

Armadillo 実践開発ガイド

～組み込み Linux の導入から製品化まで～

第 1 部

Version 1.0.0
2010/10/19

Armadillo 実践開発ガイド: ~組み込み Linux の導入から製品化まで~: 第 1 部

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
TEL 011-207-6550 FAX 011-207-6570

製作著作 © 2010 Atmark Techno, Inc.

Version 1.0.0
2010/10/19

目次

- 1. はじめに 10
 - 1.1. 対象読者 10
 - 1.2. 本書の構成 10
 - 1.3. 表記方法 11
 - 1.3.1. 使用するフォント 11
 - 1.3.2. コマンド入力例の表記方法 11
 - 1.3.3. コラムの表記方法 12
 - 1.4. サンプルソースコード 12
 - 1.5. 困った時は 12
 - 1.6. お問い合わせ先 13
 - 1.7. 商標 13
 - 1.8. ライセンス 13
 - 1.9. 謝辞 14
- 2. 組み込み Linux システムとは 15
 - 2.1. Linux システムとは 15
 - 2.2. 組み込み Linux システムとは 17
- 3. Armadillo を使った組み込みシステム開発 19
 - 3.1. Armadillo シリーズの概要 19
 - 3.2. Armadillo-400 シリーズ 22
 - 3.2.1. Armadillo-400 シリーズの基本仕様 22
 - 3.2.2. Armadillo-400 シリーズで出来ること 25
 - 3.3. Armadillo の開発環境 30
 - 3.4. Armadillo を採用した場合の製品開発サイクル 31
 - 3.4.1. 検討、試作 31
 - 3.4.2. 設計、開発 32
 - 3.4.3. 量産 32
 - 3.4.4. 保守 32
- 4. Armadillo の基本操作 33
 - 4.1. 取り扱い上の注意事項 33
 - 4.2. Armadillo と作業用 PC との接続 33
 - 4.2.1. 準備するもの 33
 - 4.2.2. 接続方法 34
 - 4.2.3. シリアル通信ソフトウェアの設定 35
 - 4.3. Armadillo の起動 41
 - 4.4. ログイン 42
 - 4.5. プロンプト 43
 - 4.6. 動作の停止と電源オフ 44
 - 4.7. ディレクトリとファイルの操作 45
 - 4.7.1. ディレクトリとファイル 45
 - 4.7.2. ディレクトリとファイルを操作するコマンド 45
 - 4.8. ファイルの編集 / vi エディタ 50
 - 4.8.1. vi の起動 50
 - 4.8.2. 文字の入力 50
 - 4.8.3. カーソルの移動 51
 - 4.8.4. 文字の削除 51
 - 4.8.5. 保存と終了 52
 - 4.9. ネットワークを使う 52
 - 4.9.1. ネットワークの設定 52
 - 4.9.2. vi エディタを使用したネットワーク設定 53
 - 4.9.3. ネットワーク設定の反映 54

4.9.4. PC から Armadillo の Web サーバーにアクセスする	55
4.10. 変更した Armadillo の設定を保存する	55
5. Armadillo が動作する仕組み	57
5.1. ソフトウェア構成	57
5.1.1. ブートローダー	57
5.1.2. Linux カーネル	57
5.1.3. ユーザーランド	57
5.2. 起動の仕組み	58
5.2.1. カーネルが動作するまで	59
5.2.2. カーネルの初期化処理	62
5.2.3. ユーザーランドの初期化処理	63
5.3. ルートファイルシステムのディレクトリ構成	65
5.3.1. 実行ファイル	66
5.3.2. ホームディレクトリ	66
5.3.3. ライブラリ	66
5.3.4. デバイスファイル	66
5.3.5. プロセス、システムの状態	67
5.3.6. ログ	67
5.3.7. 設定ファイル	67
6. 開発環境の構築	68
6.1. Windows PC 上に ATDE を構築する	69
6.1.1. インストールの前に	69
6.1.2. VMware Player のインストール	69
6.1.3. ATDE の起動	73
6.1.4. ATDE の終了	76
6.1.5. シリアルポートの設定	78
6.1.6. 共有フォルダの設定	79
6.2. Linux 上に ATDE を構築する	83
6.2.1. インストールの前に	83
6.2.2. VMware Player のインストール	83
6.2.3. ATDE の起動	85
6.2.4. ATDE の終了	89
6.2.5. Linux 上でのシリアルポートの設定	91
6.2.6. 共有フォルダの設定	93
6.3. ATDE のネットワーク設定	97
6.3.1. DHCP 接続の設定	97
6.3.2. 固定 IP アドレス接続の設定	98
6.4. 最新の状態にアップデートする	100
7. 開発の基本的な流れ	101
7.1. アプリケーションプログラムの作成	101
7.1.1. Hello World!	101
7.1.2. ライブラリとヘッダファイル	105
7.1.3. make	107
7.2. Atmark Dist を使ったルートファイルシステムの作成	111
7.2.1. ソースコードの取得	111
7.2.2. 独自プロダクトの追加	112
7.2.3. 基本的なコンフィギュレーション	113
7.2.4. ルートファイルシステムのビルドとイメージファイルの作成	117
7.2.5. イメージファイルの書き込み	117
7.3. ルートファイルシステムのカスタマイズ	119
7.3.1. アプリケーションプログラムの追加	119
7.3.2. ファイルの追加	120
7.3.3. コンフィギュレーションの変更	121

7.4. 量産に向けた準備	121
7.4.1. カスタマイズサービスを利用する	121
7.4.2. ライセンスに関する注意事項	122
7.5. 製品出荷後のメンテナンス	122
7.5.1. ソフトウェアアップデートへの対応	122
7.5.2. ハードウェアの変更通知	122

目次

- 1.1. コマンド入力表記例 11
- 1.2. クリエイティブコモンズライセンス 14
- 2.1. Linux システムアーキテクチャ 15
- 3.1. Armadillo ロゴ 19
- 3.2. Armadillo シリーズ 21
- 3.3. Armadillo-400 シリーズブロック図 23
- 3.4. Armadillo-420 ベーシックモデル見取り図 24
- 3.5. Armadillo-440 液晶モデル見取り図 25
- 3.6. Armadillo-400 シリーズで出来ること 26
- 3.7. Armadillo を採用した場合の開発の流れ 31
- 4.1. Armadillo-440 液晶モデル接続例 35
- 4.2. Armadillo-420 ベーシックモデル接続例 35
- 4.3. Tera Term: 新しい接続画面 36
- 4.4. Tera Term 画面 37
- 4.5. Tera Term: シリアルポート設定画面 37
- 4.6. デバイスマネージャー画面 38
- 4.7. Tera Term 画面 38
- 4.8. minicom の設定の起動 39
- 4.9. minicom の設定 39
- 4.10. minicom のシリアルポートの設定 40
- 4.11. 例. USB to シリアル変換ケーブル接続時のログ 40
- 4.12. minicom のシリアルポートのパラメータの設定 41
- 4.13. minicom シリアルポートの設定値 41
- 4.14. 起動ログ 41
- 4.15. ユーザー名とパスワードを入力してログインする 42
- 4.16. Armadillo-440 でのプロンプト表示例 43
- 4.17. コマンドの書式 43
- 4.18. echo コマンドの書式 43
- 4.19. echo コマンドの実行例 43
- 4.20. 入力モードに移行するコマンドの説明 51
- 4.21. 文字を削除するコマンドの説明 52
- 4.22. ネットワーク構成例 53
- 4.23. 設定変更後の interfaces ファイル 53
- 4.24. interfaces ファイルを開く 53
- 4.25. 標準イメージの interfaces ファイル 54
- 4.26. interfaces ネットワーク設定 54
- 4.27. ネットワークの設定を反映させる 54
- 4.28. Web サーバーのトップページ 55
- 5.1. Hermit-At 保守モード時の表示 60
- 5.2. ブートローダーのコピー 60
- 5.3. カーネルとユーザーランドをコピー 61
- 5.4. カーネルの起動 61
- 5.5. Hermit-At オートブートモード時の表示 62
- 5.6. Armadillo-420 カーネルブートログ 62
- 5.7. inittab の書式 63
- 5.8. Armadillo-420 の inittab 64
- 6.1. VMware Player インストール画面 1 70
- 6.2. VMware Player インストール画面 2 70
- 6.3. VMware Player インストール画面 3 71
- 6.4. VMware Player インストール画面 4 71

6.5. VMware Player インストール画面 5	72
6.6. VMware Player インストール画面 6	72
6.7. VMware Player インストール画面 7	73
6.8. ライセンスの確認	73
6.9. VMware Player 画面	74
6.10. 仮想マシンを開く画面	74
6.11. VMware Player 画面	75
6.12. ログイン画面	75
6.13. シャットダウンを選択する	76
6.14. 「このシステムをシャットダウンしますか？」画面	77
6.15. System halted の表示	77
6.16. パワーオフを選択する	77
6.17. 仮想マシンの設定を選択する	78
6.18. 仮想マシン設定画面	79
6.19. VMware Player 画面	80
6.20. 仮想マシン設定画面	81
6.21. 共有フォルダのプロパティ画面	81
6.22. 端末の起動	82
6.23. 共有フォルダをマウントするディレクトリを作成	82
6.24. 共有フォルダをマウントする	82
6.25. zip ファイルの展開	83
6.26. インストーラーの起動	83
6.27. VMware Player インストール画面 1	84
6.28. VMware Player インストール画面 2	84
6.29. VMware Player インストール画面 3	85
6.30. VMware Player インストール画面 4	85
6.31. VMware Player を起動	86
6.32. ライセンスの確認	86
6.33. VMware Player 画面	87
6.34. Open Virtual Machine 画面	87
6.35. VMware Player 画面	88
6.36. ログイン画面	88
6.37. シャットダウンを選択する	89
6.38. 「このシステムをシャットダウンしますか？」画面	90
6.39. System halted の表示	90
6.40. Power Off を選択する	91
6.41. Settings を選択する	92
6.42. Virtual Machine Settings 画面	93
6.43. VMware Player 画面	94
6.44. Virtual Machine Settings 画面	95
6.45. Shared Folder Properties 画面	95
6.46. 端末の起動	96
6.47. 共有フォルダをマウントするディレクトリを作成	96
6.48. 共有フォルダをマウントする	96
6.49. ATDE のネットワーク構成	97
6.50. ネットワーク設定ファイルを開く	98
6.51. DHCP 接続の interfaces ファイル設定例	98
6.52. ネットワーク設定を反映させる	98
6.53. ネットワーク設定ファイルを開く	99
6.54. 固定 IP アドレス接続の interfaces ファイル設定例	99
6.55. ネットワーク設定ファイルを開く	99
6.56. DNS サーバー設定例	99
6.57. ネットワーク設定を反映させる	100

7.1. hello.c	102
7.2. hello.c をコンパイルするコマンド	102
7.3. hello の実行結果	103
7.4. hello.c をクロスコンパイルするコマンド	103
7.5. クロスコンパイルした hello の実行結果(ATDE 上)	103
7.6. lftp をインストールするコマンド	104
7.7. Armadillo へのファイル転送	104
7.8. クロスコンパイルした hello の実行結果(Armadillo 上、実行権限なし)	104
7.9. クロスコンパイルした hello の実行結果(Armadillo 上、実行権限あり)	105
7.10. sin 関数のプロトタイプ	105
7.11. sin.c	106
7.12. sin.c をコンパイルするコマンド	106
7.13. sin.c をコンパイルするコマンド(-lm オプション付き)	106
7.14. sin の実行結果	106
7.15. sin.c をクロスコンパイルするコマンド	106
7.16. sin の実行結果(Armadillo 上)	107
7.17. makefile のルール	107
7.18. sin.c をビルドする Makefile	108
7.19. make コマンドの実行結果	109
7.20. make コマンドの再実行結果	109
7.21. make clean コマンドの実行結果	110
7.22. sin.c をビルドする Makefile(クロスコンパイル対応版)	110
7.23. ソースアーカイブの取得	112
7.24. ソースアーカイブの展開	112
7.25. armak-dist ディレクトリのシンボリックリンクの作成	112
7.26. armak-dist ディレクトリのシンボリックリンクの作成	112
7.27. 独自プロダクトの追加	113
7.28. make menuconfig コマンドの実行	113
7.29. menuconfig: Main Menu 画面	114
7.30. menuconfig: Vendor/Product Selection 画面	114
7.31. menuconfig: Vendor 画面	115
7.32. menuconfig: Vendor/Product Selection 画面	115
7.33. menuconfig: AtmarkTechno Products 画面	116
7.34. menuconfig: Do you wish to save your new kernel configuration?	116
7.35. ルートファイルシステムのビルドとイメージファイルの作成の開始	117
7.36. 作成されたイメージファイル	117
7.37. イメージファイルのコピー	117
7.38. Hermit-At のプロンプト	118
7.39. tftpd command	118
7.40. アプリケーションプログラム用ディレクトリの作成	119
7.41. Atmark Dist で sin.c をビルドする Makefile	120
7.42. 追加したディレクトリをビルド対象に含めるための Makefile の修正箇所	120

表目次

- 1.1. 使用するフォント 11
- 1.2. コマンドの実行環境と対応する表記 11
- 1.3. ユーザーの種類と対応する表記 11
- 3.1. Armadillo シリーズの発表時期と CPU 20
- 3.2. Armadillo-400 シリーズ基本仕様 22
- 3.3. Armadillo-400 シリーズ拡張インターフェース 23
- 3.4. Armadillo-420 ベーシックモデル フラッシュメモリ メモリマップ 24
- 3.5. Armadillo-440 液晶モデル フラッシュメモリ メモリマップ 25
- 3.6. Armadillo-400 シリーズで使用可能なインターフェース 26
- 3.7. Armadillo-400 シリーズで実現可能なソフトウェア機能一覧 29
- 3.8. Armadillo-400 シリーズで使用可能なソフトウェア製品 30
- 4.1. 必要な機材 34
- 4.2. シリアル通信設定 37
- 4.3. シリアル通信設定 39
- 4.4. シリアルコンソールログイン時のユーザー名とパスワード 42
- 4.5. ディレクトリとファイル操作に関するコマンド 45
- 4.6. 入力モードに移行するコマンド 51
- 4.7. カーソルの移動コマンド 51
- 4.8. 削除コマンド 52
- 4.9. 保存・終了コマンド 52
- 4.10. 固定 IP アドレス設定例 53
- 4.11. flagfsd の主な引数 56
- 5.1. ジャンパ設定 60
- 5.2. init の action に指定可能な値 63
- 5.3. ディレクトリ構成 65
- 6.1. ATDE3 のユーザ名とパスワード 75
- 6.2. デフォルトのユーザ名とパスワード 89
- 6.3. 固定 IP アドレス設定例 98
- 7.1. Atmark Dist と Linux カーネルソースコードのダウンロード URL 111
- 7.2. 使用する Atmark Dist と Linux カーネルソースコードのバージョン 111
- 7.3. menuconfig の操作方法 113
- 7.4. カスタマイズサービスの種類 122

1. はじめに

Linux が動く ARM CPU 搭載の汎用ボードコンピュータというコンセプトで、初代 Armadillo (HT1070) が発売されたのは 2001 年 11 月のことでした。発売当初は、「ARM よりも SuperH で作ってほしい」、「組み込み機器で Linux を動かして意味があるの?」、「リアルタイム OS でなくて良いの?」といった、どちらかというとな否定的な意見が多かったと聞きます^[1]。

初代 Armadillo の発売から約 10 年の月日が経ち、Linux+ARM という組み合わせは、携帯電話を初めとして、もはや当たり前前の選択となってきました。それに伴い、Armadillo シリーズも毎年のように新製品を発表し、多くのユーザーに支えられて成長してきました。

ユーザー数が増えるにつれ、「組み込みで Linux を使うにあたり、系統立てて学ぶ方法はないか?」といったご相談を受けることが多くなりました。

Linux で組み込みシステムの開発を行うにあたっては、Linux そのものの使いかた、Linux の仕組みについての理解、クロス開発についての理解、そして、ターゲットとなるボードごとの知識の習得など、多くの課題があります。しかしながら、それらを体系的に解説している書籍などは、なかなか見つからないのが現状です。

本書は、そういった問題を少しでも解決できればと思い、執筆しました。

本書では、何らかの組み込みシステムを開発したいと考えているユーザーが、Armadillo を使用してシステムを構築し、量産につなげるために必要な一連の手順を示します。ユーザーは、上から順番に読んでいくとシステムを開発するときに行うべき手順を把握できます。また、Armadillo を使う上でのノウハウを詰め込んでありますので、開発で行き詰まった時にリファレンスとして活用することもできます。

本書が、皆様の開発のお役に立てば幸いです。

1.1. 対象読者

本書が主な対象読者としているのは、Armadillo を使って組み込みシステムを開発したいと考えているソフトウェア開発者です。ソフトウェア開発者は、少なくとも C 言語での開発経験が必要です。ただし、Linux の使用経験はなくても読み進められるように配慮してあります。

また、本書は、Armadillo と組み込み Linux の組み合わせでどのようなことが実現可能か知りたいと願っている設計者、企画者も対象としています。Armadillo は、汎用ボードコンピュータなので、標準で有効になっている機能以外にも様々な機能を実現することができます。どのようなことができて、できないことは何なのかについても説明しています。

1.2. 本書の構成

本書は、大きく分けて二部構成になっています。

第 1 部では、Armadillo を使った組み込みシステム開発というものがどういうものか、一連の流れとして説明します。これまで組み込みシステム開発の経験がない方や、Linux での開発経験がない方も理解できるよう、基本的な用語や操作方法から説明します。第 1 部を読了すると、開発を始めるための基本的な知識を習得し、開発の全体的な流れを把握できるようになります。

第 2 部では、実践的な開発に役立つ事柄について説明します。効率的に開発するための開発環境の整備、Linux の仕組みについてのより詳細な説明、組み込みや Armadillo 特有のプログラミング技法、シ

^[1]特集:最新組み込み Linux 実践講座 Part 1 <http://armadillo.atmark-techno.com/articles/sd-a500-embedded-course-ch1>

システム構築上のノウハウなどについて述べます。第 2 部を読了すると、実際の開発で直面するであろう、種々の問題を解決できるようになっているでしょう。

1.3. 表記方法

本書で使用している表記方法について説明します。

1.3.1. 使用するフォント

フォントは以下のものを使用します。

表 1.1 使用するフォント

フォント例	使用箇所
本文中のフォント	本文
等幅	コンソールやソースコード
太字	ユーザーが入力する文字
斜体	状況によって置き換えられるもの
下線	キー入力

1.3.2. コマンド入力例の表記方法

コマンド入力例は、以下のように表記します。

```
[PC ~/]$ ls
```

図 1.1 コマンド入力表記例

「[PC ~/]\$」の部分をプロンプトと呼びます。ユーザーは、プロンプトに続けてコマンドを入力します。

「PC」の部分は、コマンドを実行する環境によって使い分けます。実行環境には、以下のものがあります。

表 1.2 コマンドの実行環境と対応する表記

表記	実行環境
PC	作業用 PC
ATDE	ATDE (Atmark Techno Development Environment ^[1])
armadillo	Armadillo (Atmark Dist で作成したユーザーランドの場合)
darmadillo	Armadillo (ユーザーランドが Debian GNU/Linux の場合)

^[1]アットマークテクノ社製品用の開発環境

「\$」の部分は、コマンドを実行するユーザーの種類によって使い分けます。ユーザーの種類には、以下の二種類があります。

表 1.3 ユーザーの種類と対応する表記

表記	権限
#	特権ユーザー
\$	一般ユーザー

プロンプトの表記方法やそれぞれの用語については、本文中で詳しく説明します。

1.3.3. コラムの表記方法

本書では、随所にコラムを記載しています。コラムの内容によって、以下の表記を用います。



メモ

用語の説明や補足的な説明は、このアイコンで示します。



ヒント

知っていると便利な情報は、このアイコンで示します。



注意

ユーザーの注意が必要な情報は、このアイコンで示します。このアイコンが付いているコラムの内容に従わない場合、以降の作業に支障をきたす場合があります。再度、ご確認ください。



危険

危険な操作に関する情報は、このアイコンで示します。このアイコンが付いているコラムの内容に従わない場合、ハードウェアやシステムを破壊してしまう場合があります。必ず確認してください。

1.4. サンプルソースコード

本書で紹介するサンプルソースコードは、<http://download.atmark-techno.com/armadillo-guide/source/> からダウンロードできます。サンプルソースコードは、MIT ライセンス^[2]の下に公開します。

1.5. 困った時は

もし、本書を読んで困ったことがあった場合は、ぜひ Armadillo 開発者サイト^[3]で情報を探してください。本書に含まれていない FAQ や How To が掲載されています。

^[2]<http://opensource.org/licenses/mit-license.php>

^[3]<http://armadillo.atmark-techno.com>

また、Armadillo 開発者サイトでも情報がみつからない場合は、Armadillo シリーズに関する話題を扱っている Armadillo メーリングリスト^[4]で質問してみてください。同じ問題で困ったことがある人や開発者が参加しているので、情報を集めることができるかもしれません。



メーリングリストに参加する場合の心構え

メーリングリストには数百人の人が参加しています。メーリングリストに送られたメールはメーリングリスト参加者すべてに送られ、アーカイブとして Web 上で誰でも閲覧可能な状態になります。

そのため、通常の対人関係と同様に、受け取り手に対して失礼にならないように配慮する必要があります。

ただし、技術的に簡単なものだからといって、質問することをためらう必要性はまったくありません。

また、適切な質問をすると適切なアドバイスが得られる可能性が高まりますが、その逆もまた然りです。

メーリングリストに不慣れな人は、質問する前に「技術系メーリングリストで質問するときのパターン・ランゲージ」^[5]などに目を通しておくとい良いでしょう。

1.6. お問い合わせ先

本書に関するご意見やご質問は、Armadillo メーリングリスト^[6]にご連絡ください。

何らかの事情でメーリングリストが使えない場合は、以下にご連絡ください。

株式会社アットマークテクノ
〒 060-0035 北海道札幌市中央区北 5 条東 2 丁目 AFT ビル 6F
電話 011-207-6550
FAX 011-207-6570
電子メール sales@atmark-techno.com

1.7. 商標

Armadillo は、株式会社アットマークテクノの登録商標です。その他の記載の商品名および会社名は、各社・各団体の商標または登録商標です。

1.8. ライセンス

本書は、クリエイティブコモンズの表示-改変禁止 2.1 日本ライセンスの下に公開します。ライセンスの内容は <http://creativecommons.org/licenses/by-nd/2.1/jp/> でご確認ください。

^[4]<http://armadillo.atmark-techno.com/maillinglists>

^[5]<http://www.hyuki.com/writing/techask.html>

^[6]<http://armadillo.atmark-techno.com/maillinglists>



図 1.2 クリエイティブコモンズライセンス

1.9. 謝辞

Armadillo 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や関係者の貢献によって成り立っています。この場を借りて感謝の意を表します。

2. 組み込み Linux システムとは

2.1. Linux システムとは

Linux とは、1991 年に Linus Torvalds 氏が公開した、OS(オペレーティングシステム) です。当初は x86 PC 上でしか動作しない、わずか 1 万行程度の小さな OS でした。しかしながら、Linux は驚異的な進化を遂げ、今日では x86 以外にも ARM、PowerPC、MIPS、SH などの様々なアーキテクチャのコンピュータで動作し、組み込みシステムから PC、サーバー、スーパーコンピュータまで様々な用途で使用されています。

厳密にいうと、Linux とはプロセス管理や CPU、メモリなどのリソース管理、デバイスドライバ、プロトコルスタックなどをおこなう OS の中心となる部分(これをカーネルとよびます)だけを指します。一つのシステムとして動作するには、アプリケーションプログラムや各種ライブラリなどのユーザーランドが別途必要です。そのため、カーネル部分だけを意味する場合は Linux カーネルといい、ユーザーランドも含めたシステム全体をさす場合は、Linux システムといいます。

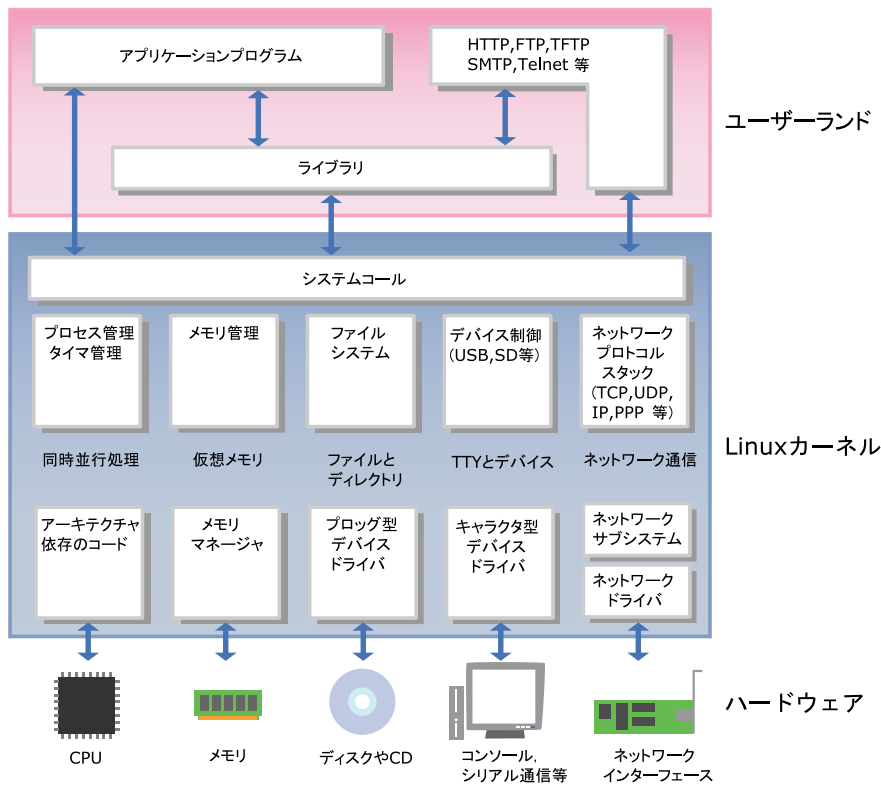


図 2.1 Linux システムアーキテクチャ

Linux カーネルの大きな特徴として、UNIX ライクであることと、オープンソースであることが挙げられます。

UNIX とは、AT&T のベル研究所で開発された OS です。UNIX は当時の OS としては先進的であった、マルチユーザー、マルチタスクという特徴を持っていたため、多くの分野で使用されてきました。その分、様々な派生バージョンが発生することになります。そのため、標準化が必要となり POSIX(Portable Operating System Interface)が制定されました。UNIX ライクな OS とは、POSIX に

沿うよう開発されている OS のことを指します。Linux カーネルも POSIX に沿うよう開発されていますが、すべて準拠しているわけではありません。

オープンソースであるとは、ソフトウェアのソースコードがすべて公開されており、誰でも利用、改変、再配布できることを意味します。Linux カーネルでは、このオープンソースという特徴を維持するために、ソースコードのライセンスとして GPL v2(General Public License version 2)を適用しています。GPL v2 が適用されたソフトウェアでは、ソースコードの改変を自由に行える代わりに、改変したソフトウェアを配布する際には、改変部分を含むソースコードを公開する必要があります。なお、GPL ではソースコードの公開義務は生じますが、そのソフトウェアを商業利用することは禁じていません。GPL に関する詳細な説明は、GNU 一般公衆利用許諾契約書(GNU General Public License)^[1]を参照してください。



GNU/Linux

Linux システムでは、ユーザーランドで動作する基本的なソフトウェアの多くを GNU プロジェクトの成果物で構成することが一般的です。そのため、GNU プロジェクトの成果物を使った Linux システムを、GNU/Linux と表記することがあります。

GNU プロジェクトは、フリーソフトウェアという考え方のもとに、GPL を適用したソフトウェアだけの UNIX 互換のオペレーティングシステムと開発環境の構築を目指し開始されたプロジェクトです。GNU は、GNU's not Unix という再帰的な意味を持っています。GNU プロジェクトについては、The GNU Operating System^[2]を参照してください。

Linux システムでは、GNU ソフトウェアを始めとしたフリー/オープンソースソフトウェアや、プロプライエタリなソフトウェアなど、多くのソフトウェアを使用してシステムを構築します。しかしながら、これらを一つ一つ自分で組み合わせて目的に適合する安定したシステムとするのは大変な労力を必要とします。そこで、カーネルを始め、様々なツールやアプリケーション、ライブラリなどシステムを構成するのに必要なものすべてを収めたディストリビューションというものが存在します。

PC やサーバー用途向けのディストリビューションとして主なものを下記に示します。現在主流のディストリビューションは、GUI によるインストーラで簡単にインストールでき、コンパイル済みのソフトウェアをパッケージという単位でインストール/アンインストールできるなど、複雑な Linux システムを簡単に扱えるよう工夫されています。ここに挙げたもの以外にも多くのディストリビューションがあり^[3]、様々な用途に使われています。

- ・ Debian GNU/Linux [<http://www.debian.org>]
- ・ コミュニティによって開発、サポートされているディストリビューションです。フリーなソフトウェアだけで構成されています。ソフトウェアの管理は Debian パッケージと呼ばれるパッケージ単位で行われます。2010 年 8 月現在の最新版である Debian GNU/Linux 5.0(コードネーム lenny)では、24,000 以上のパッケージが用意されています。
- ・ Ubuntu Linux [<http://www.ubuntu.com>]

^[1]<http://www.gnu.org/licenses/licenses.ja.html#TOCGPL>

^[2]<http://www.gnu.org/>

^[3]<http://distrowatch.com/>

- ・ Debian GNU/Linux から派生したディストリビューションです。Canonical 社がバックアップしており、コンシューマ市場での使いやすさを重視しています。近年では、Ubuntu Linux をプリインストールした PC を販売するメーカーも出現してきています。
- ・ Fedora [<http://fedoraproject.org>]
- ・ 開発が終了した Red Hat Linux の後継として、コミュニティベースで開発、サポートされているディストリビューションです。RPM と呼ばれるパッケージでソフトウェアの管理を行います。
- ・ RedHat Enterprise Linux [<http://www.redhat.com>]
- ・ Red Hat 社が提供する商用ディストリビューション^[4]です。Fedora を元にして開発されています。
- ・ OpenSUSE [<http://opensuse.org>]
- ・ Novell 社によって支援されたコミュニティにより開発されている、ディストリビューションです。以前は SUSE Linux と呼ばれていました。技術者以外の一般のユーザーにとっての使いやすさを重視しており、RPM と呼ばれるパッケージでソフトウェアの管理をおこないます。
- ・ SUSE Linux Enterprise [<http://suse.com>]
- ・ Novell 社が提供する商用ディストリビューションです。OpenSUSE を元にして開発されています。
- ・ Gentoo Linux [<http://www.gentoo.org>]
- ・ コミュニティにより開発、サポートされているディストリビューションです。使いやすさよりも最適化や自由度を重視しており、ソースコードベースでソフトウェアを管理するのが特徴です。

2.2. 組み込み Linux システムとは

PC やサーバーで使用される"普通の"Linux システムと、組み込み Linux システムでは、同じソースコードをベースとしたカーネルを使用するという意味では、変わりはありません。但し、PC は Intel の x86 系アーキテクチャで動作するのに対して、組み込みでは ARM、MIPS、PowerPC、SH などのアーキテクチャが採用されることが多く、デバイスドライバなどのプロセッサ依存部分は普通の Linux とは異なります。また、組み込みシステムでは、特有の機能が必要になることが一般的ですので、組み込みシステム用のカーネルではボードごとに修正を加える必要があります。

ボード固有の部分に修正が必要になるとはいえ、ユーザーランドで動作するアプリケーションとカーネルとのインターフェースであるシステムコールに関しては、"普通の"Linux システムと組み込み Linux システムで代わりはありません。そのため、"普通の"Linux システムで動作するソフトウェアは、多くの場合、組み込み Linux システムでも動作します。

PC やサーバーと比較すると、組み込みシステムでは CPU クロックや、ストレージ容量、メモリ容量などのリソースが少ない場合が多く^[5]、また、外部環境も PC やサーバーに比べると過酷な場合があります。そのため、組み込み Linux システム向けにソフトウェアを作成する場合は、メモリ不足やストレージへの書き込みエラーが発生した場合の対処などに関してより慎重な姿勢が必要になります。

組み込み Linux で特に注意を要する事項として、ライセンスの問題があります。Linux システムでは、様々なライセンスが適用されたソフトウェアを組み合わせて使用することになります。前章でも説明したとおり、GPL が適用されるソフトウェアを機器に組み込んで出荷する場合は、該当ソフトウェアに関

^[4]特定の企業が開発しており、企業によるサポートを受けることができるディストリビューションを商用ディストリビューションと呼びます。

^[5]これは、相対的なものでマイコンボードに比べると豊富なリソースを持っていることになります。

するソースコードも機器に添付しなければなりません^[6]。もちろん、ライセンスでソースコードの公開が義務付けられていないソフトウェアの場合は、公開する必要はありません。

