Armadillo 標準ガイド Armadillo 入門編

～組み込み Linux の導入から製品化まで～

Version 1.0.1
2019/04/25

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

Armadillo サイト [http://armadillo.atmark-techno.com]

Armadillo 標準ガイド Armadillo 入門編: 〜組み込み Linux の導入から製品化まで〜

株式会社アットマークテクノ

Version 1.0.1
2019/04/25

目次

Armadillo 標準ガイド Armadillo 入門編

図目次

Armadillo 標準ガイド Armadillo 入門編

Armadillo 標準ガイド Armadillo 入門編

表目次

1. はじめに

Linux が動作する ARM CPU 搭載の汎用ボードコンピューターというコンセプトでデザインした初代 Armadillo (HT1070)を発売したのは、2002 年 3 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、どちらかというと否定的な意見も多く聞かれました[1]。

初代 Armadillo の発売から 20 年近い歳月が過ぎ、Linux+ARM という組み合わせは今では当たり前の選択となりましたが、Linux で組み込みシステムの開発を行うにあたっては、Linux そのものの使い方、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。

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-600 シリーズ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解説しています。初めから順番に読んでいきながら、システムを開発する際に行うべき手順を把握することができます。また、Armadillo-600 シリーズ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的に活用することもできると思います。

過去にいただいたお問い合わせや、メーリングリストでのやりとりなどを踏まえて、多くのお客様にとってつまづきやすい点、知りたいとリクエストいただいた内容をなるべく多く取り入れたつもりです。つたない記述もあるかと思いますが、本書を皆様の開発に役立てていただければ幸いです。

1.1. 対象読者

本書が主な対象読者としているのは、Armadillo-600 シリーズ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。少なくとも C 言語での開発経験があることを前提としています。Linux システムに関する内容が含まれていますが、Linux の使用経験はなくとも読み進められるように配慮してあります。

また、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考えている設計者、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ーですので、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。Armadillo-600 シリーズで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は製品マニュアルをご一読いただき、実際に使いこなす際に本書をお使いいただければと思います。

1.2. 本書の構成

本書は、大きく分けて 2 部構成になっています。

本書「Armadillo 入門編」では、Armadillo-600 シリーズ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でも理解できるよう、基本的な用語や操作方法から説明します。「Armadillo 入門編」を読み終えると、開発を始めるための基本的な知識が習得でき、また開発の全体的な流れが把握できるようになります。

「ハードウェア拡張編」では、様々な外部機器を Armadillo に接続して使う方法を紹介します。1 つの機器ごとに Howto 形式で書かれており、ここまでを習得された方であれば容易に読み解ける内容になっています。

[1]特集:最新組み込み Linux 実践講座 Part1https://armadillo.atmark-techno.com/articles/sd-a500-embedded-course-ch1

1.3. 表記について

1.3.1. コマンド入力例

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

表 1.1 表示プロンプトと実行環境の関係

プロンプト	コマンドの実行環境
[PC /]#	作業用 PC(Linux)の root ユーザで実行
[PC /]$	作業用 PC(Linux)の一般ユーザで実行
[ATDE /]#	ATDE 上の root ユーザで実行
[ATDE /]$	ATDE 上の一般ユーザで実行
[armadillo /]#	Armadillo 上 Linux の root ユーザで実行
[armadillo /]$	Armadillo 上 Linux の一般ユーザで実行
⇒	Armadillo 上 U-Boot の保守モードで実行

コマンド中で、変更の可能性のあるものや、環境により異なるものに関しては以下のように表記します。適宜読み替えて入力してください。

表 1.2 コマンド入力例での省略表記

表記	説明
[version]	ファイルのバージョン番号

1.3.2. アイコン

本書では以下のようにアイコンを使用しています。

注意事項を記載します。

役に立つ情報を記載します。

用語の説明や補足的な説明を記載します。

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、 https://download.atmark-techno.com/armadillo-guide-std/sample/ 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

本書を読んでわからなかったり困ったことがあった際は、ぜひ Armadillo サイト[3]で情報を探してみてください。本書には記載しきれていない FAQ や Howto が掲載されています。

Armadillo サイトでも知りたい情報が見つからない場合は、「Armadillo フォーラム」[4]で質問してみてください。Armadillo フォーラムは、アットマークテクノユーザーズサイト内に設けられた、Armadillo ブランド製品での開発や周辺技術に関する話題を扱うユーザー向けコミュニティです。Armadillo に関する技術的な話題なら何でも投稿できます。多くのユーザーや開発者が参加しているので、知識のある人や同じ問題で困ったことがある人から情報を集めることができます。

フォーラムに参加するときの心構え

Armadillo フォーラムには、その前身となったメーリングリストから引き続き、数百人のユーザーが参加しています。また、フォーラムへ投稿した内容は Web 上で誰でも閲覧・検索可能になるほか、通知を希望しているユーザーにメールで送信されます。

フォーラムには多くの人が参加しており、投稿内容は多くの人の目に触れますので、そこにはマナーが存在します。一般的な対人関係と同様に、受け取り手に対して失礼にならないよう一定の配慮はすべきです。技術系コミュニティに不慣れな方は、投稿する前に「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] をご一読されることをお勧めします。メーリングリストに投稿するときの心構えや、適切な回答を得るために有用なテクニックが分かりやすく紹介されています。メーリングリストとフォーラムの違いはあれど、基本的な考え方は共通しており、とても参考になります。

とはいえ、技術的に簡単なものであるとか、ちょっとした疑問だからという理由で、投稿をためらう必要はありません。Armadillo に関係のある内容であれば、難しく考えることなく気軽にお使いください。

1.6. お問い合わせ先

本書に関するご意見やご質問は、Armadillo フォーラム[4]にご連絡ください。

[2]http://opensource.org/licenses/mit-license.php
[3]https://armadillo.atmark-techno.com
[4]https://users.atmark-techno.com/forum/armadillo
[5]結城浩氏によるサイトより http://www.hyuki.com/writing/techask.html

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。™、®マークは省略しています。

1.8. ライセンス

図 1.1 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の多くは Free Software / Open Source Software で構成されています。Free Software / Open Source Software は世界中の多くの開発者の成果によってなりたっています。この場を借りて感謝の意を表します。

2. 注意事項

注意: 本書の内容を実践する前に

ご使用になる製品のマニュアル(ハードウェアマニュアル、ソフトウェアマニュアル、その他関連資料)をよく読み、それらに記述されている注意事項に従って正しく安全にお使いください。

3. 組み込み Linux システムとは

3.1. Linux システムとは

Linux 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。当初はインテル x86 アーキテクチャの PC 向けの、わずか 1 万行程度のソースコードによって記述された小さな OS でした。しかしその後 Linux は驚異的な進化を遂げ、今日では x86 以外にも ARM、PowerPC、MIPS、SuperH など様々なアーキテクチャのコンピューターで動作するようになり、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ーまで幅広い用途で使用されています。

厳密にいうと、Linux とは CPU、メモリ、タイマーなどのリソース管理、プロセス管理、デバイス制御などをおこなう OS の中心となる部分(これをカーネルと呼びます)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(これらをユーザーランドと呼びます)が別途必要です。こうしたことから本書では、カーネル部分だけを指す場合は Linux カーネル、ユーザーランドも含めたシステム全体を指す場合は Linux システムと呼びます。

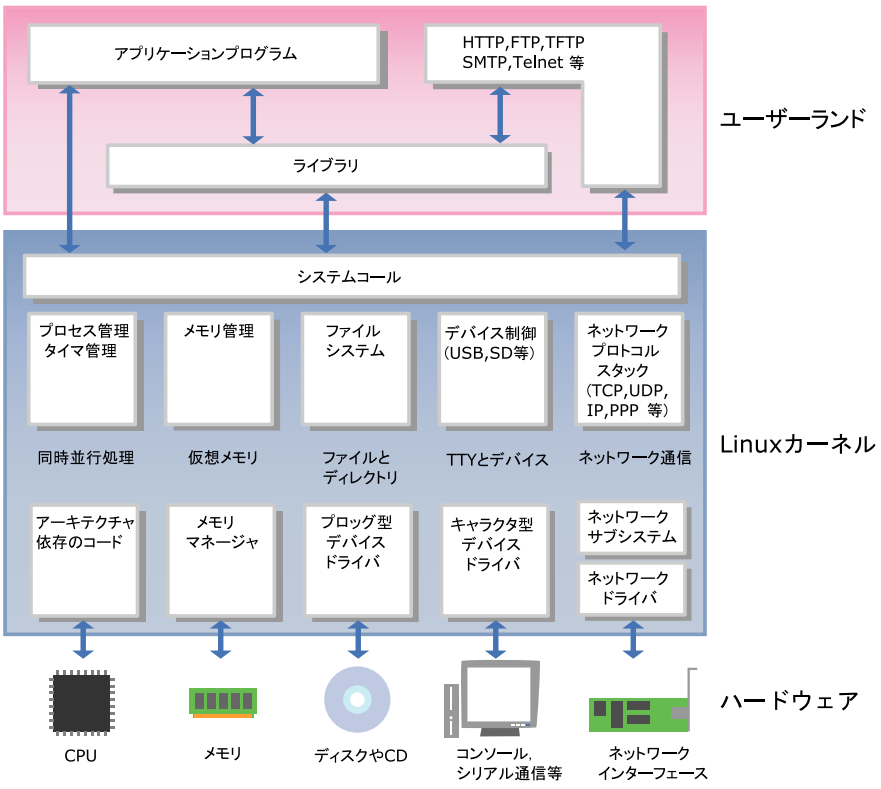

図 3.1 Linux システムアーキテクチャ

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 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長を持っており、多くの分野で使用されてきました。そして、広く普及した分だけ様々な派生バージョンが発生することになります。その結果として標準化作業が必要

となり、POSIX(Portable Operating System Interface)が制定されました。こうした背景から、UNIX ライクな OS とは「POSIX に沿うよう開発されている OS」を指します[1]。

オープンソースとは、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義務があります。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)[2]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[3]。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での 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 プロジェクトについては、The GNU Operating System[4]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[5]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[6]、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。2019 年 2 月現在の安定版である Debian GNU/Linux 9(コードネーム stretch)では、2017 年の初版リリース時点で 51,000 以上のパッケージが用意されています。

- Ubuntu Linux [http://www.ubuntulinux.org]

[1]Linux カーネルは POSIX に沿うように開発されていますが、完全準拠ではないため UNIX の一種であるとはいえません。
[2]http://www.gnu.org/licenses/licenses.ja.html#TOCGPL
[3]必ずしも、GNU ソフトウェアを使わなければならないという意味ではありません。例えば、近年携帯電話等に採用が進む Android の場合、カーネルは Linux ですが、ユーザーランドは Apache ライセンスを用いたソフトウェアで構成します。
[4]http://www.gnu.org/
[5]フリー/オープンソースソフトウェアのようにソースコードが公開されていて、誰でも利用、改変、再配布できるソフトウェアに対して、それらに制限のあるソフトウェアをプロプライエタリ・ソフトウェアと呼びます。
[6]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管理システムを使用します。

- Red Hat Enterprise Linux [http://www.redhat.com]

  Red Hat 社が提供するエンタープライズサーバー向け商用ディストリビューション[7]です。Fedora の成果を活用して開発されています。

- openSUSE [http://www.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行います。

- SUSE Linux Enterprise [http://www.novell.com/linux/]

  Novell 社が提供するエンタープライズサーバー向け商用ディストリビューションです。openSUSE を基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。

3.2. 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ありません。しかしながら、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、SuperH 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。

組み込み Linux ではボード固有の部分には修正が必要になるというものの、ユーザーランドで動作するアプリケーション、及び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 システムとの違いはありません。そのため、普通の Linux システム上で動作するソフトウェアは、一部の例外を除いて組み込み Linux システム上でも動作します[8]。

PC やサーバーと比較すると、組み込みシステムでは CPU クロックが低い場合や、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[9]、また、PC やサーバーに比べると過酷な環境で使用される場合があります。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。

[7]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。
[8]Linux の場合多くのソフトウェアがソースコードで提供されるためこのようにいえますが、バイナリしか提供されないものについては CPU アーキテクチャなどが一致している必要があります。
[9]これは相対的な表現であり、マイコンボードに比べれば豊富なリソースを持っているともいえます。

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するソースコードも機器に添付しなければなりません[10]。もちろん、ライセンスによってはソースコードの公開が義務付けられていないものもありますので、そのようなライセンスを持つソフトウェアやその派生物については非公開としても問題ありません。

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。ただし、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[11]。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 があったように、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。

[10]厳密には、機器を使用するすべてのユーザーに対してソースコードの入手方法を提供することが条件です。
[11]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。

4. Armadillo を使った組み込みシステム開発

4.1. 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(Armadillo)はスペイン語の「armado (英:armed)」に縮小辞[1]「illo」を付けた「武装した小さなもの」が語源とされていますが、ボードコンピューターの Armadillo は「ARM CPU 搭載の小さなもの」との意味になっており、開発コードネーム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 4.1. 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[2]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。

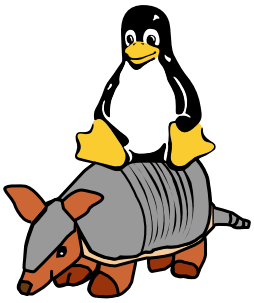

図 4.1 Armadillo ロゴ

ARM と SoC

Arm プロセッサは、英 Arm 社が開発する 32bit/64bit RISC アーキテクチャです。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長を持ちます。

歴代の iPhone/iPad、Android 等のほぼすべてのスマートフォンが Arm を採用しているほか、Nintendo Switch、PlayStation Vita などの携帯ゲーム機等、モバイル機器を中心に様々なところで採用されています。

[1]「小さい」「少し」といった意味を表す接辞のこと。http://ja.wikipedia.org/wiki/縮小辞
[2]http://ja.wikipedia.org/wiki/タックス

Arm は、Arm プロセッサのアーキテクチャを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 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ーとして必要な周辺機能を追加し、SoC(System on Chip)等としてパッケージ化して製造、販売します。

自社製品向けに SoC を使用する Apple、Samsung、任天堂などのほか、代表的なスマートフォン向け SoC の Snapdragon を設計する Qualcomm、FPGA と組み合わせた製品にしている Intel や Xilinx、Raspberry Pi 等に採用されている BCM シリーズの Broadcom、その他、様々な SoC・汎用プロセッサを製造する NXP Semiconductors、Texas Instruments、Silicon Labs、Microchip、STMicroelectronics、Nordic、ルネサス、東芝など多数の半導体メーカーが Arm からライセンスを受け、設計・製造しています。この中には独自 CPU を設計していたメーカーも多いですが、高性能製品を中心に Arm へ移行しています。

初代 Armadillo(HT1070)が 2002 年 3 月に発売されてから、毎年のように新製品を発表してきました。「表 4.1. 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 4.1 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/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[a]/ 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	2007 年 5 月	NXP[b]/i.MX31[c]	ARM1136JF-S/533MHz[d]
Armadillo-500 FX	2008 年 9 月	NXP[b]/i.MX31	ARM1136JF-S/533MHz[d]
Armadillo-440	2010 年 2 月	NXP[b]/i.MX257	ARM926EJ-S/400MHz
Armadillo-420	2010 年 5 月	NXP[b]/i.MX257	ARM926EJ-S/400MHz
Armadillo-460	2011 年 5 月	NXP[b]/i.MX257	ARM926EJ-S/400MHz
Armadillo-800 EVA	2011 年 11 月	ルネサス/R-Mobile A1	Cortex-A9/800MHz
Armadillo-810	2012 年 11 月	ルネサス/R-Mobile A1	Cortex-A9/800MHz
Armadillo-840	2012 年 11 月	ルネサス/R-Mobile A1	Cortex-A9/800MHz
Armadillo-410	2013 年 10 月	NXP[b]/i.MX257	ARM926EJ-S/400MHz
Armadillo-EVA 1500	2014 年 7 月	ルネサス/RZG1M	Cortex-A15/1.5GHz
Armadillo-IoT	2014 年 12 月	NXP[b]/i.MX257	ARM926EJ-S/400MHz
Armadillo-IoT G2	2015 年 6 月	NXP[b]/i.MX257	ARM926EJ-S/400MHz
Armadillo-Box WS1	2015 年 7 月	NXP[b]/i.MX257	ARM926EJ-S/400MHz
Armadillo-IoT G3	2016 年 2 月	NXP/i.MX7Dual	Cortex-A7/1GHz
Armadillo-X1	2016 年 5 月	NXP/i.MX7Dual	Cortex-A7/1GHz
Armadillo-IoT G3L	2016 年 11 月	NXP/i.MX7Dual	Cortex-A7/1GHz
Armadillo-840m	2017 年 4 月	ルネサス/R-Mobile A1	Cortex-A9/800MHz
Armadillo-640	2017 年 11 月	NXP/i.MX6ULL	Cortex-A7/528MHz

[a]発売当時は NetSilicon 社

[b]発売当時は Freescale 社

[c]初期モデルは i.MX31L

[d]初期モデルは 400MHz 動作

現在の Armadillo シリーズは、大きく 2 つの流れに分類できます。従来からある組込み用途向けの汎用製品群と、IoT 向けにセンサやクラウドとつなぐことを意識した製品群です。本書では前者の製品群を中心に記載します。

前者をさらに分類すると、

1. 外部拡張バスを持つ汎用製品群
2. 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
3. 無線 LAN や Wi-SUN のような無線機能を有した製品群
4. カメラやタッチパネル LCD、HDMI への画面出力に対応した製品群

となります。

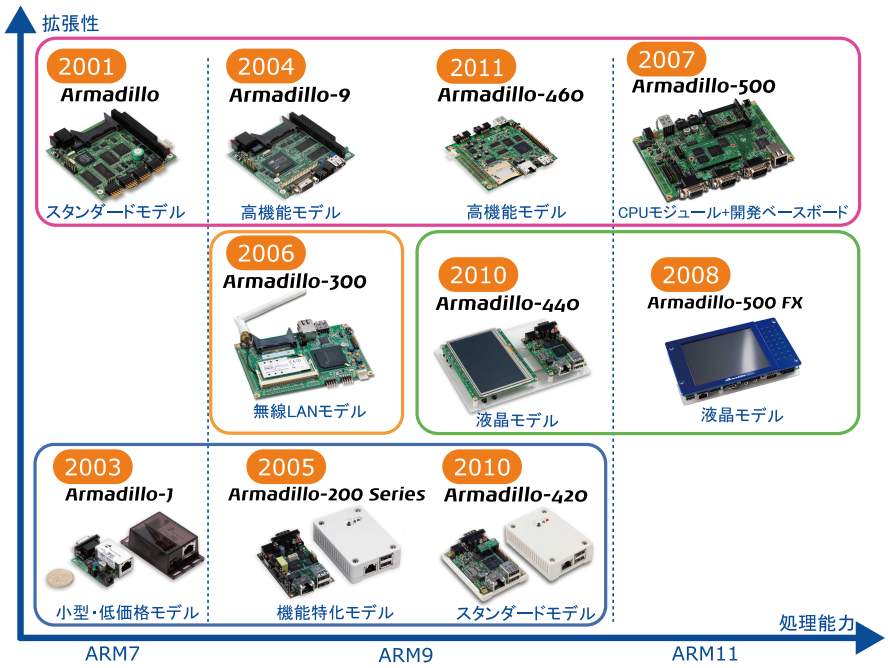

図 4.2 Armadillo シリーズ

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 拡張バスを持った、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 です。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、より多くの用途に対応できるようになりました。

