



AJ010 User's Guide

Version 2.03

2004 年 12 月 6 日

株式会社アットマークテクノ
<http://www.atmark-techno.com/>

Armaddillo 公式サイト
<http://armaddillo.atmark-techno.com/>

目次

1.	はじめに.....	1
1.1.	マニュアルについて.....	1
1.2.	フォントについて.....	1
1.3.	コマンド入力例の表記について.....	1
1.4.	商標について.....	1
1.5.	謝辞.....	2
2.	安全に関する注意事項.....	3
2.1.	安全に関する注意事項.....	3
2.2.	取り扱い上の注意事項.....	3
2.3.	ソフトウェア使用に関しての注意事項.....	3
3.	Armadillo-J の概要.....	4
3.1.	特長.....	4
3.2.	機能および利用例.....	4
3.2.1.	ネットワーク経由の機器制御.....	4
3.2.2.	Linux 端末.....	4
3.2.3.	オリジナルの組込み機器の開発.....	4
4.	作業の前に.....	5
4.1.	準備するもの.....	5
4.2.	接続方法.....	5
5.	Flash メモリの書き換え方法.....	6
5.1.	ダウンローダのインストール.....	6
5.2.	書き換え手順.....	6
5.2.1.	ジャンパピンの設定.....	6
5.2.2.	書き換えイメージの転送.....	7
6.	使用方法.....	9
6.1.	起動の前に.....	9
6.2.	起動.....	10
6.3.	ディレクトリ構成.....	12
6.4.	データ保存方法.....	12
6.5.	終了.....	13
6.6.	ネットワーク設定.....	13
6.6.1.	固定 IP アドレスで使用する場合.....	13
6.6.2.	DHCP を使用する場合.....	14
6.7.	telnet ログイン.....	14
6.8.	ファイル転送.....	14
6.9.	Web サーバ.....	14
7.	開発環境の準備.....	15
7.1.	ツールチェーンのインストール.....	15
7.2.	環境変数の設定.....	15
7.3.	ソースコードの準備.....	15
8.	uClinux の image を作成.....	16
8.1.	作業の流れ.....	16
8.2.	Menuconfig.....	18
8.2.1.	メインメニュー.....	18
8.2.2.	Target Platform Selection メニュー.....	19
8.3.	ビルド.....	21
9.	uClinux の image を作成.....	22
9.1.	カーネルの設定変更.....	22

9.2.	ユーザランドの設定変更.....	22
10.	自作アプリケーションの作成.....	25
10.1.	ディレクトリの作成.....	25
10.2.	Makefile の作成.....	26
10.3.	C ソースコード.....	27
10.4.	コンパイル.....	27
10.5.	ROMFS ディレクトリへのインストール.....	27
10.6.	イメージファイルの生成と実行.....	28
11.	Flat Binary Format.....	29
11.1.	Flat Binary Format の特徴.....	29
11.2.	実行ファイルの圧縮.....	29
11.2.1.	コンパイル済バイナリファイルを圧縮.....	29
11.2.2.	コンパイル時に圧縮.....	30
11.3.	スタックサイズの指定.....	31
11.3.1.	コンパイル済バイナリファイルスタックサイズを変更.....	31
11.3.2.	コンパイル時にスタックサイズを指定.....	31
12.	トラブルシューティング.....	33
12.1.	正常に起動しない場合.....	33
12.2.	購入時の状態に戻すには.....	33
12.3.	ブートローダ(hermit)の更新方法.....	33
12.4.	ブートローダ(hermit)の復旧方法.....	33
13.	Appendix.....	36
13.1.	Windows 上に開発環境を構築する方法.....	36
13.1.1.	coLinux のインストール.....	36
13.1.2.	環境構築用ファイルの準備.....	36
13.1.3.	coLinux の実行.....	36
13.1.4.	ネットワークの設定.....	36
13.1.5.	coLinux ユーザの作成.....	37
13.1.6.	Windows-coLinux 間のファイル共有.....	37
13.1.7.	クロス開発環境の導入.....	38
13.1.8.	特殊な場合の Windows ネットワーク設定方法.....	38
13.1.9.	coLinux のネットワーク設定方法.....	38
13.2.	ベースイメージについて.....	41
13.2.1.	ベースイメージのメモリマップ.....	41
13.2.2.	コマンド一覧.....	42
13.2.3.	起動デーモン一覧.....	42
13.3.	オリジナルデバイスドライバ仕様.....	43
13.3.1.	パラレルポートドライバ.....	43
13.3.2.	LED 制御用ドライバ.....	45
13.4.	URL リスト.....	47

表目次

表 1-1	使用しているフォント	1
表 1-2	表示プロンプトと実行環境の関係	1
表 5-1	ジャンパピン切り替え時の対応動作.....	7
表 5-2	Hermit for WIN32 版使用時のパラメータ例.....	8
表 6-1	シリアル通信設定	9
表 6-2	シリアルコンソールログイン時のユーザ名とパスワード	11
表 6-3	ディレクトリ構成の一覧	12
表 6-4	書き込み属性の説明.....	12
表 6-5	ネットワーク設定詳細	13
表 6-6	telnet ログイン時のユーザ名とパスワード	14
表 6-7	ftp のユーザ名とパスワード.....	14
表 12-1	シリアル通信設定	34
表 13-1	ネットワーク設定	39
表 13-2	メモリマップ(Flash メモリ).....	41
表 13-3	メモリマップ(RAM)	41
表 13-4	コマンド一覧	42
表 13-5	起動デーモン一覧	42
表 13-6	パラレルポートノード一覧.....	43
表 13-7	LED 制御用デバイスノードのパラメータ	45
表 13-8	書き込みデータと対応する状態.....	45
表 13-9	読み込みデータと対応する状態.....	45
表 13-10	URL リスト	47

図目次

図 4-1	Armadillo-J と作業用 PC の接続イメージ	5
図 5-1	展開処理コマンド入力例	6
図 6-1	起動ログ	11
図 6-2	データ保存用コマンド(killall -USR1 flatfsd)の発行例	13
図 6-3	ネットワーク設定例(固定 IP アドレス時)	13
図 6-4	ネットワーク設定例(DHCP 使用時)	14
図 7-1	開発用ツールチェーンの展開例	15
図 7-2	環境変数「PATH」の設定(bash の場合)	15
図 7-3	ソースコードの展開例	15
図 8-1	イメージファイル作成の流れ	17
図 8-2	uClinux v3.1.0 Configuration Main Menu	19
図 8-3	Target Platform Selection	19
図 8-4	Target Platform Selection	20
図 8-5	Default all setting の選択	21
図 9-1	ユーザランド用 Main Menu	22
図 12-1	ブートローダの更新手順	33
図 12-2	NS7520 XK BootLoader のメニュー	34
図 12-3	TFTP メニューの表示	34
図 12-4	TFTP setting での入力例	35

1.はじめに

1.1. マニュアルについて

本マニュアルは、Armadillo-J を使用する上で必要な情報のうち、以下の点について記載されています。

- Flash メモリの書き換え
- 基本的な使い方
- カーネルとユーザランドのビルド
- アプリケーション開発

Armadillo-J の機能を最大限に引き出すために、ご活用いただければ幸いです。

1.2. フォントについて

このマニュアルでは以下のようにフォントを使っています。

表 1-1 使用しているフォント

フォント例	説明
本文中のフォント	本文
[PC ~]\$ ls	プロンプトとユーザ入力文字列

1.3. コマンド入力例の表記について

このマニュアルに記載されているコマンドの入力例は、表示されているプロンプトによって、それぞれに対応した実行環境を想定して書かれています。「/」の部分はカレントディレクトリによって異なります。各ユーザのホームディレクトリは「~」で表わします。

表 1-2 表示プロンプトと実行環境の関係

プロンプト	コマンドの実行環境
[PC /]#	作業用 PC 上の特権ユーザで実行
[PC /]\$	作業用 PC 上の一般ユーザで実行
[AJ /]#	Armadillo-J 上の特権ユーザで実行
[AJ /]\$	Armadillo-J 上の一般ユーザで実行

1.4. 商標について

Armadillo は(株)アットマークテクノの登録商標です。
 その他の記載の会社名、製品名は、それぞれの登録商標または商標です。

1.5. 謝辞

Armadillo-J で使用しているソフトウェアは Free Software / Open Source Software で構成されています。Free Software / Open Source Software は世界中の多くの開発者の成果によってなりたっています。この場を借りて感謝の意を示したいと思います。

uClinux は D. Jeff Dionne 氏や Greg Ungere 氏、David McCullough 氏、さらに uClinux development list に参加しているすべての人の成果によって支えられています。

uClibc、Busybox は Eric Andersen 氏によって開発・保守されています。

2.安全に関する注意事項

2.1.安全に関する注意事項

Armadillo-J を安全にご使用いただくために、特に以下の点にご注意くださいますようお願いいたします。



本製品には一般電子機器用（OA機器・通信機器・計測機器・工作機械等）に製造された半導体部品を使用していますので、その誤作動や故障が直接生命を脅かしたり、身体・財産等に危害を及ぼす恐れのある装置（医療機器・交通機器・燃焼制御・安全装置等）に組み込んで使用したりしないでください。また、半導体部品を使用した製品は、外来ノイズやサージにより誤作動したり故障したりする可能性があります。ご使用になる場合は万一誤作動、故障した場合においても生命・身体・財産等が侵害されることのないよう、装置としての安全設計（リミットスイッチやヒューズ・ブレーカ等の保護回路の設置、装置の多重化等）に万全を期されますようお願い申し上げます。

2.2.取り扱い上の注意事項

劣化、破損、誤動作、発煙、発火の原因となることがあります。取り扱い時には以下のような点にご注意ください。

- **電源の投入**
Armadillo-J や周辺回路に電源が入っている状態では絶対に拡張 I/O コネクタの着脱を行わないで下さい。
- **静電気**
Armadillo-J には CMOS デバイスを使用しておりますので、ご使用になるまでは帯電防止対策のされている、購入時のパッケージ等にて保管して下さい。
- **インターフェース**
各インターフェース(外部 I/O、RS232C、Ethernet)には規定以外の信号を接続しないでください。信号の極性および入出力方向を間違わえないでください。
- **衝撃、振動**
落下や衝突などの強い衝撃を与えないでください。
振動部や回転部などへの搭載はしないでください。強い振動や遠心力を与えないでください。
- **高温低温、多湿**
極度に高温や低温になる環境や、湿度が高い環境では使用はしないでください。
- **塵埃**
塵埃の多い環境では使用はしないでください。

2.3.ソフトウェア使用に関しての注意事項

本製品に含まれるソフトウェア(付属のドキュメント等も含みます)は、現状のまま(AS IS)提供されるものであり、特定の目的に適合することや、その信頼性、正確性を保証するものではありません。また、本製品の使用による結果についてもなんら保証するものではありません。

3. Armadillo-J の概要

Armadillo-J はイーサネット内蔵 32 ビット ARM プロセッサ(NetSilicon 社 NS7520)を採用し、基板サイズで名刺の半分のサイズを実現した超小型ネットワークコンピュータです。

3.1. 特長

- Linux 搭載
標準 OS に Linux (uClinux Kernel 2.4.22 ベース) を採用しており、豊富なソフトウェア資産と実績のある安定性を提供します。また GNU 開発環境を利用して容易に開発が行えます。
- ネットワーク(Ethernet)
10Base-T/100Base-Tx の通信が可能です。
- シリアルポート
RS-232C (D-sub 9) 準拠のシリアルポートを持ち、シリアル機器との通信が可能です。データ転送レートは 600bps~230,400bps に対応しています。
- 汎用パラレル入出力(汎用 I/O)
汎用パラレル入出力 (5 ピン)があり、外部装置の制御に利用できます。

