

# Armadillo-640 製品マニュアル

A6400-U00Z

A6400-D00Z

A6400-B00Z

A6400-N00Z

Version 3.1.0

2023/07/27

Armadillo Base OS 対応

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

Armadillo サイト [<https://armadillo.atmark-techno.com>]

---

# Armadillo-640 製品マニュアル

株式会社アットマークテクノ

製作著作 © 2023 Atmark Techno, Inc.

Version 3.1.0  
2023/07/27

# 目次

- 1. はじめに ..... 19
  - 1.1. 本書で扱うこと扱わないこと ..... 19
    - 1.1.1. 扱うこと ..... 19
    - 1.1.2. 扱わないこと ..... 19
  - 1.2. 本書で必要となる知識と想定する読者 ..... 20
  - 1.3. ユーザー限定コンテンツ ..... 20
  - 1.4. 本書および関連ファイルのバージョンについて ..... 20
  - 1.5. 本書の構成 ..... 20
  - 1.6. 表記について ..... 21
    - 1.6.1. フォント ..... 21
    - 1.6.2. コマンド入力例 ..... 21
    - 1.6.3. アイコン ..... 22
  - 1.7. 謝辞 ..... 22
- 2. 注意事項 ..... 23
  - 2.1. 安全に関する注意事項 ..... 23
  - 2.2. 取扱い上の注意事項 ..... 24
  - 2.3. 製品の保管について ..... 25
  - 2.4. ソフトウェア使用に関する注意事項 ..... 26
  - 2.5. 電波障害について ..... 26
  - 2.6. 無線モジュールの安全規制について ..... 26
  - 2.7. 保証について ..... 27
  - 2.8. 輸出について ..... 27
  - 2.9. 商標について ..... 28
- 3. 製品概要 ..... 29
  - 3.1. 製品の特長 ..... 29
    - 3.1.1. Armadillo とは ..... 29
    - 3.1.2. Armadillo-640 とは ..... 29
    - 3.1.3. Armadillo Base OS とは ..... 31
  - 3.2. 製品ラインアップ ..... 33
    - 3.2.1. Armadillo-640 ベーシックモデル開発セット ..... 33
    - 3.2.2. Armadillo-640 量産ボード ..... 33
  - 3.3. 仕様 ..... 33
  - 3.4. ブロック図 ..... 34
  - 3.5. ストレージデバイスのパーティション構成 ..... 35
- 4. Armadillo の電源を入れる前に ..... 37
  - 4.1. 準備するもの ..... 37
  - 4.2. 開発/動作確認環境の構築 ..... 37
    - 4.2.1. ATDE のセットアップ ..... 38
      - 4.2.1.1. VMware のインストール ..... 38
      - 4.2.1.2. ATDE のアーカイブを取得 ..... 38
      - 4.2.1.3. ATDE のアーカイブを展開 ..... 38
      - 4.2.1.4. Windows で ATDE のアーカイブ展開する ..... 39
      - 4.2.1.5. Linux で tar.xz 形式のファイルを展開する ..... 41
      - 4.2.1.6. ATDE の起動 ..... 41
    - 4.2.2. 取り外し可能デバイスの使用 ..... 43
    - 4.2.3. コマンドライン端末(GNOME 端末)の起動 ..... 44
    - 4.2.4. シリアル通信ソフトウェア(minicom)の使用 ..... 45
  - 4.3. インターフェースレイアウト ..... 49
  - 4.4. 接続方法 ..... 49
    - 4.4.1. USB シリアル変換アダプタの接続方法 ..... 52

- 4.5. ジャンパピンの設定について ..... 52
- 4.6. スライドスイッチの設定について ..... 53
- 4.7. vi エディタの使用方法 ..... 53
  - 4.7.1. vi の起動 ..... 54
  - 4.7.2. 文字の入力 ..... 54
  - 4.7.3. カーソルの移動 ..... 55
  - 4.7.4. 文字の削除 ..... 55
  - 4.7.5. 保存と終了 ..... 55
- 5. 起動と終了 ..... 56
  - 5.1. 起動 ..... 56
  - 5.2. ログイン ..... 58
  - 5.3. 終了方法 ..... 59
- 6. ユーザー登録 ..... 61
  - 6.1. 購入製品登録 ..... 61
- 7. 動作確認方法 ..... 62
  - 7.1. ネットワーク ..... 62
    - 7.1.1. 接続可能なネットワーク ..... 62
    - 7.1.2. IP アドレスの確認方法 ..... 62
    - 7.1.3. ネットワークの設定方法 ..... 63
      - 7.1.3.1. nmcli について ..... 63
    - 7.1.4. nmcli の基本的な使い方 ..... 63
      - 7.1.4.1. コネクションの一覧 ..... 63
      - 7.1.4.2. コネクションの有効化・無効化 ..... 63
      - 7.1.4.3. コネクションの作成 ..... 64
      - 7.1.4.4. コネクションの削除 ..... 65
      - 7.1.4.5. 固定 IP アドレスに設定する ..... 65
      - 7.1.4.6. DNS サーバーを指定する ..... 65
      - 7.1.4.7. DHCP に設定する ..... 65
      - 7.1.4.8. コネクションの修正を反映する ..... 66
      - 7.1.4.9. デバイスの一覧 ..... 66
      - 7.1.4.10. デバイスの接続 ..... 66
      - 7.1.4.11. デバイスの切断 ..... 67
    - 7.1.5. 有線 LAN ..... 67
  - 7.2. ストレージ ..... 67
    - 7.2.1. ストレージの使用方法 ..... 68
    - 7.2.2. ストレージのパーティション変更とフォーマット ..... 69
  - 7.3. LED ..... 70
    - 7.3.1. LED を点灯/消灯する ..... 70
    - 7.3.2. トリガを使用する ..... 71
  - 7.4. ユーザースイッチ ..... 72
    - 7.4.1. イベントを確認する ..... 72
- 8. 開発の基本的な流れ ..... 74
  - 8.1. アプリケーション開発の流れ ..... 74
    - 8.1.1. Armadillo への接続 ..... 74
      - 8.1.1.1. シリアルコンソール ..... 74
      - 8.1.1.2. ssh ..... 74
    - 8.1.2. overlayfs の扱い ..... 74
    - 8.1.3. Podman のデータを eMMC に保存する ..... 74
    - 8.1.4. ベースとなるコンテナを取得する ..... 75
    - 8.1.5. デバイスのアクセス権を与える ..... 75
    - 8.1.6. アプリケーションを作成する ..... 76
    - 8.1.7. コンテナやデータを保存する ..... 76
  - 8.2. アプリケーションコンテナの運用 ..... 76

8.2.1. アプリケーションの自動起動 .....	76
8.2.2. アプリケーションの送信 .....	76
8.2.3. インストール確認：初期化 .....	76
8.2.4. アプリケーションのアップデート .....	77
9. Web UI によるセットアップ .....	78
9.1. ABOS Web を使った Armadillo のセットアップ .....	78
9.1.1. ABOS Web のパスワード登録 .....	78
9.1.2. ABOS Web の設定操作 .....	82
9.1.3. ログアウト .....	82
9.2. ABOS Web ではできないこと .....	82
9.3. ABOS Web の設定機能一覧と設定手順 .....	82
9.3.1. WWAN 設定 .....	83
9.3.2. WLAN 設定 .....	84
9.3.2.1. WLAN 設定（クライアントとしての設定） .....	85
9.3.2.2. WLAN 設定（アクセスポイントとしての設定） .....	87
9.3.3. 各接続設定（各ネットワークインターフェースの設定） .....	89
9.3.3.1. LAN 接続設定 .....	90
9.3.3.2. WWAN 接続設定 .....	90
9.3.3.3. WLAN 接続設定 .....	90
9.3.4. DHCP サーバー設定 .....	91
9.3.5. NAT 設定 .....	91
9.3.5.1. NAT 設定 .....	92
9.3.5.2. ポートフォワーディング設定 .....	92
9.3.6. VPN 設定 .....	93
9.3.7. コンテナ管理 .....	95
9.3.8. SWU インストール .....	95
9.3.9. 状態一覧 .....	97
10. Howto .....	98
10.1. CUI アプリケーションを開発する .....	98
10.1.1. CUI アプリケーション開発の流れ .....	98
10.1.2. ATDE 上でのセットアップ .....	98
10.1.2.1. ソフトウェアのアップデート .....	99
10.1.2.2. VSCode に開発用エクステンションをインストールする .....	99
10.1.2.3. プロジェクトの作成 .....	100
10.1.2.4. 初期設定 .....	100
10.1.2.5. アプリケーション実行用コンテナイメージの作成 .....	102
10.1.3. Armadillo 上でのセットアップ .....	103
10.1.3.1. アプリケーション実行用コンテナイメージのインストール .....	103
10.1.4. アプリケーション開発 .....	104
10.1.4.1. VSCode の起動 .....	104
10.1.4.2. ディレクトリ構成 .....	104
10.1.4.3. ssh_config の準備 .....	104
10.1.4.4. アプリケーションの実行 .....	105
10.1.5. リリース版のビルド .....	106
10.1.6. 製品への書き込み .....	107
10.2. アプリケーションをコンテナで実行する .....	107
10.2.1. Podman - コンテナ仮想化ソフトウェア .....	107
10.2.1.1. Podman - コンテナ仮想化ソフトウェアとは .....	107
10.2.2. コンテナを操作する .....	107
10.2.2.1. イメージからコンテナを作成する .....	108
10.2.2.2. イメージ一覧を表示する .....	109
10.2.2.3. コンテナ一覧を表示する .....	109
10.2.2.4. コンテナを起動する .....	110

10.2.2.5. コンテナを停止する .....	111
10.2.2.6. コンテナの変更を保存する .....	111
10.2.2.7. コンテナの自動作成やアップデート .....	112
10.2.2.8. コンテナを削除する .....	114
10.2.2.9. イメージを削除する .....	114
10.2.2.10. コンテナとコンテナに関連するデータを削除する .....	115
10.2.2.11. 実行中のコンテナに接続する .....	116
10.2.2.12. コンテナ間で通信をする .....	117
10.2.2.13. 開発時に有用な—privileged オプション .....	118
10.2.3. アットマークテクノが提供するイメージを使う .....	118
10.2.3.1. Docker ファイルからイメージをビルドする .....	118
10.2.3.2. ビルド済みのイメージを使用する .....	119
10.2.4. 入出力デバイスを扱う .....	119
10.2.4.1. GPIO を扱う .....	119
10.2.4.2. I2C を扱う .....	122
10.2.4.3. SPI を扱う .....	122
10.2.4.4. CAN を扱う .....	123
10.2.4.5. PWM を扱う .....	124
10.2.4.6. シリアルインターフェースを扱う .....	124
10.2.4.7. USB を扱う .....	125
10.2.4.8. RTC を扱う .....	127
10.2.4.9. 音声出力を行う .....	128
10.2.4.10. ユーザースイッチのイベントを取得する .....	129
10.2.4.11. LED を扱う .....	130
10.2.5. 近距離通信を行う .....	130
10.2.5.1. Bluetooth デバイスを扱う .....	130
10.2.5.2. Wi-SUN デバイスを扱う .....	131
10.2.5.3. EnOcean デバイスを扱う .....	132
10.2.5.4. Thread デバイスを扱う .....	132
10.2.6. ネットワークを扱う .....	133
10.2.6.1. コンテナの IP アドレスを確認する .....	133
10.2.6.2. コンテナに固定 IP アドレスを設定する .....	133
10.2.7. サーバを構築する .....	134
10.2.7.1. HTTP サーバを構築する .....	134
10.2.7.2. FTP サーバを構築する .....	135
10.2.7.3. Samba サーバを構築する .....	136
10.2.7.4. SQL サーバを構築する .....	137
10.2.8. 画面表示を行う .....	138
10.2.8.1. X Window System を扱う .....	138
10.2.8.2. フレームバッファに直接描画する .....	139
10.2.8.3. タッチパネルを扱う .....	139
10.2.9. パワーマネジメント機能を使う .....	140
10.2.9.1. サスペンド状態にする .....	140
10.2.9.2. 起床要因を有効化する .....	141
10.2.10. コンテナからの poweroff か reboot .....	142
10.2.11. 異常検知 .....	142
10.2.11.1. ソフトウェアウォッチドッグタイマーを扱う .....	142
10.3. コンテナの運用 .....	143
10.3.1. コンテナの自動起動 .....	143
10.3.2. pod の作成 .....	148
10.3.3. network の作成 .....	149
10.3.4. コンテナからのコンテナ管理 .....	150
10.3.5. コンテナの配布 .....	150

10.3.5.1. リモートリポジトリにコンテナを送信する方法 .....	150
10.3.5.2. イメージを eMMC に保存する方法 .....	151
10.3.5.3. イメージを SWUpdate で転送する方法 .....	153
10.4. Armadillo のソフトウェアをビルドする .....	154
10.4.1. ブートローダーをビルドする .....	154
10.4.2. Linux カーネルをビルドする .....	155
10.4.3. Alpine Linux ルートファイルシステムをビルドする .....	158
10.5. SD ブートの活用 .....	161
10.5.1. ブートディスクの作成 .....	162
10.5.2. SD ブートの実行 .....	163
10.6. Armadillo のソフトウェアの初期化 .....	164
10.6.1. インストールディスクの作成 .....	165
10.6.1.1. 初期化インストールディスクの作成 .....	165
10.6.1.2. 開発が完了した Armadillo をクローンするインストールディスクの作成 .....	166
10.6.2. インストールディスクを使用する .....	167
10.7. Armadillo のソフトウェアをアップデートする .....	167
10.7.1. SWU イメージとは .....	168
10.7.2. SWU イメージの作成 .....	168
10.7.3. イメージのインストール .....	170
10.7.4. swupdate がエラーする場合の対処 .....	173
10.7.5. hawkBit サーバーから複数の Armadillo に配信する .....	173
10.7.5.1. hawkBit のアップデート管理を CLI で行う .....	185
10.7.5.2. SWU で hawkBit を登録する .....	185
10.7.6. mkswu の desc ファイル .....	187
10.7.6.1. 例: sshd を有効にする .....	190
10.7.6.2. 例: Armadillo Base OS アップデート .....	191
10.7.6.3. 例: swupdate_preserve_files で Linux カーネル以外の Armadillo-640 向けのイメージをインストールする方法 .....	191
10.7.7. swupdate_preserve_files について .....	192
10.7.8. SWU イメージの内容の確認 .....	193
10.7.9. SWUpdate と暗号化について .....	193
10.8. Armadillo Base OS の操作 .....	193
10.8.1. アップデート .....	193
10.8.2. overlayfs と persist_file について .....	194
10.8.3. ロールバック状態の確認 .....	196
10.8.4. ボタンやキーを扱う .....	196
10.8.5. Armadillo Base OS 側の起動スクリプト .....	198
10.8.6. u-boot の環境変数の設定 .....	199
10.8.7. Network Time Protocol (NTP, ネットワーク・タイム・プロトコル) .....	201
10.9. Device Tree をカスタマイズする .....	201
10.9.1. at-dtweb のインストール .....	202
10.9.2. at-dtweb の起動 .....	202
10.9.3. Device Tree をカスタマイズ .....	204
10.9.3.1. 機能の選択 .....	204
10.9.3.2. 信号名の確認 .....	205
10.9.3.3. プロパティの設定 .....	206
10.9.3.4. 機能の削除 .....	208
10.9.3.5. Device Tree のファイルの生成 .....	209
10.9.4. DT overlay によるカスタマイズ .....	211
10.9.4.1. 提供している DT overlay .....	211
10.10. LCD オプションセット(7 インチタッチパネル WVGA 液晶)を利用する .....	212
10.10.1. LCD オプションセット(7 インチタッチパネル WVGA 液晶)を利用する準備 ..	212
10.10.2. LCD オプションセット(7 インチタッチパネル WVGA 液晶)を利用する .....	212

10.11. Armadillo-600 シリーズ WLAN コンボオプションモジュールを利用する .....	212
10.11.1. 無線 LAN を利用する準備 .....	212
10.11.2. 無線 LAN アクセスポイント (AP) に接続する .....	213
10.11.3. 無線 LAN アクセスポイント (AP) として設定する .....	215
10.11.3.1. hostapd を使用して設定する .....	215
10.11.4. ルータとして設定する .....	218
10.11.4.1. 無線 LAN 側に接続した機器から Ethernet 経由でインターネットに接続する .....	218
10.11.4.2. Ethernet 側に接続した機器から無線 LAN 経由でインターネットに接続する .....	219
10.12. Armadillo-600 シリーズ BT/TH オプションモジュールを利用する .....	220
10.12.1. Armadillo-600 シリーズ BT/TH オプションモジュールを利用する準備 .....	220
10.12.2. BT 機能を利用する .....	221
10.12.3. Thread 機能を利用する .....	222
10.13. 動作中の Armadillo の温度を測定する .....	223
10.13.1. 温度測定的重要性 .....	223
10.13.2. atmark-thermal-profiler をインストールする .....	223
10.13.3. atmark-thermal-profiler を実行・停止する .....	224
10.13.4. atmark-thermal-profiler が出力するログファイルを確認する .....	224
10.13.5. 温度測定結果の分析 .....	225
10.13.5.1. サーマルシャットダウン温度の確認 .....	225
10.13.5.2. 温度測定結果のグラフ化 .....	225
10.13.5.3. CPU 使用率の確認 .....	226
10.14. eMMC の GPP(General Purpose Partition) を利用する .....	226
10.14.1. squashfs イメージを作成する .....	226
10.14.2. squashfs イメージを書き込む .....	227
10.14.3. GPP への書き込みを制限する .....	227
10.14.4. 起動時に squashfs イメージをマウントされるようにする .....	228
10.15. eFuse を変更する .....	228
10.15.1. ブートモード .....	229
10.15.1.1. Internal Boot モード .....	229
10.15.2. ブートデバイス .....	230
10.15.3. eFuse の書き換え .....	230
10.15.4. eFuse の設定によるブートデバイスの選択 .....	231
10.15.4.1. BT_FUSE_SEL .....	231
10.15.4.2. eMMC からのブートに固定 .....	232
10.15.4.3. eFuse のロック .....	233
11. 動作ログ .....	234
11.1. 動作ログについて .....	234
11.2. 動作ログを取り出す .....	234
11.3. ログファイルのフォーマット .....	234
11.4. ログ用パーティションについて .....	234
12. 製品機能 .....	235
12.1. UART .....	235
12.2. Ethernet .....	235
12.3. SD ホスト .....	236
12.4. USB ホスト .....	237
12.5. リアルタイムクロック .....	238
12.6. LED .....	238
12.7. ユーザースイッチとイベント信号 .....	238
12.8. I2C .....	239
12.9. パワーマネジメント .....	240
12.10. GPIO .....	241



12.11.	温度センサー .....	242
12.12.	ウォッチドッグタイマー .....	242
12.13.	WLAN .....	243
12.14.	BT .....	244
12.15.	LCD .....	245
13.	ソフトウェア仕様 .....	246
13.1.	SWUpdate .....	246
13.1.1.	SWUpdate とは .....	246
13.1.2.	swu パッケージ .....	246
13.1.3.	A/B アップデート(アップデートの2面化) .....	246
13.1.4.	ロールバック (リカバリー) .....	247
13.2.	hawkBit .....	247
13.2.1.	hawkBit とは .....	247
13.2.2.	データ構造 .....	247
14.	ハードウェア仕様 .....	249
14.1.	電氣的仕様 .....	249
14.1.1.	絶対最大定格 .....	249
14.1.2.	推奨動作条件 .....	249
14.1.3.	入出力インターフェースの電氣的仕様 .....	249
14.1.4.	電源回路の構成 .....	250
14.1.5.	外部からの電源制御 .....	252
14.1.5.1.	ONOFF ピンの制御について .....	252
14.1.5.2.	PWRON ピンの制御について .....	253
14.1.5.3.	RTC_BAT ピンについて .....	253
14.2.	インターフェース仕様 .....	253
14.2.1.	CON1(SD インターフェース) .....	254
14.2.1.1.	microSD カードの挿抜方法 .....	255
14.2.2.	CON2、CON7(LAN インターフェース) .....	257
14.2.3.	LED1、LED2(LAN LED) .....	258
14.2.4.	CON3、CON4(シリアルインターフェース) .....	258
14.2.5.	CON5(USB ホストインターフェース) .....	259
14.2.6.	CON8、CON9、CON14(拡張インターフェース) .....	260
14.2.7.	CON10(JTAG インターフェース) .....	262
14.2.8.	CON11(LCD 拡張インターフェース) .....	263
14.2.9.	CON12、CON13(電源インターフェース) .....	265
14.2.10.	LED3、LED4、LED5(ユーザー LED) .....	266
14.2.11.	SW1(ユーザースイッチ) .....	267
14.2.12.	JP1、JP2(起動デバイス設定ジャンパ) .....	267
14.3.	形状図 .....	268
14.3.1.	基板形状図 .....	268
14.4.	設計情報 .....	269
14.4.1.	信頼性試験データについて .....	269
14.4.2.	放射ノイズ .....	269
14.4.3.	ESD/雷サージ .....	270
15.	オプション品 .....	271
15.1.	USB シリアル変換アダプタ(Armadillo-640 用) .....	271
15.1.1.	概要 .....	271
15.2.	Armadillo-600 シリーズ オプションケース(樹脂製) .....	273
15.2.1.	概要 .....	273
15.2.2.	組み立て .....	274
15.2.3.	形状図 .....	276
15.3.	Armadillo-600 シリーズ オプションケース(金属製) .....	277
15.3.1.	概要 .....	277

