atmark-dist
Developers Guide

Version 1.0

2005 年 4 月 20 日

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

Armadillo オフィシャルサイト
http://armadillo.atmark-techno.com/

SUZAKU オフィシャルサイト
http://suzaku.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. 謝辞……2
2. uClinux-dist について……3
3. デフォルトイメージのビルド……4
3.1. ソースコードの取得……4
3.2. ソースコードアーカイブの展開……4
3.3. 設定……5
3.4. ビルド……6
3.5. イメージ……6
3.6. まとめ……6
4. ディレクトリ構成……7
4.1. Makefile……7
4.2. config……7
4.3. tools……7
4.4. glibc と uClibc……8
4.5. user……8
4.6. vendors……8
5. Make の基本……10
6. 基本ターゲット……12
6.1. 設定(config)……12
6.1.1. テキスト画面での設定 (make config)……12
6.1.2. メニュー画面での設定 (make menuconfig)……12
6.1.3. GUI 画面での設定 (make xconfig)……13
6.2. クリーン(clean)……14
6.3. 依存関係の解決(dep)……14
6.4. デフォルト(all)……14
7. イメージファイルの作成……15
7.1. 全体の流れ……15
7.2. メニューベースコンフィグレーションの基本操作……16
7.2.1. 移動……16
7.2.2. サブメニューの選択……16
7.2.3. リストからの選択……16
7.2.4. 有効無効の選択……16
7.3. コンフィグレーション……16
7.3.1. Main Menu(メインメニュー)……17
7.3.2. Vendor/Product Selection(ベンダ/プロダクト選択)……18
7.3.3. Kernel/Library/Defaults Selection(カーネル/ライブラリ/デフォルト選択)……18
7.3.4. Kernel Version(カーネルの選択)……18
7.3.5. Libc Version(C ライブラリの選択)……18
7.3.6. Default all settings (デフォルトの設定に戻す)……18
7.3.7. Customize Kernel Settings(カーネル設定の変更)……18
7.3.8. Customize Vendor/User Settings(ベンダ/ユーザ設定の変更)……18
7.3.9. Update Default Vendor Settings(デフォルトベンダ設定の更新)……18
7.4. ユーザランドの設定……18
7.4.1. Core Application……18

7.4.2. Library Configuration ····· 18
7.4.3. Flash Tools ····· 19
7.4.4. Filesystem Applications ····· 19
7.4.5. Network Applications ····· 19
7.4.6. Miscellaneous Applications ····· 19
7.4.7. Busybox ····· 19
7.4.8. Tinylogin ····· 19
7.4.9. MicroWindows ····· 19
7.4.10. Game ····· 19
7.4.11. Miscellaneous Configuration ····· 19
7.4.12. Debug Builds ····· 19
7.5. カーネルソースコードの依存関係解決 ····· 19
7.6. ビルド ····· 20
7.7. 詳細なビルドの流れ ····· 20
7.7.1. subdirs ターゲット ····· 20
7.7.2. romfs ターゲット ····· 21
7.7.3. module ····· 22
7.7.4. module_install ····· 22
7.7.5. image ····· 23
8. プロダクトディレクトリ ····· 24
8.1. config.arch ····· 24
8.2. config.linux-2.4.x ····· 24
8.3. config.vendor ····· 24
8.4. config.uClibc ····· 25
8.5. Makefile ····· 25
9. romfs インストールツール ····· 26
9.1. 概要 ····· 26
9.2. ファイルのインストール ····· 28
9.3. ディレクトリのインストール ····· 28
9.4. リンクの作成 ····· 29
9.5. ファイルへの情報追記 ····· 29
9.6. 条件実行 ····· 30
10. 新規アプリケーションの追加方法 ····· 31
10.1. Out of Tree コンパイル ····· 31
10.1.1. 準備 ····· 31
10.1.2. ソースコードの用意 ····· 31
10.1.3. ビルド（uClinux） ····· 32
10.1.4. インストール ····· 33
10.1.5. image ファイルの作成 ····· 33
10.1.6. 複数のソースコード ····· 33
10.1.7. pthread 対応 ····· 35
10.2. プロダクト別のアプリケーション ····· 35
10.2.1. ディレクトリの準備 ····· 35
10.2.2. ソースコードの用意 ····· 36
10.2.3. 追加アプリケーションの設定 ····· 36
10.2.4. ビルド ····· 36
10.3. user ディレクトリへのマージ ····· 37
10.3.1. ディレクトリの準備 ····· 37
10.3.2. ソースコードの用意 ····· 37
10.3.3. 追加アプリケーションの設定 ····· 37
10.3.4. アプリケーションの選択 ····· 38
10.3.5. ビルド ····· 38

10.3.6. コンフィグの命名規則 ····· 38
10.3.7. 複数のアプリケーション ····· 39
11. 複数カーネルの利用 ····· 41
12. 特有なアプリケーションの説明 ····· 42
12.1. NetFlash ····· 42
12.2. Flatfsd ····· 43

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

図目次

図 6-1 メニューベースコンフィグレーションの画面……13
図 6-2 GUI ベースコンフィグレーションの画面……13
図 7-1 イメージ作成の流れ……15
図 10-1 メニューに追加された hello……38

例目次

例 3-1 atmark-dist のファイル名 4
例 5-1 Makefile 練習用のディレクトリを作成 10
例 5-2 簡単な Makefile 10
例 5-3 make の実行 10
例 5-4 ゴールを指定して make を実行 11
例 5-5 複数のゴールを指定して make を実行 11
例 6-1 テキストベースコンフィグレーション 12
例 6-2 メニューベースコンフィグレーションの起動コマンド 13
例 6-3 GUI ベースコンフィグレーションの起動コマンド 13
例 7-1 make dep の実行 19
例 7-2 make の実行 20
例 7-3 user/Makefile からプロダクト Makefile が呼び出される 21
例 7-4 トップレベル Makefile での romfs ターゲット処理 22
例 7-5 user/Makefile での romfs ターゲット処理 22
例 9-1 romfs-inst.sh のヘルプ 26
例 9-2 romfs-inst.sh 構文 27
例 10-1 atmark-dist/config/config.in の変更点 37
例 10-2 atmark-dist/user/Makefile の変更点 37
例 10-3 atmark-dist/config/config.in の変更点(複数アプリケーション) 39
例 10-4 atmark-dist/user/Makefile の変更点(複数アプリケーション) 39
例 10-5 Makefile(複数アプリケーション) 40
例 12-1 netflash の実行 42
例 12-2 netflash のヘルプ 42

1.はじめに

1.1.マニュアルについて

本マニュアルは、アットマークテクノが配布している atmark-dist 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。

- カーネルとユーザランドのビルド
- カスタマイズ
- アプリケーション開発
- 特有のアプリケーション

また、特別な表記がないかぎり作業用のコンピュータでは Linux をベースにした OS が動作しているものと仮定します。 Windows 環境でも coLinux を使うことで簡単に Linux 環境を構築することが可能です。coLinux について詳しくは http://www.colinux.org/ を参照してください。

本マニュアルが atmark-dist の機能を最大限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
: :	コマンド実行後の出力を省略

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。ユーザホームディレクトリは「~」で表します。

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
[Target /]#	ターゲットボード上の特権ユーザで実行
[Target /]$	ターゲットボード上の一般ユーザで実行

1.4. 謝辞

atmark-dist は uClinux-dist をベースに作成されています。

uClinux-dist 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 McCulloughu 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、さらにそれぞれのコミュニティによって開発・保守されています。

2.uClinux-dist について

uClinux-dist は uClinux.org が配布する、ソースコードベースのディストリビューションです。

最初 uClinux-dist は uClinux[1]用に作成されましたが、uClinux 専用というわけではありません。uClinux に対応するために追加変更を行ったもので、設定時に既存の Linux を選択することで i386 や ARM、PowerPC のような MMU をもった CPU にも対応しています。

uClinux では既存の Linux にはない制限がいくつか存在します。そのため、uClinux を使う場合には、既存の Linux アプリケーションを uClinux に対応させる必要がありました。uClinux に対応させたアプリケーションを、Linux カーネルのビルドシステムと組み合わせ、Flash メモリなどに書き込むためのファイルの生成まで自動でおこなってくれるようにしたものが、uClinux-dist です。

既存の Linux ディストリビューションは、主にデスクトップやサーバ用に構成されています。このため、組み込み機器のベースとして採用するには、ファイルサイズやメモリ使用量などに問題がありました。uClinux-dist では最初から組み込み機器を想定しているため、コンパイル時に必要な機能だけを選択することが可能になっています。

また、製品別に専用の設定を持つことができ、それを選択することで容易に特定製品向けのイメージファイルを作成できることや、カーネルとユーザランドの設定を同じメニューから選択でき、一つのイメージファイルとして出力するなど、多くの優れた特長があります。

このため uClinux を使用する製品に限らず、MMU を搭載し Linux が動作する組込み製品においても、利用すべきメリットがあります。

[1] uClinux とは MMU を持たないマイクロコンピュータでも動作するように作成された Linux です。通常 HDD などの大型補助記憶装置を持たない製品で利用されます。2.6 系の Linux では uClinux の成果がマージされています。

3.デフォルトイメージのビルド

はじめに、使用しているターゲットボードのデフォルトイメージを作成してみます。
なお作業を行う際は、必ず一般ユーザで行ってください。ルートユーザで行うと作業ミスなどにより、開発用マシンの環境を壊す可能性があります。

3.1.ソースコードの取得

アットマークテクノが配布している atmark-dist は、以下の URL からダウンロードすることができます。

http://download.atmark-techno.com/dist/

また、アットマークテクノの製品ポータルサイトや開発キットでも入手可能です。

uClinux.org で配布している オリジナルの uClinux-dist は、uClinux-dist-YYYYMMDD.tar.gz と名前に日付が付いています。YYYY は年を、MM は月を、DD は日を表しています。アットマークテクノではファイル名に、atmark-dist-YYYYMMDD という名前をつけて配布しています。

例 3-1 atmark-dist のファイル名

```
[PC ~]$ ls
atmark-dist-20050413.tar.gz
```

3.2.ソースコードアーカイブの展開

atmark-dist は、どこに展開しても問題ありません。作業しやすく、ハードドライブの容量に余裕のある場所を選んで展開してください。展開時で 500MB、コンパイル後には 1GB 近くの容量が必要になる場合もあります。ここでは、便宜上ユーザのホームディレクトリ (~/)に atmark-dist を展開することにします。

```
[PC ~]$ tar xvzf atmark-dist-YYYYMMDD.tar.gz
    :
    :
[PC ~]$ ls
atmark-dist-YYYYMMDD/
```

以降、atmark-dist-YYYYMMDD は、atmark-dist とします。

uClinux.org で配布している uClinux-dist と違い、アットマークテクノで配布している atmark-dist は Linux カーネルのソースコードを含んでいません。お使いの製品に合ったカーネルを製品のポータルサイトからダウンロードするか、または開発キットに含まれる CD-ROM に収録されているものをお使いください。