3.2. 機能および利用例

3.2.1. ネットワーク経由の機器制御

購入状態の Armadillo-J は

- シリアルーEthernet 変換
- ネットワーク経由汎用 I/O 制御

等の機能を持っており、そのままネットワーク経由の機器制御に利用することができます。

詳しくは別冊子「スタートアップガイド」を参照して下さい。

3.2.2. Linux 端末

開発キット付属の CD に収納されているベースイメージを Armadillo-J に書き込むことで通常の Linux 端末のように利用することができます。

ベースイメージでは

- telnet
- ftp サーバ
- Web サーバ

等のアプリケーションが動作します。

詳細は、本マニュアル「[5.Flash メモリの書き換え方法](#)」「[6.使用方法](#)」を参照して下さい。

3.2.3. オリジナルの組込み機器の開発

GNU 開発環境が利用でき、また多くのソースコードが公開されているため、Armadillo-J をカスタマイズして容易にオリジナルの組込み機器を開発することができます。詳しくは、本マニュアル「[10.自作アプリケーションの作成](#)」を参照して下さい。

4. 作業の前に

4.1. 準備するもの

Armadillo-J を使用する前に、次のものを準備して下さい。

- 作業用 PC
Linux もしくは Windows が動作し、1 ポート以上のシリアルポートを持つ PC です。
- シリアルクロスケーブル
D-Sub9 ピン (メス-メス) の「クロス接続用」ケーブルです。
- 開発キット付属 CD-ROM (以降、「付属 CD」)
Armadillo-J に関する各種マニュアルやソースコードが収納されています。
- シリアルコンソールソフト
minicom や Tera Term などのシリアルコンソールソフトです。(Linux 用のソフトは付属 CD の「tools」ディレクトリにあります。)
- AC アダプタ
DC8~48V 出力のものです。Armadillo-J の消費電力は 1.2W 程度です。

4.2. 接続方法

「シリアルクロスケーブル」を使って Armadillo-J と作業用 PC を接続します。

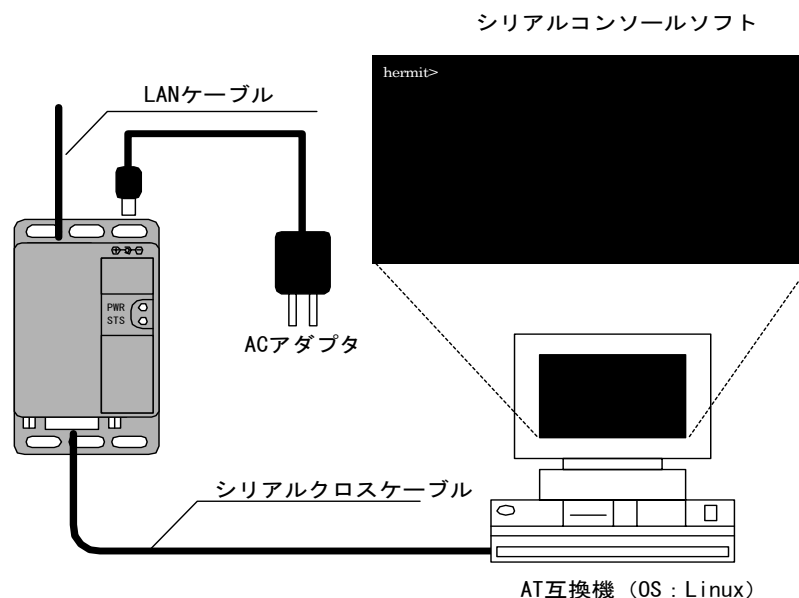


図 4-1 Armadillo-J と作業用 PC の接続イメージ

5. Flash メモリの書き換え方法

Flash メモリの内容を書き換えることで、Armadillo-J の機能を変更することができます。この章では Linux 端末として動作する「ベースイメージ」の書き込みを例に、Flash メモリの書き換え方法を説明します。



何らかの原因により「書き換えイメージの転送」に失敗した場合、Armadillo-J が正常に起動しなくなる場合があります。書き換えの最中は次の点に注意して下さい。

- Armadillo-J の電源を切らない。
- Armadillo-J と開発用 PC を接続しているシリアルケーブルを外さない。

Armadillo-J が正常に起動しなくなった場合は「[12.1.正常に起動しない場合](#)」を参照して下さい。

5.1. ダウンローダのインストール

作業用 PC に「ダウンローダ(hermit)」をインストールします。ダウンローダは Armadillo-J の Flash メモリの書き換えに使用します。

1) Linux の場合

付属 CD より「tools/hermit-1.3-armadillo.tgz」を用意し、ファイルを展開します。必ず root 権限で行って下さい

```
[PC ~]# tar xzvf hermit-1.3-armadillo.tgz -C /
```

図 5-1 展開処理コマンド入力例

付属 CD には、rpm(Red Hat パッケージ)、deb(Debian パッケージ)も用意しています。お使いの OS にあわせてご利用下さい。

2) Windows の場合。

付属 CD より「Hermit host for Win32 (tools/hermit-1.3-armadillo-4_win32.zip)」を任意のフォルダに展開します。

5.2. 書き換え手順

以下の手順で Flash メモリの書き換えを行います。

5.2.1. ジャンパピンの設定

電源を投入する前に、ジャンパピンを次のように設定します。

- JP1 : 「ショート」に設定
- JP2,3 : 使用する状況に合わせて設定(汎用 IO に何も接続していなければ「ショート」でよい)
- JP4 : 「ショート」に設定

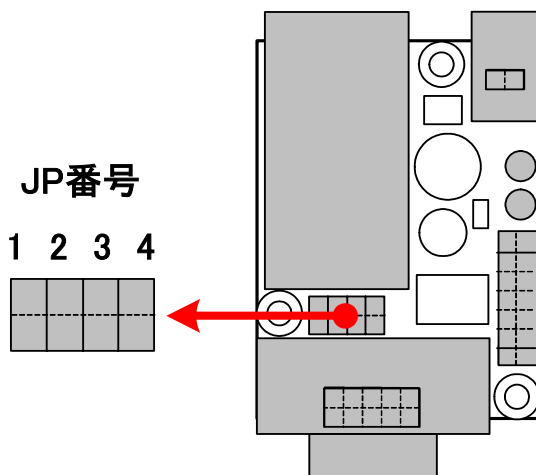


図 5-2 各 JP の位置

表 5-1 ジャンパピン切り替え時の対応動作

設定	ショート	開放
JP1	ネットワークモジュールとシリアルインターフェースの TXD,RXD を接続	ネットワークモジュールとシリアルインターフェースの TXD,RXD を分離
JP2	ネットワークモジュールとシリアルインターフェースの CTS,RTS を接続	ネットワークモジュールとシリアルインターフェースの CTS,RTS を分離
JP3	ネットワークモジュールとシリアルインターフェースの DTR,DSR,DCD を接続	ネットワークモジュールとシリアルインターフェースの DTR,DSR,DCD を分離
JP4	書き換えモードで起動	通常モードで起動

5.2.2. 書き換えイメージの転送

1) Linux の場合

Linux が動作する作業用 PC で terminal を起動し、hermit コマンドを入力します。
 下の図ではファイル名にベースイメージ(base. img)を指定しています。

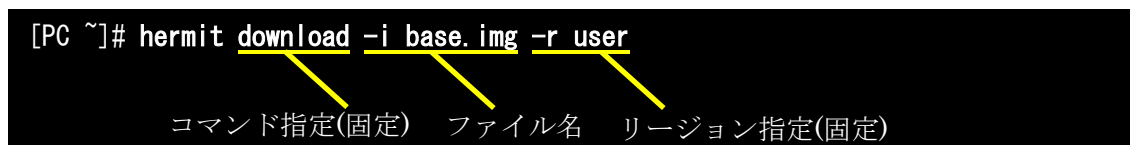


図 5-3 コマンド入力例

作業用 PC で使用するシリアルポートが「ttyS0」以外の場合、オプション「--port “ポート名”」を追加して下さい。

転送データのバイト数が「0x001c0000 (1835008) bytes.」と表示された時点で書き換えは終了です。
 書き換え終了後、JP4 を開放して Armadillo-J を再起動すると、ベースイメージが起動されます。

2) Windows の場合

「5.1 ダイナローダのインストール」にてファイルを展開したディレクトリにある、「Hermit host for Win32」を起動します。

各パラメータは以下の表を参考に設定してください。

表 5-2 Hermit for WIN32 版使用時のパラメータ例

項目名	説明	設定値
Port	Armadillo-J と接続しているポート名	COM1 (COM1 利用時)
Input file	書き込みを行うファイル名	base.img (パス名も含む)
Region	書き込む場所の指定	user (固定)

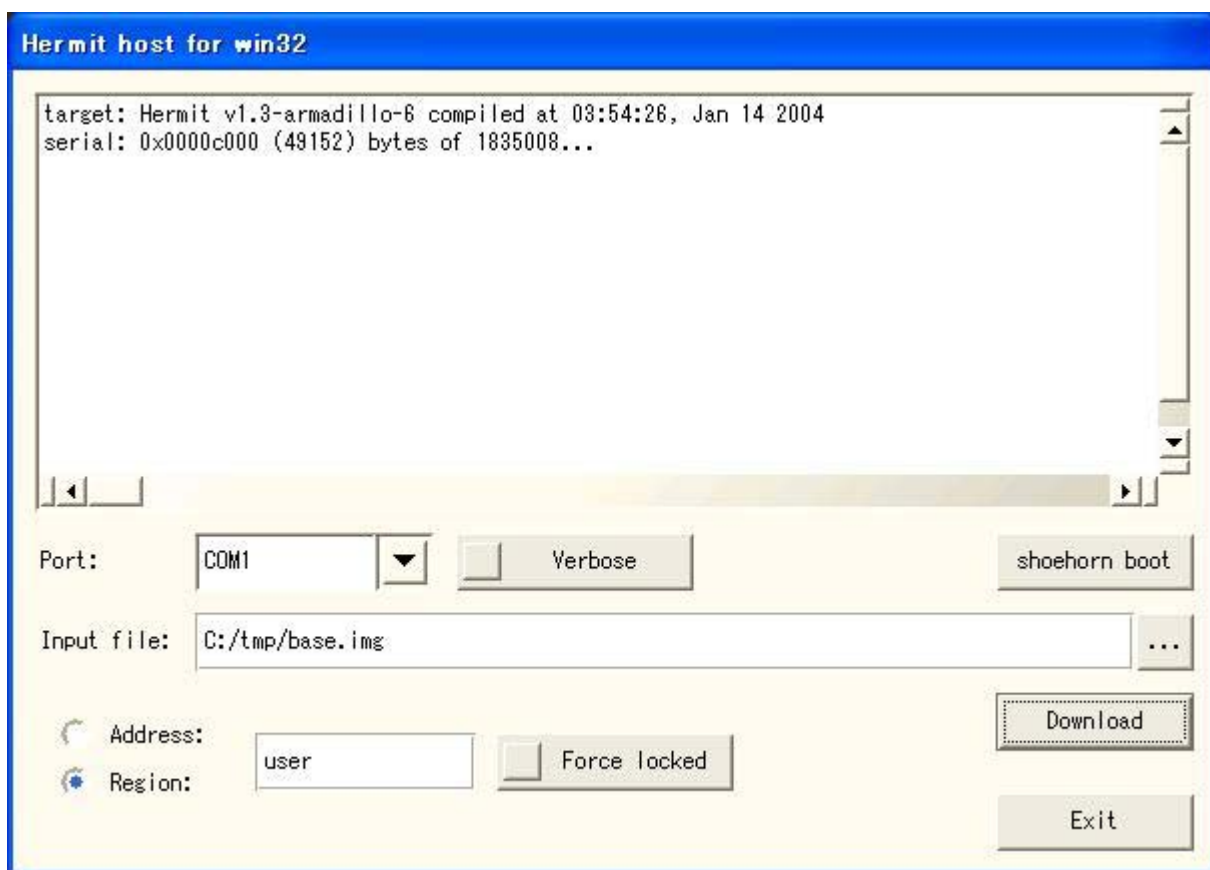


図 5-4 書き換えイメージの転送例

転送データのバイト数が「0x001c0000 (1835008) bytes.」と表示された時点で書き換えは終了です。書き換え終了後、JP4 を開放して Armadillo-J を再起動すると、ベースイメージが起動されます。

