルの原因になる可能性がありますので、推奨致しません（万が一、Flash メモリの内容破壊により起動できなくなった場合の Flash メモリの書き換え作業は実費、有償で承ります）。**実施される場合は、お客様自身の技量に応じて、慎重に作業を進められるようにお願い致します。

弊社から提供する「Tiny システム」、「Large システム」等の各ルートファイルシステムに含まれる内容、容量はあくまでも弊社が標準的な運用を想定して、ご提供するものです。利用目的に合わせて、含まれる内容は自由に追加、削除、改変してご利用頂く事が可能です。

1.10.2. Tiny システム

Tiny システムは添付 CD-ROM の中の tiny/rootfs-aml3-tiny.tar.bz2 として入っています。（以前のバージョンでは cf-p4b1.tgz という名前でした）最新版の Tiny システムは、http://e-kit.jp/products/E!Kit1100/ のホームページからダウンロード可能です。

Tiny システムを CF メモリカードに配置して利用するためには、ルートファイルシステム全体（ファイルとディレクトリを全て含んだもの、以下単にシステムと呼ぶ場合があります）を CF メモリカードに転送する必要があります。CF メモリカードにシステムを転送する方法としては、次のような方法があります。

a. E!Kit-1100 を使用する方法

NFS を利用して Large システムで起動する E!Kit-1100 を使用して、CF メモリカードにシステムを転送する方法です。すでに Large システムで E!Kit-1100 が起動できるような環境が構築されている必要があります。

b. ホスト PC で USB 接続の CF カードリーダを使用する方法

Linux が動作しているホスト PC に USB CF カードリーダを接続して、そのカードリーダに CF メモリカードをマウントして、システムを転送する方法です。Linux で動作確認ができている、USB CF カードリーダが必要です。

c. CF スロットや PC カードスロットを持つノート型 PC を使用する方法

Linux が動作しているノート型のホスト PC に内蔵されている、CF スロットや PC カードスロットに CF メモリカードを装着して（PC カード用 CF アダプタ経由でも可）、システムを転送する方法です。Linux で PC カードスロットの動作確認ができている、ノート型 PC が必要です。

次の操作例は添付 CD-ROM をホスト PC の CD-ROM ドライブにマウントして、E!Kit-1100 を使用して、CF メモリカードに Tiny システムを転送する方法を示します。そのためには、まずホスト PC と Large システムを立ち上げ（「1.10.3 Large システム」を参照）、CF 全体を ext2 ファイルシステムでフォーマットし（「5.2 CF のフォーマット手順」

を参照)、Tiny システムを CF に書き込みます。

ホスト PC での操作（CD-ROM ファイルの NFS エクスポート）：

```
# mount -t iso9660 -o ro /dev/cdrom /mnt/cdrom
# exportfs -o ro 192.168.1.150:/mnt/cdrom
```

E!Kit-1100 Large システムでの操作（CF へのファイルシステム内容書き込み）：

```
# mount -t nfs 192.168.1.201:/mnt/cdrom /mnt/cdrom
# mount -t ext2 /dev/hda1 /mnt/disk
# cd /mnt/disk
# tar xvjf /mnt/cdrom/tiny/rootfs-aml3-tiny.tar.bz2
# cd /
# sync; sync; sync;
# umount /mnt/cdrom
# umount /mnt/disk
```

ホスト PC での操作（CDROM ファイルの NFS アンエクスポート）：

```
# exportfs -u 192.168.1.150:/cdrom
# umount /cdrom
```

Tiny システムを利用するためには、YAMON モニタ環境変数を次のように設定して CF の中の 2.4.26-ek1.srec（モトローラ S レコード形式で保存された Linux カーネル） から Linux カーネルが起動するようにします。この設定により、次回起動以降 E!Kit-1100 では Tiny システムが立ち上がります。

```
YAMON> setenv bootfile 2.4.26-ek1.srec
YAMON> setenv bootprot file
YAMON> setenv start “ load //hda1; go . root=/dev/hda1 ”
```

「2.4.26-ek1.srec」は起動する S レコード形式の Linux カーネルイメージのファイル名です。以前は ekit1100.srec という名前のファイルでしたが、今後のリリースではカーネルのバージョン名がわかる名前になります。必要に応じてファイルシステムに搭載している Linux カーネルイメージ・ファイル名に合わせて変更して下さい。

このように start 環境変数には、自動実行する複数の YAMON コマンドを順に「;」（セミコロン）で区切って登録することが可能です。YAMON モニタ環境変数に設定したい値が 1 行に収まらない場合には、「¥」(バックスラッシュ)を入力することによって、入力行を次の

行に継続することができます。

1.10.3. **Large システム**

Large システムは、添付 CD-ROM の中の /large/rootfs-aml3.tar.bz2 （以前のバージョンでは rootfs-aml2.tgz という名前でした）に入っています。

Large システムは、NFS で E!Kit-1100 にマウントして運用されることを想定して、標準設定としてホスト PC から提供される TFTP, DHCP, NFS, NTP のサービスを利用するようになっています。そのため、Large システムを利用するためには、まずホスト PC がこれらのサービスを提供するように設定する必要があります（「3. セルフ開発環境」を参照)。

ネットワークから起動する Large システムを利用するためには、YAMON モニタ環境変数を次のように設定します。これで次回以降、E!Kit-1100 を起動すると Large システムが立ち上がります。

```
YAMON> setenv bootfile 2.4.26-ek1.srec
YAMON> setenv bootprot tftp
YAMON> setenv start “ load; go . nfsroot=192.168.1.201:/home/rootfs/rootfs ”
```

「2.4.26-ek1.srec」のカーネル名（以前は ekit1100.srec）は、ファイルシステムに搭載している Linux カーネルイメージ・ファイル名に合わせて変更する必要があるのは、Tiny システムの場合と同様です。

E!Kit-1100 のカーネルは、デフォルトのルートデバイスが「NFS」として、NFS でマウントするルートファイルシステムの所在が、192.168.1.201:/home/rootfs/rootfs としてコンパイルしてあります。そのため、NFS からブートする場合には、「root=」のカーネルパラメータは必要ありません。また、NFS でマウントするルートファイルシステムの所在が、192.168.1.201:/home/rootfs/rootfs の場合には、「**nfsroot=」**のカーネルパラメータは不要です。

rootfs-aml3.tar.bz2 以降の Large システムは、そのまま Microdrive 等の大容量 CF メモリカードに転送して、起動することも可能です。その場合には、次のように YAMON モニタ環境変数を設定します。

```
YAMON> setenv bootfile /boot/2.4.26-ek1.srec
YAMON> setenv bootprot file
YAMON> setenv start “ load //hda1; go . root=/dev/hda1 ”
```

※Large システムで rpm コマンドを利用するためには、"rpm --rebuilddb"を一度実行する必要があります。

2. クロス開発環境

2.1. 概要

最初に E!Kit-1100 用のカーネル、デバイスドライバ、アプリケーションを開発するための、クロス開発環境をインストールしていきます。E!Kit-1100 用のクロス開発環境構築としては、ホスト・コンピュータとして Linux2.4 系オペレーティング・システムが動作している、x86 アーキテクチャの標準的な PC を使用します。ターゲット・コンピュータは、MIPS32 アーキテクチャを Little Endian で使用する E!Kit-1100 になります。