Linux カーネルソースコードは atmark-dist の中に展開します。各製品用のカーネルを展開し、atmark-dist ディレクトリ内に linux-2.4.x という名前でシンボリックリンクを作成してください。

```
[PC ~]$ ls
linux-2.4.24-foo.tar.gz atmark-dist-YYYYMMDD/
atmark-dist-YYYYMMDD.tar.gz
[PC ~]$ tar xvzf linux-2.4.24-foo.tar.gz
    :
    :
[PC ~]$ cd atmark-dist
[PC ~/atmark-dist]$ ln -s ../linux-2.4.24-foo ./linux-2.4.x
[PC ~/atmark-dist]$ ls -l linux-2.4.x
lrwxrwxrwx           linux-2.4.x -> ../linux-2.4.24-foo
```

3.3.設定

ターゲットボード用の atmark-dist をコンフィギュレーションします。以下の例のようにコマンドを入力し、コンフィギュレーションを開始します。

```
[PC ~/atmark-dist]$ make config
```

使用するボードのベンダー名を聞かれるので「**AtmarkTechno**」と入力してください。

```
[PC ~/atmark-dist]$ make config
config/mkconfig > config.in
#
# No defaults found
#
*
* Vendor/Product Selection
*
*
* Select the Vendor you wish to target
*
Vendor (3com, ADI, Akizuki, Apple, Arcturus, Arnewsh, AtmarkTechno, Atmel, Avnet, Cirrus,
Cogent, Conexant, Cwlinux, CyberGuard, Cytek, Exys, Feith, Future, GDB, Hitachi, Imt,
Insight, Intel, KendinMicrel, LEOX, Mecel, Midas, Motorola, NEC, NetSilicon, Netburner,
Nintendo, OPENcores, Promise, SNEHA, SSV, SWARM, Samsung, SecureEdge, Signal, SnapGear,
Soekris, Sony, StrawberryLinux, TI, TeleIP, Triscend, Via, Weiss, Xilinx, senTec)
[SnapGear] (NEW) AtmarkTechno
```

次にボード名を聞かれます。使用しているボード名を入力してください。ここでは例として「**Armadillo-9**」を入力します。

```
*
* Select the Product you wish to target
*
AtmarkTechno Products (Armadillo-9, SUZAKU) [Armadillo-9] (NEW) Armadillo-9
```

次は、使用する C ライブラリを指定します。使用するボードによってサポートされているライブラリは異なります。Armadillo-9 では、None を選択します。詳しくは、お使いの製品のソフトウェアマニュアルを参照してください。

```
*
* Kernel/Library/Defaults Selection
*
```

```
*
* Kernel is linux-2.4.x
*
Libc Version (None, glibc, uC-libc, uClibc) [uClibc] (NEW) None
```

次にデフォルトの設定にするかどうか質問されます。Yes を選択してください。

```
Default all settings (lose changes) (CONFIG_DEFAULTS_OVERRIDE) [N/y/?] (NEW) y
```

最後の3つの質問は No と答えてください。

```
Customize Kernel Settings (CONFIG_DEFAULTS_KERNEL) [N/y/?] n
Customize Vendor/User Settings (CONFIG_DEFAULTS_VENDOR) [N/y/?] n
Update Default Vendor Settings (CONFIG_DEFAULTS_VENDOR_UPDATE) [N/y/?] n
```

質問事項が終わるとビルドシステムが設定を行います。すべての設定が終わるとプロンプトに戻ります。

3.4. ビルド

ビルドするには以下のコマンドを入力してください。ビルドの途中でいくつか新しく質問される場合があります。その場合は Enter キーを押してください。

```
[PC ~/atmark-dist]$ make dep all
```

3.5. イメージ

イメージファイルは images ディレクトリに生成されます。選択した製品によって、生成されるファイル数やファイルの種類が異なる場合があります。お使いの製品のソフトウェアマニュアルを参照してください。

できあがったイメージファイルを製品に書き込む方法は、お使いの製品のソフトウェアマニュアルを参照してください。

3.6. まとめ

この章で行った一連のコマンドと選択したオプションを以下にまとめます。

```
[PC ~]$ ls
linux-2.4.24-foo.tar.gz  atmark-dist-YYYYMMDD.tar.gz
[PC ~]$ tar xvzf atmark-dist-YYYYMMDD.tar.gz
[PC ~]$ tar xvzf linux-2.4.24-foo.tar.gz
[PC ~]$ cd atmark-dist
[PC ~/atmark-dist]$ ln -s ../linux-2.4.24-foo ./linux-2.4.x
[PC ~/atmark-dist]$ make config
  Vendor AtmarkTechno
  AtmarkTechno Products Armadillo-9
  Libc Version None
  Default all settings y
  Customize Kernel Settings n
  Customize Vendor/User Settings n
  Update Default Vendor Settings n
[PC ~/atmark-dist]$ make dep all
[PC ~/atmark-dist]$ ls image/
image.bin  linux.bin  romfs.img
```

4.ディレクトリ構成

この章では、atmark-dist のディレクトリ構成について説明します。ディレクトリ構成を理解することは、atmark-dist をベースに開発を行うあたり、非常に重要なポイントとなります。

この章では各ディレクトリの概要を説明するとともに、各ディレクトリに関連した説明がなされている章を紹介します。

```
[PC ~]$ cd atmark-dist
[PC ~/atmark-dist]$ tree -L 1 -F
.
|-- COPYING
|-- Documentation/
|-- Makefile
|-- README
|-- SOURCE
|-- bin/
|-- config/
|-- freeswan/
|-- glibc/
|-- lib/
|-- linux-2.4.x -> ../kernel/linux-suzaku/
|-- tools/
|-- uClibc/
|-- user/
`-- vendors/

11 directories, 4 files
```

4.1.Makefile

atmark-dist は、すべて make コマンドによって作業を行います。この atmark-dist/Makefile をトップレベル Makefile と呼び、他のディレクトリにある Makefile とは区別することにします。トップレベル Makefile は、atmark-dist のビルドをコントロールする大事なファイルです。ここで定義されているターゲットは「6.基本ターゲット」の章で詳しく説明します。また Make コマンドについては、「5.Make の基本」で簡単に説明します。

4.2.config

config ディレクトリには設定に必要な script や Makefile が収録されています。

Makefile と config.in については、「10.2.プロダクト別のアプリケーション」の章で詳しく説明します。

4.3.tools

tools には、ビルドに必要ないくつかのツールが収められています。このディレクトリに収められている romfs-inst.sh はプロダクトディレクトリの Makefile でよく使います。「9.romfs インストールツール」で、詳しく説明します。

cksum は、NetFlash で使うことがあります。NetFlash については、「12.1.NetFlash」を参照してください。

4.4. glibc と uClibc

GNU C library (glibc)と uClibc は、atmark-dist で採用している C ライブラリです。

4.5. user

このディレクトリにはユーザランドアプリケーションが収められています。多くのアプリケーションは、GNU/Linux 用のアプリケーションを uClinux でも使用できるように変更したものですが、uClinux 専用に開発されたものも含まれます。

atmark-dist をベースにソフトウェア開発を行う上で特徴的なアプリケーションは「12.特有なアプリケーションの説明」で詳しく説明します。

4.6. vendors

vendors ディレクトリは以下のようになっています。

```
[PC ~/atmark-dist]$ tree vendors
vendors
    :
    :
|-- AtmarkTechno
|   |-- Armadillo
|   |-- Armadillo-9
|   |-- Armadillo-J
|        |-- Makefile
|        |-- config.arch
|        |-- config.linux-2.4.x
|        |-- config.uClibc
|        |-- config.vendor
|        |-- default
|        `-- etc
|   `-- SUZAKU
|        |-- Makefile
|        |-- config.arch
|        |-- config.linux-2.4.x
|        |-- config.uClibc
|        |-- config.vendor
|        |-- default
|        `-- etc
    :
    :
|-- config
|   |-- arm
|   |    `-- config.arch
|   |-- armnommu
|   |    `-- config.arch
|   |-- h8300
|   |    `-- config.arch
|   |-- i386
```

```
|   |    `-- config.arch
|   |-- i960
|   |    `-- config.arch
|   |-- m68knommu
|   |    `-- config.arch
|   |-- microblaze
    :
    :
```

vendors ディレクトリの中には、ベンダー名のディレクトリがたくさん入っています。AtmarkTechno もそのうちのひとつです。ベンダー名のディレクトリの中にはプロダクト用のサブディレクトリが複数入っています。AtmarkTechno ディレクトリ内には、Armadillo をはじめ、atmark-dist に対応しているプロダクトがそれぞれ入っています。これらのディレクトリをプロダクトディレクトリと呼びます。プロダクトディレクトリには、個々の製品用のイメージをビルドするためのさまざまなファイルが入っています。プロダクトディレクトリについては、「8.プロダクトディレクトリ」を参照してください。

ベンダー名のディレクトリ以外には、config というディレクトリが vendors ディレクトリに中にあります。アーキテクチャごとのデフォルト設定が config.arch という名前でそれぞれのディレクトリ内に保存されています。この config.arch はプロダクトディレクトリの config.arch から参照されています。詳しくは「8.1.config.arch」を参照してください

5.Make の基本

make は、プログラムのコンパイルに広く使われているコマンドです。どのソースコードをどのようにコンパイルすれば良いかを判断し、コンパイルに必要なコマンドを実行してくれます。make を使うには Makefile と呼ばれるファイルが必要です。Makefile には、コンパイルされるソースコードの依存関係や、コンパイルに必要なコマンドなどの情報が書かれています。

atmark-dist のビルドシステムでも Makefile をベースとしています。ここでは、make コマンドと Makefile の基本について簡単に説明します。

最初に練習用のディレクトリを作成します。ここでは、「maketest」という名前にします。

例 5-1 Makefile 練習用のディレクトリを作成

```
[PC ~]$ mkdir maketest
[PC ~]$ cd maketest
[PC ~/maketest]$
```

次に、maketest ディレクトリ内に簡単な Makefile を用意します。

例 5-2 簡単な Makefile

```
hello:
        echo  'Hello World'

bye:
        echo  'Bye bye'
```

Makefile の中で、hello と bye はターゲットと呼ばれます。ターゲットは必ずコロンで終わらなければいけません。ターゲットの次の行が実際に実行されるコマンドです。ここではコンパイルをせず、「Hello World」を出力するために、echo コマンドを使っています。実行されるコマンドの前はタブで始まらなければならない規則があります。

例 5-3 make の実行

```
[PC ~/maketest]$ ls
Makefile
[PC ~/maketest]$ make
echo  'Hello World'
Hello World
[PC ~/maketest]$
```