6.使用方法

開発キットの付属 CD の「image」ディレクトリには 3 種類の Flash メモリイメージが収納されていて、それぞれ機能が異なります。はじめにこれらの概要を説明します。

- リカバリーイメージ (recover. img)
製品購入時に Flash メモリへ書き込まれているイメージです。「シリアルーEthernet 変換」、「ネットワーク経由汎用 I/O 制御」等が使えます。このイメージを Flash メモリへ書き込むことによって、購入状態へ戻すことができます。このイメージを書き込んだ場合の使用方法は別冊子「スタートアップガイド」を参照して下さい。
- ベースイメージ (base. img)
アプリケーション開発のベースとなるイメージです。telnet、ftp、Web サーバの機能を持ち、シリアルポートまたはネットワークから Armadillo-J にログインすることができます。
- JFFS2 イメージ (jffs2. img)
変更の保存が可能なファイルシステムを利用しており、すべてのファイルの変更内容を保存することができます。機能的にはベースイメージと同等です。

この章ではベースイメージの使用方法について説明します。ベースイメージのメモリマップ、コマンド一覧は「[13.2.ベースイメージについて](#)」を参照してください。

6.1.起動の前に

ベースイメージを Flash メモリに書き込んでいない場合は、「[5.Flash メモリの書き換え方法](#)」を参照して、Flash メモリにベースイメージを書き込んでください。

Armadillo-J と作業用 PC をシリアルケーブルで接続し、シリアルコンソールソフトを起動します。次のように通信設定を行って下さい。

表 6-1 シリアル通信設定

項目	設定
転送レート	115,200bps
データ長	8bit
ストップビット	1bit
パリティ	なし
フロー制御	なし

6.2. 起動

JP4 を開放して Armadillo-J の電源を投入すると、ベースイメージが起動されます。正常に起動した場合、起動時に次ようなログが出力されます。

```
Copying      kernel...done.
Linux version 2.4.22-uc0-aj1 (somebody@colinux) (gcc version 2.95.3 20010315 (release) (ColdFire patches - 20010318 from http://fiddes.net/coldfire/) (uClinux XIP and shared lib patches from http://www.snapgear.com/)) #1 Tue Jun 8 05:13:34 UTC 2004
Processor: ARM/VLSI ARM 7 TDMI revision 0
Architecture: NET+ARM
fixup_netarm: Kernel memory start 0x00000000 end 0x000ea000
On node 0 totalpages: 2048
zone(0): 0 pages.
zone(1): 2048 pages.
zone(2): 0 pages.
Kernel command line: root=/dev/rom0
setup_timer : T2 CTL = D0000008
setting up timer IRQ
Calibrating delay loop... 8.93 BogoMIPS
Memory: 8MB = 8MB total
Memory: 6288KB available (728K code, 1072K data, 36K init)
Dentry cache hash table entries: 1024 (order: 1, 8192 bytes)
Inode cache hash table entries: 512 (order: 0, 4096 bytes)
Mount cache hash table entries: 512 (order: 0, 4096 bytes)
Buffer cache hash table entries: 1024 (order: 0, 4096 bytes)
Page-cache hash table entries: 2048 (order: 1, 8192 bytes)
POSIX conformance testing by UNIFIX
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
Initializing RT netlink socket
Starting kswapd
Net+ARM serial driver version 0.2 (2002-02-27) with CONSOLE enabled
ttyS00 at 0x0001 (irq = 15) is a NetARM
ttyS01 at 0x0002 (irq = 13) is a NetARM
ns7520port: port driver, (C) 2004 Atmark Techno, Inc.
Software Watchdog Timer: 0.05, timer margin: 60 sec
NS7520 Ethernet Driver Initialized
uclinux[mtd]: RAM probe address=0xe8b98 size=0xda000
uclinux[mtd]: root filesystem index=0
Initializing Armadillo-J MTD mappings
Amd/Fujitsu Extended Query Table v1.0 at 0x0040
number of CFI chips: 1
cfi_cmdset_0002: Disabling fast programming due to code brokenness.
Creating 7 MTD partitions on "Flash":
0x00000000-0x00020000 : "Flash/Reserved"
0x00020000-0x00040000 : "Flash/Hermit"
0x00040000-0x00200000 : "Flash/Image"
0x00040000-0x000b0000 : "Flash/Kernel"
```

```

0x000b0000-0x00200000 : "Flash/User"
0x001e0000-0x001f0000 : "Flash/Backup"
0x001f0000-0x00200000 : "Flash/Config"
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 512 bind 512)
VFS: Mounted root (romfs filesystem) readonly.
init started: BusyBox v0.60.5 (2004.06.08-05:15+0000) multi-call binary
Mounting proc: done
Mounting var: done
Mounting /etc/config: done
Populating /etc/config: FLATFSD: created 6 configuration files (502 bytes)
Populating /var: done
Mounting /home/guest/pub: done
Setting hostname: done
Setting up interface lo: done
Running local start scripts.
Starting flatfsd: done
Setting up network: Starting DHCP for interface :
ns7520_eth: PHY (0x13, 0x78e2) = LXT971A detected
ns7520_eth: link mode 100 Mbps full duplex (auto)
done
Starting inetd: done
Starting thttpd: done
Setting local time: done
Starting ledctrl: done

aj login:
    
```

図 6-1 起動ログ

ベースイメージには、次の 2 種類のユーザが用意されています。

表 6-2 シリアルコンソールログイン時のユーザ名とパスワード

ユーザ名	パスワード	権限
root	root	スーパーユーザ
guest	guest	一般ユーザ

6.3. ディレクトリ構成

ディレクトリ構成は次のようになっています。

表 6-3 ディレクトリ構成の一覧

ディレクトリ名	書き込み	説明	ファイルシステム
/bin	×	アプリケーション用	romfs
/dev		デバイスノード用	
/etc		システム設定用	
/etc/config	○	システム設定用(保存可能)	ramfs
/etc/default	×	システム設定復旧用	romfs
/lib		共有ライブラリ用	
/mnt		マウントポイント用	
/proc		プロセス情報用	
/root		root ホームディレクトリ	
/sbin		システム管理コマンド用	
/usr		ユーザ共有情報用	
/home		ユーザホームディレクトリ	
/home/guest/pub	△	ftp データ送受信用	ramfs
/tmp		テンポラリ保存用	
/var		変更データ用	

表 6-4 書き込み属性の説明

マーク	状態
×	書き込み不可
△	書き込み可能、電源の OFF/ON 後にデータが保持されない
○	書き込み可能、flatfsd の利用により電源の OFF/ON 後もデータが保持される

6.4. データ保存方法

ベースイメージでは/etc/config を、システム設定用ファイルを保存するディレクトリとして使用します。このディレクトリは flatfsd を使用することで、電源の OFF/ON 後も変更したデータを保持することが可能です。flatfsd はベースイメージの起動時に実行されます。

- データの保存方法
flatfsd に USR1 シグナルを送信すると、/etc/config ディレクトリのデータが Flash メモリの設定情報用の領域に書き込まれます。「killall -USR1 flatfsd」とコマンドを入力することで flatfsd に USR1 シグナルを送信できます。Flash メモリの設定情報用の領域は 64kbytes です。このサイズ以上の情報は保存できません。
- データの復元方法
「flatfsd - r」とコマンドを入力すると、flatfsd は Flash メモリの設定情報用の領域からデータを読み出し、/etc/config ディレクトリにコピーします。ベースイメージでは起動時に「flatfsd - r」コマンドを実行するため、自動的に設定情報が復元されます。Flash メモリの設定情報用の領域に書き込まれているデータが異常な時は、/etc/default ディレクトリの内容を/etc/config にコピーします。

ネットワーク設定など、/etc/config ディレクトリにあるシステム情報ファイルを更新したときには、「killall -USR1 flatfsd」とコマンドを入力してデータを保存してください。このコマンドはルート権限で実行する必要があります。

```
[Armadillo-J ~]# killall -USR1 flatfsd
[Armadillo-J ~]#
```

図 6-2 データ保存用コマンド(killall -USR1 flatfsd)の発行例

6.5. 終了

電源を切断することで Armadillo-J を終了させます。ただしシステム設定用の/etc/config ディレクトリ内に変更を加え、次回起動時にもその変更を保持したい場合には、終了前に「killall -USR1 flatfsd」コマンドを発行する必要があります。

6.6. ネットワーク設定

Armadillo-J 内の「/etc/config/network」ファイルを編集することで、ネットワークの設定を変更することができます。Armadillo-J にログインし、必ず root 権限で設定ファイルの変更作業を行って下さい。変更作業用のエディタとして vi が利用できます。

6.6.1. 固定 IP アドレスで使用する場合

固定 IP アドレスを指定する場合の設定例を次に示します。

表 6-5 ネットワーク設定詳細

項目	設定値
IP アドレス	192.168.10.10
ネットマスク	255.255.255.0
ブロードキャストアドレス	192.168.10.255
デフォルトゲートウェイ	192.168.10.1

```
#!/bin/sh

PATH=/bin:/sbin:/usr/bin:/usr/sbin

ifconfig eth0 192.168.10.10 netmask 255.255.255.0 \
    broadcast 192.168.10.255 up
route add default gw 192.168.10.1 eth0
```

図 6-3 ネットワーク設定例(固定 IP アドレス時)

6.6.2. DHCP を使用する場合

DHCP を利用して IP アドレスを取得する場合の設定例を次に示します。

```
#!/bin/sh

PATH=/bin:/sbin:/usr/bin:/usr/sbin

/bin/dhcpd &
```

図 6-4 ネットワーク設定例(DHCP 使用時)

変更した設定を電源 OFF 後も保持するためには、終了前に「killall -USR1 flatfsd」コマンドを発行する必要があります。

6.7.telnet ログイン

次のユーザ名/パスワードで telnet ログインが可能です。root でのログインは行えません。root 権限が必要な作業を行う場合、guest でログイン後、「su」コマンドにて root 権限を取得して下さい。

表 6-6 telnet ログイン時のユーザ名とパスワード

ユーザ名	パスワード
guest	guest

6.8. ファイル転送

ftp によるファイル転送が可能です。次のユーザ/パスワードでログインして下さい。ホームディレクトリは「/home/guest」です。「/home/guest/pub」ディレクトリに移動することでデータの書き込みが可能になります。

表 6-7 ftp のユーザ名とパスワード

ユーザ名	パスワード
guest	guest

6.9.Web サーバ

tthttpd という小さな HTTP サーバが起動しており、Web ブラウザから Armadillo-J をブラウズすることが出来ます。データディレクトリは「/home/www」です。URL は「http://(Armadillo-J の IP アドレス)」になります。(例 http://192.168.0.100/)

7. 開発環境の準備

Linux および Windows 上で Armadillo-J の開発を行うことができます。

Windows 上で開発を行う場合は、Windows 上に Linux 環境を構築する「coLinux」が必要になりますので、「[13.1. Windows 上に開発環境を構築する方法](#)」を参照してインストールして下さい。

7.1. ツールチェーンのインストール

付属 CD より「cross-dev/arm-elf-tools-20030314.sh」を用意して実行します。実行は必ず root 権限で行って下さい。「arm-elf-tools-20030314.sh」はコンパイラ、binutils、uClibc ライブラリを含んだ、開発ツールチェーンのインストーラです。開発ツールチェーンは「/usr/local/bin」にインストールされます。

```
[PC ~]# sh ./arm-elf-tools-20030314.sh
```