自分で作成したアプリケーションプログラムは、基本的にはソースコード公開の義務はありません。しかしながら、GPL が適用されたライブラリをアプリケーションプログラムから使用する場合は、アプリケーションプログラムまで GPL が伝搬します。そのため、作成したアプリケーションプログラムを第三者へ配布する場合は、ソースコード公開の義務が生じます^[7]。アプリケーションプログラムを作成する際には、利用するライブラリのライセンスも注意深く確認する必要があります。

"普通の"Linux システム向けのディストリビューションとして、Debian GNU/Linux や Fedora があつたように、組み込み向けのディストリビューションも存在します。組み込み向けのディストリビューションは、開発フレームワークとしての側面も持ち合わせており、「開発ディストリビューション」と呼ぶこともあります。

組み込み向けの開発ディストリビューションとしては、uClinux-dist や OpenWRT などがあります。

uClinux-dist は、ソースコードベースの開発ディストリビューションです。当初は MMU を搭載していないプロセッサ向けにカスタマイズした uClinux 用の開発ディストリビューションでしたが、現在ではそのような制限はありません。Linux カーネルと、ユーザーランドアプリケーション、ライブラリなどのソースコードを一つのツリーに統合し、それらを一括でビルドできるようになっています。

OpenWRT は、パッケージベースの開発ディストリビューションです。主に、ルータに採用されています。

^[6]または、ソースコードを入手する方法を提供する必要があります。

^[7]ライブラリに適用されることが多い LGPL(Lesser GPL)では、ライブラリ利用によってライセンス条件が伝搬しませんので、アプリケーションプログラムのソースコード公開義務は生じません。

3. Armadillo を使った組み込みシステム開発

3.1. Armadillo シリーズの概要

「Armadillo」は、株式会社アットマークテクノが開発、販売している ARM CPU を搭載した組み込み用途向けの小型汎用ボードコンピューターのシリーズ名称です。

動物のアルマジロは、スペイン語の「armado」を語源とした「武装した小さな動物」ということのようにですが、ボードコンピューターの Armadillo は「ARM CPU 搭載の小さな製品」という解釈に変えています。もともとは開発コードネームだったものが、そのまま製品シリーズの名称として使われています。



Armadillo のロゴ

「図 3.1. Armadillo ロゴ」が Armadillo シリーズのキャラクタです。アルマジロの上にペンギン(Tux)が乗っています。Tux は Linux の公式マスコット^[1]ですので、Armadillo というハードウェアの上に Linux というソフトウェアが載っていることを表します。

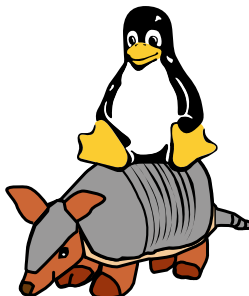


図 3.1 Armadillo ロゴ



ARM と SoC

ARM プロセッサは、英 ARM Ltd.が開発する 32bit RISC プロセッサです。同クラスの処理能力を持つ他のプロセッサと比較して、低消費電力で動作するという特徴を持ちます。

任天堂の Nintendo DS や米 Apple, Inc.の iPhone、iPad など近年話題になったモバイルデバイスの多くが ARM プロセッサを採用するなど、多くの身近な機器で使用されています。特に携帯電話で圧倒的なシェアを持

^[1]<http://ja.wikipedia.org/wiki/%E3%82%BF%E3%83%83%E3%82%AF%E3%82%B9>

ち、携帯電話 1 台に ARM プロセッサ搭載チップが平均 2.6 個使用されている^[2]とされています。

ARM Ltd.は、ARM プロセッサのアーキテクチャを IP として各社にライセンス販売し、自社では CPU を生産しないというビジネスモデルを取っています。ARM Ltd.からライセンスを供与された半導体メーカーは、ARM コアにコンピュータとして必要な周辺機能を追加し、SoC(System on Chip)としてパッケージ化して製造、販売します。

Apple Inc.、任天堂のほか、Cirrus Logic、Freescale、Intel、Marvel、Texas Instruments など多数の半導体メーカーが ARM Ltd.からライセンスを受けています。また、これまで独自アーキテクチャのプロセッサを製造してきた、ルネサス エレクトロニクスなど日本の半導体メーカーも ARM プロセッサを採用した SoC を製造しています。

初代 Armadillo(HT1070)が 2001 年 11 月に発売されてから、Armadillo-400 シリーズが 2010 年に発売されるまで、毎年のように新製品を発表してきました。「表 3.1. Armadillo シリーズの発表時期と CPU」に Armadillo シリーズの一覧表を示します。

表 3.1 Armadillo シリーズの発表時期と CPU

製品名	発表時期	LSI メーカー/型番	CPU コアの種類/動作クロック
Armadillo(HT1070)	2001 年 11 月	Cirrus Logic/ EP7312	ARM720T/74MHz
Armadillo-J	2003 年 10 月	Digi International/ NS7520	ARM7TDMI/55MHz
Armadillo-9	2004 年 7 月	Cirrus Logic/ EP9315	ARM920T/200MHz
Armadillo-210	2005 年 11 月	Cirrus Logic/ EP9307	ARM920T/200MHz
Armadillo-220/230/240	2006 年 4 月	Cirrus Logic/ EP9307	ARM920T/200MHz
Armadillo-300	2006 年 11 月	Digi International/ NS9750	ARM926EJ-S/200MHz
Armadillo-500 開発セット	2007 年 5 月	Freescale/ i.MX31 ^[1]	ARM1136JF-S/ 533MHz ^[2]
Armadillo-500 FX 液晶モデル	2008 年 11 月	Freescale/ i.MX31	ARM1136JF-S/ 533MHz ^[2]
Armadillo-440 液晶モデル	2010 年 2 月	Freescale/ i.MX257	ARM926EJ-S/400MHz
Armadillo-420 ベーシックモデル	2010 年 5 月	Freescale/ i.MX257	ARM926EJ-S/400MHz

^[1]発売当初は i.MX31L

^[2]発売当初は 400MHz 動作

^[2]ARM Ltd. 2010 年度第 2 四半期業績報告 <http://www.jp.arm.com/pressroom/10/100728.html>

Armadillo シリーズは、大きく 4 つの流れに分類できます。1 つ目は、外部拡張バスを持つ汎用製品群です。2 つ目は、搭載するインターフェースを絞って小型、低価格な機能特化型の製品群です。3 つ目は、無線 LAN 機能を有した製品の流れです。そして、4 つ目の最新の製品群がタッチパネル搭載製品群です。

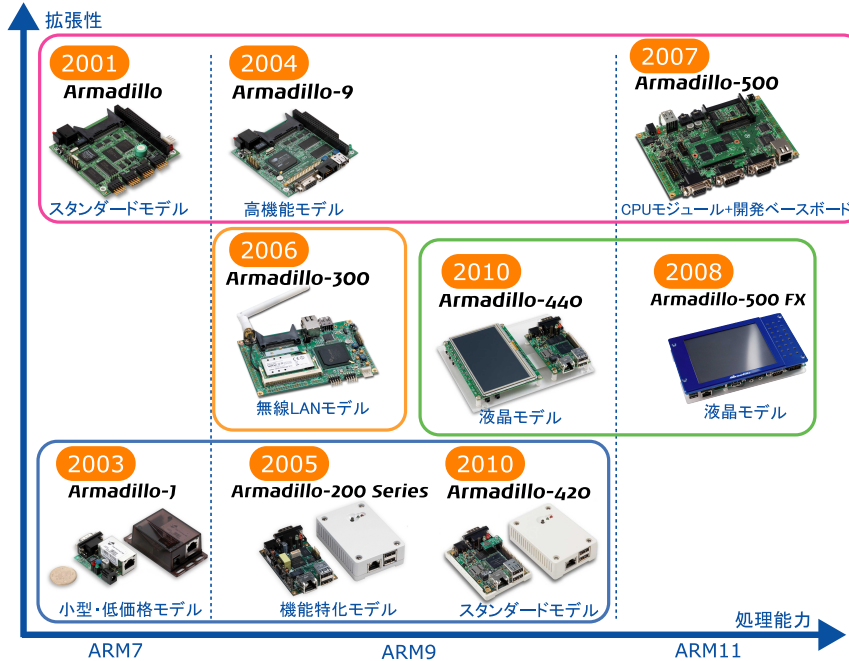


図 3.2 Armadillo シリーズ

汎用製品群の最初の製品は、Armadillo シリーズ最初の製品でもある初代 Armadillo (HT1070^[3]) です。初代 Armadillo は、外部拡張バスとして PC/104 バスを持った、汎用ボードとして設計されました。

初代 Armadillo の流れを組むのが Armadillo-9 で、USB ホスト、VGA 出力、CompactFlash、100Mbps 対応 LAN、ATA など多くのインターフェースが追加されました。

それをさらに発展させたのが、Armadillo-500 シリーズです。Armadillo-500 シリーズは、CPU やメモリというコア機能は Armadillo-500 CPU モジュールに集約し、拡張基板で機能を拡張するというコンセプトで設計されています。Armadillo-500 開発セット付属の拡張基板(ベースボード)では、Armadillo-9 と比べ USB ホストが High-Speed になった他、ストレージとして IDE の代わりに NAND フラッシュメモリと SD カードスロットが搭載されました。開発セット購入者にはベースボードの回路図が公開されており、自由にカスタマイズ可能です。周辺機能が自由に拡張可能という意味で、Armadillo-500 は究極の汎用ボードと言えるでしょう。

汎用性を追求した初代 Armadillo の流れとは異なり、Armadillo-J は、インターフェースをシリアルと LAN、GPIO のみに絞った機能特化型の製品です。組み込み機器では、インターフェースはこれで必要十分というケースも多々あります。また、機能を絞ることで、サイズと値段を抑えることができます。

この流れは、Armadillo-200 シリーズに受け継がれています。Armadillo-210 は、Armadillo-J と同様に、シリアル、LAN、GPIO しか持っていません。Armadillo-220 は、それに加えて USB ホスト機能を有しています。Armadillo-230 は、USB の代わりに LAN が二つあります。Armadillo-240 は、シリアルを一つ少なくする代わりに、VGA 出力を備えています。Armadillo-200 シリーズと Armadillo-9 は

^[3]初代 Armadillo は梅澤無線電機株式会社と共同で開発が進められたため、HT1070 という梅澤無線電機株式会社の製品としての型番も持っています。

ソフトウェア互換なので、機能が豊富な Armadillo-9 で試作をおこない、量産は必要十分な機能を持った Armadillo-200 シリーズで、といった選択が可能になりました。

機能特化型の最新製品である Armadillo-420 は、Armadillo-220 とピン互換で処理能力が2倍になった製品です。また、Armadillo-200 シリーズでは NAND ストレージはオプション品でしたが、Armadillo-420 では microSD スロットを標準搭載しており、使い勝手も向上しています。

無線 LAN 機能を有した製品としては、Armadillo-300 があります。それまで、産業用の組み込み用途としてボード製品を提供する場合、長期供給保証の面から無線 LAN 機能を提供するのは、困難でした。Armadillo-300 では、サイレックステクノロジー社から無線 LAN モジュールの供給を受け、長期供給保証できる製品として提供可能になりました。

無線 LAN 機能は、Armadillo-WLAN モジュールに受け継がれています。Armadillo-WLAN モジュールは、SDIO または SPI ホスト機能を持ったボードであれば接続可能なので、様々なボードに無線 LAN 機能を付加することができます。現在、Armadillo-500 開発セット、Armadillo-500 FX 液晶モデル、Armadillo-400 シリーズが標準で対応しています。

Armadillo シリーズの最新の流れは、液晶付きのパネルコンピュータ向けの製品です。この流れの最初の製品は、Armadillo-500 FX 液晶モデルです。Armadillo-500 FX 液晶モデルは、5.7 インチ TFT タッチパネル液晶とオーディオ、SSD を搭載しており、パネルコンピュータ開発のプラットフォームとして活用できます。いち早く、Google 社が発表した携帯電話用 OS である Android に対応したこともあり、大きな反響を呼びました。

Armadillo-440 は、Armadillo-500 FX 液晶モデルより小型にし、値段も抑え、より量産に適したパネルコンピュータプラットフォームとしての製品です。

Armadillo シリーズの詳しい仕様は、Armadillo 開発者サイトの Armadillo シリーズ仕様比較^[4]をご参照ください。

3.2. Armadillo-400 シリーズ

Armadillo-420 と Armadillo-440 を合わせて Armadillo-400 シリーズと呼びます。本書の内容は Armadillo を使った開発全般に適用できるものですが、具体例は Armadillo-400 シリーズを対象として説明します。そのため、本章では Armadillo-400 シリーズについて詳しく説明します。

3.2.1. Armadillo-400 シリーズの基本仕様

Armadillo-400 シリーズは、Freescale 社製 ARM9 プロセッサ i.MX257、LPDDR SDRAM、NOR フラッシュメモリを中心に、LAN、HighSpeed 対応 USB 2.0 ホスト、microSD スロット、GPIO といった組み込み機器に求められる機能を小さな基板面積に凝縮した、汎用 CPU ボードです。Armadillo-420 は、Armadillo-220 の後継モデルです。Armadillo-220 とピン互換を維持しながら、CPU クロック、RAM 容量、フラッシュメモリ容量がそれぞれ2倍になっています。Armadillo-440 は、Armadillo-400 シリーズの基本機能に加え、液晶、タッチパネル、オーディオといったマルチメディア機能を拡張基板によって追加可能な製品です。

Armadillo-400 シリーズの基本仕様とブロック図を「表 3.2. Armadillo-400 シリーズ基本仕様」と「図 3.3. Armadillo-400 シリーズブロック図」に示します。

表 3.2 Armadillo-400 シリーズ基本仕様

	Armadillo-420	Armadillo-440
プロセッサ	Freescale i.MX257 (MCIMX257)	

^[4]<http://armadillo.atmark-techno.com/specs>

	Armadillo-420	Armadillo-440
CPU コアクロック	400MHz	
RAM	LPDDR SDRAM: 64MByte	LPDDR SDRAM: 128MByte
フラッシュメモリ	NOR 16MByte	NOR 32MByte
本体基板サイズ	75.0mm x 50.0mm	
電源電圧	DC3.1 ~ 5.25V	
消費電力	約 1.2W	
使用温度範囲	-20~70°C (ただし結露なきこと)	

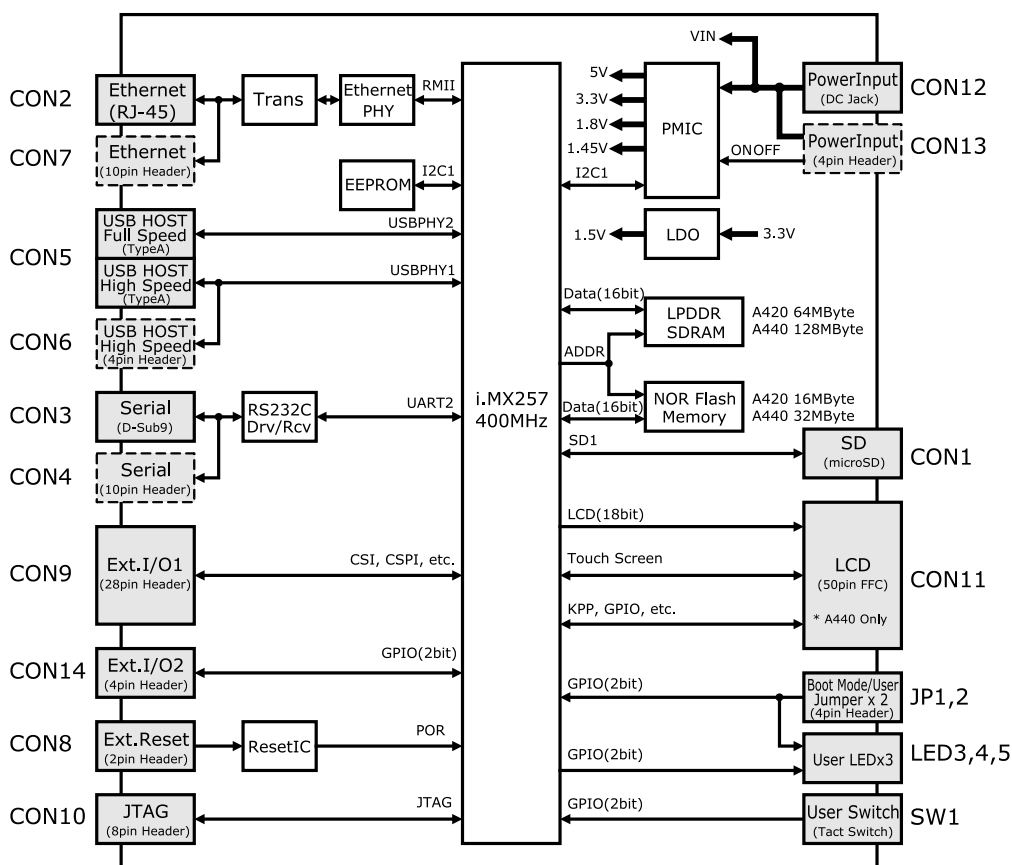


図 3.3 Armadillo-400 シリーズブロック図

Armadillo-400 シリーズは、拡張インターフェースに拡張基板を接続することで、機能を追加することができます。各製品の拡張インターフェースで追加可能な機能を「表 3.3. Armadillo-400 シリーズ拡張インターフェース」に示します^[5]。

表 3.3 Armadillo-400 シリーズ拡張インターフェース

	Armadillo-420	Armadillo-440
拡張インターフェース 1 (CON9)	GPIO、UART、SPI、one wire、PWM、SDHC、Audio	

^[5]Armadillo-400 シリーズでは、一つのピンに複数の機能が割り当てられているため、これらの機能がすべて同時に使えるわけではありません。どの機能を使用するかは、ソフトウェアで選択します。

	Armadillo-420	Armadillo-440
拡張インターフェース 2(CON14)	GPIO、I2C、CAN	
LCD インターフェース (CON11)	(なし)	GPIO、UART、I2C、Audio、LCD、Keypad

Armadillo-400 シリーズの開発セットには、これらの拡張インターフェースに接続可能な拡張基板が付属します。Armadillo-420 ベーシックモデル開発セットには、拡張インターフェース 2(CON14)に接続可能な「Armadillo-400 シリーズ RTC オプションモジュール」が、Armadillo-440 液晶モデル開発セットには、LCD インターフェース(CON11)に接続可能な「Armadillo-440 LCD 拡張ボード」が付属します。

開発セット購入者にはこれらの拡張基板の回路図が公開されているため、オリジナルの拡張基板を作成する際に、参考にすることができます。

Armadillo-420 ベーシックモデルと Armadillo-440 液晶モデルの見取り図及びフラッシュメモリの標準のメモリマップを以下に示します。

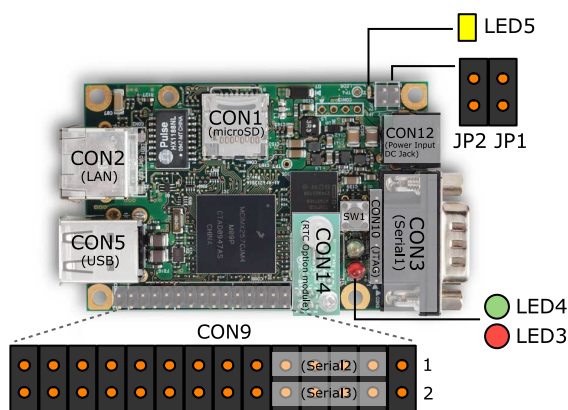


図 3.4 Armadillo-420 ベーシックモデル見取り図

表 3.4 Armadillo-420 ベーシックモデル フラッシュメモリ メモリマップ

物理アドレス	リージョン名	サイズ
0xa0000000 ~ 0xa001ffff	ブートローダー	128KB
0xa0020000 ~ 0xa021ffff	カーネル	2MB
0xa0220000 ~ 0xa0fdffff	ユーザーランド	13.75MB
0xa0fe0000 ~ 0xa0ffffff	コンフィグ	128KB



図 3.5 Armadillo-440 液晶モデル見取り図

表 3.5 Armadillo-440 液晶モデル フラッシュメモリ メモリマップ

物理アドレス	リージョン名	サイズ
0xa0000000 ~ 0xa001ffff	ブートローダー	128KB
0xa0020000 ~ 0xa021ffff	カーネル	2MB
0xa0220000 ~ 0xa1fdffff	ユーザーランド	29.75MB
0xa0fe0000 ~ 0xa1ffffff	コンフィグ	128KB

Armadillo-400 シリーズの詳細な仕様については、「Armadillo-400 シリーズハードウェアマニュアル」及び「Armadillo-400 シリーズソフトウェアマニュアル」を参照してください。

3.2.2. Armadillo-400 シリーズで出来ること

Armadillo-400 シリーズの特徴として、ハードウェア、ソフトウェア両面でのカスタマイズの自由度が高い点が挙げられます。Armadillo-400 シリーズでは、以下のことが実現できます。

1. 様々なハードウェア機能を追加することができます
2. オリジナルのアプリケーションを作成することができます
3. 豊富なオープンソースソフトウェア資産を活用することができます
4. Armadillo-400 シリーズに標準対応した有償ソフトウェアを活用することができます

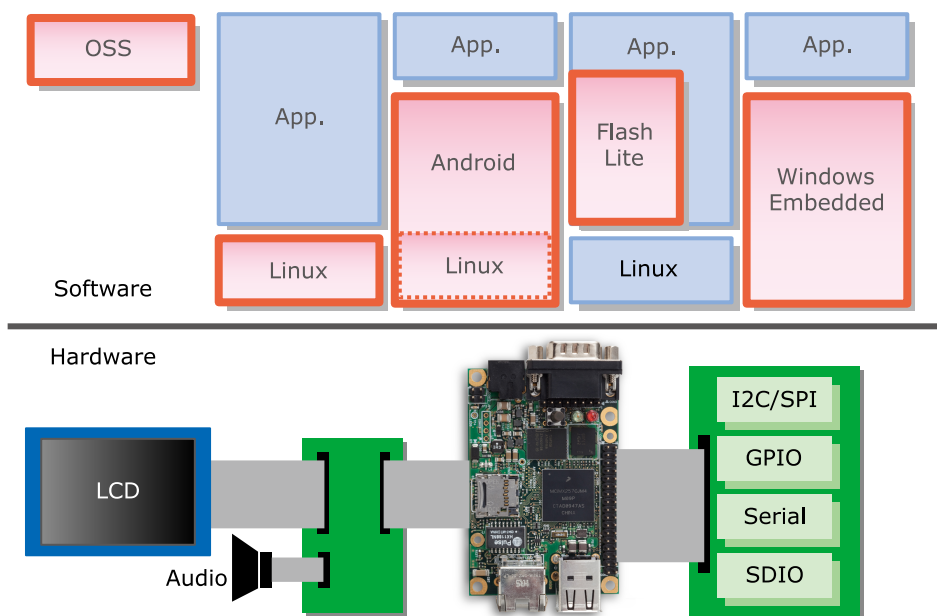


図 3.6 Armadillo-400 シリーズで出来ること

3.2.2.1. ハードウェア機能の追加

Armadillo-400 シリーズは、USB や LAN、シリアル(RS232C)といった標準インターフェースを持っていますので、豊富な外部機器を接続することができます。また、前章でも述べたように、拡張インターフェースに拡張基板を接続することで、ハードウェア機能を追加することもできます。

Armadillo-400 で使用可能なインターフェースの一覧を、「表 3.6. Armadillo-400 シリーズで使用可能なインターフェース」に示します。Armadillo-400 シリーズは標準の OS として Linux を採用しており、これらのインターフェースに関するデバイスドライバはすべて提供されます。

表 3.6 Armadillo-400 シリーズで使用可能なインターフェース

インターフェース	コネクタ	i.MX25 モジュール名	備考
シリアル RS232C レベル	CON3 (CON4)	UART2	標準で有効
シリアル 3.3V レベル	CON9	UART3	標準で有効、CON11 UART3 と排他
	CON9	UART5	標準で有効
	CON11	UART3	CON9 UART3、キーパッドの一部のピンと排他
	CON11	UART4	CON11 AUD5、キーパッドの一部のピンと排他
LAN	CON2 (CON7)		標準で有効
CAN	CON14	CAN2	CON14 I2C2、PWM4 と排他
USB2.0 ホスト HighSpeed	CON5 下段 (CON6)		標準で有効
USB2.0 ホスト FullSpeed	CON5 上段		標準で有効

インターフェース	コネクタ	i.MX25 モジュール名	備考
SD/SDIO/MMC	CON9	eSDHC2	CON9 SPI3、CON9 AUD6 と排他
microSD/SDIO/MMC	CON1	eSDHC1	標準で有効
I2C マスター	CON11	I2C3	キーパッドの一部のピンと排他
	CON14	I2C2	CON14 CAN2、PWM4 と排他
SPI マスター	CON9	SPI1	CON9 UART3 と排他、スレーブ最大 2 つ(CON9 PWM2 使用時 1 つ)
	CON9	SPI3	CON9 eSDHC2、CON9 MXC_OWIRE と排他、スレーブ最大 4 つ(CON9 AUD6 使用時 2 つ)
one wire	CON9	MXC_OWIRE	最大 1 ピン
	CON9	GPIO	最大 1 ピン、CON9 SPI1 と排他
PWM	CON9	PWM2	最大 1 ピン、CON9 SPI1 の SS 信号の 1 つと排他
	CON14	PWM4	CON14 I2C2、CAN2 と排他
I2S	CON9	AUD6	CON11 AUD5、CON9 eSDHC2、CON9 SPI3 の SS 信号の 2 つと排他
	CON11	AUD5	CON9 AUD6、キーパッドの一部のピン、CON11 UART4 と排他
16bit LCD	CON11		標準で有効
4 線式タッチパネル	CON11		標準で有効
キーパッド	CON11		キーパッド用のピンを使用する他のインターフェースと排他、最大 6x4
JTAG	CON10		標準で使用可
GPIO	CON9		標準で 18 ピン有効、CON9 を使用する他のインターフェースと排他、最大 22 ピン
	CON11		CON11 を使用する他のインターフェースと排他、最大 10 ピン
	CON14		CON14 を使用する他のインターフェースと排他、最大 2 ピン

USB インターフェースに接続できる機器のリストの一部を、以下に示します。Armadillo-400 シリーズで動作可能な機器はこれがすべてではありません。動作確認が取れているデバイスは、Armadillo 開発者サイトの動作デバイスのページ^[6]に随時追加していますので、そちらもご参照ください。

1. USB to Ethernet 変換ケーブル
2. USB 無線 LAN アダプタ
3. USB G3 モデム
4. USB to シリアル変換ケーブル
5. USB メモリ
6. USB マウス
7. USB キーボード
8. USB オーディオ
9. USB ディスプレイ

^[6]<http://armadillo.atmark-techno.com/tested-devices>

10. UVC 対応 USB カメラ

3.2.2.2. オリジナルアプリケーションプログラムの追加

Armadillo-400 シリーズは汎用ボードとして設計されているため、オリジナルのアプリケーションプログラムを作成し、それを Armadillo に組み込むことができます。

Armadillo は完全な Linux システムですので、他の Linux システム用に作成したプログラムの多くは、ほとんど修正せずに Armadillo でも動作することでしょう。

アプリケーションプログラムは、C/C++言語や各種スクリプト言語(シェルスクリプト、PHP、Perl など)で作成することができます。Armadillo 用に新しい言語を習得する必要はありません。

3.2.2.3. オープンソースソフトウェア資産の活用

近年の複雑、大規模化したシステムを構築するにあたり、すべての機能を自前で開発することは現実的ではありません。ネットワークプロトコルスタックや、ファイルシステムなど汎用的なソフトウェアコンポーネントにオープンソースソフトウェアを活用することで、オリジナルのアプリケーション開発にリソースを集中することができます。

Armadillo-400 シリーズでは、標準の開発ディストリビューションとしてアットマークテクノ独自の Atmark Dist を採用しています。Atmark Dist は、多くのユーザーランドアプリケーション及びライブラリを含んでおり、これらを簡単に Armadillo に組み込んで動作させることができます。

また、Atmark Dist の代わりに、Debian GNU/Linux 5.0 (コードネーム"lenny")を選択することもできます。Debian GNU/Linux 5.0 には、オープンソースのアプリケーションやライブラリなどのパッケージが 24,000 個以上用意されており、これらの多くを簡単な手順で Armadillo-400 シリーズで動作させることができます。

オープンソースソフトウェアを活用することにより、Armadillo-400 シリーズで実現可能なソフトウェア機能の一覧を「表 3.7. Armadillo-400 シリーズで実現可能なソフトウェア機能一覧」に示します。このリストは実現可能な機能のほんの一部にすぎません。

表 3.7 Armadillo-400 シリーズで実現可能なソフトウェア機能一覧

分類	機能	アプリケーション/ライブラリ名	実現方法
ネットワーク	Web(HTTP)サーバー	lighttpd	A ^[1]
		Apache2	B ^[2]
	FTP サーバー/クライアント	ftpd/ftp	A
	Telnet サーバー/クライアント	telnetd/telnet	A
	SSH サーバー/クライアント	OpenSSH	A
	DHCP サーバー/クライアント	dhcpd/udhcpd	A
	NTP サーバー/クライアント	ntpd/ntpclient	A
	PPP	pppd	A
	カメラサーバー	mjpg-streamer	A
	HTTP/FTP クライアント	wget/ftpget/ftpget	A
	Zeroconf	avahi	A
	メール送信(SMTP)	mail	A
		sendmail	B
	ファイヤーウォール	iptables	A
SNMP	net-snmp	A	
マルチメディア	オーディオ再生/録音	alsa-utils	A
	MP3 再生	mp3play	A
	画像処理ライブラリ	libjpeg62/libpng12	B
データベース	データベース	sqlite3	A
ファイルシステム	VFAT(FAT32)	mkdosfs	A, C ^[3]
	EXT2	mke2fs	A, C
	EXT3	mke2fs	A, C
	jffs2	flasher	A, C
	NFS	-	A, C
	samba	samba	B, C
GUI	ウィンドウシステム	The KDrive Tiny X Server	A
		X.org X Window System	B
	GUI ツールキット	Gtk+ 2.0	B
言語	Perl	Perl 5.0	A
	PHP	PHP 5.2	B
	Python	Python 2.5	B
	Ruby	Ruby 4.2	B

分類	機能	アプリケーション/ライブラリ名	実現方法
その他	シェル	ash	A
		bash	B

[1]Atmark Dist に含まれます

[2]Debian パッケージに含まれます

[3]Linux カーネル機能に含まれます

さらに、Armadillo-440 ではオープンソースの OS である Android を選択することもできます^[7]。Android を使うことによって、更に可能性が広がります。

3.2.2.4. 有償ソフトウェアの活用

Armadillo-400 シリーズではオープンソースソフトウェアだけではなく、Armadillo-400 シリーズ用にポーティングされた、プロプライエタリなソフトウェアも使用することができます。オープンソースソフトウェアと組み合わせることで、更にシステム開発の効率を上げることができるでしょう。

表 3.8 Armadillo-400 シリーズで使用可能なソフトウェア製品

製品名	概要	参照
eSOL Flash Lite	Flash Player	http://armadillo.atmark-techno.com/adobe-flash-lite
Windows Embedded CE 6	組み込み OS	http://armadillo.atmark-techno.com/howto/try-wince60r3-on-armadillo440
Windows Embedded Compact 7	組み込み OS	http://armadillo.atmark-techno.com/howto/try-wince60r3-on-armadillo440

3.3. Armadillo の開発環境

アットマークテクノでは Armadillo 用の標準開発環境として、クロス開発用ツールチェーン、クロスライブラリ、フラッシュライター(ダウンローダー)などをインストールし、設定済みの環境を ATDE(AtmarkTechno Development Environment) という名称で提供しています。ATDE を使うと、簡単に開発環境を構築することができます。

また、Armadillo 用の開発ディストリビューションとして、Atmark Dist というアットマークテクノ独自の開発ディストリビューションも提供しています。Atmark Dist を使うと、ターゲットに最適なカーネルのビルド、ルートファイルシステムの作成、及び、それらをフラッシュメモリに書き込むためのイメージファイルの作成を自動でおこなうことができます。

ATDE 及び Atmark Dist を使った開発の手順は、「7. 開発の基本的な流れ」で説明します。

これらの開発に最低限必要なものは、すべて開発セット付属の DVD か Armadillo 開発者サイト^[8]から無償で入手することができます。

さらに、有償のサードパーティ製ツールを活用することで、開発効率をあげることができます。

Windows 上で動作する統合開発環境として、株式会社エスパークが提供する μ SPaX^[9]を使用することができます。 μ SPaX を使用すると、グラフィカルな統合開発環境で、GUI/CUI アプリケーションの開発やリモートデバッグが可能になります。