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

[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製品ラインナップとしての名称も持っています。

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いえるでしょう。

Armadillo-460 は、Armadillo-9 の後継モデルです。後述する Armadillo-440 の基本機能に加え、PC/104 バスを搭載した拡張性の高い製品です。

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るため量産に適したモデルとなっています。

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。

機能特化型の製品である Armadillo-420 は、Armadillo-220 とピン互換で処理能力が約 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、Armadillo-WLAN モジュール(AWL13)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。Armadillo-500 開発セット、Armadillo-500 FX 液晶モデル、Armadillo-400 シリーズが対応しています。Armadillo-WLAN モジュール(AWL13)は、SDIO または USB ホスト機能を持ったボードであれば接続可能で、Armadillo-400 シリーズ、Armadillo-600 シリーズが対応しています。

液晶付きのパネルコンピューターやカメラ向けの最初の製品は、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ー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ープラットフォーム向けの製品で、Armadillo-420 をベースに、メモリサイズの拡張、LCD 接続を可能にしたモデルです。パネルコンピューターのみならず、Armadillo-440 をベースに Wi-SUN モジュールを搭載した Armadillo-Box WS1 もラインナップしています。Armadillo-410 は、Armadillo-440 と同じ機能を持ちながら、拡張基板を使って各インターフェースを自由に配置できるモジュールモデルです。

画像処理は Armadillo-800 シリーズで進化しました。Armadillo-840 では画面出力と表示機能に特化し、液晶パネルの他に HDMI による画面出力を可能にしました。また、リアルタイム制御用のサブ CPU やアクセラレータを持ち、H.264/AVC、AAC のエンコード・デコードに対応する動画再生や画像処理をメイン CPU に負荷をかけずに実現できるモデルです。Armadillo-840m は Armadillo-840 と同じ機能を持ちながら、拡張基板を使って各インターフェースを自由に配置できるモジュールモデルです。Armadillo-810 は、Armadillo-840 と同じ CPU を持ち、小型カメラ用途に特化した 50mm×50mm の製品です。

Armadillo-X1 は、i.MX7 デュアルコアを搭載し、ギガビットイーサネットにも対応したハイエンドモデルです。4GB の大容量 eMMC を搭載し、Armadillo-IoT シリーズでラインナップされているアドオンモジュールを実装することによる機能拡張も可能となっています。

Armadillo-640 は Armadillo-440 の思想を引き継ぎ、液晶パネルや無線 LAN 機能にも容易に対応可能なプラットフォーム製品となっています。

Armadillo-600 シリーズについては次章で詳しく紹介します。

Armadillo シリーズの詳しい仕様は、Armadillo サイトの製品を探す[4]により各製品の仕様をご確認ください。

4.2. Armadillo-600 シリーズ

Armadillo-640 は Armadillo-600 シリーズという製品群に属します。本書の内容の多くは Armadillo を使った開発全般に応用できるものですが、具体例は Armadillo-600 シリーズを対象としています。本章では、Armadillo-600 シリーズについて詳しく解説します。

4.2.1. Armadillo-600 シリーズの基本仕様

Armadillo-600 シリーズは、NXP Semiconductors 製 ARM Cortex-A7 プロセッサ i.MX6ULL、DDR3 SDRAM、eMMC 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-640 は、Armadillo-440 の後継モデルです。Armadillo-440 と形状互換を保ちながら、CPU 処理能力、RAM 容量が強化されています。また、Armadillo-400 シリーズと同様に、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-600 シリーズの基本仕様とブロック図を「表 4.2. Armadillo-600 シリーズ基本仕様」、「図 4.3. Armadillo-640 ブロック図」に示します。

表 4.2 Armadillo-600 シリーズ基本仕様

	Armadillo-640	プロセッサ	NXP i.MX6ULL
CPU コアクロック	528MHz	RAM	DDL3 SDRAM: 512MByte
eMMC	SLC NAND 4GB	本体基板サイズ	75.0mm × 50.0mm
電源電圧	DC4.75V~5.25V	消費電力	約 1.2W

[4]https://armadillo.atmark-techno.com/products

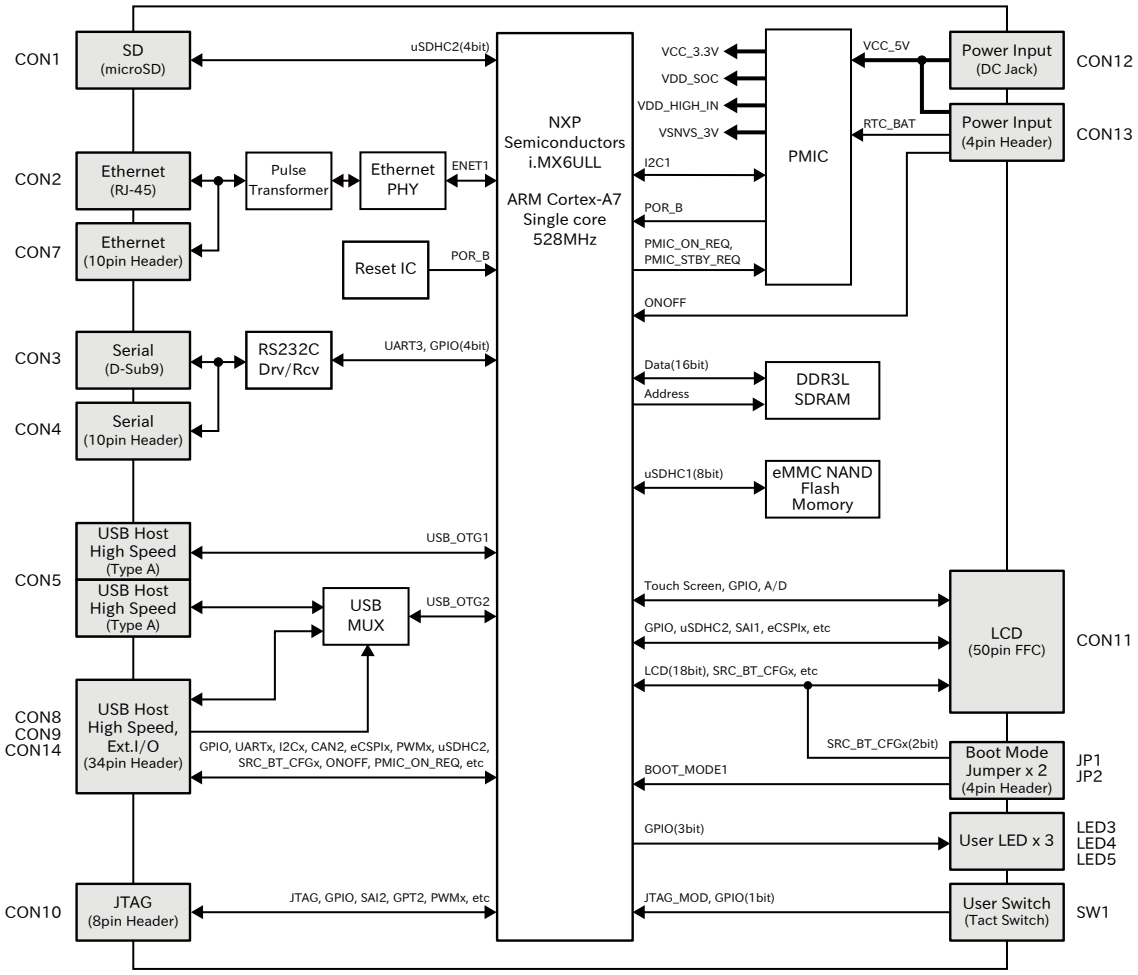

図 4.3 Armadillo-640 ブロック図

Armadillo-640 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 4.3. Armadillo-640 シリーズ拡張インターフェース」に示します[5]。

表 4.3 Armadillo-640 シリーズ拡張インターフェース

	Armadillo-640	拡張インターフェース (CON9)	GPIO、1-Wire、UART、SPI、PWM、SDHC
拡張インターフェース (CON14)	GPIO、1-Wire、I2C	LCD 拡張インターフェース (CON11)	GPIO、1-Wire、UART、I2C、SPI、PWM、SDHC、Audio、LCD

Armadillo-640 のインターフェースレイアウトを以下に示します。

[5]Armadillo-640 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。

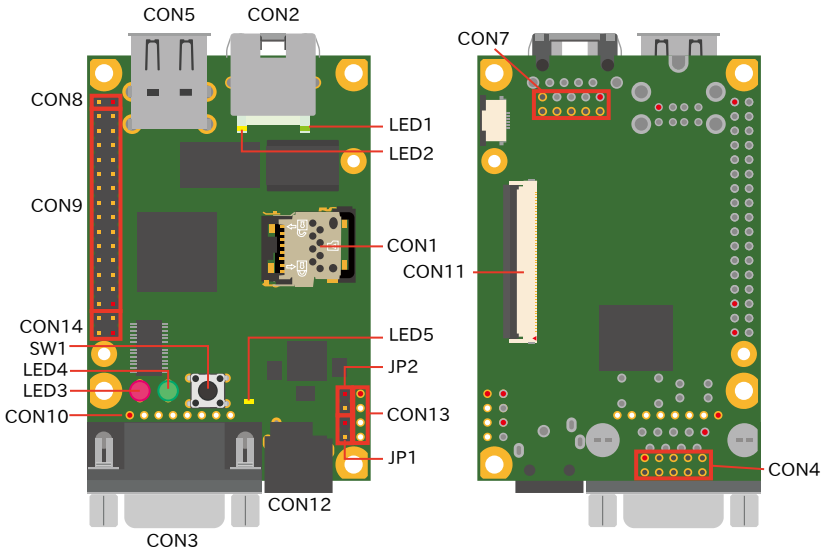

図 4.4 Armadillo-640 インターフェースレイアウト

Armadillo-600 シリーズの詳細な仕様については、Armadillo-640 製品マニュアル[6]を参照してください。

4.2.2. Armadillo-600 シリーズでできること

Armadillo-600 シリーズの特長として、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-600 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-600 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

4.2.2.1. ハードウェア機能の追加

Armadillo-600 シリーズは、USB や LAN、シリアル(RS-232C)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。

CON8, CON9, CON11, CON14 には、複数の機能(マルチプレクス)をもった i.MX6ULL の信号線、パワーマネジメント IC の ON/OFF 信号、i.MX6ULL の PWRON 信号等が接続されています。拡張できる機能の詳細につきましては、Armadillo-640 マルチプレクス表[7]をご参照ください。

複数箇所に割り当て可能な信号(USDHC2、UART1、ESPI1、I2C2 等)がありますが、同じ信号は複数ピンで同時利用できません。

[6]https://users.atmark-techno.com/armadillo-640/manual
[7]https://users.atmark-techno.com/armadillo-640/manual-multiplex

CTS/RTS 信号線を利用する際の注意点

i.MX6ULL の CTS、RTS 信号は一般的な UART の信号と名前が逆になっています。誤接続に注意してください。

CON9 のブートモード設定ピンについて

CON9 の 17 ピン(GPIO3_IO27)は、i.MX6ULL の内蔵 ROM によるブートモード設定ピンを兼用しています。電源投入時、ブードモード設定のために、基板上のプルダウン抵抗で Low レベルの状態を保持しています。High レベルの状態で電源投入した場合、SPI EEPROM ブートを有効/無効にするためのピン(BOOT_CFG4[6])に割り当てられているため、EEPROM recovery モードが有効になり、意図しない動作を引き起こす原因となります。電源投入時から U-Boot が動作するまでは、Low レベルを保持した状態でご使用ください。ブートモード設定の詳細につきましては、NXP Semiconductors のホームページからダウンロード可能な『i.MX6ULL Applications Processor Reference Manual』をご参照ください。

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-600 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 サイトの動作デバイスのページ[8]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 to シリアル変換ケーブル
3. USB メモリ
4. USB マウス
5. USB キーボード

4.2.2.2. オリジナルアプリケーションプログラムの追加

Armadillo-600 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 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。

4.2.2.3. 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-600 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 Debian 9(stretch)を採用しています。Debian GNU/Linux 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

[8]https://armadillo.atmark-techno.com/devices

ジが 51,000 個以上用意されており、これらの多くを簡単な手順で Armadillo-600 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-600 シリーズで実現可能なソフトウェア機能の一覧を「表 4.4. Armadillo-600 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 4.4 Armadillo-600 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
ネットワーク	Web(HTTP)サーバー	lighttpd
		Apache2
	FTP サーバー/クライアント	ftpd/ftp
	Telnet サーバー/クライアント	telnetd/telnet
	SSH サーバー/クライアント	OpenSSH
	DHCP サーバー/クライアント	dhcpd/udhcpc
	NTP サーバー/クライアント	ntpd/ntpclient
	PPP	pppd
	カメラサーバー	mjpg-streamer
	HTTP/FTP クライアント	wget/ftpget/ftpput
	Zeroconf	avahi
	メール送信(SMTP)	mail
		sendmail
	ファイヤーウォール	iptables
	SNMP	net-snmp
マルチメディア	オーディオ再生/録音	alsa-utils
	MP3 再生	mpg321
	画像処理ライブラリ	libjpeg62/libpng16
データベース	データベース	sqlite3
ファイルシステム	VFAT(FAT32)	mkdosfs
	EXT2	mke2fs
	EXT3	mke2fs
	jffs2	mtd-utils
	NFS	-
	samba	samba
GUI	ウィンドウシステム	The KDrive Tiny X Server
		X.org X Window System
	GUI ツールキット	Gtk+ 2/Gtk+ 3
wxWidgets/wxPython		言語
Perl	Perl 5	
PHP	PHP 7.0	
Python	Python 2/Python 3	
Ruby	Ruby 2.3	その他
シェル	ash	

4.3. 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イン、ユーザーランドアーカイブ作成ツールなどをインストールし、設定済みの環境を ATDE(Atmark Techno 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。

Armadillo での開発の手順は、「8. 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 Armadillo サイト[9]から無償で入手することができます。

4.4. 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。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、Armadillo-640 を例に示します。

4.4.1. 検討、試作

検討段階においては、まず、Armadillo-640 製品マニュアル[10]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-600 シリーズの場合は、OS として Linux システム(Debian GNU/Linux)にのみ対応しています。

OS として Linux システムを採用した場合の GUI 環境としては、これまでの X Window System やその上で動作する GTK+のみならず、wxWidgets のようなツールキットも候補に挙げることができます。その他、株式会社エイチアイの「exbeans® UI Conductor SDK」などの外部ツールにも対応可能です(外部ツールの情報は Armadillo サイトなどで随時公開しています)。

また、「表 4.4. Armadillo-600 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。様々なネットワークプロトコルに対応したサーバー/クライアントソフトウェアや、データベース等多くのソフトウェアが使用可能です。

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。

4.4.2. 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 サイト[11]を運営しています。Armadillo 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 サイトで探してみても疑問が解決しない場合は、Armadillo フォーラム[12]に質問することもできます。フォーラムでは、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のある人から、ヒントが得られるかもしれません。

マニュアル、Howto、FAQ 及びメーリングリスト[13]のアーカイブ(過去ログ)とフォーラムへの投稿は、Armadillo サイトのサイト内検索機能で串刺し検索できます。Armadillo サイトの右上にある検索ボックスにキーワードを入力して検索を実行すると、関連する情報が表示されます。ぜひ、ご活用下さい。

4.4.3. 量産

Armadillo のような汎用ボードコンピューターを使用して組み込みシステムを量産製造する場合、ケースへの組み込みやフラッシュメモリ(eMMC)の書き込みなどが課題となることがあります。

[9]https://armadillo.atmark-techno.com/
[10]https://users.atmark-techno.com/armadillo-640/manual
[11]https://armadillo.atmark-techno.com
[12]https://users.atmark-techno.com/forum/armadillo
[13]2014 年 2 月 1 日をもって、"フォーラム"サービス https://users.atmark-techno.com/forum に移行

アットマークテクノでは、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など、量産製造の効率化に役立つプランをご用意しています。サービスの詳細については、アットマークテクノ営業部までお問い合わせください。

4.4.4. 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して頂いたユーザーに対しては、製品のサポート情報(不具合情報、アップデート情報など)を E-mail 等にて通知しています。ユーザー登録はユーザーズサイト[14]で行えます。ぜひ、ご登録をお願い致します。

また、アットマークテクノ製品は製品ポリシーとして、製品発売から 5 年間は製品を供給できるよう設計開発を行っています[15]。もし製造中止となる場合は、事前にアナウンスを行います。

「ユーザーズサイト」では、製品ライフや部品変更などをお知らせする「変更通知」情報を公開しています。ユーザーズサイトでアカウント登録したユーザーを対象に、変更通知の内容を随時メールでお知らせするサービスも実施しています。このように、量産後の保守に必要な様々な情報を提供しているため、Armadillo は安心して量産にお使いいただけます。

[14] https://users.atmark-techno.com

[15] 他の部品への置き換えや基板改版など対応方法が存在する限り、供給継続努力を行います。万一 CPU など置き換え不能な部品が製造中止となるなど、やむを得ない事由により調達できなくなった場合、5 年の期間をまたず該当製品の販売を終了することがありますのでご了承ください。

5. 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 とはどういう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ーなのか、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。

5.1. Armadillo と作業用 PC との接続

5.1.1. 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 5.1. 必要な機材」に示します。

表 5.1 必要な機材

機材	説明
Armadillo	「Armadillo-640 ベーシックモデル開発セット」
作業用 PC	Linux または Windows が動作する i386 もしくは AMD64 互換 PC
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および Armadillo をインターネットに接続する場合に必要
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
microSD カード	初期化用インストールディスクを作成する場合やストレージデバイスとして使用する場合に必要

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す。

Armadillo のフラッシュメモリを書き換えている場合は、「Armadillo-640 製品マニュアル」の「イメージファイルの書き換え方法」を参照して、フラッシュメモリを初期化してください。

また、U-Boot の環境変数を設定している場合は、「Armadillo-640 製品マニュアル」の「ブートローダー (U-Boot) 仕様」を参照して、環境変数をデフォルトに戻してください。

5.1.2. 接続方法

「図 5.1. Armadillo-640 の接続例」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。