図 7-1 開発用ツールチェーンの展開例

7.2. 環境変数の設定

開発ツールチェーンを使いやすくするために、開発ツールチェーンの実行ファイルが入っているディレクトリを PATH 環境変数に追加します。シェルによって設定方法が異なりますので、詳しくはお使いのシェルのマニュアルを参照してください。

ここでは bash の設定方法を例に取ります。

```
[PC ~]$ export PATH="$PATH:/usr/local/bin"
[PC ~]$ echo $PATH
/usr/bin:/bin:/usr/bin/X11:/usr/sbin:/sbin:/usr/local/bin
[PC ~]$
```

図 7-2 環境変数「PATH」の設定(bash の場合)

7.3. ソースコードの準備

uClinux.org ではカーネル、ライブラリ、アプリケーションのソースコードをまとめて Full Source Distribution として配布しています。uClinux.org で配布している Full Source Distribution のファイル名は uClinux-dist-YYYYMMDD.tar.gz となっています。この uClinux-dist にいくつかの修正を加えたものを Armadillo-J 用に作成しています。Armadillo-J 用の uClinux-dist は CDROM の dist ディレクトリに uClinux-dist.tar.gz というファイル名で入っています。

uClinux-dist.tar.gz はどこに展開しても問題はありません。作業のしやすい場所に展開してください。このマニュアルでは便宜上~/に展開することにします。

```
[PC ~]$ gzip -cd uClinux-dist.tar.gz | tar xvf -
[PC ~]$ ls
uClinux-dist
[PC ~]$
```

図 7-3 ソースコードの展開例

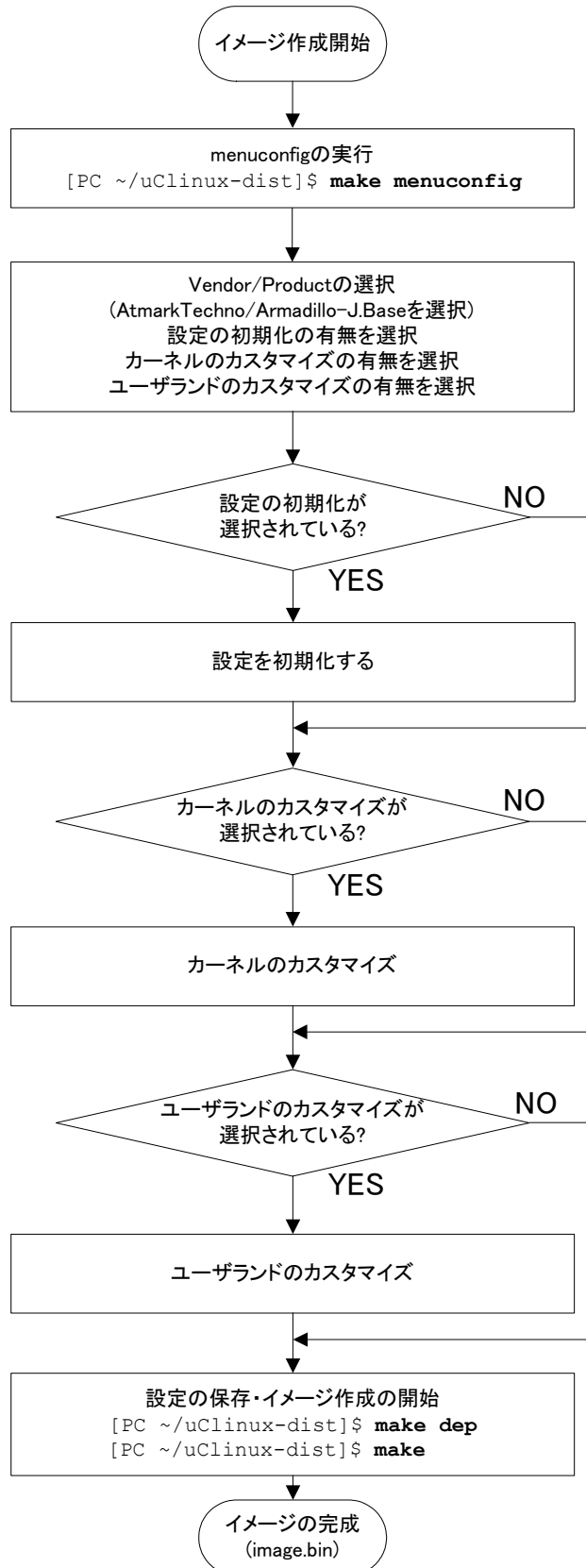
8. uClinux の image を作成

実際に、インストールした開発ツールチェーンとソースコードを使って Flash メモリに書き込むためのイメージファイルを作成します。uClinux-dist では Linux Kernel のビルドシステムと同じ方法でカスタマイズできるようになっています。Linux Kernel を自分でコンパイルしている人にとっては扱いやすいと思います。このマニュアルでは、menuconfig を例にとって説明していきます。

8.1. 作業の流れ

まず、イメージファイル作成の流れを示します。

図 8-1 イメージファイル作成の流れ



作業は大きく初期化とカーネル、ユーザランドの 3 つに分られます。



初期化を選択するとカーネルとユーザランドの両方が初期化されてしまいます。また、初期化を選ぶ前までの作業は破棄されてしまいますので、注意が必要です。

8.2. Menuconfig

Linux カーネルと同じく uClinux-dist のビルドシステムでは config、menuconfig、xconfig が使用できます。ここでは menuconfig を使って uClinux-dist のビルドを行います

まず、「7.3.ソースコードの準備」で展開した uClinux-dist ディレクトリに移動して `make menuconfig` とコマンドを入力してください。

例 8-1 make menuconfig の実行

```
[PC ~]$ cd uClinux-dist
[PC ~/uClinux-dist]$ make menuconfig
```



`make menuconfig` では最初の実行時に ncurses ライブラリを必要とする画面制御プログラムをコンパイルします。このため ncurses ライブラリがインストールされている必要があります。OS によっては ncurses ライブラリパッケージだけではなく、同ライブラリの開発用パッケージをインストールする必要があります。

8.2.1. メインメニュー

コマンドを入力すると、下の図のように画面が表示されます。この画面が uClinux-dist のメインメニューです。

図 8-2 uClinux v3.1.0 Configuration Main Menu

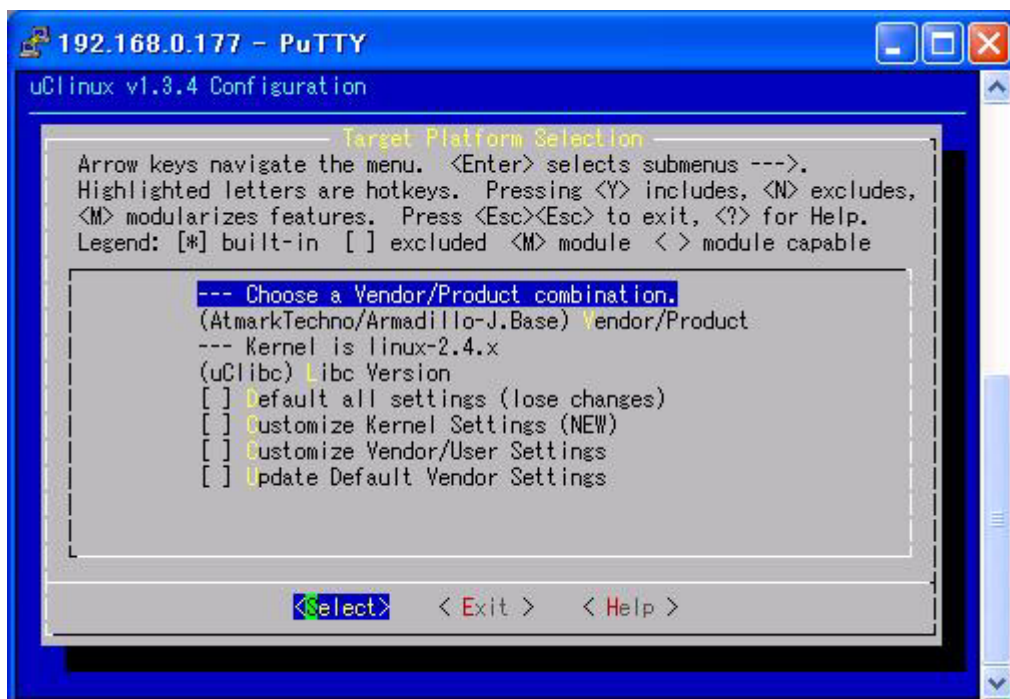


menuconfig ではキーボードのカーソルキーを使って選択メニューの移動を行います。選択メニューの右側が「--->」となっている場所はサブメニューへ移動できることを表わします。メニュー選択時には画面下の「<select>」がハイライトされていることを確認してからエンターキーを押します。

8.2.2. Target Platform Selection メニュー

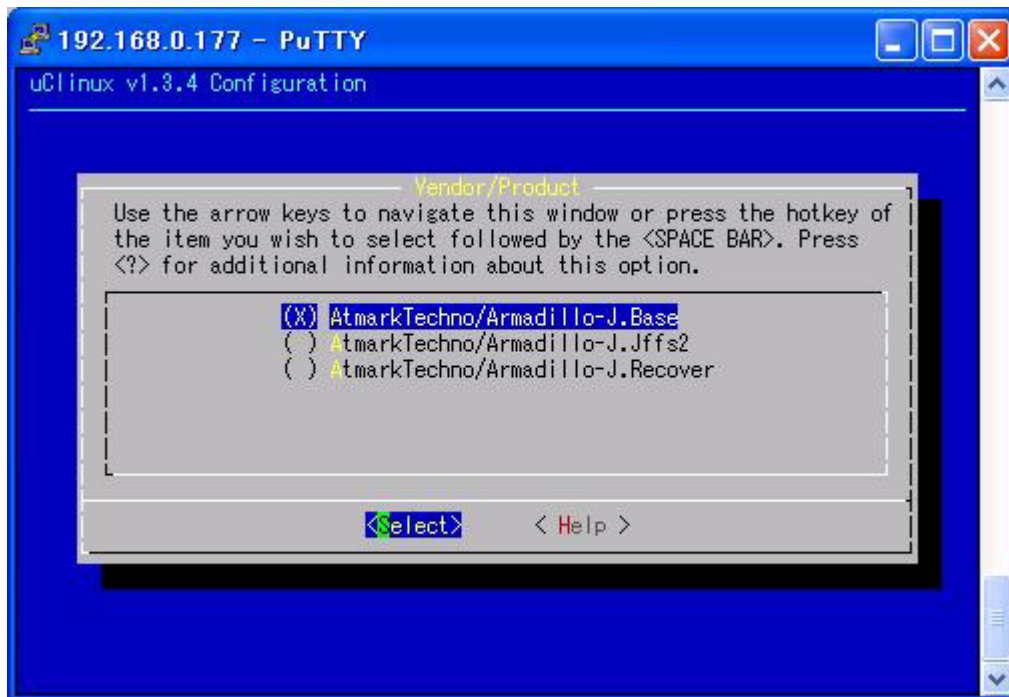
メインメニューから「Target Platform Selection」を選択して、サブメニュー画面へ移動します。

図 8-3 Target Platform Selection



「(AtmarkTechno/Armadillo-J.Recover) Vendor/Product」にカーソルをあわせてエンターキーを押すことで他のイメージを選択できます。今回は Linux 端末として動作するベースイメージを作成します。リストの中から Armadillo-J.Base を選択してください。