[7]<http://armadillo.atmark-techno.com/android>

[8]<http://armadillo.atmark-techno.com>

[9]http://www.espark.co.jp/index.php?option=com_content&view=article&id=96&Itemid=87

また、Armadillo では JTAG-ICE を使用することも可能です。Armadillo-400 シリーズの JTAG インターフェース(CON10、8 ピン 2.54 mm ピッチ)を、ARM 標準コネクタ(20 ピン、2.54 mm ピッチ)に変換する JTAG変換ケーブルをオプション品として販売していますので、サードパーティー製の JTAG-ICE を使用して、Armadillo-400 シリーズのデバッグを行うことができます。

3.4. Armadillo を採用した場合の製品開発サイクル

Armadillo は、開発セットによる試作開発から多品種少量生産の量産品にまで対応できるよう、様々なサービスを提供しています。

「図 3.7. Armadillo を採用した場合の開発の流れ」 に Armadillo をプラットフォームとして採用した場合の開発の流れを示します。

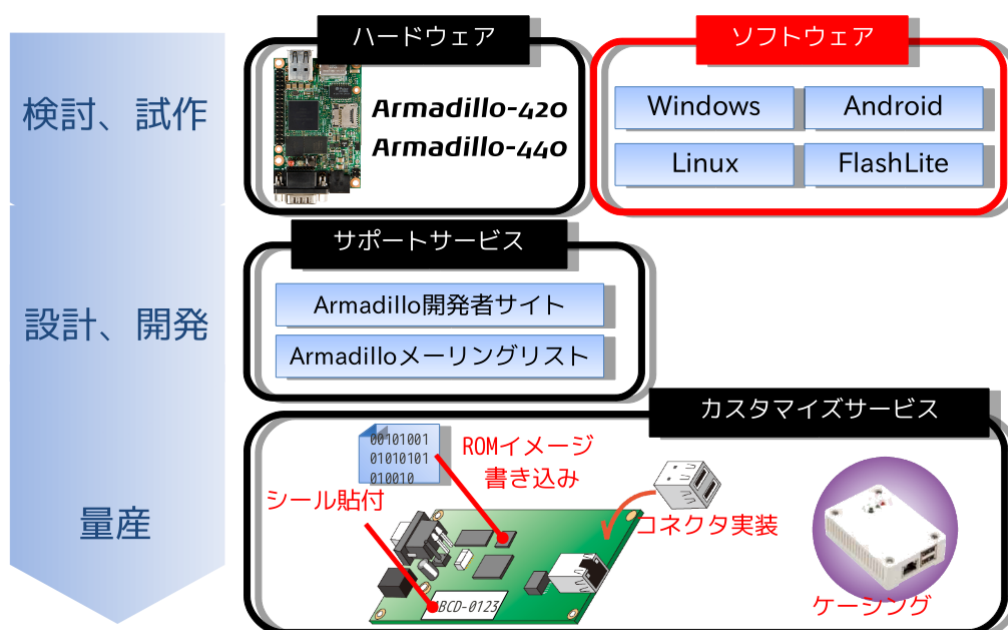


図 3.7 Armadillo を採用した場合の開発の流れ

3.4.1. 検討、試作

検討段階においては、まず、「表 3.6. Armadillo-400 シリーズで使用可能なインターフェース」や「Armadillo-400 シリーズ ハードウェアマニュアル」を参照し、必要な機能がハードウェア的に実現可能か検討します。

ハードウェア的に実現可能であれば、次はソフトウェアプラットフォームの選択をおこないます。Armadillo-400 シリーズの場合は、OS として Linux システム(Atmark Dist または Debian GNU/Linux)、Windows Embedded CE 6.0、Windows Embedded Compact 7、Android が選択できます。OS として Linux システムを採用した場合の GUI 環境としては、X window system/Gtk、Flash Lite、μSPaX などが選択肢に挙げられます。また、「表 3.7. Armadillo-400 シリーズで実現可能なソフトウェア機能一覧」に示したようなオープンソースソフトウェアを活用できないかも検討の価値があります。

ハードウェア、ソフトウェア的に実現可能であると判断できれば、試作に取りかかります。Armadillo は、1 台から購入可能な開発セットと無償の開発環境があるので、すぐに試作開発を開始することが可能です。

3.4.2. 設計、開発

設計、開発段階においては、様々なサポートサービスを受けることができます。

無償で誰でもアクセス可能な情報源として、Armadillo 開発者サイト^[10]を運営しています。Armadillo 開発者サイトには、マニュアル類、最新ソフトウェアのソースコードとすぐに動かすことができるイメージファイル、豊富な Howto、FAQ や動作確認デバイス一覧などの情報が満載です。

Armadillo 開発者サイトで探してみても疑問が解決しない場合は、Armadillo メーリングリスト^[11]に質問することもできます。メーリングリストでは、数百人規模の読者がいるので、同じような問題に遭遇したことがある人から、ヒントが得られるかもしれません。また、これまでメーリングリストでやりとりされた内容はすべてアーカイブされ、検索できるようになっています。質問する前にアーカイブを検索することで、答えが見つかる場合も多々あります。

3.4.3. 量産

量産時への対応として、カスタマイズサービスを提供しています。カスタマイズサービスでは、フラッシュメモリへのユーザー指定のイメージ書き込み、コネクタの実装/非実装の選択、シリアルナンバーなどのシール貼り付け、ケーシング、梱包といったカスタマイズを Armadillo に施し、お客様へ納品します。



何台から量産？

量産といっても、対象とするものによって規模感がまったく異なるものです。Armadillo のカスタマイズサービスでは、1 ロット 50 台以上を量産と定義し、サービスを適用することができます。

3.4.4. 保守

組み込みシステムを製造、販売する場合には、どのような保守サービスが受けられるかも重要なポイントです。

製品購入後にユーザー登録をすることで、ハードウェアやソフトウェアに重要な変更(リビジョン変更、バグフィックス)があれば、通知されます。

また、アットマークテクノ製品では、製品発売開始後、標準で 5 年間の供給保証をしています。基本的なスタンスとして、部品を調達可能な間は製品の提供を続けています。

Armadillo を使えば、量産後も安心です。

^[10]<http://armadillo.atmark-techno.com>

^[11]<http://armadillo.atmark-techno.com/maillinglists>

4. Armadillo の基本操作

これまでに、組み込み Linux システムとは何か、Armadillo とはどのようなものかについて説明してきました。本章では、実際の開発作業に入る前に、Armadillo の基本的な操作方法について説明します。

本章で説明することは、Armadillo を起動して、簡単なコマンドを入力し、Armadillo を終了するといった、本当に基本的なことと、ファイルの編集、ネットワーク接続だけです。あえて、詳しい説明はおこないません。Armadillo とはどのようなコンピュータなのか、実際に動かして確認してください。

Linux や Armadillo の取扱いに慣れている方には、本章の内容は不要かもしれません。しかし、随所に Armadillo を上手く扱うためのヒントや、あまり意識されていないけれども実は重要なことが書かれていますので、一度目を通してみてください。

4.1. 取り扱い上の注意事項

Armadillo を使用するにあたって、以下のような点にご注意ください。場合によっては、Armadillo が壊れる可能性があります。

- ・ 基板に、落下や衝撃などの強い振動を与えないでください。
- ・ 電源が入っている状態で端子に触らないでください。
- ・ 基板のまわりに金属ゴミ等がある状態では使用しないでください。
- ・ microSD コネクタおよびそのカバーや、Armadillo-440 と LCD 拡張ボードを接続しているフラットケーブルコネクタは、破損しやすい部品になっています。無理に力を加えて破損することのないよう十分注意してください。
- ・ Armadillo や周辺回路に電源が入っている状態で、活線挿抜対応インターフェース(LAN、USB、マイク、ヘッドフォン)以外へのコネクタ着脱は、絶対に行わないでください。
- ・ 電源および入出力からの過大なノイズやサージ、電源電圧の急激な変動等により、使用している CMOS デバイスがラッチアップを起こす可能性があります。いったんラッチアップ状態となると、電源を切断しないかぎりこの状態が維持されるため、デバイスの破損につながる可能性があります。ノイズの影響を受けやすい入出力ラインには、保護回路を入れることや、ノイズ源となる装置と共通の電源を使用しない等の対策をとることをお勧めします。
- ・ Armadillo には CMOS デバイスを使用していますので、ご使用になる時までは、帯電防止対策された出荷時のパッケージ等にて保管してください。

4.2. Armadillo と作業用 PC との接続

4.2.1. 準備するもの

Armadillo を使った組み込み Linux システム開発に必要な機材を「表 4.1. 必要な機材」に示します。

最低限、Armadillo 本体と、ソフトウェアの開発や Armadillo の操作を行うための作業用 PC、シリアルクロスケーブルだけで開発をスタートすることができます。特殊な JTAG-ICE やフラッシュライタなどは必要ありません。この手軽さが Armadillo の魅力でもあります。

表 4.1 必要な機材

機材	説明
Armadillo	「Armadillo-420 ベーシックモデル 開発セット」または「Armadillo-440 液晶モデル 開発セット」
作業用 PC	Linux または Windows が動作し、1 ポート以上のシリアルインターフェース ^[1] を持つ PC
シリアルクロスケーブル	D-Sub 9 ピン(メス - メス)のシリアルクロスケーブル
シリアル通信ソフトウェア	Linux では「minicom」、Windows では「Tera Term」など
LAN ケーブル	Armadillo と LAN を経由した通信を行う場合に必要
スイッチングハブ	作業用 PC と Armadillo をハブを介して接続する場合に必要 ^[2]

^[1]USB ポートがある場合は、USB to シリアル変換ケーブルで代替できます。USB to シリアル変換ケーブルのドライバのインストールや使用方法などは、シリアル変換ケーブルのマニュアル等を参照してください。

^[2]Armadillo-400 シリーズは Auto MDIX に対応しているため、ストレートまたはクロス LAN ケーブルで作業用 PC と直接接続することもできます。



注意: フラッシュメモリは出荷状態に

以降の説明では、Armadillo は出荷状態になっていることを想定していません。

Armadillo のフラッシュメモリを書き換えている場合は、「Armadillo-400 シリーズソフトウェアマニュアル」の「フラッシュメモリの書き換え方法」を参照して、ブートローダー、カーネル、ユーザーランドの各領域を標準の最新のイメージファイルに書き換えてください。

また、コンフィグ領域を書き換えている場合は、「Armadillo-420 ベーシックモデル開発セット スタートアップガイド」、または「Armadillo-440 液晶モデル開発セット スタートアップガイド」の「コンフィグ領域の初期化」を参照して、コンフィグ領域を出荷状態に戻してください。


4.2.2. 接続方法

「図 4.1. Armadillo-440 液晶モデル接続例」もしくは「図 4.2. Armadillo-420 ベーシックモデル接続例」に示す接続例を参考に、Armadillo と作業用 PC および周辺機器を接続してください。

Armadillo-400 シリーズの標準状態ではシリアルインターフェース 1(CON3、D-Sub 9 ピンコネクタ)をシリアルコンソールとして使用します。そのため、作業用 PC と Armadillo のシリアルインターフェースをクロスケーブルで繋がます。

また、Armadillo は様々なネットワーク機能を持っています。作業用 PC から Armadillo にネットワーク越しに接続するために、LAN ケーブルで接続します。

なお、ジャンパピンにジャンパソケットがささってショート状態になっている場合は、外してオープン状態にしておいてください。ジャンパの設定の詳細については「表 5.1. ジャンパ設定」で説明します。



ジャンパソケットをなくさない方法

ジャンパソケットは開発セットに一つしか入っていませんが、すぐになくしてしまいがちです。

ジャンパソケットの片足をジャンパピンにさしたままにしておくと、なくさずに済みます。

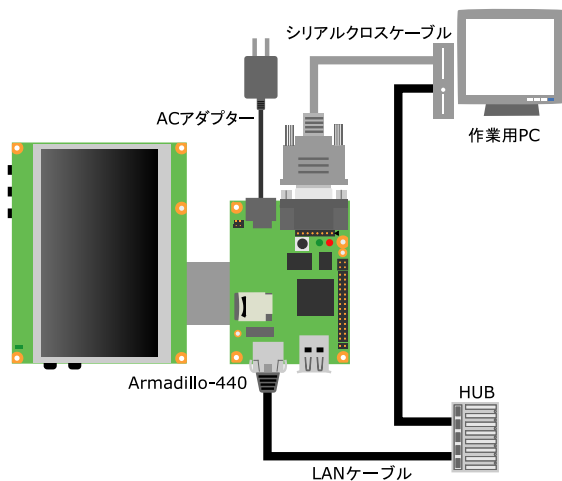


図 4.1 Armadillo-440 液晶モデル接続例

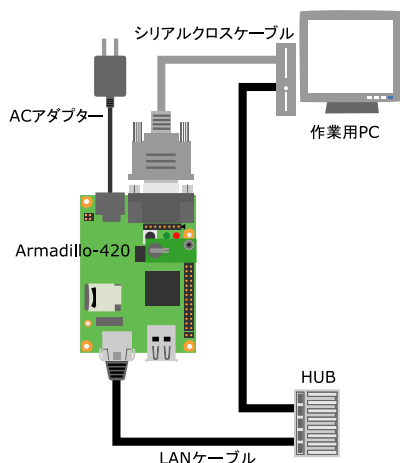


図 4.2 Armadillo-420 ベーシックモデル接続例

4.2.3. シリアル通信ソフトウェアの設定

作業用 PC から Armadillo のコンソールにシリアル経由で接続するために、シリアル通信ソフトウェアを使用します。

作業用 PC の OS が Windows の場合は、「4.2.3.1. Tera Term の設定」を参照し、シリアル通信ソフトウェアの設定を行ってください。

作業用 PC の OS が Linux の場合は、「4.2.3.2. minicom の設定」を参照し、シリアル通信ソフトウェアの設定を行ってください。

4.2.3.1. Tera Term の設定

本章では、Tera Term を使用して、Armadillo にシリアル経由で接続するための設定方法を順を追って説明します。

SourceForge の Tera Term プロジェクト^[1]からダウンロードし、インストールした Tera Term 4.67^[2]を使用します。

Tera Term のインストール方法や、使い方についての詳細は Tera Term Home Page^[3]を参照してください。

以下の手順で、Tera Term のシリアルポートの設定を行ってください。

1. 最初に起動した時には、「図 4.3. Tera Term: 新しい接続画面」ダイアログが表示されます。「キャンセル」ボタンを押して、「図 4.3. Tera Term: 新しい接続画面」ダイアログを閉じてください。

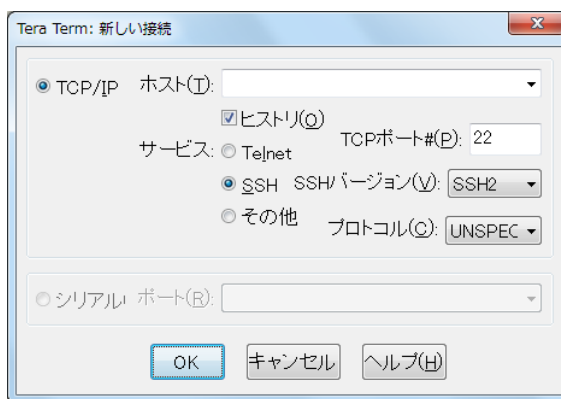


図 4.3 Tera Term: 新しい接続画面

2. 「図 4.4. Tera Term 画面」が表示されますので、「設定」 - 「シリアルポート」メニューを選択してください。

^[1]<http://sourceforge.jp/projects/ttssh2/>

^[2]2010年9月現在の最新バージョン。

^[3]<http://ttssh2.sourceforge.jp/>

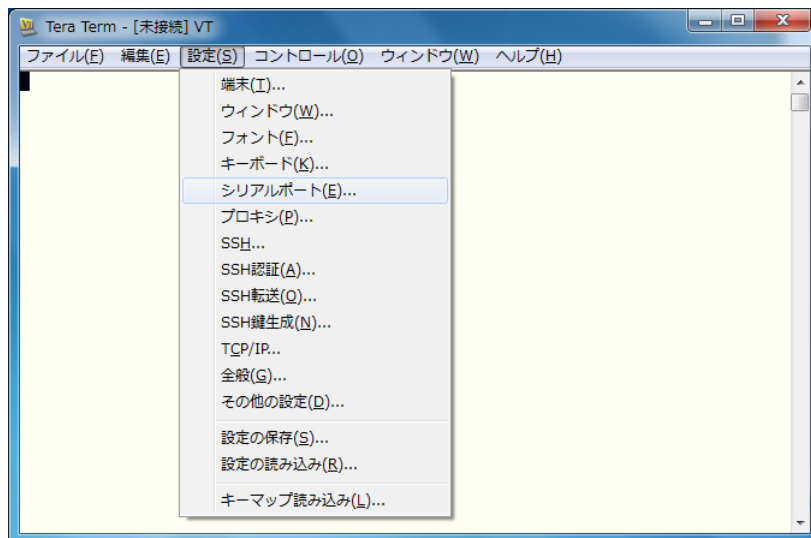


図 4.4 Tera Term 画面

3. 「図 4.5. Tera Term: シリアルポート設定画面」が表示されます。



図 4.5 Tera Term: シリアルポート設定画面

- 「ポート」に Armadillo と接続されているシリアルポート番号を設定してください。
- その他の設定項目は「表 4.2. シリアル通信設定」を参照し、「図 4.5. Tera Term: シリアルポート設定画面」のように設定してください。
- 「OK」ボタンを押してください。

表 4.2 シリアル通信設定

項目	設定
ボーレート	115,200 bps
データ長	8 bit
ストップビット	1 bit
パリティビット	なし

項目	設定
フロー制御	なし



シリアルポートのポート番号の確認方法

シリアルポートのポート番号は、デバイスマネージャーの「ポート (COM と LPT)」にあるデバイスで確認できます。

「図 4.6. デバイスマネージャー画面」は USB to シリアル変換ケーブルを使用した場合のポート番号表示例です。この例では、シリアルポートのポート番号に COM6 が割り当てられています。

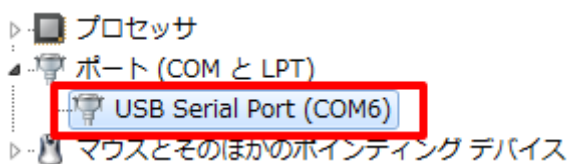


図 4.6 デバイスマネージャー画面

- 画面上部のタイトルバーの表示を確認してください。接続されていれば「図 4.7. Tera Term 画面」のようにタイトルバーに「COM6:115200baud」^[4]と表示されます。

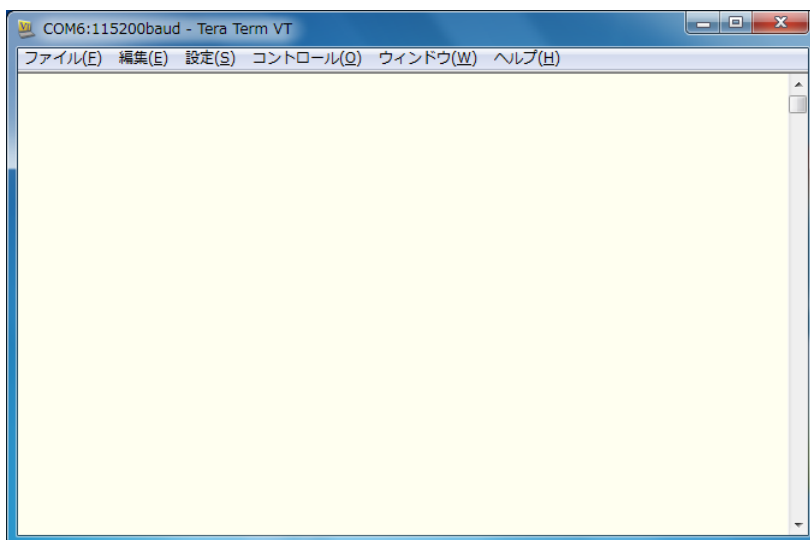


図 4.7 Tera Term 画面



Tera Term の設定を保存する方法

接続の設定を設定ファイル(TERATERM.INI)に保存しておくことで、次回起動時からは自動的にシリアルポートに接続することができます。

^[4]COM6 ポートに接続した場合

設定を保存するには、「設定」メニューから「設定の保存」を選択してください。「Tera Term: 設定の保存」ダイアログが表示されますので、保存するファイルを指定し、「OK」ボタンを押すことで保存できます。

4.2.3.2. minicom の設定

Linux から Armadillo に接続する場合は、minicom を使用します。

本章では、minicom を使用した場合の接続設定について説明します。「表 4.3. シリアル通信設定」に示すパラメーターに設定します。

表 4.3 シリアル通信設定

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

1. 「図 4.8. minicom の設定の起動」に示すコマンドを実行し、minicom の設定画面を起動してください。

```
[PC ~]$ LC_ALL=C minicom -s
```

図 4.8 minicom の設定の起動

2. 「図 4.9. minicom の設定」が表示されますので、「Serial port setup」を選択してください。

```
+-----[configuration]-----+
| Filenames and paths         |
| File transfer protocols     |
| Serial port setup          |
| Modem and dialing          |
| Screen and keyboard        |
| Save setup as dfl          |
| Save setup as..            |
| Exit                        |
| Exit from Minicom          |
+-----+
```

図 4.9 minicom の設定

3. 「図 4.10. minicom のシリアルポートの設定」が表示されますので、A キーを押して Serial Device を選択してください。

```

+-----+
| A - Serial Device      : /dev/ttyS0
| B - Lockfile Location  : /var/lock
| C - Callin Program     :
| D - Callout Program    :
| E - Bps/Par/Bits       : 115200 8N1
| F - Hardware Flow Control : No
| G - Software Flow Control : No
|
| Change which setting?
+-----+
    
```

図 4.10 minicom のシリアルポートの設定

- Serial Device に使用するシリアルポートを入力して Enter キーを押してください。



USB to シリアル変換ケーブル使用時のデバイスファイル確認方法

Linux で USB to シリアル変換ケーブルを接続した場合、コンソールに以下のようなログが表示されます。

```

usb 5-1: new full speed USB device using uhci_hcd and address 15
usb 5-1: configuration #1 chosen from 1 choice
ftdi_sio 5-1:1.0: FTDI USB Serial Device converter detected
usb 5-1: Detected FT232RL
usb 5-1: Number of endpoints 2
usb 5-1: Endpoint 1 MaxPacketSize 64
usb 5-1: Endpoint 2 MaxPacketSize 64
usb 5-1: Setting MaxPacketSize 64
usb 5-1: FTDI USB Serial Device converter now attached to ttyUSB0
    
```

図 4.11 例. USB to シリアル変換ケーブル接続時のログ

上記のログから USB to シリアル変換ケーブルが ttyUSB0 に割り当てられたことが分かります。

- F キーを押して Hardware Flow Control を No に設定してください。
- G キーを押して Software Flow Control を No に設定してください。
- キーボードの E キーを押してください。「図 4.12. minicom のシリアルポートのパラメータの設定」が表示されます。


```

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....done.
Linux version 2.6.26-at9 (2.6.26) (atmark@atde3) (gcc version 4.3.2 (Debian 4.3.2-1.1) ) #1 PREEMP
T Tue Jun 8 16:16:57 JST 2010
CPU: ARM926EJ-S [41069264] revision 4 (ARMv5TEJ), cr=00053177
Machine: Armadillo-440
:
:
:
    
```

図 4.14 起動ログ

4.4. ログイン

起動が完了すると、ログインプロンプトが表示されます。

例として、Armadillo-440 のログインプロンプトを以下に示します。

```

atmark-dist v1.26.0 (AtmarkTechno/Armadillo-440)
Linux 2.6.26-at9 [armv5tejl arch]

armadillo440-0 login:
    
```

「表 4.4. シリアルコンソールログイン時のユーザー名とパスワード」に示すユーザとパスワードでログインすることができます。ユーザーの詳しい説明については第 2 部「Linux の仕組みと運用、管理」の「ユーザー管理」で説明します。

表 4.4 シリアルコンソールログイン時のユーザー名とパスワード

ユーザ名	パスワード	権限
root	root	特権ユーザ
guest	(なし)	一般ユーザ

本章では、以下のように入力して root ユーザーでログインしてください。

```

atmark-dist v1.26.0 (AtmarkTechno/Armadillo-440)
Linux 2.6.26-at9 [armv5tejl arch]

armadillo440-0 login: root ↵ ❶
Password: root ↵ ❷
    
```

図 4.15 ユーザー名とパスワードを入力してログインする

- ❶ 「root」と入力した後、Enter キーを入力します。

② パスワードは入力しても表示されません。

4.5. プロンプト

正常にログインできると、プロンプトが表示されます。

例として、Armadillo-440 でログインした場合のプロンプトを以下に示します。^[5]

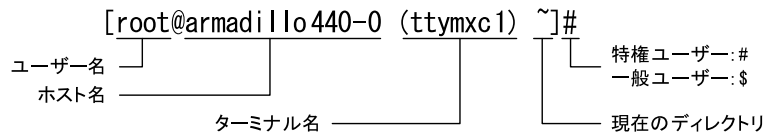


図 4.16 Armadillo-440 でのプロンプト表示例

プロンプトは、シェルというプログラムが表示しています。コマンドを実行する場合はプロンプトの右側にコマンドを入力します。

シェルは、ユーザーの入力を読み取り、コマンドを実行するコマンドラインインタプリターです。カーネルの外層として機能し、ユーザーとのインターフェースになることから、シェル(殻)という名称になっています。Windows のコマンドプロンプトのようなものです。

本書では、コマンドは以下の書式で記述します。

コマンド名 <必須引数> [省略可能な引数] [複数指定できる省略可能な引数...]

図 4.17 コマンドの書式

例えば、指定した文字を表示する echo コマンドの場合、以下のように表記します。

echo [string...]

図 4.18 echo コマンドの書式

これは、echo コマンドの引数として 0 個以上の値を指定できることを意味します。

echo コマンドを実行すると、以下のような表示が得られます。

```
[armadillo ~]# echo ↵ ①
[armadillo ~]# echo hello ↵ ②
hello
[armadillo ~]# echo hello world! ↵ ③
hello world!
```

図 4.19 echo コマンドの実行例

^[5]以降は、得に理由がない限り「1.3. 表記方法」のプロンプトの表記を使用します。

- ❶ 「echo」と入力した後、Enter キーを入力すると echo コマンドが実行されます。echo コマンドは引数を指定せずに実行すると改行のみを表示します。
- ❷ 引数を指定して echo コマンドを実行すると、引数に指定された文字列と改行を表示します。
- ❸ echo コマンドには複数の引数を指定することができます。



改行記号の省略

「[図 4.15. ユーザー名とパスワードを入力してログインする](#)」や「[図 4.19. echo コマンドの実行例](#)」で示したように、大抵の場合、ユーザー入力の最後には Enter キーの入力を行います。

これ以降の入力例では、Enter キーを入力するという意味での改行記号は省略します。

4.6. 動作の停止と電源オフ

色々な操作を説明する前に、Armadillo を安全に終了する方法について確認しておきます。

Armadillo の標準状態では、ファイルはすべて RAM ディスク上にあります。RAM ディスクはメモリ (RAM) の一部を仮想的なストレージ^[6]として使う仕組みです。そのため、ファイルに対して色々な操作をおこなっても、電源を切るとすべて消えてしまいます。つまり、どのような操作を(誤って)おこなったとしても、再起動すれば最初からやり直しができることになります。これは、瞬断などが起きやすい環境で動作する組み込みシステムとしては、重要な挙動です。

もちろん、再起動しても変更を保存する方法もあります。そのことについては、「[4.10. 変更した Armadillo の設定を保存する](#)」で説明します。

Armadillo を安全に終了させるには、次のようにコマンドを実行してください。

```
[armadillo ~]# halt
System is going down for system reboot now.

Starting local stop scripts.
Exiting Syslogd!
Syncing all filesystems: done
Unmounting all filesystems: done
The system is going down NOW !!
Sending SIGTERM to all processes.
Sending SIGKILL to all processes.
The system is halted. Press Reset or turn off power
```

「The system is halted. Press Reset or turn off power」と表示されたら、Armadillo の動作は停止します。この状態で AC アダプタを抜くことで、Armadillo を安全に電源オフすることができます。

[6]ハードディスクドライブや USB メモリのような、記憶装置。



危険: 電源遮断時のリムーバブルメディアの扱い

USB メモリや microSD カードなどのリムーバブルメディアにデータを書き込んでいる途中で電源を切断した場合、ファイルシステム、及び、データが破損する恐れがあります。必ず、リムーバブルディスクをアンマウントするか、もしくは `halt` コマンドを実行^[7]してから電源を切断してください。

4.7. ディレクトリとファイルの操作

基本的なコマンドの例として、ディレクトリとファイルを操作するコマンドについて、いくつか説明します。

4.7.1. ディレクトリとファイル

Linux を含む Unix 系 OS のファイルシステムは、ディレクトリとファイルを階層的に配置したものです。ディレクトリは、Windows でいうところのフォルダと同様の概念で、ディレクトリの中に複数のディレクトリとファイルを配置することができます。

ディレクトリとファイルの階層構造を「木の枝分かれ」に例えてディレクトリツリーといい、ディレクトリツリーの「根」の部分に当たる、最上位のディレクトリをルートディレクトリといいます。すべてのディレクトリまたはファイルは、ルートディレクトリから辿ることができます。

自分が現在いるディレクトリを、カレントディレクトリ(またはワーキングディレクトリ)といいます。Linux システムでは、ログインした直後のカレントディレクトリは、ユーザーごとに決まったディレクトリになっています。この、ログイン直後のカレントディレクトリをホームディレクトリといいます。

ディレクトリやファイルの位置を示す文字列を、パスといいます。ルートディレクトリを示すパスは、`/`です。ルートディレクトリの下に `home` ディレクトリがある場合、`home` ディレクトリのパスは `/home` と表します。

パスにはいくつか特殊な意味を持つ文字があります。最初の文字以外にある `/`(即ち、ルートディレクトリを示す/`以外の/`)は、ディレクトリ名の区切りを意味します。例えば、`home` ディレクトリの下に `guest` ディレクトリがある場合、`guest` ディレクトリのパスは、`/home/guest` となります。`~`は、ホームディレクトリを意味します。また、`.`は現在のディレクトリを、`..`は一つ上のディレクトリを意味します。

なお、ルートディレクトリからの位置を示すパスを絶対パスといいます。それに対して、あるディレクトリからの相対的な位置を示すパスを相対パスといいます。`/home/guest` は絶対パスで、`../hoge.txt` は一つ上のディレクトリの `hoge.txt` というファイルを意味する相対パスです。

4.7.2. ディレクトリとファイルを操作するコマンド

本章で使用するディレクトリとファイル操作に関するコマンドを表に示します。

表 4.5 ディレクトリとファイル操作に関するコマンド

コマンド	説明
<code>cd [dir]</code>	ディレクトリを移動します
<code>pwd</code>	カレントディレクトリを表示します

^[7]umount コマンドを使用します。

コマンド	説明
<code>mkdir <dir></code>	ディレクトリを作成します
<code>rmdir <dir></code>	空のディレクトリを削除します
<code>ls [dir]</code>	指定したディレクトリにあるファイルを表示します
<code>cat <file></code>	ファイルの内容を表示します
<code>cp <from> <to></code>	ファイルとディレクトリをコピーします
<code>mv <from> <to></code>	ファイルやディレクトリを移動または名称変更します
<code>rm <file></code>	ファイルとディレクトリを削除します

本章では、以下の内容をコマンドを使って行っていきます。

1. カレントディレクトリを表示する。
2. ディレクトリを移動し、カレントディレクトリが変わっている事を確認する。
3. ホームディレクトリに移動する。
4. 空のディレクトリを作成する。
5. ディレクトリにファイルをコピーする。
6. ディレクトリの内容を表示する。
7. ファイルの内容を表示する。
8. ファイルの名前を変更する。
9. ファイルを削除する。
10. ディレクトリを削除する。

それでは、実際にやってみましょう。

1. カレントディレクトリを表示する。

まず最初に、カレントディレクトリ（現在いるディレクトリ）を調べてみましょう。カレントディレクトリは `pwd` コマンドを使用して調べることができます。

以下のコマンドを実行して、カレントディレクトリを表示させてください。

```
[armadillo ~]# pwd
/root
```

`pwd` コマンドを実行すると `/root` という結果が表示されました。これでカレントディレクトリは `/root` ディレクトリであるとわかりました。

2. ディレクトリを移動し、カレントディレクトリが変わっている事を確認する。

次に `/home` ディレクトリに移動してみましょう。ディレクトリの移動に使用するコマンドは `cd` コマンドです。 `cd` コマンドの引数として `/home` ディレクトリを絶対パスで指定してください。