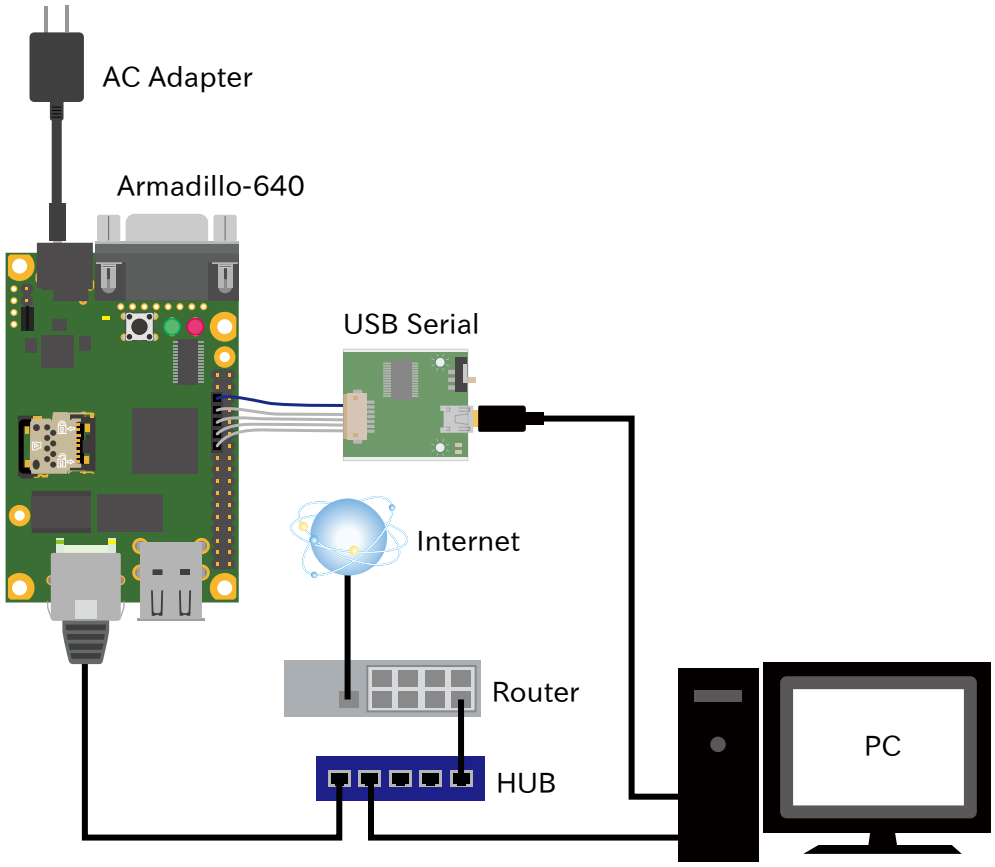

図 5.1 Armadillo-640 の接続例

5.1.3. USB シリアル変換アダプタの接続方法

USB シリアル変換アダプタは、青色のケーブルを 1 ピンとして、Armadillo-640 の CON9 1,3,5,7,9 ピンと接続します。

USB シリアル変換アダプタを接続するピンの隣だけ、CON9,CON14 を囲っているシルクが太くなっているのでそれを目印にして、下図のように接続してください。

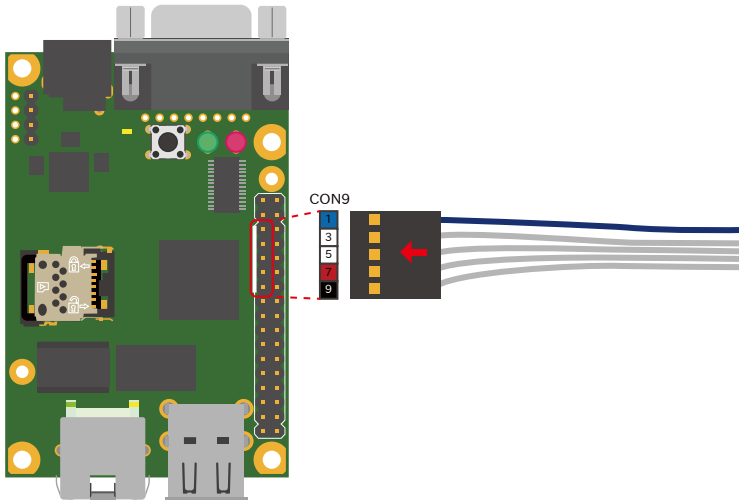

図 5.2 CON9-USB シリアル変換アダプタ接続図

5.1.4. Windows PC とのシリアル通信

作業用 PC が Linux の場合は、「5.1.5. Linux PC とのシリアル通信」に進んでください。

ここでは、Tera Term 4.102 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。

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(https://ttssh2.osdn.jp/)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 5.3. 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 5.3. Tera Term: 新しい接続画面」ダイアログを閉じてください。

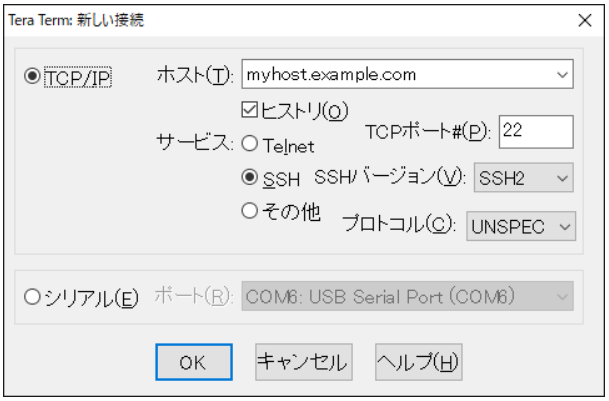

図 5.3 Tera Term: 新しい接続画面

2. 「図 5.4. 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。

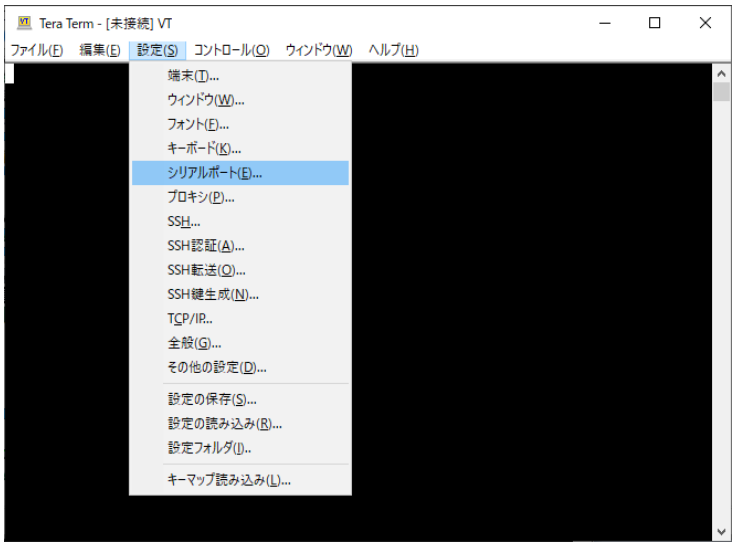

図 5.4 Tera Term 画面

3. 「図 5.5. Tera Term: シリアルポート設定画面」が表示されます。

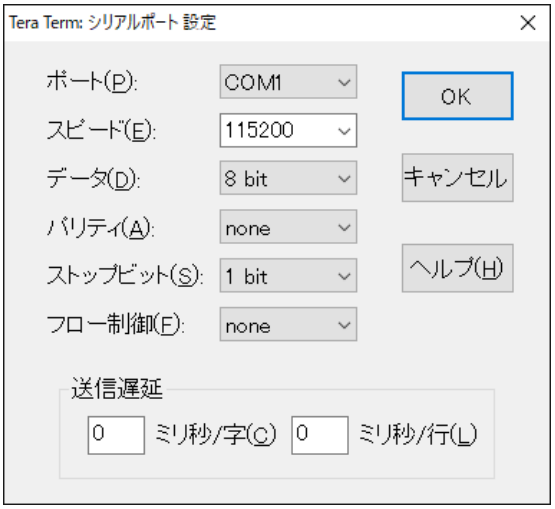

図 5.5 Tera Term: シリアルポート設定画面

4. 「ポート」に Armadillo と接続されているシリアルのポート番号を設定してください。

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 5.6. 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 COM1 が割り当てられています。

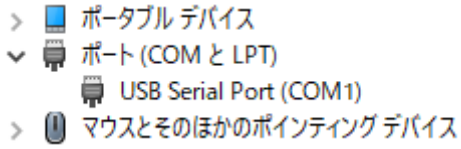

図 5.6 デバイスマネージャー画面

5. その他の設定項目は「表 5.2. シリアル通信設定」を参照し、「図 5.5. 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。

表 5.2 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
フロー制御	なし

6. 「OK」ボタンを押してください。

7. COM1 と接続できた場合、タイトルバーに「COM1」と表示されます。

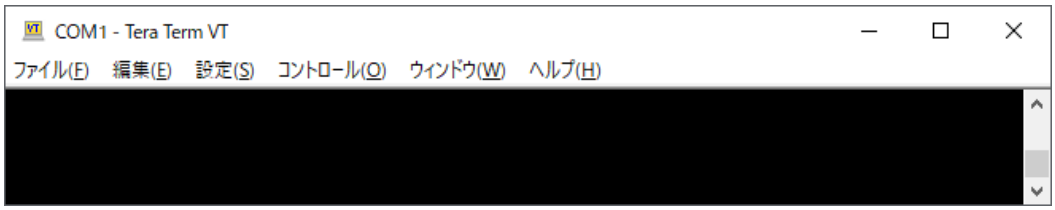

図 5.7 Tera Term 画面

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。

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。

5.1.5. Linux PC とのシリアル通信

ここでは minicom を使用します。

Debian などの作業用 PC に minicom をインストールするには「図 5.8. minicom のインストール」を実行します。

```
[PC ~]$ sudo apt install minicom
```

図 5.8 minicom のインストール

「表 5.3. シリアル通信設定」に示すパラメーターに設定します。

表 5.3 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 5.9. 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 ~]$ sudo LC_ALL=C minicom -s
```

図 5.9 minicom の設定の起動

2. 「図 5.10. 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 5.10 minicom の設定

3. 「図 5.11. 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/ttyUSB0                              |
     | 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 5.11 minicom のシリアルポートの設定

4. 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。ログが表示されなくても、dmesg コマンドを実行することで、ログを確認することができます。

```
usb 2-1.2: new full-speed USB device number 5 using ehci-pci
usb 2-1.2: New USB device found, idVendor=0403, idProduct=6001
usb 2-1.2: New USB device strings: Mfr=1, Product=2,
SerialNumber=3
usb 2-1.2: Product: FT232R USB UART
usb 2-1.2: Manufacturer: FTDI
usb 2-1.2: SerialNumber: A702ZLZ7
usbcore: registered new interface driver usbserial
usbcore: registered new interface driver usbserial_generic
usbserial: USB Serial support registered for generic
usbcore: registered new interface driver ftdi_sio
usbserial: USB Serial support registered for FTDI USB Serial
Device
ftdi_sio 2-1.2:1.0: FTDI USB Serial Device converter detected
usb 2-1.2: Detected FT232RL
usb 2-1.2: Number of endpoints 2
usb 2-1.2: Endpoint 1 MaxPacketSize 64
usb 2-1.2: Endpoint 2 MaxPacketSize 64
usb 2-1.2: Setting MaxPacketSize 64
usb 2-1.2: FTDI USB Serial Device converter now attached to
ttyUSB0
```

図 5.12 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。

5. F キーを押して Hardware Flow Control を No に設定してください。
6. G キーを押して Software Flow Control を No に設定してください。
7. キーボードの E キーを押してください。「図 5.13. 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。

```
                +---------[Comm Parameters]----------+
                |                                    |
```

```
|     Current: 115200 8N1            |
| Speed            Parity      Data  |
| A: <next>        L: None     S: 5  |
| B: <prev>        M: Even     T: 6  |
| C:   9600        N: Odd      U: 7  |
| D:  38400        O: Mark     V: 8  |
| E: 115200        P: Space          |
|                                    |
| Stopbits                           |
| W: 1             Q: 8-N-1          |
| X: 2             R: 7-E-1          |
|                                    |
|                                    |
| Choice, or <Enter> to exit?        |
+------------------------------------+
```

図 5.13 minicom のシリアルポートのパラメータの設定

8. 「図 5.13. minicom のシリアルポートのパラメータの設定」では、転送レート、データ長、ストップビット、パリティの設定を行います。

9. 現在の設定値は「Current」に表示されています。それぞれの値の内容は「図 5.14. minicom シリアルポートの設定値」を参照してください。

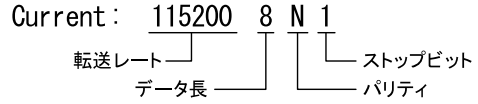

図 5.14 minicom シリアルポートの設定値

10. E キーを押して、転送レートを 115200 に設定してください。

11. Q キーを押して、データ長を 8、パリティを None、ストップビットを 1 に設定してください。

12. Enter キーを 2 回押して、「図 5.10. minicom の設定」に戻ってください。

13.「図 5.10. minicom の設定」から、「Save setup as dfl」を選択し、設定を保存してください。

14.「Exit from Minicom」を選択し、minicom の設定を終了してください。

minicom を起動させるには、「図 5.15. minicom 起動方法」のようにしてください。

```
[PC ~]$ sudo LANG=C minicom --wrap
```

図 5.15 minicom 起動方法

デバイスファイルを指定して起動させるには「図 5.16. minicom 起動方法(デバイスファイル指定)」のようにしてください。

```
[PC ~]$ sudo LANG=C minicom --wrap --device /dev/ttyUSB0
```

図 5.16 minicom 起動方法(デバイスファイル指定)

デバイスファイル名は、環境によって /dev/ttyS0 や /dev/ttyUSB1 など、本書の実行例とは異なる場合があります。

minicom がオープンする /dev/ttyS0 や /dev/ttyUSB0 といったデバイスファイルは、 root または dialout グループに属しているユーザーしかアクセスできません。

ユーザーを dialout グループに入れることで、以降、sudo を使わずに minicom で /dev/ttyUSB0 をオープンすることができます。

```
[PC ~]$ sudo usermod -aG dialout atmark
[PC ~]$ LANG=C minicom --wrap
```

minicom を終了させるには、まず Ctrl-a に続いて q キーを入力します。その後、以下のように表示されたら「Yes」にカーソルを合わせて Enter キーを入力すると minicom が終了します。

```
+-----------------------+
| Leave without reset? |
|     Yes       No     |
+-----------------------+
```

図 5.17 minicom 終了確認

Ctrl-a に続いて z キーを入力すると、minicom のコマンドヘルプが表示されます。

5.2. Armadillo の起動

Armadillo は、電源(AC アダプタ)と接続すると自動で起動するようになっています。

Armadillo が起動すると、PC のシリアル通信ソフトウェアに以下のような起動ログが表示されます。Armadillo の起動シーケンス及び起動ログの詳細については、「6. Armadillo が動作する仕組み」で説明します。

```
U-Boot 2018.03-at4 (Dec 26 2018 - 19:21:13 +0900)

CPU:   Freescale i.MX6ULL rev1.0 at 396 MHz
Reset cause: POR
I2C:   ready
DRAM:  512 MiB
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
```

```
Loading Environment from MMC... OK
In:    serial
Out:   serial
Err:   serial
PMIC: PFUZE3000 DEV_ID=0x30 REV_ID=0x11
Net:   FEC
Hit any key to stop autoboot:  0
6105312 bytes read in 194 ms (30 MiB/s)
27015 bytes read in 54 ms (488.3 KiB/s)
## Booting kernel from Legacy Image at 82000000 ...
   Image Name:   Linux-4.14-at11
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    6105248 Bytes = 5.8 MiB
   Load Address: 82000000
   Entry Point:  82000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 83000000
   Booting using the fdt blob at 0x83000000
   Loading Kernel Image ... OK
   Loading Device Tree to 9eefc000, end 9ef05986 ... OK

Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Linux version 4.14-at11 (atmark@atde7) (gcc version 6.3.0 2017051
6 (Debian 6.3.0-18)) #1 Tue Jan 29 10:11:05 JST 2019
[    0.000000] CPU: ARMv7 Processor [410fc075] revision 5 (ARMv7), cr=10c53c7d
[    0.000000] CPU: div instructions available: patching division code
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instructio
n cache
[    0.000000] OF: fdt: Machine model: Atmark Techno Armadillo-640
:
:
:
```

図 5.18 起動ログ

5.3. 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-640 のログインプロンプトを以下に示します。

```
Debian GNU/Linux 9 armadillo ttymxc0

armadillo login:
```

Armadillo-640 では、「表 5.4. 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」に示すユーザーとパスワードでログインすることができます。

表 5.4 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

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

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。

```
Debian GNU/Linux 9 armadillo ttymxc0

armadillo login: root ❶
Password: root ❷
```

図 5.19 ユーザー名とパスワードを入力してログインする

❶ ユーザー名に「root」と入力した後、Enter キーを入力します。

❷ パスワードに「root」と入力した後、Enter キーを入力します。パスワードは入力しても表示されません。

5.4. 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-640 でログインした場合のプロンプトを以下に示します。[1]

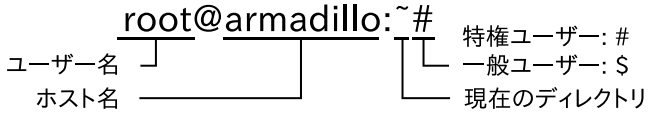

図 5.20 Armadillo-640 でのプロンプト表示例

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プログラムで、コマンドラインインタープリターとよばれることもあります。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 5.21 コマンドの書式

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

```
echo [string...]
```

図 5.22 echo コマンドの書式

「図 5.22. echo コマンドの書式」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。

[1]以降は、得に理由がない限り「1.3. 表記について」のプロンプトの表記を使用します。

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 ❶

[armadillo ~]# echo hello ❷
hello
[armadillo ~]# echo hello world! ❸
hello world!
```

図 5.23 echo コマンドの実行例

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 5.19. ユーザー名とパスワードを入力してログインする」や「図 5.23. 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。

5.5. 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 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
[  OK  ] Stopped target Timers.
[  OK  ] Stopped target Graphical Interface.
[  OK  ] Stopped target Multi-User System.
[  OK  ] Stopped target Login Prompts.
         Stopping System Logging Service...
         Unmounting /opt/license...
[  OK  ] Stopped Daily Cleanup of Temporary Directories.
[  OK  ] Stopped Daily apt upgrade and clean activities.
         Stopping User Manager for UID 0...
         Stopping D-Bus System Message Bus...
         Stopping Serial Getty on ttymxc0...
         Stopping Getty on tty1...
:
:
:
[   26.197097] reboot: System halted
```

「reboot: System halted」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。

注意: 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/SD 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に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[2]してから電源を切断してください。

5.6. 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。

5.6.1. 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、.はカレントディレクトリを、..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。

5.6.2. 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。

[2] umount コマンドを使用します。

表 5.5 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
cd [dir]	ディレクトリを移動します
pwd	カレントディレクトリを表示します
mkdir <dir>	ディレクトリを作成します
rmdir <dir>	空のディレクトリを削除します
ls [dir]	指定したディレクトリにあるファイルを表示します
cat <file>	ファイルの内容を表示します
cp <from> <to>	ファイルとディレクトリをコピーします
mv <from> <to>	ファイルやディレクトリを移動または名称変更します
rm <file>	ファイルとディレクトリを削除します

「表 5.5. ディレクトリとファイル操作に関するコマンド」に示したコマンドを使用して、以下の内容を行っていきます。

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、実際にコマンドを実行し、ファイルの操作を行います。

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を調べてみます。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ことが確認できます。

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す。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す。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。

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す。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。

```
[armadillo ~]# 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す。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す。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/hostname)を dir ディレクトリにコピーしてください。[3]

```
[armadillo ~]# cp /etc/hostname dir
```

   dir ディレクトリに hostname ファイルをコピーしました。

[3]Linux ではファイル名の大文字、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。この時、表示されるファイル数が多い場合は「Display all 141 possibilities? (y or n)」と画面に表示され、y キーを押すと 1 ページずつ表示することができます。表示するファイルが残っている場合は「--More--」と表示され、スペースキーを押すたびに 1 ページ分を追加で表示し、q キーを押すと途中で表示をやめることができます。

```
[armadillo ~]# cp /etc/TabTab
Display all 141 possibilities? (y or n)
.pwd.lock                   manpath.config
X11/                        mime.types
adduser.conf                mke2fs.conf
aliases                     modules-load.d/
alternatives/               motd
apt/                        mtab
bash.bashrc                 mysql/
bash_completion             network/
bash_completion.d/          networks
bindresvport.blacklist      nsswitch.conf
binfmt.d/                   opt/
ca-certificates/            os-release
ca-certificates.conf        pam.conf
calendar/                   pam.d/
cron.d/                     passwd
cron.daily/                 passwd-
cron.hourly/                perl/
cron.monthly/               ppp/
cron.weekly/                profile
crontab                     profile.d/
dbus-1/                     protocols
debconf.conf                python2.7/
debian_version              rc.local
--More--
```