ホスト PC に E!Kit-1100 用のクロス開発環境を構築するには、以下のものを用意します（添付 CD-ROM に収録）。通常は、構築済みのクロス開発環境 export.tgz を /export 以下に解凍するだけで、次項以降で説明するインストール手順は必要ありません。しかしながら、これらの開発環境をカスタマイズする場合や、構成するコンパイラやライブラリ、カーネルのバージョンを変更する時にはこのような手順が必要になります。ここで紹介するのは、各ツール、ライブラリとも、現在動作確認しているバージョンです。他のバージョンを使用した場合、正しく動作するかどうかは、確認していません。

a. 構築済みクロス開発環境関連ファイル

名前・内容	入手先
linux-2.4.26-ek1.tar.bz2 カーネルソース	デバイスドライバーズ http://e-kit.jp/products/E!Kit1100/
export.tgz クロス開発環境ファイル一式	デバイスドライバーズ http://e-kit.jp/products/E!Kit1100/
busybox.tgz busybox ソース	デバイスドライバーズ http://e-kit.jp/products/E!Kit1100/
rootfs-aml3.tar.bz2 NFS サーバ用 mips-el Root FS	デバイスドライバーズ http://e-kit.jp/products/E!Kit1100/ 参考：（上記の元） ftp://ftp.mips.com/pub/linux/mips/rootfs_nfs/

通常、クロス開発環境をインストールする場合には、以下の手順で export.tgz のファイルを展開するだけでインストールが完了します。この例では、/export をクロス開発環境のインストール先として、/export ディレクトリ以下を使用していますが、システムに存在しない場合には、最低 1GB 程度を目安に確保して下さい。

```
# mount -t iso9660 -o ro /dev/cdrom /cdrom
```

```
# cd /export
# tar xvzf /cdrom/tar/export.tgz
```

また開発に必要な環境変数は、mipsel-env.sh というシェルスクリプトに記述して使用しますので、クロスコンパイル作業を行う際には、このファイルを事前に読み込んで下さい。開発用シェルで読み込む例:

```
# . /cdrom/tar/mipsel-env.sh
```

以下に、mipsel-env.sh の内容を載せます。クロス開発環境用ディレクトリの構造を変更する場合には、必要に応じて修正して下さい。

```
TARGET=mipsel-linux
PREFIX=/export/local
KERNELSRC=/export/src/linux
TOOLROOT=/export/tool
LANG=en_US
PATH=/export/local/bin:$PATH
export TARGET PREFIX KERNELSRC TOOLROOT PATH LANG
```

b. クロス開発環境構築に必要なファイル（自分で環境を構築する場合）

名前・内容	入手先
linux-2.4.26-ek1.tar.bz2 カーネルソース	デバイスドライバーズ http://e-kit.jp/products/E!Kit1100/
binutils-2.13.2.1.tar.bz2 binutils	GNU プロジェクト ftp://ftp.ring.gr.jp/pub/GNU/binutils/
gcc-3.2.3.tar.bz2 gcc コンパイラ	GNU プロジェクト ftp://ftp.ring.gr.jp/pub/GNU/gcc/gcc-3.2.3/
glibc-2.3.2.tar.bz2 glibc-linuxthreads-2.3.2.tar.bz2 glibc	GNU プロジェクト ftp://ftp.ring.gr.jp/pub/GNU/glibc/
busybox-1.00.tar.gz BusyBox	Erik Andersen http://busybox.net/downloads/
rootfs-aml3.tar.bz2 ルートファイルシステム （Large システム）	デバイスドライバーズ http://www.devdrv.co.jp/download/ 参考：（上記の元） ftp://ftp.mips.com/pub/linux/mips/rootfs_nfs/
mipsel-env.sh 環境変数設定用シェルスクリプト	デバイスドライバーズ http://e-kit.jp/products/E!Kit1100/
mk_ramdisk.sh initrd 作成用シェルスクリプト	デバイスドライバーズ http://e-kit.jp/products/E!Kit1100/
glibc-fix.sh glibc 修正用シェルスクリプト	デバイスドライバーズ http://e-kit.jp/products/E!Kit1100/
glibc-2.3.2-cross.patch glibc-2.3.2-mips.patch glibc-2.3.2-without-fp.patch MIPS クロス開発用 glibc パッチ	デバイスドライバーズ http://e-kit.jp/products/E!Kit1100/ 参考：（上記の元） http://kegel.com/crosstool/current/patches/ ほか

※これらのファイルは添付 CD-ROM に収録されています。

2.2. クロス開発環境構築

2.2.1. インストール手順

これから示すのは、E!Kit-1100 用のクロス開発環境をホスト PC の /export 以下にインストールする場合の手順です。必要な tar ファイルがすでに /export/tar ディレクトリ以下にコピーしてあるという前提で作業を進めて行きます。別なディレクトリにインストールする場合には、適宜ディレクトリ名を変更して下さい。作業に当たって同じディレクトリ名や定義名を何回も使用するので、必要な環境変数を設定しておきます。またインストールしたクロスコンパイラを実行させるために、クロスコンパイラのインストール先へのパスを通しておく必要もあります。

2.2.2. ディレクトリの確保

まず、/export ディレクトリを作成してその下に必要なサブディレクトリを作成し、必要なファイルをコピーしておきます。今後カーネルを何回かコンパイルし直す事を考慮すると、/export 以下には、数百 MB から 1GB 程度の容量が必要になりますので、/ パーティションで容量を確保できない場合には、シンボリックリンクを使用して他のパーティションで必要な容量を確保します。

以下は /home 以下に実体となるディレクトリを確保する場合の操作例です。

```
# mkdir /home/export
# cd /
# ln -s home/export .
```

もし、/ パーティションに十分な余裕がある場合には、直接、

```
# mkdir /export
```

として構いません。以下、必要なサブディレクトリを作成します。

```
# cd /export
# mkdir local src tool
```

環境変数と、パスの設定をします（mipsel-env.sh を読み込む）。LANG 変数を「en_US」にしておくのは、標準の日本語環境のまま（**LANG=ja_JP.eucJP**）では、コンパイラやツールのバージョンによっては日本語メッセージを出力して、その結果、思わぬトラブルを招く場合があるからです。以降のインストール手順で、ここで設定する環境変数をインストール先の指定に使用しますので、必ず設定して下さい。インストール先を間違ってしまうと、ホスト PC の動作に必要なファイルを上書きしてしまい、ホスト PC の動作に支障をき

たす場合があります。

```
# cd /export
# . tar/mipsel-env.sh
```

2.2.3. binutils の展開とインストール

binutils はクロス開発に必要なアセンブラと、リンカなどの基本的な言語開発ツールをまとめたものです。展開後に生成される config.in ファイルを見ればわかる通り、世の中のほとんどとも思われる膨大な種類のプロセッサとオペレーティングシステムに対応しています。binutils を展開した後、インストールを実行します。

```
# cd $TOOLROOT
# tar xvjf ../tar/binutils-2.13.2.1.tar.bz2
# mkdir build-binutils
# cd build-binutils
# ../ binutils-2.13.2.1/configure --target=$TARGET --prefix=$PREFIX
# make
# make install
```

2.2.4. カーネルヘッダファイルの準備

以降の手順で参照するカーネルヘッダファイルを以下の手順で準備します。まず、カーネルソースを展開して、E!Kit-1100 用のカーネル設定ファイルを読み込み、カーネルを設定します。これにより、E!Kit-1100(MIPS アーキテクチャ)用のヘッダファイルとシンボリックリンクが作成されます。次に、glibc 用の include ディレクトリを作成してシンボリックリンクを作成します。

```
# cd /export/src
# tar xvjf ../tar/linux-2.4.26-ek1.tar.bz2
# ln -s linux-2.4.26-ek1 linux
# cd linux
# make mrproper
# cp arch/mips/defconfig-ekit1100 .config
# make oldconfig
# make dep
# cd $PREFIX/$TARGET
# mkdir include
```

```
# cd include
# ln -s $KERNELSRC/include/linux .
# ln -s $KERNELSRC/include/asm .
# ln -s $KERNELSRC/include/asm-generic .
```