上記は、実際に make コマンドを実行した例です。make コマンドを実行すると、Makefile 中の実行されるコマンド「echo 'Hello World'」と、コマンドの結果「Hello World」が表示されます。

make コマンドは Makefile 内で定義されているターゲットを引数としてとることが可能です。指定されたターゲットをゴールと呼びます。

例 5-4 ゴールを指定して make を実行

```
[PC ~/maketest]$ make hello
echo  'Hello World'
Hello World
[PC ~/maketest]$ make bye
echo  'Bye bye'
Bye bye
[PC ~/maketest]$
```

make コマンドが引数を取らない場合は、Makefile 内の一番上にあるターゲットをゴールとして実行されます。このため、「make」と「make hello」では同じ動作になります。多くの Makefile では、一番上のターゲットは「all」という名前で定義されています。

複数のゴールを羅列することもできます。この場合は、左から順に実行されます。

例 5-5 複数のゴールを指定して make を実行

```
[PC ~/maketest]$ make hello bye
echo  'Hello World'
Hello World
echo  'Bye bye'
Bye bye
[PC ~/maketest]$
```

make と Makefile の詳しい情報は、make のマニュアルまたは info をご覧ください。

6.基本ターゲット

atmark-dist でビルドする場合に、よく使うターゲットをここで説明します。atmark-dist の目的が「ターゲットボードに書き込むためのイメージファイル作成」であることは、すでに説明したとおりです。「デフォルトイメージのビルド」で使用したターゲットについて詳しく説明します。

6.1.設定（config）

設定用のターゲットには、以下の 3 種類が用意されています。表示方法が異なるだけで、すべて atmark-dist の設定変更を行うためのターゲットです。

6.1.1. テキスト画面での設定 (make config)

make config は、「3.デフォルトイメージのビルド」の章で使用した設定方法です。デフォルトのイメージを作成するような簡単な設定をするときに便利です。

例 6-1 テキストベースコンフィグレーション

```
[PC ~/atmark-dist]$ make config
config/mkconfig > config.in
#
# Using defaults found in .config
#
*
* Vendor/Product Selection
*
*
* Select the Vendor you wish to target
*
Vendor (3com, ADI, Arcturus, Arnewsh, AtmarkTechno, Atmel, Avnet, Conexant, Cwlinux,
Cytek, Exys, Feith, Future, GDB, Hitachi, Imt, Insight, Intel, KendinMicrel, LEOX, Mecel,
Midas, Motorola, NEC, NetSilicon, Netburner, Nintendo, OPENcores, Promise, SSV, SWARM,
Samsung, SecureEdge, SnapGear, Soekris, TelelP, Triscend, Via, Weiss, Xilinx, senTec)
[AtmarkTechno]
  defined CONFIG_DEFAULTS_ATMARKTECHNO
*
* Select the Product you wish to target
*
AtmarkTechno Products (SUZAKU) [SUZAKU]
  defined CONFIG_DEFAULTS_ATMARKTECHNO_SUZAKU
*
* Kernel/Library/Defaults Selection
*
*
* Kernel is linux-2.4.x
*
Libc Version (glibc, uC-libc, uClibc) [uClibc]
```

6.1.2. メニュー画面での設定 (make menuconfig)

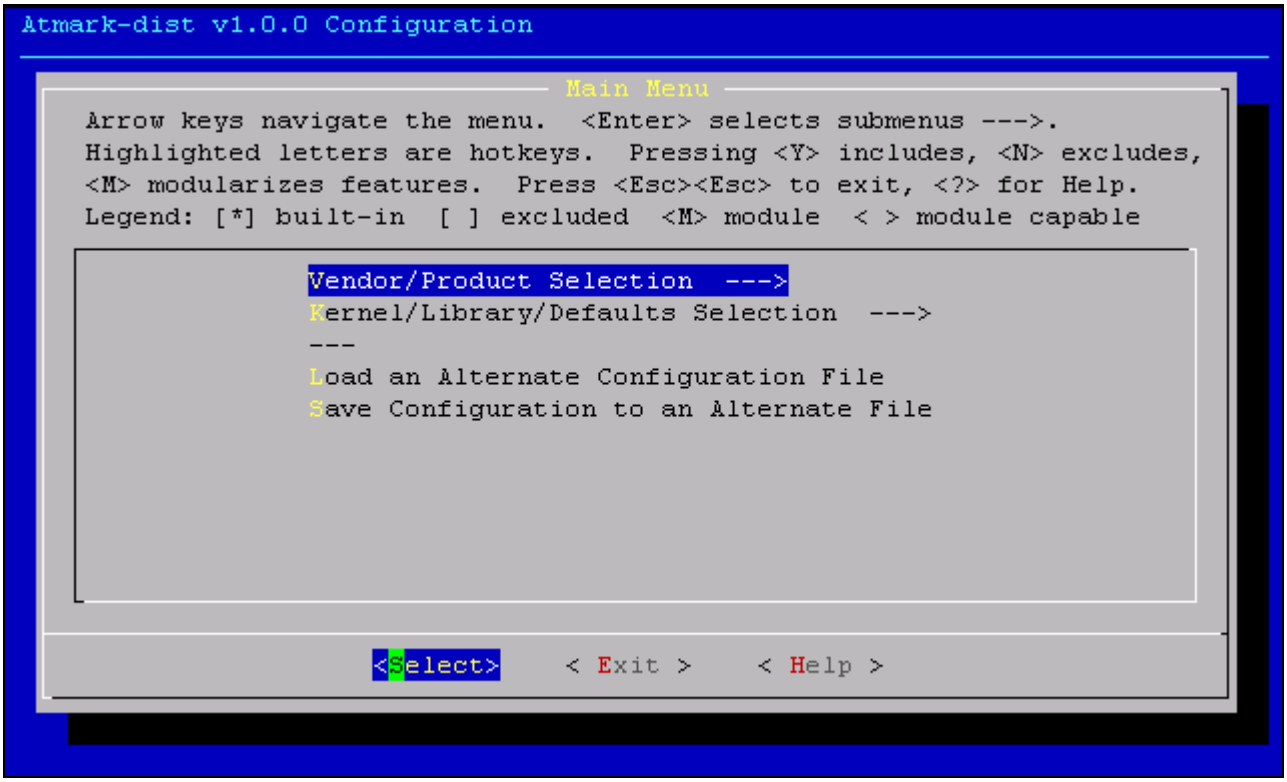
make menuconfig では、メニュー画面を使って設定を行うことができます。メニュー画面は Ncurses を使って描画されます。このターゲットが指定されたときに画面などをコントロールするプログラムをビルドするため、

Ncurses のライブラリとヘッダファイルが必要になります。多くのディストリビューションでは、Ncurses の開発用パッケージが用意されていますので、インストールしてください。

例 6-2 メニューベースコンフィグレーションの起動コマンド

```
[PC ~/atmark-dist]$ make menuconfig
```

図 6-1 メニューベースコンフィグレーションの画面



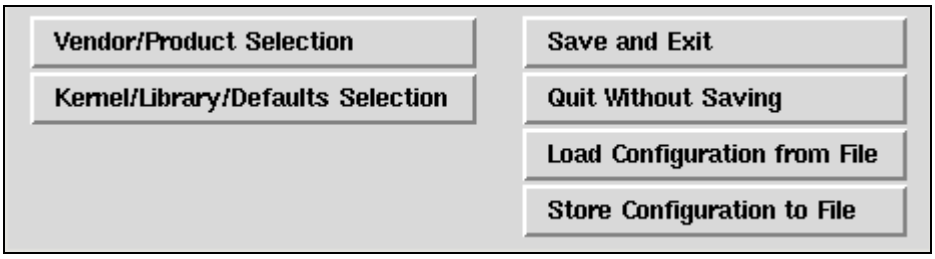
6.1.3. GUI 画面での設定 (make xconfig)

X Window System が使える環境であれば、make xconfig ターゲットを使用することができます。make xconfig では、ターミナル中のメニュー画面と違い、マウスで操作することができます。

例 6-3 GUI ベースコンフィグレーションの起動コマンド

```
[PC ~/atmark-dist]$ make xconfig
```

図 6-2 GUI ベースコンフィグレーションの画面



6.2. クリーン（clean）

atmark-dist 内をきれいにするためのターゲットです。atmark-dist には以下の 3 種類の clean ターゲットが用意されています。

make clean
カーネルディレクトリの clean と、romfs/image ディレクトリ内のファイルの削除を行う。

make real_clean
romfs/image ディレクトリの削除や、.config などのコンフィグ用ファイルの削除を行う

make distclean
カーネルディレクトリで、distclean ターゲットの実行などを行う

6.3. 依存関係の解決（dep）

2.4 系までの Linux カーネルのビルドシステムでは、make 前に依存関係の解決を行わなければなりません。このターゲットは、カーネルディレクトリで依存関係の解決を行います。atmark-dist のソースアーカイブを展開した後や distclean を行った後に必ず実行しなければならないターゲットです。