補完は、コマンドにも有効です。以下のように「c」と入力したあと、Tab キーを 2 回連続で入力すると、「c」から始まる実行可能なコマンドをすべて表示してくれます。

```
[armadillo ~]# cTabTab
c2ph                            clear_console
c_rehash                        cmp
cal                             col
calendar                        colcrt
caller                          colrm
capsh                           column
captoinfo                       comm
case                            command
cat                             compgen
catchsegv                       complete
catman                          compopt
cd                              compose
cfdisk                          continue
chage                           coproc
chattr                          corelist
chcon                           cp
chcpu                           cpan
chfn                            cpan5.24-arm-linux-gnueabihf
chgpasswd                       cpgr
chgrp                           cppw
chmod                           cron
chown                           crontab
chpasswd                        cryptdir
--More--
```

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す。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す。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
armadillo
```

   cat コマンドの結果として armadillo と表示されます。armadillo 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す。ファイルの名前を変更するコマンドは mv コマンドです[4]。

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
armadillo
```

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 ファイルができています。cat コマンドで name ファイルの中身を見てみると、確かに hostname ファイルと同じであることが確認できます。

9. ファイルを削除する。

   次は、ファイルを削除してみます。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。

    次に、ディレクトリを削除してみます。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。rmdir コマンドは引数に指定したディレクトリが空でない場合はエラーが発生し、ディレクトリを削除することができません。空でないディレクトリを削除する場合は、rm コマンドに-r オプションを付けて実行することで削除できます。

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]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」コマンドとしても使用できます。

5.7. 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。

5.7.1. 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起動します。

5.7.2. 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 5.6. 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 5.6 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 5.24. 入力モードに移行するコマンドの説明」に示します。

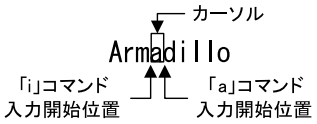

図 5.24 入力モードに移行するコマンドの説明

vi での文字削除

コンソールの環境によっては BS(Backspace)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、「5.7.4. 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。

5.7.3. 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 5.7. 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 5.7 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

5.7.4. 文字の削除

文字を削除する場合は、コマンドモードで「表 5.8. 削除コマンド」に示すコマンドを入力します。

表 5.8 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 5.25. 文字を削除するコマンドの説明」に示します。

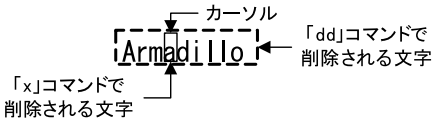

図 5.25 文字を削除するコマンドの説明

5.7.5. 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 5.9. 保存・終了コマンド」に示します。

表 5.9 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。

5.8. ネットワークを使う

Armadillo の標準状態では、 DHCP(Dynamic Host Configuration Protocol) でネットワークの設定を行うため、PC と同じ様に LAN ケーブルを接続するだけでネットワークに接続可能です。固定 IP アドレスの設定が必要な場合は「5.8.1. インターフェースの無効化」以降の手順を実施してください。

ネットワーク接続に関する不明な点については、ネットワークの管理者へ相談してください。

ここで Armadillo をインターネットに接続できるようにしておくことをおすすめします。

Debian GNU/Linux を標準のルートファイルシステムとしている Armadillo では、開発時に Armadillo をインターネットに接続することは非常に有用です。インターネットに接続していれば、パッケージ管理システム APT(Advanced Packaging Tool) を使用することで、Debian Project が供給している様々なパッケージを簡単に Armadillo に追加することができます。

5.8.1. インターフェースの無効化

設定を行う前にインターフェースを無効化します。有効化したまま設定を変更しても、設定が反映されない場合があります。

```
[armadillo ~]# ifdown eth0
```

5.8.2. ネットワークの設定

ここでの設定はあくまでも例です。実際の IP アドレス等はネットワークの管理者に確認してください。

Armadillo のネットワークの設定ファイルは/etc/network/interfaces です。

「表 5.10. 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/network/interfaces を、「図 5.26. 設定変更後の interfaces ファイル」で示す内容に変更します。

表 5.10 固定 IP アドレス設定例

項目	設定
IP アドレス	192.0.2.10
ネットマスク	255.255.255.0
ネットワークアドレス	192.0.2.0
ブロードキャストアドレス	192.0.2.255
デフォルトゲートウェイ	192.0.2.1

```
[armadillo ~]# vi /etc/network/interfaces
# interfaces(5) file used by ifup(8) and ifdown(8)

auto lo eth0
iface lo inet loopback
iface eth0 inet static
    address 192.0.2.10
    netmask 255.255.255.0
    network 192.0.2.0
    broadcast 192.0.2.255
    gateway 192.0.2.1
```

図 5.26 設定変更後の interfaces ファイル

5.8.3. 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 5.27. 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/network/interfaces
```

図 5.27 interfaces ファイルを開く

2. 「図 5.28. 標準イメージの interfaces ファイル」が表示されます。製品出荷のイメージでは、DHCP サーバーから IP アドレスをもらいうけるように設定されています。

```
# interfaces(5) file used by ifup(8) and ifdown(8)

allow-hotplug eth0
auto lo
iface lo inet loopback
iface eth0 inet dhcp
```

図 5.28 標準イメージの interfaces ファイル

3. カーソルを「図 5.29. interfaces ネットワーク設定」の位置に合わせます。

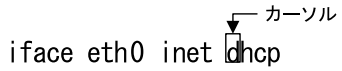

図 5.29 interfaces ネットワーク設定

4. 「x」コマンドを 4 回入力し「dhcp」を削除します。
5. 「a」コマンドを入力し、入力モードに移行します。
6. 「static」と入力し、固定 IP アドレスの設定に変更します。
7. Enter キーを入力し改行します。
8. 「図 5.26. 設定変更後の interfaces ファイル」と同じ内容になるよう、「address」と「netmask」「network」「broadcast」「gateway」を入力してください。
9. ESC キーを押して、コマンドモードに戻ります。
10. 「: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。

5.8.4. ネットワーク設定の反映(インターフェースの有効化)

インターフェースの有効化することで、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 ~]# ifup eth0
```

図 5.30 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「ip」コマンドで確認できます。

```
[armadillo ~]# ip address
ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
 link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
 inet 127.0.0.1/8 scope host lo
 valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP gr
oup default qlen 1000
 link/ether 00:11:0c:00:07:a4 brd ff:ff:ff:ff:ff:ff
 inet 192.0.2.10/24 brd 192.168.2.255 scope global eth0
 valid_lft forever preferred_lft forever
```

「inet」(IP アドレス/サブネットマスク)が「192.0.2.10/24」になっていることを確認してください。

5.8.5. パッケージ管理システム APT(Advanced Packaging Tool)を使用する

APT を使用可能な環境か確認するために、「図 5.31. 利用可能なパッケージのインデックスを取得する」を実施してみます。apt-get update はインターネット上の Debian サイト(HTTP サーバー)から利用可能なパッケージのインデックスを取得します。

```
[armadillo ~]# apt-get update ; echo "Result: $?"
Ign:1 http://download.atmark-techno.com/debian stretch InRelease
Hit:2 http://download.atmark-techno.com/debian stretch Release
Ign:3 http://ftp.jp.debian.org/debian stretch InRelease
Hit:4 http://ftp.jp.debian.org/debian stretch Release
Get:5 http://security.debian.org stretch/updates InRelease [94.3 kB]
Get:8 http://security.debian.org stretch/updates/main Sources [196 kB]
Get:9 http://security.debian.org stretch/updates/main armhf Packages [464 kB]
Get:10 http://security.debian.org stretch/updates/main Translation-en [212 kB]
Fetched 966 kB in 6s (148 kB/s)
Reading package lists... Done
Result: 0
```

図 5.31 利用可能なパッケージのインデックスを取得する

上記コマンドを実行して "Result: 0" とならない場合、Debian サイトが一時的に利用不可能になっている、インターネットに接続できていない、などの可能性が考えられます。

Armadillo を製品として運用する際には、「Armadillo がインターネットに接続できているか確認する」目的で apt-get update を利用しないでください。

パッケージを追加するなどの apt-get update 以外の APT の機能については、以降の手順で実際に必要になった際に都度説明することにします。

6. 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。

6.1. 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。

6.1.1. 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-600 シリーズでは U-boot ブートローダー(以降、単に U-boot と記述します)を使用します。

U-boot は、二つの動作モードを持っています。一つは、オートブートモードです。オートブートモードでは、カーネルイメージをブートデバイス[1]から読み出し、メモリに展開してからカーネルに制御を移します。この動作は、「6.2.1. ブートローダーが行う処理」で詳しく説明します。

もう一つの動作モードは保守モードで、コンソールにプロンプトを表示し、ユーザーが入力したコマンドに応じて U-Boot の環境変数の変更などを行えます。

6.1.2. Linux カーネル

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。Armadillo-600 シリーズでは、標準のカーネルとして Linux を使用します。

Linux カーネルの初期化処理については、「6.2.2. カーネルの初期化処理」で詳しく説明します。

6.1.3. ユーザーランド

ユーザーランドとは、アプリケーションプログラムやライブラリ、設定ファイル、データファイル、デバイスファイルなど Linux システムが動作する上で必要なカーネル以外のものをいいます。

カーネルとユーザーランドとのインターフェース

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(Application Program Interface)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を呼び出します。

Linux システムでは、ユーザーランドのファイルとディレクトリ[2]同士の位置関係を階層的な木構造として表現します。ファイルとディレクトリの木構造をファイルシステムといいます。また、木構造の最

[1]標準設定の場合は、eMMC メモリです。Armadillo-600 シリーズでは microSD も指定可能です
[2]Windows でのフォルダと同様の概念です。

上位に位置するディレクトリをルートディレクトリといいます。全ての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。

ファイルシステムは、通常、ストレージデバイス[3]に書き込まれて使用されます。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。特に、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を、ルートファイルシステムといいます。

Windows のファイルシステムとの違い

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。一方で、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。つまり、USB メモリを/mnt/usb ディレクトリにマウントし、SD カードを/mnt/usb/sd 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。

Armadillo-600 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、Debian 9(stretch)を採用しています。

6.2. 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詳しく説明します。

大まかな起動の流れは、以下のようになります。

1. ブートローダーが行う処理
    a. ブートローダが起動し最低限のハードウェア初期化を行う
    b. カーネルイメージをメモリにロードする
    c. ブートローダーを抜けて、カーネルに実行を移す
2. カーネルの初期化処理
    a. カーネルが様々なコアやデバイスドライバの初期化処理をおこなう
    b. ルートファイルシステムをマウントする
    c. 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
3. ユーザーランドの初期化処理
    a. systemd がシステム初期化処理を実行する

[3]HDD(ハードディスクドライブ)や USB メモリなど

b. systemd が systemd-logind.service を起動する

c. systemd が serial-getty@ttymxc0.service を起動する

d. systemd-logind がログイン画面を表示する

6.2.1. ブートローダーが行う処理

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。

Armadillo-640 では、ジャンパの設定を変更することで、microSD カードと eMMC のどちらからブートローダーを起動するのか選択できます。ユーザーが操作を誤って eMMC の内容を消去してしまっても、microSD カードからブートローダーを起動することで復旧が可能です。

表 6.1 ジャンパの設定と起動デバイス

JP1	JP2	起動デバイス
オープン	ショート	eMMC
ショート	ショート	microSD

ジャンパのオープン、ショートとは

「オープン」とはジャンパピンにジャンパソケットを接続していない状態です。

「ショート」とはジャンパピンにジャンパソケットを接続している状態です。

起動した U-boot は、最低限のハードウェアの初期化を行った後、自分自身をメモリ(RAM)にコピーします。

その後の動作は、スライドスイッチの設定によって決定されます。スライドスイッチを、「図 6.1. スライドスイッチの設定」の 1 側に設定しておくと保守モード、2 側に設定しておくとオートブートモードとなります。

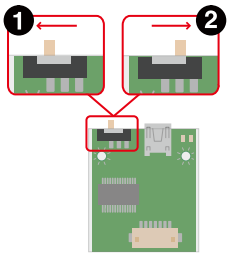

図 6.1 スライドスイッチの設定

❶ ブートローダーは保守モードになります。保守モードでは、ブートローダーのコマンドプロンプトが起動します。

❷ ブートローダーはオートブートモードになります。オートブートモードでは、ブートローダーのコマンドプロンプトが表示されず、OS を自動起動します。

6.2.1.1. 保守モード

スライドスイッチを、「図 6.1. スライドスイッチの設定」の 1 側に設定した状態で起動すると、U-Boot は保守モードとなります。保守モードの場合、U-Boot は、コンソールにプロンプトを表示してコマンド入力待ち状態となります。保守モードでは、U-Boot の環境変数の変更などを行うことができます。

boot コマンドを実行することで、U-Boot はカーネルを起動するための処理を開始します。

参考として、「図 6.2. U-boot 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。

```
U-Boot 2018.03-at4 (Dec 26 2018 - 19:21:13 +0900)

CPU:   Freescale i.MX6ULL rev1.0 at 396 MHz
Reset cause: POR
I2C:   ready
DRAM:  512 MiB
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
Loading Environment from MMC... OK
In:    serial
Out:   serial
Err:   serial
PMIC: PFUZE3000 DEV_ID=0x30 REV_ID=0x11
Net:   FEC
=>
```

図 6.2 U-boot 保守モード時の表示

6.2.1.2. オートブートモード

スライドスイッチを、「図 6.1. スライドスイッチの設定」の 2 側に設定した状態で起動すると、U-Boot はオートブートモードとなります。これが通常運用での設定です。オートブートモードの場合、U-Boot は、コマンド入力待ち状態になることなく、自動的にカーネルを起動するための処理を開始します。

6.2.1.3. Linux カーネルのブート

次に U-Boot は、環境変数で設定されたストレージデバイスから Linux カーネルイメージと DTB(Device Tree Blob)をメモリにロードします。

Device Tree は、ハードウェア情報を記述したデータ構造体です。ハードウェアの差分を Device Tree に記述することによって、1 つの Linux カーネルイメージを複数のハードウェアで利用することができるようになります。

DTB(Device Tree Blob)は、Device Tree Source(Device Tree を記述したソースファイル)をビルドして生成されるバイナリファイルです。

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。

U-Boot 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 6.3. U-Boot オートブートモード時の表示」に示します。

```
U-Boot 2018.03-at4 (Dec 26 2018 - 19:21:13 +0900)  ❶

CPU:   Freescale i.MX6ULL rev1.0 at 396 MHz
Reset cause: POR
I2C:   ready
DRAM:  512 MiB
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
Loading Environment from MMC... OK
In:    serial
Out:   serial
Err:   serial
PMIC: PFUZE3000 DEV_ID=0x30 REV_ID=0x11
Net:   FEC
Hit any key to stop autoboot:  0
6105312 bytes read in 194 ms (30 MiB/s)
27015 bytes read in 54 ms (488.3 KiB/s)
## Booting kernel from Legacy Image at 82000000 ...  ❷
   Image Name:   Linux-4.14-at11
   Image Type:   ARM Linux Kernel Image (uncompressed)

   Data Size:    6105248 Bytes = 5.8 MiB
   Load Address: 82000000
   Entry Point:  82000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 83000000  ❸
   Booting using the fdt blob at 0x83000000
   Loading Kernel Image ... OK
   Loading Device Tree to 9eefc000, end 9ef05986 ... OK

Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0  ❹
[    0.000000] Linux version 4.14-at11 (atmark@atde7) (gcc version 6.3.0 2017051
6 (Debian 6.3.0-18)) #1 Tue Jan 29 10:11:05 JST 2019
[    0.000000] CPU: ARMv7 Processor [410fc075] revision 5 (ARMv7), cr=10c53c7d
[    0.000000] CPU: div instructions available: patching division code
:
:
:
```

図 6.3 U-Boot オートブートモード時の表示

❶ U-Boot はシリアル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 この行から数行に渡って、ロードされたカーネルの情報を表示されます。

❸ この行から数行に渡って、ロードされた Device Tree Blob の情報を表示します。

❹ この行以降は、カーネルが表示しています。

6.2.2. 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-640 の起動ログを「図 6.4. Armadillo-640 カーネルブートログ」に示します。

```
[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Linux version 4.14-at11 (atmark@atde7) (gcc version 6.3.0 2017051
6 (Debian 6.3.0-18)) #1 Tue Jan 29 10:11:05 JST 2019
[    0.000000] CPU: ARMv7 Processor [410fc075] revision 5 (ARMv7), cr=10c53c7d
[    0.000000] CPU: div instructions available: patching division code
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instructio
n cache
[    0.000000] OF: fdt: Machine model: Atmark Techno Armadillo-640
[    0.000000] Memory policy: Data cache writeback
[    0.000000] cma: Reserved 16 MiB at 0x9f000000
[    0.000000] CPU: All CPU(s) started in SVC mode.
[    0.000000] Built 1 zonelists, mobility grouping on.  Total pages: 129920 ❶
[    0.000000] Kernel command line: root=/dev/mmcblk0p2 rootwait ❷
[    0.000000] PID hash table entries: 2048 (order: 1, 8192 bytes)
[    0.000000] Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
[    0.000000] Inode-cache hash table entries: 32768 (order: 5, 131072 bytes)
[    0.000000] Memory: 491980K/524288K available (5120K kernel code, 227K rwdata
, 1156K rodata, 3072K init, 230K bss, 15924K reserved, 16384K cma-reserved)
[    0.000000] Virtual kernel memory layout:
[    0.000000]     vector  : 0xffff0000 - 0xffff1000   (   4 kB)
[    0.000000]     fixmap  : 0xffc00000 - 0xfff00000   (3072 kB)
[    0.000000]     vmalloc : 0xe0800000 - 0xff800000   ( 496 MB)
[    0.000000]     lowmem  : 0xc0000000 - 0xe0000000   ( 512 MB)
[    0.000000]       .text : 0xc0008000 - 0xc0600000   (6112 kB)
[    0.000000]       .init : 0xc0800000 - 0xc0b00000   (3072 kB)
[    0.000000]       .data : 0xc0b00000 - 0xc0b38e60   ( 228 kB)
[    0.000000]        .bss : 0xc0b3dac8 - 0xc0b77644   ( 231 kB)
[    0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
[    0.000000] NR_IRQS: 16, nr_irqs: 16, preallocated irqs: 16
[    0.000000] Switching to timer-based delay loop, resolution 41ns ❸
:
各デバイスやカーネル内部の初期化処理
:
[    2.194104] mmcblk0: mmc0:0001 Q2J55L 3.53 GiB
[    2.203781] mmcblk0boot0: mmc0:0001 Q2J55L partition 1 2.00 MiB
[    2.214910] mmcblk0boot1: mmc0:0001 Q2J55L partition 2 2.00 MiB
[    2.226019] mmcblk0gp0: mmc0:0001 Q2J55L partition 4 8.00 MiB
[    2.237064] mmcblk0gp1: mmc0:0001 Q2J55L partition 5 8.00 MiB
[    2.247819] mmcblk0gp2: mmc0:0001 Q2J55L partition 6 8.00 MiB
[    2.258427] mmcblk0gp3: mmc0:0001 Q2J55L partition 7 8.00 MiB
[    2.268840] mmcblk0rpmb: mmc0:0001 Q2J55L partition 3 4.00 MiB
[    2.285905]  mmcblk0: p1 p2 p3 ❹
[    2.296675] input: gpio-keys as /devices/soc0/gpio-keys/input/input0
[    2.308365] snvs_rtc 20cc000.snvs:snvs-rtc-lp: setting system clock to 1970-0
1-01 00:00:00 UTC (0)
[    2.326849] 5V: disabling
[    2.338196] Warning: unable to open an initial console.
[    2.353434] Freeing unused kernel memory: 3072K
[    2.510160] systemd-udevd[74]: starting version 215
[    2.529550] random: systemd-udevd: uninitialized urandom read (16 bytes read)
[    3.730124] EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. Opts: (null) ❺
[    4.102740] systemd[1]: System time before build time, advancing clock.
[    4.132433] random: systemd: uninitialized urandom read (16 bytes read)
```

```
[    4.147338] random: systemd: uninitialized urandom read (16 bytes read)
[    4.166155] systemd[1]: systemd 232 running in system mode. (+PAM +AUDIT +SEL
INUX +IMA +APPARMOR +SMACK +SYSVINIT +UTMP +LIBCRYPTSETUP +GCRYPT +GNUTLS +ACL +
XZ +LZ4 +SECCOMP +BLKID +ELFUTILS +KMOD +IDN)
[    4.200049] systemd[1]: Detected architecture arm.

Welcome to Debian GNU/Linux 9 (stretch)!
```