15.3.2. 組み立て .....	277
15.3.3. 形状図 .....	278
15.4. LCD オプションセット(7 インチタッチパネル WVGA 液晶) .....	279
15.4.1. 概要 .....	279
15.4.2. 組み立て .....	280
15.5. Armadillo-600 シリーズ RTC オプションモジュール .....	281
15.5.1. 概要 .....	281
15.5.2. 仕様 .....	281
15.5.3. ブロック図 .....	282
15.5.4. インターフェース仕様 .....	282
15.5.4.1. CON1(Armadillo-600 シリーズ接続インターフェース) .....	283
15.5.4.2. CON3(USB ホストインターフェース) .....	283
15.5.4.3. CON4、CON5、CON6(RTC バックアップインターフェース) .....	284
15.5.4.4. JP1(ONOFF ピン接続ジャンパ) .....	285
15.5.5. 組み立て .....	285
15.5.5.1. Armadillo-640 と RTC オプションモジュールの組み立て .....	285
15.5.5.2. 電池の取り付け、取り外し .....	286
15.5.6. 形状図 .....	287
15.6. Armadillo-600 シリーズ WLAN コンボ、BT/TH オプションモジュール .....	288
15.6.1. 概要 .....	288
15.6.2. 仕様 .....	289
15.6.3. ブロック図 .....	290
15.6.4. インターフェース仕様 .....	290
15.6.4.1. CON1(Armadillo-600 シリーズ接続インターフェース) .....	291
15.6.4.2. CON2(拡張インターフェース) .....	292
15.6.5. 形状図 .....	292
15.6.6. 組み立て .....	293
15.6.6.1. オプションボードの組み立て .....	293
15.6.6.2. アンテナの組み立て .....	293
15.6.6.3. ケースの組み立て .....	294
15.6.7. シリアルコンソールの使用方法 .....	295
15.7. 無線 LAN 用 外付けアンテナセット 08 .....	297
15.7.1. 概要 .....	297
15.7.2. 組み立て .....	297
15.7.2.1. アンテナケーブルコネクタの嵌合 .....	297
15.7.2.2. アンテナケーブルコネクタの抜去 .....	299
15.7.3. 形状図 .....	300
15.8. 外付けアンテナ固定金具 00 .....	300
15.8.1. 概要 .....	300
15.8.2. 組み立て .....	301
15.8.3. 形状図 .....	303

## 目次

2.1. WLAN+BT コンボモジュール: Sterling LWB5+ 認証マーク	27
2.2. EYSKBNZWB 認証マーク	27
3.1. Armadillo-640 とは	30
3.2. Armadillo Base OS とは	31
3.3. コンテナによるアプリケーションの運用	32
3.4. ロールバックの仕組み	32
3.5. Armadillo-640 ブロック図	35
4.1. GNOME 端末の起動	44
4.2. GNOME 端末のウィンドウ	45
4.3. minicom の設定の起動	45
4.4. minicom の設定	45
4.5. minicom のシリアルポートの設定	46
4.6. 例. USB to シリアル変換ケーブル接続時のログ	46
4.7. minicom のシリアルポートのパラメータの設定	47
4.8. minicom シリアルポートの設定値	47
4.9. minicom 起動方法	48
4.10. minicom 終了確認	48
4.11. インターフェースレイアウト	49
4.12. Armadillo-640 の接続例	50
4.13. COM7 が競合している状態	51
4.14. Bluetooth に割当の COM を変更した状態	51
4.15. CON9-USB シリアル変換アダプタ接続図	52
4.16. JP1、JP2 の位置	52
4.17. スライドスイッチの設定	53
4.18. vi の起動	54
4.19. 入力モードに移行するコマンドの説明	54
4.20. 文字を削除するコマンドの説明	55
7.1. IP アドレスの確認	62
7.2. IP アドレス(eth0)の確認	62
7.3. nmcli のコマンド書式	63
7.4. コネクションの一覧	63
7.5. コネクションの有効化	64
7.6. コネクションの無効化	64
7.7. コネクションの作成	64
7.8. コネクションファイルの永続化	64
7.9. コネクションの削除	65
7.10. コネクションファイル削除時の永続化	65
7.11. 固定 IP アドレス設定	65
7.12. DNS サーバーの指定	65
7.13. DHCP の設定	66
7.14. コネクションの修正の反映	66
7.15. デバイスの一覧	66
7.16. デバイスの接続	66
7.17. デバイスの切断	67
7.18. 有線 LAN の PING 確認	67
7.19. mount コマンド書式	68
7.20. ストレージのマウント	69
7.21. ストレージのアンマウント	69
7.22. fdisk コマンドによるパーティション変更	69
7.23. EXT4 ファイルシステムの構築	70

7.24. LED を点灯させる .....	71
7.25. LED を消灯させる .....	71
7.26. LED の状態を表示する .....	71
7.27. 対応している LED トリガを表示 .....	71
7.28. LED のトリガに timer を指定する .....	72
7.29. LED のトリガに heartbeat を指定する .....	72
7.30. evtest コマンドのインストール .....	72
7.31. ユーザースイッチ: イベントの確認 .....	73
9.1. パスワード登録画面 .....	79
9.2. パスワード登録完了画面 .....	80
9.3. ログイン画面 .....	81
9.4. トップページ .....	81
9.5. WWAN 設定画面 .....	84
9.6. WLAN クライアント設定画面 .....	86
9.7. WLAN アクセスポイント設定画面 .....	88
9.8. 現在の接続情報画面 .....	89
9.9. LAN 接続設定で固定 IP アドレスに設定した画面 .....	90
9.10. eth0 に対する DHCP サーバー設定 .....	91
9.11. LTE を宛先インターフェースに指定した設定 .....	92
9.12. LTE からの受信パケットに対するポートフォワーディング設定 .....	93
9.13. VPN 設定 .....	94
9.14. コンテナ管理 .....	95
9.15. SWU インストール .....	96
9.16. SWU 管理対象ソフトウェアコンポーネントの一覧表示 .....	97
10.1. CUI アプリケーション開発の流れ .....	98
10.2. ソフトウェアをアップデートする .....	99
10.3. VSCode を起動する .....	99
10.4. VSCode に開発用エクステンションをインストールする .....	99
10.5. プロジェクトを作成する .....	100
10.6. プロジェクト名を入力する .....	100
10.7. 初期設定を行う .....	101
10.8. VSCode で初期設定を行う .....	101
10.9. VSCode のターミナル .....	101
10.10. SSH 用の鍵を生成する .....	101
10.11. VSCode でコンテナイメージの作成を行う .....	103
10.12. コンテナイメージの作成完了 .....	103
10.13. VSCode で my_project を起動する .....	104
10.14. ssh_config を編集する .....	104
10.15. Armadillo 上でアプリケーションを実行する .....	105
10.16. 実行時に表示されるメッセージ .....	105
10.17. アプリケーションを終了する .....	106
10.18. リリース版をビルドする .....	107
10.19. コンテナを作成する実行例 .....	108
10.20. イメージ一覧の表示実行例 .....	109
10.21. podman images --help の実行例 .....	109
10.22. コンテナ一覧の表示実行例 .....	109
10.23. podman ps --help の実行例 .....	110
10.24. コンテナを起動する実行例 .....	110
10.25. コンテナを起動する実行例(a オプション付与) .....	110
10.26. podman start --help 実行例 .....	110
10.27. コンテナを停止する実行例 .....	111
10.28. podman stop --help 実行例 .....	111
10.29. my_container を保存する例 .....	111

10.30. chattr によって copy-on-write を無効化する例 .....	112
10.31. podman build の実行例 .....	113
10.32. podman build でのアップデートの実行例 .....	113
10.33. コンテナを削除する実行例 .....	114
10.34. \$ podman rm --help 実行例 .....	114
10.35. イメージを削除する実行例 .....	114
10.36. podman rmi --help 実行例 .....	115
10.37. Read-Only のイメージを削除する実行例 .....	115
10.38. abos-ctrl container-clear 実行例 .....	116
10.39. コンテナ内部のシェルを起動する実行例 .....	116
10.40. コンテナ内部のシェルから抜ける実行例 .....	117
10.41. podman exec --help 実行例 .....	117
10.42. コンテナを作成する実行例 .....	117
10.43. コンテナの IP アドレスを確認する実行例 .....	117
10.44. ping コマンドによるコンテナ間の疎通確認実行例 .....	118
10.45. Docker ファイルによるイメージのビルドの実行例 .....	119
10.46. ビルド済みイメージを load する実行例 .....	119
10.47. GPIO を扱うためのコンテナ作成例 .....	120
10.48. コンテナ内からコマンドで GPIO を操作する例 .....	120
10.49. gpiodetect コマンドの実行 .....	120
10.50. gpioinfo コマンドの実行 .....	120
10.51. I2C を扱うためのコンテナ作成例 .....	122
10.52. i2cdetect コマンドによる確認例 .....	122
10.53. SPI を扱うためのコンテナ作成例 .....	122
10.54. spi-config コマンドによる確認例 .....	123
10.55. CAN を扱うためのコンテナ作成例 .....	123
10.56. CAN の設定例 .....	123
10.57. PWM を扱うためのコンテナ作成例 .....	124
10.58. PWM の動作設定例 .....	124
10.59. シリアルインターフェースを扱うためのコンテナ作成例 .....	125
10.60. setserial コマンドによるシリアルインターフェイス設定の確認例 .....	125
10.61. USB シリアルデバイスを扱うためのコンテナ作成例 .....	125
10.62. setserial コマンドによる USB シリアルデバイス設定の確認例 .....	125
10.63. USB カメラを扱うためのコンテナ作成例 .....	126
10.64. USB メモリをホスト OS 側でマウントする例 .....	126
10.65. ホスト OS 側でマウント済みの USB メモリを扱うためのコンテナ作成例 .....	126
10.66. USB メモリに保存されているデータの確認例 .....	127
10.67. USB メモリをマウントするためのコンテナ作成例 .....	127
10.68. コンテナ内から USB メモリをマウントする例 .....	127
10.69. RTC を扱うためのコンテナ作成例 .....	127
10.70. hwclock コマンドによる RTC の時刻表示と設定例 .....	128
10.71. 音声出力を行うためのコンテナ作成例 .....	128
10.72. alsa-utils による音声出力を行う例 .....	128
10.73. ユーザースイッチのイベントを取得するためのコンテナ作成例 .....	129
10.74. evtest コマンドによる確認例 .....	129
10.75. LED を扱うためのコンテナ作成例 .....	130
10.76. LED の点灯/消灯の実行例 .....	130
10.77. Bluetooth デバイスを扱うためのコンテナ作成例 .....	130
10.78. Bluetooth を起動する実行例 .....	131
10.79. bluetoothctl コマンドによるスキャンとペアリングの例 .....	131
10.80. Wi-SUN デバイスを扱うためのコンテナ作成例 .....	132
10.81. EnOcean デバイスを扱うためのコンテナ作成例 .....	132
10.82. Thread デバイスを扱うためのコンテナ作成例 .....	132

10.83. コンテナの IP アドレス確認例 .....	133
10.84. ip コマンドを用いたコンテナの IP アドレス確認例 .....	133
10.85. ユーザ定義のネットワーク作成例 .....	134
10.86. IP アドレス固定のコンテナ作成例 .....	134
10.87. コンテナの IP アドレス確認例 .....	134
10.88. コンテナに Apache をインストールする例 .....	134
10.89. コンテナに lighttpd をインストールする例 .....	135
10.90. コンテナに vsftpd をインストールする例 .....	135
10.91. ユーザを追加する例 .....	136
10.92. 設定ファイルの編集例 .....	136
10.93. vsftpd の起動例 .....	136
10.94. コンテナに samba をインストールする例 .....	136
10.95. ユーザを追加する例 .....	137
10.96. samba の起動例 .....	137
10.97. コンテナに sqlite をインストールする例 .....	137
10.98. sqlite の実行例 .....	137
10.99. X Window System を扱うためのコンテナ起動例 .....	138
10.100. コンテナ内で X Window System を起動する実行例 .....	138
10.101. フレームバッファに直接描画するためのコンテナ作成例 .....	139
10.102. フレームバッファに直接描画する実行例 .....	139
10.103. タッチパネルを扱うためのコンテナ作成例 .....	139
10.104. パワーマネジメント機能を使うためのコンテナ作成例 .....	140
10.105. サスペンド状態にする実行例 .....	141
10.106. サスペンド状態にする実行例、rtc で起こす .....	141
10.107. コンテナから shutdown を行う .....	142
10.108. ソフトウェアウォッチドッグタイマーを使うためのコンテナ作成例 .....	142
10.109. コンテナ内からソフトウェアウォッチドッグタイマーを起動する実行例 .....	143
10.110. ソフトウェアウォッチドッグタイマーをリセットする実行例 .....	143
10.111. ソフトウェアウォッチドッグタイマーを停止する実行例 .....	143
10.112. コンテナを自動起動するための設定例 .....	143
10.113. ボリュームを shared でサブマウントを共有する例 .....	145
10.114. /proc/devices の内容例 .....	146
10.115. pod を使うコンテナを自動起動するための設定例 .....	148
10.116. network を使うコンテナを自動起動するための設定例 .....	149
10.117. abos-ctrl podman-rw の実行例 .....	151
10.118. abos-ctrl podman-storage のイメージコピー例 .....	152
10.119. デフォルトコンフィギュレーションの適用 .....	154
10.120. Linux カーネルを SWU でインストールする方法 .....	157
10.121. Linux カーネルを build_rootfs でインストールする方法 .....	158
10.122. 自動マウントされた microSD カードのアンマウント .....	162
10.123. hawkBit コンテナの TLS なしの場合（テスト用）の実行例 .....	174
10.124. hawkBit コンテナの TLS ありの場合の実行例 .....	175
10.125. persist_file のヘルプ .....	194
10.126. persist_file 保存・削除手順例 .....	194
10.127. persist_file ソフトウェアアップデート後も変更を維持する手順例 .....	195
10.128. persist_file 変更ファイルの一覧表示例 .....	195
10.129. persist_file でのパッケージインストール手順例 .....	196
10.130. /var/at-log/atlog の内容の例 .....	196
10.131. buttond で SW1 を扱う .....	197
10.132. local サービスの実行例 .....	198
10.133. uboot_env.d のコンフィグファイルの例 .....	199
10.134. chronyd のコンフィグの変更例 .....	201
10.135. at-dtweb の起動開始 .....	202

10.136. ボード選択画面 .....	203
10.137. Linux カーネルディレクトリ選択画面 .....	203
10.138. at-dtweb 起動画面 .....	203
10.139. UART1 (RXD/TXD)のドラッグ .....	204
10.140. CON9 3/5 ピンへのドロップ .....	205
10.141. 信号名の確認 .....	206
10.142. プロパティの設定 .....	207
10.143. プロパティの保存 .....	207
10.144. 全ての機能の削除 .....	208
10.145. UART1 (RXD/TXD)の削除 .....	209
10.146. DTS/DTB の生成 .....	210
10.147. dtbo/desc の生成完了 .....	210
10.148. /boot/overlays.txt の変更例 .....	211
10.149. DT overlay の設定(LCD) .....	212
10.150. コンソールを CON3 に移動(WLAN) .....	213
10.151. DT overlay の設定(WLAN) .....	213
10.152. 無線 LAN AP に接続する .....	213
10.153. nmtui を起動する .....	214
10.154. bridge インターフェースを作成する .....	215
10.155. wlan0 インターフェースを NetworkManager の管理から外す .....	216
10.156. hostapd.conf を編集する .....	216
10.157. dnsmasq の設定ファイルを編集する .....	217
10.158. ip_forward を有効にする .....	218
10.159. NAT を設定する .....	218
10.160. bridge に eth0 を割り当てる .....	219
10.161. dnsmasq の設定ファイルを編集する .....	219
10.162. NAT を設定する .....	220
10.163. コンソールを CON3 に移動(BT/TH) .....	220
10.164. DT overlay の設定(BT/TH) .....	221
10.165. BT/TH オプションモジュールの BT 機能を有効化する .....	221
10.166. Sterling LWB5+ の BT 機能を無効化する .....	222
10.167. BT/TH オプションモジュールの Thread 機能を有効化する .....	222
10.168. atmark-thermal-profiler をインストールする .....	223
10.169. atmark-thermal-profiler を実行する .....	224
10.170. atmark-thermal-profiler を停止する .....	224
10.171. ログファイルの内容例 .....	224
10.172. サーマルシャットダウン温度の確認(Armadillo-IoT ゲートウェイ G4 を例に) .....	225
10.173. Armadillo-IoT ゲートウェイ G4 で取得した温度のグラフ .....	226
10.174. squashfs イメージの作成 .....	227
10.175. mmc-utils のインストール .....	227
10.176. eMMC の GPP に Temporary Write Protection をかける .....	227
11.1. 動作ログのフォーマット .....	234
13.1. hawkBit が扱うソフトウェアのデータ構造 .....	248
14.1. 電源回路の構成 .....	251
14.2. 電源シーケンス .....	252
14.3. Armadillo-640 のインターフェース .....	254
14.4. カバーのロックを解除する .....	255
14.5. カバーを開ける .....	255
14.6. microSD カードの挿抜 .....	256
14.7. カードマークの確認 .....	256
14.8. カバーを閉める .....	257
14.9. カバーをロックする .....	257
14.10. USB OTG2 の接続先の変更 .....	260

14.11. USB ホストインターフェースの電源制御 .....	260
14.12. リセットシーケンス .....	262
14.13. AC アダプタの極性マーク .....	265
14.14. バックアップ電源供給 .....	266
14.15. 基板形状および固定穴寸法 .....	268
14.16. コネクタ中心寸法 .....	268
14.17. コネクタ穴寸法 .....	269
15.1. USB シリアル変換アダプタの配線 .....	271
15.2. Armadillo-640 のシリアル信号線 .....	272
15.3. Armadillo-600 シリーズ オプションケース(樹脂製)の組み立て .....	274
15.4. 樹脂ケース形状図 .....	276
15.5. Armadillo-600 シリーズ オプションケース(金属製)の組み立て .....	277
15.6. 金属ケース(上板)寸法図 .....	278
15.7. 金属ケース(下板)寸法図 .....	279
15.8. LCD の接続方法 .....	280
15.9. フレキシブルフラットケーブルの形状 .....	281
15.10. RTC オプションモジュールのブロック図 .....	282
15.11. RTC オプションモジュールのインターフェース .....	283
15.12. Armadillo-640 CON5 上段と RTC オプションモジュール CON3 は排他利用 .....	284
15.13. RTC オプションモジュールの組み立て .....	286
15.14. 電池ホルダに電池を取り付ける .....	286
15.15. 電池ホルダから電池を取り外す .....	287
15.16. RTC オプションモジュール形状 .....	287
15.17. WLAN コンボ、BT/TH オプションモジュール接続時に Armadillo-640 CON5 上段は使用不可 .....	289
15.18. WLAN コンボ、BT/TH オプションモジュールのブロック図 .....	290
15.19. WLAN コンボ、BT/TH オプションモジュールのインターフェース .....	291
15.20. WLAN コンボ、BT/TH オプションモジュール形状図 .....	292
15.21. WLAN コンボ、BT/TH オプションモジュールの組み立て .....	293
15.22. 外付けアンテナの組み立て(OP-MNT-ANT-MET-00 使用) .....	294
15.23. 外付けアンテナの組み立て(オプションケース対応のアンテナ固定金具使用) .....	294
15.24. ケースの組み立て .....	295
15.25. オプションモジュールの CON2 をシリアルコンソールとして使用する場合の接続例 .....	296
15.26. CON4 をシリアルコンソールとして使用する場合の接続例 .....	296
15.27. CON3 をシリアルコンソールとして使用する場合の接続例 .....	297
15.28. Sterling LWB5+上のコネクタの位置 .....	298
15.29. 挿抜治具によるアンテナケーブルコネクタの嵌合 .....	298
15.30. アンテナケーブルコネクタのセット .....	299
15.31. アンテナケーブルコネクタの嵌合 .....	299
15.32. 挿抜治具によるアンテナケーブルコネクタの抜去 .....	299
15.33. 無線 LAN 用 外付けアンテナ 08 形状図 .....	300
15.34. 無線 LAN 用 外付けアンテナケーブル 08 形状図 .....	300
15.35. 無線 LAN 用 外付けアンテナ 取り付け穴寸法 .....	300
15.36. 外付けアンテナ固定金具 00 の外観 .....	300
15.37. Armadillo-640 へのアンテナ固定金具の固定 .....	301
15.38. Armadillo-640 へのアンテナ固定金具の固定 .....	302
15.39. アンテナ固定金具へのアンテナの固定 .....	302
15.40. アンテナ固定金具 形状図 .....	303