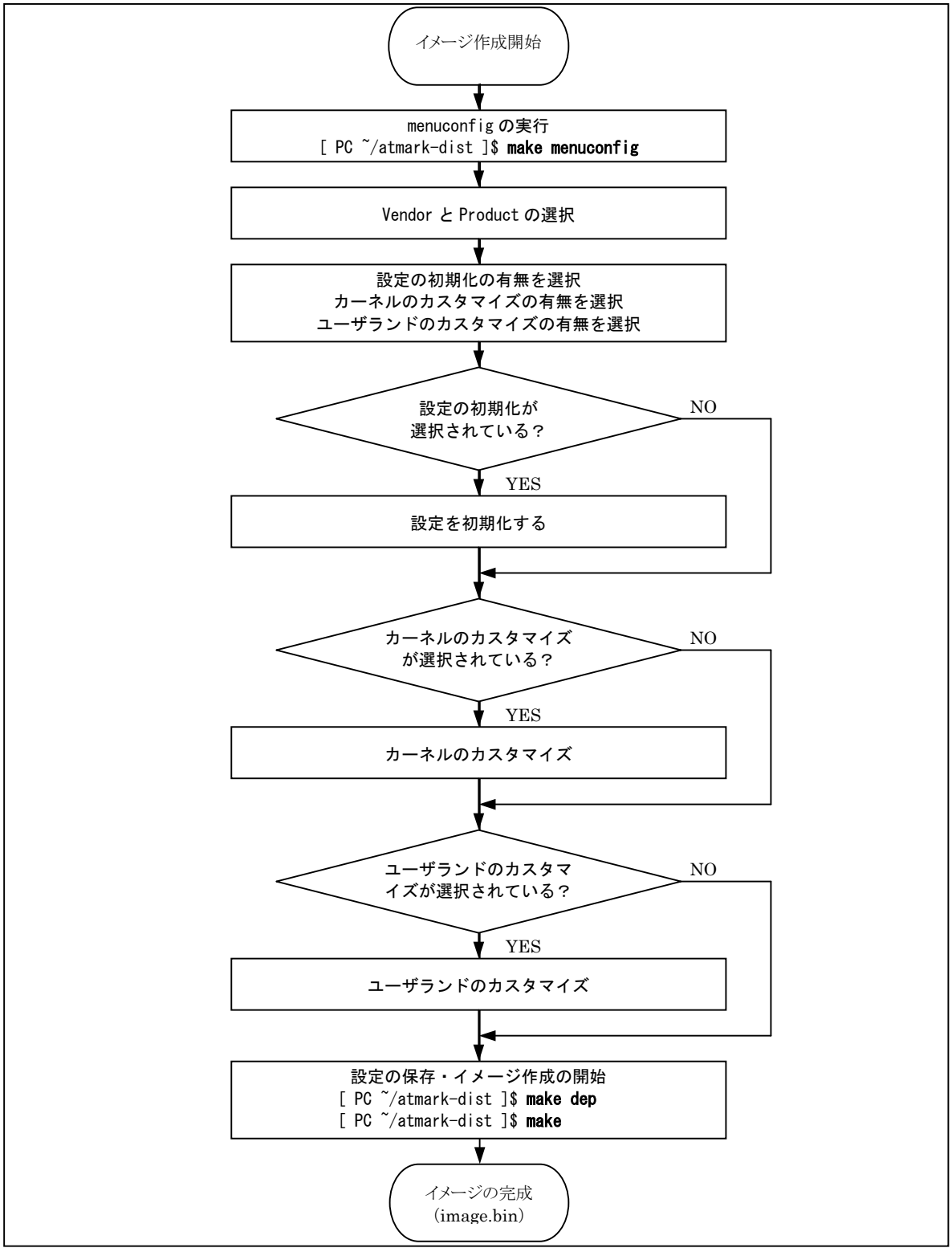
6.4. デフォルト（all）

all ターゲットは、atmark-dist のデフォルトターゲットです。make コマンドをオプションなしで実行することで、このターゲットが実行されます。all では、必要なコード（カーネル、ユーザランドアプリケーション、ライブラリ）のコンパイルを行い、ターゲットボードに転送できるイメージファイルを生成します。

7.イメージファイルの作成

7.1.全体の流れ

図 7-1 イメージ作成の流れ



デフォルトのイメージを作成する一連の手順を順番に説明します。一連の手順は、大きく3つに分割することができます。

- コンフィグレーション
- カーネルソースコードの依存関係解決
- ビルド

7.2. メニューベースコンフィグレーションの基本操作

7.2.1. 移動

カーソルキーでメニュー内の移動を行います。

7.2.2. サブメニューの選択

Enter キーを押すことで、サブメニューを選択できます。サブメニューは --->で表示されます。

7.2.3. リストからの選択

小括弧「( )」で表示されている部分は、リストから選択する部分です。Enter キーでリスト画面に移動し、上下のカーソルキーで選択対象に移動し、Enter キーで選択します。

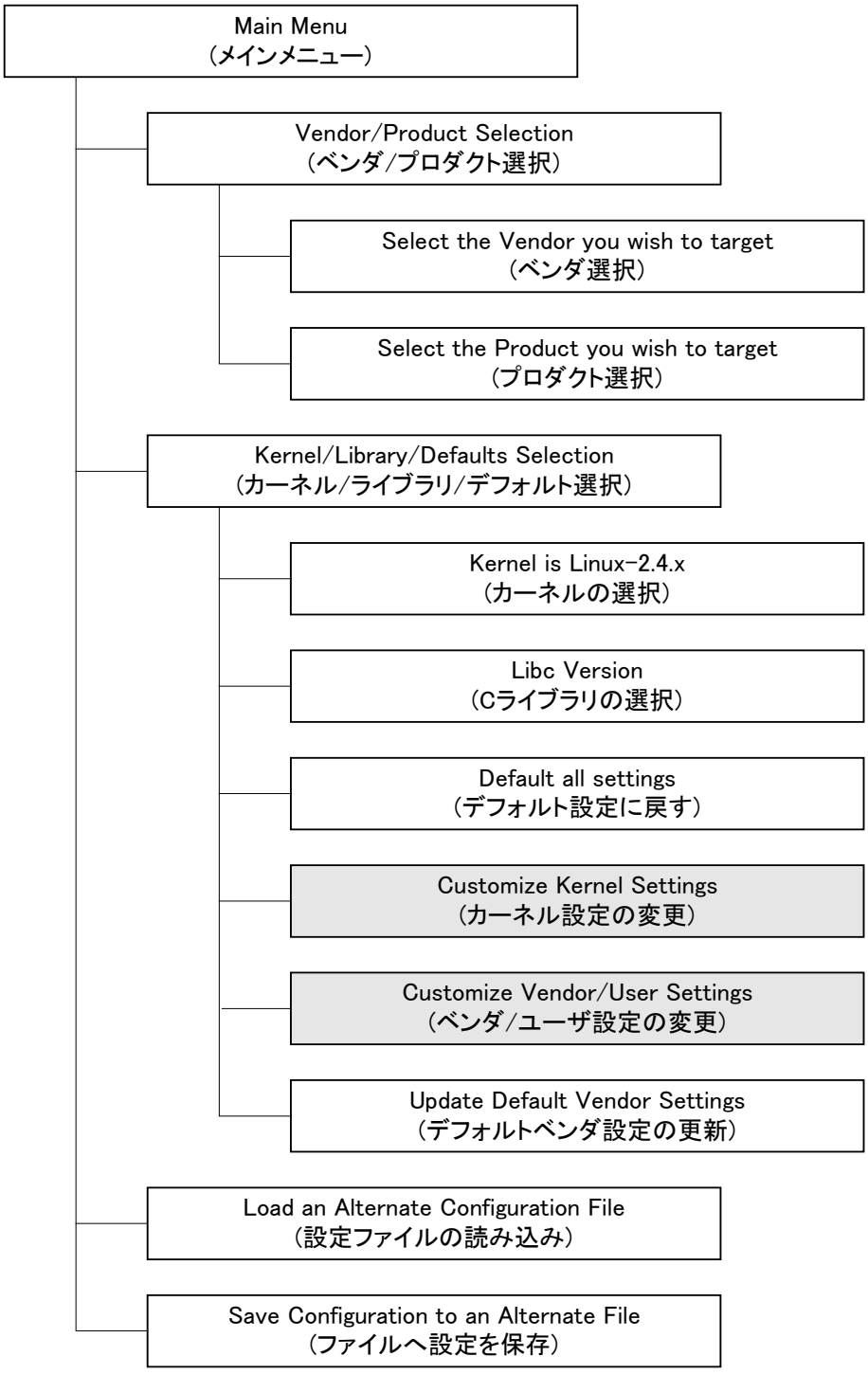
7.2.4. 有効無効の選択

大括弧「[ ]」は、有効無効の選択を表します。選択されるとアスタリスク「*」が大括弧内に表示されます。

7.3. コンフィグレーション

「6.1.設定(config)」で紹介したコマンドのうち、どれか一つを使ってコンフィグレーションを行います。ここでは make menuconfig を例に説明します。

コンフィグレーションは、大きく次のようにわけることができます。



この中でカーネル設定とベンダー/ユーザ設定は、選択後に一度メニューを終了することで、それぞれの設定画面が自動的に表示されます。

コンフィグレーションは、以下のようにはじめます。

```
[PC ~/atmark-dist]$ make menuconfig
```

7.3.1. Main Menu（メインメニュー）

menuconfig を実行するとメインメニュー画面が表示されます。この画面から各サブメニューに移動することができます。

7.3.2. Vendor/Product Selection（ベンダ/プロダクト選択）

ベンダーとプロダクトを選択します。先にベンダーを選択し、その後プロダクトを選択します。

7.3.3. Kernel/Library/Defaults Selection（カーネル/ライブラリ/デフォルト選択）

カーネルやライブラリ、デフォルトの選択を行います。以降のメニューはこのサブメニュー中にあります。

7.3.4. Kernel Version（カーネルの選択）

複数のカーネルをサポートしたプロダクトの場合、ここでビルドするカーネルのバージョンを選択します。アットマークテクノで現在対応しているカーネルは 2.4 系だけのため、他のバージョンを選択することはできません。

7.3.5. Libc Version（C ライブラリの選択）

複数の C ライブラリをサポートしたプロダクトの場合、ここでビルドする C ライブラリを選択します。選択肢としては、以下の4つが選択対象です。

- None
- glibc – GNU C ライブラリ
- uC-libc
- uClibc

None を選択すると、すでに開発環境にインストールされているバイナリを使用します。そのほかの選択肢では、atmark-dist に含まれているコードをビルドします。

製品によって対応しているライブラリが異なりますので、製品のソフトウェアマニュアルを参照してください。

7.3.6. Default all settings （デフォルトの設定に戻す）

すべての設定をデフォルトの状態に変更します。変更されている設定情報はすべて無くなってしまうので注意が必要です。

7.3.7. Customize Kernel Settings（カーネル設定の変更）

Linux カーネルのコンフィグレーションを行うか否かを選択します。選択した場合、atmark-dist の設定終了後に自動的にカーネルの設定画面が起動されます。

7.3.8. Customize Vendor/User Settings（ベンダ/ユーザ設定の変更）

ユーザランドのコンフィグレーションを行うか否かを選択します。選択した場合、atmark-dist の設定終了後に自動的にユーザランドの設定画面が起動されます。ユーザランドのメニューについては次章で説明します。

7.3.9. Update Default Vendor Settings（デフォルトベンダ設定の更新）

現在の設定で、デフォルト設定を更新します。更新した場合、古い設定に戻す方法はありませんので、注意が必要です。

7.4. ユーザランドの設定

ユーザランドのメニューは以下のような項目に別れています。ここでは簡単に各項目について説明します。

7.4.1. 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。

7.4.2. 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。

7.4.3. 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。以降の章で説明する 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。

7.4.4. 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。あとの章で説明する Flatfsd はここで選択することができます。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。

7.4.5. 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。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。

7.4.6. 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。

7.4.7. Busybox

Busybox のカスタマイズを行います。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。

7.4.8. Tinylogin

Tinylogin も複数コマンドの機能をもつアプケーションです。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なため別セクションになっています。

7.4.9. 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。

7.4.10. Game

ゲームです。説明はいらないですよね。

7.4.11. Miscellaneous Configuration

いろいろな設定がまとめられています。SUZAKU の root ユーザのパスワードもここで変更できます。

7.4.12. 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うときに選択します。

7.5. カーネルソースコードの依存関係解決

イメージのビルドにはカーネルのビルドも含まれています。2.4 系の Linux カーネルはビルドの前に依存関係を解決しなければなりません。これは、uClinux でも同じです。

コマンドは make dep を使います。

例 7-1 make dep の実行

```
[PC ~/atmark-dist]$ make dep
    :
    :
[PC ~/atmark-dist]$
```