図 6.4 Armadillo-640 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、「root=/dev/mmcblk0p2 rootwait」が使用されています。

❸ ここから、各デバイスやカーネル内部の初期化処理が始まります。

❹ eMMC の第 1 から第 3 パーティションが認識されています。

❺ eMMC の第 2 パーティションをマウントしています。

6.2.3. ユーザーランドの初期化処理

カーネルは、初期化処理が終えた後、 initramfs を一時的なルートファイルシステムとしてマウントし、initramfs 上のルートディレクトリにある init を実行します。

initramfs の init は、procfs [4]や sysfs のマウントや、systemd-udevd の起動などを行います。その後、/dev/mmcblk0p2[5](eMMC の第 2 パーティション)を本当のルートファイルシステムとしてマウントし、/sbin/init を実行します。

Armadillo-640(systemd を使用するユーザーランド)では、/sbin/init は/lib/systemd/systemd へのシンボリックリンクとなっており、実際には systemd というアプリケーションプログラムが実行されます。

起動時の処理状況を表示するシステムコンソールには、/dev/console を使用します。/dev/console の実体は、Device Tree の/chosen/stdout-path で指定したデバイスがデフォルトとして使用されます。このデバイスは、カーネルパラメータの console オプションを指定することで変更することができます。「図 6.4. Armadillo-640 カーネルブートログ」で示したように、Armadillo-640 では標準では console が指定されていません。Armadillo-640 の標準イメージでは、/chosen/stdout-path に UART1 (/dev/ttymxc0) が指定されているため、/dev/ttymxc0 がシステムコンソールとして使われます。

Systemd の概要

Systemd とは Linux のシステム管理デーモンの一つです。高速なシステム起動/終了や設定ファイルによるシステム管理の共通化、柔軟なプロセス起動を行うことができます。

Linux の起動を行う処理は今まで SysVinit が広く用いられてきました。しかし、並列処理ができない事や、タイマー起動やプロセス起動を行う事ができず、柔軟な起動を行うことができないという問題を抱えていました。

[4]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。
[5]このデバイスは、カーネルパラメータの root に指定されています。

この問題を解決するために Systemd が登場し、多くの Linux ディストリビューションで採用されるようになりました。

Debian GNU/Linux では、Debian 8.0 コードネーム: jessie から、デフォルトのシステム管理デーモンに Systemd が使われています。

Systemd では、プロセスの起動やファイルシステムのマウントなどの処理を "Unit" という単位で管理します。ある Unit が起動した後に起動する、ある Unit と同時に起動する、など細かな制御が可能です。

Systemd はまず、Systemd unit generators(もしくは単に Generators)と呼ばれるプログラム群[6]を実行します。Armadillo-640 ではこれらのプログラムは /lib/systemd/system-generators ディレクトリに配置されています。

表 6.2 Systemd unit generators

名称	説明
systemd-cryptsetup-generator	/etc/crypttab を元に Unit を生成するジェネレータ
systemd-debug-generator	ランタイムデバッグシェルを有効にし、起動時に特定のユニットをマスクするためのジェネレータ
systemd-fstab-generator	/etc/fstab を元に Unit [a]を生成するジェネレータ
systemd-getty-generator	カーネルコンソールを使用する serial-getty@.service Unit [b]を自動的に生成するジェネレータ
systemd-gpt-auto-generator	GPT パーティションタイプの GUID に基づいて、ルートパーティション、 /home パーティション、および /srv パーティションを自動的に検出してマウントし、スワップパーティションを検出して有効にするジェネレータ
systemd-hibernate-resume-generator	カーネルパラメータ resume= の値に従って systemd-hibernate-resume@.service Unit を生成するジェネレータ
systemd-rc-local-generator	/etc/rc.local [c]を起動時に、 /usr/sbin/halt をシャットダウン時に実行するようにするためジェネレータ
systemd-system-update-generator	/system-update が存在する場合に、起動プロセスを system-update.target に自動的にリダイレクトするジェネレータ
systemd-sysv-generator	SysV init のスクリプトを Systemd 上で実行するための Unit を自動的に生成するジェネレータ

[a]標準状態の Armadillo-640 では、-.mount と opt-license.mount が生成されます。
[b]標準状態の Armadillo-640 では、serial-getty@ttymxc0.service が生成されます。
[c]標準状態の Armadillo-640 では、 rc-local.service が生成されます。

次に Systemd は、すべての Unit ファイルを読み込み、各 Unit の依存関係にしたがって各起動処理を行います。この時に、serial-getty@ttymxc0.service および systemd-logind.service が起動され、systemd-logind によってログイン画面が表示されます。

Systemd が起動した Unit の順番は systemd-analyze コマンドで調べることができます。

次のように systemd-analyze に plot 引数を付けて実行すると、各 Unit の起動順序、起動時間などがわかる画像ファイル(SVG 形式)が生成されます。

[6]Generators は Unit の動的生成や Unit ファイルへのシンボリックリンクの生成を行います。

```
[armadillo ~]# systemd-analyze plot > /home/atmark/systemd-
analyze.prot.svg
```

この画像ファイルを ATDE で表示するには eog コマンドを実行します。なお、SVG 形式の画像ファイルは Firefox などの Web ブラウザでも表示することができます。

```
[ATDE ~]$ eog systemd-analyze.prot.svg
```

6.3. 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-600 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 6.3 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
/boot	起動に必要なファイル
/dev	デバイスファイル
/etc	設定ファイル
/home	ホームディレクトリ
/home/atmark	atmark ユーザーのホームディレクトリ
/lib	基本的なライブラリ
/media	CD-ROM や USB メモリなどのリムーバブルメディアのマウントポイント
/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
/opt	サードパーティが提供する追加データ・アプリケーション
/opt/license	Armadillo のライセンス情報
/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
/run	再起動後に保持されない揮発性のランタイムデータ

ディレクトリ	ディレクトリの内容
/sbin	基本的なシステム管理用コマンドの実行ファイル
/srv	システム上で運用されているサーバが使うデータを格納するディレクトリ
/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

6.3.1. 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。

6.3.2. 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-600 シリーズでは、atmark ディレクトリがあります。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。

6.3.3. 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[7]。

6.3.4. 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、Armadillo-640 では、/dev/ttymxc2 は CON3/CON4 のシリアルインターフェース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/ttymxc2 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。

6.3.5. 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。

[7]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。

6.3.6. 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。

6.3.7. 設定ファイル

設定ファイルは、/etc に置かれます。

7. 開発環境の構築と基本操作

7.1. ATDE (Atmark Techno Development Environment) とは

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ーです。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ー(ホストコンピューター)のアーキテクチャと、アプリケーションを実行するコンピューター(ターゲット)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インが必要です。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、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いるのではないでしょうか。

Debian GNU/Linux をベースに Armadillo に必要なクロス開発環境を構築し、仮想マシンのイメージとしたものが ATDE です。

仮想マシンとは

物理的なコンピューター上で「論理的なコンピューター」を動作させることを「(コンピューターの)仮想化」と言います。例えば、Windows PC 上で仮想化ソフトウェアを起動し、論理的なコンピューターを作成することで、Linux を使用したりすることができます。この論理的なコンピューターのことを一般的には「仮想マシン」や「VM(Virtual Machine)」といいます。

以降の説明で使う言葉

- ホスト OS

  物理的なコンピューターで動作する OS。本書では引き続き、物理的なコンピューターを「作業用 PC」と表記します。「ホスト OS」は作業用 PC に直接インストールされている Windows や Linux を指します。

- ゲスト OS

  仮想マシンで動作する OS。本書では ATDE がゲスト OS にあたるため、「ATDE」と表記します。

開発環境が仮想マシンであるメリット

仮想マシンの情報はすべてファイルとして管理されています。そのため、あるプロジェクトで使用した仮想マシンのファイルを保存しておけば、プロジェクト終了後に保守しなければいけない状況になった時でも、環境の再現が簡単にできます。

また、ファイルをコピーするだけで環境を複製することができます。複数の作業用 PC に環境を作成しなければならない場合に、一つの PC で環境を作成し、他の PC にファイルをコピーすることで、簡単に同じ環境を再現できます。

ATDE が動作する仮想化ソフトウェアは以下の二つです。

1. VMware Workstation 15 Player
   - 商用利用は有償です。
2. Oracle VM VirtualBox 6
   - Mac OS X にも対応しています。
   - VMware のイメージを起動することができます。
   - フリーソフトですが、商用利用で「Oracle VM VirtualBox Extension Pack」の機能を使用する場合は「Oracle VM VirtualBox Enterprise」のライセンス購入が必要です。[1]
     - インストールしない場合、USB3.0 以降に対応しているポートとデバイスの組み合わせではゲスト OS から認識不可となります。
     - ポートとデバイスのどちらかが USB2.0 までの対応の場合、USB1.1 デバイスとして認識されるようです。
   - Armadillo との通信のため、ネットワークアダプタの設定は「割り当て」を「ブリッジアダプター」に変更する必要があります。

本書では VMware を使用します。

7.2. ATDE のセットアップ

VMware 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。

必要な作業は、大きく分けると 3 つあります。. VMware のインストール. ATDE アーカイブの取得. ATDE アーカイブの展開

7.2.1. VMware Workstation 15 Player のインストール

ATDE を使用するために、まず作業用 PC に VMware をインストールします。商用利用の場合は、ライセンスを購入する必要があります。

下記リンクから購入・ダウンロードできます。

[1]個人利用は PUEL(VirtualBox Personal Use and Evaluation License)が適用されており、無料で使用可能

https://www.vmware.com/jp/products/workstation-player.html

VMware を起動後、購入したライセンスを入力してください。

作業用 PC が VMware の動作要件を満たしているか確認してください。なお、ATDE7 は 32bit OS です。

参考

- VMware Workstation Player for Windows

  https://docs.vmware.com/jp/VMware-Workstation-Player-for-Windows/15.0/com.vmware.player.win.using.doc/GUID-3CF87F1D-FD14-4FBA-A00C-F13D65825CA5.html

- VMware Workstation Player for Linux

  https://docs.vmware.com/jp/VMware-Workstation-Player-for-Linux/15.0/com.vmware.player.linux.using.doc/GUID-3CF87F1D-FD14-4FBA-A00C-F13D65825CA5.html

7.2.2. ATDE アーカイブの取得

Armadillo-640 用の対応バージョンである「ATDE7」のアーカイブを下記リンクからダウンロードしてください。

https://armadillo.atmark-techno.com/atde

7.2.3. ATDE アーカイブの展開

ATDE のアーカイブを展開します。ATDE のアーカイブは、tar.xz 形式の圧縮ファイルです。

Windows での展開方法を「7.2.3.1. Windows での展開」に、Linux での展開方法を手順「7.2.3.2. Linux での展開」に示します。

7.2.3.1. Windows での展開

1. 7-Zip のインストール

   7-Zip をインストールします。7-Zip は、圧縮解凍ソフト 7-Zip のサイト (http://sevenzip.sourceforge.jp)からダウンロード取得可能です。

2. xz 圧縮ファイルの展開

   エクスプローラーでダウンロードしたディレクトリを開きます。

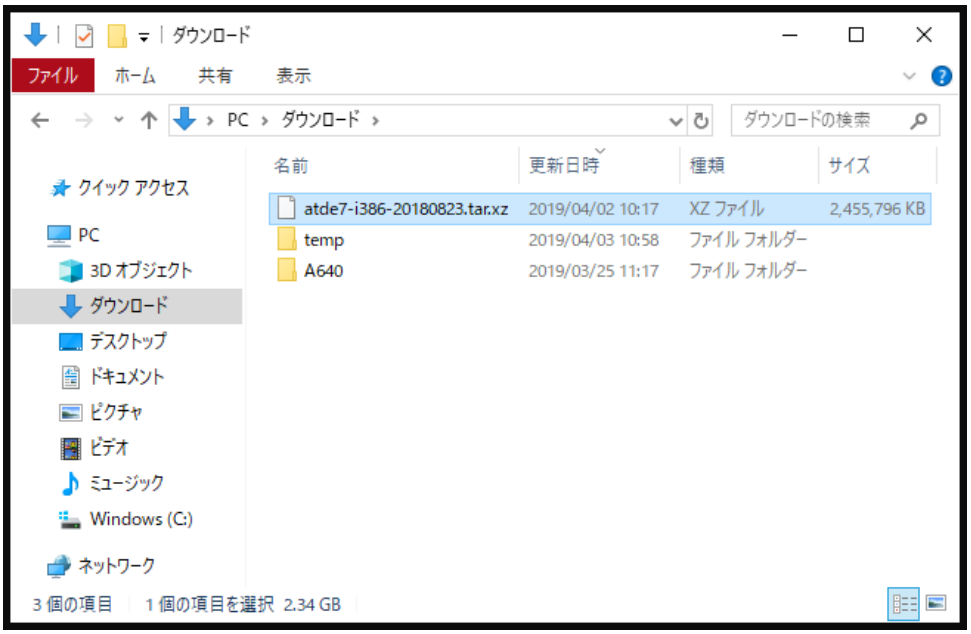

図 7.1 ダウンロードしたディレクトリ

ATDE のアーカイブを右クリックし、「7-Zip」-「ここに展開」をクリックします。

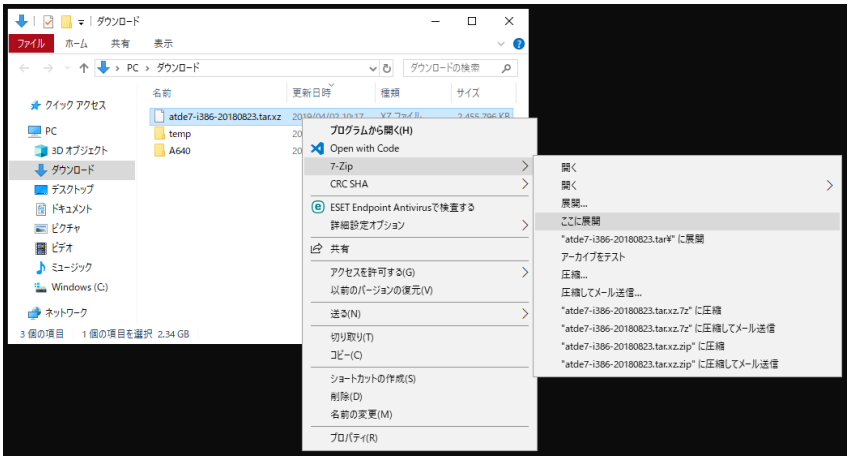

図 7.2 アーカイブの展開

展開してできた.tar ファイルを右クリックし、「7-Zip」-「ここに展開」をクリックします。画像のように **atde7-i386-[version]** ディレクトリができます。

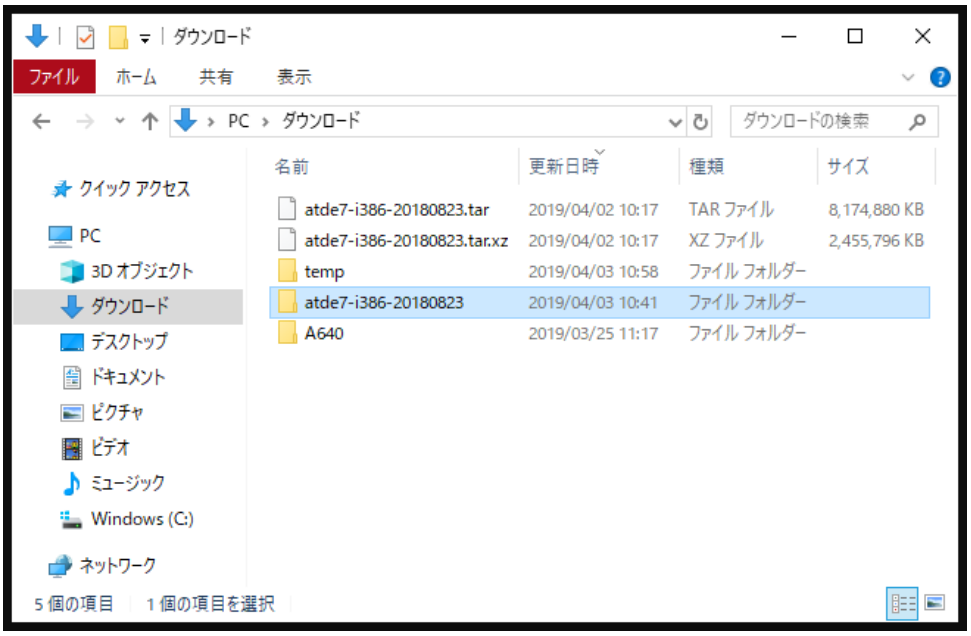

図 7.3 作成されたディレクトリ

ディレクトリの中には展開されたファイルがあります。

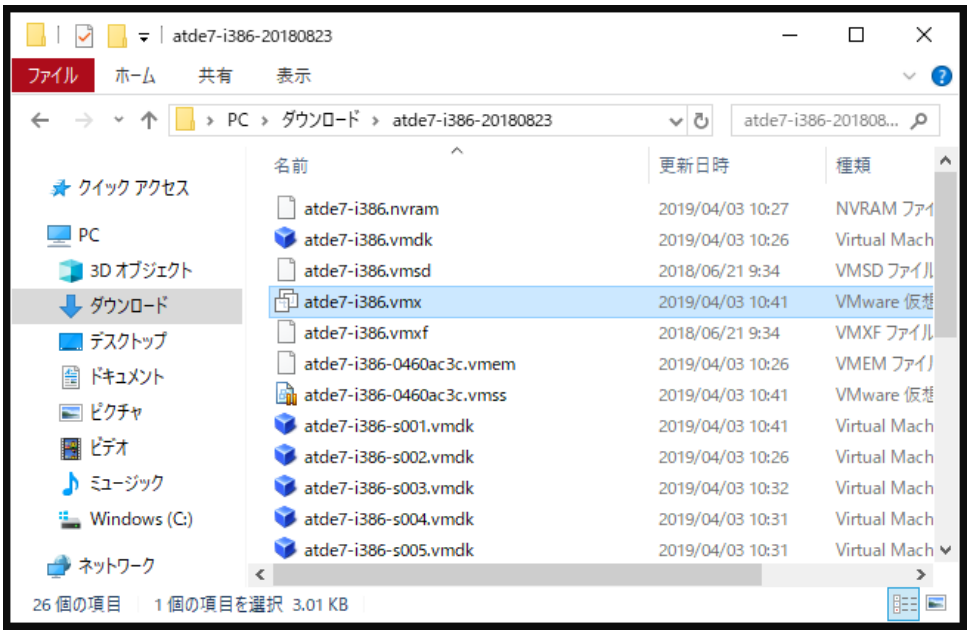

図 7.4 展開されたファイル

アーカイブファイル(.tar.xz、.tar)は削除しても構いません。

7.2.3.2. Linux での展開

1. tar.xz 圧縮ファイルの展開

   tar の xf オプションを使用して tar.xz 圧縮ファイルを展開します。

```
[PC ~]$ tar xf atde7-i386-[version].tar.xz
```