# 表目次

1.1. 使用しているフォント .....	21
1.2. 表示プロンプトと実行環境の関係 .....	21
1.3. コマンド入力例での省略表記 .....	22
2.1. 推奨温湿度環境について .....	25
2.2. WLAN+BT コンボモジュール: Sterling LWB5+ 適合証明情報 .....	27
2.3. EYSKBNZWB 適合証明情報 .....	27
3.1. Armadillo-640 ラインアップ .....	33
3.2. 仕様 .....	34
3.3. eMMC メモリマップ .....	35
3.4. eMMC ブートパーティション構成 .....	36
3.5. eMMC GPP 構成 .....	36
4.1. ユーザー名とパスワード .....	41
4.2. 動作確認に使用する取り外し可能デバイス .....	43
4.3. シリアル通信設定 .....	45
4.4. インターフェース内容 .....	49
4.5. 入力モードに移行するコマンド .....	54
4.6. カーソルの移動コマンド .....	55
4.7. 文字の削除コマンド .....	55
4.8. 保存・終了コマンド .....	55
7.1. ネットワークとネットワークデバイス .....	62
7.2. 固定 IP アドレス設定例 .....	65
7.3. ストレージデバイス .....	67
7.4. eMMC の GPP の用途 .....	68
7.5. LED クラスディレクトリと LED の対応 .....	70
7.6. LED トリガの種類 .....	71
7.7. インプットデバイスファイルとイベントコード .....	72
10.1. CON9 ピンと GPIO の対応 .....	121
10.2. 対応するパワーマネジメント状態 .....	140
10.3. add_hotplugs オプションに指定できる主要な文字列 .....	146
10.4. build-rootfs のファイル説明 .....	159
10.5. microSD カードのパーティション構成 .....	163
10.6. u-boot の主要な環境変数 .....	200
10.7. thermal_profile.csv の各列の説明 .....	224
10.8. GPIO override と eFuse .....	230
10.9. ブートデバイスと eFuse .....	230
10.10. オンボード eMMC のスペック .....	232
12.1. キーコード .....	239
12.2. I2C デバイス .....	239
12.3. 対応するパワーマネジメント状態 .....	240
14.1. 絶対最大定格 .....	249
14.2. 推奨動作条件 .....	249
14.3. 入出力インターフェース(電源)の電氣的仕様 .....	249
14.4. 入出力インターフェースの電氣的仕様(OVDD = VCC_3.3V) .....	249
14.5. Armadillo-640 インターフェース一覧 .....	254
14.6. CON1 信号配列 .....	255
14.7. CON2 信号配列 .....	257
14.8. CON7 信号配列 .....	258
14.9. LAN LED の動作 .....	258
14.10. CON3 信号配列 .....	259
14.11. CON4 信号配列 .....	259

14.12. CON5 信号配列 .....	260
14.13. CON8 信号配列 .....	261
14.14. CON9 信号配列 .....	261
14.15. CON14 信号配列 .....	262
14.16. CON10 信号配列 .....	263
14.17. CON11 信号配列 .....	263
14.18. CON13 信号配列 .....	266
14.19. LED3、LED4、LED5 .....	267
14.20. SW1 信号配列 .....	267
14.21. ジャンパの設定と起動デバイス .....	267
14.22. JP1 信号配列 .....	268
14.23. JP2 信号配列 .....	268
15.1. Armadillo-640 関連のオプション品 .....	271
15.2. 各ピンに対応する UART コントローラ .....	272
15.3. USB シリアル変換アダプタのスライドスイッチによる起動モードの設定 .....	272
15.4. Armadillo-600 シリーズ オプションケース(樹脂製)について .....	273
15.5. Armadillo-600 シリーズ オプションケース(樹脂製)の仕様 .....	273
15.6. Armadillo-600 シリーズ オプションケース(金属製)について .....	277
15.7. LCD オプションセット(7 インチタッチパネル WVGA 液晶)について .....	279
15.8. LCD の仕様 .....	279
15.9. Armadillo-600 シリーズ RTC オプションモジュールについて .....	281
15.10. RTC オプションモジュールの仕様 .....	282
15.11. RTC オプションモジュール インターフェース一覧 .....	283
15.12. CON1 信号配列 .....	283
15.13. CON3 信号配列 .....	284
15.14. 対応バッテリー例 .....	284
15.15. CON4、CON5、CON6 信号配列 .....	284
15.16. Armadillo-640 WLAN コンボ、BT/TH オプションモジュールの搭載デバイス .....	288
15.17. WLAN コンボ、BT/TH オプションモジュールの同梱物 .....	288
15.18. 推奨外付けアンテナ .....	288
15.19. WLAN コンボ、BT/TH オプションモジュールの仕様 .....	289
15.20. WLAN コンボ、BT/TH オプションモジュール インターフェース一覧 .....	291
15.21. CON1 信号配列 .....	291
15.22. CON2 信号配列 .....	292
15.23. CON2 搭載コネクタ例 .....	292
15.24. 無線 LAN 用 外付けアンテナセット 08 について .....	297
15.25. 外付けアンテナ固定金具 00 について .....	301
15.26. 外付けアンテナ固定金具 00 の仕様 .....	301

# 1. はじめに

---

このたびは Armadillo-640 をご利用いただき、ありがとうございます。

Armadillo-640 は、NXP Semiconductors 製アプリケーションプロセッサ「i.MX6ULL」を採用し、標準インターフェースとして、USB2.0 ホストポートやイーサネットポート、microSD を搭載した小型シングルボードコンピュータです。

Armadillo-640 は、Armadillo-440 の形状を継承しつつ、処理能力や搭載メモリなどの基本機能を向上したモデルです。また、標準インターフェース以外に多くの拡張インターフェースを搭載しており、お客様の用途に合わせた柔軟な機能拡張に対応することができます。

Armadillo-640 には Linux ベースのディストリビューションとして専用設計の Armadillo Base OS を搭載しています。Armadillo Base OS はユーザーアプリケーションをコンテナとして管理する機能、Armadillo Base OS 自体とコンテナの両方を安全にリモートアップデートする機能、ネットワークや HW セキュリティに関する機能を集約したコンパクトな Armadillo 専用 OS です。

ユーザーアプリケーションは OCI 規格に準拠した Podman コンテナ内で動作するため、ライブラリの依存関係はコンテナ内に限定されます。コンテナ内では Debian Linux や Alpine Linux といった様々なディストリビューションをユーザーが自由に選択し、Armadillo Base OS とは無関係に動作環境を決定、維持することが可能です。また、コンテナ内からデバイスへのアクセスはデバイスファイル毎に決定することができるので、必要以上にセキュリティリスクを高めることなく装置を運用することが可能です。

Armadillo Base OS とユーザーアプリケーションを含むコンテナはどちらも、Armadillo Base OS のリモートアップデート機能で安全にアップデートすることができます。Armadillo Base OS はアップデートの状態を 2 面化しているため電源やネットワークの遮断によって中断してもアップデート前の状態に復旧します。

以降、本書では他の Armadillo ブランド製品にも共通する記述については、製品名を Armadillo と表記します。

## 1.1. 本書で扱うこと扱わないこと

### 1.1.1. 扱うこと

本書では、Armadillo-640 の使い方、製品仕様(ソフトウェアおよびハードウェア)、オリジナルの製品を開発するために必要となる情報、その他注意事項について記載しています。Linux あるいは組み込み機器に不慣れな方でも読み進められるよう、コマンドの実行例なども記載しています。

また、本書では、アットマークテクノが運営する Armadillo サイトをはじめ、開発に有用な情報を得る方法についても、随時説明しています。

### 1.1.2. 扱わないこと

本書では、一般的な Linux のプログラミング、デバッグ方法やツールの扱い方、各種モジュールの詳細仕様など、一般的な情報や、他に詳しい情報があるものは扱いません。また、(Armadillo-640 を使用した)最終製品あるいはサービスに固有な情報や知識も含まれていません。

## 1.2. 本書で必要となる知識と想定する読者

本書は、読者として Armadillo-640 を使ってオリジナルの機器を開発するエンジニアを想定して書かれています。また、「Armadillo-640 を使うと、どのようなことが実現可能なのか」を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-640 は組込みプラットフォームとして実績のある Armadillo をベースとしているため、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。

**ソフトウェアエンジニア** 端末からのコマンドの実行方法など、基本的な Linux の扱い方を知っているエンジニアを対象読者として想定しています。プログラミング言語として C/C++ を扱えることは必ずしも必要ではありませんが、基礎的な知識がある方が理解しやすい部分もあります。

**ハードウェアエンジニア** 電子工学の基礎知識を有したエンジニアを対象読者として想定しています。回路図や部品表を読み、理解できる必要があります。

## 1.3. ユーザー限定コンテンツ

アットマークテクノ Armadillo サイトで購入製品登録を行うと、製品をご購入いただいたユーザーに限定して公開している限定コンテンツにアクセスできるようになります。主な限定コンテンツには、下記のものがあります。

- ・ 各種信頼性試験データ・納入仕様書等製造関連情報

限定コンテンツを取得するには、「6. ユーザー登録」を参照してください。

## 1.4. 本書および関連ファイルのバージョンについて

本書を含めた関連マニュアル、ソースファイルやイメージファイルなどの関連ファイルは最新版を使用することをおすすめいたします。本書を読み始める前に、Armadillo サイトで最新版の情報をご確認ください。

**Armadillo サイト - Armadillo-640 ドキュメントダウンロード**

<https://armadillo.atmark-techno.com/armadillo-640/resources/documents>

**Armadillo サイト - Armadillo-640 ソフトウェアダウンロード**

<https://armadillo.atmark-techno.com/armadillo-640/resources/software>

## 1.5. 本書の構成

本書には、Armadillo-640 をベースに、オリジナルの製品を開発するために必要となる情報を記載しています。また、取扱いに注意が必要な事柄についても説明しています。

- ・ はじめにお読みください。
  - 「1. はじめに」、「2. 注意事項」
- ・ Armadillo-640 の仕様を紹介します。

- 「3. 製品概要」
- ・ ソフトウェアの使い方や、動作を確認する方法を紹介します。
- 「4. Armadillo の電源を入れる前に」、「5. 起動と終了」、「7. 動作確認方法」
- ・ ソフトウェア仕様について紹介します。
- 「11. 動作ログ」、「13. ソフトウェア仕様」
- ・ システム開発に必要な情報を紹介します。
- 「8. 開発の基本的な流れ」、「10. Howto」、「12. 製品機能」
- ・ 拡張基板の開発や、ハードウェアをカスタマイズする場合に必要な情報を紹介します。
- 「14. ハードウェア仕様」、「15. オプション品」
- ・ ご購入ユーザーに限定して公開している情報の紹介やユーザー登録について紹介します。
- 「6. ユーザー登録」

## 1.6. 表記について

### 1.6.1. フォント

本書では以下のような意味でフォントを使いわけています。

表 1.1 使用しているフォント

フォント例	説明
本文中のフォント	本文
[PC ~]\$ ls	プロンプトとユーザ入力文字列
text	編集する文字列や出力される文字列。またはコメント

### 1.6.2. 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 1.2 表示プロンプトと実行環境の関係

プロンプト	コマンドの実行環境
[PC ~/]#	作業用 PC の root ユーザで実行
[PC ~/\$	作業用 PC 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
[container ~/]#	Podman コンテナ内で実行
⇒	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 1.3 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

### 1.6.3. 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。

## 1.7. 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

### 2.1. 安全に関する注意事項

本製品を安全にご使用いただくために、特に以下の点にご注意ください。



- ・ ご使用の前に必ず製品マニュアルおよび関連資料をお読みにになり、使用上の注意を守って正しく安全にお使いください。
- ・ マニュアルに記載されていない操作・拡張などを行う場合は、弊社 Web サイトに掲載されている資料やその他技術情報を十分に理解した上で、お客様自身の責任で安全にお使いください。
- ・ 水・湿気・ほこり・油煙等の多い場所に設置しないでください。火災、故障、感電などの原因になる場合があります。
- ・ 本製品に搭載されている部品の一部は、発熱により高温になる場合があります。周囲温度や取扱いによってはやけどの原因となる恐れがあります。本体の電源が入っている間、または電源切断後本体の温度が下がるまでの間は、基板上の電子部品、及びその周辺部分には触れないでください。
- ・ 本製品を使用して、お客様の仕様による機器・システムを開発される場合は、製品マニュアルおよび関連資料、弊社 Web サイトで提供している技術情報のほか、関連するデバイスのデータシート等を熟読し、十分に理解した上で設計・開発を行ってください。また、信頼性および安全性を確保・維持するため、事前に十分な試験を実施してください。
- ・ 本製品は、機能・精度において極めて高い信頼性・安全性が必要とされる用途(医療機器、交通関連機器、燃焼制御、安全装置等)での使用を意図しておりません。これらの設備や機器またはシステム等に使用された場合において、人身事故、火災、損害等が発生した場合、当社はいかなる責任も負いかねます。
- ・ 本製品には、一般電子機器用(OA 機器・通信機器・計測機器・工作機械等)に製造された半導体部品を使用しています。外来ノイズやサージ等により誤作動や故障が発生する可能性があります。万一誤作動または故障などが発生した場合に備え、生命・身体・財産等が侵害されることのないよう、装置としての安全設計(リミットスイッチやヒューズ・ブレーカー等の保護回路の設置、装置の多重化等)に万全を期し、信頼性および安全性維持のための十分な措置を講じた上でお使いください。
- ・ 電池をご使用の際は、極性(プラスとマイナス)を逆にして装着しないでください。また、電池の使用推奨期限を過ぎた場合や RTC の時刻を保持できなくなった場合には、直ちに電池を交換してください。そ

のまま使用すると、電池が漏液、発熱、破裂したり、ケガや製品の故障の原因となります。万一、漏れた液が身体に付着した場合は多量の水で洗い流してください。

- 無線 LAN 機能を搭載した製品は、心臓ペースメーカーや補聴器などの医療機器、火災報知器や自動ドアなどの自動制御器、電子レンジ、高度な電子機器やテレビ・ラジオに近接する場所、移動体識別用の構内無線局および特定小電力無線局の近くで使用しないでください。製品が発生する電波によりこれらの機器の誤作動を招く恐れがあります。

## 2.2. 取扱い上の注意事項

本製品に恒久的なダメージをあたえないよう、取扱い時には以下のような点にご注意ください。

破損しやすい箇所	microSD コネクタおよびそのカバーやフラットケーブルコネクタは、破損しやすい部品になっています。無理に力を加えて破損することのないよう十分注意してください。
本製品の改造	本製品に改造 <sup>[1]</sup> を行った場合は保証対象外となりますので十分ご注意ください。また、改造やコネクタ等の増設 <sup>[2]</sup> を行う場合は、作業前に必ず動作確認を行ってください。
電源投入時のコネクタ着脱	本製品や周辺回路に電源が入っている状態で、活線挿抜対応インターフェース (LAN, USB) <sup>[3]</sup> 以外へのコネクタ着脱は、絶対に行わないでください。
静電気	本製品には CMOS デバイスを使用しており、静電気により破壊されるおそれがあります。本製品を開封するときは、低湿度状態にならないよう注意し、静電防止用マットの使用、導電靴や人体アースなどによる作業者の帯電防止対策、備品の放電対策、静電気対策を施された環境下で行ってください。また、本製品を保管する際は、静電気を帯びやすいビニール袋やプラスチック容器などは避け、導電袋や導電性の容器・ラックなどに収納してください。
ラッチアップ	電源および入出力からの過大なノイズやサージ、電源電圧の急激な変動等により、使用している CMOS デバイスがラッチアップを起こす可能性があります。いったんラッチアップ状態となると、電源を切断しないかぎりこの状態が維持されるため、デバイスの破損につながる可能性があります。ノイズの影響を受けやすい入出力ラインには、保護回路を入れることや、ノイズ源となる装置と共通の電源を使用しない等の対策をとることをお勧めします。
衝撃	落下や衝撃などの強い振動を与えないでください。
使用場所の制限	無線機能を搭載した製品は、テレビ・ラジオに近接する場所で使用すると、受信障害を招く恐れがあります。
振動	振動が発生する環境では、Armadillo が動かないよう固定して使用してください。

<sup>[1]</sup>本書を含めた関連マニュアルで改造方法を記載している箇所および、コネクタ非搭載箇所へのコネクタ等の増設は除く。

<sup>[2]</sup>改造やコネクタを増設する際にはマスキングを行い、周囲の部品に半田くず、半田ボール等付着しないよう十分にご注意ください。

<sup>[3]</sup>別途、活線挿抜を禁止している場合を除く

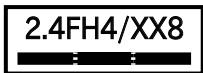


**電池の取り扱い** 電池の使用推奨期限を過ぎる前に電池の交換をしてください。使用推奨期限を超えて使用すると、電池の性能が充分に発揮できない場合や、電池を漏液させたり、製品を破損させるおそれがあります。

**電波に関する注意事項(2.4GHz 帯無線)** 2.4GHz 帯の電波を使用する機能(無線 LAN 等)は、自動ドアなどの自動制御電子機器に影響が出る場合、すぐに使用を中止してください。

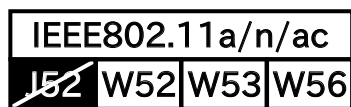


この無線機(Sterling LWB5+)は 2.4GHz 帯を使用します。全帯域を使用し、かつ移動体識別装置の帯域が回避可能です。変調方式として DS-SS および OFDM 方式を採用し、想定される与干渉距離は 40m 以下です。



この無線機(Sterling LWB5+)は 2.4GHz 帯を使用します。全帯域を使用し、かつ移動体識別装置の帯域が回避不可です。変調方式として FH-SS 方式を採用し、想定される与干渉距離は 40m 以下です。

**電波に関する注意事項(5GHz 帯無線)** この無線機(Sterling LWB5+)は 5GHz 帯を使用します。



W52、W53 の屋外での利用は電波法により禁じられています。W53、W56 での AP モードは、工事設計認証を受けていないため使用しないでください。

## 2.3. 製品の保管について



- ・ 製品を在庫として保管するときは、高温・多湿、埃の多い環境、水濡れの可能性のある場所、直射日光のあたる場所、有毒ガス(特に腐食性ガス)の発生する場所を避け、精密機器の保管に適した状態で保管してください。
- ・ 保管環境として推奨する温度・湿度条件は以下のとおりです。

表 2.1 推奨温湿度環境について

推奨温湿度環境	5~35°C/70%RH 以下 <sup>[a]</sup> <sup>[b]</sup>
---------	---

<sup>[a]</sup>半田付け作業を考慮した保管温度範囲となっております。半田付けを行わない、または、すべての半田付けが完了している場合の推奨温度・湿度条件は、製品の動作温度・湿度範囲となります。

<sup>[b]</sup>温度変化の少ない場所に保管してください。保管時の急激な温度変化は結露が生じ、金属部の酸化、腐食などが発生し、はんだ濡れ性に影響が出る場合があります。

- ・ 製品を包装から取り出した後に再び保管する場合は、帯電防止処理された収納容器を使用してください。

## 2.4. ソフトウェア使用に関する注意事項

本製品に含まれるソフトウェアについて

本製品の標準出荷状態でプリインストールされている Linux 対応ソフトウェアは、個別に明示されている（書面、電子データでの通知、口頭での通知を含む）場合を除き、オープンソースとしてソースコードが提供されています。再配布等の権利については、各ソースコードに記載のライセンス形態にしたがって、お客様の責任において行使してください。また、本製品に含まれるソフトウェア（付属のドキュメント等も含む）は、現状有姿（AS IS）にて提供します。お客様ご自身の責任において、使用用途・目的の適合について事前に十分な検討と試験を実施した上でお使いください。アットマークテクノは、当該ソフトウェアが特定の目的に適合すること、ソフトウェアの信頼性および正確性、ソフトウェアを含む本製品の使用による結果について、お客様に対し何らの保証も行いません。

パートナー等の協力により Armadillo ブランド製品向けに提供されているミドルウェア、その他各種ソフトウェアソリューションは、ソフトウェア毎にライセンスが規定されています。再頒布権等については、各ソフトウェアに付属する readme ファイル等をご参照ください。その他のバンドルソフトウェアについては、各提供元にお問い合わせください。



以下のソフトウェアは、オープンソースソフトウェアではありません。  
ボード情報取得ツール(get-board-info)

## 2.5. 電波障害について



この装置は、クラス A 情報技術装置です。この装置を住宅環境で使用すると電波妨害を引き起こすことがあります。この場合には使用者が適切な対策を講ずるよう要求されることがあります。VCCI-A

## 2.6. 無線モジュールの安全規制について

Armadillo-600 シリーズ WLAN コンボオプションモジュール、Armadillo-600 シリーズ BT/TH オプションモジュール WLAN コンボ対応に搭載している無線モジュールは、電波法に基づく工事設計認証を受けています。

これらの無線モジュールを国内で使用するときには無線局の免許は必要ありません。



以下の事項を行うと法律により罰せられることがあります。

- ・ 無線モジュールやアンテナを分解/改造すること。
- ・ 無線モジュールや筐体、基板等に直接印刷されている証明マーク・証明番号、または貼られている証明ラベルをはがす、消す、上からラベルを貼るなどし、見えない状態にすること。

認証番号は次のとおりです。

表 2.2 WLAN+BT コンボモジュール: Sterling LWB5+ 適合証明情報

項目	内容
型式又は名称	Sterling LWB5+
電波法に基づく工事設計認証における認証番号	201-200402

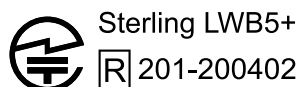


図 2.1 WLAN+BT コンボモジュール: Sterling LWB5+ 認証マーク

表 2.3 EYSKBNZWB 適合証明情報

項目	内容
型式又は名称	EYSKBN
電波法に基づく工事設計認証における認証番号	001-A14398

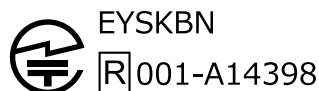


図 2.2 EYSKBNZWB 認証マーク

## 2.7. 保証について

本製品の本体基板は、製品に添付もしくは弊社 Web サイトに記載している「製品保証規定」に従い、ご購入から 1 年間の交換保証を行っています。添付品およびソフトウェアは保証対象外となりますのでご注意ください。

製品保証規定 <http://www.atmark-techno.com/support/warranty-policy>

## 2.8. 輸出について

- ・ 当社製品は、原則として日本国内での使用を想定して開発・製造されています。
- ・ 海外の法令および規則への適合については当社はなんらの保証を行うものではありません。
- ・ 当社製品を輸出するときは、輸出者の責任において、日本国および関係する諸外国の輸出関連法令に従い、必要な手続を行っていただきますようお願いいたします。
- ・ 日本国およびその他関係諸国による制裁または通商停止を受けている国家、組織、法人または個人に対し、当社製品を輸出、販売等することはできません。