2.2.5. gcc の構築その１

まず C ライブラリを構築するために、gcc, cpp といった C クロスコンパイラ本体をコンパイル、インストールします。

```
# cd $TOOLROOT
# tar xvjf ../tar/gcc-3.2.3.tar.bz2
# mkdir build-boot-gcc
# cd build-boot-gcc
# ../gcc-3.2.3/configure --target=$TARGET --prefix=$PREFIX --without-headers  \
  --with-newlib --enable-languages=c --disable-threads --disable-shared
# make all-gcc
# make install-gcc
```

2.2.6. glibc の構築

アプリケーション開発時にクロスコンパイラが使用する C ライブラリを、前項で作成したクロスコンパイラを使用してソースコードからコンパイルし、インストールします。

```
# cd $TOOLROOT
# tar xvjf ../tar/glibc-2.3.2.tar.bz2
# cd glibc-2.3.2
# tar xvjf ../../tar/glibc-linuxthreads-2.3.2.tar.bz2
# patch -p1 < ../../tar/glibc-2.3.2-cross.patch
# patch -p1 < ../../tar/glibc-2.3.2-mips.patch
# patch -p1 < ../../tar/glibc-2.3.2-without-fp.patch
# cd ..
# mkdir build-glibc
# cd build-glibc
# CC=${TARGET}-gcc ../glibc-2.3.2/configure --host=$TARGET --prefix=/usr  \
  --enable-add-ons --with-headers=$PREFIX/$TARGET/include
# make
# make install_root=$PREFIX/$TARGET prefix=”” install
```

2.2.7. glibc 構築の確認

glibc が正しくインストールされたことを確認します。

```
# cd $PREFIX/$TARGET/usr/include
# find . -print | cpio -pumd $PREFIX/$TARGET/include
3457 blocks などと表示されます

# cd $PREFIX/$TARGET/usr/lib
# find . -print | cpio -pumd $PREFIX/$TARGET/lib
210892 blocks などと表示されます
```

2.2.8. リンカスクリプトの修正

glibc-fix.sh を実行して、リンカスクリプト libc.so と libpthread.so 内に誤った絶対パスの参照が入ってしまう問題を修正します。

```
# cat /export/tar/glibc-fix.sh

（以下は cat コマンドによるシェルスクリプトの表示結果です）
#! /bin/sh
SYSROOT=/export/local/mipsel-linux
for file in libc.so libpthread.so; do
  mv $SYSROOT/lib/$file $SYSROOT/lib/${file}.orig
  sed 's,/lib/,,g' < $SYSROOT/lib/${file}.orig > $SYSROOT/lib/$file
done

（シェルスクリプトを実行します）
# sh /export/tar/glibc-fix.sh
```

2.2.9. gcc の構築その２

クロスコンパイル用 C ライブラリのインストールができたところで、そのライブラリを使用して再度クロスコンパイラをコンパイルし、インストールします。

```
# cd $TOOLROOT
# mkdir build-gcc
# cd build-gcc
# ../gcc-3.2.3/configure --target=$TARGET --prefix=$PREFIX  \
  --enable-languages=c,c++,f77,objc
# make all
# make install
```

ここまでの手順で、カーネルとアプリケーションをクロスコンパイルするためのクロス開発環境のインストールが終りました。

2.3. カーネルのクロスコンパイル

2.3.1. 概略

プレインストール版のカーネルの設定をコピーして、オリジナルの E!Kit-1100 用カーネルをソースコードから、カーネルの名前を変えて、クロスコンパイル／リンクする手順を紹介します。

2.3.2. コンパイル手順

この例では、元となるカーネルのバージョンを 2.4.26-ek1 として、新しく作るカーネルのバージョンを 2.4.26-ek1test とした場合の例を示します。

```
# . /export/tar/mipsel-env.sh
# cd /export/src
# mv linux-2.4.26-ek1 linux-2.4.26-ek1test
```

元のカーネルソースを参照できるように戻しておきます。

```
# tar xvjf ../tar/linux-2.4.26-ek1.tar.bz2
```

コンパイル前に Makefile で定義してあるバージョン番号を変更します。

```
# cd linux-2.4.26-ek1test
# vi Makefile
（ファイル先頭部分に定義されている EXTRAVERSION 変数の値を’-ek1test’とします。）
```

```
# make mrproper （カーネルソースを初期状態にします）
# cp arch/mips/defconfig-ekit1100 .config （E!Kit-1100 用設定ファイルを標準設定ファイルにコピーします）
# make oldconfig （標準設定ファイル .config の設定内容をそのまま使用する場合）
または
# make menuconfig (必要なカーネル・コンパイル・オプションを設定変更する場合)
```

ローダブル・モジュールを先にコンパイルして、カーネルに組み込む initrd RAMDISK イメージを作成します。

```
# make dep
# make modules
# sh ../../tar/mk_ramdisk.sh
```

カーネルをコンパイルして、RAMDISK イメージとともにリンクします。

```
# make zImage
```

2.3.3. カーネルとモジュールの転送

ネットワークブート用またはフラッシュ ROM に焼き込むカーネルの場合には、TFTP 用ディレクトリに転送しておきます。この場合の名前は、YAMON の環境変数で設定してある名前(**2.4.26-ek1**.srec)と一致させる必要があります。

```
# cp -p arch/mips/zboot/images/ekit1100.srec /tftpboot/2.4.26-ek1.srec
```

ローダブル・モジュールは以下のように lib ディレクトリごと、ネットワークブートまたはフラッシュ ROM からの起動で使用するルートファイルシステムのメディアに別途インストールする必要があります。

例：NFS でエクスポートするルートファイルシステムが /home/rootfs/rootfs の場合

```
# make INSTALL_MOD_PATH=/home/rootfs/rootfs modules_install
```

カーネルを CF から起動する場合、カーネルを CF に転送するには、CF をマウントしてローダブル・モジュールとともに CF に書き込みます。

例：フォーマット済みの CF が（ホスト PC の）/mnt/cf にマウントされている場合

```
# cp -p arch/mips/zboot/images/ekit1100.srec /mnt/cf/2.4.26-ek1.srec
# make INSTALL_MOD_PATH=/mnt/cf modules_install
```

（カーネルを CF から起動するには、CF 内にルートファイルシステムが存在し、また、YAMON モニタの環境変数が適切に設定されている必要があります。）

※ご注意

これらのコマンドを実行すると「not for this architecture」という警告メッセージが表示されますが、これはアーキテクチャの違うマシンでのクロス開発のために depmod が実行できないためですので無視します。

3. セルフ開発環境

3.1. 概要

セルフ開発環境での開発作業は、PC Linux 環境での操作とほとんど同じですので、クロス開発環境よりずっと簡単で使い易くなります。E!Kit-1100 用の環境の場合、PC と違ってハードウェアの構成が現在１種類しかないので、必要なファイルシステムを展開して、ネットワークとサービスの設定を実行するだけです。以下にその手順を説明します。

3.2. NFS サーバを利用したセルフ開発

3.2.1. 基本方針

E!Kit-1100 においてセルフ開発用に、必要なアプリケーション実行環境を全部インストールしたフルサポート・システムを構築するためには、NFS サーバを利用する方法が現実的です。CF スロットや USB に、大容量の Microdrive や外部ストレージを接続して、セルフ開発に必要なディスク容量を確保することも可能ですが、ここでは比較的安価なホスト PC 用の大容量ハードディスク・ドライブをネットワーク経由で利用する方法を紹介します。

ここではホスト PC（NFS ホスト）上に開発するプログラム・ソースコードを配置し、セルフコンパイルを行う E!Kit-1100 から、同一のソースコードを NFS で同期して参照する環境を構築します。そのため X Window システムや Emacs エディタ等の開発に使用するリソースを消費するアプリケーションを NFS ホスト側で実行して、ソースコードの編集、デバッグを行い、オブジェクトモジュール作成のためのセルフコンパイルといった、必要最低限の作業を E!Kit-1100 で実行させて、ターゲット環境（E!Kit-1100）でのリソースの消費を軽減し、足りない性能を補う事が可能です。