以下のコマンドを実行して、ディレクトリを移動してください。

```
[armadillo ~]# cd /home
[armadillo /home]# pwd
/home
```

これでカレントディレクトリが `/home` ディレクトリに変わりました。

3. ホームディレクトリに移動する。

次はホームディレクトリに戻ってみましょう。先ほどは `cd` コマンドに移動したいディレクトリを引数として使用しましたが、ホームディレクトリに移動する場合、引数は必要ありません。

以下のコマンドを実行して、ホームディレクトリに移動してください。

```
[armadillo /home]# cd
[armadillo ~]# pwd
/root
```

カレントディレクトリが `/root` ディレクトリに戻りました。これでディレクトリを移動することができますようになりました。

4. 空のディレクトリを作成する。

次は、ディレクトリの操作について説明します。まずはディレクトリを作成してみましょう。ディレクトリを作成するコマンドは `mkdir` コマンドです。

以下のコマンドを実行して、ディレクトリの作成をしてください。

```
[armadillo ~]# mkdir dir
```

`mkdir` コマンドで `dir` ディレクトリを作成しました。

ディレクトリを作成したことを確認するには `ls` コマンドを使用します。 `ls` コマンドは引数に指定したディレクトリにある、ファイルを表示します。引数に何も指定しなかった場合は、カレントディレクトリにあるファイルを表示します。

以下のコマンドを実行して、カレントディレクトリにあるファイルを表示してください。

```
[aramadillo ~]# ls
dir/
```

さきほど作成した `dir` ディレクトリが表示されます。

5. ディレクトリにファイルをコピーする。

ディレクトリを作成しましたので、ファイルを `dir` ディレクトリに保存してみましょう。すでに Armadillo 上にあるファイルを `dir` ディレクトリにコピーしましょう。ファイルをコピーするコマンドは `cp` コマンドです。

以下のコマンドを実行し、ホスト名を設定するファイル (`/etc/config/HOSTNAME`) を `dir` ディレクトリにコピーしてください。^[8]

```
[armadillo ~]# cp /etc/config/HOSTNAME dir
```

`dir` ディレクトリに `HOSTNAME` ファイルをコピーしました。

^[8]Linux ではファイル名の太文字、小文字を区別します。本書に記載されている通りにコマンドを入力してください。



シェルの補完機能

長いファイル名を間違えずに入力するのは、大変です。そのため、シェルにはコマンドやパスを補完してくれる機能があります。

例えば、以下のように「cp /e」まで入力したあと、Tab キーを入力すると「cp /etc/」まで補完してくれます。

```
[armadillo ~]# cp /eTab
[armadillo ~]# cp /etc/
```

候補が複数ある場合は、2 回タブを入力することで、候補を列挙してくれます。例えば、以下のように「cp /etc/」まで入力したあと、Tab キーを 2 回入力すると、「/etc」以下のディレクトリがすべて表示されます。

```
[armadillo ~]# cp /etc/TabTab
/etc/avahi/      /etc/dhcpc/      /etc/ppp/        /etc/ssh/
/etc/config/     /etc/init.d/     /etc/rc.d/       /etc/terminfo/
/etc/default/    /etc/network/    /etc/snmp/       /etc/udev/
```

補完は、コマンドにも有効です。そのため、プロンプトで 2 回 Tab キーを入力すると、実行可能なコマンドをすべて表示してくれます。

```
[armadillo ~]# TabTab
#[
addgroup      fold          makedevs      spawn-fcgi
adduser       free          md5sum         ssh
adjtimex      freeramdisk  mesg           ssh-keygen
amixer        fsck         mjpg_streamer sshd
:
:
:
```

6. ディレクトリの内容を表示する。

実際にコピーされているかを ls コマンドを使って確認してみましょう。ls コマンドは引数に dir を指定することで、dir ディレクトリにあるファイルを表示することができます。

以下のコマンドを実行し、dir ディレクトリに HOSTNAME ファイルがあることを確認してください。

```
[armadillo ~]# ls dir
HOSTNAME
```


ls コマンドの結果から、HOSTNAME ファイルが dir ディレクトリにコピーされたことがわかります。

7. ファイルの内容を表示する。

次に、HOSTNAME ファイルの内容を表示してみましょう。ファイルの内容を表示するコマンドは cat コマンドです。

以下のコマンドを実行し dir/HOSTNAME ファイルの内容を表示してください。

```
[armadillo ~]# cat dir/HOSTNAME
armadillo440-0
```

cat コマンドの結果として armadillo440-0 と表示されます。armadillo440-0 というのは HOSTNAME ファイルに書かれている内容が表示されたものです。

8. ファイルの名前を変更する。

次は、ファイルの名前を変更してみましょう。ファイルの名前を変更するコマンドは mv コマンドです^[9]。

以下のコマンドを実行し、HOSTNAME ファイルの名前を name ファイルに変更してください。

```
[armadillo ~]# mv dir/HOSTNAME dir/name
[armadillo ~]# ls dir
name
[armadillo ~]# cat dir/name
armadillo440-0
```

ls コマンドを使用して dir ディレクトリのファイルを見ると、HOSTNAME ファイルがなくなって、代わりに name ファイルが出来ています。cat で name ファイルの中身を見てみると、確かに HOSTNAME ファイルと同じ中身のようです。

9. ファイルを削除する。

次は、ファイルを削除してみましょう。ファイルを削除するコマンドは rm コマンドです。

以下のコマンドを実行し、dir/name ファイルを削除してください。「rm :remove 'dir/name'?」と削除してもよいかの確認がでますので、y キーを押してから、Enter キーを押してください。

```
[armadillo ~]# rm dir/name
rm: remove `dir/name'? y
[armadillo ~]# ls dir
```

ls コマンドを使用して dir ディレクトリのファイルを見ると、name ファイルが削除されていることがわかります。

10. ディレクトリを削除する。

^[9]mv コマンドは「ファイルやディレクトリを移動する」コマンドですが、移動先の名前を指定できるので「ファイルやディレクトリの名前を変更する」コマンドとして使用できます。

次に、ディレクトリを削除してみましょう。ディレクトリの削除をするコマンドは `rmdir` コマンドです。

以下のコマンドを実行し、`dir` ディレクトリを削除してください。

```
[armadillo ~]# rmdir dir
[armadillo ~]# ls
```

`ls` コマンドを実行しても何も表示されません。これで `dir` ディレクトリが削除されたことがわかります。

以上で、ディレクトリとファイルを扱うための基本的なコマンドは終了です。

次の章ではファイルを作成、編集するための方法を説明していきます。

4.8. ファイルの編集 / vi エディタ

Linux では、多くのアプリケーションの設定がテキスト形式で保存されており、テキストエディタを使用する機会が頻繁にあります。本章では、ほとんどの Linux システムで標準でインストールされている `vi` エディタの使い方を簡単に説明します。

`vi` エディタは、Windows で一般的なエディタ(メモ帳など)とは異なり、モードを持っていることが大きな特徴です。

`vi` のモードには、コマンドモードと入力モードがあります。コマンドモードの時に入力した文字はすべてコマンドとして扱われます。入力モードでは、通常のエディタと同じように、文字の入力ができます。

4.8.1. vi の起動

`vi` を起動するには、以下のコマンドを入力します。

```
[armadillo ~]# vi [ファイル名]
```

`file` オプションにファイル名のパスを指定すると、ファイルの編集(指定されたファイルが存在しない場合は新規作成)を行ないます。

`vi` はコマンドモードの状態です。

4.8.2. 文字の入力

`vi` エディタでの文字の入力は、入力モードで行うことができます。`vi` エディタは起動直後はコマンドモードになっているため、文字を入力するにはコマンドモードから入力モードへ移行しなければなりません。

コマンドモードから入力モードに移行するには、「表 4.6. 入力モードに移行するコマンド」に示すコマンドを入力します。入力モードへ移行後は、キーを入力すればそのまま文字が入力されます。

コマンドはすべて、コマンドモードでのキー入力により実行できます。例えば、「`i`」コマンドを実行したい場合は、`i` キーを押すことで実行できます。

入力モードからコマンドモードに戻りたい場合は、`ESC` キーを入力することで戻ることができます。現在のモードがわからなくなった場合は、`ESC` キーを入力し、一旦コマンドモードへ戻ることにより混乱を防げます。



注意: 日本語変換機能を OFF に

vi のコマンドを入力する時は日本語変換機能(IME 等)を OFF にしてください。


表 4.6 入力モードに移行するコマンド

コマンド	動作
i	カーソルのある場所に挿入
a	カーソルの後ろに挿入

入力モードに移行するコマンドには、「i」と「a」の二つがあります。「i」、「a」それぞれのコマンドを入力した場合の、文字入力の開始位置を「図 4.20. 入力モードに移行するコマンドの説明」に示します。



図 4.20 入力モードに移行するコマンドの説明



vi での文字削除

コンソールの環境によっては BS キーで文字が削除できず、「^H」文字が入力される場合があります。その場合は、「4.8.4. 文字の削除」で説明するコマンドを使用し、文字を削除してください。

4.8.3. カーソルの移動

方向キーでカーソルの移動ができますが、コマンドモードで「表 4.7. カーソルの移動コマンド」に示すコマンドを入力することでもカーソルを移動することができます。

表 4.7 カーソルの移動コマンド

コマンド	動作
h	左に 1 文字移動
j	下に 1 文字移動
k	上に 1 文字移動
l	右に 1 文字移動

4.8.4. 文字の削除

文字を削除する場合は、コマンドモードで「表 4.8. 削除コマンド」に示すコマンドを入力します。

表 4.8 削除コマンド

コマンド	動作
x	カーソル上の文字を削除
dd	現在行を削除

「x」コマンド、「dd」コマンドを入力した場合に削除される文字を「図 4.21. 文字を削除するコマンドの説明」に示します。

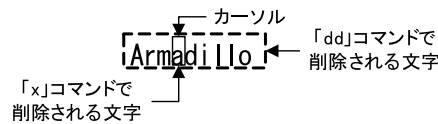


図 4.21 文字を削除するコマンドの説明

4.8.5. 保存と終了

ファイルの保存、終了をおこなうコマンドを「表 4.9. 保存・終了コマンド」に示します。

表 4.9 保存・終了コマンド

コマンド	動作
:q! ↵	変更を保存せずに終了
:w ファイル名 ↵	ファイル名を指定して保存
:wq ↵	ファイルを上書き保存して終了

保存と終了を行うコマンドは「:」からはじまるコマンドを使用します。:キーを入力すると画面下部にカーソルが移り入力したコマンドが表示されます。コマンドを入力した後 Enter キーを押すことで、コマンドが実行されます。

現在編集中のファイルを保存せず終了する場合は「:q!」コマンドを、ファイルを保存して終了する場合は「:wq」コマンドを実行してください。

4.9. ネットワークを使う

Armadillo の標準状態では、起動すると各種サーバーが自動で起動するようになっています。本章では Armadillo で動作している Web サーバーに、ネットワーク越しに作業用 PC からアクセスする方法を説明します。

4.9.1. ネットワークの設定

まずは、ネットワーク接続の設定方法について説明します。

ここでは、「図 4.22. ネットワーク構成例」のネットワークの接続を例として説明します。

PC の IP アドレスを 192.168.0.1 に設定し、以下の手順を行ってください。

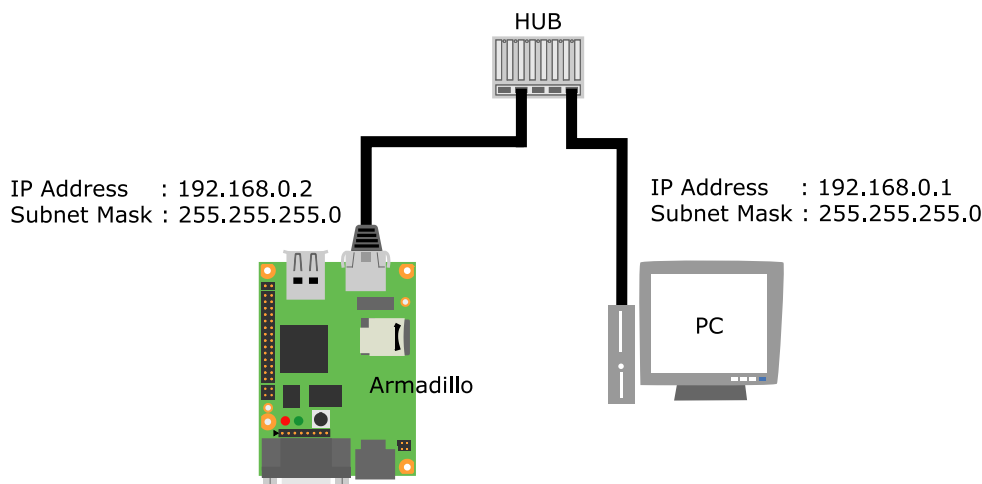


図 4.22 ネットワーク構成例

ネットワークの設定は/etc/config/interfaces で設定されています。

「表 4.10. 固定 IP アドレス設定例」に示す内容に設定変更するには、エディタで/etc/config/interfaces を、「図 4.23. 設定変更後の interfaces ファイル」で示す内容に変更します。

表 4.10 固定 IP アドレス設定例

項目	設定
IP アドレス	192.168.0.2
サブネットマスク	255.255.255.0

```
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)
auto lo eth0
iface lo inet loopback
iface eth0 inet static
address 192.168.0.2
netmask 255.255.255.0
```

図 4.23 設定変更後の interfaces ファイル

4.9.2. vi エディタを使用したネットワーク設定

ここでは vi エディタでネットワークの設定を変更する手順を説明します。

vi エディタの使用方法をご存知の方は、この章を読み飛ばしても構いません。

1. 「図 4.24. interfaces ファイルを開く」に示すコマンドを実行し、interfaces のファイルを開いてください。

```
[armadillo ~]# vi /etc/config/interfaces
```

図 4.24 interfaces ファイルを開く

製品出荷のイメージでは、`/etc/config/interfaces` は「[図 4.25. 標準イメージの interfaces ファイル](#)」のように設定されています。

```
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)

auto lo eth0
iface lo inet loopback
iface eth0 inet dhcp
```

図 4.25 標準イメージの interfaces ファイル

- カーソルを「[図 4.26. interfaces ネットワーク設定](#)」の位置に合わせます。


```
iface eth0 inet  dhcp
```

図 4.26 interfaces ネットワーク設定

- 「x」 コマンドを 4 回入力し「dhcp」を削除します。
- 「a」 コマンドを入力し、入力モードに移行します。
- 「static」と入力し、固定 IP アドレスの設定に変更します。
- Enter キーを入力し改行します。
- 「[図 4.23. 設定変更後の interfaces ファイル](#)」と同じ内容になるよう、「address」と「netmask」を入力してください。
- ESC キーを押して、コマンドモードに戻ります。
- 「:wq」 コマンドを入力して、変更内容を保存し終了します。

以上で、ネットワークの設定は完了です。

4.9.3. ネットワーク設定の反映

「[図 4.27. ネットワークの設定を反映させる](#)」に示すコマンドを実行し、変更したネットワークの設定をシステムに反映させます。

```
[armadillo ~]# ifdown -a
[armadillo ~]# ifup -a
```

図 4.27 ネットワークの設定を反映させる

変更後の IP アドレスは「ifconfig」コマンドで確認できます。

```
[armadillo ~]# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:11:0C:xx:xx:xx
          inet addr:192.168.0.2  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  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:1000
RX bytes:107585 (105.0 KiB) TX bytes:9273 (9.0 KiB)
Base address:0x6000
```

「inet addr」(IP アドレス)が「192.168.0.2」、「Mask」(サブネットマスク)が「255.255.255.0」になっていることを確認してください。

4.9.4. PC から Armadillo の Web サーバーにアクセスする

Armadillo で動作している Web サーバーにアクセスするには、作業用 PC の Web ブラウザのアドレスバーに「http://192.168.0.2/」と入力してください。

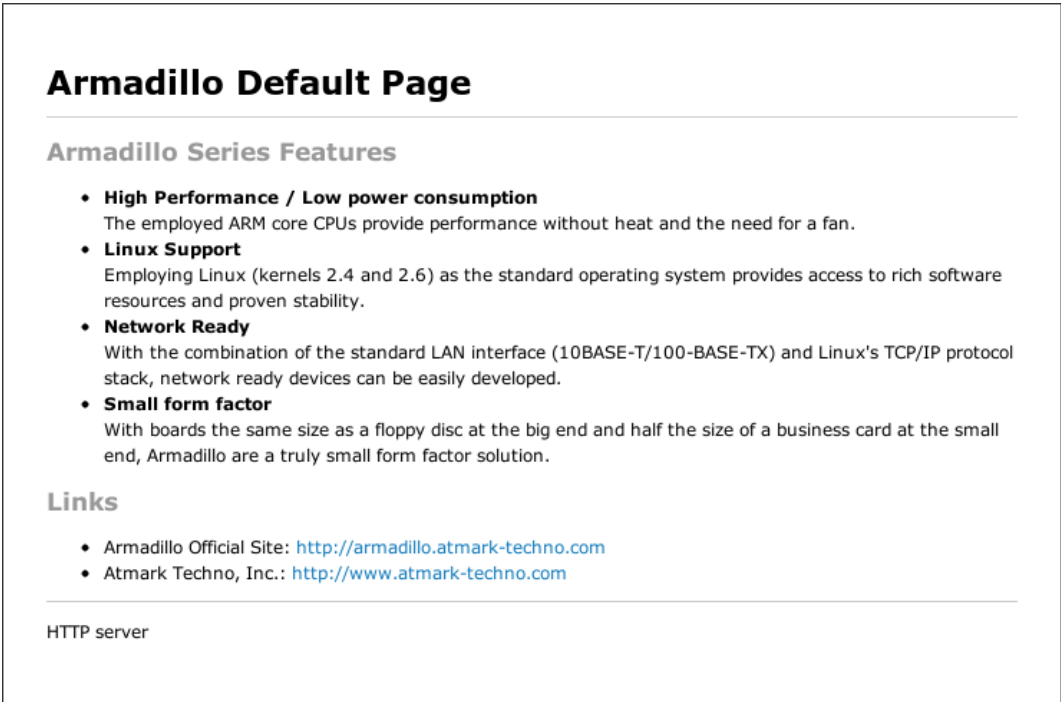
ネットワーク設定が適切になされていれば、Web ブラウザに「 4.28. Web サーバーのトップページ」が表示されます。



図 4.28 Web サーバーのトップページ

4.10. 変更した Armadillo の設定を保存する

「4.6. 動作の停止と電源オフ」で説明したように、Armadillo ではファイルやディレクトリを変更しても再起動を行うと、その変更は消えてしまいます。但し、特定のディレクトリ(/etc/config ディレクトリ)にあるファイルは、変更を保存することができます。

Armadillo では、フラッシュメモリの一部をコンフィグ領域として使用し、そこに/etc/config ディレクトリの内容を書き込むことができます。コンフィグ領域に保存した内容は起動時に復元されるため、/etc/config ディレクトリ内のファイルへの変更を保存することができます。コンフィグ領域からのデータの読み出し、またはコンフィグ領域への書き込みには、flatfsd コマンドを使用します。

これまでに説明した手順を順番に実行してきている場合、/etc/config/interfaces の内容を書き換えているはずですが、この状態で再起動すると、その変更内容が失われます。

下記のように reboot コマンドを入力して、Armadillo を再起動してください。

```
[armadillo ~]# reboot

System is going down for system reboot now.

Starting local stop scripts.
Exiting Syslogd!
Syncing all filesystems: done
Unmounting all filesystems: done
The system is going down NOW !!
Sending SIGTERM to all processes.
Please stand by while rebooting the system.
Hermit-At v2.0.3.1 (armadillo4x0) compiled at 15:31:54, Aug 03 2010
:
:
:
```

再起動後に、`/etc/config/interfaces` の中身を確認すると、修正前のものに戻っているはずですが。

再度、「4.9.1. ネットワークの設定」でおこなったように、`/etc/config/interfaces` を編集してください。

以下の `flatfsd -s` コマンドを実行すると、`/etc/config` ディレクトリの内容が、フラッシュメモリのコンフィグ領域に保存されます。

```
[armadillo ~]# flatfsd -s
```

再起動してみて、`/etc/config/interfaces` の内容が保存されていることを確認してください。

`flatfsd` コマンドは `-s` 以外にもいくつかのオプションを取ります。主な引数を「表 4.11. `flatfsd` の主な引数」に示します。

表 4.11 `flatfsd` の主な引数

引数	動作
<code>-r</code>	コンフィグ領域から設定を読み込む
<code>-s</code>	コンフィグ領域に設定を保存する
<code>-w</code>	コンフィグ領域にデフォルトの設定を保存する ^[1]
<code>-h</code>	<code>flatfsd</code> コマンドのヘルプを表示します

^[1]デフォルトの設定は `/etc/default` ディレクトリに保存されています。

5. Armadillo が動作する仕組み

前章で実際に Armadillo を動かしてみて、Armadillo とはどのようなコンピューターなのか、大体のイメージが掴めたと思います。

この章では、Armadillo がどのようなソフトウェアで構成されていて、それらがどのように動いているのか、その仕組みを詳しく説明していきます。

5.1. ソフトウェア構成

Armadillo は、以下のソフトウェアによって動作します。

5.1.1. ブートローダー

ブートローダーは、電源投入後、最初に動作するソフトウェアです。Armadillo-400 シリーズでは Hermit-At ブートローダー(以降、単に Hermit-At と記述します)を使用します。

Hermit-At は、最低限の初期化処理を行った後、カーネルをブートします。Hermit-At の動作については、「5.2.1. カーネルが動作するまで」で詳しく説明します。

5.1.2. Linux カーネル

Armadillo-400 シリーズでは、標準のカーネルとして Linux 2.6 系を使用します。

標準ではカーネルイメージはフラッシュメモリのカーネルリージョンに配置されます。カーネルイメージは、Hermit-At のブートオプションを変更することで、ストレージ(microSD)または TFTP サーバー上にも配置することができます。

Linux カーネルは、プロセス管理(スケジューリング)、時間管理、メモリ管理、デバイスドライバ、プロトコルスタック、ファイルシステムなどの OS としてのコア機能を提供します。Linux カーネルは、ユーザーランドで動作するプログラムとの唯一の API として、システムコールを提供します。ユーザーランドのプログラムは、必ずシステムコールを通してカーネルの機能呼び出します。

Linux カーネルの初期化処理については、「5.2.2. カーネルの初期化処理」で詳しく説明します。

5.1.3. ユーザーランド

Armadillo-400 シリーズでは、ユーザーランドの標準ルートファイルシステムとして、Atmark Dist と呼ばれるソースコードベースの開発ディストリビューションから作成した initrd^[1]イメージを使用します。



ルートファイルシステムとは

ファイルとディレクトリの階層的な構造をファイルシステムといいます。

^[1]initial RAM disk。一般的な Linux システムでは、initrd は HDD などにあるルートファイルシステムをマウントする前に一時的に使用する「ミニ」ルートファイルシステムとして使用されます。Armadillo-400 シリーズでは、initrd をそのままルートファイルシステムとして使用します。

Windows も階層的なファイル構造を持っていますが、Linux システムとは決定的な違いがあります。それは、Linux システムのファイルシステムにはドライブという考え方がないことです。

Windows では複数のストレージデバイスがある場合、デバイスごとにドライブという形で区別し、別々の階層構造を持ちます。Linux システムでは、複数のデバイスがある場合でも、ルートディレクトリから連なる一つの階層構造として表現します。

ストレージデバイスをファイルシステムと関連付け、システムから使用できるようにすることを、「マウントする」といいます。Linux システムでは、任意のディレクトリにデバイスをマウントすることができます。つまり、USB メモリを/mnt ディレクトリにマウントし、SD カードを/home/atmark/sdcard ディレクトリにマウントするといったことができます。デバイスがどれだけ増えようとも、必ずルートディレクトリから辿ることができます。

ルートファイルシステムとは、ルートディレクトリにマウントされたファイルシステムの事です。

ルートファイルシステムには、アプリケーションプログラム、ライブラリ、設定ファイル、データファイルなどが置かれます。本書では、主にユーザーランドで動作するアプリケーションプログラム開発方法についての解説をおこないます。Linux カーネルのデバイスドライバの開発などは、本書の範囲外になります。

標準では initrd イメージはフラッシュメモリのユーザーランドリージョンに配置され、Hermit-At によって RAM ディスクに展開されます。initrd イメージは、Hermit-At のブートオプションを変更することで、TFTP サーバー上にも配置することができます。

また、ルートファイルシステムは、カーネルパラメータを設定することで、RAM ディスク以外にストレージ(microSD/USB)または NFS サーバー^[2]上に配置することもできます。



標準以外のユーザーランド

Atmark Dist で作成した標準ユーザーランドの他に、オプションとして Debian GNU/Linux 5.0 (コードネーム "lenny")ベースのユーザーランドも提供しています。

Debian GNU/Linux ベースのシステムでは、ソフトウェアパッケージのインストールが一つのコマンドできたり、セルフコンパイル環境を整えることもできます。開発の初期段階や、サーバーとして使用する場合には Debian GNU/Linux をユーザーランドとして選択するのも良いでしょう。

5.2. 起動の仕組み

本章では、標準状態の Armadillo に電源を投入してから、ログイン画面が表示されるまでの起動シーケンスを説明します。

^[2]カーネルで NFS サポートを有効にした場合

ブートローダー、カーネル、ユーザーランドイメージは、フラッシュメモリのそれぞれ対応する領域に書き込まれているものとします。

大まかな起動の流れは、以下のようになります。

1. ブートローダーが起動し、カーネルとユーザーランドを RAM に展開する。
2. ブートローダーを抜けて、カーネルに実行を移す。
3. カーネルが様々なデバイスの初期化処理をおこなう。
4. カーネルがユーザーランドの init というアプリケーションプログラムを実行する。
5. init が様々な初期化処理を呼び出す。
6. init が getty というアプリケーションプログラムを起動する。
7. getty が login というアプリケーションプログラムを起動する。
8. login がログイン画面を表示する。

5.2.1. カーネルが動作するまで

Armadillo シリーズは、汎用 CPU ボードという性格上、ユーザーが書き換え可能なフラッシュメモリを搭載しています。ユーザーが操作を誤ってフラッシュメモリを消去してしまっても復旧が可能なように、Armadillo はジャンパによって、電源投入後の動作(ブートモード)を変更することができます。

Armadillo-400 シリーズでは、JP1 の設定によって UART ブートモードと、オンボードフラッシュメモリブートモードを選択することができます。

JP1 をショート^[3]にすると、UART ブートモードとなります。UART ブートモードは、フラッシュメモリのブートローダーが壊れた場合など、システム復旧のために使用します。詳しくは、「Armadillo-400 シリーズ ソフトウェアマニュアル」の「ブートローダーを出荷状態に戻す」を参照してください。

JP1 をオープン^[4]にしておくと、オンボードフラッシュメモリブートとなります。オンボードフラッシュメモリブートモードでは、フラッシュメモリのブートローダー領域に配置されたブートローダーが起動されます。これが、通常運用での設定です。

Hermit-At ブートローダーは起動されると、まず、DRAM の初期化、拡張インターフェースの IO ポートを入力にするなどの必要最小限の設定をおこないます。その後の動作は、JP2 の設定によって決定されます。JP2 をショートにしておくと保守モード、オープンにしておくとオートブートモードとなります。

保守モードの場合、Hermit-At ブートローダーはプロンプトを表示し、コマンド入力待ちとなります。保守モードでは、フラッシュメモリの更新、ブートデバイスやカーネルパラメータの設定などを行うことができます。詳しくは、「Armadillo-400 シリーズ ソフトウェアマニュアル」の「付録 A Hermit-At ブートローダー」を参照してください。

参考として、「[図 5.1. Hermit-At 保守モード時の表示](#)」に保守モード起動時のシリアルコンソールへの表示を示します。このような表示になった場合は、ジャンパピンの設定を確認してください。

[3]ジャンパソケットを挿した状態

[4]ジャンパソケットを挿していない状態

```
Hermit-At v2.0.1 (armadillo4x0) compiled at 21:10:18, Apr 27 2010
hermit>
```

図 5.1 Hermit-At 保守モード時の表示

ジャンパの設定による、ブートモードの違いを「表 5.1. ジャンパ設定」にまとめます。

表 5.1 ジャンパ設定

JP1	JP2	ブートモード
オープン	オープン	オンボードフラッシュメモリブート/オートブートモード
オープン	ショート	オンボードフラッシュメモリブート/保守モード
ショート	オープン/ショート	UART ブート

Hermit-At ブートローダーは、まず自分自身を RAM にコピーします。

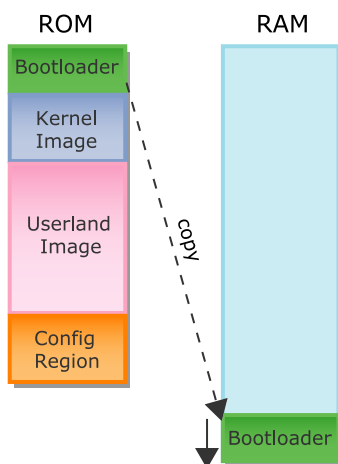


図 5.2 ブートローダーのコピー

次にブートデバイス^[5]からカーネルイメージとユーザーランドの `initrd` イメージを読み込み、RAM 上にコピーします。

Armadillo では、RAM ディスクという仕組みを使い、RAM の一部をブロックデバイス^[6]として使用します。Atmark Dist で作成した `initrd` イメージは、RAM ディスク上にコピーすると EXT2 ファイルシステムとして見えます。Armadillo では、これをルートファイルシステムとして使用します。

^[5]標準設定の場合、フラッシュメモリ。Armadillo-400 シリーズでは microSD も指定可。

^[6]HDD などのブロック単位でアクセスし、ランダムリード/ライトが可能なデバイス。バイト単位でアクセスし、シーケンシャルリード/ライトしかできないデバイスはキャラクタデバイスといいます。

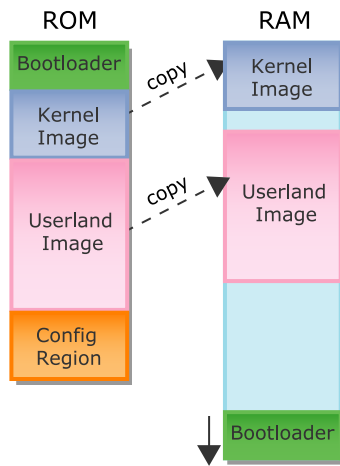


図 5.3 カーネルとユーザーランドをコピー

最後に、カーネルパラメータを RAM 上の特定番地には書き込みます。その後、ブートローダーを抜け、カーネルの開始番地へ実行を移します。この一連の処理を、Linux カーネルをブートすると表現します。

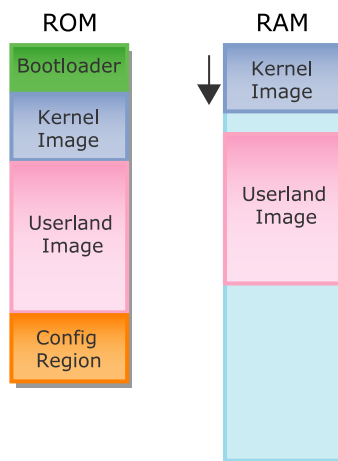


図 5.4 カーネルの起動

Hermit-At ブートローダーが Linux カーネルをブートするときの、コンソールへの出力を「図 5.5. Hermit-At オートブートモード時の表示」に示します。

```

Hermit-At v2.0.1 (armadillo4x0) compiled at 21:10:18, Apr 27 2010 ❶
Uncompressing kernel..... done. ❷
Uncompressing ramdisk..... done. ❸
Linux version 2.6.26-at8 (2.6.26) (atmark@atde3) (gcc version 4.3.2 (Debian 4.3.2-1.1) ) #5 PREEMP
T Fri Jul 9 16:50:08 JST 2010 ❹
:
:
:
    
```

図 5.5 Hermit-At オートブートモード時の表示

- ❶ Hermit-At ブートローダーはシリアルでの初期化が完了すると自身のバージョンを表示します。
- ❷ カーネルイメージが圧縮されている場合、展開しながら RAM 上にコピーします。
- ❸ 同様に、ユーザーランドイメージを展開しながら RAM 上にコピーします。ユーザーランドイメージのコピーが完了すると、カーネルに実行を移します。
- ❹ この行以降は、カーネルが表示しています。