7.6. ビルド

依存関係の解決が終わったら実際にビルドします。ビルドシステムがすべてを行ってくれるため、開発者は make コマンドを入力するだけです。

例 7-2 make の実行

```
[PC ~/atamrk-dist]$ make
    :
    :
[PC ~/atmark-dist]$ ls images
image.bin  linux.bin  romfs.img
```

make コマンドが終了すると、images ディレクトリにイメージファイルが生成されます。

7.7. 詳細なビルドの流れ

内部的にどのようにビルドが進み、最終的にイメージファイルになるのかを理解すると、必要とされる部分だけビルドすることもできます。これによりビルドに必要な時間を大幅に短縮できます。また、特定製品向けのカスタマイズを行うためにも、この知識は欠かせないでしょう。

デフォルトのビルドを行うとイメージファイルが作成されますが、この間にたくさんのターゲットが実行されています。デフォルトターゲットのビルドルールは、atmark-dist ルートディレクトリの Makefile に以下のように記載されています。

```
ifeq (.config,$(wildcard .config))
include .config

all: subdirs romfs modules modules_install image
else
all: config_error
endif
```

デフォルトビルドを実行すると subdirs, romfs, modules, modules_install, image の順にビルドが行われるのが分かります。
この流れに沿って、各ターゲットのビルドを説明します。

7.7.1. subdirs ターゲット

atmark-dist ルートディレクトリの Makefile の subdirs ビルドルールを以下に示します。

```
DIRS    = $(VENDOR_TOPDIRS) include lib include user
:
:
subdirs: linux
        for dir in $(DIRS) ; do [ ! -d $$dir ] || $(MAKEARCH_KERNEL) -C $$dir ||
 exit 1 ; done
```

subdirs ターゲットをビルドすると、linux, $(VENDOR_TOPDIRS), include lib, include user の順にビルドが行われます。

linux ターゲットは、選択された Linux カーネルのビルドを行います。atmark-dist では version 2.0、2.4、2.6 のカーネルに対応していますが、現在 アットマークテクノの製品では 2.4 系の Linux カーネルのみ対応になっています。

VENDOR_TOPDIRS は、プロダクトディレクトリにある config.arch で指定することが可能ですが、このディレクトリを指定しているプロダクトは少数派のようです。VENDOR_TOPDIR を指定している多くのプロダクトでは、boot というディレクトリを指定しています。アットマークテクノの製品では、この変数は使っていません。

lib は、ライブラリを収めたディレクトリです。uClibc と glibc は lib ディレクトリに入っていませんが、コンフィグ時に lib ディレクトリ内にシンボリックリンクを生成するようになっています。

user は、ユーザランドアプリケーションを集めたディレクトリです。user ディレクトリには専用の Makefile が用意されており、トップレベルの Makefile はそちらに制御を任せるようになっています。

user ディレクトリにある Makefile の中でプロダクトディレクトリの Makefile が呼び出されます。以下の例は user/Makefile からの抜粋です。

例 7-3 user/Makefile からプロダクト Makefile が呼び出される

```
VEND=$(ROOTDIR)/vendors

#
# must run the vendor build first
#
dir_v = $(VEND)/$(CONFIG_VENDOR)/$(CONFIG_PRODUCT)/.
dir_p = $(ROOTDIR)/prop

dir_y =
dir_n =
dir_  =
    :
    :
    :
    :

all: config
        for i in $(sort $(dir_y)) $(dir_v) $(dir_p); do ¥
                [ ! -d $$i ] || make -C $$i || exit $$? ; ¥
        done
```

7.7.2. romfs ターゲット

romfs ターゲットでは、各ディレクトリに対して romfs ターゲットを再帰的に呼びだします。多くの場合、romfs-inst.sh をつかって必要なファイルを、atmark-dist/romfs ディレクトリにインストールします。

例 7-4 トップレベル Makefile での romfs ターゲット処理

```
DIRS    = $(VENDOR_TOPDIRS) include lib include user
        :
        :
        :

romfs:
         for dir in $(DIRS) ; do [ ! -d $$dir ] || $(MAKEARCH) -C $$dir romfs || exit
1 ; done
         -find $(ROMFSDIR)/. -name CVS | xargs -r rm -rf
         done
```

例 7-5 user/Makefile での romfs ターゲット処理

```
VEND=$(ROOTDIR)/vendors

#
# must run the vendor build first
#
dir_v = $(VEND)/$(CONFIG_VENDOR)/$(CONFIG_PRODUCT)/.
dir_p = $(ROOTDIR)/prop

dir_y =
dir_n =
dir_  =
    :
    :
    :
    :

romfs:
          for i in $(dir_v) $(sort $(dir_y)) $(dir_p) ; do ¥
                    [ ! -d $$i ] || make -C $$i romfs || exit $$? ; ¥
          done
```

ビルド時には、user ディレクトリ以下にあるディレクトリのうち、選択されたアプリケーションのビルドが最初に行なわれ、その後、プロダクト Makefile が呼び出されます。

しかし、romfs ターゲットの時には、プロダクトディレクトリの Makefile が最初に呼びだされます。これは、プロダクトごとに柔軟性を与えるためですが、よほど複雑なビルドを行わなければならない場合以外、気にしなくても良いことです。

7.7.3. module

Linux カーネルでは、多くのドライバーなどが module という形で分離できるようになっています。このターゲットは Linux カーネルのビルドシステムにある modules ターゲットを実行します。

7.7.4. module_install

上記のターゲットでビルドされたカーネルモジュールを romfs にインストールします。romfs/lib/modules 内にインストールされます。

7.7.5. image

image ターゲットは、プロダクト Makefile の image ターゲットを実行するためのターゲットです。多くのプロダクトでは、カーネルのイメージとユーザランドのイメージをまとめて、一つのイメージファイルにしています。

直接プロダクト Makefile のターゲットが呼び出されるため、開発者が自由に処理することができます。一般的な流れとしては

1. Linux カーネルの binary file を生成（elf から binary に変換）
2. romfs ディレクトリからイメージファイルを生成(genext2fs や genromfs を使用)
3. 上記 2 つのファイルを 1 つにまとめる
4. NetFlash 用にチェックサムなどを生成しバイナリファイルに添付する

と、なっています。

8.プロダクトディレクトリ

プロダクトディレクトリとは、atmark-dist/vendors/[ベンダー名]/以下にあるディレクトリ群のことです。例えばatmark-dist/vendors/AtmarkTechno/以下には、アットマークテクノの製品名が並びます。

プロダクトディレクトリの下には、ビルドの動作を決める Makefile や、ビルド時にデフォルト値として使われるconfig ファイル、アプリケーションに必要な設定ファイルなど、ビルドシステム内でプロダクトごとに異なる部分が含まれています。

ここでは、デフォルトの設定に使われる config ファイル郡と Makefile について説明します。

8.1.config.arch

config.arch ファイルには、アーキテクチャに依存した設定を記述します。実際には、アーキテクチャごとにデフォルトの値がすでに用意されているため、上書きする設定だけを書くようになっています。

設定できる変数は以下のとおりです。

CPUFLAGS
: CPU のコンパイルフラグを指定することができます。

VENDOR_CFLAGS
: ベンダー依存の CFLAGS フラグを指定することができます。

DISABLE_XIP
: XIP (Execute In Place)を無効にする場合、1を指定します。

DISABLE_SHARED_LIBS
: 共有ライブラリを無効にする場合、1を指定します。

DISABLE_MOVE_RODATA
: 読み込み専用領域の移動を無効にする場合、1を指定します。

LOPT
: ライブラリをコンパイルするときに、コンパイラに渡すオプションを指定します。

UOPT
: ユーザランドアプリケーションをコンパイルするときに、コンパイラに渡すオプションを指定します。

CONSOLE_BAUD_RATE
: シリアルコンソールのボーレートを指定します。

8.2.config.linux-2.4.x

config.linux-2.4.x は、Linux カーネルのビルドシステムが生成する .config を、別の名前でプロダクトディレクトリに保存しています。

このファイルは、カーネルコンフィグレーションのデフォルト値として扱われます。カーネルの値を変更し、変更したものをデフォルト値としたい場合は、このファイルを上書きしてください。また「7.3.9.Update Default Vendor Settings(デフォルトベンダ設定の更新)」メニューを使うことも可能です。

8.3.config.vendor

config.vendor は、ユーザランドアプリケーション等の情報が含まれています。atmark-dist のビルドシステムがmenuconfig などで設定された情報を保持しています。

このファイルは、atmark-dist/config/.config のコピーになっています。

config.linux-2.4.x と同様に上書きすることで、デフォルトの値を変更することができます。

8.4. config.uClibc

config.uClibc は、uClibc の設定ファイルです。dist のメニューから uClibc の変更をすることは、現在できません。uClibc の設定を変更する場合は、uClibc のディレクトリに移動し、uClibc 専用のコンフィグツールを使います。uClibc の設定方法も atmark-dist と同じく Linux カーネルビルドシステムを使用しています。

8.5. Makefile

Makefile では、実際のイメージファイルの生成を制御します。デバイスファイルやプログラムのインストール先、プログラムが必要としている設定ファイルやデータファイルのインストール先、チェックサムの生成などです。

9.romfs インストールツール

コンパイルされたアプリケーションや各種設定ファイルは、Makefile の romfs ターゲットによって atmark-dist/romfs ディレクトリにインストールされます。アプリケーションが必要とする設定ファイルやデータファイルなども、この時点で atmark-dist/romfs ディレクトリにインストールされます。

ディレクトリ名に romfs という名前が使われているのは、多くの組込みシステムでは、デスクトップやサーバ用途の Linux システムで使われている ext2 や ext3、 reiserfs、xfs などではなく romfs が使われるためです。しかし、romfs ディレクトリ自体は romfs に依存しているわけではありません。後述する jffs2 などでも同じ romfs ディレクトリを使います。

romfs ディレクトリの構成は、ターゲットシステムが起動したときにターゲットシステム上で見えるディレクトリ構成と同じ構成になっています。romfs ディレクトリをルートディレクトリとして、その下に bin や dev、etc などが配置されます。

atmark-dist には romfs ディレクトリにファイルを簡単にインストールするために romfs-inst.sh と呼ばれるスクリプトが用意されています。このスクリプトは atmark-dist/tools ディレクトリに入っています。
romfs-inst.sh は、atmark-dist ディレクトリにある Makefile によって ROMFSINST という変数に代入されます。このため、プロダクトディレクトリをはじめとする atmark-dist 内の各ディレクトリの Makefile では romfs-inst.sh がどこに入っているかを気にせず、ROMFSINST という変数で使うのが一般的な方法になっています。