設定ファイルは、ホスト PC の /home/rootfs/rootfs というディレクトリを E!Kit-1100 用の NFS エクスポート・ファイルシステムとしています。実際には、ひとつ上のディレクトリである、 /home/rootfs をエクスポートしておいて、その下に開発バージョンに応じた名前のサブディレクトリを作成しておき、/home/rootfs/rootfs は、各開発用サブディレクトリへのシンボリックリンクとする事で、複数のルートファイルシステムを簡単に切り替えることができます。以下にこのようなエクスポート環境を構築する操作例を示します。

```
# cd /home/rootfs
# mkdir rootfs-aml3
# cd rootfs-aml3
# tar xvjf /export/tar/rootfs-aml3.tar.bz2
# cd ..
# ln -s rootfs-aml3 rootfs
```

3.2.2. ネットワークとサービスの設定

Linux 環境のホスト PC を用意して、E!Kit-1100 でのセルフ開発に必要な、以下のサービスを立ち上げます。

- TFTP サーバ
- DHCP サーバ
- NFS サーバ
- NTP サーバ

それぞれの設定ファイルの内容例を以下に示します。

ここでの設定内容は、説明のために subnet 192.168.1.0 netmask 255.255.255.0 の環境で、前記の /home/rootfs を NFS でエクスポートする事を想定した内容になっています。

別の IP アドレスのネットワーク環境でサーバを立ち上げて、そのサービスを利用して、E!Kit-1100 が固有の IP アドレスで TFTP/DHCP を利用してネットワークブートし、別のディレクトリを NFS マウントする事も設定によって勿論可能です。しかし各サービスの詳細な設定に関しては多岐に渡り、また多くのの参考文献があるため、E!Kit-1100 の説明としては取り上げません。詳細は WWW 上の検索エンジンの検索結果等を参考にして下さい。

a. TFTP サーバ

TFTP サーバ用の特別な設定ファイルはありません。xinetd 経由でのサービスを有効にするために、/etc/xinetd.d 以下に以下のような内容の tftp ファイルを作成します。

xinetd 設定ファイル /etc/xinetd.d/tftp

```
service tftp
{
        socket_type             = dgram
        protocol                = udp
        wait                    = yes
        user                    = root
        server                  = /usr/sbin/in.tftpd
        server_args             = -c -s /tftpboot
        disable                 = no
        per_source              = 11
        cps                     = 100 2
}
```

また、ネットワークブートで使用するカーネルを TFTP サーバが参照するディレクトリにコピーしておきます。

```
# cp -p /home/rootfs/rootfs-aml3/boot/2.4.26-ek1.srec /tftpboot
```

b. DHCP サーバ

DHCP サーバ設定ファイル /etc/dhcpd.conf

```
ddns-update-style interim;
allow bootp;
not authoritative;
subnet 192.168.1.0 netmask 255.255.255.0 {
    range dynamic-bootp 192.168.1.10 192.168.1.99;
    default-lease-time 1209600;   # two weeks
    max-lease-time 31557600;      # one year

    host aurum {
      hardware ethernet 00:0E:6C:00:00:??;
      fixed-address 192.168.1.150;
      option root-path "/home/rootfs/rootfs";
    }
}
```

※上記の **hardware ethernet** の項目は、保証書に書いてあるシリアル番号(Mac アドレス)を':'(コロン)で区切って記述して下さい。

c. NFS サーバ

NFS サーバ設定ファイル /etc/exports

```
/home/rootfs *(rw,no_root_squash,no_all_squash)
```

NFS サーバを設定するためには、portmap、 nfs、 nfslock の 3 つのサービス・デーモンが動作している必要があります。

d. NTP サーバ

NTP サーバ設定ファイル /etc/ntp.conf

```
restrict 127.0.0.1
restrict 192.168.1.0 mask 255.255.255.0
server ntp1.jst.mfeed.ad.jp
server ntp2.jst.mfeed.ad.jp
server ntp3.jst.mfeed.ad.jp
server 127.127.1.0
fudge 127.127.1.0 stratum 10
driftfile /etc/ntp.drift
```

3.2.3. サービスの開始

ホスト PC にサーバ用バイナリパッケージをインストールし、各サービスの設定を行った後、chkconfig コマンド等を利用して、起動時に各ランレベルで自動的にサービスが立ち上がるようにホスト環境のシステムを設定します。設定するサービスのうち、常駐型サービスは ps コマンド等で動作中であるかどうかを確認する事ができます。動作中でない場合には再起動するか、/etc/init.d 以下のサービスデーモン・スクリプトを手動で起動して、サービスを開始して下さい。

ここでは、Red Hat 系の一般的なインストール方法を例にして説明していますので、パッケージやサーバの入手先、サービス形式によっては必ずしも以下の手順でサービスを開始させるとは限りません。ホスト環境において必要な４つのサービスを行う事が目的ですので、他の手順で設定しても構いません。

各ランレベルでのサービスの確認

```
# chkconfig --list
```

または

```
# chkconfig --list nfs(サービス名)
```

NFS 関連のサービスが on になっていない場合には、以下のように実行します。

```
# chkconfig --level 345 nfs on
# chkconfig --level 345 nfslock on
# chkconfig --level 345 portmap on
```

DHCP サービスが on になっていない場合には、以下のように実行します。

```
# chkconfig --level 345 dhcpd on
```

NTP 関連のサービスが on になっていない場合には、以下のように実行します。

```
# chkconfig --level 345 ntpd on
```

※Red Hat 系の例では、tftp サービスは通常、/etc/xinetd.d/tftp ファイルにて定義される、xinetd から呼び出されるサービスのため、サービスの立ち上げ設定は不要です。

3.2.4. ソースコードの入手先

ホスト PC にこれらのサーバをインストールするための、Red Hat 8.0 用のソースコードを入手する場合を例にして、入手先を以下に示します。

サーバプログラムのソースコード入手先

サーバ	ファイル名	入手先
NFS サーバ	nfs-utils-1.0.1-2.src.rpm	ftp://ftp.kddlabs.co.jp/pub/Linux/packages/RedHat/redhat/linux/8.0/ja/os/i386/SRPMS/
DHCP サーバ	dhcp-3.0pl1-9.src.rpm	ftp://ftp.kddlabs.co.jp/pub/Linux/packages/RedHat/redhat/linux/8.0/ja/os/i386/SRPMS/
TFTP サーバ	tftp-server-0.29-3.src.rpm	ftp://ftp.kddlabs.co.jp/pub/Linux/packages/RedHat/redhat/linux/8.0/ja/os/i386/SRPMS/
NTP サーバ	ntp-4.1.1.tar.gz	ntp プロジェクト http://www.ntp.org/downloads.html

これらのファイルは、ソースコードの形式で提供されますので、実際にサービスを運用するためには、ソースコードをコンパイルしてインストールし、その後各サービスの設定をする必要があります。

各サーバプログラムのコンパイルと設定方法は、それぞれのホスト PC やインストールしようとするソースコードの種類、バージョンによっても異なりますので、ここでは詳しく紹介しません。また、どうしても対応するソースコードが見つからない場合等には、例えば同じ Red Hat 系のディストリビューションから、上記 RPM ソースパッケージ(SRPM)を入手して、コンパイルして利用するなど、ネット上で探したものを利用してインストールする事も可能だと思われます。

3.2.5. RPM バイナリパッケージの入手先

NFS サーバ、DHCP サーバ、TFTP サーバ、NTP サーバのファイルは、実際にはほとんどの場合、Linux（ディストリビューション）のインストール CD-ROM に収録されている事が多いので、インストール時にオプション設定で選択しておく方が簡単です。通常は、「全てインストール」で必要な全てのサービスがインストールされます。またインストール後でも RPM バイナリパッケージから、これらサーバプログラムのコンパイル済みバイナリをインストールする事も可能です。