5.2.2. カーネルの初期化処理

ブートローダーから実行を移されると、ようやく Linux カーネルが動作を開始します。

Armadillo のカーネル起動ログの例として、Armadillo-420 の起動ログを「図 5.6. Armadillo-420 カーネルブートログ」に示します。

```

Linux version 2.6.26-at8 (2.6.26) (atmark@atde3) (gcc version 4.3.2 (Debian 4.3.2-1.1) ) #5 PREEMP
T Fri Jul 9 16:50:08 JST 2010
CPU: ARM926EJ-S [41069264] revision 4 (ARMv5TEJ), cr=00053177
Machine: Armadillo-420
Memory policy: ECC disabled, Data cache writeback
CPU0: D VIVT write-back cache
CPU0: I cache: 16384 bytes, associativity 4, 32 byte lines, 128 sets
CPU0: D cache: 16384 bytes, associativity 4, 32 byte lines, 128 sets
Built 1 zonelists in Zone order, mobility grouping on. Total pages: 16256 ❶
Kernel command line: console=ttymxc1,115200 ❷
MXC IRQ initialized ❸
:
各デバイスの初期化処理
:
RAMDISK: ext2 filesystem found at block 0 ❹
RAMDISK: Loading 13621KiB [1 disk] into ram disk... done.
VFS: Mounted root (ext2 filesystem). ❺
Freeing init memory: 132K
init started: BusyBox v1.00 (2010.06.03-06:57+0000) multi-call binary ❻
    
```

図 5.6 Armadillo-420 カーネルブートログ

- ❶ ブートローダーから実行が移ると、カーネルはまず自身のバージョン、CPU アーキテクチャ、マシン名、メモリの状態などを表示します。
- ❷ カーネルパラメータは、「console=ttymxcl,115200」が使用されています。
- ❸ ここから、ボード固有のデバイスの初期化が始まります。
- ❹ RAM ディスクの block0 に ext2 ファイルシステムを検出しました。
- ❺ 検出したファイルシステムをルートファイルシステムとしてマウントします。
- ❻ ルートファイルシステムの所定のパスに init プログラムが見つかり、カーネルは init を実行します。この行から、init によって表示されています。

5.2.3. ユーザーランドの初期化処理

カーネルの初期化が完了すると、最初のプロセスとして init というアプリケーションプログラムが実行されます。init は、まず、コンソールの初期化をおこない、次に/etc/inittab というファイルに書かれた設定に従って、コマンドを実行します。

デフォルトでは、システムコンソールとしてカーネルパラメータの console オプションで指定されたデバイスを使用します。もし、console オプションが指定されていない場合は、/dev/console を使おうとします。

「図 5.6. Armadillo-420 カーネルブートログ」のように、Hermit-At ブートローダーは、標準で console オプションに ttymxcl を指定します。ttymxcl は、Armadillo-400 シリーズのシリアルインターフェース 1 に対応する、シリアルデバイスです^[7]。そのため、init は、シリアルインターフェース 1 をシステムコンソールとして設定します。

次に、init は、/etc/inittab ファイルの中身を読み込み、その内容に従って処理を行います。

inittab には、一行ごとに init が実行すべきコマンドが記述してあります。inittab の書式は、以下のようになっています。

```
id:runlevel:action:process
```

図 5.7 inittab の書式

id には起動されるプロセスが使用するコンソールを指定します。省略した場合は、システムコンソールが使用されます。

runlevel は、Atmark Dist で生成されるユーザーランドに含まれる init では、無効です。

action はどのような状態のときに process に指定したコマンドを実行すべきかをあらわします。action に指定可能な値を「表 5.2. init の action に指定可能な値」に示します。

表 5.2 init の action に指定可能な値

値	いつ process を実行するか
sysinit	システム初期化時。
respawn	sysinit 終了後。このアクションで起動されたプロセスが終了すると、再度 process を実行する。
shutdown	システムをシャットダウンする前。

^[7] 「Armadillo-400 シリーズ ソフトウェアマニュアル」の「UART」参照

値	いつ process を実行するか
ctrlaltdel	Ctrl-Alt-Delete キーの組み合わせが入力されたとき。

例として、Armadillo-420 の inittab は以下のようになっています。

```

::sysinit:/etc/init.d/rc

::respawn:/sbin/getty -L 115200 ttyxc1 vt102

::shutdown:/etc/init.d/reboot
::ctrlaltdel:/sbin/reboot
    
```

図 5.8 Armadillo-420 の inittab

「図 5.8. Armadillo-420 の inittab」からわかるように、システム初期化時には /etc/init.d/rc が実行されます。

Armadillo では、/etc/init.d/rc はシェルスクリプトになっています。シェルスクリプトは、シェルで実行できるコマンドを記述したスクリプトです。詳細は、第 2 部の「シェルスクリプトプログラミング」で説明します。

Armadillo-420 の /etc/init.d/rc は以下の処理を実行します。

1. 一度ルートファイルシステムをリードオンリーでリマウントし、fsck により、ルートファイルシステムをチェックします。問題があればリブートします。
2. ルートファイルシステムをリード/ライト可能でリマウントします。
3. procfs、usbfs、sysfs をマウントします。^[8]
4. ログ関連のファイルをクリアします。
5. /etc/issue、/etc/issuet.net ファイルの設定をします。^[9]
6. /etc/rc.d/ディレクトリにある、S から始まるファイル(初期化スクリプト)を順番に実行します。
7. 赤色 LED が点滅していたら、点滅を停止し赤色 LED を消灯させます。

/etc/rc.d ディレクトリ内には、S からファイル名が始まる、初期化スクリプトがあります。これらの初期化スクリプトで、コンフィグ領域のリストアや、ファイヤーウォールの設定、ネットワークインターフェースの有効化、各サーバーの起動などをおこないます。

また、処理の一番最後に /etc/config/rc.local というファイル名で実行可能なファイルがあるか確認し、ファイルがあればそれを実行します。/etc/config/ディレクトリは、flatfsd でコンフィグ領域に保存可能なディレクトリですので、開発時などに自動起動プログラムを試したいという場合に使用することができます。

sysinit アクションに指定された以上の処理が完了すると、init は次に、respawn アクションに指定されたコマンドを実行します。Armadillo-400 シリーズの場合は、getty というプログラムを起動します。

^[8]これらは、仮想ファイルシステムと呼ばれるもので、カーネル内部の状態がファイル内容に動的に反映されます。

^[9]これらは、コンソール(issue)またはネットワーク経由(issue.net)のログイン時に表示される文字を制御します。


getty は、/etc/issue の内容を読み取り、コンソールに表示します。そして、ログインユーザー名の入力待ちになります。ログインユーザー名が入力されると、それを引数として、login というプログラムを起動します。

login は、ログイン名を指定されて起動されると、パスワードの入力待ちになります。パスワードが入力されると、/etc/passwd 及び/etc/shadow の内容とログインユーザー名、パスワードを照合して、認証をおこないます。ユーザー名とパスワードが一致すると、/etc/passwd で指定されたユーザー用のシェルを起動します。

シェルは、起動されるとプロンプトを表示して、コマンドの入力待ちになります。

5.3. ルートファイルシステムのディレクトリ構成

Linux システムでのディレクトリ構成には、デファクトスタンダード(事実上の標準)となっている構成があります。それぞれのディレクトリに何を格納するかということも、慣習的に決まっています。それぞれのディレクトリには特定の役割がありますので、それを理解することで、ファイルを探す場合にどのディレクトリを探せばよいか、また、ファイルを追加する場合にどのディレクトリに置けば適切かということが分かります。



ディレクトリ構成の標準規格

ディレクトリ構成の標準規格として FHS(Filesystem Hierarchy Standard : ファイルシステム階層標準)があります。

すべての Linux システムがこの標準に従っているわけではありませんが、特別な理由がない限り、FHS に従ったディレクトリ構成とするのが望ましいでしょう。

Armadillo-400 シリーズのルートファイルシステムの主なディレクトリの構成は、以下のようになっています。

表 5.3 ディレクトリ構成

ディレクトリ	ディレクトリの内容
/	ルートディレクトリ、ルートファイルシステムのマウントポイント
/bin	基本的なユーザーコマンドの実行ファイル
/dev	デバイスファイル
/etc	設定ファイル
/etc/config	フラッシュメモリのコンフィグ領域に保存できるファイル
/etc/default	コンフィグ領域の初期化のためのファイル
/etc/init.d	初期化スクリプト
/etc/rc.d	初期化スクリプトへのシンボリックリンク
/home	ホームディレクトリ
/home/ftp	FTP サーバーのドキュメントルート
/home/guest	guest ユーザーのホームディレクトリ
/home/www-data	Web サーバーのドキュメントルート
/lib	基本的なライブラリ
/mnt	一時的にマウントするファイルシステムのマウントポイント

ディレクトリ	ディレクトリの内容
/proc	procfs のマウントポイント(プロセス情報)
/root	root ユーザーのホームディレクトリ
/sbin	基本的なシステム管理用コマンドの実行ファイル
/sys	sysfs のマウントポイント(システム情報)
/tmp	一時的なファイル
/usr	ユーザー共有情報ファイル
/usr/bin	必須でないユーザーコマンドの実行ファイル
/usr/sbin	必須でないシステム管理者用コマンドの実行ファイル
/usr/lib	必須でないライブラリ
/var	頻繁に更新されるファイル
/var/log	ログが保存されるディレクトリ

5.3.1. 実行ファイル

アプリケーションの実行ファイルは、/bin、/usr/bin、/sbin、/usr/sbin ディレクトリに置かれます。

これらのディレクトリ内にあるファイルが、シェルでコマンドを入力したときに検索されます。そしてコマンドと同じファイル名のファイルがこれらのディレクトリに合った場合、そのファイルが実行されます。

5.3.2. ホームディレクトリ

ユーザーごとのホームディレクトリは/home ディレクトリに用意されています。

Armadillo-400 シリーズでは、ftp、guest、www-data ディレクトリがあります。但し、ftp と www-data ユーザーでは、ログインすることはできません。また、root ユーザーでログインした場合のホームディレクトリは、/root ディレクトリになります。

/home/ftp と /home/www-data は、それぞれ FTP サーバーと Web サーバーのドキュメントルートになっています。特に、/home/ftp/pub ディレクトリは、ftp で書き込み可能なディレクトリです。ramfs^[10]でマウントされているので、RAM が許す限りの大きなファイルを書き込むことができます。

5.3.3. ライブラリ

ライブラリファイルは/lib または/usr/lib ディレクトリに置かれます。

共有ライブラリを使用するプログラムを実行する場合は、これらのディレクトリ内に共有ライブラリを置いておく必要があります^[11]。

5.3.4. デバイスファイル

/dev ディレクトリには、デバイスファイルが置かれます。

デバイスファイルは、デバイスを仮想的にファイルとして表したものです。デバイスファイルに対して操作を実行することにより、デバイスを制御することができます。

例として、/dev/ttyxc1 はシリアルインターフェース 1 のシリアルデバイスをファイルとして表したものです。/dev/ttyxc1 に対してデータの読み書きをすることで、シリアルデバイスからデータを受信/送信することができます。

^[10]RAM の一部を直接使用するファイルシステム

^[11]LD_LIBRARY_PATH 環境変数を指定することによって、これらのディレクトリ以外に共有ファイルを置くこともできます。

5.3.5. プロセス、システムの状態

/proc、/sys ディレクトリ内のファイルを操作することで、プロセス、システムの状態を参照、変更することができます。

/proc には procfs、/sys には sysfs がマウントされています。procfs、sysfs はカーネル内部のデータ構造にアクセスすることができる機能を提供する仮想的なファイルシステムです。

5.3.6. ログ

カーネルメッセージやアプリケーションの動作ログなどは/var/log ディレクトリに保存します。

5.3.7. 設定ファイル

設定ファイルは、/etc に置かれます。

/etc/config ディレクトリ以下のファイルは、flatfsd コマンドを使って、フラッシュメモリのコンフィグ領域に保存することができます。コンフィグ領域の内容は、初期化スクリプトで/etc/config ディレクトリにリストアします。

このディレクトリに rc.local という名前で実行可能なファイルを置くと、起動時に初期化スクリプトによって実行されます。

6. 開発環境の構築

これまでに説明したように、Armadillo は ARM プロセッサを搭載したコンピュータです。対して、作業用 PC は、一般に x86(i386)または x86_64(amd64)互換アーキテクチャのプロセッサを搭載しています。そのため、単純に作業用 PC でアプリケーションの(C言語)ソースコードを作成しコンパイルしても、Armadillo 上で動作する実行ファイルを得ることはできません。このように、アプリケーションをコンパイルするコンピュータ(ホストコンピュータ)と、アプリケーションを実行するコンピュータ(ターゲットコンピュータ)のアーキテクチャが異なる場合を、クロス開発といいます。

クロス開発には、ターゲットのアーキテクチャごとに専用のクロスコンパイラやリンカ、アセンブラなどのクロス開発用ツールチェーンが必要です。当然、ライブラリもターゲット用のものを用意しなければなりません。また、組み込み開発では、ターゲットがストレージとしてフラッシュメモリしか持っていない場合が多いので、フラッシュメモリの書き換え手段も必要となります。フラッシュメモリに書き込むためのイメージファイルを作成するツールも必要でしょう。これらのクロス開発用の開発環境を作成することは、一般に手間のかかることです。クロス開発環境を構築しようとして、既存の環境を壊してしまい、OS から再インストールという事態に陥った方も多いのではないのでしょうか^[1]。

開発環境構築に関わる手間をなるべく軽減できるように、Armadillo 用のクロス開発用ツールチェーン、クロスライブラリ、ダウンローダー(フラッシュメモリ書き換えプログラム)などをインストールし、設定済みの環境を ATDE という名称で提供しています。ATDE は、VMware のイメージですので、作業用 PC で動作している OS が Windows でも Linux でも VMware Player が動作すれば、すぐに使用することができます。



VMware とは

コンピュータ上で動作する、ある OS(ホスト OS)の上に仮想のコンピュータ(仮想マシン)を作成し、仮想マシン上でもう一つの OS(ゲスト OS)を動作させることができる、仮想化ソフトウェアの一つです。VMware, Inc.が開発、提供しています。VMware を使うと、Windows PC 上に仮想マシンを作成し Linux を動作させたり、またはその逆をおこなうことができます。

VMware Player は、いくつかある VMware 製品のうち、無償で使用できる製品です。元々は、仮想マシンイメージの実行だけできる製品でしたが、最新版ではイメージの作成をおこなうこともできるようになっています。詳しくは、VMware Player に関する FAQ^[2]を参照してください。

本章では、VMware Player のインストール方法と、ATDE の実行、設定、運用方法について説明します。

ATDE は、本書執筆時点での最新バージョンである ATDE3 を使用します。ATDE3 では、ゲスト OS として Debian GNU/Linux 5.0(コードネーム "lenny")を使用します。

^[1]少なくとも、筆者はそのような経験があります。

^[2]<http://www.vmware.com/jp/products/player/faqs.html>

6.1. Windows PC 上に ATDE を構築する

作業用 PC の OS が Windows の場合に、VMware Player をインストールして、ATDE を実行し、各種設定をおこなう方法を説明します。作業用 PC の OS が Linux の場合は、「6.2. Linux 上に ATDE を構築する」を参照してください。

6.1.1. インストールの前に

ATDE をインストールするには、以下のものがが必要です。

ATDE イメージ	ATDE イメージ(atde3-[<i>version</i>].zip)は、付属 DVD の/atde/vmware フォルダにあります。弊社ダウンロードサイト [http://download.atmark-techno.com/atde/]からも取得できます。
VMware Player インストーラー	VMware Player のインストーラー(VMware-player-[<i>version</i>].exe)は、VMware Player ダウンロードサイト [http://www.vmware.com/jp/download/player/]から取得できます。

以降の ATDE 構築例では、以下のバージョンのファイルを使用し、説明します。

- ・ ATDE イメージ: atde3-20100309.zip
- ・ VMware Player: VMware-player-3.1.1-282343.exe

ATDE イメージは任意のフォルダに展開しておく必要があります。



ATDE イメージの展開

ATDE イメージの圧縮ファイルはサイズが大きいため、展開(解凍)ソフトによっては展開に失敗する場合があります。

Windows 7 の標準機能で正常に展開できることを確認していますので、お使いの展開ソフトで失敗する場合は、標準機能で展開してみてください。

6.1.2. VMware Player のインストール

VMware Player のインストール手順を説明します。

1. ダウンロードした VMware-player-3.1.1-282343.exe を実行すると、インストーラーが起動します。「図 6.1. VMware Player インストール画面 1」が表示されますので、「次へ」ボタンを押します。



図 6.1 VMware Player インストール画面 1

2. インストール先フォルダを指定し、「次へ」ボタンを押します。

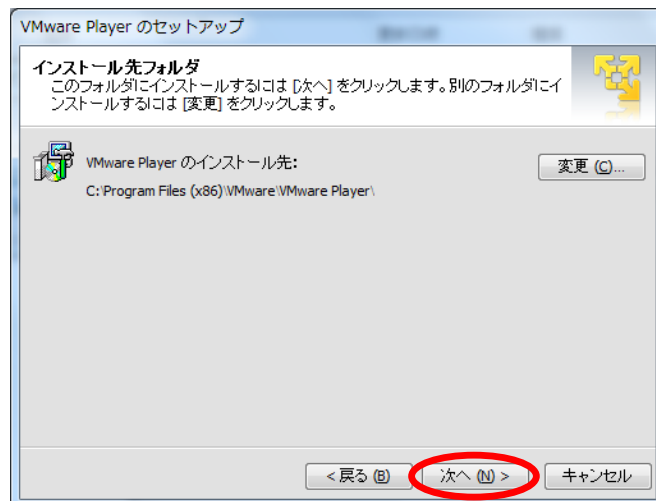


図 6.2 VMware Player インストール画面 2

3. VMware Player 起動時に製品の更新を確認するかどうかを選択し、「次へ」ボタンを押します。

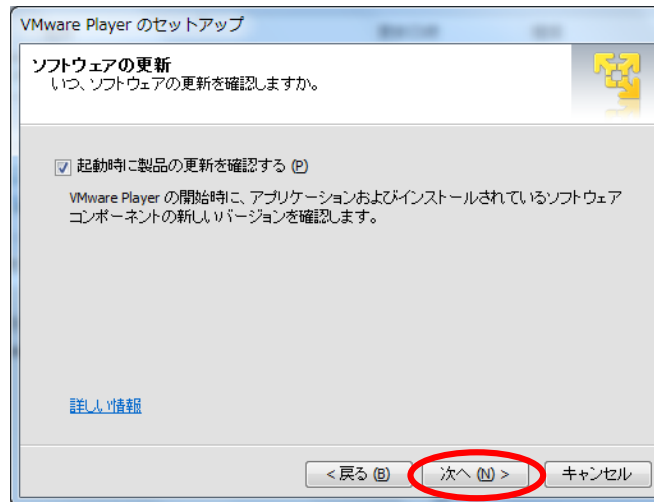


図 6.3 VMware Player インストール画面 3

4. 匿名のシステムデータおよび使用統計を VMware に送信するかどうかを選択し、「次へ」ボタンを押します。

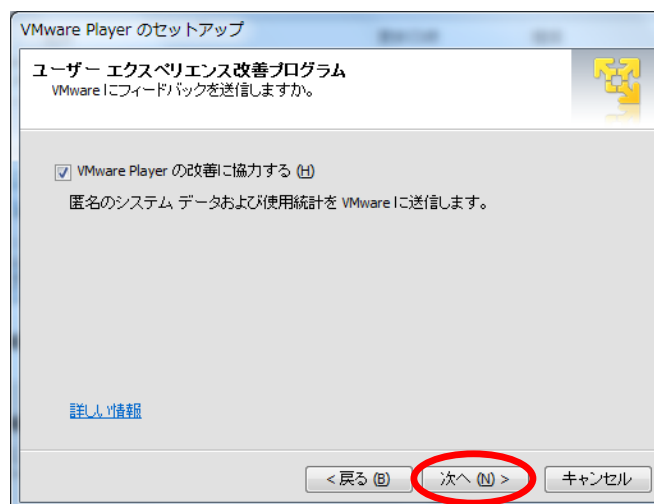


図 6.4 VMware Player インストール画面 4

5. ショートカットを作成する場所を指定し、「次へ」ボタンを押します。

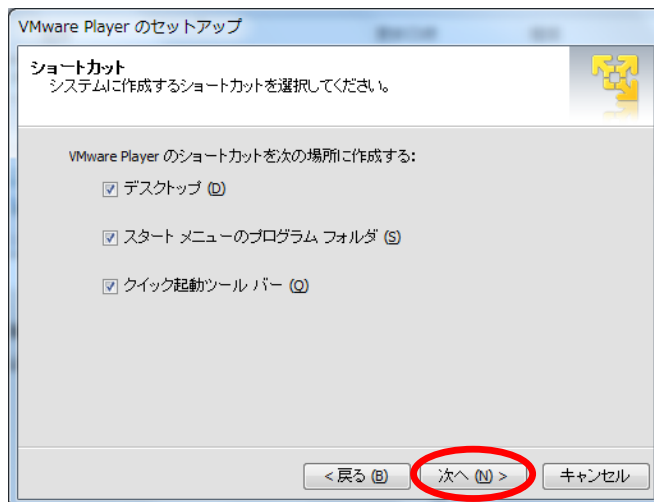


図 6.5 VMware Player インストール画面 5

6. 「続行」 ボタンを押すと、インストールが開始されます。

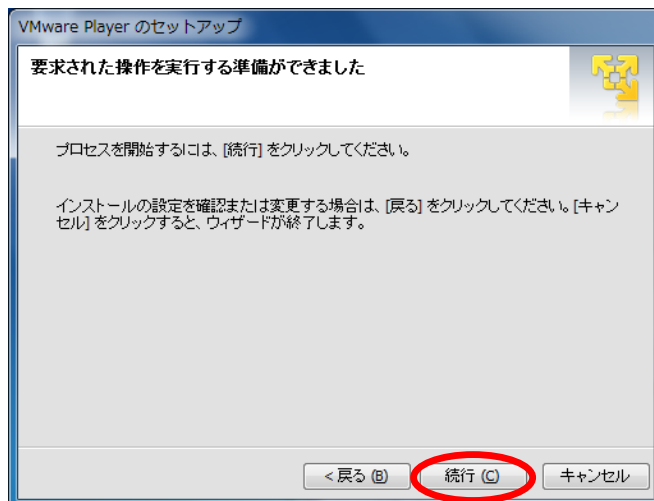


図 6.6 VMware Player インストール画面 6

7. インストールが終了すると、「図 6.7. VMware Player インストール画面 7」が表示されます。「今すぐ再起動」 ボタンを押して再起動してください。

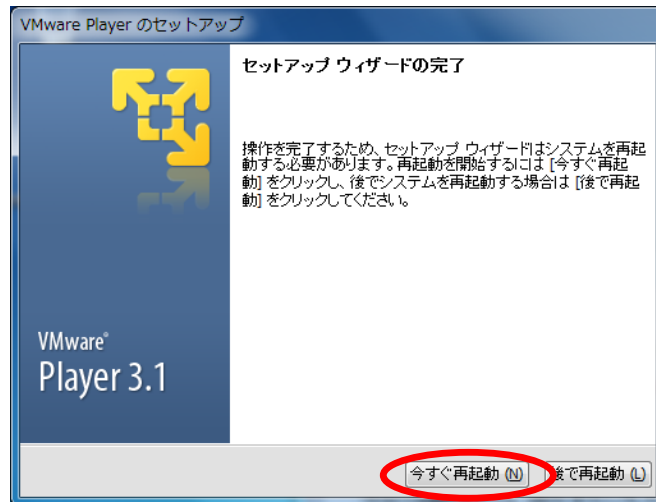


図 6.7 VMware Player インストール画面 7

6.1.3. ATDE の起動

1. インストールした vmplayer.exe を実行します。
2. 「図 6.8. ライセンスの確認」が表示されます。内容を確認し同意する場合には、「使用許諾契約の条項に同意します」をチェックし、「OK」ボタンを押してください。

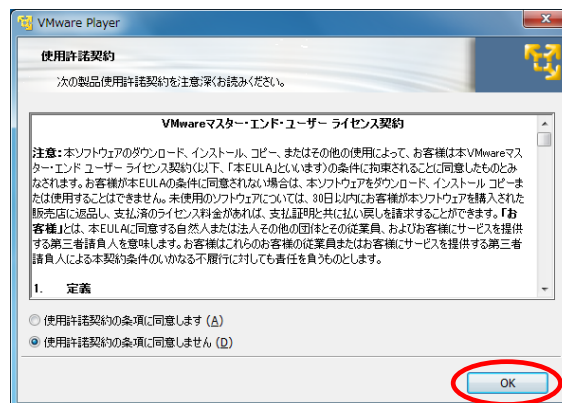


図 6.8 ライセンスの確認

3. VMware Player が起動します。「仮想マシンを開く」を選択してください。



図 6.9 VMware Player 画面

4. ファイルダイアログが開きます。「6.1.1. インストールの前に」で展開した atde3-20100309 フォルダにある atde3.vmx を指定し、「開く」ボタンを押してください。

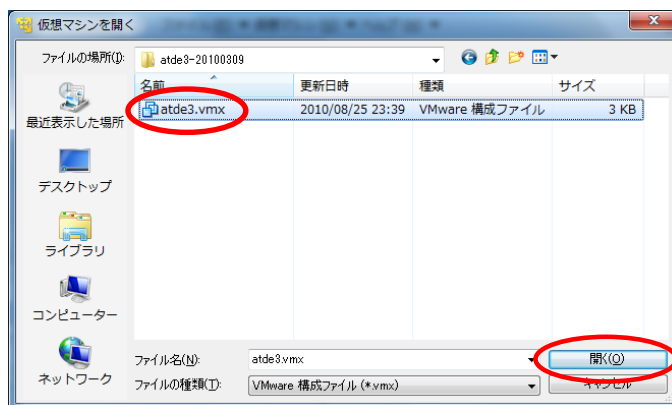


図 6.10 仮想マシンを開く画面

5. 画面左側の「atde3」を選択してください。右側に選択した仮想マシンの情報が表示されます。
6. 「仮想マシンの再生」を選択すると ATDE3 の起動を開始します。

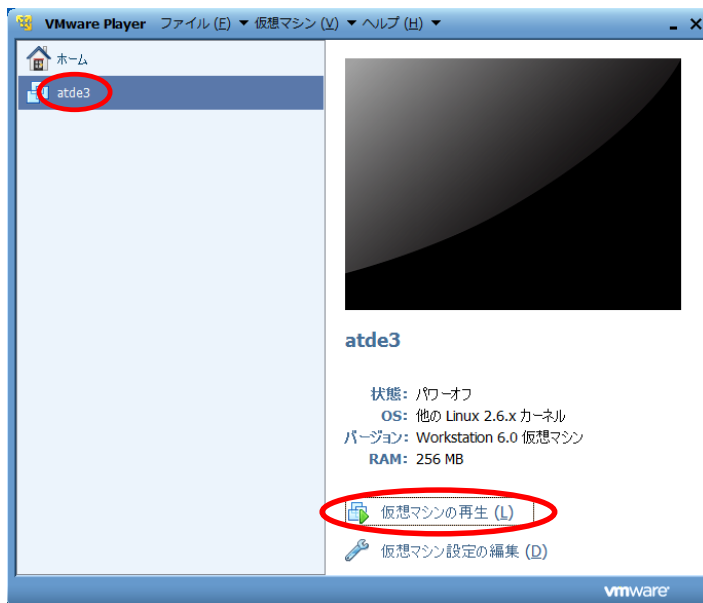


図 6.11 VMware Player 画面

7. ATDE3 が起動すると「図 6.12. ログイン画面」が表示されます。この画面からでは特権ユーザーでログインできませんので、atmark ユーザーでログインしてください。
8. 「図 6.12. ログイン画面」で atmark と入力し、Enter キーを押してください。
9. atmark ユーザーのパスワードを要求されますので、atmark と入力し、Enter キーを押してください。ATDE3 にログインします。



図 6.12 ログイン画面

ログインユーザーは、次の 2 種類が用意されています。

表 6.1 ATDE3 のユーザ名とパスワード

種類	ユーザー名	パスワード
一般ユーザー	atmark	atmark

種類	ユーザー名	パスワード
特権ユーザー	root	root



注意: 特権ユーザーで操作しない

ATDE 上での操作はすべて一般ユーザで実行してください。特権ユーザでの操作が必要になる場合は、sudo コマンドを使用します。

6.1.4. ATDE の終了

ATDE を終了するには、二つの方法があります。

通常は、ATDE3 のウィンドウの「x」 ボタンを押すか、VMware Player の「ファイル」 - 「終了」メニューを選択することで終了します。このとき、ATDE はサスペンド状態で終了します。そのため、終了時点での状態が次回起動時に復元されます。

サスペンドではなく、パワーオフの状態を終了する場合は、以下の手順を行ってください。後述するファイル共有の設定を行うときなどは、ATDE がパワーオフの状態である必要があります。

1. ATDE の「システム」 - 「シャットダウン」メニューを選択してください。

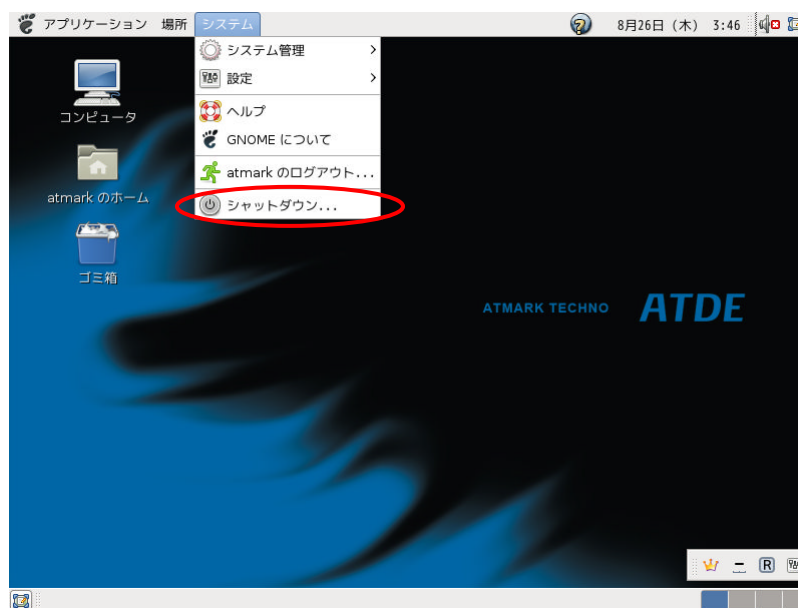


図 6.13 シャットダウンを選択する

2. 「このシステムをシャットダウンしますか？」と表示されている画面が表示されるので、「シャットダウン」 ボタンを押してください。



図 6.14 「このシステムをシャットダウンしますか？」画面

- 画面に「System halted」と表示されると、安全に終了できる状態になります。このままだと、キーボードやマウスの入力を VMWare Player に取られたままになるので、Ctrl+Alt キーを押してください。