9.1.概要

romfs-inst.sh は、romfs ディレクトリを指定する環境変数 ROMFSDIR が設定されていない場合、簡単な help を出力します。

例 9-1 romfs-inst.sh のヘルプ

```
[PC ~/atmark-dist]$ tools/romfs-inst.sh
ROMFSDIR is not set
tools/romfs-inst.sh: [options] [src] dst
    -v          : output actions performed.
    -e env-var  : only take action if env-var is set to "y".
    -o option   : only take action if option is set to "y".
    -p perms    : chmod style permissions for dst.
    -a text     : append text to dst.
      -A pattern  : only append text if pattern doesn't exist in file
    -l link     : dst is a link to 'link'.
    -s sym-link : dst is a sym-link to 'sym-link'.

    if "src" is not provided,  basename is run on dst to determine the
    source in the current directory.

      multiple -e and -o options are ANDed together.  To achieve an OR affect
      use a single -e/-o with 1 or more y/n/"" chars in the condition.

      if src is a directory,  everything in it is copied recursively to dst
      with special files removed (currently CVS dirs).
```

romfs-inst.sh のコマンド構文は以下のとおりです。

例 9-2 romfs-inst.sh 構文

```
romfs-inst.sh [options] [src] dst
```

[]の部分は省略することができます。もし、src が指定されなかった場合、basename コマンドが dst に適用されて、戻り値を src として使います。romfs-inst.sh はその値を現在のディレクトリ(つまりプロダクト Makefile の場合はプロダクトディレクトリ)から探します。

```
[PC ~]$ basename /foo/bar
bar
```

もし、src がディレクトリの場合は、そのディレクトリ以下すべてのファイルとディレクトリをインストールします。ただし、CVS ディレクトリだけはコピーされません。
以下はオプションの簡単な説明です。

-v
　実際に実行した内容を出力

-e env-var
　env-var が "y"のときだけ、指定されたアクションを実行

-o option
　option が "y"のときだけ、指定されたアクションを実行

-p perms
　chmod 方式で dst のパーミッションを指定

-a text [-A pattern]
　text を dst に追加。-A pattern が指定されているときは、pattern が
　dst に含まれていない場合に text を追加

-l link
　dst で指定された名前で、link へのハードリンクを作成

-s sym-link
　dst で指定された名前で、sym-link へのシンボリックリンクを作成

以降の章では、romfs-inst.sh の使用例を説明します。

9.2. ファイルのインストール

ファイルをインストールする場合は、以下のように romfs ターゲットを Makefile に記述します。

```
romfs:
	$(ROMFSINST) src.txt /etc/dst.txt
```

これは、プロダクトディレクトリ内にある src.txt を romfs ディレクトリの中の /etc/dst.txt にインストールすることを意味しています。もし、romfs ディレクトリが ~/atmark-dist/romfs であれば、
~/atmark-dist/romfs/etc/dst.txt というファイルができあがります。

プロダクトディレクトリにある src.txt の名前を変更して dst.txt としておくことで、以下のように簡単に記述することができます。

```
$(ROMFSINST) /etc/dst.txt
```

上で簡単にふれましたが、src が省略されたとき romfs-inst.sh は dst の basename を使ってカレントディレクトリからファイルを探します。

/etc/dst.txt の basename は、dst.txt なので上記の文は

```
$(ROMFSINST) dst.txt /etc/dst.txt
```

と同じ意味を持ちます。

9.3. ディレクトリのインストール

ターゲットデバイスに多くのファイルをインストールする場合は、ディレクトリごとインストールすると簡単です。たとえば、ターゲットの/etc ディレクトリに多くの設定ファイルをインストールする場合などです。

プロダクトディレクトリに etc というディレクトリを作成し、必要なファイルを置きます。そして Makefile に以下のように記述します。

```
$(ROMFSINST) /etc
```

この例でも src が省略されているので、romfs-inst.sh は dst の basename を使います。/etc の basename は etc なので、romfs-inst.sh はプロダクトディレクトリにある etc というファイルまたはディレクトリを探します。そして今回のようにディレクトリの場合、romfs-inst.sh はディレクトリ内にあるファイルも一緒にインストールしてくれます。

以下のように tree コマンドを使うと簡単に確認できます。

```
[PC ~/]$ tree ~/atmark-dist/vendors/AtmarkTechno/test/etc
:
[PC ~/]$ tree ~/atmark-dist/romfs/etc
:
```

もちろん、保存している名前とは別の名前でインストールすることも可能です

```
	$(ROMFSINST) /etc /var
```

このコマンドでは、プロダクトディレクトリにある etc というディレクトリを romfs/var にインストールします。

9.4. リンクの作成

romfs-inst.sh を使って簡単にリンクを作成することができます。ただし、hard link と symbolic link をきちんと理解しなければいけません。

symbolic link を作成するときは、オプション '-s'を使います。例として a.txt へのシンボリックリンクを作成してみます。プロダクト Makefile の romfs ターゲットは以下のようになります。

```
romfs:
	[ -d $(ROMFSDIR) ] || mkdir -p $(ROMFSDIR)
	$(ROMFSINST) /a.txt
	$(ROMFSINST) -s a.txt /b.txt
```

```
[PC ~/atmark-dist]$ make clean; make romfs
    :
    :
[PC ~/atmark-dist]$ ls -l romfs
total 0
-rw-r--r--  1 guest guest 0 Sep 24 05:43 a.txt
lrwxrwxrwx  1 guest guest 5 Sep 24 05:43 b.txt -> a.txt
```

次は hard link の例です。オプションは'-l'を使います。

```
romfs:
	[ -d $(ROMFSDIR) ] || mkdir -p $(ROMFSDIR)
	$(ROMFSINST) /a.txt
	$(ROMFSINST) -l a.txt /b.txt
```

```
[PC ~/atmark-dist]$ make clean; make romfs
    :
    :
[PC ~/atmark-dist]$ ls -i1 romfs
6077732 a.txt
6296750 b.txt
[PC ~/atmark-dist]$ ls -i1 vendors/AtmarkTechno/test/a.txt
6296750 vendors/ATmarkTechno/test/a.txt
```

romfs 内にできた b.txt は、romfs 内の a.txt へのハードリンクではなく、プロダクトディレクトリにある a.txt へのハードリンクということが inode の番号によってわかります。

ハードリンクの使用は混乱を招きますので、よほどハードドライブの容量に困っていないかぎりお勧めしません。現在の atmark-dist でも、romfs 内への hardlink はあまり使われていないようです。

9.5. ファイルへの情報追記

romfs-inst.sh を使うことで、すでに存在するファイルに簡単に情報を追記することができます。

構文は以下のようになります。

```
$(ROMFSINST) -a "文字列" romfs ディレクトリ内のファイル名
```

```
romfs:
	[ -d $(ROMFSDIR) ] || mkdir -p $(ROMFSDIR)
	$(ROMFSINST) -a 'Hello' /a.txt
	$(ROMFSINST) -a 'World' /a.txt
```

```
[PC ~/atmark-dist]$ make clean; make romfs
    :
    :
[PC ~/atmark-dist]$ cat romfs/a.txt
Hello
World
```

9.6.条件実行

romfs-inst.sh は条件による実行制御が可能です。

```
$(ROMFSINST) -e 変数名 実行するコマンド
```

変数名としてよく用いられるのは、CONFIG_ではじまる環境変数です。

```
romfs:
	[ -d $(ROMFSDIR) ] || mkdir -p $(ROMFSDIR)
	$(ROMFSINST) -e CONFIG_DEFAULTS_ATMARKTECHNO -a 'Hello' /a.txt
	$(ROMFSINST) -e CONFIG_DEFAULTS_UNKNOWN -a 'World' /a.txt
```

```
[PC ~/atmark-dist]$ make clean; make romfs
    :
    :
[PC ~/atmark-dist]$ cat romfs/a.txt
Hello
```

CONFIG_DEFAULTS_UNKNOWN は定義されていないので条件に当てはまらず、a.txt に「World」の文字列は書き出されません。CONFIG_DEFAULTS_ATMARKTECHNO は、ベンダー名で AtmarkTechno を選択すると、atmark-dist/.config に定義されます。

10. 新規アプリケーションの追加方法

この章では、ターゲットボードで動作する新しくアプリケーションを作成する方法と、作成したアプリケーションを atmark-dist 内に追加する方法を説明します。

10.1. Out of Tree コンパイル

Out of Tree コンパイルは、atmark-dist に変更を加えることなく手軽に開発を行うことができる方法です。atmark-dist のビルドシステムを使うため、複雑な Makefile を書く必要もありません。atmark-dist のディレクトリ構造を木に見たて、そのディレクトリ外でコンパイルするためにこう呼ばれています。

ここでは実際にターゲットボード用の"Hello World"を作成します。

10.1.1. 準備

Out of Tree コンパイルでは atmark-dist に入っているビルドシステムやライブラリ群を使うために、一度ビルドされている atmark-dist が必要です。まず、atmark-dist がターゲットボード用にコンフィグかつビルドされていることを確認してください。

10.1.2. ソースコードの用意

次に、開発するアプリケーション用のディレクトリを atmark-dist のディレクトリ構造の外に作ってください。この中には Makefile と必要な C のソースコードやヘッダファイルが入ります。

```
[PC ~]$ ls
atmark-dist
[PC ~]$ mkdir hello
[PC ~]$ ls
hello  atmark-dist
[PC ~]$ ls hello
Makefile  hello.c
```

hello.c は、図のようにどの C の教科書にでもでてくるような簡単なものです。

```
#include <stdio.h>

int main(int argc, char * argv[])
{
        printf( “Hello World¥n” );
        return 0;
}
```

Makefile は図のようなものを使用します。

```
# ROOTDIR=/usr/src/atmark-dist ---------------------------------------- ①
ifndef ROOTDIR
ROOTDIR=../atmark-dist
endif
ROMFSDIR  = $(ROOTDIR)/romfs
ROMFSINST = romfs-inst.sh
```

```
PATH      := $(PATH):$(ROOTDIR)/tools

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


EXEC = hello ------------------------------------------------------------ ②
OBJS = hello.o ---------------------------------------------------------- ③


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 $@ $<
```

この Makefile は"Hello World"以外のアプリケーションを開発するときにもテンプレートとして使用することができます。環境に合わせて変更する点は以下の 3 つです。

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 atmark-dist ディレクトリがあると仮定します。他の場所に atmark-dist がある場合は、この行のコメントを外して適切なディレクトリ名に変更してください。
② 生成される実行ファイル名を指定します。ここでは、hello とします。
③ 生成される実行ファイルが依存するオブジェクトファイルを指定します。ここでは hello.o を指定します。