- ・ 当社製品および関連技術は、大量破壊兵器の開発等の軍事目的、その他国内外の法令により製造・使用・販売・調達が禁止されている機器には使用することができません。

## 2.9. 商標について

- ・ 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。
- ・ SD、SDHC、SDXC、microSD、microSDHC、microSDXC、SDIO ロゴは SD-3C, LLC の商標です。



## 3. 製品概要

---

### 3.1. 製品の特長

#### 3.1.1. Armadillo とは

「Armadillo(アルマジロ)」は、ARM コアプロセッサ搭載・Linux 対応の組み込みプラットフォームのブランドです。Armadillo ブランド製品には以下の特長があります。

- ・ ARM プロセッサ搭載・省電力設計

ARM コアプロセッサを搭載しています。1～数ワット程度で動作する省電力設計で、発熱が少なくファンを必要としません。

- ・ 小型・手のひらサイズ

CPU ボードは名刺サイズ程度の手のひらサイズが主流です。名刺の 1/3 程度の小さな CPU モジュールや無線 LAN モジュール等、超小型のモジュールもラインアップしています。

- ・ 標準 OS として Linux をプリインストール

標準 OS に Linux を採用しており、豊富なソフトウェア資産と実績のある安定性を提供します。ソースコードをオープンソースとして公開しています。

- ・ 開発環境

Armadillo の開発環境として、「Atmark Techno Development Environment ATDE)」を無償で提供しています。ATDE は、VMware など仮想マシン向けのデータイメージです。このイメージには、Linux デスクトップ環境をベースに GNU クロス開発ツールやその他の必要なツールが事前にインストールされています。ATDE を使うことで、開発用 PC の用意やツールのインストールなどといった開発環境を整える手間を軽減することができます。

#### 3.1.2. Armadillo-640 とは

Armadillo-600 シリーズは、フィールド向けの機器・端末のプラットフォームとして豊富な採用実績を持つ小型・省電力で Linux 採用の組み込み CPU ボード「Armadillo-400 シリーズ」の思想を継承しつつ、処理能力と搭載メモリをともに大幅にグレードアップさせた、次世代の Linux 組み込みプラットフォームです。高性能ながら省電力性能を向上、さらに耐環境性を追求するなど、量産時の使いやすさを重視した堅実な設計が特長です。

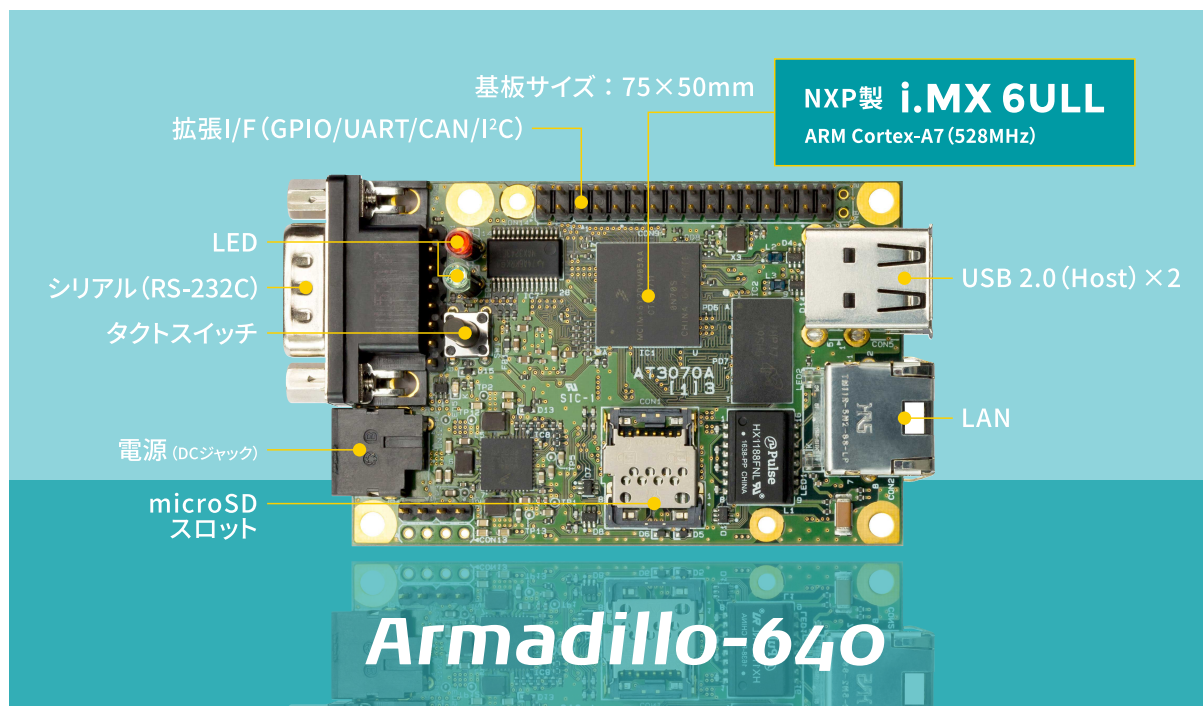


図 3.1 Armadillo-640 とは

- ・ i.MX6ULL 搭載/Armadillo-440 と形状互換で性能向上

Armadillo-640 は、従来の Armadillo-440 のコネクタ配置を踏襲したシングルボード型モデルです。CPU コアクロックは 528MHz にアップ、メモリは Armadillo-440 の約 4 倍の 512MB(DDR3-800)、オンボードストレージは約 4GB(eMMC)を搭載し、microSD カードスロットも備えています。従来の Armadillo-400 シリーズに比べてハードウェア性能が大幅に向上し、アプリケーション開発の自由度が高くなりました。Armadillo-440 向けと同じ形状のオプションケース(樹脂製・金属製)もラインアップしているので、Armadillo-440 から乗り換えるときも筐体を新規設計する必要がありません。

- ・ 省電力モード搭載・バッテリー駆動の機器にも最適

省電力モードを搭載し、「アプリケーションから Armadillo-640 本体の電源を OFF にする」「RTC(リアルタイムクロック)のアラームで決まった時間に本体の電源を ON にする」といった細かな電源制御が可能です。必要な時だけ本体を起動するという運用が可能なので、バッテリーで稼働させるような機器にも適しています。

- ・ 使用温度範囲-20°C~+70°C対応

Armadillo-640 は使用温度範囲-20°C~+70°Cをカバーしています。

- ・ シングルボード型ながら拡張性にも十分に配慮

Armadillo-640 は、シングルボード型ながら多くの拡張インターフェースを搭載しており、USB、LCD、シリアル、GPIO、I2C、I2S、SPI などの拡張に対応します。量産向けに、リード部品コネクタを搭載したモデルの他、リード部品非搭載のモデルも提供する予定です。

- ・ Armadillo Base OS 搭載

「Armadillo Base OS」を搭載しています。ユーザー自身がボードの機能を自由に設計・開発して書き込むことで、多様な製品を作ることができます。

### 3.1.3. Armadillo Base OS とは

Armadillo Base OS は、アットマークテクノが提供する専用ディストリビューションです。Linux5.10をベースに、コンテナ管理機能、ソフトウェアアップデート機能、ネットワークマネージャーなどに対応。機能を限定したコンパクトな OS で、安全性の高い運用を実現します。

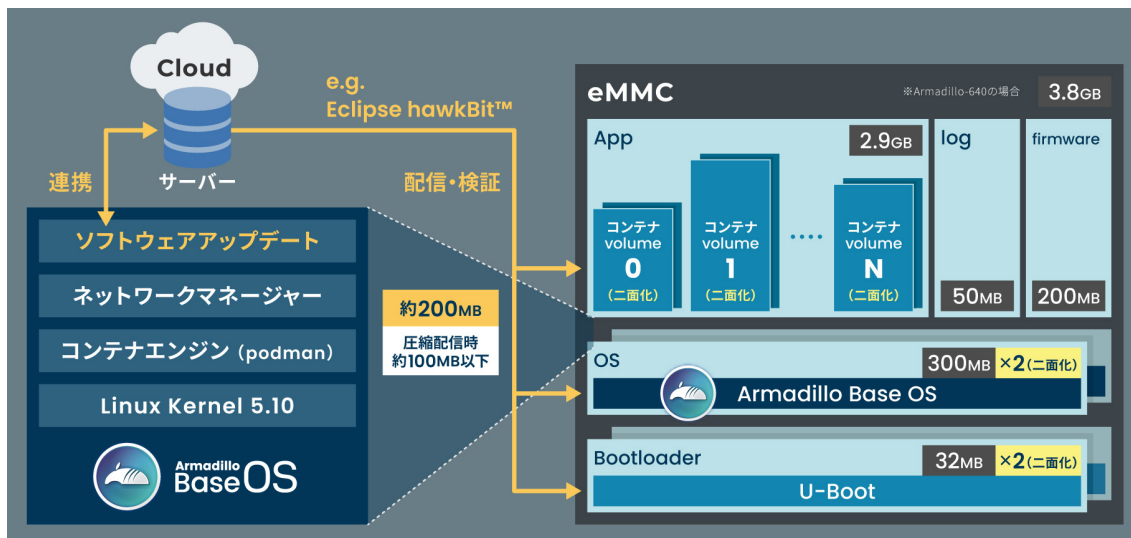


図 3.2 Armadillo Base OS とは

- ・ OS のコンパクト化

OS 基盤の機能を最小限にしたことで、セキュリティリスクを低減しています。アットマークテクノが継続的にアップデートを提供するため、高セキュリティな IoT 機器として長期間に渡り運用することができます。

- ・ コンテナによるアプリケーション運用

アプリケーションを「コンテナ」単位で OS から分離して管理できるため、コンテナごとのアップデートが可能です。サンドボックス化されることにより、悪意あるソフトウェアからの攻撃に対する機器全体の保護に有効性を発揮します。

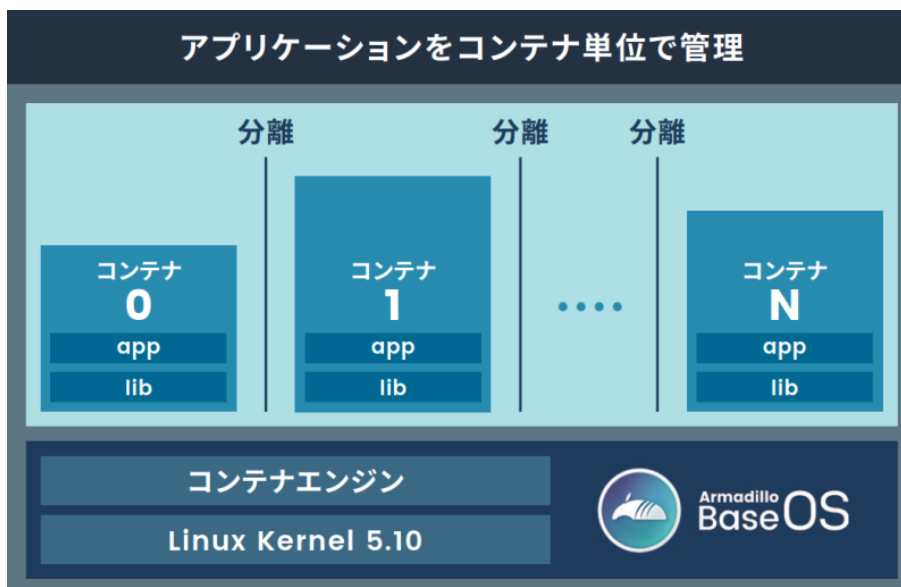


図 3.3 コンテナによるアプリケーションの運用

- ・アップデート機能を標準搭載

ネットワークや USB メモリ、microSD カードによるアップデート機能を標準搭載しています。正しく署名されたソフトウェアのみアップデートできる仕組みや、差分アップデート機能も用意されています。OS・ブートローダー・コンテナ部分は、安全性を担保するため二面化し、リカバリー機能を備えています。万が一アップデートに失敗した場合でも、作業前の状態にロールバックすることができます。

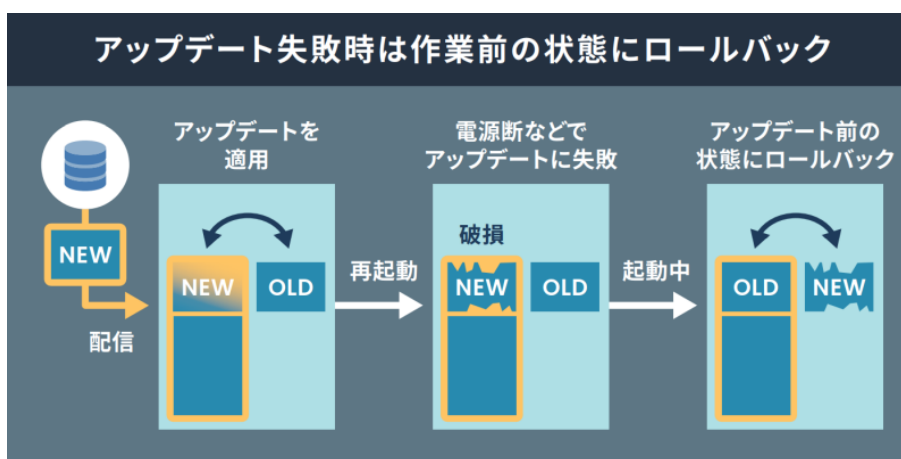


図 3.4 ロールバックの仕組み

- ・堅牢性の向上

安定性の高いファイルシステムで、ストレージへの書き込みを減らして消費を抑制するなど、高い堅牢性を有します。運用ログの記録機能も標準搭載しています。

- ・セキュリティ機能の向上

コンテナにアクセス権限を設けて管理することができます。



## 3.2. 製品ラインアップ

Armadillo-640 の製品ラインアップは次のとおりです。

表 3.1 Armadillo-640 ラインアップ

名称	型番
Armadillo-640 ベーシックモデル開発セット	A6400-D00Z
Armadillo-640 量産ボード	A6400-U00Z
Armadillo-640 量産ボード(リード部品未実装・部品付)	A6400-B00Z
Armadillo-640 量産ボード(リード部品未実装)	A6400-N00Z

### 3.2.1. Armadillo-640 ベーシックモデル開発セット

Armadillo-640 ベーシックモデル開発セット(型番: A6400-D00Z)は、Armadillo-640 を使った開発がすぐに開始できるように、開発に必要なものを一式含んだセットです。

- ・ Armadillo-640
- ・ Armadillo-600 シリーズオプションケース(樹脂製)
- ・ USB(A オス-miniB)ケーブル
- ・ USB シリアル変換アダプタ <sup>[1]</sup>
- ・ AC アダプタ(5V/2.0A, EIAJ#2 準拠)
- ・ ジャンパソケット
- ・ ねじ
- ・ ゴム足
- ・ スペーサ

### 3.2.2. Armadillo-640 量産ボード

Armadillo-640 量産ボードは、Armadillo-640 ベーシックモデル開発セットのセット内容を必要最小限に絞った量産向けのラインアップです。リード部品実装(A6400-U00Z)、リード部品未実装、部品付(型番: A6400-B00Z)、リード部品未実装(型番: A6400-N00Z)の 3 種類あります。

## 3.3. 仕様

Armadillo-640 の主な仕様は次のとおりです。

<sup>[1]</sup>Armadillo-800 シリーズ、Armadillo-IoT シリーズなどで使用していた USB シリアル変換アダプタ(型番: SA-SCUSB-00)とはコネクタ形状が異なりますので、ご注意ください。

表 3.2 仕様

プロセッサ	NXP Semiconductors i.MX6ULL ARM Cortex-A7 x 1 ・命令/データキャッシュ 32KByte/32KByte ・L2 キャッシュ 128KByte ・内部 SRAM 128KByte ・メディアプロセッシングエンジン(NEON)搭載 ・Thumb code(16bit 命令セット)サポート
システムクロック	CPU コアクロック(ARM Cortex-A7): 528MHz DDR クロック: 396MHz 源発振クロック: 32.768kHz, 24MHz
RAM	DDR3L: 512MByte バス幅: 16bit
ROM	eMMC: 約 3.8GB(約 3.6GiB)
LAN(Ethernet)	100BASE-TX/10BASE-T x 1 AUTO-MDIX 対応
シリアル(UART)	RS232C レベル x 1 3.3V CMOS レベル 最大 6 ポート拡張可能 <sup>[a]</sup>
USB	USB 2.0 Host x 2
SD	microSD スロット x 1
ビデオ	LCD インターフェース 最大 1 ポート拡張可能 <sup>[a]</sup> 最大解像度: WXGA (1366 x 768), 18bpp タッチパネル対応可能
オーディオ	I2S 最大 3 ポート拡張可能 <sup>[a]</sup> S/PDIF 最大 1 ポート拡張可能 <sup>[a]</sup>
GPIO	最大 61 bit 拡張可能 <sup>[a]</sup>
I2C	最大 3 ポート拡張可能 <sup>[a]</sup>
SPI	最大 4 ポート拡張可能 <sup>[a]</sup>
CAN	最大 2 ポート拡張可能 <sup>[a]</sup>
PWM	最大 8 ポート拡張可能 <sup>[a]</sup>
カレンダー時計	SoC 内蔵リアルタイムクロック <sup>[b]</sup> I2C 拡張可能
スイッチ	ユーザースイッチ x 1
LED	ユーザ LED x 3
電源電圧	DC 5V±5%
消費電力(参考値)	約 1.1W(待機時)、約 1.5W(LAN 通信時) <sup>[c]</sup>
使用温度範囲	-20~+70°C(結露なきこと) <sup>[d]</sup>
外形サイズ	75×50mm(突起部を除く)

<sup>[a]</sup>i.MX6ULL のピンマルチプレスの設定で、優先的に機能を割り当てた場合に拡張可能な最大数を記載しています。

<sup>[b]</sup>電池は付属していません。

<sup>[c]</sup>外部接続機器の消費分は含みません。

<sup>[d]</sup>Armadillo-600 シリーズ オプションケースセット(樹脂製)を組み合わせた場合の使用温度範囲は-10°C~+50°Cです。

### 3.4. ブロック図

Armadillo-640 のブロック図は次のとおりです。

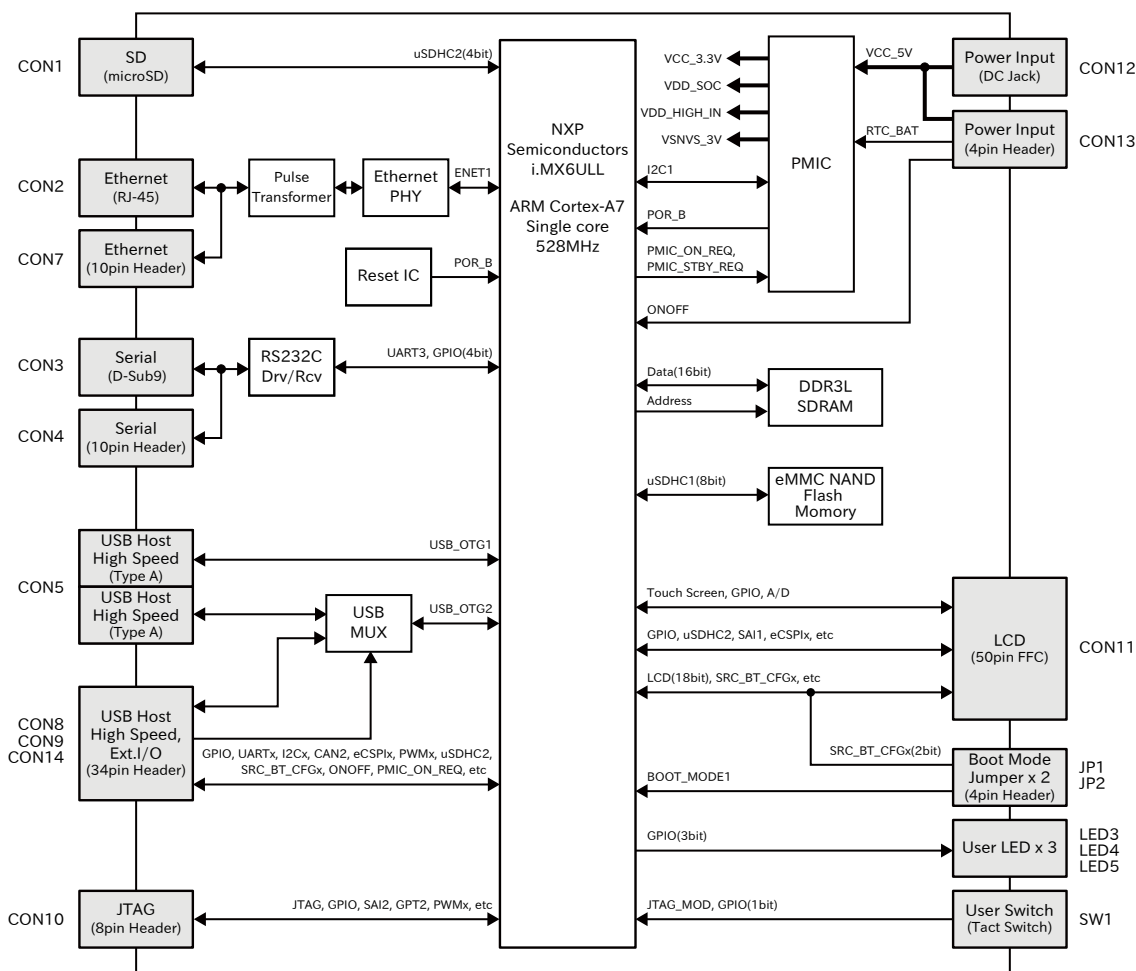


図 3.5 Armadillo-640 ブロック図

### 3.5. ストレージデバイスのパーティション構成

Armadillo-640 の eMMC のパーティション構成を「表 3.3. eMMC メモリマップ」に示します。

表 3.3 eMMC メモリマップ

パーティション	サイズ	ラベル	説明
1	300MiB	rootfs_0	A/B アップデートの A 面パーティション(Linux カーネルイメージ, Device Tree Blob, Alpine Linux rootfs を含む)
2	300MiB	rootfs_1	A/B アップデートの B 面パーティション(Linux カーネルイメージ, Device Tree Blob, Alpine Linux rootfs を含む)
3	50MiB	logs	ログ書き込み用パーティション
4	200MiB	firm	ファームウェア用パーティション
5	2.7GiB	app	アプリケーション用パーティション