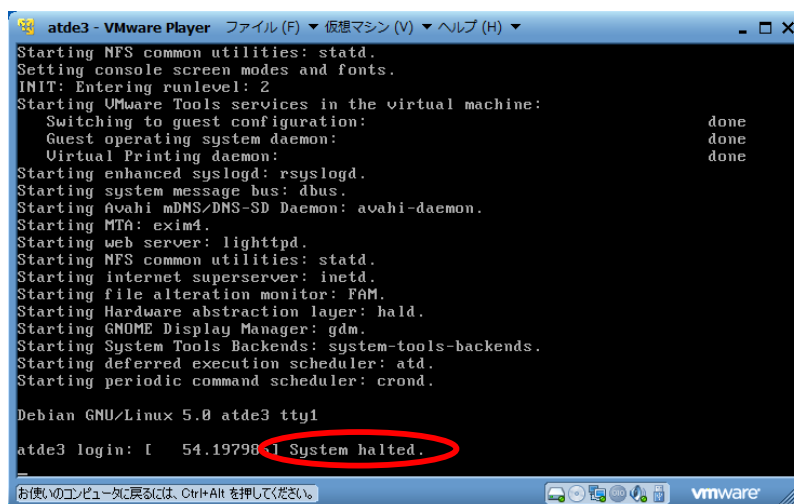


図 6.15 System halted の表示

- VMware Player の「仮想マシン」 - 「パワー」 - 「パワーオフ」メニューを選択すると、仮想マシンがパワーオフ状態となります。

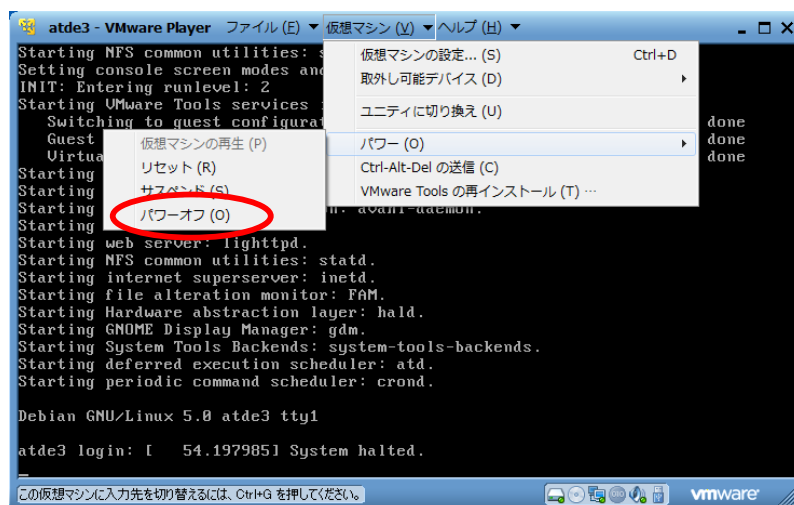



図 6.16 パワーオフを選択する

6.1.5. シリアルポートの設定

ホストである作業用 PC のシリアルポートを、ATDE から使用できるようにする設定方法について説明します。



注意: シリアルポートの共有

ゲスト OS(ATDE)でシリアルポートを使うよう設定する前に、ホスト OS 上で動作しているシリアルポートを使うソフトウェアはすべて終了しておいてください。

シリアルポートの設定は、ATDE が起動した状態で行います。ATDE が起動していない場合は、起動してから以降の作業を行ってください。

1. VMware Player の「仮想マシン」 - 「仮想マシンの設定」メニューを選択してください。

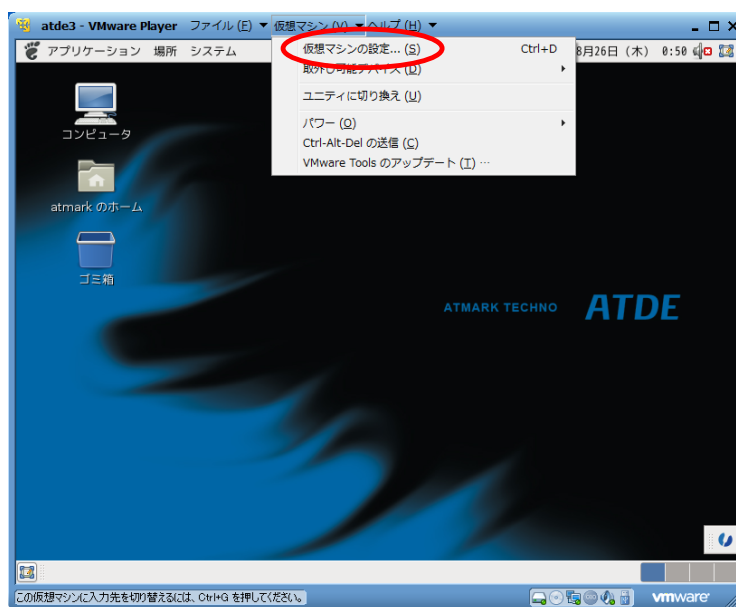


図 6.17 仮想マシンの設定を選択する

2. 「図 6.18. 仮想マシン設定画面」画面左側にある、「シリアルポート」を選択してください。
3. 「図 6.18. 仮想マシン設定画面」画面右側にある、「接続」の「物理シリアルポートを使用」を選択し、コンボボックスで使用するシリアルポートを設定してください。
4. 接続済みにチェックを入れ、ホスト側のシリアルポートを ATDE に接続します。
5. 「OK」ボタンを押し、設定を保存します。

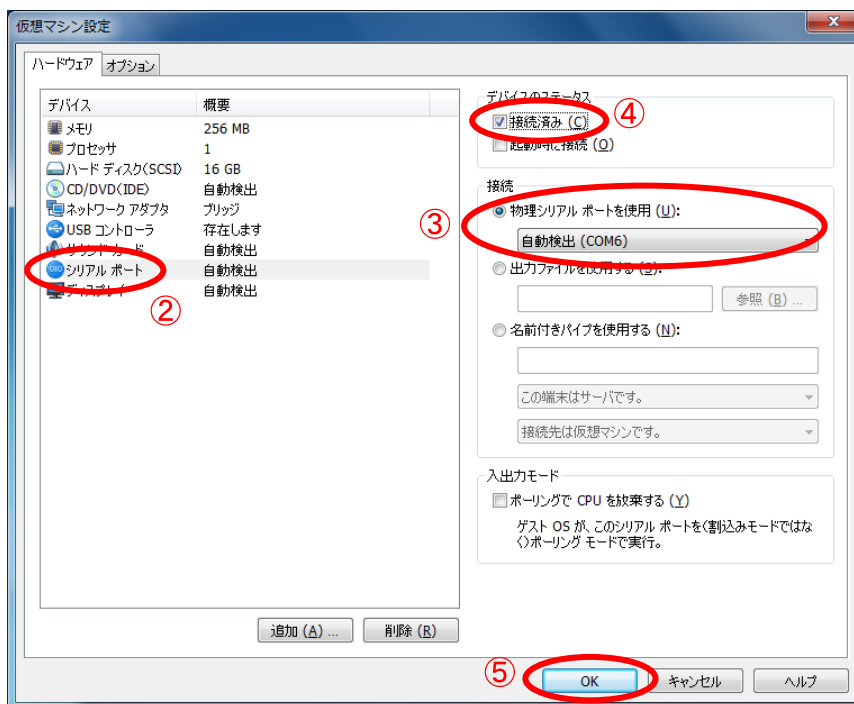



図 6.18 仮想マシン設定画面

- 設定が完了したら、ATDE でシリアルポートが使用できるようになります。シリアルポートのデバイスファイルは/dev/ttyS0 等になります。

6.1.6. 共有フォルダの設定

ホストである作業用 PC 上のディレクトリを、ATDE から使用できるようにする設定方法について説明します。



注意: 共有フォルダの設定をする前に

共有フォルダの設定は、ATDE がパワーオフの状態でおこないます。

サスペンド状態で終了している場合は、一度 ATDE を起動し、「6.1.4. ATDE の終了」に示した手順に従って、パワーオフの状態を終了してください。

- VMware Player を起動します。
- 「図 6.19. VMware Player 画面」で「atde3」を選択します。
- 「状態」が「パワーオフ」になっていることを確認してください。
- 「仮想マシン設定の編集」を選択してください。



図 6.19 VMware Player 画面

5. 「図 6.20. 仮想マシン設定画面」の「オプション」タブを選択してください。
6. 画面左側の「共有フォルダ」を選択してください。
7. 画面右側の「フォルダ共有」の「常に有効」を選択してください。
8. 画面右側の「share」を選択してください。
9. 「プロパティ」ボタンを押し、「図 6.21. 共有フォルダのプロパティ画面」を表示してください。

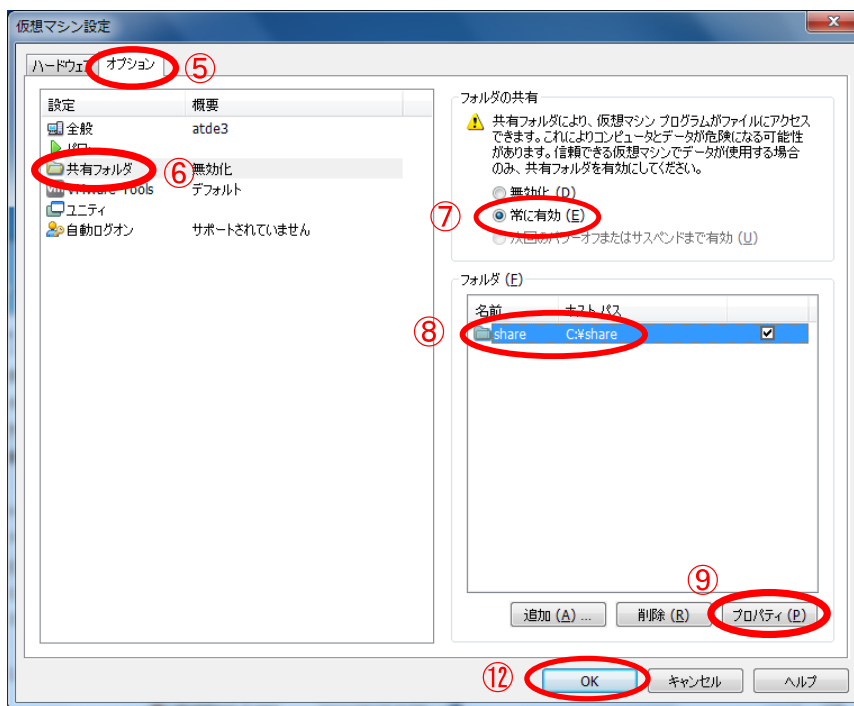


図 6.20 仮想マシン設定画面

10. 「図 6.21. 共有フォルダのプロパティ画面」で、共有したいフォルダを「ホストパス」に入力してください。
11. 「OK」ボタンを押し設定を終了してください。

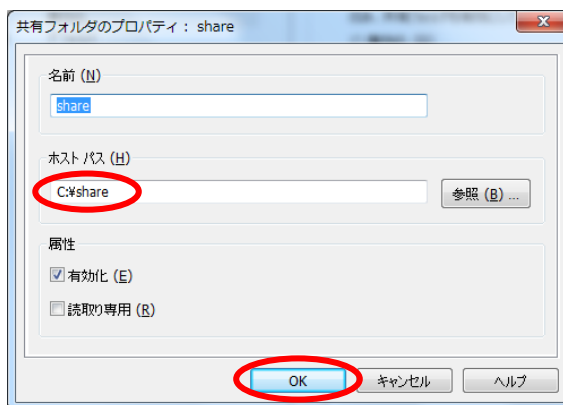


図 6.21 共有フォルダのプロパティ画面

12. 「図 6.19. VMware Player 画面」に戻ってきますので、「OK」ボタンを押し設定を保存してください。
13. ATDE を起動し、ログインしてください。
14. ATDE の「アプリケーション」 - 「アクセサリ」 - 「端末」メニューを選択してください。

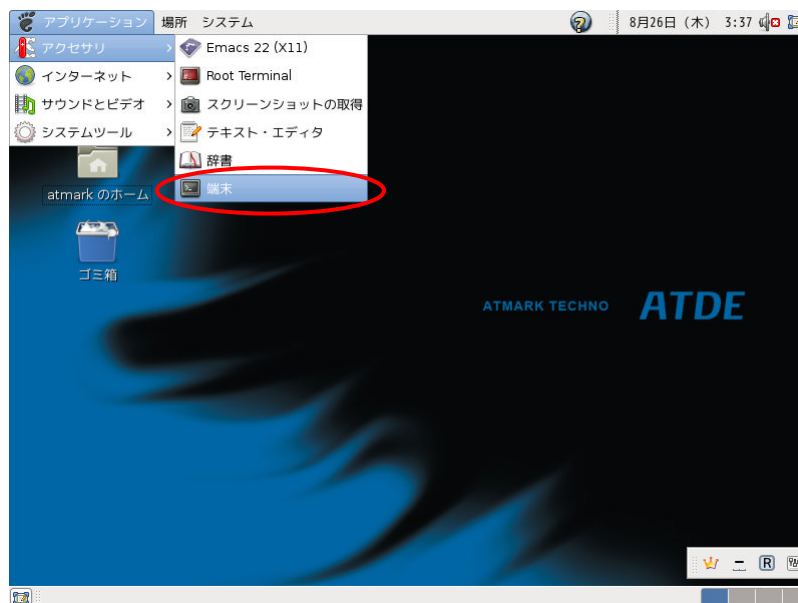



図 6.22 端末の起動

15. 端末で以下のコマンドを実行してください。

```
[ATDE ~]$ sudo mkdir -p /media/hgfs
[sudo] password for atmark:
```

図 6.23 共有フォルダをマウントするディレクトリを作成

コマンド実行時にパスワードを聞かれることがありますので、atmark ユーザーのパスワードを入力してください。



sudo コマンド

sudo コマンドはほかのユーザーとしてコマンドを実行するために使用されるコマンドです。sudo [command]と入力することで特権ユーザーとしてコマンドを実行することができます。コマンド実行時にパスワードの入力が必要になりますが、毎回必要なわけではありません。一度パスワードを入力した後、数分間はパスワードの入力の必要がありません。

16. 端末で以下のコマンドを実行してください。

```
[ATDE ~]$ sudo mount -t vmhgfs .host:/share /media/hgfs
```

図 6.24 共有フォルダをマウントする

以上の手順で、Windows 側のホストパスに設定したディレクトリが、ATDE の/media/hgfs ディレクトリにマウントされます。

6.2. Linux 上に ATDE を構築する

作業用 PC の OS が Linux の場合に、VMware Player をインストールして、ATDE を実行し、各種設定をおこなう方法を説明します。作業用 PC の OS が Windows の場合は、「6.1. Windows PC 上に ATDE を構築する」を参照してください。

6.2.1. インストールの前に

ATDE をインストールするには、以下のものがが必要です。

ATDE イメージ	ATDE イメージ(atde3-[<i>version</i>].zip)は、付属 DVD の/atde/vmware フォルダにあります。弊社ダウンロードサイト [http://download.atmark-techno.com/atde/]からも取得できます。
VMware Player のインストーラー	VMware Player のインストーラー(VMware-Player-[<i>version</i>].bundle)は、VMware Player ダウンロードサイト [http://www.vmware.com/jp/download/player/]から取得できます。

以降の ATDE 構築例では、以下のバージョンのファイルを使用し、説明します。

- ・ ATDE イメージ: atde3-20100309.zip
- ・ VMware Player: VMware-Player-3.1.1-282343.i386.bundle

用意した atde3-20100309.zip を任意のフォルダに展開します。コンソールから以下のコマンドを実行してください。

```
[PC ~]$ unzip atde3-20100309.zip
[PC ~]$ ls
atde3-20100309      atde3-20100309.zip
```

図 6.25 zip ファイルの展開

6.2.2. VMware Player のインストール

VMware Player のインストール手順を説明します。

1. ダウンロードした VMware-Player-3.1.1-282343.i386.bundle を実行し、インストーラーを起動します。コンソールで以下のコマンドを実行してください。

コマンド実行時にパスワードを聞かれることがありますので、atmark ユーザーのパスワードを入力してください。

```
[PC ~]$ chmod +x VMware-Player-3.1.1-282343.i386.bundle
[PC ~]$ sudo ./VMware-Player-3.1.1-282343.i386.bundle
```

図 6.26 インストーラーの起動

2. VMware Player 起動時に製品の更新を確認するかどうかを選択し、「Next」ボタンを押します。

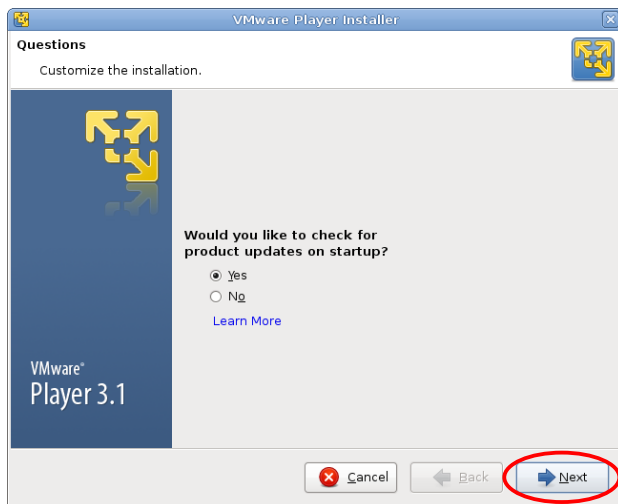


図 6.27 VMware Player インストール画面 1

3. 匿名のシステムデータおよび使用統計を VMware に送信するかどうかを選択し、「Next」ボタンを押します。

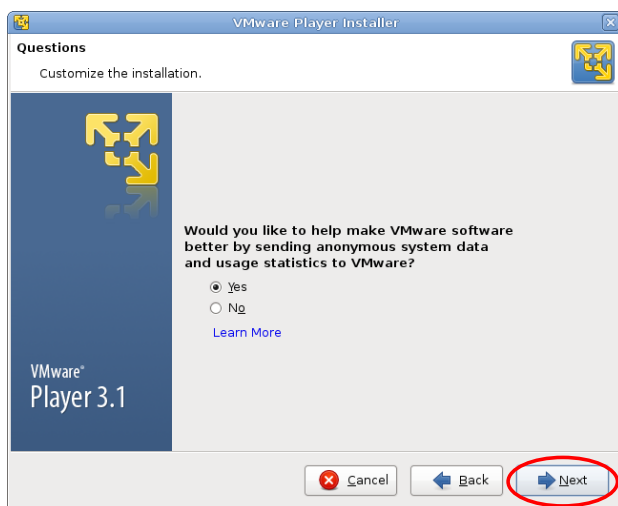


図 6.28 VMware Player インストール画面 2

4. 「Install」ボタンを押すと、インストールが開始されます。

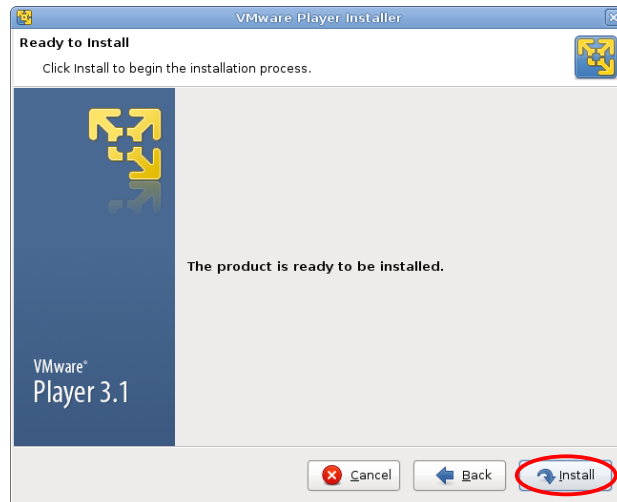


図 6.29 VMware Player インストール画面 3

5. インストールが終了すると、「図 6.30. VMware Player インストール画面 4」が表示されます。「Close」ボタンを押してインストーラーを終了させてください。

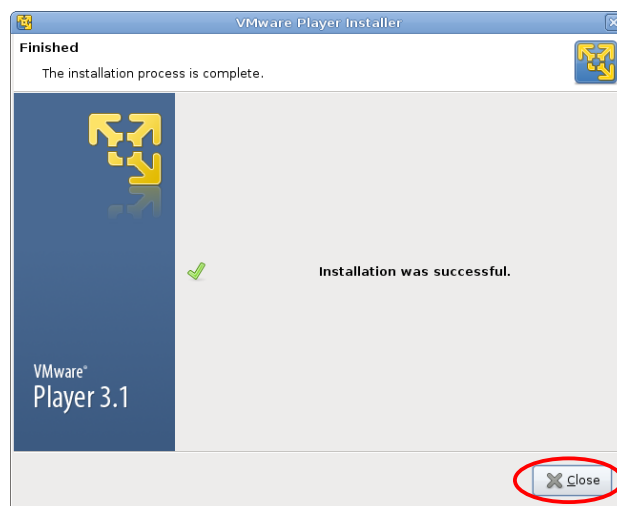


図 6.30 VMware Player インストール画面 4

6.2.3. ATDE の起動

1. 「アプリケーション」 - 「システムツール」 - 「VMware Player」メニューを選択してください。



図 6.31 VMware Player を起動

2. 「図 6.32. ライセンスの確認」が表示されます。内容を確認し同意する場合には、「Accept」ボタンを押してください。

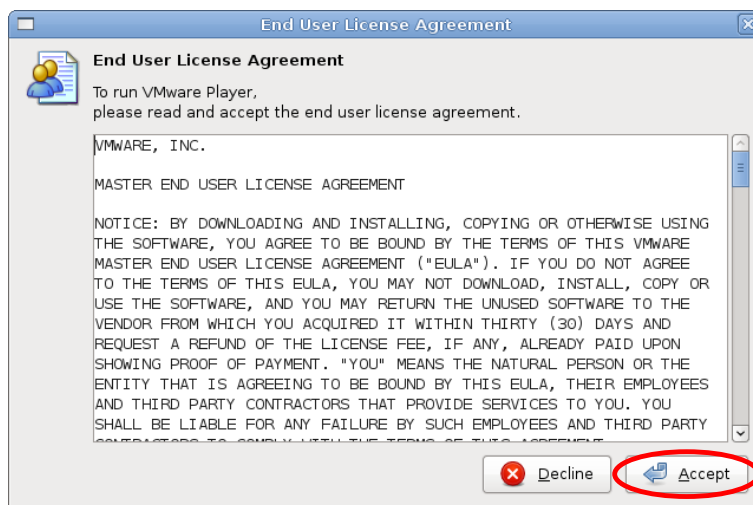


図 6.32 ライセンスの確認

3. VMware Player が起動します。「Open a Virtual Machine」を選択してください。

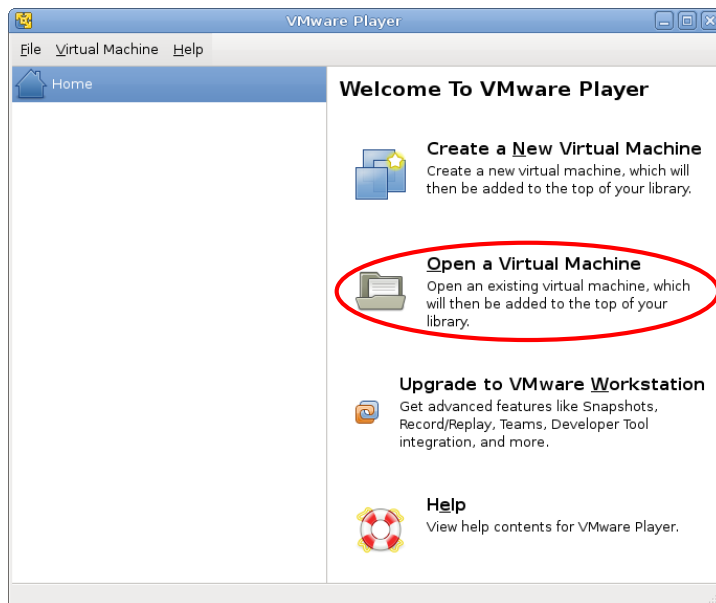


図 6.33 VMware Player 画面

- Linux 上に ATDE を構築するファイルダイアログが開きます。「3.1.2. ATDE イメージの展開」で展開した atde3-20100309 フォルダにある atde3.vmx を指定し、「開く」ボタンを押してください。

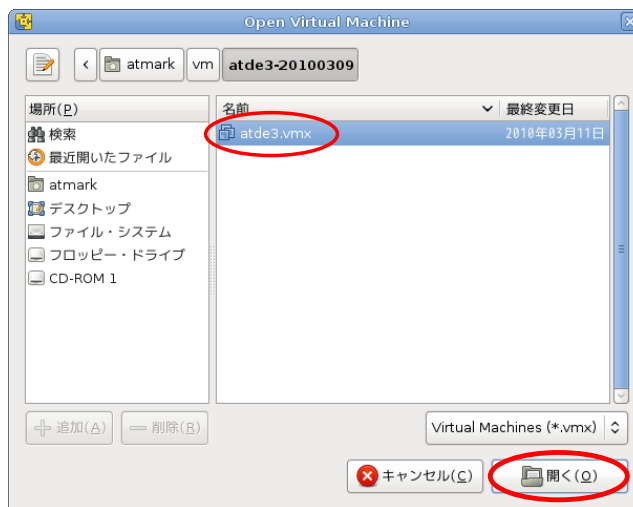


図 6.34 Open Virtual Machine 画面

- 仮想マシンのライブラリに atde3 が登録されました。左側の「atde3」を選択し、「Play virtual machine」を選択すると atde3 の起動を開始します。

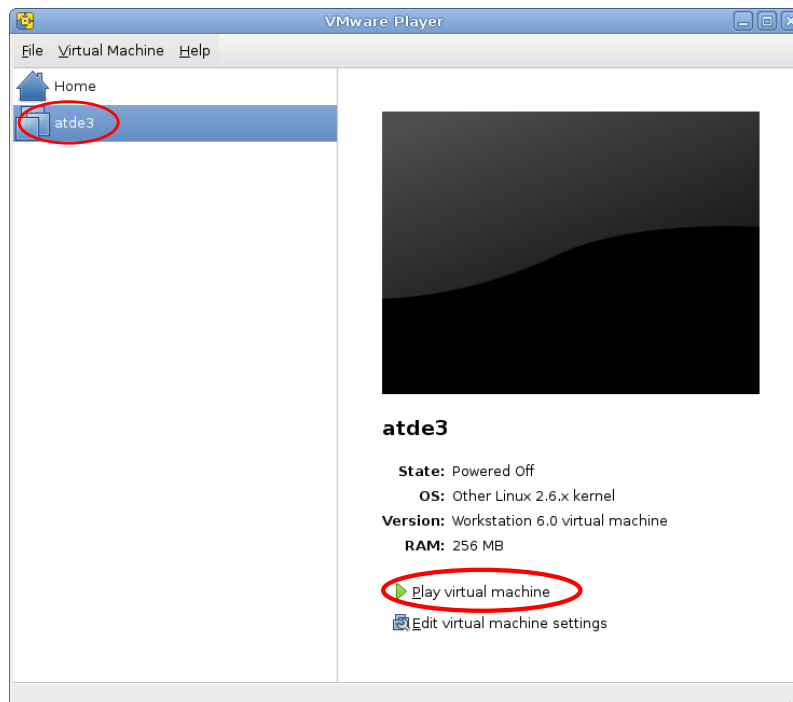


図 6.35 VMware Player 画面

6. ATDE3 が起動すると「図 6.36. ログイン画面」が表示されます。この画面からでは特権ユーザーでログインできませんので、atmark ユーザーでログインしてください。
7. 「図 6.36. ログイン画面」で atmark と入力し、Enter キーを押してください。
8. atmark ユーザーのパスワードを要求されますので、atmark と入力し、Enter キーを押してください。ATDE3 にログインします。




図 6.36 ログイン画面

ログインユーザーは、次の 2 種類が用意されています。

表 6.2 デフォルトのユーザ名とパスワード

種類	ユーザー名	パスワード
一般ユーザー	atmark	atmark
特権ユーザー	root	root



危険: 特権ユーザーで操作しない

ATDE 上での操作はすべて一般ユーザで実行してください。特権ユーザでの操作が必要になる場合は、`sudo` コマンドを使用します。

6.2.4. ATDE の終了

ATDE を終了するには、二つの方法があります。

通常は、ATDE3 のウィンドウの「x」ボタンを押すか、VMware Player の「File」 - 「Suspend and Quit」メニューを選択することで、終了します。このとき、ATDE はサスペンド状態で終了します。そのため、終了時点での状態が次回起動時に復元されます。

サスペンドではなく、パワーオフの状態を終了する場合は、以下の手順を行ってください。後述するファイル共有の設定を行うときなどは、ATDE がパワーオフの状態である必要があります。

1. ATDE の「システム」 - 「シャットダウン」メニューを選択してください。

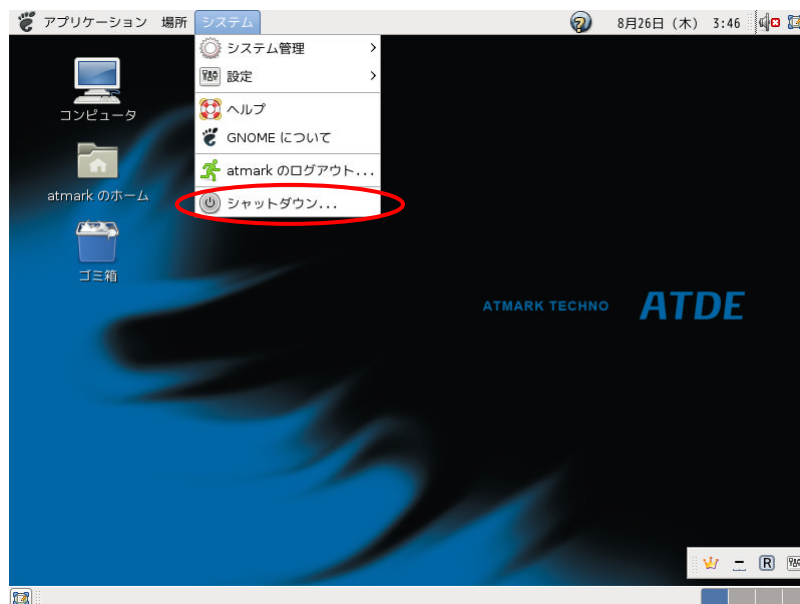


図 6.37 シャットダウンを選択する

2. 「このシステムをシャットダウンしますか？」と表示されている画面が表示されるので、「シャットダウン」ボタンを押してください。

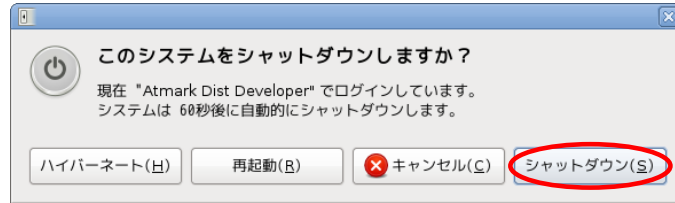


図 6.38 「このシステムをシャットダウンしますか？」画面

3. ATDE 終了後、画面に「System halted」と表示されていることを確認し、Ctrl+Alt キーを押してください。

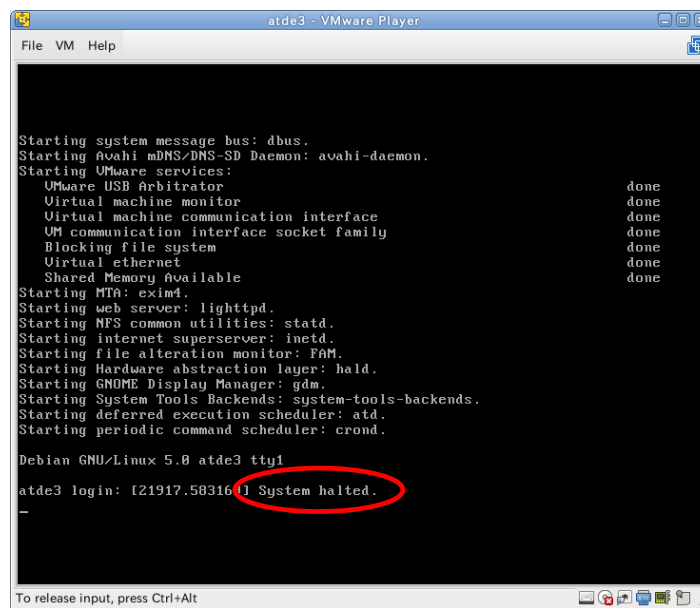


図 6.39 System halted の表示

4. VMware Player の「VM」 - 「Power」 - 「Power Off」メニューを選択すると、仮想マシンがパワーオフ状態となります。

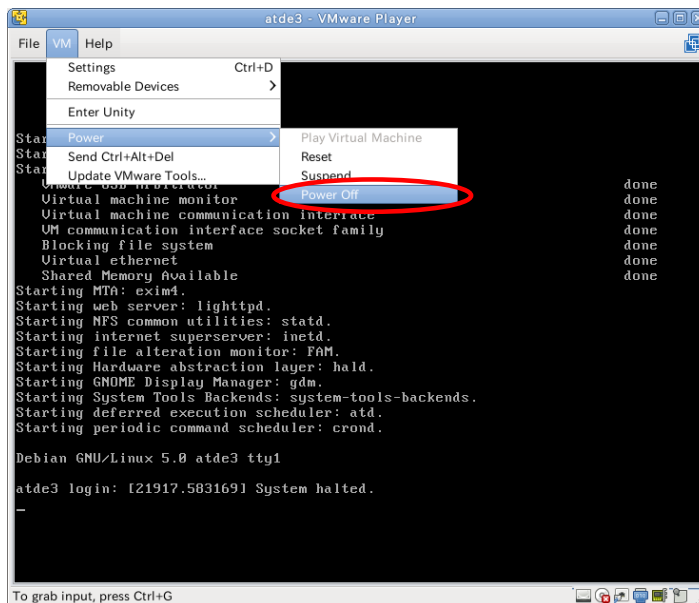



図 6.40 Power Off を選択する

6.2.5. Linux 上でのシリアルポートの設定

ホストである作業用 PC のシリアルポートを、ATDE から使用できるようにする設定方法について説明します。



注意: シリアルポートの共有

ゲスト OS(ATDE)でシリアルポートを使うよう設定する前に、ホスト OS 上で動作しているシリアルポートを使うソフトウェアはすべて終了しておいてください。

シリアルポートの設定は、ATDE が起動した状態で行います。ATDE が起動していない場合は、起動してから以降の設定を行ってください。

1. VMware Player の「VM」 - 「Settings」メニューを選択してください。

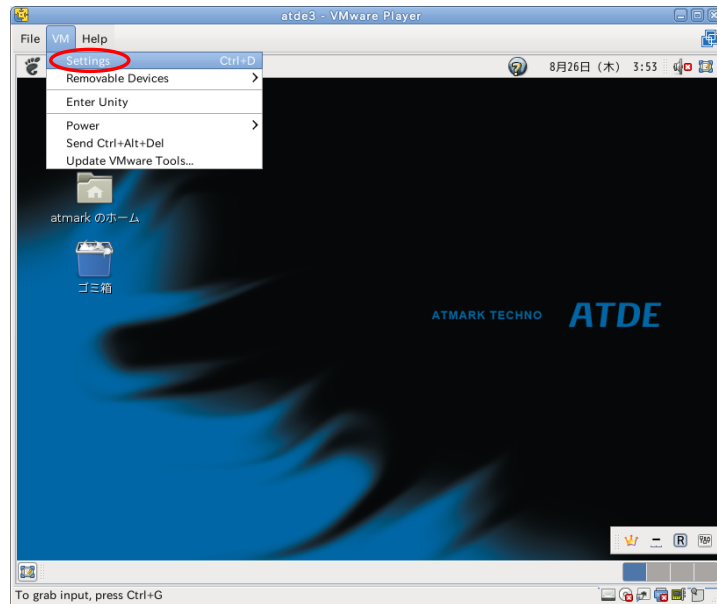


図 6.41 Settings を選択する

2. 画面左側にある「Device」の「Serial Port」を選択してください。
3. 画面右側にある、「Connection」の「Device」にホスト PC のシリアルポートのデバイスファイルを設定してください。
4. 「Connected」チェックボックスをオンにして、ホスト側のシリアルポートを ATDE に接続します。
5. 「Save」ボタンを押し、設定を保存します。