10.1.3. ビルド（uClinux）

Makefileと hello.c が用意できたら、helloをビルドします。ビルドには makeコマンドを使用します。ビルドが完了すると実行ファイル hello がディレクトリ内に生成されています。(uClinux の場合。Linux の場合生成されるファイルが異なります。)

```
[PC ~/hello]$ make
    :
    :
[PC ~/hello]$ ls hello*
hello  hello.c  hello.gdb  hello.o
[PC ~/hello]$ file hello*
hello:     BFLT executable - version 4 ram
hello.c:   ASCII C program text
```

```
hello.gdb: ELF 32-bit MSB executable, version 1 (SYSV), statically linked, not stripped
hello.o:   ELF 32-bit MSB relocatable, version 1 (SYSV), not stripped
```

hello.gdb は ELF フォーマットの実行ファイルです。デバッグに使います。hello は Flat Binary フォーマットと呼ばれる uClinux 専用のバイナリフォーマットです。

10.1.4. インストール

実行ファイルを atmark-dist の romfs ディレクトリにインストールするために、make コマンドで romfs ターゲットを指定します。

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

10.1.5. image ファイルの作成

make romfs の後 atmark-dist のディレクトリに移動して、make image を実行することで、hello を含んだ ターゲットボード用のイメージファイルが image ディレクトリに生成されます。

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

image ターゲットについては、「7.7.5.image」を参照してください。

10.1.6. 複数のソースコード

実行ファイルが複数のソースコードに分割されている場合は、Makefile を変更することで対応できます。ここでは、hello.c と print.c から実行ファイル hello を生成する場合を例に説明します。

```
[PC ~/hello]$ ls
Makefile  hello.c  print.c
```

hello.c

```
#include <stdio.h>

extern void print_hello(char * string);

int main(int argc, char * argv[])
{
        print_hello( “World” );
        return 0;
}
```

print.c

```
void print_hello(char * string)
{
        printf( “Hello %s¥n”, string);
}
```

Makefile の OBJS に print.o を追加します。

```
# ROOTDIR=/usr/src/atmark-dist
ifndef ROOTDIR
ROOTDIR=../atmark-dist
endif
ROMFSDIR  = $(ROOTDIR)/romfs
ROMFSINST = romfs-inst.sh
PATH      := $(PATH):$(ROOTDIR)/tools

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


EXEC = hello
OBJS = hello.o print.o ---------------------------------- print.o を追加


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 $@ $<
```

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

10.1.7. pthread 対応

スレッドを使うアプリケーションの場合は、スレッド用のライブラリをリンクする必要があります。Makefile の一部を以下のように変更します。

```
# ROOTDIR=/usr/src/atmark-dist
ifndef ROOTDIR
ROOTDIR=../atmark-dist
endif
ROMFSDIR  = $(ROOTDIR)/romfs
ROMFSINST = romfs-inst.sh
PATH      := $(PATH):$(ROOTDIR)/tools

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


EXEC = hello
OBJS = hello.o


all: $(EXEC)

$(EXEC): $(OBJS)
	$(CC) $(LDFLAGS) -o $@ $(OBJS) $(LIBPTHREAD) $(LDLIBS) ------------ ①

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

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

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

① $(LIBPTHREAD)を追加する

10.2. プロダクト別のアプリケーション

組み込み機器の場合、プロダクト固有のアプリケーションは、必ずと言っていいほど存在すると思います。ここでは、プロダクト固有のアプリケーションを atmark-dist に統合する方法を紹介します。

10.2.1. ディレクトリの準備

atmark-dist/vendors/$(CONFIG_VENDOR)/$(CONFIG_PRODUCT)以下に、アプリケーション用のディレクトリを作成します。$(CONFIG_VENDOR)と$(CONFIG_PRODUCT)は、お使いのボードにあわせて変更してください。ここでは、ベンダーを AtmarkTechno、プロダクトを SUZAKU、そしてアプリケーションを hello とします。

```
[PC ~]$ mkdir atmark-dist/vendors/AtmarkTechno/SUZAKU/hello
```

```
[PC ~]$ ls - d atmark-dist/vendors/AtmarkTechno/SUZAKU/hello
hello
```

10.2.2. ソースコードの用意

Cのソースコードは前項で使った hello.cを使用します。Makefileは Out of Tree Compileで使用したものよりもシンプルになります。

```
EXEC = hello
OBJS = hello.o


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 $@ $<
```

10.2.3. 追加アプリケーションの設定

追加したアプリケーションをプロダクトMakefileに追加します。プロダクトMakefileには、DIRSと呼ばれる変数が用意されています。ここに、アプリケーションのディレクトリ名を追加します。

```
    :
    :
DIRS = hello
    :
    :
```

10.2.4. ビルド

ビルド方法は「7.6.ビルド」と同じです。all、clean、romfsターゲットは、以下のように定義されているので、DIRSで指定したディレクトリすべてに対して、適切なコマンドを実行するようになっています。

```
all:
	dirs=$(DIRS); ¥
	for i in $$dirs; do $(MAKE) - C $$i clean || exit $? ; done

clean:
	-dirs=$(DIRS); ¥
	for i in $$dirs; do [ ! - d $$I ] || $(MAKE) - C $$i clean; done
```

```
romfs:
        :
        :
        dirs=$(DIRS); ¥
        for I in $$dirs; do $(MAKE) - C $$i romfs || exit $?; done
```

10.3. user ディレクトリへのマージ

最初はプロダクト固有のアプリケーションとして開発されたものでも、多くのプロダクトで使われるようになることがあります。そんな時は、アプリケーションを user ディレクトリに移動し、プロダクト間で共有することができます。

10.3.1. ディレクトリの準備

atmark-dist/user 以下に、アプリケーション用のディレクトリを作成します。ここでは、hello とします。

```
[PC ~]$ mkdir atmark-dist/user/hello
[PC ~]$ ls - d atmark-dist/user/hello
hello
```

10.3.2. ソースコードの用意

Cのソースコードおよび Makefile は「10.2.プロダクト別のアプリケーション」の章で使ったものと同じものを使用します。

10.3.3. 追加アプリケーションの設定

変更個所は、atmark-dist/config/config.in と atmark-dist/user/Makefile です。この例では、追加するアプリケーションを Miscellaneous Application に追加します。他にならってアルファベット順に並べます。

例 10-1 atmark-dist/config/config.in の変更点

```
--- config.in.orig      2004-04-18 04:03:57.000000000 +0900
+++ config.in  2004-05-26 17:58:38.000000000 +0900
@@ -515,6 +515,7 @@
 bool 'gdbreplay'                CONFIG_USER_GDBSERVER_GDBREPLAY
 bool 'gdbserver'                CONFIG_USER_GDBSERVER_GDBSERVER
 bool 'hd'                       CONFIG_USER_HD_HD
+bool 'hello'                     CONFIG_USER_HELLO_HELLO
 bool 'lcd'                      CONFIG_USER_LCD_LCD
 bool 'ledcon'                   CONFIG_USER_LEDCON_LEDCON
 bool 'lilo'                     CONFIG_USER_LILO_LILO
```

例 10-2 atmark-dist/user/Makefile の変更点

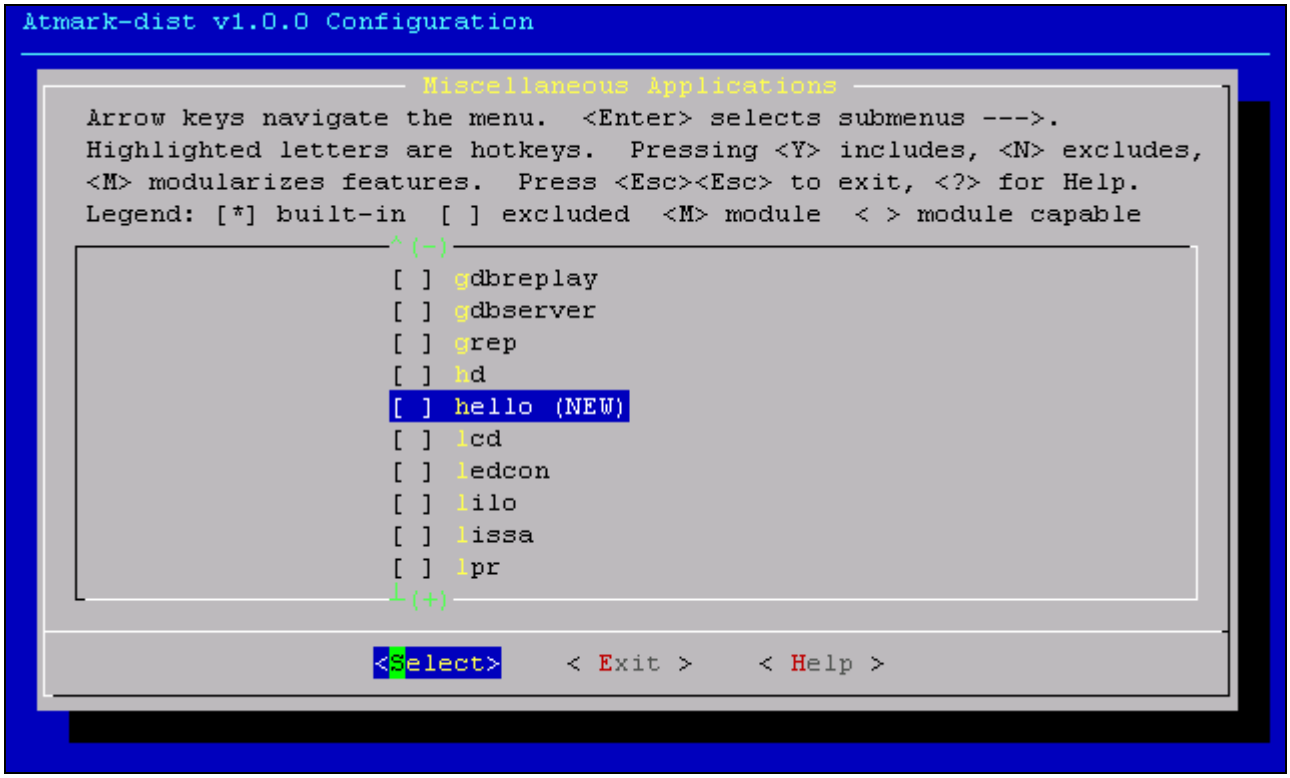
```
--- Makefile.orig       2004-02-20 13:22:55.000000000 +0900
+++ Makefile    2004-05-26 17:56:09.000000000 +0900
@@ -123,6 +123,7 @@
 dir_$(CONFIG_USER_GDBSERVER_GDBSERVER)      += gdbserver
 dir_$(CONFIG_USER_GETTYD_GETTYD)            += gettyd
 dir_$(CONFIG_USER_HD_HD)                    += hd
+dir_$(CONFIG_USER_HELLO_HELLO)              += hello
```