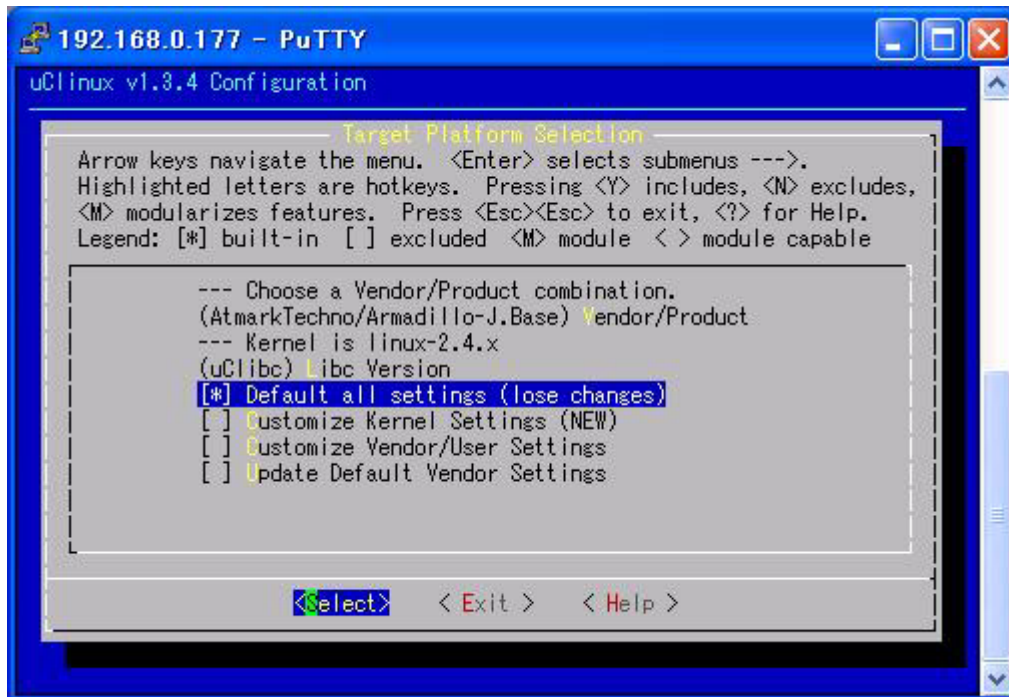
図 8-4 Target Platform Selection



カーソルキーで<Exit>を選択して、前の画面に戻ります

次に「Default all settings (lose changes) (NEW)」を選択してください。メニューの左側にある[]は選択状態を表しています。選択するにはカーソルを上下キーで動かし、スペースバーで選択します。選択すると[]の中に「*」が表示されます。

図 8-5 Default all setting の選択



「Default all settings (lose changes)」を選択後、<Exit>を選択しメインメニューに戻ります。メインメニューでExitを選択すると、設定を保存するか聞かれますので<Yes>を選択して保存します。

画面上に設定のログが流れた後、プロンプトに戻ります

8.3. ビルド

実際にコンパイルを行ったり、イメージファイルを生成することをビルドと言います。ビルドにはmakeを使用します。下の例のようにプロンプトで入力します

例 8-2 ビルドコマンド

```
[PC ~/uClinux-dist]$ make dep; make
```

最初のmake depはLinux Kernelの依存関係を解決するためのコマンドです。2.4系までのLinux Kernelコンパイル経験者には見なれたコマンドだと思います。

次のmakeコマンドは全てのビルドプロセスを行ない、最終的にFlashメモリに書き込むことのできるイメージファイルを生成してくれます。ビルドプロセスが問題なく終了すると、uClinux-dist/images/ディレクトリにimage.binというファイルが生成されます。

実際に生成されたイメージファイルをArmadillo-Jに書き込む方法は「5.Flashメモリの書き換え方法」を参照してください。

ここでのイメージファイルとは、uClinuxカーネルとユーザランドが一つにまとまっているバイナリファイルを意味します。このバイナリファイルをHermitでFlashメモリに書き込むことにより、Armadillo-J上でuClinuxが起動します。

9. uClinux の image を作成

この章では uClinux-dist のメニューを使用して、カスタマイズしたイメージの作成方法を説明します。

9.1. カーネルの設定変更

カーネルのカスタマイズを行なう場合は、uClinux Configuration の Target Platform Selection メニューで「Customize Kernel Settings」を選択します。このメニューを選択し uClinux v1.3.4 Configuration のメインメニューを終了させると、自動的に Linux Kernel Configuration のメニュー画面が表示されます。Linux Kernel Configuration 自体は本家 Linux と同じものです。

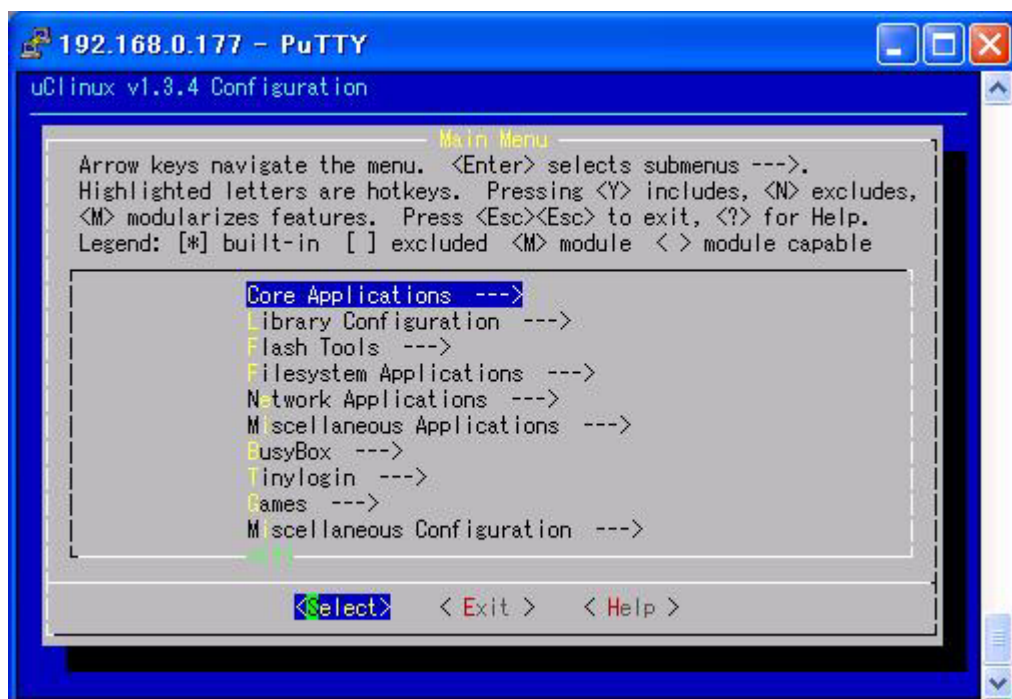
カーネルをカスタマイズ後、<Exit>を選んでください。最後に設定を保存するか聞かれますので<Yes>を選択してください。

ユーザランドのカスタマイズが無い場合は、ビルドを行ないます。ビルドについては「8.3.ビルド」を参照してください。

9.2. ユーザランドの設定変更

ユーザランドのカスタマイズを行なう場合は、uClinux Configuration の Target Platform Selection メニューで「Customize Vendor/User Settings」を選択します。このメニューを選択し uClinux v1.3.4 Configuration のメインメニューを終了させるとユーザランド用の Main Menu になります。

図 9-1 ユーザランド用 Main Menu



ユーザランドのカスタマイズ画面では、uClinux Full Source Distribution で配布されているアプリケーションを生成するイメージに追加するか否か、または追加するアプリケーションのカスタマイズを行ないます。

uClinux-dist には 150 以上ものアプリケーションが集められています。このため、uClinux Configuration のメニューではアプリケーションをいくつかのカテゴリに分類しています。

- Core Application

システムとして動作するために必要な基本的なアプリケーションが入っています。システムの初期化を行なう `init` やユーザ認証の `login` などがこのセクションで選択できます。

- Library Configuration

アプリケーションが必要とするライブラリの選択ができます。

- Flash Tools

Flash メモリに関係のあるアプリケーションが選択できます。Armadillo-J.Base では、`netflash` と呼ばれるネットワークアップデート用アプリケーションがここで選択されています。

- Filesystem Applications

ファイルシステムに関係のあるアプリケーションが選択できます。Armadillo-J.Base では `flatfsd` を選択しています。その他、`mount`、`fdisk`、`ext2` ファイルシステム、`reiser` ファイルシステム、`Samba` などが含まれます。

- Network Applications

ネットワークに関係のあるアプリケーションが選択できます。Armadillo-J.Base で使用している `dhcpcd-new`、`ftpd`、`ifconfig`、`inetd`、`thttpd` の他にも `ppp` やワイヤレスネットワークのユーティリティなども含まれます。

- Miscellaneous Applications

上記のカテゴリに属さないアプリケーションが収められています。Unix の一般的なコマンド(`cp`、`ls`、`rm` 等)やエディタ、オーディオ関連、スクリプト言語インタプリタなどが含まれます。

- Busybox

Busybox のカスタマイズを行ないます。Busybox は複数のコマンド機能をもった単一コマンドで多くの組み込み Linux での実績を持っています。Busybox はとても多くカスタマイズできるため、別セクションになっています。

- Tinylogin

Tinylogin も複数コマンドの機能をもつアプリケーションです。`login` や `passwd`、`getty` など認証に関係のある機能を提供します。多くのカスタマイズが可能のため別セクションになっています。

- MicroWindows

MicroWindows は組み込み機器をターゲットにしたグラフィカルウインド環境です。LCD などを持つ機器を開発する場合に使えます。

- Game

ゲームです。説明はいらないでしょう。

- Miscellaneous Configuration

いろいろな設定がまとめられています。

- Debug Builds

デバッグ用のオプションがまとめられています。開発中にアプリケーションのデバッグを行なうときに選択します。

すべてのアプリケーションが動作することは保証されていませんが、多くの場合ほんのわずかな変更(ソースコードレベルであったり、Makefileの変更だったり)で動作させることが可能です。

10. 自作アプリケーションの作成

Hello World アプリケーションを作成して Armadillo-J 上で動かしてみます。コンパイルは uClinux-dist のパッケージの外(これを OTC: Out of Tree Compile と呼びます)で行います。

uClinux-dist の外でコンパイルする場合でも、uClinux-dist で提供されているライブラリや Makefile を使用しますので、一度 Armadillo-J 用にビルドした uClinux-dist ディレクトリが必要です。まだ一度もビルドしていない場合は「[8.uClinux の image を作成](#)」を参照してビルドを行ってください。

Armadillo-J で自作アプリケーションを動作させる際には次のことに注意して下さい。



I/O ポートやシリアルポートの制御を行うアプリケーションを開発する場合、ジャンパピンの設定、I/O ポート関連のレジスタ、及び I/O ポートへ接続する機器を適切に設定して下さい。不正な設定で Armadillo-J を動作させた場合、誤動作・機器の故障につながる場合があります。ジャンパピンや I/O ポート関連のレジスタについては「[Hardware manual](#)」に記載されています。

10.1. ディレクトリの作成

実際に作業を行なう場所はどこでもかまいません。ここでは~/helloを使用します。

例 10-1 Hello World 用のディレクトリを用意する

```
[PC ~]$ mkdir hello
[PC ~]$
```

10.2. Makefile の作成

Makefile のサンプルテンプレートが付属 CDROM の `sample/hello/Makefile` にありますので、このファイルをコピーします。

例 10-2 Hello World Makefile

```

ifndef ROOTDIR
ROOTDIR=../uClinux-dist ----- ①
endif
ROMFSDIR = $(ROOTDIR)/romfs
ROMFSINST = romfs-inst.sh
PATH      := $(PATH) :$(ROOTDIR)/tools

UCLINUX_BUILD_USER = 1 ----- ②
include $(ROOTDIR)/.config
LIBCDIR = $(CONFIG_LIBCDIR)
include $(ROOTDIR)/config.arch

EXEC = hello ----- ③
OBJS = hello.o ----- ④

all: $(EXEC)

$(EXEC): $(OBJS)
        $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LDLIBS)

clean:
        -rm -f $(EXEC) *.elf *.gdb *.o

romfs:
        $(ROMFSINST) /bin/$(EXEC)

%.o: %.c
        $(CC) -c $(CFLAGS) -o $@ $< ----- ⑤
    
```