Armadillo-640 の eMMC のブートパーティションの構成を「表 3.4. eMMC ブートパーティション構成」に示します。

表 3.4 eMMC ブートパーティション構成

ディスクデバイス	サイズ	説明
/dev/mmcblk0boot0	31.5 MiB	A/B アップデートの A 面
/dev/mmcblk0boot1	31.5 MiB	A/B アップデートの B 面

Armadillo-640 の eMMC の GPP(General Purpose Partition)の構成を「表 3.5. eMMC GPP 構成」に示します。

表 3.5 eMMC GPP 構成

ディスクデバイス	サイズ	説明
/dev/mmcblk0gp0	8 MiB	ライセンス情報等の為の予約領域
/dev/mmcblk0gp1	8 MiB	動作ログ領域
/dev/mmcblk0gp2	8 MiB	動作ログ予備領域 <sup>[a]</sup>
/dev/mmcblk0gp3	8 MiB	ユーザー領域

<sup>[a]</sup>詳細は「11.4. ログ用パーティションについて」を参照ください。

## 4. Armadillo の電源を入れる前に

### 4.1. 準備するもの

Armadillo を使用する前に、次のものを必要に応じて準備してください。

作業用 PC	Linux または Windows が動作し、ネットワークインターフェースと 1 つ以上の USB ポートを持つ PC です。「開発/動作確認環境の構築」を参照して、作業用 PC 上に開発/動作確認環境を構築してください。
ネットワーク環境	Armadillo と作業用 PC をネットワーク通信ができるようにしてください。
microSD カード	microSD スロットの動作を確認する場合などに利用します。
USB メモリ	USB の動作を確認する場合などに利用します。
tar.xz 形式のファイルを展開するソフトウェア	開発/動作確認環境を構築するために利用します。Linux では、tar で展開できます。Windows では、7-Zip や Lhaz などが対応しています。

### 4.2. 開発/動作確認環境の構築

アットマークテクノ製品のソフトウェア開発や動作確認を簡単に行うために、VMware 仮想マシンのデータイメージを提供しています。この VMware 仮想マシンのデータイメージを ATDE (Atmark Techno Development Environment) と呼びます。ATDE の起動には仮想化ソフトウェアである VMware を使用します。ATDE のデータは、tar.xz 圧縮されています。環境に合わせたツールで展開してください。



仮想化ソフトウェアとして、VMware の他に Oracle VM VirtualBox が有名です。Oracle VM VirtualBox には以下の特徴があります。

- ・ GPL v2 (General Public License version 2) で提供されている <sup>[1]</sup>
- ・ VMware 形式の仮想ディスク (.vmdk) ファイルに対応している

Oracle VM VirtualBox から ATDE を起動し、ソフトウェア開発環境として使用することができます。

ATDE は、バージョンにより対応するアットマークテクノ製品が異なります。本製品に対応している ATDE は、ATDE9 の v20230328 以降です。

ATDE9 は Debian GNU/Linux 11 (コードネーム bullseye) をベースに、Armadillo-640 のソフトウェア開発を行うために必要なクロス開発ツールや、Armadillo-640 の動作確認を行うために必要なツールが事前にインストールされています。

<sup>[1]</sup>バージョン 3.x までは PUEL (VirtualBox Personal Use and Evaluation License) が適用されている場合があります。

## 4.2.1. ATDE のセットアップ

### 4.2.1.1. VMware のインストール

ATDE を使用するためには、作業用 PC に VMware がインストールされている必要があります。VMware 社 Web ページ(<http://www.vmware.com/>)を参照し、利用目的に合う VMware 製品をインストールしてください。また、ATDE のアーカイブは tar.xz 圧縮されていますので、環境に合わせたツールで展開してください。



VMware は、非商用利用限定で無償のものから、商用利用可能な有償のものまで複数の製品があります。製品ごとに異なるライセンス、エンドユーザー使用許諾契約書(EULA)が存在するため、十分に確認した上で利用目的に合う製品をご利用ください。



VMware や ATDE が動作しないことを未然に防ぐため、使用する VMware のドキュメントから以下の項目についてご確認ください。

- ・ ホストシステムのハードウェア要件
- ・ ホストシステムのソフトウェア要件
- ・ ゲスト OS のプロセッサ要件

VMware のドキュメントは、VMware 社 Web ページ (<http://www.vmware.com/>)から取得することができます。

### 4.2.1.2. ATDE のアーカイブを取得

ATDE のアーカイブは Armadillo サイト(<http://armadillo.atmark-techno.com>)から取得可能です。



本製品に対応している ATDE のバージョンは ATDE9 v20230328 以降です。



作業用 PC の動作環境(ハードウェア、VMware、ATDE の対応アーキテクチャなど)により、ATDE が正常に動作しない可能性があります。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動作環境を確認してください。

### 4.2.1.3. ATDE のアーカイブを展開

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「4.2.1.4. Windows で ATDE のアーカイブ展開する」に、Linux での展開方法を手順「4.2.1.5. Linux で tar.xz 形式のファイルを展開する」に示します。

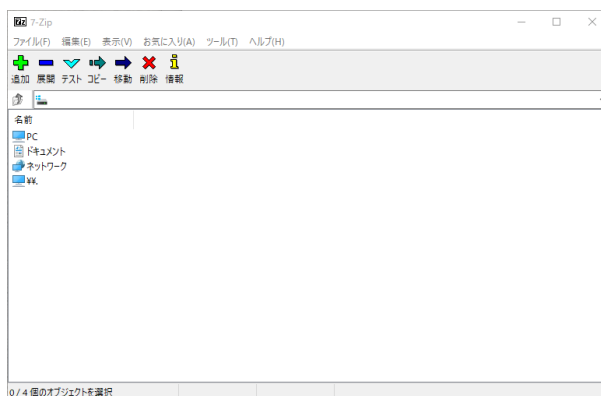
### 4.2.1.4. Windows で ATDE のアーカイブ展開する

#### 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://sevenzzip.sourceforge.jp>)からダウンロード取得可能です。

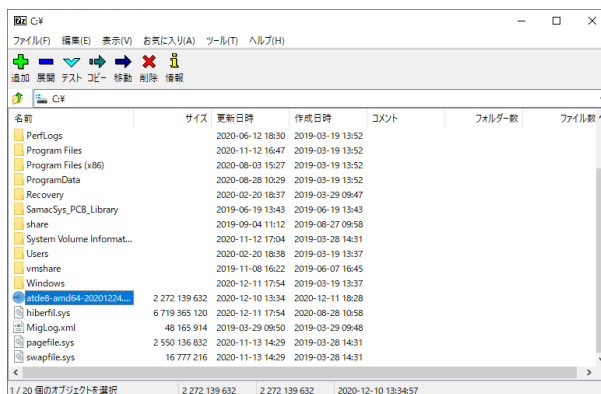
#### 2. 7-Zip の起動

7-Zip を起動します。



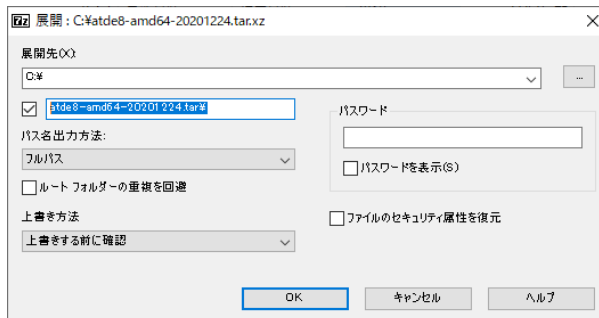
#### 3. xz 圧縮ファイルの選択

xz 圧縮ファイルを展開して、tar 形式のファイルを出力します。tar.xz 形式のファイルを選択して、「展開」をクリックします。



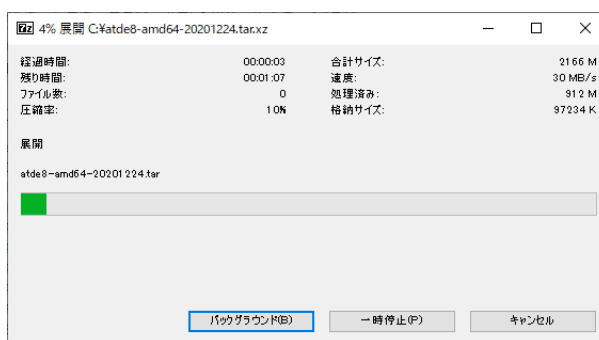
#### 4. xz 圧縮ファイルの展開先の指定

「展開先」を指定して、「OK」をクリックします。



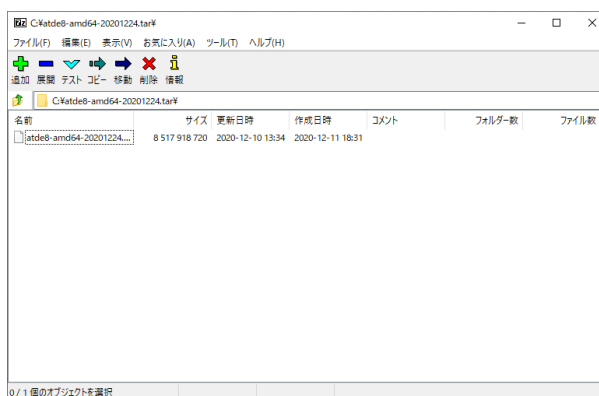
### 5. xz 圧縮ファイルの展開

展開が始まります。



### 6. tar アーカイブファイルの選択

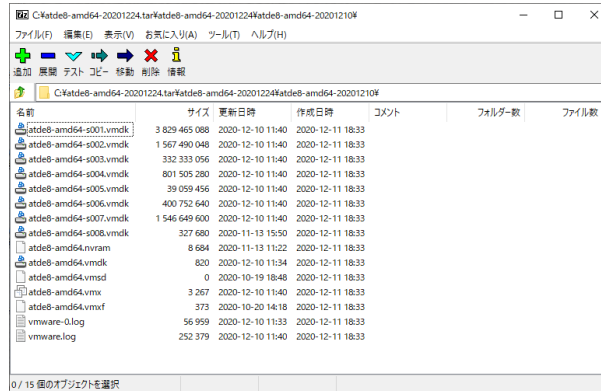
xz 圧縮ファイルの展開が終了すると、tar 形式のファイルが出力されます。tar アーカイブファイルを出力したのと同様の手順で、tar アーカイブファイルから ATDE のデータイメージを出力します。tar 形式のファイルを選択して「展開」をクリックし、「展開先」を指定して、「OK」をクリックします。



### 7. 展開の完了確認

tar アーカイブ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。「展開先」に指定したフォルダに ATDE のデータイメージが出力されています。





### 4.2.1.5. Linux で tar.xz 形式のファイルを展開する

#### 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 atde9-amd64-[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。 **atde9-amd64-[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 atde9-amd64-[VERSION]/
atde9-amd64-s001.vmdk  atde9-amd64-s008.vmdk
atde9-amd64-s002.vmdk  atde9-amd64-s009.vmdk
atde9-amd64-s003.vmdk  atde9-amd64.nvram
atde9-amd64-s004.vmdk  atde9-amd64.vmdk
atde9-amd64-s005.vmdk  atde9-amd64.vmsd
atde9-amd64-s006.vmdk  atde9-amd64.vmx
atde9-amd64-s007.vmdk  atde9-amd64.vmx
```

### 4.2.1.6. ATDE の起動

ATDE のアーカイブを展開したディレクトリに存在する仮想マシン構成(.vmx)ファイルを VMware 上で開くと、ATDE を起動することができます。ATDE9 にログイン可能なユーザーを、「表 4.1. ユーザー名とパスワード」に示します [2]。

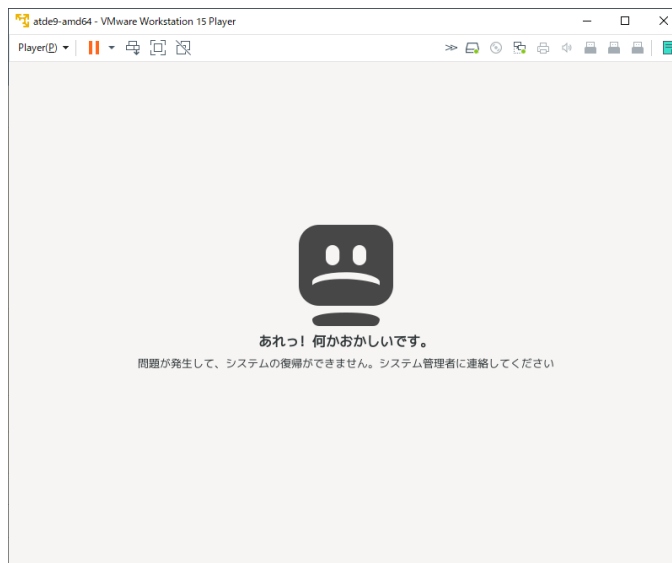
表 4.1 ユーザー名とパスワード

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



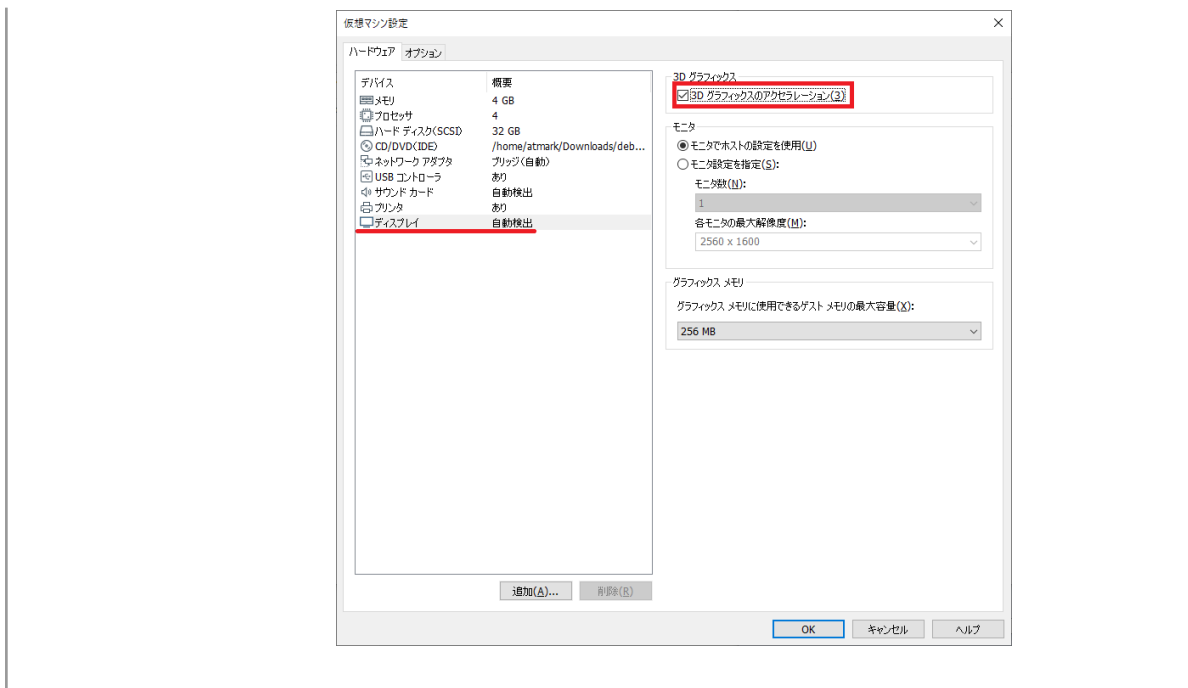
ATDE を起動する環境によっては、GUI ログイン画面が表示されずに以下のようなエラー画面が表示される場合があります。


[2]特権ユーザーで GUI ログインを行うことはできません



この場合は、VMware の設定で「3D グラフィックスのアクセラレーション」を ON にした後、ATDE を起動すると正常に GUI ログイン画面が表示されます。設定箇所を以下に示します。








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。

### 4.2.2. 取り外し可能デバイスの使用

VMware は、ゲスト OS (ATDE)による取り外し可能デバイス(USB デバイスや DVD など)の使用をサポートしています。デバイスによっては、ホスト OS (VMware を起動している OS)とゲスト OS で同時に使用することができません。そのようなデバイスをゲスト OS で使用するためには、ゲスト OS にデバイスを接続する操作が必要になります。



取り外し可能デバイスの使用方法については、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。

Armadillo-640 の動作確認を行うためには、「表 4.2. 動作確認に使用する取り外し可能デバイス」に示すデバイスをゲスト OS に接続する必要があります。

表 4.2 動作確認に使用する取り外し可能デバイス

デバイス	デバイス名
USB シリアル変換アダプタ	Future Devices FT232R USB UART
作業用 PC の物理シリアルポート	シリアルポート

### 4.2.3. コマンドライン端末(GNOME 端末)の起動

ATDE で、CUI (Character-based User Interface)環境を提供するコマンドライン端末を起動します。ATDE で実行する各種コマンドはコマンドライン端末に入力し、実行します。コマンドライン端末にはいくつかの種類がありますが、ここでは GNOME デスクトップ環境に標準インストールされている GNOME 端末を起動します。

GNOME 端末を起動するには、「図 4.1. GNOME 端末の起動」のようにデスクトップ左上のアプリケーションの「ユーティリティ」カテゴリから「端末」を選択してください。

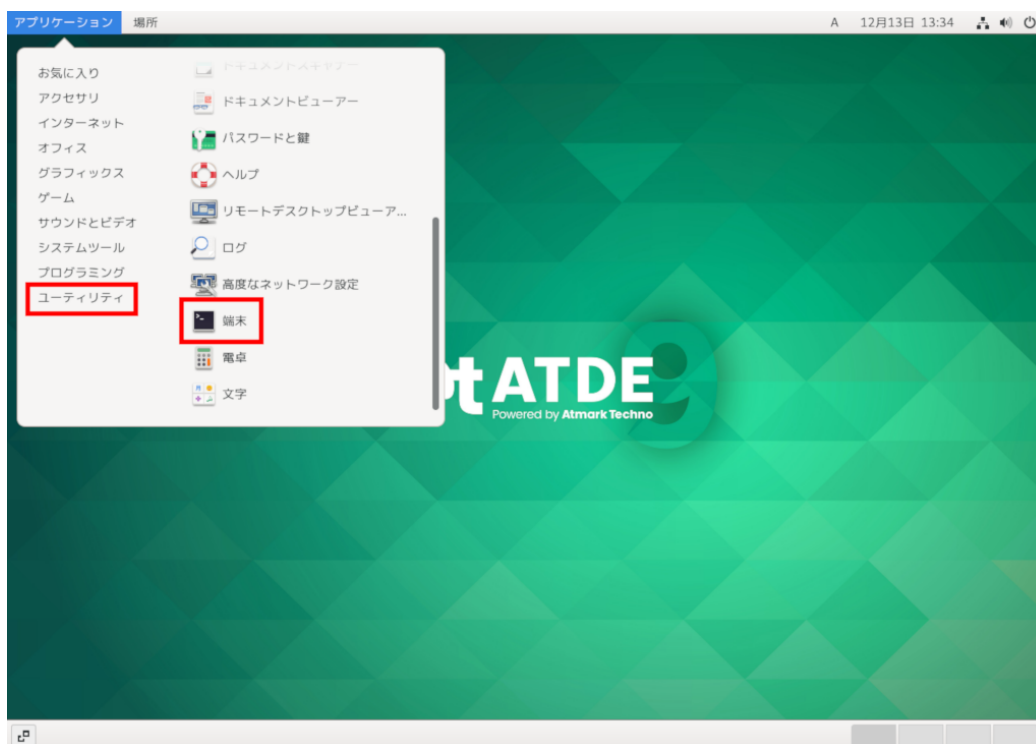


図 4.1 GNOME 端末の起動

「図 4.2. GNOME 端末のウィンドウ」のようにウィンドウが開きます。

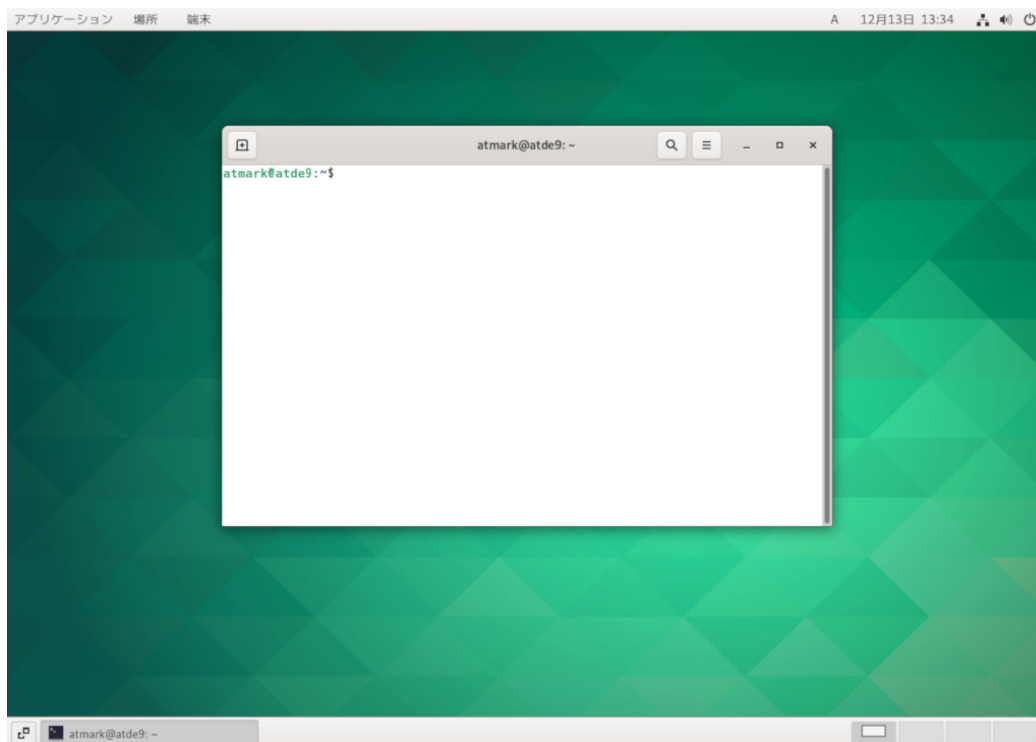


図 4.2 GNOME 端末のウィンドウ

#### 4.2.4. シリアル通信ソフトウェア(minicom)の使用

シリアル通信ソフトウェア(minicom)のシリアル通信設定を、「表 4.3. シリアル通信設定」のように設定します。また、minicom を起動する端末の横幅を 80 文字以上にしてください。横幅が 80 文字より小さい場合、コマンド入力中に表示が乱れることがあります。

表 4.3 シリアル通信設定

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

1. 「図 4.3. 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。