例えば、Red Hat 8.0 の場合を例にして、ホスト PC にこれらのサーバプログラムをインストールするための RPM バイナリパッケージの入手先を以下に示します。他のディストリビューションにも同様のパッケージが含まれますので、インストール CD-ROM か、それぞれの該当する入手先からの入手をお願い致します。ファイル名には該当のパッケージに依存したバージョン番号が付いていますので、パッケージによっては違う名前になります。

RPM バイナリパッケージの入手先（Red Hat 8.0 の例）

サーバ	ファイル名	入手先
NFS サーバ	nfs-utils-1.0.1-2.rpm	ftp://ftp.kddlabs.co.jp/pub/Linux/packages/RedHat/redhat/linux/8.0/ja/os/i386/Redhat/RPMS
DHCP サーバ	dhcp-3.0pl1-9.rpm	ftp://ftp.kddlabs.co.jp/pub/Linux/packages/RedHat/redhat/linux/8.0/ja/os/i386/Redhat/RPMS
TFTP サーバ	tftp-server-0.29-3.rpm	ftp://ftp.kddlabs.co.jp/pub/Linux/packages/RedHat/redhat/linux/8.0/ja/os/i386/Redhat/RPMS
NTP サーバ	ntp-4.1.1a-9.rpm	ftp://ftp.kddlabs.co.jp/pub/Linux/packages/RedHat/redhat/linux/8.0/ja/os/i386/Redhat/RPMS

3.2.6. RPM バイナリパッケージのインストール例

まず各 RPM バイナリパッケージが既にインストールされているかどうかを”rpm –q”コマンドを実行して確認します。この方法では、前記手順でソースコードを入手して rpm コマンドを使用せずにインストールしたものは見つかりませんので、注意が必要です。

```
# rpm -q nfs-utils
nfs-utils-1.0.1-2
```

と答えが返って来た場合には、nfs-utils-1.0.1-2 が既にインストールされているので、改め

て nfs-utils-1.0.1-2 をインストールする必要はありません。

```
# rpm -q dhcp
パッケージ dhcp-3.0pl1-9 はインストールされていません
```
または、
```
package dhcp-3.0pl1-9 is not installed
```

と答えが返ってきた場合には、dhcp-3.0pl1-9 がまだインストールされていないので、追加インストールが必要です。

次に不足しているサーバプログラムのインストールを、”rpm -ivh”コマンドを使用して行います。例えば、DHCP サーバ関連のバイナリを RPM バイナリパッケージからインストールする場合には、以下のようにします。

```
# rpm -ivh dhcp-*.rpm
```

同様に、TFTP サーバ関連のバイナリのインストール例です。

```
# rpm -ivh tftp-server-*.rpm
```

このようにして必要なすべてのサーバプログラムをインストールします。その後で、必要な「ネットワークとサービスの設定」を行って下さい。

4. カーネルのアップデート

4.1. 概要

この章では、E!Kit-1100 で使用するカーネルをアップデートする手順を示します。今回の手順では、2.4.26-ek1 にアップデートする場合を例に説明していますので、カーネルのバージョン番号「2.4.26-ek1」の部分は、アップデート対象のカーネルのバージョン名に合わせて変更をして下さい。またこの手順に従って頂ければ、独自に作成したカーネルを Tiny システムや、Large システムに組み込んで利用することができます。

Tiny システム用ルートファイルシステム rootfs-aml3-tiny と、Large システム用ルートファイルシステム rootfs-aml3 には、すでにカーネル 2.4.26-ek1 が含まれていますので、2.4.26-ek1 へのアップデートは必要ありません。「1.10.2 Tiny システム」あるいは「1.10.3 Large システム」を参照してブート時の設定をすれば、2.4.26-ek1 のバージョンで起動します。

アップデート手順は、(1)ホスト PC 上でカーネルソースをクロスコンパイル、(2)カーネルおよびモジュールをインストール、(3)inittab を編集、(4)YAMON モニタ環境変数を設定、となります。これらの手順について以下に説明します。

4.2. Tiny システムでのカーネルのアップデート

4.2.1. カーネルソースのクロスコンパイル（Tiny システム）

ホスト PC 上にクロス開発環境をまだ構築していない場合には、「2. クロス開発環境」を参照して構築して下さい。

まず、クロス開発環境用環境変数を設定します。

```
# . /export/tar/mipsel-env.sh
```

カーネルソースを展開します。

```
# cd /export/src
# tar xvjf ../tar/linux-2.4.26-ek1.tar.bz2
# ln -sf linux-2.4.26-ek1 linux
```

カーネルソースを初期状態にします。

```
# cd linux
# make mrproper
```

E!Kit-1100 用設定ファイルを標準設定ファイルにコピーします。

```
# cp arch/mips/defconfig-ekit1100 .config
```

カーネルを設定します。

```
# make oldconfig （標準設定ファイル .config の設定内容をそのまま使用する場合）
または
# make menuconfig（必要なカーネル・コンパイル・オプションを設定変更する場合）
```

ローダブル・モジュールを先にコンパイルして、カーネルに組み込む initrd RAMDISK イメージを作成します。

```
# make dep
# make modules
# sh ../../tar/mk_ramdisk.sh
```

カーネルをコンパイルして、RAMDISK イメージとともにリンクします。

```
# make zImage
```

4.2.2. カーネルおよびモジュールのインストール（Tiny システム）

まず、ホスト PC 上で一時的なディレクトリを作成し、そのディレクトリにカーネルとモジュールをインストールします。その後、このディレクトリを E!Kit-1100 Large システムに対してエクスポートします。

ホスト PC での操作（module のインストールと NFS エクスポート）：

```
# mkdir /export/tmp
# cp -p arch/mips/zboot/image/ekit1100.srec /export/tmp/2.4.26-ek1.srec
# make INSTALL_MOD_PATH=/export/tmp modules_install
# exportfs -o ro 192.168.1.150:/export/tmp
```

そして、E!Kit-1100 Large システム上で、ホスト PC からエクスポートされたディレクトリを NFS マウントし、また同時に CF をマウントします。そして、NFS マウントされたディレクトリにインストールされていたカーネルとモジュールを CF にインストールします。

E!Kit-1100 Large システムでの操作（S レコード形式のカーネルと module のコピー）：

```
# mkdir /mnt/nfs
# mount -t nfs 192.168.1.201:/export/tmp /mnt/nfs
```

```
# mount -t ext2 /dev/hda1 /mnt/disk
# cp -p /mnt/nfs/2.4.26-ek1.srec /mnt/disk
# cp -a /mnt/nfs/lib/modules/2.4.26-ek1 /mnt/disk/lib/modules
```

4.2.3. inittab の編集（Tiny システム）

カーネル 2.4.26-ek1 では、コンソール用の内蔵シリアルポートが従来までの ttyS3 から ttyS2 に変更されました。このため、 /etc/inittab を修正する必要があります。 /etc/inittab の以下の下線で示す部分を ttyS3 から ttyS2 に変更します。

E!Kit-1100 Large システムでの操作（inittab ファイルの書き換え）：

```
# vi /mnt/disk/etc/inittab
```

/mnt/disk/etc/inittab ファイル内容

```
::sysinit:/etc/init.d/rcS

::respawn:/sbin/getty tty1 38400 linux
::respawn:/sbin/getty tty2 38400 linux
::respawn:/sbin/getty tty3 38400 linux
::respawn:/sbin/getty -L ttyS2 115200 vt102

::restart:/sbin/init

::ctrlaltdel:/sbin/reboot

#::shutdown:/bin/umount -a -r
#::shutdown:/sbin/swapoff -a
```

E!Kit-1100 Large システム上でマウントしたファイルシステムをアンマウントします。

E!Kit-1100 Large システムでの操作（アンマウント）：

```
# cd /
# sync; sync; sync;
# umount /mnt/nfs
# umount /mnt/disk
```