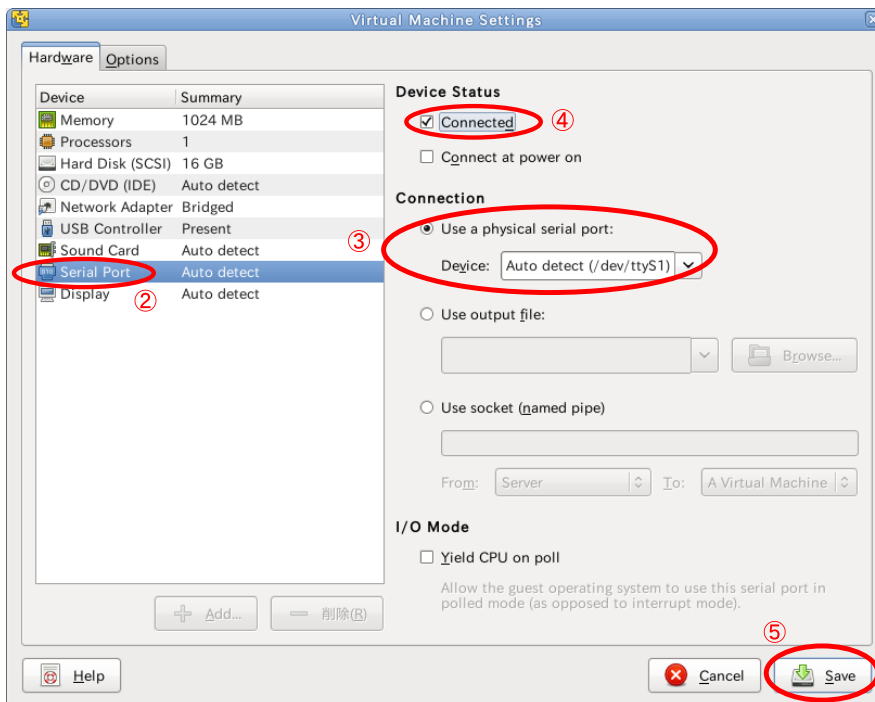



図 6.42 Virtual Machine Settings 画面

- 設定が完了したら、ATDE でシリアルポートが使用できるようになります。シリアルポートのデバイスファイルは/dev/ttyS0 等になります。

6.2.6. 共有フォルダの設定

ホストである作業用 PC 上のディレクトリを、ATDE から使用できるようにする設定方法について説明します。



注意: 共有フォルダの設定をする前に

共有フォルダの設定は、ATDE がパワーオフの状態でおこないます。

サスペンド状態で終了している場合は、一度 ATDE を起動し、「6.1.4. ATDE の終了」に示した手順に従って、パワーオフの状態を終了してください。

- VMware Player を起動します。
- 「図 6.43. VMware Player 画面」で「atde3」を選択します。
- 「State」が「Powered Off」になっていることを確認してください。
- 「Edit virtual machine settings」を選択してください。

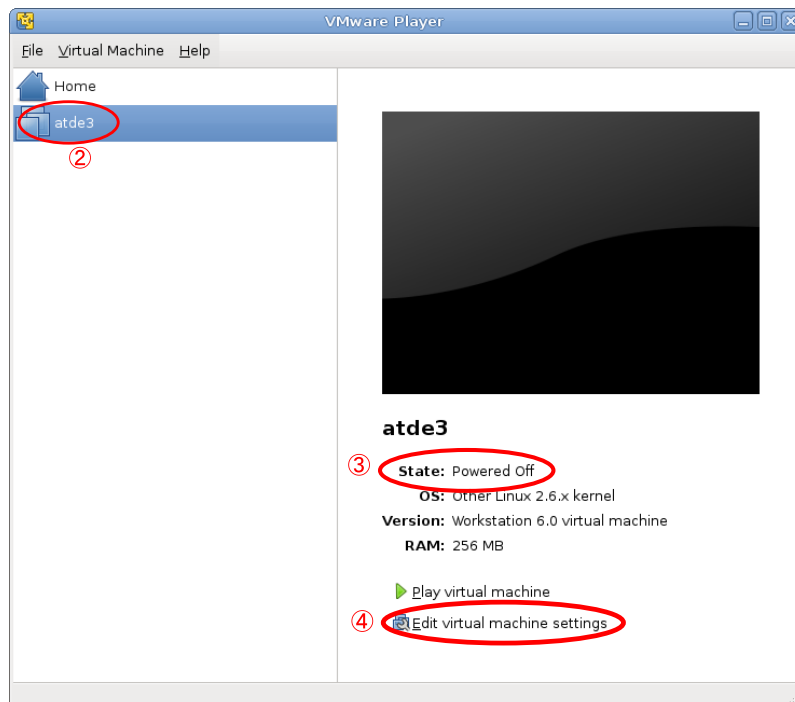


図 6.43 VMware Player 画面

5. 「図 6.44. Virtual Machine Settings 画面」の「Options」タブを選択してください。
6. 画面左側の「Shared Folders」を選択してください。
7. 画面右側の「Folder Sharing」の「Always enable」を選択してください。
8. 画面右側の「share」を選択してください。
9. 「プロパティ」ボタンを押し、「図 6.45. Shared Folder Properties 画面」を表示してください。

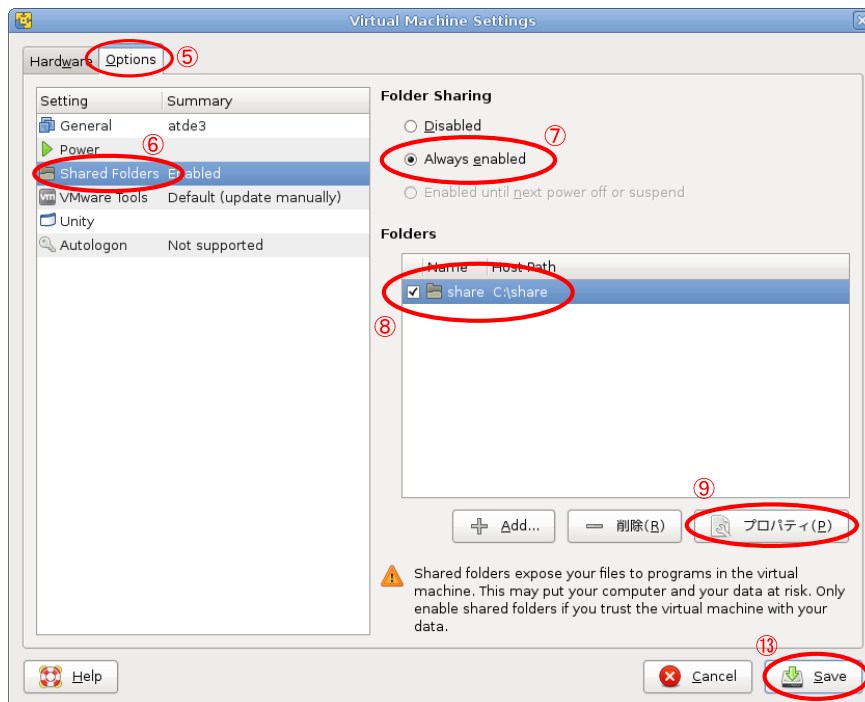


図 6.44 Virtual Machine Settings 画面

10. 「図 6.45. Shared Folder Properties 画面」で、共有したいディレクトリを「Host Path」に入力してください。
11. 「OK」ボタンを押し設定を終了してください。

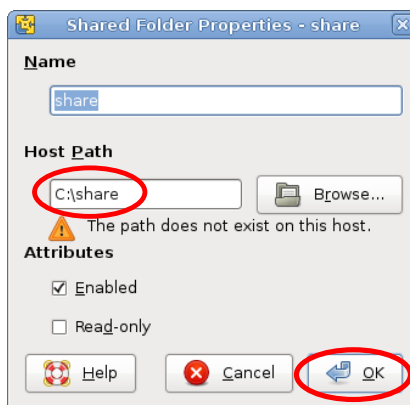


図 6.45 Shared Folder Properties 画面

12. 「図 6.44. Virtual Machine Settings 画面」に戻ってきますので、「Save」ボタンを押して、変更した設定を保存してください。
13. ATDE を起動し、ログインしてください。
14. ATDE の「アプリケーション」 - 「アクセサリ」 - 「端末」メニューを選択してください。

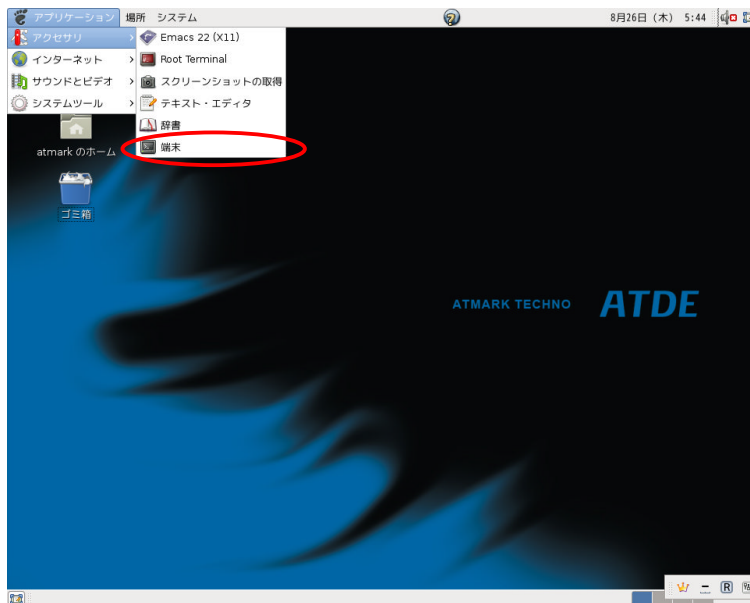



図 6.46 端末の起動

15. 端末で以下のコマンドを実行してください。

```
[ATDE ~]$ sudo mkdir -p /media/hgfs
[sudo] password for atmark:
```

図 6.47 共有フォルダをマウントするディレクトリを作成

16. コマンド実行時にパスワードを聞かれることがありますので、atmark ユーザーのパスワードを入力してください。



sudo コマンド

sudo コマンドはほかのユーザーとしてコマンドを実行するために使用されるコマンドです。sudo *[command]*と入力することで特権ユーザーとしてコマンドを実行することができます。コマンド実行時にパスワードの入力が必要になりますが、毎回必要なわけではありません。一度パスワードを入力した後、数分間はパスワードの入力の必要がありません。

17. 端末で以下のコマンドを実行してください。

```
[ATDE ~]$ sudo mount -t vmhgfs .host:/share /media/hgfs
```

図 6.48 共有フォルダをマウントする

以上の手順で、Linux 側のホストパスに設定したディレクトリが、ATDE の/media/hgfs ディレクトリにマウントされます。

6.3. ATDE のネットワーク設定

本章では、ATDE のネットワーク設定について説明します。

VMware Player で ATDE を実行した場合の、ネットワーク構成は「図 6.49. ATDE のネットワーク構成」のようになります。作業用 PC と ATDE3 は、同じネットワークに参加している別々のコンピューターとして扱われます。^[3]

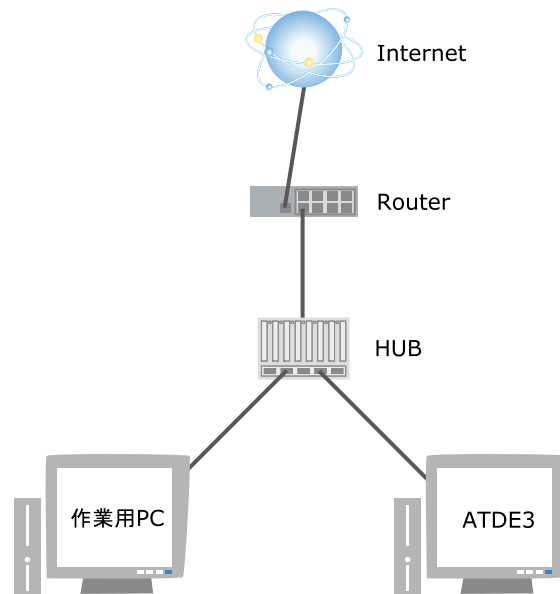


図 6.49 ATDE のネットワーク構成

DHCP を使用して IP アドレスを取得するよう設定する方法を「6.3.1. DHCP 接続の設定」に、固定 IP アドレスに設定する方法を「6.3.2. 固定 IP アドレス接続の設定」に示します。

ATDE の標準の設定では DHCP を使用して IP アドレスを取得する設定になっています。DHCP を使用する場合は標準の設定から変更する必要はありません。

6.3.1. DHCP 接続の設定

本章では、DHCP を使用して IP アドレスを取得するよう設定する方法について説明します。

1. ATDE の画面左上の「アプリケーション」 - 「アクセサリ」 - 「端末」メニューを選択し、端末を起動します。
2. 端末上で以下のコマンドを入力し、interfaces ファイルを gedit^[4]で開きます。

^[3]このような接続方法をブリッジ接続といいます。ネットワークに新しいコンピューターを追加するのが難しい場合などは、作業用 PC の接続を ATDE と共有する NAT 接続にすることもできます。NAT 接続に変更する場合は VMware Player のマニュアルを参照し、設定してください。

^[4]gedit は ATDE3 に標準でインストールされているテキストエディタです。Windows のメモ帳と同じで、簡単に操作することができます。

```
[ATDE ~]$ sudo gedit /etc/network/interfaces
```

図 6.50 ネットワーク設定ファイルを開く

3. interfaces ファイルを「図 6.51. DHCP 接続の interfaces ファイル設定例」のように編集します。

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug eth0
iface eth0 inet dhcp
```

図 6.51 DHCP 接続の interfaces ファイル設定例

4. gedit の「ファイル」 - 「保存」メニューを選択して変更内容を保存してから、gedit を終了してください。
5. 以下のコマンドを入力し、変更したネットワークの設定を ATDE に反映させます。

```
[ATDE ~]$ sudo ifdown eth0
[ATDE ~]$ sudo ifup eth0
```

図 6.52 ネットワーク設定を反映させる

以上の手順でネットワーク設定は完了です。

6.3.2. 固定 IP アドレス接続の設定

本章では、固定 IP アドレスに設定する方法について説明します。ここでは例として、ネットワーク設定を「表 6.3. 固定 IP アドレス設定例」の値に設定します。実際に設定する場合は、ネットワーク環境に応じて設定値を置き換えてください。

ネットワークの設定値についてわからない場合は、ネットワークの管理者に相談してください。

表 6.3 固定 IP アドレス設定例

項目	設定
IP アドレス	192.168.0.10
ネットマスク	255.255.255.0
ネットワークアドレス	192.168.0.0
ブロードキャストアドレス	192.168.0.255
デフォルトゲートウェイ	192.168.0.1
DNS サーバー	192.168.0.2

1. ATDE の画面左上の「アプリケーション」 - 「アクセサリ」 - 「端末」メニューを選択し、端末を起動します。
2. 端末上で以下のコマンドを入力し、interfaces ファイルを gedit^[5]で開きます。

```
[ATDE ~]$ sudo gedit /etc/network/interfaces
```

図 6.53 ネットワーク設定ファイルを開く

3. interfaces ファイルを「図 6.51. DHCP 接続の interfaces ファイル設定例」のように編集します。

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug eth0
iface eth0 inet static
    address 192.168.0.10
    netmask 255.255.255.0
    network 192.168.0.0
    broadcast 192.168.0.255
    gateway 192.168.0.1
```

図 6.54 固定 IP アドレス接続の interfaces ファイル設定例

4. gedit の「ファイル」 - 「保存」メニューを選択して変更内容を保存してから、gedit を終了してください。
5. 固定 IP 接続の場合は interfaces ファイルの他に DNS サーバーの設定をする必要があります。DNS サーバーの設定は/etc/resolv.conf ファイルで行ないます。

/etc/resolv.conf ファイルを開き、DNS サーバーの設定を行います。

```
[ATDE ~]$ sudo gedit /etc/resolv.conf
```

図 6.55 ネットワーク設定ファイルを開く

6. resolv.conf ファイルを「図 6.56. DNS サーバー設定例」のように編集してください。

```
nameserver 192.168.0.2
```

図 6.56 DNS サーバー設定例

^[5]gedit は ATDE3 に標準でインストールされているテキストエディタです。Windows のメモ帳と同じで、簡単に操作することができます。

- gedit の「ファイル」 - 「保存」メニューを選択して変更内容を保存してから、gedit を終了してください。
- 以下のコマンドを入力し、変更したネットワークの設定を ATDE に反映させます。

```
[ATDE ~]$ sudo ifdown eth0  
[ATDE ~]$ sudo ifup eth0
```

図 6.57 ネットワーク設定を反映させる

以上の手順でネットワーク設定は完了です。

6.4. 最新の状態にアップデートする

ATDE の基本的な設定が完了したら、ATDE にインストールされているソフトウェアを最新のものにするため、ソフトウェアアップデートをおこなってください。

端末から以下のコマンドを実行することでソフトウェアアップデートが行えます。

```
[ATDE ~]$ sudo apt-get update && sudo apt-get upgrade
```

7. 開発の基本的な流れ

本章では、Armadillo を使った製品開発を行うために必要な一連の手順を順を追って説明します。

基本的な流れは、以下のようになります。

1. アプリケーションプログラムを作成する
2. Atmark Dist を使ってユーザーランドのルートファイルシステムを作成する
3. ユーザーランドに、作成したアプリケーションプログラムを追加する
4. 量産に向けた準備を行う
5. 製品出荷後のメンテナンスを行う

まずは、製品の機能の特徴づけるアプリケーションプログラムの作成方法について、説明します。アプリケーションプログラムは、C 言語で作成するものとします。これまでに C 言語での開発経験がある方でも、Linux での開発スタイルやクロス開発特有の問題など、注意すべき点がいくつかあります。

次に、Atmark Dist と呼ばれるアットマーク社製品用の開発ディストリビューションを用いて、ユーザーランドのルートファイルシステムを作成する方法について説明します。ここでは、Armadillo の標準と同じルートファイルシステムのイメージファイルを作成し、それを Armadillo に書き込む方法を紹介します。

続いて、標準のルートファイルシステムをカスタマイズする方法について説明します。ユーザーランドに、作成したオリジナルのアプリケーションプログラムや設定ファイルを追加する方法を紹介します。

一通りの開発が完了したら、量産に向けた準備を行います。アットマークテクノでは、Armadillo を使った製品の量産をなるべく簡単に行えるよう、カスタマイズサービスを提供しています。

最後に、製品出荷後のメンテナンスについて気をつけなければならないことについて説明します。

本章では、Armadillo を使った場合の開発サイクルの全体像を掴んでもらうために、各手順の概略的な説明のみをおこないます。詳細な説明は、第 2 部でおこないますので、そちらをご参照ください。

7.1. アプリケーションプログラムの作成

この章では、C 言語でアプリケーションプログラムのソースコードを作成し、コンパイル、実行する方法について説明します。

Linux でのアプリケーション開発は初めてという方でも読み進められるように、コンパイルと実行は、まずホストである作業用 PC でおこなう方法を説明したあと、ターゲットとなる Armadillo でおこなう方法について順を追って説明します。

7.1.1. Hello World!

まずは、定番である「Hello World!」を表示するだけのアプリケーションプログラムを作成し、実行してみましょう。

以下に示す、「図 7.1. hello.c」を作成し、atmark ユーザーのホームディレクトリ(/home/atmark)に保存してください。

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

int main(int argc, char *argv[])
{
    printf("Hello World!\n");

    return EXIT_SUCCESS;
}
```

図 7.1 hello.c



テキストエディタ

Linux での定番のテキストエディタといえば vi や emacs ですが、これらは操作を覚えるだけでも大変です。Debian GNU/Linux では、Windows でのメモ帳のように気軽に使えるテキストエディタとして gedit というアプリケーションが標準でインストールされています。

gedit は「アプリケーション」 - 「アクセサリ」 - 「テキスト・エディタ」メニューから起動することができます。操作方法は、Windows アプリケーションに似ているので、すぐに覚えることができるでしょう。

入力したソースコードが意図したとおりに動作するか、まずは、ホストとなる作業用 PC 上で実行して確認します。

ソースコードをコンパイルするには、端末を起動して、以下のコマンドを実行してください。

```
[ATDE ~]$ gcc hello.c -o hello
```

図 7.2 hello.c をコンパイルするコマンド

Linux では、C コンパイラとして gcc(GNU C Compiler)を使用します。gcc の引数にソースコードのファイル名を与えて実行すると、コンパイル、アセンブル、リンクの一連の処理を自動で行い、実行ファイルを出力します。-o オプションに続いて指定した引数で、実行ファイルの名前を指定することができます^[1]。

なお、コンパイル、アセンブル、リンクの一連の処理を行い、実行ファイルを生成することを、「ビルドする」と表現します。

実行ファイルは、カレントディレクトリに hello というファイル名で作成されます。カレントディレクトリにあるファイルを実行するには、「./」を付けて相対パスでファイル名を指定します。

^[1]実行ファイル名を指定しない場合、実行ファイル名は a.out となります。

```
[ATDE ~]$ ./hello
Hello World!
```

図 7.3 hello の実行結果

エラーやワーニングなくコンパイルでき、意図したとおりに実行結果が表示されたでしょうか？何か問題があれば、ソースコードを修正して、問題がなくなるまでコンパイル、実行を繰り返してください。

ホスト上で問題なく実行できたら、ターゲットとなる Armadillo 用にクロスコンパイルします。

```
[ATDE ~]$ arm-linux-gnueabi-gcc hello.c -o hello
```

図 7.4 hello.c をクロスコンパイルするコマンド

Armadillo(ARM)用にコンパイルするときは、arm-linux-gnueabi-gcc という名前のクロスコンパイラを使用します。

クロスコンパイラでコンパイルしたものは、ARM 用のバイナリとなっているため、もちろんホストでは実行できません。

```
[ATDE ~]$ ./hello
bash: hello: cannot execute binary file
```

図 7.5 クロスコンパイルした hello の実行結果(ATDE 上)



ファイル形式の簡単な見分け方

file コマンドを使用すると、作成された実行ファイルが i386(x86)用なのか、ARM 用なのかを簡単に見分ける事ができます。

i386 用の実行ファイルを file コマンドで調べると、以下のように「Intel 80386」と表示されます。

```
[ATDE ~]$ file hello
hello: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.8, not stripped
```

ARM 用のバイナリでは、「ARM」と表示されます。

```
[ATDE ~]$ file hello
hello: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.14, not stripped
```

ターゲット上で実行するために、Armadillo に実行ファイルを転送します。

Armadillo の標準イメージでは、FTP サーバーが自動で起動しており、pub ディレクトリに書き込みが可能になっていますので、転送には FTP を使用することにします。

ATDE 側の FTP クライアントとして、lftp を使用します。lftp は ATDE3 では標準でインストールされていませんので、以下のコマンドを実行して、インストールしてください。

```
[ATDE ~]$ sudo apt-get install lftp
```

図 7.6 lftp をインストールするコマンド

Armadillo に FTP でファイルを転送するために、以下のコマンドを実行してください。

```
[ATDE ~]$ lftp Armadillo の IP アドレス -e "cd pub;rm hello;put hello;quit"
```

図 7.7 Armadillo へのファイル転送



同じコマンドを入力する手間を省く

一度入力したことがあるコマンドを繰り返し入力するのは、大変面倒です。

シェルには、一度入力したコマンドを記憶しておくヒストリー機能が備わっています。

↑キーで、それまでに入力したコマンドを表示します。

また、Ctrl+r でそれまでに入力したコマンドを遡って検索できます。

例えば、Ctrl+r に続いて、lf と入力すると、「lf」を含む以前入力したコマンドを検索して表示します。表示されたコマンドを実行するには、そのまま Enter キーを入力してください。

```
[ATDE ~]$  
(reverse-i-search)`lf': lftp 172.16.25.11 -e "cd pub;rm hello;put  
hello;quit"
```

転送した実行ファイルを Armadillo で実行してみます。FTP の pub ディレクトリに転送したファイルは、Armaddillo の /home/ftp/pub ディレクトリに保存されます。

```
[armadillo ~]# /home/ftp/pub/hello  
-ash: /home/ftp/pub/hello: Permission denied
```

図 7.8 クロスコンパイルした hello の実行結果(Armadillo 上、実行権限なし)

「Permission denied」というエラーが表示されました。これは、転送した実行ファイルの実行権限がないことを意味しています。Linux システムでは、ファイル一つ一つに、どのユーザーに対して読み、書き、実行する権限を与えるか、指定することができます。

ファイルの権限を変更するには `chmod` コマンドを使用します。`+x` オプションを付けて `chmod` コマンドを実行すると、ファイルに実行権限を付けることができます。

```
[armadillo ~]# chmod +x /home/ftp/pub/hello
[armadillo ~]# /home/ftp/pub/hello
Hello World!
```

図 7.9 クロスコンパイルした `hello` の実行結果(Armadillo 上、実行権限あり)

今度は、無事に「Hello World!」という実行結果を確認できました。

このように、Armadillo 上で動作させるアプリケーションも、最初はホスト上でビルド、実行を繰り返してあらかたのバグを取り除いてから、ターゲットとなる Armadillo で動作確認するというのが、アプリケーション開発の基本的な流れになります。

7.1.2. ライブラリとヘッダファイル

ATDE を使ってシステムを構築するメリットの一つに、豊富なライブラリが利用可能である点が挙げられます。

本章では、ライブラリを使ったアプリケーションプログラムの作成方法について説明します。

例として、算術演算ライブラリに含まれる `sin` 関数を使用します。`sin` 関数は `double` 型の引数の一つとり、その正弦の値を返す関数です。引数はラジアンで指定します。

```
double sin(double x);
```

図 7.10 `sin` 関数のプロトタイプ



関数の定義を調べる

Linux システムでは、オンラインマニュアルでシステムコールとシステムライブラリに含まれる関数の定義を調べることができます。オンラインマニュアルには、関数定義の他、関数が定義されているヘッダファイル、動作の詳細や戻り値などの情報が記載されています。オンラインマニュアルを調べるには、`man` コマンドを使用します。

`sin` 関数を調べるには、以下のコマンドを実行してください。

```
[ATDE ~]$ man sin
```

`sin` 関数を使用したサンプルプログラムを以下に示します。`math.h` は、`sin` 関数を定義しているヘッダファイルです。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char *argv[])
{
    double x = 0.5;

    printf("sin(%g) = %g\n", x, sin(x));

    return EXIT_SUCCESS;
}
```

図 7.11 sin.c

sin.c を hello.c と同様にコンパイルすると、sin が未定義というエラーになります。

```
[ATDE ~]$ gcc sin.c -o sin
/tmp/cc4q8K29.o: In function `main':
sin.c:(.text+0x23): undefined reference to `sin'
collect2: ld returned 1 exit status
```

図 7.12 sin.c をコンパイルするコマンド

Linux システムでは、ライブラリは lib ライブラリ名という名前になっています。算術演算ライブラリの場合、libm です。ビルド時にライブラリをリンクするには、-l ライブラリ名オプションを指定します。

```
[ATDE ~]$ gcc sin.c -lm -o sin
```

図 7.13 sin.c をコンパイルするコマンド(-lm オプション付き)

実行結果は以下のようになります。

```
[ATDE ~]$ ./sin
sin(0.5) = 0.479426
```

図 7.14 sin の実行結果

Armadillo 用にクロスコンパイルするには、hello.c の例と同様にコンパイラにクロスコンパイラを用いるだけです。

```
[ATDE ~]$ arm-linux-gnueabi-gcc sin.c -lm -o sin
```

図 7.15 sin.c をクロスコンパイルするコマンド

実行ファイルを Armadillo に FTP で転送し、実行結果を確認してください。

```
[armadillo ~]$ chmod +x /home/ftp/pub/sin
[armadillo ~]$ /home/ftp/pub/sin
sin(0.5) = 0.479426
```

図 7.16 sin の実行結果(Armadillo 上)

ATDE では、gcc を用いてコンパイルを行った場合、インクルードパスは /usr/include となります。「#include <ヘッダファイル名>」というディレクティブでヘッダファイルをインクルードした場合、インクルードパスにあるヘッダファイルを使用します。

クロスコンパイル用に、コンパイラとして arm-linux-gnueabi-gcc を用いた場合のインクルードパスは、/usr/arm-linux-gnueabi/include となります。gcc を用いた場合とは、参照するヘッダファイルが異なる事に注意してください。

また、ホスト用のライブラリは、/usr/lib ディレクトリにあります。算術演算ライブラリの場合、/usr/lib/libm.so^[2]です。

ARM 用のライブラリは、/usr/arm-linux-gnueabi/lib ディレクトリにあります。ライブラリを使用するプログラムをターゲットで動かす場合には、/usr/arm-linux-gnueabi/lib ディレクトリにあるライブラリファイルを実行ファイルと共にターゲットにコピーしなければなりません。

Armadillo の標準イメージでは、算術演算ライブラリは /usr/lib/libm.so.6^[3]にあるため、今回の例ではライブラリをコピーするという手順は省略しています。

7.1.3. make

プログラムをビルドする際、毎回 gcc コマンドを入力するのは手間がかかります。make を使うことで、複雑なビルド手順を自動化することができます。

make は、makefile と呼ばれる設定ファイルにプログラムをビルドするルールを記述しておく、それによって次に行うべき手順を見つけ出し、必要なコマンドをだけを実行してくれるツールです。

makefile には、以下の形式でルールを記述します。

```
ターゲット: 依存ファイル1 依存ファイル2
            コマンド1
            コマンド2
```

図 7.17 makefile のルール

makefile には、複数のルールを記述することができます。1 つのルールは必ず 1 つのターゲットを持ちます。このターゲットが、そのルールで生成されるファイルとなります。ルールには、ターゲットを生成するために必要な依存ファイルと、ターゲットを生成するためのコマンドを記述します。

依存ファイルは「<ターゲット>:」の後にスペース区切りで記述します。また、コマンドは、ターゲットの次の行から行頭のタブ(スペースではなく)に続いて記述します。依存ファイルとコマンドは、0 個以上記述することができます。つまり、依存ファイルやコマンドがない場合もあります。

^[2]/usr/lib/libm.so は /lib/libm.so.6 へのシンボリックです。さらに、/lib/libm.so.6 は /lib/libm-2.7.so へのシンボリックになっており、これがライブラリの実体です。ライブラリのバージョンが変わっても、コンパイルオプションなどを変更しないで済むようにこのような仕組みになっています。