```
[ATDE ~]$ sudo LANG=C minicom --setup
```

図 4.3 minicom の設定の起動

2. 「図 4.4. minicom の設定」が表示されますので、「Serial port setup」を選択してください。

```
+-----[configuration]-----+
| Filenames and paths      |
| File transfer protocols  |
```

```

Serial port setup
Modem and dialing
Screen and keyboard
Save setup as dfl
Save setup as..
Exit
Exit from Minicom
    
```

図 4.4 minicom の設定

- 「図 4.5. 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 4.5 minicom のシリアルポートの設定

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



### USB to シリアル変換ケーブル使用時のデバイスファ イル確認方法

Linux で USB to シリアル変換ケーブルを接続した場合、コンソールに以下のようなログが表示されます。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 4.6 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 4.7. 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。

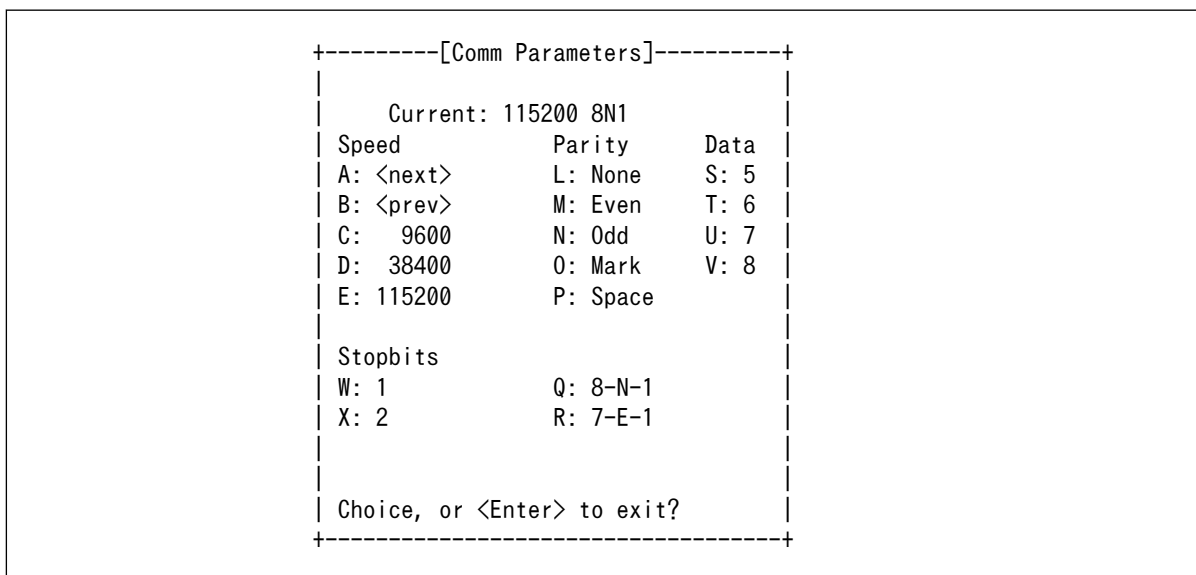


図 4.7 minicom のシリアルポートのパラメータの設定

8. 「図 4.7. 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 4.8. minicom シリアルポートの設定値」を参照してください。

Current: 115200 8 N 1  
転送レート      データ長      ストップビット  
パリティ

図 4.8 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 4.4. minicom の設定」に戻ってください。
13. 「図 4.4. 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 4.9. minicom 起動方法」のようにしてください。

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

図 4.9 minicom 起動方法



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。

```
[ATDE ~]$ sudo usermod -aG dialout atmark
[ATDE ~]$ LANG=C minicom --wrap --device /dev/ttyUSB0
```

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 4.10 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。



### 4.3. インターフェースレイアウト

Armadillo-640 のインターフェースレイアウトです。各インターフェースの配置場所等を確認してください。

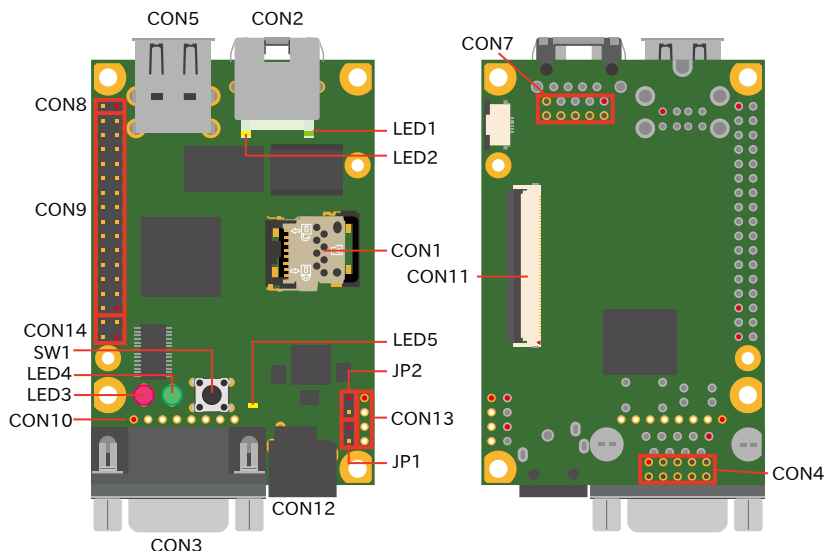


図 4.11 インターフェースレイアウト

表 4.4 インターフェース内容

部品番号	インターフェース名	形状	備考
CON1	SD インターフェース	microSD スロット (ヒンジタイプ)	
CON2	LAN インターフェース	RJ-45 コネクタ	
CON3	シリアルインターフェース	D-Sub9 ピン(オス)	
CON4	シリアルインターフェース	ピンヘッダ 10 ピン(2.54mm ピッチ)	コネクタ非搭載
CON5	USB インターフェース	Type-A コネクタ(2ポートスタック)	
CON7	LAN インターフェース	ピンヘッダ 10 ピン(2.54mm ピッチ)	コネクタ非搭載
CON8	拡張インターフェース	ピンヘッダ 2 ピン(2.54mm ピッチ)	
CON9	拡張インターフェース	ピンヘッダ 28 ピン(2.54mm ピッチ)	
CON10	JTAG インターフェース	ピンヘッダ 8 ピン(2.54mm ピッチ)	コネクタ非搭載
CON11	LCD 拡張インターフェース	FFC コネクタ 50 ピン(0.5mm ピッチ)	挿抜寿命: 20 回 <sup>[a]</sup>
CON12	電源入力インターフェース	DC ジャック	対応プラグ: EIAJ#2
CON13	電源入力インターフェース	ピンヘッダ 4 ピン(2.54mm ピッチ)	コネクタ非搭載
CON14	拡張インターフェース	ピンヘッダ 4 ピン(2.54mm ピッチ)	
JP1	起動デバイス設定ジャンパ	ピンヘッダ 2 ピン(2.54mm ピッチ)	
JP2	起動デバイス設定ジャンパ	ピンヘッダ 2 ピン(2.54mm ピッチ)	
LED1	LAN スピード LED	LED(緑色,面実装)	
LED2	LAN リンクアクティビティ LED	LED(黄色,面実装)	
LED3	ユーザー LED(赤)	LED(赤色, φ3mm)	
LED4	ユーザー LED(緑)	LED(緑色, φ3mm)	
LED5	ユーザー LED(黄)	LED(黄色,面実装)	
SW1	ユーザースイッチ	タクトスイッチ(h=17mm)	

<sup>[a]</sup>挿抜寿命は製品出荷時における目安であり、実際の挿抜可能な回数を保証するものではありません。

### 4.4. 接続方法

Armadillo-640 と周辺装置の接続例を「図 4.12. Armadillo-640 の接続例」に示します。

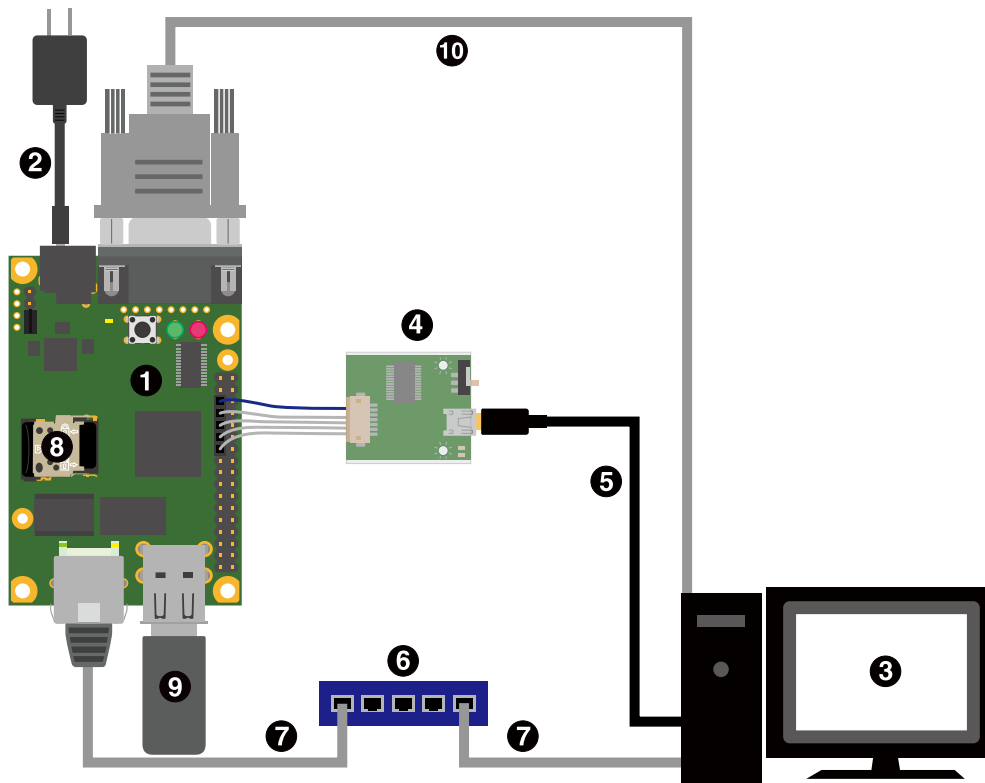


図 4.12 Armadillo-640 の接続例

- ❶ Armadillo-640
- ❷ AC アダプタ (5V/2A)
- ❸ 作業用 PC
- ❹ USB シリアル変換アダプタ
- ❺ USB2.0 ケーブル(A-miniB タイプ)
- ❻ LAN HUB
- ❼ Ethernet ケーブル
- ❽ microSD カード
- ❾ USB メモリ
- ❿ シリアルクロスケーブル



作業用 PC が Windows の場合、一部の Bluetooth デバイスドライバが USB コンソールインターフェースと同じポート番号の COM を重複して取得し、USB コンソールインターフェースが利用できないことがあります。

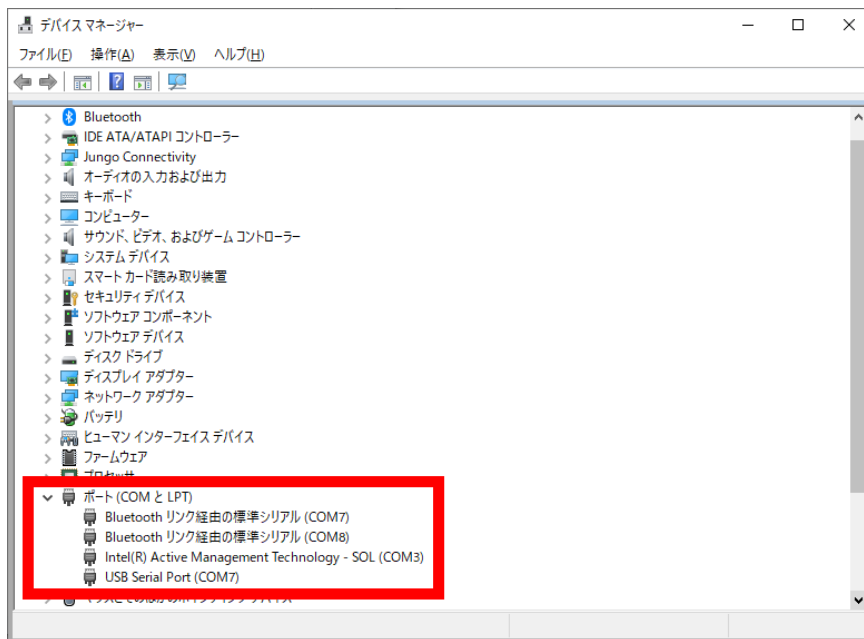


図 4.13 COM7 が競合している状態

この場合は、デバイスマネージャーから Bluetooth のデバイスを選択して「ポートの設定→詳細設定」から COM の番号を変更するか、Bluetooth デバイスを無効にしてください。

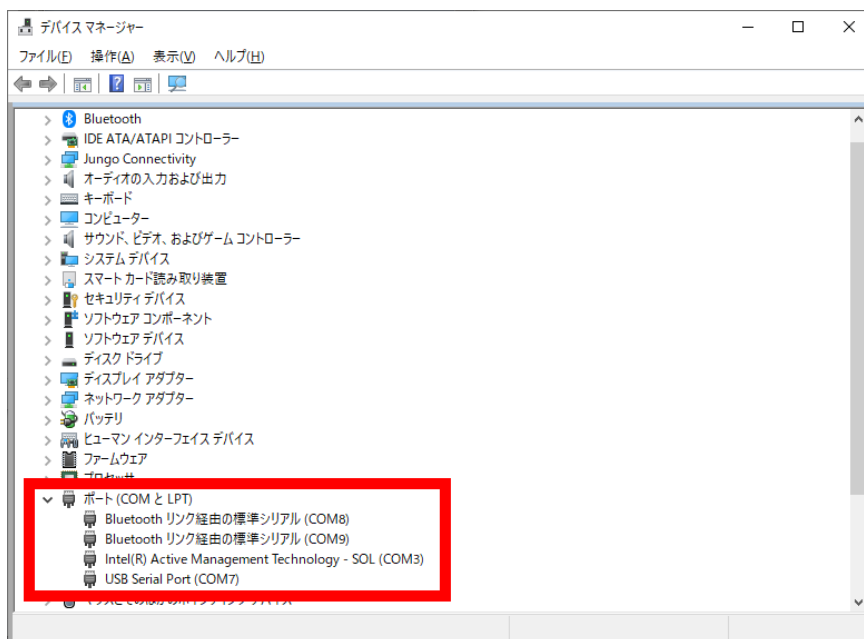


図 4.14 Bluetooth に割当の COM を変更した状態

仮想マシンである ATDE に USB コンソールインターフェースデバイスを接続する場合は、この影響はありません。

### 4.4.1. 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をそれを目印にして、下図のように接続してください。

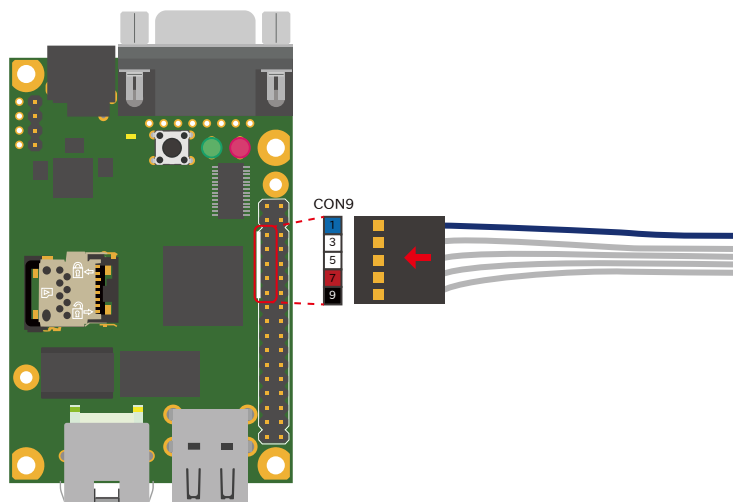


図 4.15 CON9-USB シリアル変換アダプタ接続図

## 4.5. ジャンパピンの設定について

ジャンパの設定を変更することで、Armadillo-640 の動作を変更することができます。ジャンパの機能については「14.2.12. JP1、JP2(起動デバイス設定ジャンパ)」を参照してください。

ジャンパピンの位置は「図 4.16. JP1、JP2 の位置」で確認してください。

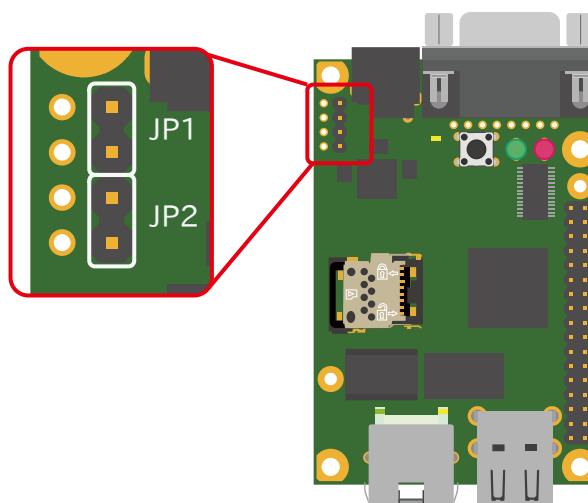




図 4.16 JP1、JP2 の位置


各ジャンパは必要に応じて切り替えの指示があります。ここでは、JP1 をオープン、JP2 をショートに設定しておきます。



### 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。

## 4.6. スライドスイッチの設定について

USB シリアル変換アダプタのスライドスイッチを操作することで、ブートローダーの起動モードを変更することができます。

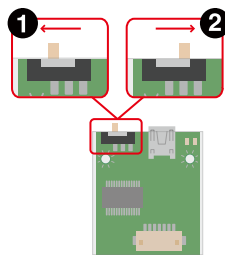



図 4.17 スライドスイッチの設定

- 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。



USB シリアル変換アダプタが未接続の場合オートブートモードとなり、Linux が起動します。

## 4.7. vi エディタの使用方法

vi エディタは、Armadillo に標準でインストールされているテキストエディタです。本書では、Armadillo の設定ファイルの編集などに vi エディタを使用します。

vi エディタは、ATDE にインストールされてる gedit や emacs などのテキストエディタとは異なり、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。入力モードでは文字の入力ができます。

本章で示すコマンド例は ATDE で実行するよう記載していますが、Armadillo でも同じように実行することができます。

### 4.7.1. 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。

[ATDE ~]# vi [file]

図 4.18 vi の起動

file にファイル名のパスを指定すると、ファイルの編集(+file+が存在しない場合は新規作成)を行います。vi はコマンドモードの状態です。


### 4.7.2. 文字の入力

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する必要があります。コマンドモードから入力モードに移行するには、「表 4.5. 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。

表 4.5 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から文字入力を開始
a	カーソルの後ろから文字入力を開始

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。現在のモードが分からなくなった場合は、ESC キーを入力し、一旦コマンドモードへ戻ることにより混乱を防げます。



**日本語変換機能を OFF に**

vi のコマンドを入力する時は ATDE の日本語入力システム(Mozc)を OFF にしてください。日本語入力システムの ON/OFF は、半角/全角キーで行うことができます。

「i」、「a」それぞれのコマンドを入力した場合の文字入力の開始位置を「図 4.19. 入力モードに移行するコマンドの説明」に示します。

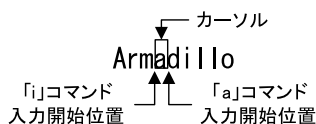


図 4.19 入力モードに移行するコマンドの説明



**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、「4.7.4. 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。

### 4.7.3. 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 4.6. 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カーソルを移動することができます。

表 4.6 カーソルの移動コマンド

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

### 4.7.4. 文字の削除

文字を削除する場合は、コマンドモードで「表 4.7. 文字の削除コマンド」に示すコマンドを入力します。

表 4.7 文字の削除コマンド

コマンド	動作
x	カーソル上の文字を削除
dd	現在行を削除

「x」コマンド、「dd」コマンドを入力した場合に削除される文字を「図 4.20. 文字を削除するコマンドの説明」に示します。

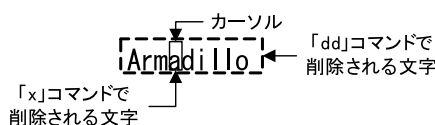


図 4.20 文字を削除するコマンドの説明

### 4.7.5. 保存と終了

ファイルの保存、終了を行うコマンドを「表 4.8. 保存・終了コマンド」に示します。

表 4.8 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[file]	ファイルを+file+に指定して保存
: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「:」(コロン)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。

## 5. 起動と終了

### 5.1. 起動

電源入力インターフェース(CON12)に電源を接続すると Armadillo-640 が起動します。



USB シリアル変換アダプタのスライドスイッチやユーザースイッチによって起動モードが変わります。詳しくは「4.4. 接続方法」、「4.6. スライドスイッチの設定について」を参照してください。

以下に起動ログの例を示します。

```
U-Boot 2020.04-at15 (Jun 09 2023 - 18:46:32 +0900)

CPU:   i.MX6ULL rev1.1 at 396 MHz
Model: Atmark Techno Armadillo-640
DRAM:  512 MiB
setup_rtc_disarm_alarm: Can't find bus
WDT:   Started with servicing (10s timeout)
PMIC:  PFUZE3000 DEV_ID=0x30 REV_ID=0x11
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
Loading Environment from MMC... OK
In:    mxc_serial
Out:   mxc_serial
Err:   mxc_serial
switch to partitions #0, OK
mmc0(part 0) is current device
flash target is MMC:0
Net:
Warning: ethernet@2188000 using MAC address from ROM
eth0: ethernet@2188000
Fastboot: Normal
Normal Boot
Hit any key to stop autoboot:  0
switch to partitions #0, OK
mmc0(part 0) is current device
6859976 bytes read in 162 ms (40.4 MiB/s)
Booting from mmc ...
37363 bytes read in 6 ms (5.9 MiB/s)
Loading fdt boot/armadillo.dtb
45 bytes read in 4 ms (10.7 KiB/s)
4587 bytes read in 5 ms (895.5 KiB/s)
Applying fdt overlay: armadillo-640-lcd70ext-l00.dtbo
## Booting kernel from Legacy Image at 80800000 ...
   Image Name:   Linux-5.10.180-2-at
   Created:     2023-06-09  9:48:24 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:   6859912 Bytes = 6.5 MiB
   Load Address: 82000000
   Entry Point: 82000000
```



```

Verifying Checksum ... OK
## Flattened Device Tree blob at 83500000
Booting using the fdt blob at 0x83500000
Loading Kernel Image
Loading Device Tree to 9ef1d000, end 9ef49fff ... OK

Starting kernel ...