ホスト PC 上でエクスポートしたディレクトリをアンエクスポートします。

ホスト PC での操作（NFS のアンエクスポート）：

```
# exportfs -u 192.168.1.150:/export/tmp
```

4.2.4. YAMON モニタ環境変数の設定（Tiny システム）

YAMON モニタ環境変数を次のように設定して CF の中の 2.4.26-ek1.srec（Motorola S-records 形式で保存された Linux カーネル）から Linux カーネルが起動するようにします。これで次回以降、E!Kit-1100 を起動すると Tiny システムがカーネル 2.4.26-ek1 で立ち上がります。

```
YAMON> setenv bootfile 2.4.26-ek1.srec
YAMON> setenv bootprot file
YAMON> setenv start “ load //hda1; go . root=/dev/hda1 ”
```

4.3. Large システムでのカーネルのアップデート

4.3.1. カーネルソースのクロスコンパイル（Large システム）

Tiny システムの場合と同様ですので、「4.2.1 カーネルソースのクロスコンパイル（Tiny システム）」を参照して下さい。同じものが使用できます。

4.3.2. カーネルおよびモジュールのインストール（Large システム）

カーネルは TFTP サーバが参照するディレクトリにインストールし、モジュールはエクスポートするルートファイルシステムにインストールします。

```
# cp -p arch/mips/zboot/image/ekit1100.srec /tftpboot/2.4.26-ek1.srec
# make INSTALL_MOD_PATH=/home/rootfs/rootfs modules_install
```

4.3.3. inittab の編集（Large システム）

カーネル 2.4.26-ek1 では、コンソール用の内蔵シリアルポートが従来までの ttyS3 から ttyS2 に変更されました。このため、 /etc/inittab を修正する必要があります。 /etc/inittab の以下で下線で示す部分を ttyS3 から ttyS2 に変更します。

```
# vi /home/rootfs/rootfs/etc/inittab
```

/home/rootfs/rootfs/etc/inittab ファイル（該当箇所を抜粋）

```
# Run gettys in standard runlevels
1:12345:respawn:/sbin/agetty -L 115200 ttyS2 vt102
~~:S:respawn:/sbin/agetty -L 115200 ttyS2 vt102
#1:2345:respawn:/sbin/mingetty tty1
#2:2345:respawn:/sbin/mingetty tty2
#3:2345:respawn:/sbin/mingetty tty3
#4:2345:respawn:/sbin/mingetty tty4
#5:2345:respawn:/sbin/mingetty tty5
#6:2345:respawn:/sbin/mingetty tty6
```

4.3.4. YAMON モニタ環境変数の設定（Large システム）

YAMON モニタ環境変数を次のように設定します。これで次回以降、E!Kit-1100 を起動すると Large システムがカーネル 2.4.26-ek1 で立ち上がります。

```
YAMON> setenv bootfile 2.4.26-ek1.srec
YAMON> setenv bootprot tftp
YAMON> setenv start “ load; go . nfsroot=192.168.1.201:/home/rootfs/rootfs ”
```

5. 補足事項

5.1. NFS サーバを利用した Large システムの設定手順

5.1.1. はじめに

NFSサーバを利用したLargeシステムでのブートの手順は以下の通りです。

a. TFTP サーバからカーネルをダウンロード
b. DHCP サーバへの問い合わせによりネットワーク情報を確定
c. ルートファイルシステムを NFS マウント
d. NTP サーバから時刻を取得

5.1.2. トラブル・シューティング

a. TFTP サーバからのカーネルのダウンロード

TFTP サーバからのカーネルのダウンロード時、シリアルポートに接続した端末エミュレーション・ソフト上に”**.....**”というカーネルロード実行中を示す表示が見られない場合には、TFTP サーバからのカーネルのダウンロードに失敗しています。YAMON モニタの環境変数の設定、カーネルのパス名、TFTP サーバへの接続を確認して下さい。

b. DHCP サーバへの問い合わせによるネットワーク情報の確定

DHCP サーバの動作を確認する場合には、以下のように実行して下さい。

```
# ps ax | grep dhcpd
```

もし動作していなければ、以下のように実行して DHCP サーバを起動して下さい。

```
# /etc/init.d/dhcpd start
```

DHCP サーバが動作しているにもかかわらず、ネットワーク情報を取得できない場合には、DHCP サーバの設定に誤りがあるかもしれません。DHCP サーバ設定ファイル /etc/dhcpd.conf を以下のように編集して、強制的に IP アドレスを E!Kit-1100 に割り当てるように設定してみて下さい。（設定変更後、以下のコマンドで DHCP サーバを再起動して下さい。）

```
# /etc/init.d/dhcpd stop
# /etc/init.d/dhcpd start
```

＜強制的にIPアドレスをE!Kit-1100に割り当てる /etc/dhcpd.conf ファイルの内容＞

```
ddns-update-style interim;
allow bootp;
```

```
not authoritative;
subnet 192.168.1.0 netmask 255.255.255.0 {
    range dynamic-bootp 192.168.1.2 192.168.1.254;
    default-lease-time 1209600;   # two weeks
    max-lease-time 31557600;      # one year

    host aurum {
      hardware ethernet 00:0E:6C:00:00:??;
      fixed-address 192.168.1.150;
      option root-path "/home/rootfs/rootfs";
    }
}
```

※上記の hardware ethernet の項目は、保証書に書いてあるシリアル番号(Macアドレス)を':'(コロン)で区切って記述して下さい。

c. ルートファイルシステムの NFS マウント

NFS 関連の設定は、以下のように実行して確認して下さい。

```
# exportfs
/home/rootfs            <world>
```

上記のように表示されない場合には、以下のように実行して手動でファイルシステムをエクスポートして下さい。

```
# exportfs -a
```

またルートファイルシステムの NFS マウントでは、ホスト PC の /home/rootfs/rootfs ディレクトリを NFS マウントする設定になっていますので、以下のコマンドを実行して、次のように正しく、Large ルートファイルシステムが展開されており、そこにシンボリックリンクが張られている事を確認して下さい。

```
# ls -l /home/rootfs
lrwxrwxrwx    1 root     root           11 Mar  4 04:24 rootfs -> rootfs-aml3
drwxrwxrwx   18 root     root         4096 Mar 25 08:09 rootfs-aml3
```

5.2. CF のフォーマット手順

5.2.1. はじめに

E!Kit-1100 では CF にファイルシステムを作成し、それをマウントして使用する事ができます。市販の CF メディアカードは、FAT 形式でフォーマットされているため、Linux のファイルシステムには適しません。以下に、E!Kit-1100 を使って、CF をフォーマットした後、マウントする手順を示します。

初期化の前にまず、cardmgr が動作しているのを確認してメディアを装着します。

```
# ps ax | grep cardmgr
```

※ご注意

E!Kit-1100 では、CF メディアカードが１枚の場合は、マウントしているスロットの位置に関係なく、常に /dev/hda として認識されます。CF メディアカードが２枚装着されている場合には、２番目の CF スロットのメディアが /dev/hdc として認識されます。

5.2.2. パーティションタイプの設定

Linux で使用するために、以下のようにしてパーティションタイプを設定します。

太字が入力部分です。

```
# fdisk /dev/hda
```

または、/dev/hdc（２台目の場合）

（現在のパーティション情報を表示します）

```
Command (m for help): p

Disk /dev/hda: 8 heads, 32 sectors, 978 cylinders
Units = cylinders of 256 * 512 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1   *         1       977    125040    6  FAT16
```

（現在のパーティション「FAT16」を削除します）

```
Command (m for help): d
Partition number (1-4): 1
```

（新しい Linux 用パーティションを作成します）

```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-978, default 1): <改行だけ入力>
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-978, default 978): <改行だけ入力>
Using default value 978
```