2. 展開の完了確認

   tar.xz 圧縮ファイルの展開が終了すると、ATDE アーカイブの展開は完了です。 **atde7-i386-[version]** ディレクトリに ATDE のデータイメージが出力されています。

```
[PC ~]$ ls atde7-i386-[version]/
atde7-i386-s001.vmdk  atde7-i386-s009.vmdk  atde7-i386-s017.vmdk
atde7-i386-s002.vmdk  atde7-i386-s010.vmdk  atde7-i386.nvram
atde7-i386-s003.vmdk  atde7-i386-s011.vmdk  atde7-i386.vmdk
atde7-i386-s004.vmdk  atde7-i386-s012.vmdk  atde7-i386.vmsd
atde7-i386-s005.vmdk  atde7-i386-s013.vmdk  atde7-i386.vmx
atde7-i386-s006.vmdk  atde7-i386-s014.vmdk  atde7-i386.vmxf
atde7-i386-s007.vmdk  atde7-i386-s015.vmdk
atde7-i386-s008.vmdk  atde7-i386-s016.vmdk
```

7.3. ATDE の起動

VMware のホーム画面で「仮想マシンを開く」を押し、先ほど展開した atde7-i386.vmx (仮想マシン構成)を開くと仮想マシンが追加されます。追加された仮想マシンをダブルクリックすると ATDE が起動します。下のダイアログが出た場合は、「コピーしました」を押してください。

図 7.5 VMware のダイアログ

VMware Tools のインストールを促すダイアログが出た場合は、「ダウンロードしてインストール」を押し、最新の VMware Tools をインストールしてください。

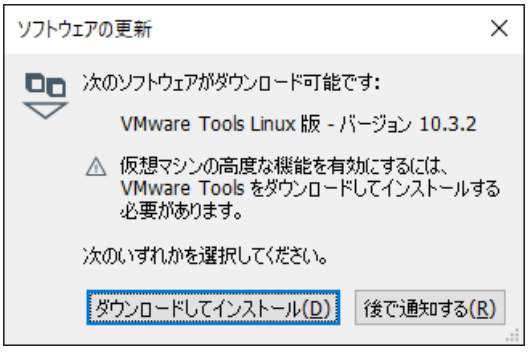

図 7.6 VMware のダイアログ

ATDE7 にログイン可能なユーザーを、「表 7.1. ユーザー名とパスワード」に示します [2]。

表 7.1 ユーザー名とパスワード

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

ATDE に割り当てるメモリおよびプロセッサ数を増やすことで、ATDE をより快適に使用することができます。仮想マシンのハードウェア設定の変更方法については、VMware 社 Web ページ (http://www.vmware.com/)から、使用している VMware のドキュメントなどを参照してください。

7.4. ATDE の終了

ATDE を終了する方法は 3 つあります。

- ATDE の電源オフ

  ATDE の通常の終了方法です。ATDE の更新時や、VM 構成の変更時に、シャットダウンが必要になることがあります。

- VM のサスペンド

  VM を一時停止します。次回起動時、作業していたままの状態から作業を再開できます。なお、VM 構成を変更したり、VM のファイルを移動したりした場合、再開できない場合があります。

- VM のパワーオフ

  VM を強制終了します。フリーズ等が発生した場合のみ使用してください。

それぞれの手順を説明します。

- ATDE の電源オフ

  ATDE の右上の電源マークをクリックし、吹き出しの右下の電源マークをクリックします。

[2]特権ユーザーで GUI ログインを行うことはできません

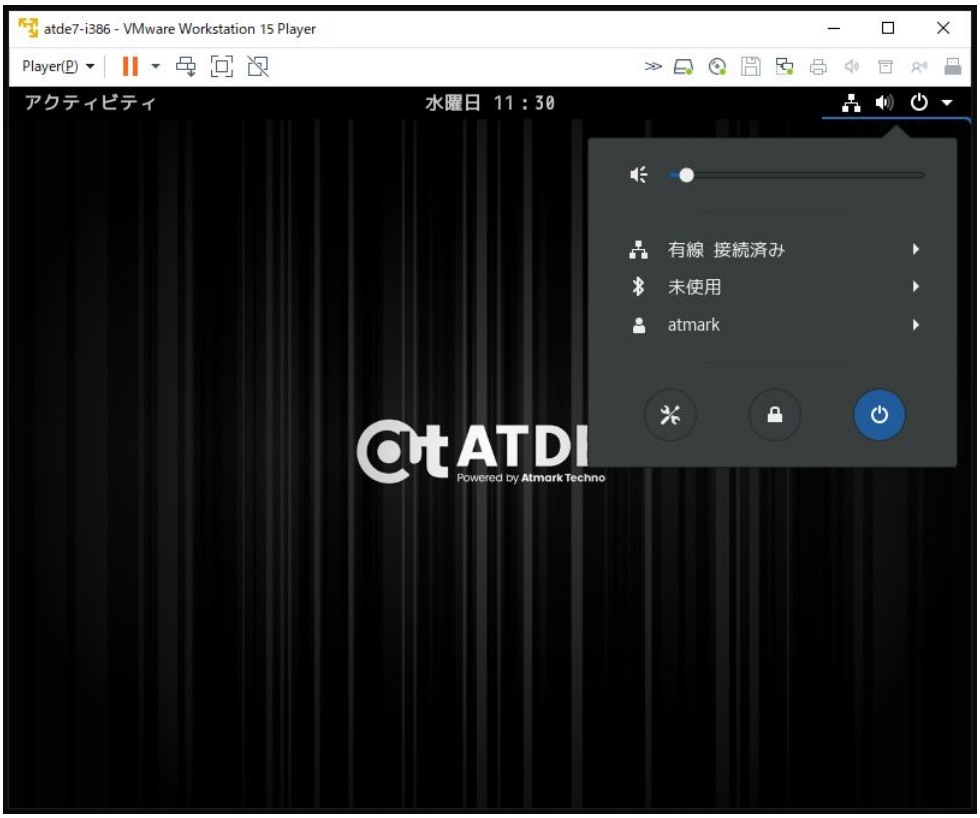

図 7.7 ATDE の電源オフ 1

「電源オフ」を選択すると、ATDE のシャットダウン処理が行われ、VM が終了します。「再起動」を選択すると、シャットダウン後にパワーオンします。

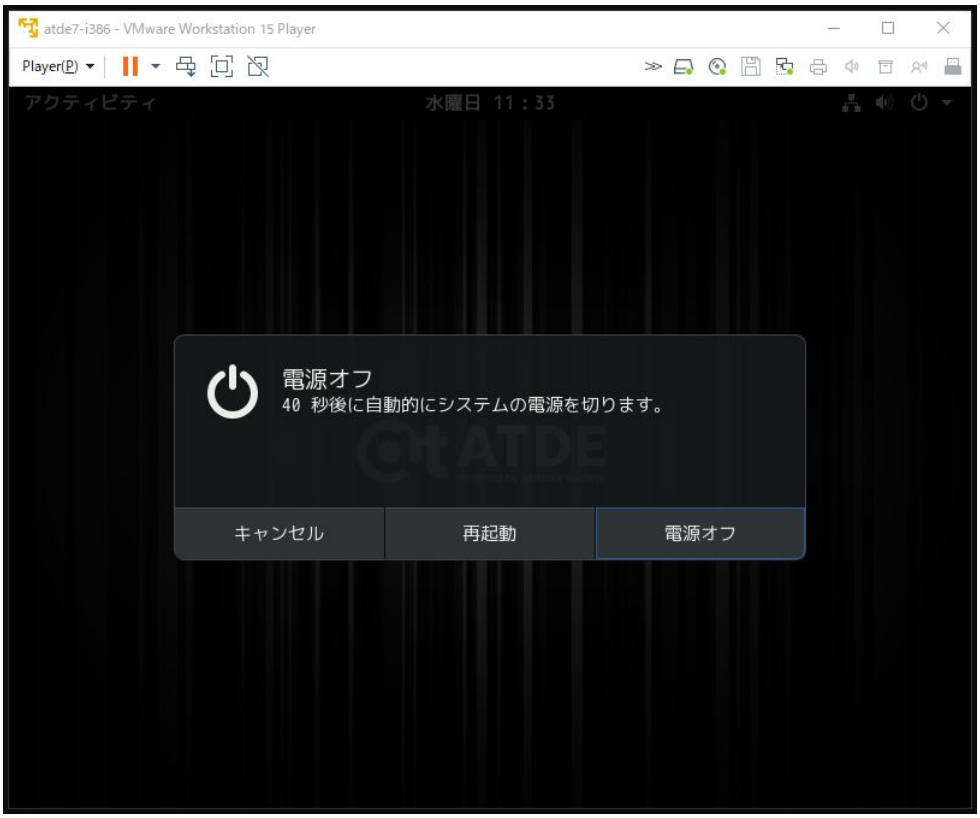

図 7.8 ATDE の電源オフ 2

- VM のサスペンド

  通常の Windows アプリケーション同様、VMware の右上の「×」ボタン(閉じる)を押すと「仮想マシンの終了方法を指定してください」と出ます。「サスペンド」を選択してください。

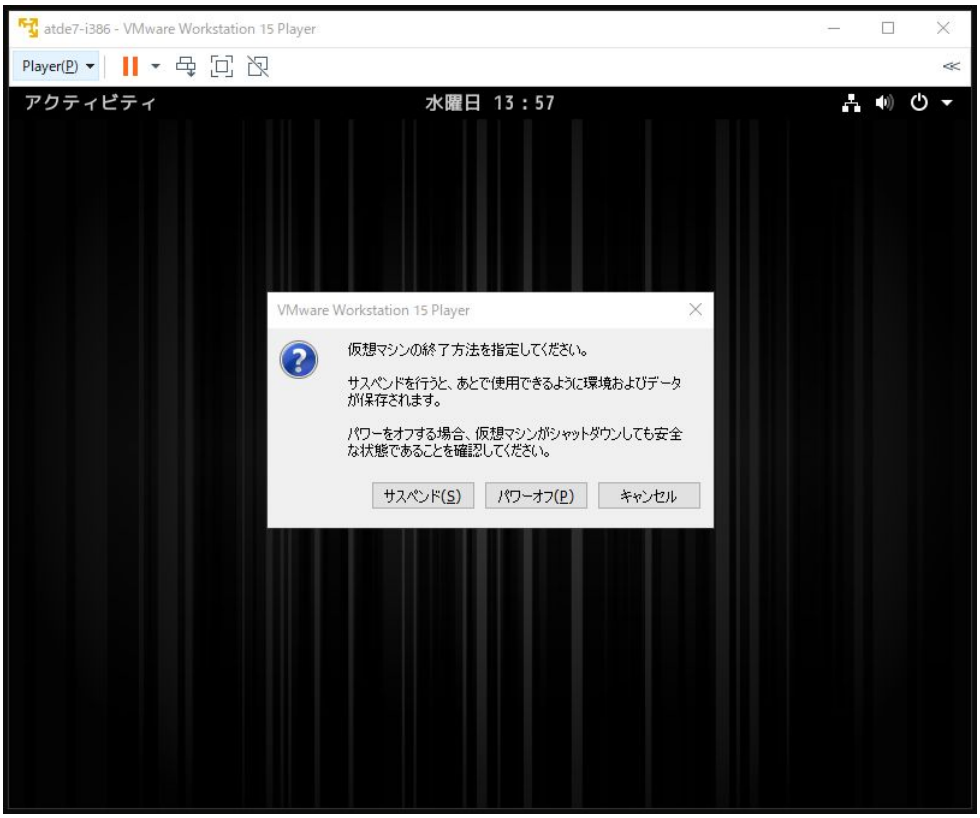

図 7.9 VM のサスペンド

・ VM のパワーオフ

「x」ボタン(閉じる)を押し、「パワーオフ」を選択してください(「電源ボタン長押し」に相当)。

7.5. 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 7.10. ATDE のネットワーク構成」のようになります。ホスト PC と ATDE 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]

[3] このような接続方法をブリッジ接続といいます。

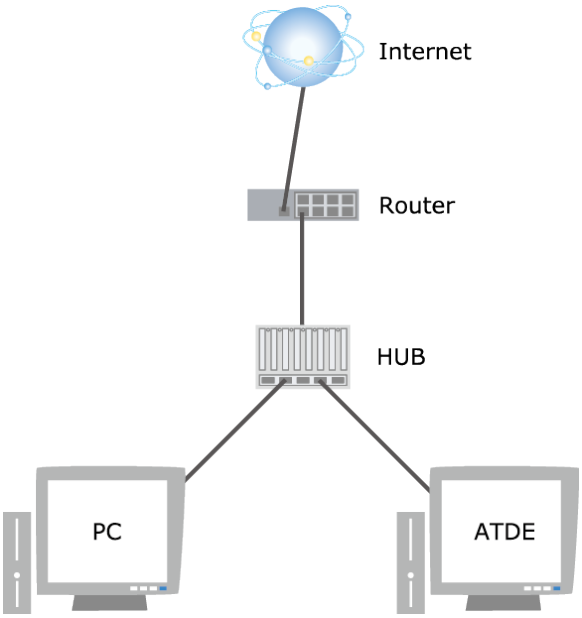

図 7.10 ATDE のネットワーク構成

7.5.1. 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。配布時は、DHCP を使用して IP アドレスを自動取得するようになっています。

1. ATDE 左上の「アクティビティ」から「net」と入力し「Network」を起動してください。

図 7.11 Network を起動

2. 現在のネットワーク設定情報が表示されます。

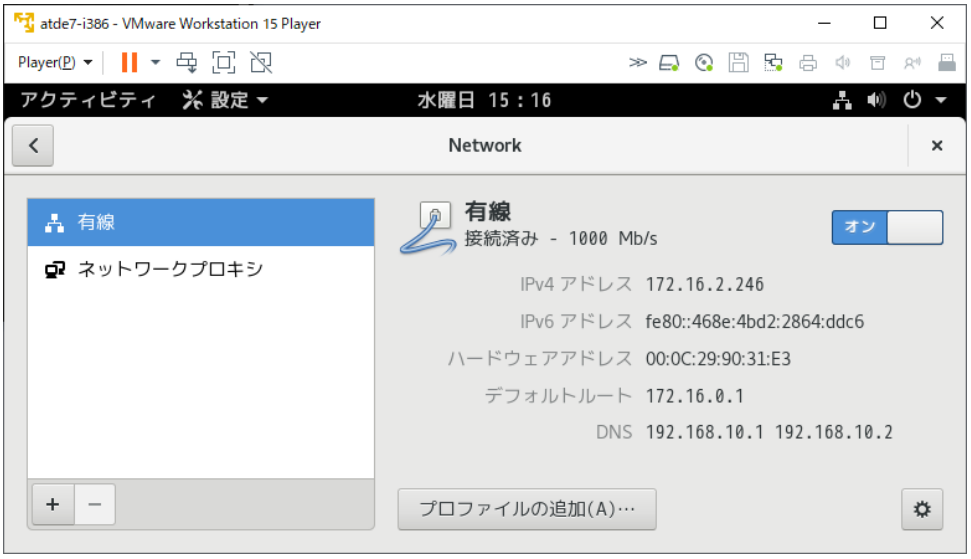

図 7.12 Network

3. 右下の歯車アイコンをクリックすると、有線ネットワーク設定の画面が開きます。

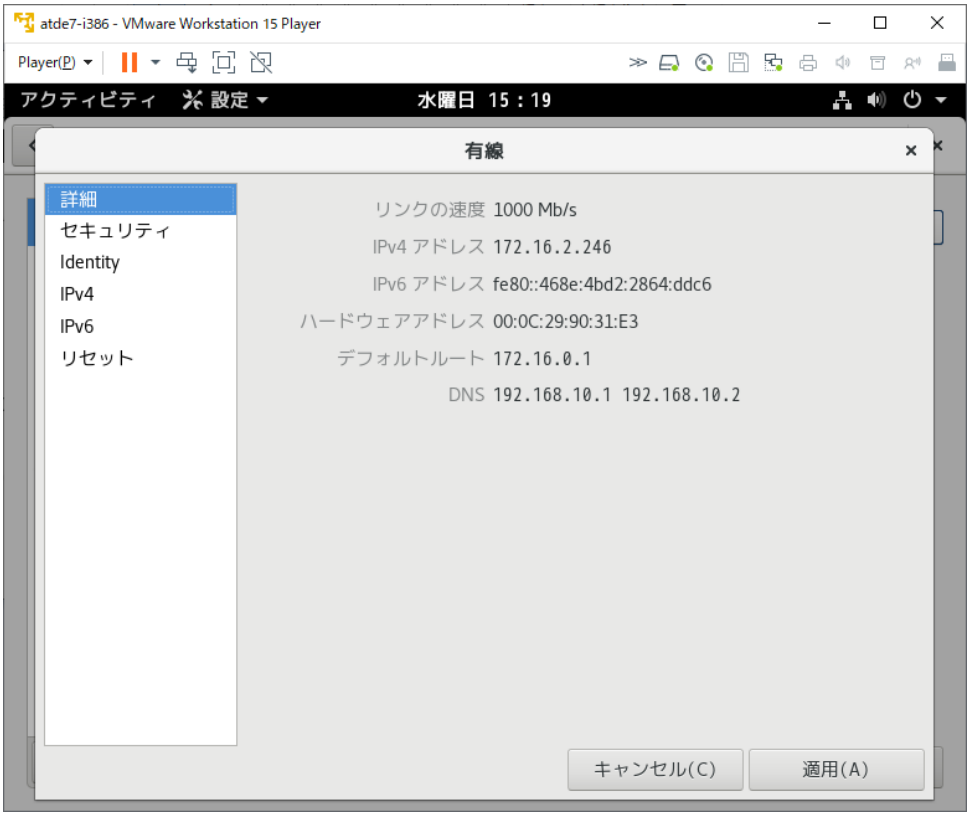

図 7.13 有線ネットワーク設定

4. ここでは例として、IPv4 のネットワーク設定を「表 7.2. ネットワーク設定例」の値に設定します。

表 7.2 ネットワーク設定例

項目	設定
IP アドレス	192.168.0.10
ネットマスク	24 (255.255.255.0)
ゲートウェイ	192.168.0.1
DNS サーバー	192.168.0.2