- ① `ROOTDIR` が指定されていない場合、現在のディレクトリと並列に `uClinux-dist` ディレクトリがあると仮定します。他の場所の場合は `ROOTDIR` を指定してください。
- ② `UCLINUX_BUILD_USER` を定義します。この変数を定義することで、ユーザランド用のコンパイラオプションが選ばれます。
- ③ 生成される実行ファイル名を変数 `EXEC` に設定します。ここでは `hello` とします。
- ④ 生成する実行ファイルに使用するオブジェクトファイルを指定します。複数ある場合はスペースで区切って羅列します。
- ⑤ C ソースコードを同名のオブジェクトコードに変換するパターンルールです。詳しくは `Make` のマニュアルをご覧ください。

10.3. C ソースコード

Hello World の C コードを以下に示します。

例 10-3 Hello World ソースコード

```
#include <stdio.h>

int main(int argc, char * argv[])
{
    printf("Hello World!%n");
    return 0;
}
```

どの C の教科書にでも載っている、普通の Hello World です。これを、`hello.c` として保存します。Makefile に従い `hello.o` にコンパイルされます。

10.4. コンパイル

以下のように `make` します。`make` に成功すると `hello.o`, `hello.gdb`, `hello` の 3 つのファイルが生成されます。`hello` が実行ファイルです。

例 10-4 make の実行

```
[PC ~/hello]$ make
[PC ~/hello]$ ls
Makefile hello hello.c hello.gdb hello.o
[PC ~/hello]$
```

10.5. ROMFS ディレクトリへのインストール

イメージファイルのユーザ領域は、`uClinux-dist/romfs` ディレクトリから作成されます。そのため自作のアプリケーションを Flash メモリに書き込むイメージに含めるためには、アプリケーションを `romfs` ディレクトリ内にコピーする必要があります。

今回のように生成される実行ファイルが一つであればコピーするのも手間ではありませんが、複数の実行ファイルを生成したり、設定ファイルやデータファイルもコピーするときは大変です。このため、Makefile のテンプレートでは `romfs` ターゲットを持っています。

例 10-5 Hello World を ROMFS ディレクトリへインストール

```
[PC ~/hello]$ make romfs
romfs-inst.sh /bin/hello
[PC ~/hello]$ ls ../uClinux-dist/romfs/bin/hello
../uClinux-dist/romfs/bin/hello
[PC ~/hello]$
```

10.6. イメージファイルの生成と実行

最後に `uClinux-dist` のディレクトリに移動して、イメージファイルを生成します。

`make` コマンドで `image` ターゲットを指定すると、`romfs` ディレクトリを元にユーザランドのイメージファイル(`romfs.img`)を作成した後、カーネルのイメージファイル(`linux.bin`)と結合して `Flash` メモリに書き込むイメージファイル(`image.bin`)を生成します。

例 10-6 イメージファイル生成

```
[PC ~/hello]$ cd ../uClinux-dist
[PC ~/uClinux-dist]$ make image
:
:
:
[PC ~/uClinux-dist]$ ls images
image.bin linux.bin romfs.img
[PC ~/uClinux-dist]$
```

生成されたイメージファイルを `Armadillo-J` にダウンロードし、`hello` を実行します。ダウンロードの方法については「[5.Flash メモリの書き換え方法](#)」を参照してください。

例 10-7 実行

```
[AJ ~]# hello
Hello World!
[AJ ~]#
```


11. Flat Binary Format

この章では uClinux の特徴の一つである Flat Binary Format について説明します。uClinux が対象としている組み込み機器では実行バイナリーの大きさは大きな問題です。一般の Linux が採用している ELF は柔軟性に富んだフォーマットですがサイズが大きすぎる問題がありました。そこで多くの uClinux では昔ながらの a.out フォーマットに似た新たなバイナリーフォーマットを採用しています。

はじめに Flat Binary Format の特徴を説明した後、実行ファイルの圧縮方法とスタックサイズの変更方法を説明します。

11.1. Flat Binary Format の特徴

Flat Binary Format には以下のような特徴があります。

- シンプル
シンプルな設計はバイナリファイルの実行速度と大きさに貢献しています。もちろん ELF のような柔軟性は減りますが、組み込み機器にとっては必要なトレードオフと言えます。
- 圧縮可能
Flat Binary Format は圧縮可能なフォーマットです。圧縮には 2 種類あり、ファイル全体の圧縮とデータ領域のみの圧縮が可能です。実行ファイルはロードされる時に伸長されるため、起動速度は非圧縮の実行ファイルに比べ遅くなります。もちろん起動してしまえば非圧縮の実行ファイルとの差はなくなってしまうので、常駐プロセスのように起動停止を繰り返さないプログラムには適しています。
- スタックサイズフィールド
再コンパイルせずに変更可能なスタックサイズのフィールドを持っています。MMU を持たない CPU では動的にスタック領域を拡張することが難しいため、固定サイズのスタック領域を持ちます。このフィールドは flthdr と呼ばれるツールで変更することが可能です。また、コンパイル時にサイズを指定することも可能です。
- XIP (eXecute In Place)対応
Flat Binary Format は XIP にも対応しています。XIP とは eXecute In Place(その場実行)の略で、一般的には RAM に実行バイナリーをコピーすることなく、保存されている ROM 上で実行する機能のことです。

11.2. 実行ファイルの圧縮

例として前章で作成した Hello World プログラムを圧縮します。後半ではコンパイル時に圧縮を指定する方法を説明します。

11.2.1. コンパイル済バイナリファイルを圧縮

flthdr を使ってコンパイル済のバイナリファイルを圧縮します。flthdr は Flat Binary Format のファイルを編集・表示するプログラムです。

通常のコンパイルで生成された実行ファイルでは以下ようになります。

例 11-1 通常の Flat Binary Format

```
[PC ~/hello]$ make
[PC ~/hello]$ flthdr hello
hello
  Magic:      bFLT
  Rev:        4
  Entry:      0x50
  Data Start: 0x3e60
  Data End:   0x4bf0
  BSS End:    0x6bf0
  Stack Size: 0x1000
  Reloc Start: 0x4bf0
  Reloc Count: 0x53
  Flags:      0x1 ( Load-to-Ram )
[PC ~/hello]$
```

次に flthdr で圧縮してみます。

例 11-2 圧縮された Flat binary Format

```
[PC ~/hello]$ flthdr -z hello ①
zflat hello --> hello
[PC ~/hello]$ flthdr hello ②
hello
  Magic:      bFLT
  Rev:        4
  Entry:      0x50
  Data Start: 0x3e60
  Data End:   0x4bf0
  BSS End:    0x6bf0
  Stack Size: 0x1000
  Reloc Start: 0x4bf0
  Reloc Count: 0x53
  Flags:      0x5 ( Load-to-Ram Gzip-Compressed ) ③
[PC ~/hello]$
```

- ① flthdr に圧縮オプション'-z'を渡す
- ② flthdr コマンドで生成された実行ファイルのヘッダ部を表示させる
- ③ Gzip-Compressed フラグが確認できる

11.2.2. コンパイル時に圧縮

コンパイル時に圧縮するには、FLTFLAGS 環境変数を使います。以下に Hello World での例を示します。

例 11-3 FLTFLAGS による圧縮の指定

```
[PC ~/hello]$ make FLTFLAGS=-z ----- ①
[PC ~/hello]$ flthdr hello ----- ②
hello
  Magic:      bFLT
  Rev:        4
  Entry:      0x50
  Data Start: 0x3e60
  Data End:   0x4bf0
  BSS End:    0x6bf0
  Stack Size: 0x1000
  Reloc Start: 0x4bf0
  Reloc Count: 0x53
  Flags:      0x5 ( Load-to-Ram Gzip-Compressed ) ----- ③
[PC ~/hello]$
```

- ① FLTFLAGS 環境変数に '-z' をセットして make を実行する
- ② flthdr コマンドで生成された実行ファイルのヘッダ部を表示させる
- ③ Gzip-Compressed フラグが確認できる

11.3. スタックサイズの指定

スタックサイズを指定する方法を 2 種類紹介します。

11.3.1. コンパイル済バイナリファイルスタックサイズを変更

flthdr の -s でスタックサイズを指定します。アーキテクチャによりませんが、スタックサイズのデフォルト値は 4096 が多いです。

例 11-4 スタックサイズの変更

```
[PC ~/hello]$ flthdr -s 8192 ----- ①
[PC ~/hello]$ flthdr hello ----- ②
hello
  Magic:      bFLT
  Rev:        4
  Entry:      0x50
  Data Start: 0x3e60
  Data End:   0x4bf0
  BSS End:    0x6bf0
  Stack Size: 0x2000 ----- ③
  Reloc Start: 0x4bf0
  Reloc Count: 0x53
  Flags:      0x1 ( Load-to-Ram )
[PC ~/hello]$
```

- ① flthdr にスタックサイズ変更オプション '-s' とスタックサイズを渡す
- ② flthdr コマンドで生成された実行ファイルのヘッダ部を表示させる
- ③ 8192byte に変更されているのが確認できる

11.3.2. コンパイル時にスタックサイズを指定

コンパイル時にスタックサイズを指定するには、FLTFLAGS 環境変数を使います。以下に Hello World での例

を示します。

例 11-5 FLTFLAGS によるスタックサイズの指定

```
[PC ~/hello]$ make FLTFLAGS='-s 8192' ----- ①
[PC ~/hello]$ flthdr hello ----- ②
hello
  Magic:      bFLT
  Rev:        4
  Entry:      0x50
  Data Start: 0x3e60
  Data End:   0x4bf0
  BSS End:    0x6bf0
  Stack Size: 0x2000 ----- ③
  Reloc Start: 0x4bf0
  Reloc Count: 0x53
  Flags:      0x1 ( Load-to-Ram )
[PC ~/hello]$
```

- ① FLTFLAGS 環境変数に '-s 8192' をセットして make を実行する
- ② flthdr コマンドで生成された実行ファイルのヘッダ部を表示させる
- ③ 8192byte に変更されたスタックが確認できる

12. トラブルシューティング

12.1. 正常に起動しない場合

誤ったイメージファイルなどを Flash メモリに書き込み正常に起動しなくなった場合、付属 CD に収納されているイメージ「image/base.img」を「5.Flash メモリの書き換え方法」を参照して Flash メモリへ書き込んで下さい。

12.2. 購入時の状態に戻すには

付属 CD に収納されているイメージ「image/recover.img」を Armadillo-J に書き込むことで、購入時の状態に戻すことができます。「5.Flash メモリの書き換え方法」を参照して Flash への書き込みを行って下さい。

12.3. ブートローダ(hermit)の更新方法



ブートローダの更新に失敗した場合、恒久的に Aramdillo-J が起動しなくなる可能性があります。必然性がある場合のみこの作業を行って下さい。通常の使用においてブートローダを更新する必要はありません。

Armadillo-J に搭載されているブートローダの書き換えには、v1.3-armadillo-6 以降のバージョンの hermit が必要です。バージョンが古い場合は最新版をダウンロードして下さい。また、Windows 版の hermit 「Hermit host for Win32」でブートローダを更新することはできません。

hermit.bin というブートローダに入れ替える例を以下の図に示します。hermit.bin は購入状態の Aramadillo-J に書き込まれているブートローダです。付属 CD の「image」ディレクトリにあります。

```
[PC ~]# hermit firmupdate -i hermit.bin
```

コマンド
ファイル名

図 12-1 ブートローダの更新手順

12.4. ブートローダ(hermit)の復旧方法



ブートローダの復旧に失敗した場合、恒久的に Aramdillo-J が起動しなくなる可能性があります。必然性がある場合のみ作業を行ってください。通常の使用においてブートローダが壊れることはありません。

ブートローダの更新に失敗した等、何らかの理由で Flash メモリのブートローダの領域に不正なデータが書き込まれてしまい、ブートローダ(hermit)が起動しなくなった場合の復旧方法をです。