（作成したパーティション情報を表示します）

```
Command (m for help): p

Disk /dev/hda: 8 heads, 32 sectors, 978 cylinders
Units = cylinders of 256 * 512 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hdc1   *         1       978     62576   83  Linux
```

（パーティション情報を書き込んで終了します）

```
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Re-read table failed with error 16: Device or resource busy.
Reboot your system to ensure the partition table is updated.

WARNING: If you have created or modified any DOS 6.x
partitions, please see the fdisk manual page for additional
information.
Syncing disks.
```

5.2.3. ext2 形式のフォーマット

E!Kit-1100 では標準の CF メディア・フォーマットとして ext2 ファイルシステムを使用します。以下のようにして CF に ext2 ファイルシステムを作成します。

```
# mke2fs /dev/hda1
```

または、**/dev/hdc1**（２台目の場合）

5.2.4. CF 内に作成したファイルシステムのマウント／アンマウント

/dev/hda1 に作成した ext2 ファイルシステムをマウントポイント /mnt/disk にマウントする例を以下に示します。

```
# mount -t ext2 /dev/hda1 /mnt/disk
```

※ご注意１

CF メディアにデータを書き込んだ直後には、キャッシュされているため、実際の書き込みが終っていない可能性があります。データの損傷を防ぐためには、CF メディアを抜く前に、必ず、sync と umount を実行して下さい。

```
# cd /
# sync; sync; sync;
# umount /dev/hda1
```

※ご注意２

この手順によっても利用できない CF メディアや、システムが認識できない CF メディアがありますので、注意して下さい。問題がある CF メモリカードでは、fdisk, mke2fs が正常に終了し、なおかつ mount も正常にできるように見えますが、以下のような症状で判別する事ができます。

a. 「mke2fs」でフォーマット後、マウントするとルート・ディレクトリに lost+found ディレクトリが見当たらない
b. マウント後ファイルの書き込みと読み出しを実行すると、エラーになったりファイルが見えなくなったりする。
c. 「fdisk」を再度実行してパーティション情報を表示させると、System が FAT16 になっている。

また、一部の古い Linux システムの mke2fs でフォーマットしたファイルシステムからは、カーネルがブートしない場合があります。そのような場合には、実績がある Red Hat 7.3 以降のシステムを使用して、フォーマットをし直して下さい。

5.3. 内蔵フラッシュ ROM への Linux カーネルの焼き込み

5.3.1. はじめに

E!Kit-1100 では、ブート ROM が CF 内の Linux カーネルを呼び出して起動するモードを持つため、通常の利用方法ではブート ROM を書き換える必要はありませんが、Linux カーネルをブート ROM 上に置くことによって、CF の記憶容量を有効に活用し、起動時間をより速くすることが可能です。また、ブート ROM から Linux カーネルを直接起動させる事ができない CF メモリカードや、USB 接続の外部記憶装置等を、起動時にルートファイルシステムとしてマウントさせる事も可能です。

なおこの方法は、お客様の便宜のために紹介していますが、**メモリアドレスの書き間違いや、Flash メモリの内容破壊によるトラブルの原因になる可能性がありますので、推奨は致しません（万が一、Flash メモリの内容破壊により起動できなくなった場合の Flash メモリの書き換え作業は実費、有償で承ります）。**実施される場合は、お客様自身の技量に応じて、慎重に作業を進められるようにお願い致します。

内蔵フラッシュ ROM への Linux カーネルの焼き込みは、tftp ブートができる環境を設定しておいて、操作はすべて YAMON モニタから行います。太字が入力部分です。カーネルの大きさによって、入力する値が違いますので、注意して下さい。

5.3.2. カーネル部分の削除

```
YAMON> erase 0xBFD00000 0x2C0000
what...
The following area will be erased:
Start address = 0x1fd00000
Size          = 0x002c0000
Confirm ? (y/n) y
Erasing...Done
```

5.3.3. カーネルのロード

TFTP サーバから焼き込むカーネルをネットワーク経由でロードします。load 実行時に表示されるカーネルサイズ(下線部)に注意します。

```
YAMON> load
About to load tftp://XXX.XXX.XXX.XXX/ekit1100.srec
Press Ctrl-C to break
.....................................
```

```
.....................................
.....................................
.....................................
.....................................
.....................
Start = 0x81000000, range = (0x81000000,0x81196fff), format = SREC
```

5.3.4. Flash メモリへの転送

下線はカーネルのサイズに適切に合わせる必要があります。

```
YAMON> copy 0x81000000 0xBFD00000 0x197000
Copying...Done
```

5.3.5. ブートパラメータ設定

YAMON モニタの環境変数に起動時に実行するブートパラメータを設定します。

a. ネットワークブート用のパラメータ設定

```
YAMON> setenv start "copy 0xBFD00000 0x81000000 0x197000; go 0x81000000"
```

b. CF ブート用のパラメータ設定

CF から起動させる場合には、明示的に root=パラメータをカーネル起動時に渡す必要があります。１行では入力しきれないため、'¥'記号を使って２行に渡って入力します。

```
YAMON> setenv start "copy 0xBFD00000 0x81000000 0x197000; go 0x81000000 \
? root=/dev/hda1"
```

c. 設定を確認します。

```
YAMON> setenv

MAC         (R/W)  0
bootfile    (R/W)  ekit1100.srec
bootprot    (R/W)  tftp
bootserport (R/W)  tty1
bootserver  (R/W)  192.168.1.201
```

```
ethaddr     (R/W)  00.0e.6c.00.00.09
gateway     (R/W)  0.0.0.0
ipaddr      (R/W)  192.168.1.150
memsize     (RO)   0x08000000
modetty0    (R/W)  115200,n,8,1,none
modetty1    (R/W)  115200,n,8,1,none
prompt      (R/W)  YAMON
start       (R/W)  copy 0xBFD00000 0x81000000 0x197000; go 0x81000000 root=/dev
                   /hda1
subnetmask  (R/W)  255.255.255.0
```

5.4. ブート ROM の更新方法

YAMON ブート ROM は、以下の手順によって利用者がご自身で更新することができます。操作を誤ったり、ブート ROM 書き換え作業中に本体電源を切ったりした場合には、E!Kit-1100 は二度と起動しなくなりますので、十分慎重に作業をお願い致します。

1. ブート ROM 更新用ファイル yamon-02.19EKIT1100-070504.zip(zip 圧縮形式) を http://e-kit.jp/products/E!Kit1100/ からダウンロードします。
2. yamon-02.19EKIT1100-070504.zip を解凍し、 yamon-02.19EKIT1100-070504.rec.m(S レコード形式)を取り出します。そして、 yamon-02.19EKIT1100-070504.rec.m を、ホスト PC の /tftpboot ディレクトリに置きます。
3. ホスト PC に TFTP サーバ環境を設定・構築して、E!Kit-1100 が TFTP 経由でホスト PC から Linux カーネルを読み込めることを確認しておきます。
4. YAMON モニタを起動して、以下の手順でブート ROM を書き換えます。**太字**が入力部分です。**この作業の間、絶対に電源を切らないようにお願い致します。**

```
YAMON> load /yamon-02.19EKIT1100-070504.rec.m
About to load tftp://192.168.1.201/yamon-02.19EKIT1100-070504.rec.m
Press Ctrl-C to break
Start dump from terminal program
.....................................
.....................
Start = 0x9fc90000, range = (0xa0100000,0xa01cd9ef), format = SREC
YAMON>

YAMON> erase bfc00000 100000
what...
The following area will be erased:
Start address = 0x1fc00000
Size = 0x00100000
Confirm ? (y/n) y
Erasing...Done
YAMON>

YAMON> copy a0100000 bfc00000 100000
Copying...Done
```