OpenRC 0.45.2 is starting up Linux 5.10.180-2-at (armv7l)

* Mounting /proc ... [ ok ]
* Mounting /run ... * /run/openrc: creating directory
* /run/lock: creating directory
* /run/lock: correcting owner
* Caching service dependencies ... [ ok ]
* Clock skew detected with `/etc/init.d/devfs'
* Adjusting mtime of `/run/openrc/deptree' to Sun Jun 11 01:34:52 2023

* WARNING: clock skew detected!
* Mounting /sys ... * Remounting devtmpfs on /dev ... [ ok ]
[ ok ]
* Mounting security filesystem ... [ ok ]
* Mounting config filesystem ... [ ok ]
* Mounting /dev/mqueue ... [ ok ]
* Mounting /dev/pts ... [ ok ]
* Mounting /dev/shm ... [ ok ]
fsck_atlog      | * Checking at-log filesystem /dev/mmcblk0gp1 ... [ ok ]
udev            | * Starting udev ... [ ok ]
fsck            | * Checking local filesystems ... [ ok ]
root            | * Remounting filesystems ... [ ok ]
localmount     | * Mounting local filesystems ... [ ok ]
overlayfs      | * Preparing overlayfs over / ... [ ok ]
* WARNING: clock skew detected!
hostname       | * Setting hostname ... [ ok ]
sysctl         | * Configuring kernel parameters ... [ ok ]
udev-trigger   | * Generating a rule to create a /dev/root symlink ... [ ok ]
udev-trigger   | * Populating /dev with existing devices through uevents ... [ ok ]
bootmisc      | * Migrating /var/lock to /run/lock ... [ ok ]
bootmisc      | * Creating user login records ... [ ok ]
bootmisc      | * Wiping /var/tmp directory ... [ ok ]
dbus           | * /run/dbus: creating directory
dbus           | * /run/dbus: correcting owner
syslog         | * Starting busybox syslog ... [ ok ]
dbus           | * Starting System Message Bus ... [ ok ]
klogd         | * Starting busybox klogd ... [ ok ]
networkmanager | * Starting networkmanager ... [ ok ]
dnsmasq       | * /var/lib/misc/dnsmasq.leases: creating file
dnsmasq       | * /var/lib/misc/dnsmasq.leases: correcting owner
dnsmasq       | * Starting dnsmasq ... [ ok ]
* WARNING: clock skew detected!
buttond       | * Starting button watching daemon ... [ ok ]
reset_bootcount | * Resetting bootcount in bootloader env ... [ ok ]
podman-atmark | * Starting configured podman containers ...Environment OK, copy 1
reset_bootcount | [ ok ]
zramswap      | [ ok ]
zramswap      | * Creating zram swap device ... [ ok ]
chronyd       | * Starting chronyd ... [ ok ]

```

```
local          | * local: waiting for chronyd (50 seconds)
local          | * Starting local ... [ ok ]

Welcome to Alpine Linux 3.17
Kernel 5.10.180-2-at on an armv7l (/dev/ttyxc0)

armadillo login:
```

## 5.2. 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。「root」か一般ユーザーの「atmark」でログインすることができます。

「10.7.2. SWU イメージの作成」の手順で `initial_setup.swu` を適用した Armadillo の「root」、「atmark」ユーザーには、`initial_setup.swu` 作成時に入力したパスワードが設定されます。

`initial_setup.swu` を適用しない場合、「root」ユーザーは初回ログイン時にパスワードを入力せずに新しいパスワードを促されます。「atmark」ユーザーは、初期状態ではロックされています。そのロックを解除するには、「root」ユーザーでログインし、`passwd atmark` コマンドで「atmark」ユーザーのパスワードを設定してください。

設定するパスワードには大文字のアルファベット、小文字のアルファベット、0 から 9 までの数字、その他(記号・句読点など)を含める事ができます。

### 1. root でログイン

初期パスワードを変更します。

```
armadillo login: root
You are required to change your password immediately (administrator enforced).
New password: ❶
Retype new password: ❷
Welcome to Alpine!
```

- ❶ 新しいパスワードを入力します
- ❷ 新しいパスワードを再入力します

### 2. atmark でログイン

初期状態でロックされてますので、root で一度パスワードを設定してからログインします。

```
armadillo:~# passwd atmark ❶
New password:
Retype new password:
passwd: password updated successfully
armadillo:~# persist_file /etc/shadow ❷
armadillo:~# exit

Welcome to Alpine Linux 3.17
Kernel 5.10.180-1-at on an armv7l (/dev/ttyxc0)

armadillo login: atmark
```

```
Password: ③
Welcome to Alpine!
```

- ① atmark ユーザーのパスワード変更コマンド。「10.7.2. SWU イメージの作成」を使用した場合には不要です
- ② パスワードファイルを永続化します。
- ③ 設定したパスワードでログインすることができます。



Armadillo BaseOS ではルートファイルシステムに overlayfs を採用しており、そのままではシステムが OFF すると内容は消えてしまいます。そのため persist\_file コマンドが用意されています。このコマンドを利用することでファイル単位で変更を反映することができます。パスワードを設定した後は以下のコマンドを実行してください。

```
[armadillo ~]# persist_file /etc/shadow
```

persist\_file コマンドに関する詳細は「10.8.2. overlayfs と persist\_file について」を参照してください。

## 5.3. 終了方法

eMMC や USB メモリ等に書き込みを行っている時に電源を切断すると、データが破損する可能性があります。安全に終了させる場合は、次のように poweroff コマンドを実行し、「reboot: Power down」と表示されたのを確認してから電源を切断します。

```
armadillo:~# poweroff
* WARNING: clock skew detected!
podman-atmark      | * Stopping all podman containers ... [ ok ]
zramswap           | * Deactivating zram swap device ... [ ok ]
local              | * Stopping local ... [ ok ]
chronyd            | * Stopping chronyd ... [ ok ]
dnsmasq            | * Stopping dnsmasq ... [ ok ]
klogd              | * Stopping busybox klogd ... [ ok ]
buttd              | * Stopping button watching daemon ... [ ok ]
networkmanager    | * Stopping networkmanager ... [ ok ]
syslog             | * Stopping busybox syslog ... [ ok ]
udev               | * Stopping udev ... [ ok ]
dbus               | * Stopping System Message Bus ... [ ok ]
nm-dispatcher: Caught signal 15, shutting down...
localmount         | * Unmounting loop devices
localmount         | * Unmounting filesystems
localmount         | *   Unmounting /var/at-log ... [ ok ]
localmount         | *   Unmounting /var/tmp ... [ ok ]
localmount         | *   Unmounting /var/app/volumes ... [ ok ]
localmount         | *   Unmounting /var/app/rollback/volumes ... [ ok ]
localmount         | *   Unmounting /var/lib/containers/storage_readonly ... [ ok ]
localmount         | *   Unmounting /var/log ... [ ok ]
localmount         | *   Unmounting /tmp ... [ ok ]
killprocs          | * Terminating remaining processes ... [ ok ]
```

```
killprocs          | * Killing remaining processes ... [ ok ]
mount-ro           | * Remounting remaining filesystems read-only ... [ ok ]
mount-ro           | * Remounting / read only ... [ ok ]
indicator_signals  | * Signaling external devices we are shutting down ... [ ok ]
The system is going down NOW!
Sent SIGTERM to all processes
Sent SIGKILL to all processes
Requesting system poweroff
[ 274.621389] imx2-wdt 20bc000.watchdog: Device shutdown: Expect reboot!
[ 274.628443] reboot: Power down
```

Podman コンテナの保存先が tmpfs であり、eMMC への書き込みを行っていない場合は、poweroff コマンドを使用せずに電源を切断することが可能です。

Podman コンテナの保存先が eMMC の場合や、頻繁に rootfs 等の eMMC にあるボリュームを変更するような開発段階においては、poweroff コマンドを実行し、「reboot: Power down」と表示されたのを確認してから電源を切断してください。



halt コマンドで終了させた場合、「reboot: System halted」と表示されてから約 128 秒後、Armadillo は自動的に再起動します。確実に終了させるためにも poweroff コマンドを利用してください。



poweroff の場合、Armadillo-640 は、ONOFF ピンを GND とショートすることで電源をオフした場合と同じ状態になります。そのため、RTC\_BAT ピンからバックアップ電源が供給されている限り、5V 電源を切ったのち 5V 電源を再入力しても Armadillo-640 が起動しません。詳しくは「14.1.5.1. ONOFF ピンの制御について」を参照してください。

## 6. ユーザー登録

アットマークテクノ製品をご利用のユーザーに対して、購入者向けの限定公開データの提供や大切なお知らせをお届けするサービスなど、ユーザー登録すると様々なサービスを受けることができます。サービスを受けるためには、「アットマークテクノ Armadillo サイト」にユーザー登録をする必要があります。

ユーザー登録すると次のようなサービスを受けることができます。

- ・ 製品仕様や部品などの変更通知の閲覧・配信
- ・ 購入者向けの限定公開データのダウンロード
- ・ 該当製品のバージョンアップに伴う優待販売のお知らせ配信
- ・ 該当製品に関する開発セミナーやイベント等のお知らせ配信

詳しくは、「アットマークテクノ Armadillo サイト」をご覧ください。

アットマークテクノ Armadillo サイト

<https://armadillo.atmark-techno.com/>

### 6.1. 購入製品登録

ユーザー登録完了後に、購入製品登録することで、「購入者向けの限定公開データ」をダウンロードすることができるようになります。

購入製品登録の詳しい手順は以下の URL をご参照ください。

Armadillo-640 購入製品登録

<https://armadillo.atmark-techno.com/armadillo-640/register>

# 7. 動作確認方法

本章では、ハードウェアの動作確認に使用するコマンドやその実行手順について説明します。

ハードウェアの動作確認以外が目的のコマンドや手順については「10. Howto」を参照してください。

## 7.1. ネットワーク

ここでは、ネットワークの設定方法について説明します。

### 7.1.1. 接続可能なネットワーク

Armadillo-640 は、1 つの Ethernet ポートが搭載されています。Linux からは、eth0 に見えます。

表 7.1 ネットワークとネットワークデバイス

ネットワーク	ネットワークデバイス	出荷時の設定
Ethernet	eth0	DHCP

### 7.1.2. IP アドレスの確認方法

Armadillo-640 の IP アドレスを確認するには、ip addr コマンドを使用します。

```
[armadillo ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN 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
   inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
   link/ether 00:11:0c:00:00:f3 brd ff:ff:ff:ff:ff:ff
   inet 192.0.2.0/24 brd 192.0.2.255 scope global dynamic noprefixroute eth0
       valid_lft 28786sec preferred_lft 28786sec
   inet6 2001:db8::/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

図 7.1 IP アドレスの確認

inet となっている箇所が IP アドレスです。特定のインターフェースのみを表示したい場合は、以下のようになります。

```
[armadillo ~]# ip addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
   link/ether 00:11:0c:00:0b:79 brd ff:ff:ff:ff:ff:ff
   inet 192.0.2.0/24 brd 192.0.2.255 scope global dynamic noprefixroute eth0
       valid_lft 28656sec preferred_lft 28656sec
```

```
inet6 2001:db8::/64 scope link noprefixroute
valid_lft forever preferred_lft forever
```

図 7.2 IP アドレス(eth0)の確認

### 7.1.3. ネットワークの設定方法

Armadillo-640 では、通常の Linux システムと同様、ネットワークインターフェースの設定は NetworkManager を使用します。NetworkManager はすべてのネットワーク設定をコネクションとして管理します。コネクションには「どのようにネットワークへ接続するか」、「どのようにネットワークを作成するか」を記述し、`/etc/NetworkManager/system-connections/` に保存します。また、1つのデバイスに対して複数のコネクションを保存することは可能ですが、1つのデバイスに対して有効化にできるコネクションは1つだけです。

NetworkManager は、従来の `/etc/network/interfaces` を使った設定方法もサポートしていますが、本書では `nmcli` を用いた方法を中心に紹介します。

#### 7.1.3.1. nmcli について

`nmcli` は NetworkManager を操作するためのコマンドラインツールです。「図 7.3. nmcli のコマンド書式」に `nmcli` の書式を示します。このことから、`nmcli` は「オブジェクト (OBJECT) というものが存在し、それぞれのオブジェクトに対してコマンド (COMMAND) を実行する。」という書式でコマンドを入力することがわかります。また、オブジェクトそれぞれに `help` が用意されていることもここから読み取れます。

```
nmcli [ OPTIONS ] OBJECT { COMMAND | help }
```

図 7.3 nmcli のコマンド書式

### 7.1.4. nmcli の基本的な使い方

ここでは `nmcli` の、基本的な使い方を説明します。

#### 7.1.4.1. コネクションの一覧

登録されているコネクションの一覧を確認するには、次のようにコマンドを実行します。<sup>[1]</sup>

```
[armadillo ~]# nmcli connection
NAME                UUID                                  TYPE      DEVICE
Wired connection 1  a6f99120-b4ed-3823-a6f0-0491d4b6101e  ethernet  eth0
```

図 7.4 コネクションの一覧

表示された NAME については、以降 [ID] として利用することができます。

#### 7.1.4.2. コネクションの有効化・無効化

コネクションを有効化するには、次のようにコマンドを実行します。

<sup>[1]</sup> `nmcli connection show [ID]` によって、より詳細な情報を表示することもできます。

```
[armadillo ~]# nmcli connection up [ID]
```

### 図 7.5 コネクションの有効化

コネクションを無効化するには、次のようにコマンドを実行します。

```
[armadillo ~]# nmcli connection down [ID]
```

### 図 7.6 コネクションの無効化

#### 7.1.4.3. コネクションの作成

コネクションを作成するには、次のようにコマンドを実行します。

```
[armadillo ~]# nmcli connection add con-name [ID] type [type] ifname [interface name]
```

### 図 7.7 コネクションの作成

[ID] にはコネクションの名前(任意)、[type] には ethernet、wifi といった接続タイプ、[interfacename] にはインターフェース名(デバイス)を入力します。これにより /etc/NetworkManager/system-connections/ に[ID]の名前でコネクションファイルが作成されます。このファイルを vi などで編集し、コネクションを修正することも可能です。

Armadillo-640 を再起動したときにコネクションファイルが消えてしまわないように、persist\_file コマンドで永続化する必要があります。persist\_file コマンドに関する詳細は「10.8.2. overlays と persist\_file について」を参照してください。

```
[armadillo ~]# persist_file /etc/NetworkManager/system-connections/<コネクションファイル名>
```

### 図 7.8 コネクションファイルの永続化



別の Armadillo-640 からコネクションファイルをコピーした場合は、コネクションファイルのパーミッションを 600 に設定してください。600 に設定後、nmcli c reload コマンドでコネクションファイルを再読み込みします。

```
[armadillo ~]# chmod 600 /etc/NetworkManager/system-connections/<コネクションファイル名>
[armadillo ~]# persist_file /etc/NetworkManager/system-connections/<コネクションファイル名>
[armadillo ~]# nmcli c reload
```

swu イメージを使用してコネクションファイルのアップデートを行う場合は、swu イメージに含めるコネクションファイルのパーミッションを 600 に設定してから、swu イメージを作成してください。アップデート実行時には swu イメージ作成時のパーミッションが維持されるため、上記のコ



マンド実行手順は不要です。swu イメージに関しては「10.7. Armadillo のソフトウェアをアップデートする」を参考にしてください。

#### 7.1.4.4. コネクションの削除

コネクションを削除するには、次のようにコマンドを実行します。

```
[armadillo ~]# nmcli connection delete [ID]
```

図 7.9 コネクションの削除

これにより /etc/NetworkManager/system-connections/ のコネクションファイルも同時に削除されます。コネクションの作成と同様に persist\_file コマンドで永続化する必要があります。

```
[armadillo ~]# persist_file -d /etc/NetworkManager/system-connections/<コネクションファイル名>
```

図 7.10 コネクションファイル削除時の永続化

#### 7.1.4.5. 固定 IP アドレスに設定する

「表 7.2. 固定 IP アドレス設定例」の内容に設定する例を、「図 7.11. 固定 IP アドレス設定」に示します。

表 7.2 固定 IP アドレス設定例

項目	設定
IP アドレス	192.0.2.10
マスク長	24
デフォルトゲートウェイ	192.0.2.1

```
[armadillo ~]# nmcli connection modify [ID] \
  ipv4.method manual ipv4.addresses 192.0.2.10/24 ipv4.gateway 192.0.2.1
```

図 7.11 固定 IP アドレス設定

#### 7.1.4.6. DNS サーバーを指定する

DNS サーバーを指定する例を、「図 7.12. DNS サーバーの指定」に示します。

```
[armadillo ~]# nmcli connection modify [ID] ipv4.dns 192.0.2.1
```

図 7.12 DNS サーバーの指定

#### 7.1.4.7. DHCP に設定する

DHCP に設定する例を、「図 7.13. DHCP の設定」に示します。

```
[armadillo ~]# nmcli connection modify [ID] ipv4.method auto
```

図 7.13 DHCP の設定



-ipv4.addresses のように、プロパティ名の先頭に "-" を付けることで設定したプロパティを削除することができます。反対に "+" を付けることでプロパティを追加することができます。

#### 7.1.4.8. コネクションの修正を反映する

有効化されているコネクションを修正した場合、かならず修正したコネクションを再度有効化してください。

```
[armadillo ~]# nmcli connection down [ID]
[armadillo ~]# nmcli connection up [ID]
```

図 7.14 コネクションの修正の反映

#### 7.1.4.9. デバイスの一覧

デバイスの一覧(デバイス名、タイプ、状態、有効なコネクション)を確認するには、次のようにコマンドを実行します。

```
[armadillo ~]# nmcli device
DEVICE TYPE      STATE      CONNECTION
eth0   ethernet  connected  Wired connection 1
lo     loopback  unmanaged  --
```

図 7.15 デバイスの一覧

#### 7.1.4.10. デバイスの接続

デバイスを接続するには、次のようにコマンドを実行します。

```
[armadillo ~]# nmcli device connect [ifname]
```

図 7.16 デバイスの接続



デバイスを接続するには、接続しようとしているデバイスの有効なコネクションが必要です。"Error: neither a valid connection nor device given" というメッセージが表示された場合には、nmcli connection など有効なコネクションがあるかを確認してください。

### 7.1.4.11. デバイスの切断

デバイスを切断するには、次のようにコマンドを実行します。

```
[armadillo ~]# nmcli device disconnect [ifname]
```

図 7.17 デバイスの切断

## 7.1.5. 有線 LAN

有線 LAN で正常に通信が可能か確認します。設定を変更した場合、必ず変更したインターフェースを再度有効化してください。

同じネットワーク内にある通信機器と PING 通信を行います。以下の例では、通信機器が「192.0.2.20」という IP アドレスを持っていると想定しています。

```
[armadillo ~]# ping -I eth0 -c 3 192.0.2.20 ❶
PING 192.0.2.20 (192.0.2.20): 56 data bytes
64 bytes from 192.0.2.20: seq=0 ttl=64 time=3.056 ms
64 bytes from 192.0.2.20: seq=1 ttl=64 time=1.643 ms
64 bytes from 192.0.2.20: seq=2 ttl=64 time=1.633 ms

--- 192.0.2.20 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 1.633/2.110/3.056 ms
```