```
dir_$(CONFIG_USER_HOSTAP_HOSTAP)        += hostap
dir_$(CONFIG_USER_HTTPD_HTTPD)          += httpd
dir_$(CONFIG_USER_HWCLOCK_HWCLOCK)      += hwclock
```

10.3.4. アプリケーションの選択

make menuconfig などで追加したアプリケーションが Miscellaneous Application セクションに表示されるか確認してください。表示された hello を選択し、設定を保存します。

図 10-1メニューに追加された hello



10.3.5. ビルド

In Tree コンパイルのビルド方法は「7.6.ビルド」と同じです。生成されたイメージファイルをターゲットボードに転送し、hello が動作するか確認してください。

10.3.6. コンフィグの命名規則

今回使用したコンフィグ名は CONFIG_USER_HELLO_HELLO です。

atmark-dist ではユーザランドアプリケーションの選択にはディレクトリ名とアプリケーション名を使う決まりになっています。

- すべてのコンフィグオプションは、CONFIG_からはじめる
- atmark-dist/user 以下のものは、CONFIG_USER_からはじめる
- ディレクトリ名が hello なので、CONFIG_USER_HELLO_からはじめる
- 最後にアプリケーション名をつけて CONFIG_USER_HELLO_HELLO となる

10.3.7. 複数のアプリケーション

ひとつのディレクトリで複数のアプリケーションを開発する方法を紹介します。In Tree コンパイルに限った話ではないので、Out of Tree コンパイルでも可能です。

atmark-dist/user/hello のディレクトリに hello2 という名前のアプリケーションを追加します。コンフィグの文字列は、CONFIG_USER_HELLO_HELLO2 とします。

まずは、atmark-dist/config/config.in と、atmark-dist/user/Makefile の変更点です。

例 10-3 atmark-dist/config/config.in の変更点(複数アプリケーション)

```
--- config.in.orig      2004-04-18 04:03:57.000000000 +0900
+++ config.in  2004-05-26 17:58:38.000000000 +0900
@@ -515,6 +515,8 @@
 bool 'gdbreplay'                CONFIG_USER_GDBSERVER_GDBREPLAY
 bool 'gdbserver'                CONFIG_USER_GDBSERVER_GDBSERVER
 bool 'hd'                       CONFIG_USER_HD_HD
+bool 'hello'                     CONFIG_USER_HELLO_HELLO
+bool 'hello2                     CONFIG_USER_HELLO_HELLO2
 bool 'lcd'                      CONFIG_USER_LCD_LCD
 bool 'ledcon'                   CONFIG_USER_LEDCON_LEDCON
 bool 'lilo'                     CONFIG_USER_LILO_LILO
```

例 10-4 atmark-dist/user/Makefile の変更点(複数アプリケーション)

```
--- Makefile.orig       2004-02-20 13:22:55.000000000 +0900
+++ Makefile    2004-05-26 17:56:09.000000000 +0900
@@ -123,6 +123,8 @@
 dir_$(CONFIG_USER_GDBSERVER_GDBSERVER)      += gdbserver
 dir_$(CONFIG_USER_GETTYD_GETTYD)            += gettyd
 dir_$(CONFIG_USER_HD_HD)                    += hd
+dir_$(CONFIG_USER_HELLO_HELLO)              += hello
+dir_$(CONFIG_USER_HELLO_HELLO2)             += hello
 dir_$(CONFIG_USER_HOSTAP_HOSTAP)            += hostap
 dir_$(CONFIG_USER_HTTPD_HTTPD)              += httpd
 dir_$(CONFIG_USER_HWCLOCK_HWCLOCK)          += hwclock
```

Makefile では、左側にコンフィグオプションを、+=の右側にはディレクトリ名を書きます。このため、hello も hello2 も指定するディレクトリ名は同じ hello になります。

続いて、atmark-dist/user/hello/Makefile です。今回はアプリケーション名を変数に入れずに直接扱ってみます。

例 10-5 Makefile(複数アプリケーション)

```
OBJS_HELLO = hello.o
OBJS_HELLO2 = hello2.o


all: hello hello2

hello: $(OBJS_HELLO)
	$(CC) $(LDFLAGS) -o $@ $(OBJS_HELLO2) $(LDLIBS)

hello2: $(OBJS_HELLO2)
	$(CC) $(LDFLAGS) -o $@ $(OBJS_HELLO2) $(LDLIBS)

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

romfs:
	$(ROMFSINST) - e CONFIG_USER_HELLO_HELLO /bin/hello
	$(ROMFSINST) - e CONFIG_USER_HELLO_HELLO2 /bin/hello2


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

romfs ターゲットで、"-e CONFIG_USER_HELLO"を使っている点に注目してください。

atmark-dist のビルドシステムは、hello ディレクトリにあるアプリケーションがひとつでも選択されると、ディレクトリ内すべてのアプリケーションをビルドするように指定します。このため、romfs ターゲットで条件によってインストールするアプリケーションを選択しなければなりません。上記のように、コンフィグオプションによる条件分岐を行い、インストールするアプリケーションを決定します。

romfsinst.sh の詳しい説明は、「9.romfs インストールツール」の使い方を参照してください。

11. 複数カーネルの利用

複数の Linux カーネルソースコードを使って開発することも可能です。その場合は、以下のようにシンボリックリンクを切り替えることで対応できます。

```
[PC ~]$ ls
kernel/   atmark-dist/
[PC ~]$ ls kernel
linux-foo/   linux-bar/
[PC ~]$ cd atmark-dist
[PC ~/atmark-dist]$ ln -s ../kernel/linux-foo linux-2.4.x
[PC ~/atmark-dist]$ ls -l linux-2.4.x
lrwxrwxrwx           linux-2.4.x -> ../kernel/linux-foo
```

12. 特有なアプリケーションの説明

atmark-dist には、一般の Linux には無いアプリケーションも含まれています。ここではその中でも特に組込み用途に合ったアプリケーションを紹介します。

12.1. NetFlash

NetFlash はネットワーク経由でイメージファイルをダウンロードし、Flash メモリに書き込むためのアプリケーションです。
組み込みシステムにおいて、ネットワーク経由によるシステムのアップグレードは、保守性とユーザの利便性の両面から非常に重宝されますが、NetFlash によって簡単に実現できます。

NetFlash がファイルのダウンロードに利用できる通信プロトコルは、http、ftp、tftp です。このため、NetFlash を実行するには、http、ftp、tftp のいずれかのプロトコルで通信が行えるサーバが必要となります。
特にオプションを指定せずに NetFlash を実行した場合には、以下の手順で Flash メモリの書き込み処理が行われます。

1. 全プロセスの終了
2. 指定されたファイルを指定されたプロトコルでサーバからダウンロード
3. ダウンロードしたファイルのチェックサムを確認
4. Flash メモリへの書き込み
5. システムのリブート

オプションを指定する事で、これらの処理を細かく制御する事が可能です。参考として、コマンドの実行例と NetFlash のヘルプを以下に示します。

例 12-1 netflash の実行

```
[Target /]# netflash http://embedded-server/images/base.img
netflash: killing tasks...
...
netflash: got "http://myserver.local/images/image.bim", length=4194304
 :
 :
```

例 12-2 netflash のヘルプ

```
[Targt /]# netflash -h
usage: netflash [-bCfFhijklntuv?] [-c <console-device>] [-d <delay>] [-o <offset
>] [-r <flash-device>] [<net-server>] <file-name>

        -b      don't reboot hardware when done
        -C      check that image was written correctly
        -f      use FTP as load protocol
        -F      force overwrite (do not preserve special regions)
        -h      print help
        -i      ignore any version information
        -H      ignore hardware type information
```

```
        -j      image is a JFFS2 filesystem
        -k      don't kill other processes
                (ignored when root filesystem is inside flash)
        -l      lock flash segments when done
        -n      file with no checksum at end (implies no version information)
        -p      preserve portions of flash segments not actually written.
        -s      stop erasing/programming at end of input data
        -t      check the image and then throw it away
        -u      unlock flash segments before programming
        -v      display version number
```

処理の途中でエラーが発生した場合は、Flash メモリへの書き込みを行わずに処理を中断します。

書き込みの途中で電源断が発生した場合には、最悪システムが起動しなくなる可能性があるので、NetFlash を実行する際には十分な注意が必要です。

12.2. Flatfsd

組み込み機器でよく求められる機能に、設定情報を初期化することなくファームウェアだけをアップグレードする機能があります。この機能を実現するためには、ユーザ設定を保持するための小さな領域を割り当てたファイルシステムを構築するのが一般的です。

このような用途に適したファイルシステムに Flat Filesystem があります。Flat Filesystem は、1 セクタからファイルシステムを構築することができます。また、シンプルな作りであること、安定した動作実績があることが特長です。
このファイルシステムを実現するためのアプリケーションが flatfsd です。

flatfsd は、ramfs にマウントされた/etc/config ディレクトリの内容を/dev/flash/config デバイスファイルに読み書きします。/dev/flash/config は、設定ファイルを保存すべきデバイスのメジャー番号とマイナー番号を指定して作成します。

```
[Target /]# ls -l /dev/flash/
crw-------    1 root     root      90,  14 Jan  1 09:00 config
crw-------    1 root     root      90,   6 Jan  1 09:00 image
[Target /]#
```

以前保存した設定ファイル情報を復元するためには、flatfsd -r コマンドを実行します。
flatfsd -r コマンドは、/dev/flash/config 内の以前保存したファイル情報を読み出して、/etc/config にコピーします。

```
[Target /]# flatfsd -r
FLATFSD: created 6 configuration files (507 bytes)
[Target /]#
```

この時、記録されているチェックサムよりファイル情報の整合性を確認し、異常だった場合には、/etc/default の内容で/etc/config ディレクトリを初期化します。通常、システムの起動時に flatfsd -r コマンドを実行します。

設定ファイルに加えた変更を Flash メモリ内に記録するためには、flatfsd プロセスに SIGUSR1 シグナルを送信します。このため、起動時などに flatfsd コマンドを実行してプロセスを立ち上げておく必要があります。起動している flatfsd のプロセス ID は/var/run/flatfsd.pid ファイルから取得できます。以下の例では、killall コマンドを使用し、flatfsd にシグナルを送る方法です。

```
[Target /]# killall - USR1 flatfsd
[Target /]#
```

SIGUSR1 シグナルを受信した flatfsd プロセスは、/etc/config ディレクトリの内容を/dev/flash/config に書き込みます。この時、整合性の確認が行えるよう、チェックサムを計算して記録します。

atmark-dist の config で flatfsd アプリケーションを選択すると、dhcpcd や passwd などのアプリケーションは、/etc ではなく/etc/config に設定ファイルを用意します。このため、設定の変更が次回起動時まで保存されています。

Flat Filesystem では flatfsd が/dev/flash/config デバイスにデータを書き込んでいる最中に電源が切断されると、保存していたデータを消失する可能性があります。
また、Flash メモリの書き込み保証回数は、およそ 10 万回ですので注意してください。

改訂履歴

Ver.	年月日	改訂内容
1.0	2005/4/20	初版作成

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 6F TEL011-207-6550 FAX011-207-6570