以下のように設定してください。なお、この画面で「アドレス(A)」の選択を「自動（DHCP)」にすると DHCP 設定に戻ります。

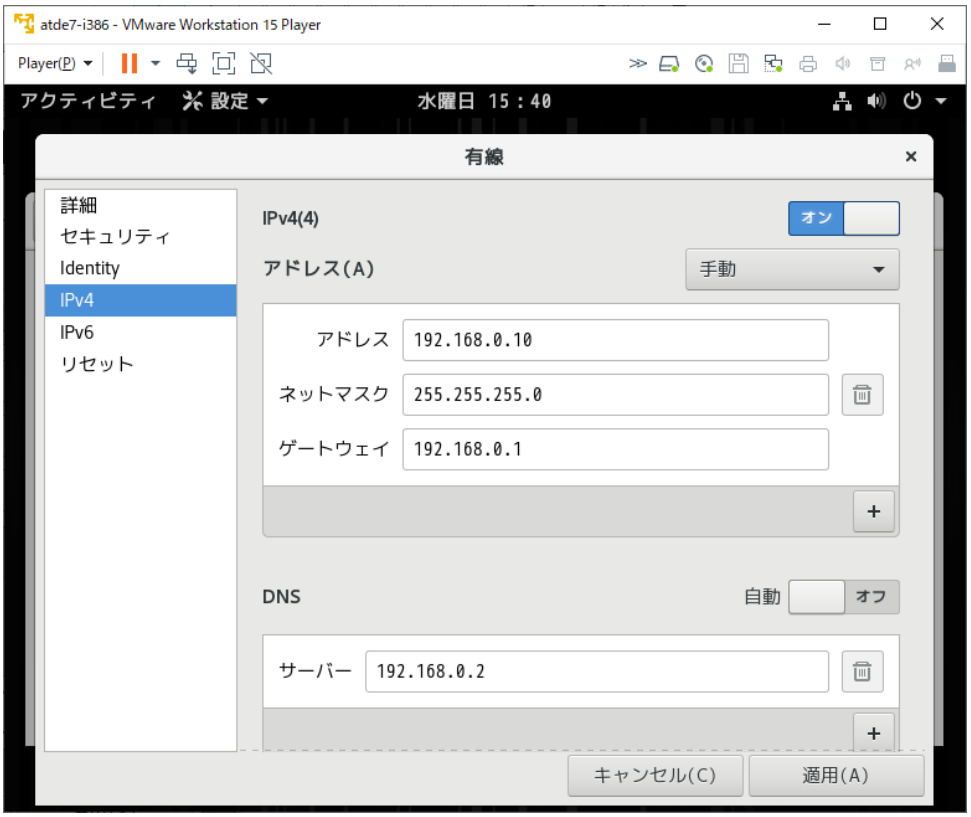

図 7.14 IPv4 のネットワーク設定

5. 「適用」をクリックして有線ネットワーク設定の画面を閉じます。Network の画面で「有線」を(「オン」の場合は一旦「オフ」し)「オン」にすると新しいネットワーク設定が反映されます。

6. 以上の手順でネットワーク設定は完了です。Network の画面を閉じてください。

7.6. ATDE を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
```

7.7. 取り外し可能デバイスの使用

VMware は取り外し可能デバイス(USB デバイス等)をサポートしており、ATDE で USB メモリー、USB 接続の SD カードリーダーや、USB シリアル変換ケーブル等が使用できます。多くのデバイスは、ホスト OS と ATDE で同時に使用することができません。ATDE で使用するためには、ATDE にデバイスを接続する操作が必要になります。

VMware が起動しているときに PC に USB デバイスを接続すると、ホスト OS と ATDE のどちらでデバイスを使用するかを選択するダイアログが出ます。ATDE で使用する場合、「仮想マシンに接続」を選択してください。

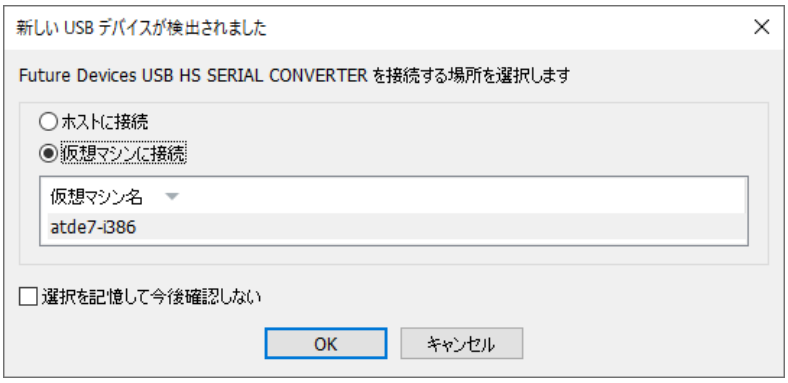

図 7.15 USB デバイス接続時のダイアログ

すでに PC に接続されているデバイスを ATDE で使用する場合、メニューバーのアイコンを右クリックし、「接続（ホストから切断)」をクリックしてください。

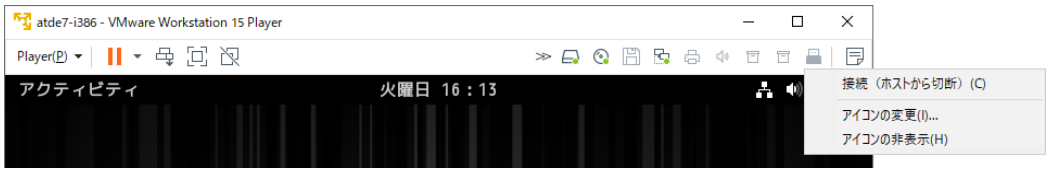

図 7.16 ATDE と USB デバイスの接続

USB シリアル変換ケーブルは認識されると /dev/ttyUSB0 というデバイスファイルが作られます。ATDE には minicom があらかじめインストールされているため、「5.1.5. Linux PC とのシリアル通信」と同様の手順で Armadillo のコンソールを操作できます。

図 7.17 USB シリアル変換ケーブル

USB メモリーは接続できると自動的にマウントされ、 /media/atmark/ 以下にディレクトリが作られます。

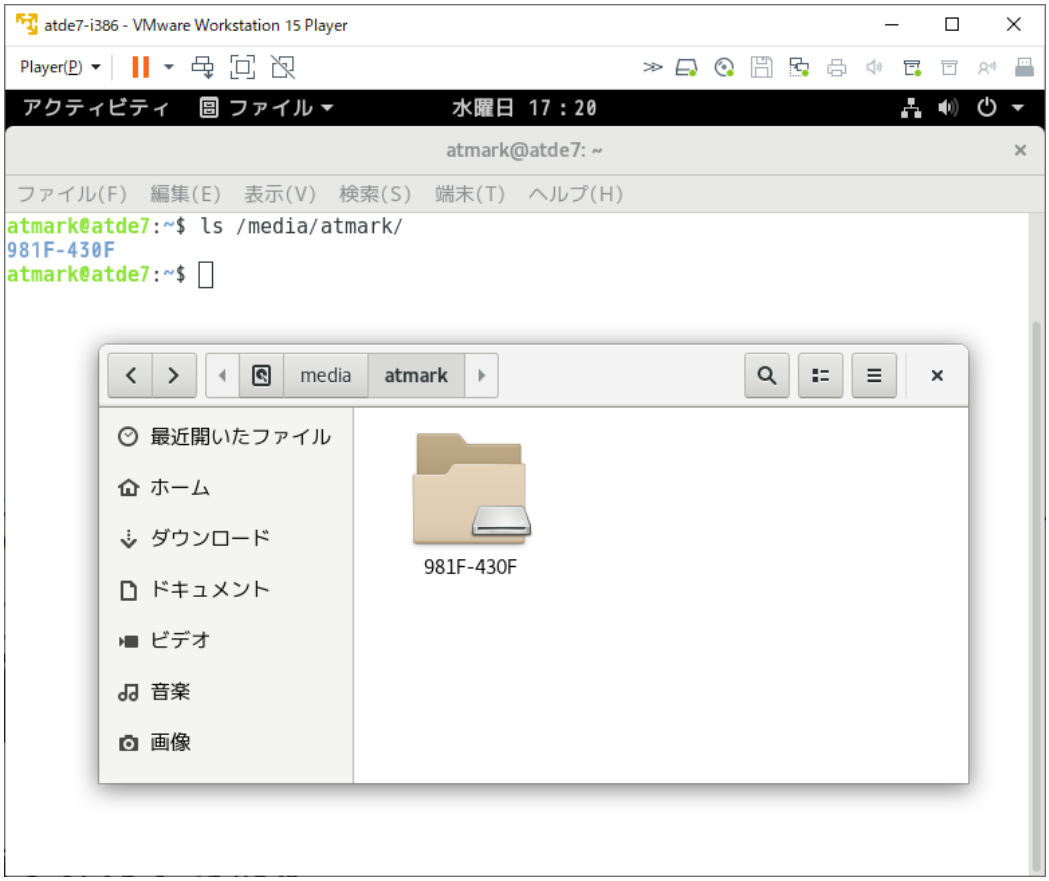

図 7.18 USB メモリー

7.8. 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。

1. ホスト OS に、共有するフォルダをあらかじめ作成(今回は C:\vmshare)
2. 仮想マシン設定で共有フォルダの設定を開く
3. 「フォルダの共有」を「常に有効」にする
4. フォルダの「追加」で名前と共有するフォルダのパスを入力

設定すると画像のようになるので、「OK」で閉じます。

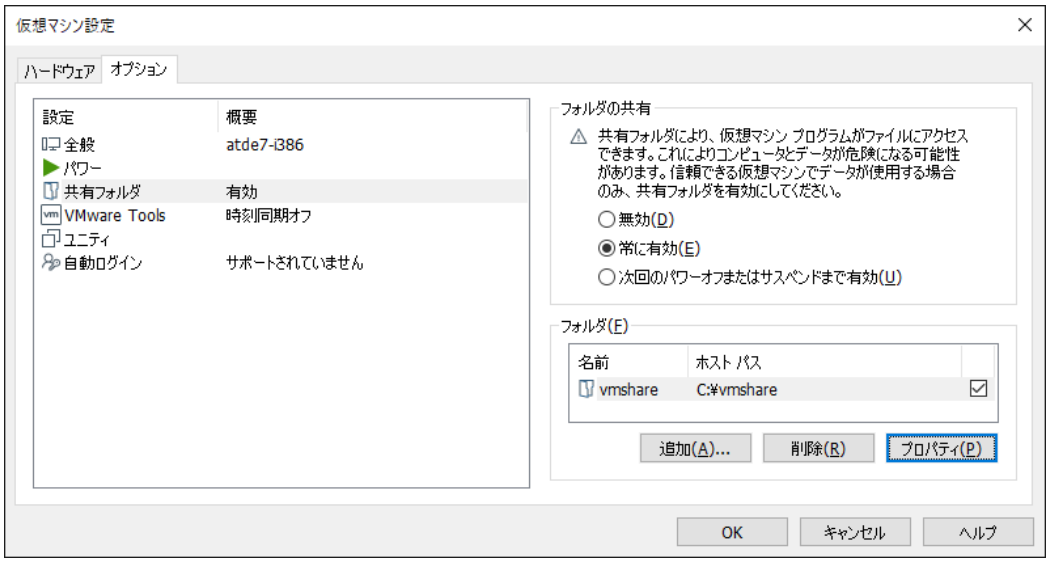

図 7.19 共有フォルダの設定

共有フォルダーは自動的にマウントされ、 /mnt/hgfs/ 以下にディレクトリが作られます。

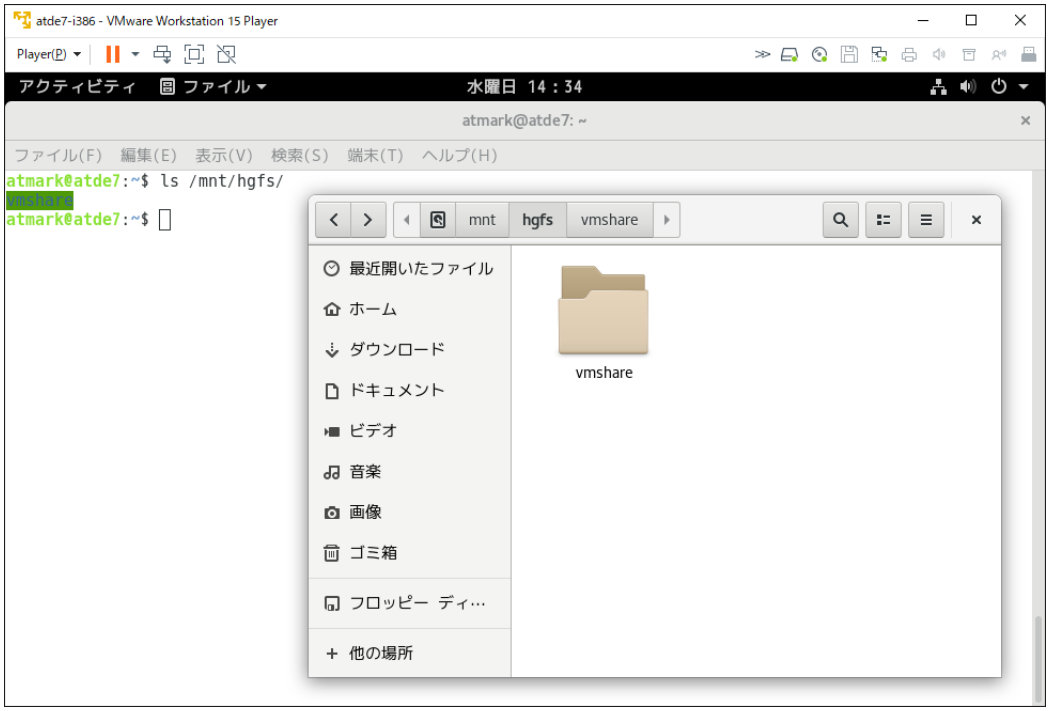

図 7.20 作成された共有フォルダ

8. 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. Debian GNU/Linux ルートファイルシステムアーカイブを作成する
3. 量産に向けた準備をする
4. 製品出荷後のメンテナンスをする

まずは、製品の機能を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、Debian ユーザーランドのルートファイルシステムアーカイブを作成する方法について説明します。ルートファイルシステムアーカイブに、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も、ここで紹介します。

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、各手順の概要的な説明のみをおこないます。

8.1. アプリケーションプログラムの作成

本章では、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、まずホストである作業用 PC でプログラムのコンパイルと実行をおこなう方法を説明します。その後、作業用 PC でターゲットとなる Armadillo 用にプログラムをコンパイルし、Armadillo で実行する方法について説明します。

8.1.1. 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、実行してみます。

以下に示す、「図 8.1. 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 8.1 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 8.2 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。

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

図 8.3 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。

[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。

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-gnueabihf-gcc hello.c -o hello
```

図 8.4 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-gnueabihf-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
/lib/ld-linux-armhf.so.3: No such file or directory
```

図 8.5 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「80386」と表示されます。

```
[ATDE ~]$ file hello
hello: ELF 32-bit LSB shared object, Intel 80386, version 1 (SYSV),
dynamically
linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.32,
BuildID[sha1]=84cc
e45ccca0ec56f0cbd2b36b299bdd484a3d94, 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 shared object, ARM, EABI5 version 1 (SYSV),
dynamically li
nked, interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 3.2.0,
BuildID[sha1]=f
e0cc3af441ae9b586159ad8a9ced593a3c1c506, not stripped
```

ターゲット上で実行するために、Armadillo に実行ファイルをコピーします。Armadillo に実行ファイルをコピーする方法には様々なものがありますが、ネットワーク(Ethernet)経由で転送する方法が手間が少なくて良いでしょう。

ここでは、scp コマンドを使用することにします。

標準状態の Armadillo では SSH Server が動作していないので、まずは Armadillo に openssh-server をインストールします。

```
[armadillo ~]# apt-get update && apt-get install openssh-server
```

図 8.6 openssh-server のインストール

次に scp コマンドを実行して、Armadillo にファイルを転送します。

scp では、リモートマシン上のファイルを次のように表現します。

```
”ユーザー名@IP アドレス:/ディレクトリ名/ディレクトリ名/.../ファイル名”
```

今回はローカルマシン上の "hello" というファイルを、リモートマシン上の atmark ユーザーのホームディレクトリ(/home/atmark/以下)にコピーすることにします。

```
[ATDE ~]$ scp ./hello atmark@<Armadillo の IP アドレス>:~/
The authenticity of host '192.168.0.2 (192.168.0.2)' can't be established.
ECDSA key fingerprint is SHA256:lrhNjYNWnytCglh881CuT5Vlgwmlq/G4UiqVDfTvohE.
Are you sure you want to continue connecting (yes/no)? yes ❶
Warning: Permanently added '192.168.0.2' (ECDSA) to the list of known hosts.
atmark@<Armadillo の IP アドレス>'s password: ❷
hello                                         100% 8376     1.6MB/s   00:00
```

図 8.7 Armadillo へのファイル転送

❶ "yes" と入力

❷ Armadillo の atmark ユーザーのパスワードを入力[2]

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、sc と入力すると、「sc」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)`sc': scp ./hello atmark@172.16.2.156:~/
```

[2]デフォルトでは "atmark"

転送した実行ファイルを Armadillo で実行してみます。

```
[armadillo ~]# /home/atmark/hello
Hello World!
```

図 8.8 クロスコンパイルした hello の実行結果

Linux でプログラムを実行しようとした時、「Permission denied」というエラーが表示されることがあります。これは、実行ファイルに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/atmark/hello
```

図 8.9 実行権限を付与する

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。

8.1.2. 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。

必要なヘッダファイルの名前や、共有ライブラリのファイル名がわかっている場合は、Debian プロジェクトサイトの「パッケージの内容を検索」からファイルの含まれるパッケージの名前を探す事ができます。

Debian — パッケージ パッケージの内容を検索

https://www.debian.org/distrib/packages#search_contents

また、パッケージの部分的な名前が分っている場合は `apt-cache search` コマンドを使って必要なパッケージを探す事もできます。

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 型の引数を一つとり、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 8.10 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 8.11 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/ccQ0sxun.o: 関数 `main' 内:
sin.c:(.text+0x30): `sin' に対する定義されていない参照です
collect2: error: ld returned 1 exit status
```

図 8.12 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 8.13 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 8.14 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-gnueabihf-gcc sin.c -lm -o sin
```

図 8.15 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 scp で転送し、実行結果を確認してください。

```
[armadillo ~]# /home/atmark/sin
sin(0.5) = 0.479426
```

図 8.16 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-gnueabihf-gcc を用いた場合のインクルードパスは、/usr/arm-linux-gnueabihf/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、/lib/i386-linux-gnu ディレクトリにあります。算術演算ライブラリの場合、/lib/i386-linux-gnu/libm.so.6[3]です。

ARM 用のライブラリは、/usr/arm-linux-gnueabihf/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、実行ファイルだけでなく、ライブラリファイルもターゲット上の適切なパスに配置されている必要があります。

Armadillo-640 で Debian から供給されるライブラリを利用する場合、基本的にはライブラリファイルの配置場所を気にする必要はありません。

[3]/lib/i386-linux-gnu/libm.so.6 は/lib/i386-linux-gnu/libm-2.24.so へのシンボリックであり、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。

Armadillo-640 および ATDE7 に同名の Debian パッケージをインストールすることで、自動的にライブラリファイルが適切なパスに配置されます。

例えば、libm の場合は、次のように Debian パッケージを Armadillo-640、ATDE7 にインストールします。

```
[armadillo ~]# apt-get update && apt-get install libc6 ❶
```

❶ Armadillo-640 に libc6 (ライブラリ本体)をインストール

```
[ATDE ~]$ sudo apt-get update
[ATDE ~]$ sudo apt-get install libc6 ❶
[ATDE ~]$ sudo apt-get install libc6-dev ❷
[ATDE ~]$ sudo apt-get install libc6-dev-armhf-cross ❸
```

❶ ATDE7 に libc6 (ライブラリ本体)をインストール

❷ ATDE7 に libc6-dev (ネイティブ開発用パッケージ)をインストール

❸ ATDE7 に libc6-dev-armhf-cross (クロス開発用パッケージ)をインストール

libc6 は、システム上のほぼ全てのプログラムが使用するため、すでに Armadillo-640 および ATDE7 にはインストール済みとなっています。そのため、今回の例ではライブラリをインストールするという手順は省略しています。

8.1.3. 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と、それに従って次に行うべき手順を見つけ出し、必要なコマンド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 8.17 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。

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 8.18 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 GNUmakefile、makefile、Makefile という名前のファイルを順番に検索し、最初に見つけたファイルを makefile として認識します。

makefile を認識後、make は makefile で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 8.18. 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: sin.o」というルールは、「sin を生成するには、sin.o が必要」という意味になります。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 $@」が実行されます。$^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:5:14: warning: unused parameter ‘argc’ [-Wunused-parameter]
 int main(int argc, char *argv[])

sin.c:5:26: warning: unused parameter ‘argv’ [-Wunused-parameter]
 int main(int argc, char *argv[])^~~~

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

図 8.19 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 8.20 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 8.18. 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。