^[3]同様に、/lib/libm-2.7.so へのシンボリックリンク。

また、makefile では変数を使用することができます。「変数名 = 値」という形式で定義し、\$(変数名) で参照します。基本的に、変数の値は文字列として扱われます。

sin.c をビルドする makefile は以下のようになります。sin.c と同じディレクトリに、Makefile(M は大文字です)という名前で保存してください。

```
CC = gcc
CFLAGS = -Wall -Wextra -O2
LDFLAGS = -lm

TARGET = sin

all: $(TARGET)

sin: sin.o
    $(CC) $(LDFLAGS) $^ $(LDLIBS) -o $@

clean:
    $(RM) *~ *.o $(TARGET)

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

図 7.18 sin.c をビルドする Makefile

make コマンドを引数を指定せずに実行した場合、make はカレントディレクトリにある Makefile という名前のファイルを makefile として認識し、一番最初に記述されたルールに従って処理を行います。

「図 7.18. sin.c をビルドする Makefile」の場合、一番最初のルールは「all: \$(TARGET)」です。これは、変数を展開すると「all: sin」となり、「all ターゲットを生成するには sin ファイルが必要である」というルールになります。

all を作成するには sin が必要ですので、make は「sin: sin.o」というルールに従って sin を生成しようと試みます。このように、make はルールに従って、最初のターゲットに必要なファイルを芋づる式に生成します。

「sin を生成するには、sin.o が必要」と記述してあります。sin.o には「%.o: %.c」というルールが適用されます。これは、特殊なルールの書き方ですが、「.o で終わるターゲットを生成するには、.c で終わるファイルが必要」という意味になります。.o と.c の前は、同じ文字列です。即ち、「sin.o を生成するには、sin.c が必要」ということになります。

sin.c は、既にあるファイルなので、ここでようやくコマンドが実行されます。%.o に対応するコマンドは、「\$(CC) \$(CFLAGS) -c -o \$@ \$<」です。CC や CFLAGS は、Makefile の最初で定義されている変数です。\$@や\$<は特殊な変数で、それぞれターゲット名と依存ファイル名を意味します。そのため、このコマンドを展開すると、「gcc -Wall -Wextra -O2 -c -o sin.o sin.c」となります。

gcc に見慣れないオプションが付いていますね。-Wall と-Wextra は、警告オプションです。ソースコードにバグを誘発しそうな構文があれば、コンパイル時に警告メッセージを表示してくれます。gcc でコンパイルを行う場合は、必ず警告オプションを付けておき、警告メッセージが出ないようなソースコードを記述することを習慣付けておくことで、C 言語の構文が原因のバグを未然に防ぐことができます。

-O2 は、最適化オプションです。gcc では、いくつかの最適化レベルを指定することができます。-O2 を指定した場合、コードサイズと実行速度をどちらも犠牲にしないような最適化を行います。

-c オプションが付いている場合、gcc はコンパイルとアセンブルまでしか行わず、リンク処理を行いません。この時、出力ファイルはアセンブラが出力したオブジェクトファイルになります。

sin.c から sin.o が生成されると、次は sin ターゲットに対応した「\$(CC) \$(LDFLAGS) \$^ \$(LDLIBS) -o \$@」コマンドが実行されます。\$^も特殊な変数で、依存ファイルを意味します^[4]。これを展開すると、「gcc -lm sin.o -o sin」となります。libm と sin.o をリンクして、実行ファイル sin を生成します。

sin が生成されると、all ターゲットに対するルールが適用されます。しかし、all ターゲットに対応するコマンドはないので、何も行われません。当然、all という名前のファイルも生成されません。そのため、all ターゲットに対する処理は make コマンドを実行するたび、毎回行われることになります。

実際の実行結果は、以下のようになります。

```
[ATDE ~]$ ls
Makefile sin.c
[ATDE ~]$ make
gcc -Wall -Wextra -O2 -c -o sin.o sin.c
sin.c: In function 'main':
sin.c:5: warning: unused parameter 'argc'
sin.c:5: warning: unused parameter 'argv'
gcc -lm sin.o -o sin
[ATDE ~]$ ls
Makefile sin sin.c sin.o
```

図 7.19 make コマンドの実行結果

make コマンドを実行すると、まず、sin.c のコンパイルが行われます。このとき、警告オプションの影響で、使用していない変数(argc と argv)があるという警告が表示されています。警告だけでエラーは出ていないので、オブジェクトファイル sin.o が生成され、それを元に実行ファイル sin が生成されています。

ここで、再度 make コマンドを実行しても何も行われません。make は、ターゲットと依存ファイルが変更された時刻を比較して、ターゲットの変更時刻が依存ファイルの変更時刻よりも新しい場合、ターゲットを再生成する必要はないと判断して、ターゲットに対応するコマンドを実行しません。ターゲットがないか、ターゲットよりも依存ファイルが新しい場合のみターゲットの生成をおこないます。

```
[ATDE ~]$ make
make: `all' に対して行うべき事はありません。
```

図 7.20 make コマンドの再実行結果

make コマンドには、引数にターゲット名を指定することもできます。「図 7.18. sin.c をビルドする Makefile」では、clean ターゲットを引数として渡すと、生成したファイルを削除します。

^[4]依存ファイルが複数指定された場合、\$^はそれらすべてを意味するのに対して、\$<は最初の一つだけを意味します。

```
[ATDE ~]$ ls
Makefile sin sin.c sin.o
[ATDE ~]$ make clean
rm -f *~ *.o sin
[ATDE ~]$ ls
Makefile sin.c
```

図 7.21 make clean コマンドの実行結果

最後に、makefile をクロスコンパイルに対応させる方法を紹介します。ホスト用にビルドするか、クロス用にするかは、コンパイラに gcc を使うか、arm-linux-gnueabi-gcc を使うかの違いだけで対応できることを利用します。

```
CROSS := arm-linux-gnueabi

ifneq ($(CROSS),)
CROSS_PREFIX := $(CROSS)-
endif

CC = $(CROSS_PREFIX)gcc
CFLAGS = -Wall -Wextra -O2
LDFLAGS = -lm

TARGET = sin

all: $(TARGET)

sin: sin.o
    $(CC) $(LDFLAGS) $^ $(LDLIBS) -o $@

clean:
    $(RM) *~ *.o $(TARGET)

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

図 7.22 sin.c をビルドする Makefile(クロスコンパイル対応版)

Makefile をこのように修正すると、引数なしで make コマンドを実行するとクロスコンパイルをおこない、「make CROSS=」として実行するとホスト用にコンパイルします。

Makefile の先頭で、CROSS 変数に arm-linux-gnueabi を代入して定義しています。「ifneq (\$(CROSS),)」は、CROSS 変数が空でなければ次の処理を実行することを意味します。CROSS 変数には値が代入されているので、次の処理が実行され、CROSS_PREFIX 変数が定義されます。CC 変数には、gcc の前に CROSS_PREFIX 変数の値をつけた文字列を代入します。そのため、CC の値は arm-linux-gnueabi-gcc となり、クロスコンパイルが行われます。

make コマンドは、引数で変数を定義でき、そのようにして定義した変数は makefile 中で定義する変数よりも優先される機能があります。そのため、「make CROSS=」というように、CROSS 変数を空文字列で定義すると、CROSS_PREFIX も定義されません。そのため、CC の値は gcc となりホスト用のコンパイルが行われます。

このように、同じソースコード、同じ makefile を使用して、クロスコンパイルとホスト用コンパイルの両方に対応することができます。

7.2. Atmark Dist を使ったルートファイルシステムの作成

Atmark Dist は、アットマークテクノ独自のソースコードベースの開発ディストリビューションです^[5]。Atmark Dist を使うと、ユーザーランドのルートファイルシステムとカーネルのイメージファイルを簡単に作成することができます。

Atmark Dist には、様々なアプリケーションプログラムとライブラリのソースコードが含まれています。一方で、Linux カーネルは対象となる製品毎に適切なソースコードが異なるために、Atmark Dist には含まれていません。製品毎に適切なカーネルソースコードを追加して使用します。

Atmark Dist では、対象となる製品毎にどのような機能を含めるかの設定をおこなうことができます。製品のことを「プロダクト」、設定のことを「コンフィギュレーション」と呼びます。Atmark Dist には、Armadillo の開発セット用のプロダクトが含まれています。この開発セット用のプロダクトを元にして、ユーザー独自のプロダクトを追加することもできます。

本章では、Atmark Dist にユーザー独自のプロダクトを追加し、ルートファイルシステムとカーネルのイメージファイルを作成し、イメージファイルをターゲットとなる Armadillo のフラッシュメモリに書き込む方法について説明します。

7.2.1. 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 と Linux カーネルのソースコードは、以下の URL からダウンロードすることができます。

表 7.1 Atmark Dist と Linux カーネルソースコードのダウンロード URL

対象	URL
Atmark Dist	http://download.atmark-techno.com/dist/
Linux カーネル	http://download.atmark-techno.com/kernel-source/linux-2.6.26-at/

基本的には、最新バージョンのソースコードを使用するようにしてください。本章の例では、以下のバージョンを使用します。

表 7.2 使用する Atmark Dist と Linux カーネルソースコードのバージョン

対象	バージョン	ダウンロード URL
Atmark Dist	atmark-dist-20100603	http://download.atmark-techno.com/dist/atmark-dist-20100603.tar.gz
Linux カーネル	linux-2.6.26-at10	http://download.atmark-techno.com/kernel-source/linux-2.6.26-at/linux-2.6.26-at10.tar.gz

ファイルの取得には wget コマンドを使います。

^[5]uClinux-dist をベースに作成しています。

```
[ATDE ~]$ wget http://download.atmark-techno.com/dist/atmark-dist-20100603.tar.gz
[ATDE ~]$ wget http://download.atmark-techno.com/kernel-source/linux-2.6.26-at/linux-2.6.26-at10.tar.gz
[ATDE ~]$ ls
atmark-dist-20100603.tar.gz  linux-2.6.26-at10.tar.gz
```



図 7.23 ソースアーカイブの取得

アーカイブの展開には tar コマンドを使います。

```
[ATDE ~]$ tar xzvf atmark-dist-20100603.tar.gz
[ATDE ~]$ tar xzvf linux-2.6.26-at10.tar.gz
[ATDE ~]$ ls
atmark-dist-20100603/  atmark-dist-20100603.tar.gz  linux-2.6.26-at10/  linux-2.6.26-at10.tar.gz
```

図 7.24 ソースアーカイブの展開

以降の操作を分かりやすくするため、atmark-dist-20100603 ディレクトリに atmark-dist という名前でシンボリックリンク^[6]を張ります。シンボリックリンクを作成するには、ln -s コマンドを使います。

```
[ATDE ~]$ ln -s atmark-dist-20100603 atmark-dist
[ATDE ~]$ $ ls -l atmark-dist
lrwxrwxrwx 1 atmark atmark 20 2010-09-27 11:57 atmark-dist -> atmark-dist-20100603
```

図 7.25 atmark-dist ディレクトリのシンボリックリンクの作成

Atmark Dist が使用する Linux カーネルを指定するために、atmark-dist ディレクトリに Linux カーネルソースディレクトリへのシンボリックリンクを張ります。

```
[ATDE ~]$ cd atmark-dist
[ATDE ~/atmark-dist]$ ln -s ../linux-2.6.26-at10 linux-2.6.x
[ATDE ~/atmark-dist]$ ls -l linux-2.6.x
lrwxrwxrwx 1 atmark atmark 20 2010-09-27 11:58 linux-2.6.x -> ../linux-2.6.26-at10
```

図 7.26 atmark-dist ディレクトリのシンボリックリンクの作成



注意: linux-2.6.x はそのまま入力

「linux-2.6.x」は、省略表記ではありません。置き換えなどをせずにそのまま入力してください。

7.2.2. 独自プロダクトの追加

Atmark Dist では、製品固有の設定やファイルは製品(プロダクト)毎のディレクトリにまとめて、管理しています。このディレクトリのことを、プロダクトディレクトリといいます。アットマークテクノ製

^[6]ファイルやディレクトリに対してつける別名のこと

品の場合、開発セット用の標準イメージに対応するプロダクトディレクトリが、製品毎に用意されています。

Armadillo を使って独自の製品を作る場合、ベースとなる Armadillo 用のプロダクトディレクトリをコピーして使用することで、基本的な設定を引き継ぐことができカスタマイズが容易になります。今回は、Armadillo-440 液晶モデル開発セット用のプロダクトディレクトリをコピーして使用することにします。独自プロダクトの名前は、my-product とします。以下のコマンドを入力し、my-product というプロダクトディレクトリを作成してください。

```
[ATDE atmark-dist]$ cp -a vendors/AtmarkTechno/Armadillo-440 vendors/AtmarkTechno/my-product
```

図 7.27 独自プロダクトの追加

7.2.3. 基本的なコンフィギュレーション

Atmark Dist では、ユーザーランドのルートファイルシステムにどのアプリケーションやライブラリを含めるか、カーネルにどの機能を含めるかといった設定をすることができます。この設定のことをコンフィギュレーションと呼びます。

本章では、ビルド対象のプロダクトとして my-product を選択し、標準のコンフィギュレーションを適用する方法について説明します。現在のところ、my-product は Armadillo-440 のプロダクトディレクトリをそのままコピーしただけなので、標準のコンフィギュレーションを適用すると、Armadillo-440 の標準イメージと同じ設定となります。

Atmark Dist のコンフィギュレーションを変更するには、make menuconfig コマンドを使用します。menuconfig によるコンフィギュレーション設定画面では、「表 7.3. menuconfig の操作方法」に示すキー操作で画面の操作を行います。

表 7.3 menuconfig の操作方法

キー操作	動作
↑↓キー	メニューの選択
←→キー	動作の選択
スペースキー	オプションの選択
Enter キー	動作の決定

具体的な手順は、次のようになります。

atmark-dist ディレクトリで、make menuconfig コマンドを実行してください。

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

図 7.28 make menuconfig コマンドの実行

make menuconfig を実行すると、「図 7.29. menuconfig: Main Menu 画面」が表示されます。

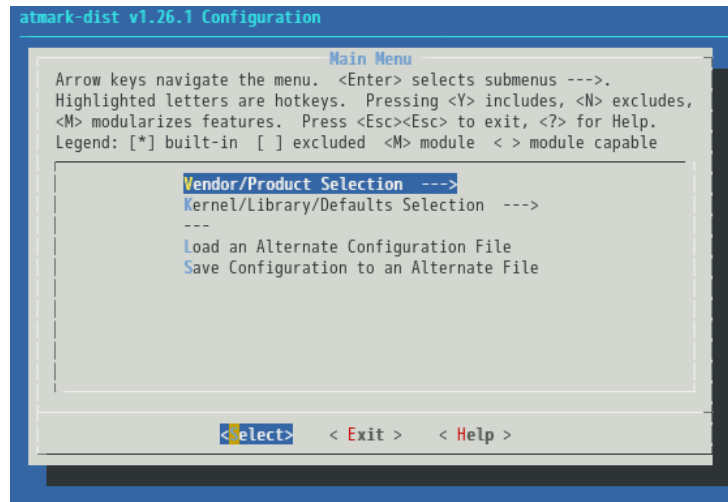


図 7.29 menuconfig: Main Menu 画面

まずは、ベンダーとプロダクトを選択します。キーボードの上下キーでフォーカスを「Vendor/Product Selection -->」に合わせ、Enter キーを押すと、「図 7.30. menuconfig: Vendor/Product Selection 画面」が表示されます。

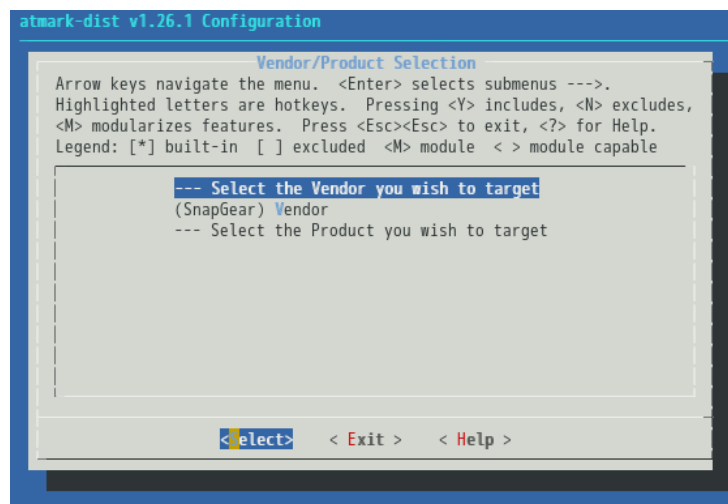


図 7.30 menuconfig: Vendor/Product Selection 画面

デフォルトのベンダーとして「SnapGear」が選択されているので、「AtmarkTechno」に変更します。「(SnapGear) Vendor」を選択すると、「図 7.31. menuconfig: Vendor 画面」が表示されます。

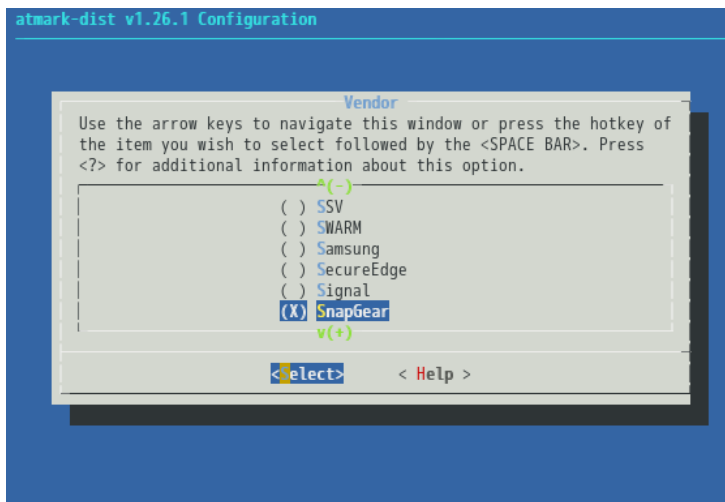


図 7.31 menuconfig: Vendor 画面

上の方にある、「AtmarkTechno」を選択すると、「図 7.32. menuconfig: Vendor/Product Selection 画面」に戻ります。

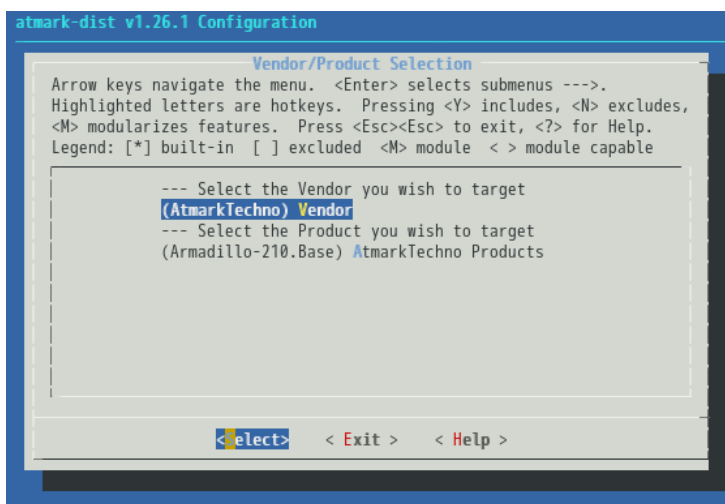


図 7.32 menuconfig: Vendor/Product Selection 画面

プロダクトは、先ほど作成した「my-product」に設定します。「(Armadillo-210.Base) AtmarkTechno Products」を選択すると、「図 7.33. menuconfig: AtmarkTechno Products 画面」が表示されます。

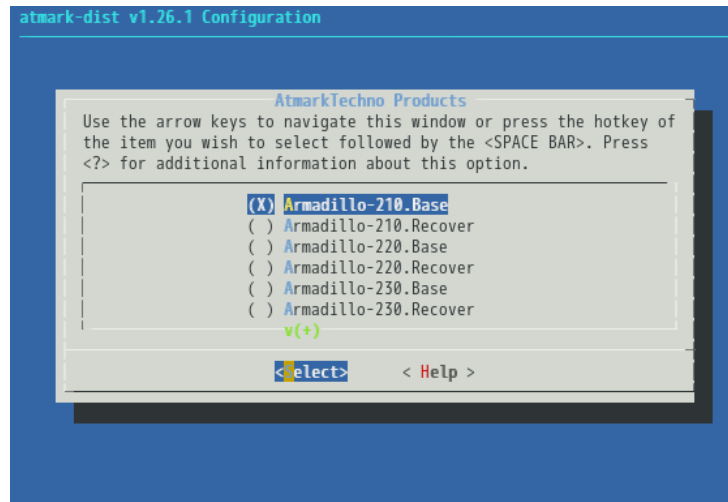


図 7.33 menuconfig: AtmarkTechno Products 画面

下の方にある「my-product」を選択すると、「図 7.32. menuconfig: Vendor/Product Selection 画面」に戻ります。

以上でベンダーとプロダクトの選択は終わりです。コンフィギュレーションの設定は以上で完了したので、menuconfigを終了します。

キーボードの右キーでフォーカスを Exit に合わせ、Enter キーを押してください。「図 7.29. menuconfig: Main Menu 画面」に戻ります。Exit にフォーカスを合わせ、Enter キーを押してください。すると、「図 7.34. menuconfig: Do you wish to save your new kernel configuration?」と表示されますので、Yes にフォーカスを合わせ、Enter キーを押してください。ここで No を選択すると、これまでの設定がすべて失われます。

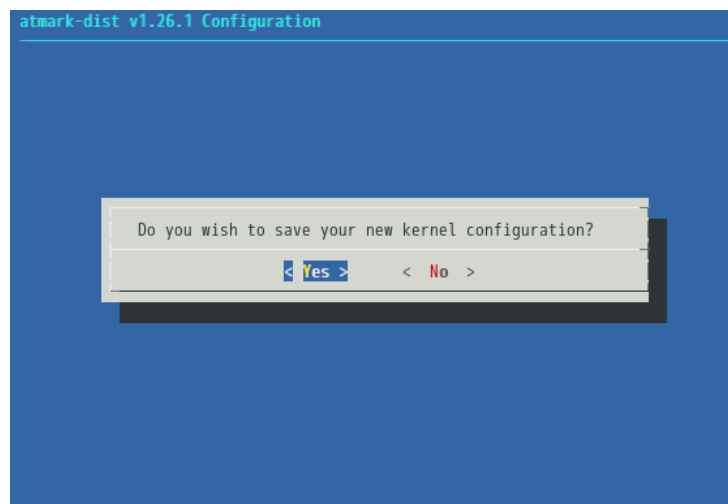


図 7.34 menuconfig: Do you wish to save your new kernel configuration?

コンフィギュレーションを my-product の標準設定にする手順は以上です。

7.2.4. ルートファイルシステムのビルドとイメージファイルの作成

コンフィギュレーションの設定が完了したら、ルートファイルシステムとカーネルをビルドします。Atmark Dist では、ビルドと共に、それらを Armadillo のフラッシュメモリに書き込める形式に変換します。このファイルを、イメージファイル又は、単にイメージといいます。

Atmark Dist でビルドとイメージの作成を開始するには、make コマンドを引数なしで実行します。

```
[ATDE ~/atmark-dist]$ make
```

図 7.35 ルートファイルシステムのビルドとイメージファイルの作成の開始

make が完了すると、atmark-dist/images ディレクトリにイメージファイルができています。

```
[ATDE ~/atmark-dist]$ ls images
linux.bin linux.bin.gz romfs.img romfs.img.gz
```

図 7.36 作成されたイメージファイル

romfs.img がユーザーランドのルートファイルシステムの、linux.bin がカーネルのイメージファイルです。末尾に.gz が付いているファイルは、それぞれを圧縮したものです。通常は、圧縮されたイメージファイルを Armadillo に書き込みます。

7.2.5. イメージファイルの書き込み

前章で作成したイメージファイルを、Armadillo のフラッシュメモリに書き込みます。

Armadillo では、フラッシュメモリをいくつかの領域(リージョン)に分割して使用しています。カーネルイメージはカーネル領域に、ユーザーランドのルートファイルシステムイメージはユーザーランド領域にそれぞれ書き込みます。

ここでは、Hermit-At ブートローダーの tftpd 機能を使用して、ネットワーク経由で書き込む方法を紹介します。Armadillo のフラッシュメモリにイメージファイルを書き込む方法はいくつかありますが、ATDE と Armadillo がネットワーク接続できている環境であれば、tftpd による書き込みが手軽で高速です。その他の書き込み方法については、「Armadillo-400 シリーズ ソフトウェアマニュアル」を参照してください。

tftpd は、TFTP プロトコルを使用して TFTP サーバーからファイルをダウンロードし、フラッシュメモリの書き換えを行う機能です。ATDE3 では標準で TFTP サーバーが動作しており、すぐに使うことができます。以下のコマンドを実行して、イメージファイルを TFTP サーバーが使用するディレクトリにコピーしてください。

```
[ATDE ~/atmark-dist]$ sudo cp images/linux.bin.gz /var/lib/tftpboot
[ATDE ~/atmark-dist]$ sudo cp images/romfs.img.gz /var/lib/tftpboot
```

図 7.37 イメージファイルのコピー

tftpd 機能は、Hermit-At ブートローダーの保守モードで使用することができます。


```
#####  
  
programing: userland  
#####  
  
completed!!
```

図 7.39 tftpdI コマンド

フラッシュメモリへの書き込みが完了したら、Armadillo の電源を入れ直すことで書き込んだイメージで起動します。

7.3. ルートファイルシステムのカスタマイズ

本章では、独自プロダクトのルートファイルシステムにアプリケーションを追加したり、コンフィギュレーションを変更する方法について説明します。

7.3.1. アプリケーションプログラムの追加

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プログラムは、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プログラムを Atmark Dist に統合する方法を紹介します。例として、「7.1. アプリケーションプログラムの作成」で作成した、sin を追加することとします。

7.3.1.1. ディレクトリの追加

まず、プロダクトディレクトリにアプリケーションプログラム用のディレクトリを追加します。追加するディレクトリ名は、sin とします。

```
[ATDE ~/atmark-dist]$ mkdir vendors/AtmarkTechno/my-product/sin
```

図 7.40 アプリケーションプログラム用ディレクトリの作成

7.3.1.2. ソースコードと makefile の追加

ソースコードは、「図 7.11. sin.c」をそのまま使用します。vendors/AtmarkTechno/my-product/sin ディレクトリに sin.c をコピーしてください。

Atmark Dist にアプリケーションプログラムを統合する場合、Makefile は若干異なり、以下のようになります。vendors/AtmarkTechno/my-product/sin ディレクトリに以下の内容で Makefile を追加してください。

```

LDFLAGS = -lm

TARGET = sin

all: $(TARGET)

sin: sin.o
    $(CC) $(LDFLAGS) $^ $(LDLIBS) -o $@

clean:
    $(RM) *~ *.o $(TARGET)

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

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

図 7.41 Atmark Dist で sin.c をビルドする Makefile

CC や CFLAGS は Atmark Dist によって適切に設定されるので、指定を省略しています。また、romfs ターゲットが追加されています。\$(ROMFSINST) コマンドによって、TARGET に指定したファイルが、ユーザーランドの/bin ディレクトリにコピーされます。

7.3.1.3. 追加したディレクトリをビルド対象に含める

アプリケーションプログラム用のディレクトリを、ビルド対象に含めるよう設定します。

プロダクトディレクトリ直下の Makefile で、どのディレクトリをビルド対象とするか決めているので、ここに先ほど追加したアプリケーションプログラム用のディレクトリを追加します。

具体的には、vendors/AtmarkTechno/my-product/Makefile の SUDIR_y 変数に、以下のように sin を代入してください。

```

:
empty :=
space := $(empty) $(empty)

SUBDIR_y += sin
SUBDIR_$(CONFIG_VENDOR_FUNCTESTER_FUNCTESTER) += functester/
SUBDIR_$(CONFIG_VENDOR_GPIOCTRL_GPIOCTRL) += gpioctrl/
SUBDIR_$(CONFIG_VENDOR_LEDCTRL_LEDCTRL) += ledctrl/
:
    
```

図 7.42 追加したディレクトリをビルド対象に含めるための Makefile の修正箇所

7.3.2. ファイルの追加

プロダクトディレクトリ直下の etc、home、usr ディレクトリ以下にファイルを追加すると、ユーザーランドのルートファイルシステムにそれらのファイルがコピーされます。

例えば atmark-dist/vendors/AtmarkTechno/my-product/etc/my-dir ディレクトリを作成し、そのディレクトリに my-file ファイルを追加したとします。その場合、Atmark Dist でビルドしたルート

ファイルシステムでは/etc/my-dir ディレクトリが作成され、そのディレクトリに my-file ファイルが配置されます。設定ファイルやデータファイルが必要な場合はここに追加してください。

7.3.3. コンフィギュレーションの変更

Atmark Dist では、コンフィギュレーションをカスタマイズすることができます。コンフィギュレーションをカスタマイズすることで、新しい機能を追加したり、逆に機能を必要最小限にしスリムなシステムを構築するといったことが可能になります。

コンフィギュレーションは、大きくわけて二つ、ユーザーランドのルートファイルシステムとカーネルに対して行うことができます。コンフィギュレーションを変更したら、再度 Atmark Dist のビルドをおこない、新たに作成されたイメージファイルを Armadillo に書き込むことで、変更を反映することができます。



ユーザーランドなのかカーネルなのか

コンフィギュレーションを変更した際に陥りがちな失敗が「ユーザーランドのルートファイルシステムのコンフィギュレーションを変更したのに、カーネルのイメージを書き込んだ」あるいはその逆です。

アプリケーションプログラムやライブラリのコンフィギュレーションを変更した場合は、ユーザーランドの変更です。その場合、Armadillo に書き込むべきイメージファイルは romfs.img.gz です。

カーネルのコンフィギュレーションを変更した場合は、linux.bin.gz です。

変更が反映されないな、と思ったときは、もう一度よくどちらのコンフィギュレーションを変更しているのか、確認してください。

具体的なコンフィギュレーションの方法は、第2部「詳解 Atmark Dist」の「コンフィギュレーションの設定」で説明します。

7.4. 量産に向けた準備

これまでに、製品独自のアプリケーションプログラムを作成し、それを Atmark Dist に統合してルートファイルシステムとカーネルをビルドし、イメージファイルを作成、そして、Armadillo に書き込むという一連の開発の流れについて説明してきました。十分にテストして、システムとして動作することが確認できれば、製品として出荷する準備は整ったことになります。

本章では、製品の量産化前に行うべきことについて説明します。

7.4.1. カスタマイズサービスを利用する

開発段階では、通常、Armadillo の開発セットを使用します。開発セットには開発に必要な周辺機器一式が含まれているので便利ですが、量産時には不要なものもあります。Armadillo は量産用にボード単体の量産モデルも販売しています。

また、アットマークテクノでは、ユーザー独自のイメージファイルの書き込みや梱包といったカスタマイズを、Armadillo におこなって納品するという、カスタマイズサービスを提供しています。カスタマイズサービスを利用すると、ユーザーはイメージファイルと作業内容を指示するだけで、ユーザー用にカスタマイズされた Armadillo を購入することができます。

カスタマイズサービスでは以下のサービスを提供しています。

表 7.4 カスタマイズサービスの種類

サービス	内容
コネクタ実装指定サービス	標準コネクタの実装/非実装の指定や、指定コネクタの実装など
ROM イメージ書き込みサービス	基板への指定のイメージファイル書き込み
梱包指定サービス	個装箱への封入や、指定シールの貼付などの梱包
ケーシングサービス	ケースへの基板を組み込み
オプション品指定サービス	AC アダプタやケーブルなど、開発用モデル添付の標準オプション品を指定

その他カスタマイズサービスに関する詳しい内容については、弊社営業部までお問い合わせください。

7.4.2. ライセンスに関する注意事項

Armadillo で使用しているソフトウェアは、その大半がオープンソースソフトウェアです。オープンソースソフトウェアの中には、GPL のようにそのソースコードを公開することが義務付けられているものがあります。製品化を行う前に、使用しているソフトウェアがどのようなライセンスを持っているのか、確認する必要があります。

Linux カーネルには、GPL が適用されます。そのため、Linux カーネルを使用したシステムを販売する際には、システムで使用している Linux カーネルのソースコードをエンドユーザーが入手できる手段を提供しなければなりません。

なお、Linux カーネルは GPL ですが、アプリケーションプログラムからシステムコールを介してその機能を使用する分にはアプリケーションプログラムは GPL の影響を受けません。

BusyBox も同様に GPL が適用されるソフトウェアです。

7.5. 製品出荷後のメンテナンス

本章では、製品化後に気をつけるべき点について説明します。

7.5.1. ソフトウェアアップデートへの対応

Armadillo 用のブートローダー、Linux カーネル、Atmark Dist は、機能追加やバグ修正などで随時アップデートがおこなわれます。これらのアップデートが行われた際、ユーザーの製品で使用しているソフトウェアもアップデートする必要があるかもしれません。

第 2 部では、最新のソースコードで製品用のイメージを作成しなおす方法や、イメージを自動アップデートする仕組みの構築方法について説明します。

7.5.2. ハードウェアの変更通知

ソフトウェアだけでなく、ハードウェアの変更がある場合もあります。ハードウェアの変更には、エラッタの修正や使用部品の変更、基板の改版などが含まれます。

ユーザー登録を行っておくと、このような変更が行われた際に送られる通知を受け取ることができます。

改訂履歴

バージョン	年月日	改訂内容
1.0.0	2010/10/19	初版リリース

Armadillo 実践開発ガイド
Version 1.0.0
2010/10/19

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 TEL 011-207-6550 FAX 011-207-6570