YAMON>

※ご購入時にユーザ登録されているお客様が、万が一、ブート ROM の書き換えに失敗された場合は、 yamon-02.19EKIT1100-070504 への書き換えに限り、各登録ボード当たり 1 回のみ、弊社にて無償でブート ROM リカバリを行います。その際、弊社にボードをお送り頂く送料は、お客様がご負担ください。それ以外の失敗の場合の ROM 修復には、送料のほかに ROM 焼き込み作業費用（実費）がかかります。

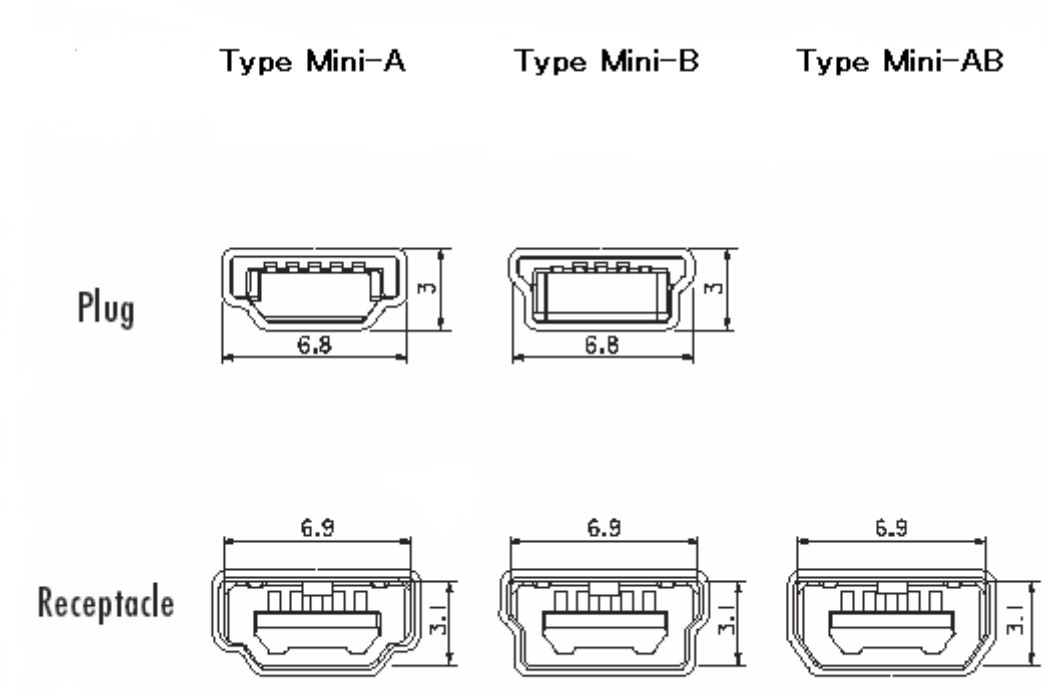
5.5. USB ミニ AB コネクタ

5.5.1. 概要

標準サイズの USB コネクタは、ホスト側で使用されるタイプ A と、デバイス側用のタイプ B があります。USB Implementers Forum では、USB OTG 規格の発表と合わせて、それぞれ小型化したミニ A コネクタ（白色）と、ミニ B コネクタ（黒色）、USB OTG 用のミニ AB コネクタ（グレー色）を規格に追加しました。ミニ AB コネクタは本来、ホストにもデバイスにもなれるデュアルロール・デバイスに採用されるコネクタです。

各ミニコネクタは携帯機器向けにサイズが小さくなった点と、ピン数が５ピンになった点が従来のタイプ A、タイプ B と異なります。ミニコネクタでは、デュアルロール・デバイスのミニ AB コネクタへの装着を想定して、新たにプラグを識別する役割の ID 端子が追加されました。USB OTG 用のミニ AB コネクタには、この新しい規格のミニ A ケーブルのプラグとミニ B ケーブルのプラグのどちらも装着できるようになっているため、ID 端子が GND に接続されていれば、プラグがミニ A で、すなわち ID 端子が接続されていなければ、ミニ B プラグが接続されていると判定します。

コネクタとプラグの形状は以下の図を参照して下さい。



5.5.2. ミニ A プラグ

ミニ A プラグを使用したケーブル（ミニ A プラグ＝ミニ B プラグや、ミニ A プラグ＝標準 B プラグ）は、現在パソコン関連ショップでは販売されておらず、NTT ドコモのショップで「ミニ A アダプタ」（ミニ A プラグ＝標準 A コネクタ・ソケット）がシグマリオン III 用として入手できるだけの様子です。

5.5.3. ミニ B プラグ

一方、ミニ B プラグを使用したケーブル（ミニ B プラグ＝標準 A プラグ）はディジタルカメラ用等で一般的に販売されています。ただしミニ B プラグに似た、小型の USB コネクタも従来からありますが、それらの多くはメーカ独自のもので、USB 規格のもとは違いますので注意して下さい。

5.5.4. USB ホスト・デバイス切り替えドライバ

E!Kit-1100 用の USB ミニ AB コネクタの機能を切り替えるとともに、現在の状態や接続されているケーブルの種類を表示する簡易的なドライバ usba.o を作成していますので、紹介します。特別なアプリケーションが無くても動作できるように、ユーザとのインタフェースは /proc ファイルを使用しています。実際の動作では、/proc ファイルの read と write 処理ルーチンを別な目的に使用しています。

a. 仕様

E!Kit-1100 でのこの USB ミニ AB コネクタの動作とケーブル識別の仕様は、次のようになっています。

Au1100 USB 機能の設定

信号名	アドレス	ビット	ターゲット設定	ホスト設定
sys_pinfunc	0xB190002C	USB(bit15)	0 (Device)	1 (Host)
usb_ctrl	0xB800000C	usbpdp(bit0)	0 (Pull-down OFF)	1 (Pull-down ON)
		usbpup(bit1)	0 (Pull-up ON)	0 (Pull-up OFF)
		usbpdm(bit2)	0 (Pull-down OFF)	1 (Pull-down ON)
		en0(bit6)	0 (VBUS OFF)	1 (VBUS ON)

E!Kit-1100 でのケーブル識別

信号名	アドレス	ビット	ミニ A	ミニ B
usb_status	0xB8000008	usbid(bit4)	0 (miniA)	1 (miniB)

b. 開発手順、動作例

デバイスドライバを作成してテストする手順です。コンパイルは、クロスコンパイルでもセルフコンパイルでも可能です。

コンパイル

```
# make usba.o          セルフコンパイルとオブジェクトの作成
                       （クロスコンパイル時には、ホスト側で同じコマンドを実行後、
                        usba.o をターゲットの E!Kit-1100 に転送しておきます）
# insmod usba.o              ローダブル・モジュールのロード
Using usba.o                 確認メッセージが表示されます

# echo 1 > /proc/net/usbsw   USB ミニ AB コネクタをホスト用に切り替え

# cat /proc/net/usbsw        現在の USB ミニ AB コネクタの状態表示
sys_pinfunc   = Host
usbpdp        = Host
usbpup        = Host
usbpdp        = Host
en0           = Host
```

```
# cat /proc/net/usbsw          デバッグ情報表示
sys_pinfunc = 00009481
usb_ctrl      = 500050C5
usb_status    = 50005013

# echo 0 > /proc/net/usbs      USB ミニ AB コネクタをデバイス用に切り替え

# cat /proc/net/usbsw          現在の USB ミニ AB コネクタの状態表示
sys_pinfunc    = Device
usbpdp          = Device
usbpup          = Device
usbpdp          = Device
en0             = Device

# cat /proc/net/usbsw          デバッグ情報表示
sys_pinfunc = 00001481
usb_ctrl      = 50005082
usb_status    = 50005013

# cat /proc/net/usbstat        USB ミニ AB コネクタのプラグ状態表示
usb_status    = miniB          未接続、またはミニ B 接続時

# cat /proc/net/usbstat        USB ミニ AB コネクタのプラグ状態表示
usb_status    = miniA          ミニ A 接続時
```