```
[ATDE ~]$ ls
Makefile  sin  sin.c  sin.o
```

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

図 8.21 make clean 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-gnueabihf-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-gnueabihf

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 8.22 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-gnueabihf-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。

8.2. Debian GNU/Linux ルートファイルシステムアーカイブの作成

Armadillo を一度起動した後にルートファイルシステム上には、使い方によっては ssh の秘密鍵や、動作ログ、シェルのコマンド履歴、ハードウェアの UUID に紐付く設定ファイル等が生成されています。そのまま、他の Armadillo にルートファイルシステムをコピーした場合は、鍵の流出や UUID の不一致による動作の相違が起きる可能性があります。そのため、量産等に使用するルートファイルシステムは新規に at-debian-builder を使って構築します。

本章では、at-debian-builder を使用して Debian GNU/Linux ルートファイルシステムアーカイブを作成する方法と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について説明します。

8.2.1. at-debian-builder の取得

at-debian-builder は、以下の URL からダウンロードすることができます。

> at-debian-builder のダウンロード URL
>
> https://download.atmark-techno.com/armadillo-640/at-debian-builder/

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 8.1 使用する at-debian-builder のバージョン

対象	バージョン	ダウンロード URL
at-debian-builder	v1.2.0	https://download.atmark-techno.com/armadillo-640/at-debian-builder/at-debian-builder-v1.2.0.tar.gz

ファイルの取得には wget コマンドを使います。

```
[ATDE ~]$ wget https://download.atmark-techno.com/armadillo-640/at-debian-builder/at-debian-
builder-v1.2.0.tar.gz
[ATDE ~]$ ls
at-debian-builder-v1.2.0.tar.gz
```

図 8.23 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 at-debian-builder-v1.2.0.tar.gz
[ATDE ~]$ ls
at-debian-builder-v1.2.0  at-debian-builder-v1.2.0.tar.gz
[ATDE ~/at-debian-builder-v1.2.0]$ ls
a600_resources  aiotg3_resources  aiotg3l_resources  ax1_resources  build.sh
```

図 8.24 アーカイブの展開

これで、ルートファイルシステムアーカイブを作成する準備が整いました。

ここで build.sh を実行すると、標準状態のルートファイルシステムアーカイブを作成することができますが、今は実行しません。

8.2.2. ルートファイルシステムのカスタマイズ

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ルートファイルシステムアーカイブに追加する方法を紹介します。

at-debian-builder では、製品名_resouces 内のファイルを変更し、build.sh を実行することで、ルートファイルシステムをカスタマイズすることができます。Armadillo-640 の場合は、a600_resources 内のファイルを修正していきます。

8.2.2.1. ファイルやディレクトリを追加する

a600_resources 以下にファイルやディレクトリを追加すると、そのファイルやディレクトリはそのままルートファイルシステムに配置されます。a600_resources ディレクトリはルートファイルシステム上のルートディレクトリ(/)に対応しており、a600_resources ディレクトリ直下に配置されたファイルは、ルートファイルシステム上のルートディレクトリ直下に配置されます。

ただし、最初から a600_resources 以下に配置されている resources ディレクトリは例外で、ルートファイルシステム上に配置されません。resources ディレクトリ以下には、ルートファイルシステムアーカイブの作成の際に使用するスクリプトやインストールするパッケージ情報などが格納されています。

例として、「8.1. アプリケーションプログラムの作成」の「8.1.2. ライブラリとヘッダファイル」で作成した、アプリケーションプログラム sin を追加します。

ルートファイルシステム上の /usr/local/bin ディレクトリに sin を配置する場合は、次のようにします。

```
[ATDE ~/at-debian-builder-v1.2.0]$ ls a600_resources
resources
[ATDE ~/at-debian-builder-v1.2.0]$ mkdir -p a600_resources/usr/local/bin
[ATDE ~/at-debian-builder-v1.2.0]$ cp ~/sin a600_resources/usr/local/bin
[ATDE ~/at-debian-builder-v1.2.0]$ ls a600_resources/usr/local/bin
sin
```

図 8.25 /usr/local/bin へのアプリケーションプログラムの追加

mkdir コマンドを -p オプションを付けて実行すると、dir1/dir2/dir3 のような複数の階層からなるディレクトリをまとめて作成することができます。

```
[ATDE ~]$ mkdir -p dir1/dir2/dir3
[ATDE ~]$ ls dir1
dir2
[ATDE ~]$ ls dir1/dir2
dir3
```

8.2.2.2. Debian パッケージを追加する

Debian パッケージを追加するには、a600_resources/resources/packages ファイルに Debian パッケージの名称を追記します。パッケージ名は 1 行に 1 つ書くことができます。パッケージ名は、Armadillo で動作確認をする際に"apt-get install"の引数に与えた正しい名前で記載します。

例えば、libc6 の場合は

```
[armadillo ~]# apt-get update && apt-get install libc6
```

のようにしてインストールするので、次のように追記します。

```
[ATDE ~/at-debian-builder-v1.2.0]$ vi a600_resources/resources/packages
# List of packaes will install in the debootstrap chroot environment.
# A line start with # is comment. Separate packages name with new-line or space.

#Init
systemd
systemd-sysv

#Utils
sudo
locales
sed
less
expect
cu
vim-tiny
dosfstools
util-linux
rsyslog
bash-completion
man
dialog

#Device
i2c-tools
evtest

#Network
iptables
net-tools
wget
iputils-ping
iputils-arping
dnsutils
openssh-client
ifupdown
iptables-persistent

#Base Libs
libglib2.0-0
```

```
#sin Libs ❶
libc6 ❷
```

図 8.26 a600_resources/resources/packages の修正

❶ "#" で始まる行はコメントとして認識されます。

❷ libc6 を追記します。

誤ったパッケージ名を指定した場合は、ビルドログに以下のようなエラーメッセージが表示されて当該のパッケージが含まれないアーカイブが生成されてしまいます。例えば、libc6 を追記する際に、誤って libc としてしまった場合、ビルドログに次のエラーメッセージが表示されます。

```
E: Unable to locate package libc
```

図 8.27 誤ったパッケージ名を指定した場合に起きるエラーメッセージ

パッケージに依存する他のパッケージは明記しなくても、apt によって自動的にインストールされます。また、apt や dpkg 等の Debian GNU/Linux の根幹となるパッケージも自動的にインストールされます。

例えば、「8.1. アプリケーションプログラムの作成」の「8.1.2. ライブラリとヘッダファイル」で作成した sin で利用している libm は libc6 に含まれていますが、libc6 は明記しなくてもインストールされます。[5]

openssh-server のような「パッケージのインストールの際に、自動的に秘密鍵を生成する」パッケージは、基本的に packages には追加せず、Armadillo を起動した後に "apt-get install" を使って個別にインストールしてください。

openssh-server を packages に追加した場合、構築したルートファイルシステムアーカイブを書き込んだ全ての Armadillo に、単一の公開鍵を使ってログインすることができてしまいます。もし、意図的に、複数の Armadillo で同一の秘密鍵を利用したい場合、脆弱性となり得ることを理解して適切な対策をとった上で利用してください。

8.2.3. ルートファイルシステムアーカイブのビルド

ルートファイルシステムアーカイブを作成するには、次のコマンドを実行します。

```
[ATDE ~/at-debian-builder-v1.2.0]$ sudo ./build.sh a600
```

図 8.28 ルートファイルシステムアーカイブのビルド

[5]libm6 は util-linux, bash, wget など様々なパッケージから利用されています。

生成されたルートファイルシステムアーカイブは「8.3. イメージファイルの更新」で使用するので、わかりやすいように ~/images ディレクトリを作成して、その中に移動しておきます。

```
[ATDE ~/at-debian-builder-v1.2.0]$ mkdir -p ~/images
[ATDE ~/at-debian-builder-v1.2.0]$ ls debian-stretch*
debian-stretch-armhf-a600-20190410.tar.gz ❶
[ATDE ~/at-debian-builder-v1.2.0]$ mv debian-stretch* ~/images
[ATDE ~/at-debian-builder-v1.2.0]$ ls ~/images
debian-stretch-armhf-a600-20190410.tar.gz
```

❶ at-debian-builder で作成されたルートファイルシステムアーカイブのファイル名は debian-stretch-armhf-a600-作成日.tar.gz となります。

8.3. イメージファイルの更新

前章ではルートファイルシステムアーカイブを作成しました。ここでは、そのルートファイルシステムアーカイブを Armadillo に書き込み、正しく動作するか確認していきます。

8.3.1. インストールディスクの作成

まずは、ルートファイルシステムアーカイブを Armadillo に書き込むために、インストールディスクを作成します。

インストールディスクとは、Armadillo の動作に必要なソフトウェアを一挙に更新することができるブートディスク footnote[Armadillo-640 では特定のイメージが書き込まれた microSD カード]のことです。

インストールディスクは make-install-disk-image を使って作成できます。

8.3.1.1. make-install-disk-image の取得

make-install-disk-image は、以下の URL からダウンロードすることができます。

make-install-disk-image のダウンロード URL

https://download.atmark-techno.com/armadillo-640/make-install-disk-image/

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 8.2 使用する make-install-disk-image のバージョン

対象	バージョン	ダウンロード URL
make-install-disk-image	1.0.0	https://download.atmark-techno.com/armadillo-640/make-install-disk-image/make-install-disk-image-v1.0.0.tar.gz

wget コマンドでアーカイブを取得し、tar コマンドでアーカイブを展開します。

```
[ATDE ~]$ wget https://download.atmark-techno.com/armadillo-640/make-install-disk-image/make-
install-disk-image-v1.0.0.tar.gz
[ATDE ~]$ tar xzvf make-install-disk-image-v1.0.0.tar.gz
```

8.3.1.2. Armadillo に書き込むイメージファイルの準備

インストールディスク作成する前に、Armadillo に書き込むイメージファイルの準備をします。必要なのは次の 4 つのファイルです。

- ブートローダーイメージ
- Linux カーネルイメージ
- DTB(Device Tree Blob)
- ルートファイルシステムアーカイブ

ルートファイルシステムアーカイブは「8.2.3. ルートファイルシステムアーカイブのビルド」で作成しました。他のファイルついても、ユーザーがカスタマイズして作成することができますが、今回は"Armadillo サイト"でダウンロードすることができる標準イメージを使用します。

使用するイメージファイルは次のとおりです。

表 8.3 使用するイメージファイル

対象	バージョン	ダウンロード URL
ブートローダーイメージ	v4	https://download.atmark-techno.com/armadillo-640/image/u-boot-a600-v2018.03-at4.imx
Linux カーネルイメージ	v11	https://download.atmark-techno.com/armadillo-640/image/uImage-a600-v4.14-at11
DTB(Device Tree Blob)	v11	https://download.atmark-techno.com/armadillo-640/image/armadillo-640-v4.14-at11.dtb
ルートファイルシステムアーカイブ	v20190326 ベース	なし(「8.2.3. ルートファイルシステムアーカイブのビルド」で作成)

~/images ディレクトリに移動してから、イメージファイルをダウンロードします。

```
[ATDE ~]$ cd ~/images
[ATDE ~/images]$ wget https://download.atmark-techno.com/armadillo-640/image/u-boot-a600-v2018.03-at4.imx
[ATDE ~/images]$ wget https://download.atmark-techno.com/armadillo-640/image/uImage-a600-v4.14-at11
[ATDE ~/images]$ wget https://download.atmark-techno.com/armadillo-640/image/armadillo-640-v4.14-at11.dtb
[ATDE ~/images]$ ls
armadillo-640-v4.14-at11.dtb               u-boot-a600-v2018.03-at4.imx
debian-stretch-armhf-a600-20190410.tar.gz  uImage-a600-v4.14-at11
```

8.3.1.3. インストールディスクイメージのビルド

いよいよ make-install-disk-image を使ってインストールディスクイメージを作成します。

まずは make-install-disk-image-v1.0.0 ディレクトリに移動します。make-install-disk-image-v1.0.0 ディレクトリの中には、build.sh というスクリプトと core ディレクトリがあります。

```
[ATDE ~]$ cd ~/make-install-disk-image-v1.0.0
[ATDE ~/make-install-disk-image-v1.0.0]$ ls
build.sh  core
```

次のようにして build.sh を実行するとインストールディスクイメージを作成することができます。ビルドに成功すると "install-disk-sd-a640-日付.img" が生成されます。

```
[ATDE ~/make-install-disk-image-v1.0.0]$ sudo ./build.sh \
a640 \
~/images/u-boot-a600-v2018.03-at4.imx \
~/images/uImage-a600-v4.14-at11 \
~/images/armadillo-640-v4.14-at11.dtb \
~/images/debian-stretch-armhf-a600-20190410.tar.gz
[ATDE ~/make-install-disk-image-v1.0.0]$ ls
build.sh  core  install-disk-sd-a640-20190410.img
```

図 8.29 インストールディスクイメージのビルド

sudo を付けずに build.sh を実行すると、sudo を付けるよう指示するメッセージが表示されます。

```
[ATDE ~/make-install-disk-image-v1.0.0]$ ./build.sh
ERROR: Please run script with sudo
```

また、引数を指定せずに build.sh を実行すると、ヘルプが表示されます。

```
[ATDE ~/make-install-disk-image-v1.0.0]$ sudo ./build.sh
Install Disk Image Build Script v1.0.0

Usage:
  sudo ./build.sh BOARD UBOOT KERNEL DTB USERLAND

  BOARD: a640
  UBOOT: u-boot image
  KERNEL: uImage
  DTB: Device Tree Blob image
  USERLAND: Debian userland archive

Example:
  sudo ./build.sh a640 u-boot-a600-v2018.03-at4.imx uImage-a600-v4.14-
at11 armadillo-640-v4.14-at11.dtb debian-stretch-armhf-
a600-20190326.tar.gz
```

8.3.1.4. インストールディスクの作成

インストールディスクイメージを microSD カードに書き込むことで、インストールディスクを作成できます。

手順を簡単にするため、ATDE に microSD カード 以外のストレージを接続しないでください。

microSD カードを ATDE に接続してください。接続方法についての詳細は「7.7. 取り外し可能デバイスの使用」を参照してください。

microSD カード以外のストレージが接続されていなければ、microSD カードには /dev/mmcblk0 もしくは /dev/sdb のデバイスファイルが割り当てられます。microSD カード以外のストレージが接続されていた場合は、ATDE がストレージを認識した順番に依存して、microSD カードに /dev/mmcblk1 や /dev/sdc などのデバイスファイルが割り当てられます。

microSD カードに /dev/sdb が割り当てられた場合は、次のようになります。

```
[ATDE ~/make-install-disk-image-v1.0.0]$ ls /dev/mmcblk0
ls: '/dev/mmcblk0' にアクセスできません: そのようなファイルやディレクトリはありません
[ATDE ~/make-install-disk-image-v1.0.0]$ ls /dev/sdb
/dev/sdb
```

dd コマンドでインストールディスクイメージを microSD カードに書き込むことで、インストールディスクを作成します。

```
[ATDE ~/make-install-disk-image-v1.0.0]$ sudo umount /dev/sdb[0-9] ❶
[ATDE ~/make-install-disk-image-v1.0.0]$ sudo dd \
if=install-disk-sd-a640-20190410.img \ ❷
of=/dev/sdb \ ❸
conv=fsync
604160+0 レコード入力
604160+0 レコード出力
309329920 バイト (309 MB) コピーされました、 108.641 秒、 2.8 MB/秒
```

図 8.30 インストールディスクの作成

❶ microSD カードを接続した際に自動で全てのパーティションがマウントされるため、一旦アンマウントします。

❷ 入力ファイルに「8.3.1.3. インストールディスクイメージのビルド」で作成したインストールディスクイメージを指定します。

❸ 出力デバイスに microSD カードに割り当てられたデバイスファイルを指定します。

dd コマンドが終了したら、インストールディスク作成完了です。

8.3.2. インストールの実行

1. Armadillo の電源が切断されていることを確認します。接続されていた場合は、電源を切断してください。

2. USB シリアル変換アダプタのスライドスイッチを確認します。スライドスイッチが「図 8.31. スライドスイッチの設定」の 1 側に設定されている事を確認してください。

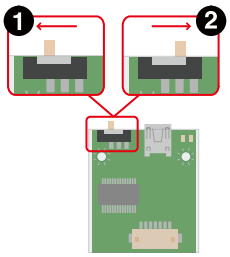

図 8.31 スライドスイッチの設定

3. インストールディスクを使用して SD ブートを行います。JP1 と JP2 を共にジャンパでショートし、インストールディスクを Armadillo に接続してください。

4. Armadillo に電源を投入すると microSD カードからブートローダーが起動し、次に示すログが表示されます。

```
U-Boot 2018.03-at1 installer+ (Apr 04 2018 - 21:17:22 +0900)

CPU:   Freescale i.MX6ULL rev1.0 at 396 MHz
Reset cause: POR
DRAM:  512 MiB
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
In:    serial
Out:   serial
Err:   serial
Net:   FEC
Warning: FEC (eth0) using random MAC address - 5a:fd:55:1c:bb:91

=>
```

5. 次のように"boot"コマンドを実行するとインストールが始まり、自動的に eMMC が書き換えられます。

```
=> boot
3345800 bytes read in 204 ms (15.6 MiB/s)
23185 bytes read in 53 ms (426.8 KiB/s)
## Booting kernel from Legacy Image at 82000000 ...
   Image Name:   Linux-4.14-at1
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    3345736 Bytes = 3.2 MiB
   Load Address: 82000000
   Entry Point:  82000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 83000000
   Booting using the fdt blob at 0x83000000
   Loading Kernel Image ... OK
   Loading Device Tree to 9eeff000, end 9ef07a90 ... OK

Starting kernel ...
: (省略)
*+*+* Install Start!! *+*+*
```

6. 以下のようにメッセージが表示されるとインストール完了です。電源を切断してください。

```
*+*+* Install Completed!! *+*+*
```

8.3.3. インストール後の動作確認

インストールが完了したら、「8.2.2. ルートファイルシステムのカスタマイズ」で行った変更が正しく反映されているか確認します。ここまでの手順で間違いがなければ、/usr/local/bin ディレクトリに sin が配置されており、sin を実行することができるはずです。

まずは Armadillo を再度起動し、root ユーザーでログインしてください。

次に以下のコマンドを実行して、/usr/local/bin/sin が存在するか、実行できるかを確認します。

```
[armadillo ~]# ls /usr/local/bin/sin
/usr/local/bin/sin
[armadillo ~]# /usr/local/bin/sin
sin(0.5) = 0.479426
```

図 8.32 インストール後の動作確認

「図 8.32. インストール後の動作確認」のようになれば問題ありません。

8.4. 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、それをルートファイルシステムに取り込み、さらにインストールディスクを作成して、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。

8.4.1. 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。

8.5. 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。

8.5.1. 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、Debian ユーザーランドは、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。製品出荷後でもソフトウェアをアップデートできるよう、手順や仕組みを準備しておくのが無難です。

8.5.2. 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。

Armadillo 標準ガイド Armadillo 入門編

改訂履歴

バージョン	年月日	改訂内容
1.0.0	2019/04/09	・初版発行
1.0.1	2019/04/25	・誤記修正

Armadillo 標準ガイド Armadillo 入門編
Version 1.0.1
2019/04/25