ブートローダが正常に起動しない場合、Flash メモリの書き込み不可能な領域に記録されている NS7520 XK BootLoader が起動メニューを表示します。ここでは、起動メニューから tftp を使用して、Hermit を復旧させます。復旧には「TFTP サーバ」と、付属 CD の「image/hermitXK.bin」が必要になるので準備して

下さい。

はじめに Armadillo-J と作業用 PC をシリアルケーブルで接続し、シリアルコンソールソフトを起動します。次のように通信設定を行って下さい。

表 12-1 シリアル通信設定

項目	設定
転送レート	9,600bps
データ長	8bit
ストップビット	1bit
パリティ	なし
フロー制御	なし

1) XK BootLoader メニュー表示

JP4 をショートさせて Armadillo-J の電源を入れると、「XK BootLoader」のメニューが表示されます。

```

NS7520 XK BootLoader  Compiled on Jun 25 2003 at 11:45:31

----- Control Routines -----

V) display all VPD.                W) test Watchdog.
b) dump bcomm.                    c) call POST function.
d) dump memory - prompt addr.     D) dump memory - next block.
e) edit memory.                   R) dump registers ASIC.
f) erase VPD.                      g) program VPD.
h) erase POST.                     i) program POST.
j) erase EOS.                       k) program EOS.
l) erase Config_FFS.               m) program Config_FFS.
n) erase Web_FFS.                  o) program Web_FFS.
p) erase BOOTPARAM.               X) TFTP menu.
x) TFTP private network.

-----

Enter choice (ESC to exit-Control Routines) [VWbcdDeRfghijklmnopXx] :
    
```

図 12-2 NS7520 XK BootLoader のメニュー

2) サブメニュー表示

機能画面より「X (TFTP menu.)」を選択すると「TFTP Related Options」が表示され、「2. Enter TFTP settings.」を指定します。

```

----- TFTP Related Options -----

1) Show current TFTP settings.    2) Enter TFTP settings.
3) Get file from TFTP server.

-----

Enter choice (ESC to exit-TFTP Related Options) [123] :2
    
```

図 12-3 TFTP メニューの表示

3) tftp の設定

tftp 転送するファイル、Armadillo-J のネットワークの設定、TFTP サーバの IP アドレスを指定します。

```
Enter TFTP filename []: hermitXK.bin

Enter IP address [0.0.0.0]: 192.168.10.21

Enter network mask [0.0.0.0]: 255.255.255.0

Enter network gateway [0.0.0.0]: (省略可能)

Enter network TFTP server [0.0.0.0]: 192.168.10.80
```

図 12-4 TFTP setting での入力例

- **TFTP filename**
転送するファイルを指定します。
(例 hermitXK.bin)
- **IP address**
Armadillo-J の IP アドレスを指定します。
(例 192.168.10.21)
- **network mask**
ネットワークマスクを指定します。
(例 255.255.255.0)
- **network gateway**
ゲートウェイアドレスを指定します。省略が可能です。
(例 0.0.0.0)
- **network TFTP server**
ファイルが存在する TFTP サーバーの IP アドレスを指定します。
(例 192.168.10.80)

4) ファイル取得

「3. Get file from TFTP server.」を選択し、TFTP サーバからファイルを取得します。
ファイル取得の完了後、「ESC」キーを押下、[図 12-2](#) のメニューに戻ります。

5) ファイル書き込み

「h. erase POST」、続いて「i. program POST」を実行して、取得したデータを Armadillo-J に書き込みます。

6) 書き込み完了

書き込み完了すると、次回起動時には hermit が復旧しています。

13. Appendix

13.1. Windows 上に開発環境を構築する方法

Linux 環境を実現する coLinux(<http://www.colinux.org/>)を利用することで、Windows 上に Armadillo-J 用のクロス開発環境を構築することができます。対応している OS は WindowsXP、Windows2000 です。

13.1.1. coLinux のインストール

- 1) 付属 CD の colinux ディレクトリにある `coLinux-0.6.1.exe` を実行する
- 2) インストール先フォルダには `c:\¥colinux` を指定し、それ以外はデフォルトの設定のままでインストール作業を行う

※インストール先に他のディレクトリを指定する場合は、次の手順で用意する `default.colinux.xml` を編集し、ディレクトリ名を適切に変更する必要があります。

13.1.2. 環境構築用ファイルの準備

付属 CD の colinux ディレクトリから以下のファイルを用意し、coLinux のインストールフォルダ (`c:\¥colinux`)に解凍します

- `root_fs.lzh` (ルートファイルシステム)
- `swap_device_256M.lzh` (swap ファイルシステム)
- `home_fs_2G.lzh` (/home にマウントされるファイルシステム)
- `default.colinux.xml.lzh` (デバイス情報の設定ファイル)

※ `swap_device_...`, `home_fs_...` のファイル名の数値は解凍後のファイルサイズです。他のサイズのファイルも用意してますので、必要と思われるサイズのファイルを解凍してください。

※ 解凍ソフトによっては解凍に失敗する場合があります。「LHA ユーティリティ 32 Ver1.46(<http://www.lhut32.com/index.shtml>)」にて正常に解凍できることを確認してあります。

13.1.3. coLinux の実行

- 1) DOS プロンプトを起動し、インストールフォルダ(`c:\¥colinux`)に移動する
- 2) 「`colinux-daemon.exe -c default.colinux.xml`」とコマンド入力する
- 3) 起動ログの表示後「`colinux login:`」と表示されたら「`root`」でログインする

13.1.4. ネットワークの設定

coLinux は Windows と別の IP アドレスを持ち、Windows を介してネットワークにアクセスするため、Windows のネットワーク設定の変更が必要となります。

設定方法には「ルーター接続」「ブリッジ接続」などがありますが、ここでは「ルーター接続」の方法を説明します。

(WindowsXP の場合)

- 1) コントロールパネルから「ネットワーク接続」を開く
- 2) 外部に接続しているネットワークを右クリックして「プロパティ」を開く
- 3) 「詳細設定」タブを開き、インターネット接続の共有を有効にする

(Windows2000 の場合)

- 1) コントロールパネルから「ネットワークとダイヤルアップ接続」を開く
- 2) 外部に接続しているネットワークを右クリックして「プロパティ」を開く
- 3) 「共有」タブを開き、インターネット接続の共有を有効にする

次に、ネットワークの設定を有効にするためのコマンドを coLinux 上で実行します。

例 13-1 ネットワークの設定コマンド

```
colinux:~# /etc/init.d/networking restart
Reconfiguring network interfaces: done.
colinux:~#
```



「ルーター接続」では 192.168.0.0/24 のネットワークアドレスが自動的に使用されるため、外部接続用のネットワークアドレスが同じ 192.168.0.0/24 の場合、設定に失敗します。この場合は外部接続用のネットワークアドレスを変更して下さい。

外部接続用のネットワークアドレスを変更できない場合は「[13.1.8.特殊な場合の Windows ネットワーク設定方法](#)」を参照して下さい。

13.1.5. coLinux ユーザの作成

coLinux の画面で以下のようにコマンドを入力し作業用ユーザを作成します。適宜パスワードなどを設定して下さい。

例 13-2 ユーザ「somebody」を作業用ユーザとして追加する場合

```
colinux:~# adduser somebody
Adding user somebody...
Adding new group somebody (1000).
Adding new user somebody (1000) with group somebody.
Creating home directory /home/somebody.
Copying files from /etc/skel
Enter new UNIX password:
```

13.1.6. Windows-coLinux 間のファイル共有

Windows の共有フォルダを利用して、coLinux と windows 間でファイルを交換する方法です。coLinux の画面で以下のように smbmount コマンドを実行して、共有フォルダのパスワードを入力して下さい。

例 13-3 Windows の IP アドレス: 192.168.0.100、共有フォルダ名: shared の場合

```
colinux:~# mkdir /mnt/smb
colinux:~# smbmount //192.168.0.100/shared /mnt/smb
212: session request to 192.168.0.100 failed (Called name not present)
212: session request to 192 failed (Called name not present)
Password:
```

ユーザ名が Windows 側と異なる場合は、ユーザ名をコマンドのオプションで指定します。詳しくは「man smbmount」を実行してヘルプを参照して下さい。

以後、windows の共有フォルダ”shared”と coLinux のディレクトリ”/mnt/smb” のデータは同じものになります。

13.1.7. クロス開発環境の導入

「7.開発環境の準備」を参照して、クロス開発環境を coLinux 上に導入して下さい。

環境の構築に必要なファイルは、前項で使用した共有フォルダを通じて coLinux からアクセスできます。

これで Windows から Armadillo-J の開発を行うことができます。以降の説明は特殊なケースです。

13.1.8. 特殊な場合の Windows ネットワーク設定方法

外部接続用のネットワークアドレスが 192.168.0.0/24 の場合のネットワーク設定方法です。

(WindowsXP の場合)

「ブリッジ接続」を利用する方法です。

- 1) コントロールパネルから「ネットワーク接続」を開く
- 2) 外部に接続しているネットワークと「TAP-Win32 adapter」というデバイス名のネットワークの二つを選択状態にする
- 3) メニューの「詳細設定」から「ブリッジ接続」を選択する

(Windows2000 の場合)

Windows2000 では 192.168.0.0/24 以外のネットワークアドレスをプライベートネットワークで使用する場合があります。ここでは 192.168.1.0/24 を使用します。

- 1) コントロールパネルから「ネットワークとダイヤルアップ接続」を開く
- 2) 外部に接続しているネットワークを右クリックして「無効」にする
- 3) 外部に接続しているネットワークを右クリックして「プロパティ」を開く
- 4) 「全般」タブの「インターネットプロトコル(TCP/IP)」を選択し「プロパティ」ボタンを押す
- 5) 「次の IP アドレスを使う」を選択して 192.168.100.100 を設定する
- 6) 「共有」タブを開き、インターネット接続の共有を有効にする
- 7) 「TAP-Win32 adapter」というデバイス名のネットワークを右クリックして「プロパティ」を開く
- 8) 「全般」タブの「インターネットプロトコル(TCP/IP)」を選択し「プロパティ」ボタンを押す
- 9) 「次の IP アドレスを使う」を選択して 192.168.1.1 を設定する
- 10) 外部に接続しているネットワークを右クリックして「プロパティ」を開く
- 11) 「全般」タブの「インターネットプロトコル(TCP/IP)」を選択し「プロパティ」ボタンを押す
- 12) IP アドレスの設定を元の設定に戻す
- 13) 外部に接続しているネットワークを右クリックして「有効」にする

13.1.9. coLinux のネットワーク設定方法

インストール状態では DHCP が使用されますが、DHCP サーバが動作していない環境等では固定で IP アドレスを設定する必要があります。

ネットワーク設定は ifconfig コマンドで表示することができます。

例 13-4 ifconfig コマンドの実行

```
colinux:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:43:4F:4E:45:30
          inet addr:192.168.0.151  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:189 errors:0 dropped:0 overruns:0 frame:0
          TX packets:115 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:24472 (23.8 KiB)  TX bytes:9776 (9.5 KiB)
          Interrupt:2

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

colinux:~#
```

eth0 デバイスの IP アドレスが表示されない場合は、固定で IP アドレスを設定する必要があります。設定すべき IP アドレスですが、「ルーター接続」場合「TAP-Win32 adapter」のネットワークに合わせ、「ブリッジ接続」の場合は外部ネットワークに合わせます。ここでは、以下の表の内容に設定を変更する方法を説明します。

表 13-1 ネットワーク設定

項目	設定
IP アドレス	192.168.1.100
ネットマスク	255.255.255.0
ゲートウェイ	192.168.1.1
DNS サーバ	192.168.1.1

- 1) coLinux 上で/etc/network/interfaces を以下のように編集する

例 13-5 /etc/network/interfaces ファイル編集例

```
auto lo eth0
iface lo inet loopback
iface eth0 inet static
    address 192.168.1.100
    gateway 192.168.1.1
    netmask 255.255.255.0
```

- 2) coLinux 上で/etc/resolv.conf を以下のように編集する

例 13-6 /etc/resolv.conf ファイル編集例

```
nameserver 192.168.1.1
```

- 3) 以下のコマンドを実行し、編集した内容でネットワーク設定を更新する

例 13-7 ネットワークの再設定コマンド

```
colinux:~# /etc/init.d/networking restart  
Reconfiguring network interfaces: done.  
colinux:~#
```

13.2. ベースイメージについて

13.2.1. ベースイメージのメモリマップ

表 13-2 メモリマップ(Flash メモリ)

アドレス	Flash メモリの内容	サイズ	説明
0x02000000 0x0201ffff	Reserved (書き換え不可)	128kB	
0x02020000 0x0203ffff	ブートローダ (hermit)	128kB	
0x02040000 0x021effff	kernel ユーザーランド	約 1.7MB	「base.img」のイメージ
0x021f0000 0x021fffff	/etc/config (書き込み可能領域)	64kB	

※ kernel とユーザーランドのみ、uClinux の起動前に RAM へコピーされる

表 13-3 メモリマップ(RAM)

アドレス	RAM の内容	ファイルシステム	説明
0x08000000	kernel		「base.img」のイメージ
	ユーザーランド	romfs	uClinux の起動前に Flash メモリからコピー
	/var	ramfs	
	/etc/config	ramfs	
	/home/guest/pub	ramfs	

13.2.2. コマンド一覧

表 13-4 コマンド一覧

コマンド名	概要
addgroup	グループを追加する
adduser	ユーザを追加する
cat	ファイルを連結して出力する
chgrp	ファイルのグループ所有権を変更する
chmod	ファイルのアクセス権を変更する
chown	ファイルの所有者とグループを変更する
cp	ファイルのコピーをする
delgroup	グループを削除する
deluser	ユーザを削除する
echo	1行のテキストを表示する
false	何もせずに終了ステータスとして”失敗”を意味する 1 を返す
flatfsd	flat ファイルシステムデーモン
ftpd	ftp デーモン
inetd	inet デーモン
kill	プロセスにシグナルを送る
ls	ディレクトリリストを表示する
mkdir	ディレクトリを作成する
more	ファイルを閲覧するためのフィルター
mount	ファイルシステムをマウントする
netflash	ネットワーク経由でオンボードフラッシュイメージを更新する
passwd	パスワードを変更する
ping	ICMP ECHO_REQUEST パケットをネットワーク上のホストに送る
ps	プロセスの状態を表示する
route	IP 経路テーブルの表示/設定
sh	シェル
su	スーパーユーザ権限を取得する
sulogin	シングルモードでログインを行う
telnetd	telnet デーモン
test	ファイル形式のチェックや値の比較を行う
thttpd	Web サーバ機能を提供するデーモン
tinylogin	ログインやユーザ管理のためのツール群
true	何もせずに終了ステータスとして”成功”を意味する 0 を返す
umount	ファイルシステムをアンマウントする
watchdog	watchdog デーモン
vi	テキストエディタ

13.2.3. 起動デーモン一覧

表 13-5 起動デーモン一覧

起動デーモン	概要
flatfsd	Flash メモリヘデータを保存する
inetd	各種ネットワークサービス(telnet、ftp 等)のインタフェースを提供する
thttpd	Web サーバ機能

13.3. オリジナルデバイスドライバ仕様

13.3.1. パラレルポートドライバ

パラレルポート(CON2)に対応するデバイスノードのパラメータは、以下の通りです。

表 13-6 パラレルポートノード一覧

タイプ	メジャー番号	マイナー番号	ノード名 (/dev/xxx)	デバイス名
キャラクタデバイス	210	0	padr0	Port A Data Register CH0 (Pin. 7)
		1	padr1	Port A Data Register CH1 (Pin. 5)
		2	padr2	Port A Data Register CH2 (Pin. 6)
		5	padr5	Port A Data Register CH5 (Pin. 3)
		6	padr6	Port A Data Register CH6 (Pin. 4)
		8	padr	Port A Data Register 全 CH(8bit)
		16	paddr0	Port A Data Direction Register CH0 (Pin. 7)
		17	paddr1	Port A Data Direction Register CH1 (Pin. 5)
		18	paddr2	Port A Data Direction Register CH2 (Pin. 6)
		21	paddr5	Port A Data Direction Register CH5 (Pin. 3)
		22	paddr6	Port A Data Direction Register CH6 (Pin. 4)

- データ型

padr0, 1, 2, 5, 6 (各 CH): unsigned char(符号なし 8bit) 0x00 / 0x01
 padr (全 CH): unsigned char(符号なし 8bit) 0x00~0xff
 paddr0, 1, 2, 5, 6 (各 CH): unsigned char(符号なし 8bit) 0x00 / 0x01 / 0x02

パラレルポート各ピンを入出力、又はシリアルなどのモードで使用するかを paddr で設定(0:入力 / 1:出力 / 2:シリアル)し、データの読み書きを padr で行うことができます。

padr0, 1, 2, 5, 6 / paddr0, 1, 2, 5, 6 は各 CH ごとについての読み書きが可能で、padr は全 CH(8bit)一括の読み書きが可能です。CH0 が最下位ビット、CH7 が最上位ビットとして、padr (全 CH) の各ビットに対応します。(CH3, 4, 7 はシリアルモード固定なため、書込みを行っても値は変化しません。)

例 13-8 パラレルポート操作のサンプルプログラム

```

#include <fcntl.h>
#include <stdio.h>

int main (void)
{
    int fd_dds, fd_dr;
    unsigned char val;

    //CHO の Direction を書き込み専用でオープン
    fd_dds = open ( "/dev/padr0" , O_WRONLY);
    if (fd_dds < 0) {
        fprintf (stderr, "Open error.¥n");
        return - 1;
    }
    // CHO を読み書き可能でオープン
    fd_dr = open ( "/dev/padr0" , O_RDWR);
    if (fd_dr < 0) {
        fprintf (stderr, "Open error.¥n");
        return - 1;
    }

    val = 1;
    write (fd_dds, &val, sizeof(unsigned char)); //CHO を出力に
    val = 1;
    write (fd_dr, &val, sizeof(unsigned char)); //CHO に High を出力

    val = 0;
    write (fd_dds, &val, sizeof(unsigned char)); //CHO を入力に
    read (fd_dr, &val, sizeof(unsigned char)); //CHO を val に読み込む
    printf ( "padr0: %d¥n" , val); //val を表示

    close (fd_dds);
    close (fd_dr);

    return 0;
}

```


13.3.2. LED 制御用ドライバ

ステータス LED(D4)を制御するためのデバイスノードのパラメータは以下の通りです。

表 13-7 LED 制御用デバイスノードのパラメータ

タイプ	メジャー 番号	マイナー 番号	ノード名 (/dev/xxx)
キャラクタ デバイス	240	0	ajled

デバイスノードをオープンし、次のデータを1バイト書き込んで制御します。

表 13-8 書き込みデータと対応する状態

データ	状態
0	消灯
0以外	点灯

また、デバイスノードを読みだすことで現在の LED の状態が取得できます。LED の状態と読みだした値の関係は次のようになります。

表 13-9 読み込みデータと対応する状態

データ	状態
0	消灯
1	点灯

LED 制御のためのサンプルソースコードを以下に示します。

例 13-9 パラレルポート操作のサンプルプログラム

```
#include <fcntl.h>
#include <stdio.h>

int main( void )
{

    int fd = 0 ;
    int data ;
    if( (fd = open( "/dev/ajled", O_RDWR ) ) < 0 ) {
        fprintf( stderr, "Open error.¥n" ) ;
        return -1 ;
    }

    /* LED を点灯 */
    data = 1 ;
    write( fd, &data, 1 ) ;

    /* 現在の LED の状態を取得 */
    read( fd, &data, 1 ) ;
    printf( "current state:0x%02X¥n", data ) ;
    sleep( 1 ) ;

    /* LED を消灯 */
    data = 0 ;
    write( fd, &data, 1 ) ;

    /* 現在の LED の状態を取得 */
    read( fd, &data, 1 ) ;
    printf( "current state:0x%02X¥n", data ) ;

    close( fd ) ;
    return 0 ;
}
```

13.4. URL リスト

表 13-10 URL リスト

URL	概 要
http://armadillo.atmark-techno.com/	Armadillo 公式サイト
http://www.digi.com/	Digi 社公式サイト
http://www.uclinux.org/	uClinux 公式サイト
http://www.uclibc.org/	uClibc 公式サイト
http://www.colinux.org/	CoLinux 公式サイト
http://www.busybox.net/	busybox 公式サイト
http://tinylogin.busybox.net/	tinylogin 公式サイト
http://www.net-snmp.org/	ucd-snmp 公式サイト
http://hp.vector.co.jp/authors/VA002416/	TeraTerm 公式サイト
http://alioth.debian.org/projects/minicom/	minicom 公式サイト
http://www.beyondlogic.org/	binary-flat についての記述
http://www.lhut32.com/index.shtml	LHA ユーティリティ 32 のページ

改訂履歴

Ver.	年月日	改訂内容
1.00	2003.12.24	・初版発行
1.01	2004.1.6	・「6.5. ネットワーク設定」の記述を訂正
1.02	2004.1.14	<ul style="list-style-type: none"> ・「6.2.2. /etc/config ディレクトリの特性」、Config 領域破損時の、復旧動作の記述を追加 ・「6.7. ftp ログイン」、ログインできるユーザを「guest のみ」に変更 ・「7.1. 開発環境の構築」に推奨環境/ディストリビューションの記述を追加 ・「8.2. 購入時の状態へ「イメージ」を書き戻すには(Hermit for WIN32 版)」を追加 ・「A.3. スタックサイズの変更」の記述を追加 ・「表 13-4 コマンド一覧」へコマンド(addgroup, adduser, delgroup, deluser, passwd, su, sulogin)の解説を追加 ・「表 13-4 コマンド一覧」から、コマンド(fsck.minix, free, ln, mkfs.minix)を削除 ・「6.2.2. /etc/config ディレクトリの特性」、「6.4. 終了方法」、「6.5.2. DHCP を使用する場合」の各説明中、Flash メモリへのデータ更新の記述を修正 ・ドキュメント内の誤植を訂正
1.03	2004.1.22	・ドキュメント内の誤植を訂正
1.04	2004.2.26	<ul style="list-style-type: none"> ・「5.2.2.書き換えイメージの転送」の記述を訂正 ・「6.9. watchdog タイマー機能」、「6.10. SNTP 機能」を追加 ・「7.2. Windows 上に開発環境を構築する」を追加 ・「表 13-4 コマンド一覧」へコマンド(msntp)を追加 ・「Appendix.C URL リスト」へ「Digi 社」、「cygwin 公式サイト」を追加 ・ドキュメント内の誤植を訂正
1.05	2004.3.9	・「5.2.2.書き換えイメージの転送」に” Windows による方法”を追加
1.06	2004.4.14	<ul style="list-style-type: none"> ・改訂履歴中、文書内へのリンクを修正 ・「パラレルポートドライバ」を追加
2.00	2004.6.10	・全面改訂
2.01	2004.9.3	<ul style="list-style-type: none"> ・msntp アプリケーション除外に伴う記述の修正 ・ドキュメント内の誤植を訂正
2.02	2004.10.5	・「LED 制御用ドライバ」に関する記述を追加
2.03	2004.12.6	・会社住所表記を変更

Armadillo-J[AJ010] User's Guide

2004 年 12 月 6 日 ver.2.03

株式会社アットマークテクノ

060-0035 札幌市中央区北 5 条東 2 丁目 AFT ビル

TEL011-207-6550 FAX011-207-6570