図 7.18 有線 LAN の PING 確認

- ❶ -I オプションでインターフェースを指定できます。



有線 LAN 以外のインターフェースが有効化されている場合、ルーティングの設定などにより、ネットワーク通信に有線 LAN が使用されない場合があります。設定を必ず確認してください。確実に有線 LAN の接続確認をする場合は、有線 LAN 以外のインターフェースを無効化してください。

## 7.2. ストレージ

Armadillo-640 でストレージとして使用可能なデバイスを次に示します。

表 7.3 ストレージデバイス

デバイス種類	ディスクデバイス	先頭パーティション	インターフェース
オンボード eMMC	/dev/mmcblk0	/dev/mmcblk0p1	オンボード
オンボード eMMC (GPP)	/dev/mmcblk0gp3	なし	オンボード
microSD/microSDHC/ microSDXC カード	/dev/mmcblk1	/dev/mmcblk1p1	microSD スロット (CON1)
USB メモリ	/dev/sd* <sup>[a]</sup>	/dev/sd*1	USB ホストインターフェース (CON5)

<sup>[a]</sup>USB ハブを利用して複数の USB メモリを接続した場合は、認識された順に sda、sdb、sdc … となります。



## GPP(General Purpose Partition)について

GPP は、eMMC の通常の記憶領域を割譲して eMMC 内部に作られた記憶領域です。 eMMC の通常の記憶領域とはアドレス空間が異なるため、 /dev/mmcblk0 および /dev/mmcblk0p\* に対してどのような書き込みを行っても /dev/mmcblk0gp\* のデータが書き換わることはありません。

Armadillo-640 では、8 MiB の GPP を 4 つ作成しています。各領域の用途を「表 7.4. eMMC の GPP の用途」に示します。

表 7.4 eMMC の GPP の用途

ディスクデバイス	用途
/dev/mmcblk0gp0	ライセンス情報等の為の予約領域
/dev/mmcblk0gp1	動作ログ領域
/dev/mmcblk0gp2	動作ログ予備領域 <sup>[a]</sup>
/dev/mmcblk0gp3	ユーザー領域

<sup>[a]</sup>詳細は「11.4. ログ用パーティションについて」を参照ください。

### 7.2.1. ストレージの使用方法

ここでは、microSDHC カードを接続した場合を例にストレージの使用方法を説明します。以降の説明では、共通の操作が可能な場合に、microSD/microSDHC/microSDXC カードを microSD カードと表記します。

Linux では、アクセス可能なファイルやディレクトリは、一つの木構造にまとめられています。あるストレージデバイスのファイルシステムを、この木構造に追加することを、マウントするといいます。マウントを行うコマンドは、mount です。

mount コマンドの典型的なフォーマットは、次の通りです。

```
mount [-t fstype] device dir
```

図 7.19 mount コマンド書式

-t オプションに続く fstype には、ファイルシステムタイプを指定します。ファイルシステムタイプの指定は省略可能です。省略した場合、mount コマンドはファイルシステムタイプを推測します。この推測は必ずしも適切なものとは限りませんので、事前にファイルシステムタイプが分かっている場合は明示的に指定してください。FAT32 ファイルシステムの場合は vfat、EXT3 ファイルシステムの場合は ext3 を指定します。



通常、購入したばかりの microSDHC カードは FAT32 または exFAT ファイルシステムでフォーマットされています。

device には、ストレージデバイスのデバイスファイル名を指定します。microSD カードのパーティション 1 の場合は /dev/mmcblk1p1、パーティション 2 の場合は /dev/mmcblk1p2 となります。

dir には、ストレージデバイスのファイルシステムをマウントするディレクトリを指定します。

microSD スロット (CON1) に microSD カードを挿入し、以下に示すコマンドを実行すると、/mnt ディレクトリに microSD カードのファイルシステムをマウントすることができます。microSD カード内のファイルは、/mnt ディレクトリ以下に見えるようになります。

```
[armadillo ~]# mount -t vfat /dev/mmcblk1p1 /mnt
[armadillo ~]# ls /mnt
:
:
```

### 図 7.20 ストレージのマウント

ストレージを安全に取り外すには、アンマウントという作業が必要です。アンマウントを行うコマンドは、umount です。オプションとして、アンマウントしたいデバイスがマウントされているディレクトリを指定します。

```
[armadillo ~]# umount /mnt
```

### 図 7.21 ストレージのアンマウント

## 7.2.2. ストレージのパーティション変更とフォーマット

通常、購入したばかりの microSD カードや USB メモリは、一つのパーティションを持ち、FAT32 ファイルシステムでフォーマットされています。

パーティション構成を変更したい場合、fdisk コマンドを使用します。fdisk コマンドの使用例として、一つのパーティションで構成されている microSD カードのパーティションを、2 つに分割する例を「図 7.22. fdisk コマンドによるパーティション変更」に示します。一度、既存のパーティションを削除してから、新たにプライマリパーティションを二つ作成しています。先頭のパーティションには 100MByte、二つめのパーティションに残りの容量を割り当てています。先頭のパーティションは /dev/mmcblk1p1、二つめは /dev/mmcblk1p2 となります。

```
[armadillo ~]# fdisk /dev/mmcblk1

Welcome to fdisk (util-linux 2.37.2).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Command (m for help): d
Selected partition 1
Partition 1 has been deleted.

Command (m for help): n
Partition type
  p   primary (0 primary, 0 extended, 4 free)
  e   extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-15138815, default 2048):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-15138815, default 15138815): +100M
```

```

Created a new partition 1 of type 'Linux' and of size 100 MiB.

Command (m for help): n
Partition type
   p   primary (1 primary, 0 extended, 3 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (2-4, default 2): 2
First sector (206848-15138815, default 206848):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (206848-15138815, default 15138815):

Created a new partition 2 of type 'Linux' and of size 7.1 GiB.

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
[ 305.798606] mmcblk1: p1 p2
Syncing disks.

```

図 7.22 fdisk コマンドによるパーティション変更

FAT32 ファイルシステムでストレージデバイスをフォーマットするには、`mkfs.vfat` コマンドを使用します。また、EXT2 や EXT3、EXT4 ファイルシステムでフォーマットするには、`mkfs.ext2` や `mkfs.ext3`、`mkfs.ext4` コマンドを使用します。microSD カードのパーティション 1 を EXT4 ファイルシステムでフォーマットするコマンド例を次に示します

```
[armadillo ~]# mkfs.ext4 /dev/mmcblk1p1
```

図 7.23 EXT4 ファイルシステムの構築

## 7.3. LED

Armadillo-640 の LED は GPIO で接続されているため、ソフトウェアで制御することができます。

利用しているデバイスドライバは LED クラスとして実装されているため、LED クラスディレクトリ以下のファイルによって LED の制御を行うことができます。LED クラスディレクトリと LED の対応を次に示します。

表 7.5 LED クラスディレクトリと LED の対応

LED クラスディレクトリ	インターフェース	デフォルトトリガ
/sys/class/leds/red/	ユーザー LED 赤	default-on
/sys/class/leds/green/	ユーザー LED 緑	default-on
/sys/class/leds/yellow/	ユーザー LED 黄	none

以降の説明では、任意の LED を示す LED クラスディレクトリを `/sys/class/leds/[LED]/` のように表記します。[LED] の部分を適宜読みかえてください。

### 7.3.1. LED を点灯/消灯する

LED クラスディレクトリ以下の `brightness` ファイルへ値を書き込むことによって、LED の点灯/消灯を行うことができます。brightness に書き込む有効な値は 0~255 です。

brightness に 0 以外の値を書き込むと LED が点灯します。

```
[armadillo ~]# echo 1 > /sys/class/leds/[LED]/brightness
```

図 7.24 LED を点灯させる



Armadillo-640 の LED には輝度制御の機能がないため、0(消灯)、1～255(点灯)の 2 つの状態のみ指定することができます。

brightness に 0 を書き込むと LED が消灯します。

```
[armadillo ~]# echo 0 > /sys/class/leds/[LED]/brightness
```

図 7.25 LED を消灯させる

brightness を読み出すと LED の状態が取得できます。

```
[armadillo ~]# cat /sys/class/leds/[LED]/brightness
```

図 7.26 LED の状態を表示する

### 7.3.2. トリガを使用する

Linux では、LED をある特定のタイミングで光らせることができます。これを「トリガ」と呼びます。LED クラスディレクトリ以下の trigger ファイルへ値を書き込むことによって LED の点灯/消灯にトリガを設定することができます。trigger でサポートされている主な値は以下の通りです。

表 7.6 LED トリガの種類

設定	説明
none	トリガを設定しません
mmc0	eMMC のアクセスランプにします
mmc1	microSD スロットのアクセスランプにします
timer	任意のタイミングで点灯/消灯を行います。この設定にすることにより、LED クラスディレクトリ以下に delay_on, delay_off ファイルが出現し、それぞれ点灯時間, 消灯時間をミリ秒単位で指定します
heartbeat	心拍のように点灯/消灯を行います
default-on	主に Linux カーネルから使用します。LED が点灯します

trigger ファイルを読み出すとサポートしているトリガと、現在有効のトリガが表示されます。[] が付いているものが現在のトリガです。

```
[armadillo ~]# cat /sys/class/leds/[LED]/trigger
[none] rkill-any rkill-none kbd-scrolllock kbd-numlock kbd-capslock kbd-kanalo
ck kbd-shiftlock kbd-altgrlock kbd-ctrllock kbd-altlock kbd-shiftlock kbd-shift
```

```
rlock kbd-ctrllock kbd-ctrlrlock timer oneshot heartbeat backlight gpio default
-on mmc1 mmc0
```

### 図 7.27 対応している LED トリガを表示

以下のコマンドを実行すると、LED が 2 秒点灯、1 秒消灯を繰り返します。

```
[armadillo ~]# echo timer > /sys/class/leds/[LED]/trigger
[armadillo ~]# echo 2000 > /sys/class/leds/[LED]/delay_on
[armadillo ~]# echo 1000 > /sys/class/leds/[LED]/delay_off
```

### 図 7.28 LED のトリガに timer を指定する

以下のコマンドを実行すると、心拍のように点灯/消灯を行います。

```
[armadillo ~]# echo heartbeat > /sys/class/leds/[LED]/trigger
```

### 図 7.29 LED のトリガに heartbeat を指定する

## 7.4. ユーザースイッチ

Armadillo-640 のユーザースイッチのデバイスドライバは、インプットデバイスとして実装されています。インプットデバイスのデバイスファイルからボタンプッシュ/リリースイベントを取得することができます。

ユーザースイッチのインプットデバイスファイルと、各スイッチに対応したイベントコードを次に示します。

表 7.7 インプットデバイスファイルとイベントコード

ユーザースイッチ	インプットデバイスファイル	イベントコード
SW1	/dev/input/by-path/platform-gpio-keys-event	28 (KEY_ENTER)



インプットデバイスは検出された順番にインデックスが割り振られます。USB デバイスなどを接続してインプットデバイスを追加している場合は、デバイスファイルのインデックスが異なる可能性があります。

### 7.4.1. イベントを確認する

ユーザースイッチのボタンプッシュ/リリースイベントを確認するために、ここでは `evtest` コマンドをインストールして使用します。`evtest` を停止するには、`Ctrl-c` を入力してください。

```
[armadillo ~]# apk add evtest
```

### 図 7.30 evtest コマンドのインストール



```
[armadillo ~]# evtest /dev/input/by-path/platform-gpio-keys-event
Input driver version is 1.0.1
Input device ID: bus 0x19 vendor 0x1 product 0x1 version 0x100
Input device name: "gpio-keys"
Supported events:
  Event type 0 (EV_SYN)
  Event type 1 (EV_KEY)
    Event code 148 (KEY_PROG1)
Properties:
Testing ... (interrupt to exit)
Event: time 1638343703.831011, type 1 (EV_KEY), code 28 (KEY_ENTER), value 1 ❶
Event: time 1638343703.831011, ----- SYN_REPORT -----
Event: time 1638343703.991022, type 1 (EV_KEY), code 28 (KEY_ENTER), value 0 ❷
Event: time 1638343703.991022, ----- SYN_REPORT -----
```

図 7.31 ユーザースイッチ: イベントの確認

- ❶ SW1 のボタン プッシュ イベントを検出したときの表示
- ❷ SW1 のボタン リリース イベントを検出したときの表示

## 8. 開発の基本的な流れ

---

### 8.1. アプリケーション開発の流れ

#### 8.1.1. Armadillo への接続

##### 8.1.1.1. シリアルコンソール

Armadillo-640 の標準状態では、UART1 (CON9)をシリアルコンソールとして使用します。シリアル通信設定等については、「4.2.4. シリアル通信ソフトウェア(minicom)の使用」を参照してください。

ログイン方法については、「5.2. ログイン」を参照してください。

##### 8.1.1.2. ssh

Armadillo-640 には openssh がインストールされていますが、デフォルトでは SSH サーバーが起動していません。

SSH サーバーを自動的に起動するようにするためには、以下のコマンドを実行してください。

```
[armadillo:~]# rc-update add sshd
* service sshd added to runlevel default
[armadillo ~]# persist_file /etc/runlevels/default/sshd
[ 2819.277066] EXT4-fs (mmcblk0p1): re-mounted. Opts: (null)
[armadillo ~]# reboot
```

上記の例では、再起動後も設定が反映されるように、`persist_file` コマンドで eMMC に設定を保存しています。

#### 8.1.2. overlayfs の扱い

Armadillo BaseOS ではルートファイルシステムに overlayfs を採用しています。

その為、ファイルを変更した後 Armadillo の電源を切ると変更内容は保持されません。開発中などに `rootfs` の変更内容を保持するには、変更したファイルに対して `persist_file` コマンドを使用します。`persist_file` コマンドの詳細は「10.8.2. overlayfs と `persist_file` について」を参照してください。

開発以外の時は安全のため、ソフトウェアアップデートによる更新を実行してください。アップデート手順に関しては「10.7. Armadillo のソフトウェアをアップデートする」を参照してください。

`rootfs` の内容を変更しても、ソフトウェアアップデートを実施した際に変更した内容が保持されない可能性があります。ソフトウェアアップデート実施後も変更内容を保持する手順に関しては「10.7.7. `swupdate_preserve_files` について」を参照してください。

#### 8.1.3. Podman のデータを eMMC に保存する

デフォルトでは、Podman のデータは `tmpfs` に保存されます。そのため、Armadillo を再起動するとデータは消えてしまいます。この挙動は、Armadillo の運用時を想定したものです。

eMMC への書き込みを最小限にする等の観点から、Armadillo の運用時は、Podman のデータは tmpfs に保存するのが適切です。eMMC への保存が必要な場合のみ SWUpdate または abos-ctrl で読み取り専用のイメージを保存します。

Armadillo 開発時のみ、eMMC に Podman のデータが保存されるようにすることを推奨します。

eMMC に Podman のデータが保存されるようにするには、以下のコマンドを実行します。

```
[armadillo ~]# abos-ctrl podman-storage --disk
Creating configuration for persistent container storage
Create subvolume '/mnt/containers_storage'
[ 2145.288677] EXT4-fs (mmcblk0p1): re-mounted. Opts: (null)
[armadillo ~]# abos-ctrl podman-storage --status
Currently in disk mode, run with --tmpfs to switch
```



podman のストレージはコンテナのイメージやランタイムのデータのみです。

コンテナのデータをボリュームに入れたら消えません。詳しくは「10.2.2.6. コンテナの変更を保存する」を参照してください。

#### 8.1.4. ベースとなるコンテナを取得する

ベースとなる OS を取得します。alpine や debian 等、任意の環境でアプリケーションを作成することができます。

ベースとなる OS はイメージの公開・共有サービスである Docker Hub [[https://hub.docker.com/search?type=image&image\\_filter=official](https://hub.docker.com/search?type=image&image_filter=official)] から取得することができます。目的に合わせて選択してください。

マルチメディアや機械学習を行うアプリケーションを作成する場合は、アットマークテクノが配布している debian コンテナがおすすめです。

#### 8.1.5. デバイスのアクセス権を与える

開発中のアプリケーションがデバイスを利用する場合は、コンテナにデバイスを渡す必要があります。

podman\_start のコンテナコンフィグに add\_devices コマンドでデバイスファイルを指定します。詳細は「10.3.1. コンテナの自動起動」を参照ください。



--privileged オプションを指定するとすべてのセキュリティーメカニズムが無効になる為、全てのデバイスが利用できるようになります。このオプションを利用することは、セキュリティー上問題がある為、デバッグ用途でのみご利用ください。

## 8.1.6. アプリケーションを作成する

「10.2. アプリケーションをコンテナで実行する」を参考にして、オリジナルのアプリケーションを開発します。

## 8.1.7. コンテナやデータを保存する

ログやデータベース、自分のアプリケーションのデータを保存する場合にボリュームを使ってください。実行中のコンテナイメージを保存するには `podman commit` コマンドで保存してください。詳しい手順は「10.2.2.6. コンテナの変更を保存する」を参考にしてください。

# 8.2. アプリケーションコンテナの運用

「10.3. コンテナの運用」を参考にしてください。

## 8.2.1. アプリケーションの自動起動

`podman_start` 用の設定ファイル (`/etc/atmark/containers/*.conf`) を作成します。その後、`podman_start -a` コマンドを実行するか、`armadillo` を再起動してコンテナが自動起動することを確認してください。コンテナの自動起動に関する詳しい説明は「10.3.1. コンテナの自動起動」を参考してください。

## 8.2.2. アプリケーションの送信

まず、コンテナをコンテナレジストリに送るか、`podman save` コマンドを実行してアーカイブを作成します。

以下の例では ATDE に `mkswu` のキーを作成して、`docker.io` のイメージをそのまま使います。

手順の詳しい説明やオプションは「10.7. Armadillo のソフトウェアをアップデートする」を参考にしてください。

```
[ATDE ~]$ sudo apt update && sudo apt install mkswu
[ATDE ~]$ mkswu --init
: (省略)
[ATDE ~]$ cd mkswu
[ATDE ~/mkswu]$ cp /usr/share/mkswu/examples/pull_container_nginx.desc .
[ATDE ~/mkswu]$ cp -r /usr/share/mkswu/examples/nginx_start .
[ATDE ~/mkswu]$ vi pull_container_nginx.desc
swdesc_option version=1

swdesc_pull_container "docker.io/nginx:alpine"
swdesc_files --extra-os nginx_start
[ATDE ~/mkswu]$ mkswu -o initial_setup_container.swu ¥
    initial_setup.desc pull_container_nginx.desc
```

ここで作成した `initial_setup_container.swu` ファイルを USB メモリに配置して、Armadillo-640 に刺すとインストールされます。

インストールが終了して再起動すると `docker.io/nginx:alpine` のコンテナを起動します。

## 8.2.3. インストール確認：初期化

購入状態で SWU をインストールできるか確認をするために、ソフトウェアの初期化を行います。

「10.6. Armadillo のソフトウェアの初期化」を参照し、Armadillo Base OS を初期化してからアプリケーションをアップデートしてください。

## 8.2.4. アプリケーションのアップデート

アップデートを行う方法は以下の二通りです：

### 1. podman run コマンドで、差分アップデートを行う

ここでは例として、アプリケーションのコンテナを myimage:1、アップデート後を myimage:2 とします。

```
[armadillo ~]# podman run --name update myimage:1 sh -c "apk upgrade --no-cache"
[armadillo ~]# podman commit update myimage:2
[armadillo ~]# podman rm update
[armadillo ~]# podman_partial_image -b myimage:1 -o myimage2_update.tar myimage:2
```

出来上がった myimage2\_update.tar は普通のコンテナと同じように扱うことができます。myimage:1 が存在しない場合はエラーとなります。

詳しいコンテナアップデートの手順は「10.2.2.7. コンテナの自動作成やアップデート」を参考にしてください。

繰り返し差分アップデートをすると、イメージサイズが大きくなってしまいます。ストレージ容量が不足する場合は、次に示す手順でコンテナを新しく構築してください。

### 2. コンテナを新しく構築する

ベースとなるコンテナをアップデートして、そのコンテナに自分のアプリケーションを入れます。

差分アップデートと異なり共有部分が無い為、コンテナ全体を送る必要があります。

自動的にイメージを作る方法は「10.2.2.7. コンテナの自動作成やアップデート」を参考にしてください。

## 9. Web UI によるセットアップ

### 9.1. ABOS Web を使った Armadillo のセットアップ

ABOS Web は、Web ブラウザから Armadillo の動作設定を行う機能で、ABOS (Armadillo Base OS) を搭載する全ての Armadillo に対応しています。ABOS Web は、バージョン v3.17.4-at.7 以降の ABOS に組み込まれていますので、お手元の Armadillo の ABOS のバージョンが、それより古い場合には、最新のインストールディスクイメージで初期化してください。以下の説明では、ABOS Web を組み込んだ版の ABOS が、Armadillo の出荷時にインストール済みであるという前提で進めます。購入直後の、工場出荷状態の Armadillo を使い始める時や、Armadillo を利用した製品の開発を行う時には、ABOS Web を利用して Armadillo をセットアップしてください。Armadillo と PC を有線 LAN で接続して、Armadillo の電源を入れて PC で Web ブラウザを起動したら、Web ブラウザのアドレスバーに次の URL を入力してください。 <https://armadillo.local:58080>

ABOS Web に対応した Armadillo が正常に起動していれば、Web ブラウザに ABOS Web の画面が表示されます。最初の接続では、「9.1.1. ABOS Web のパスワード登録」で説明するように、パスワード登録画面が表示されます。パスワード設定画面でパスワードを登録した後に、ログイン画面で ABOS Web のパスワードを入力して認証されれば、ABOS Web の設定画面に表示が変わり、設定操作を行うことができます。



ABOS Web が動作する Armadillo が、同じ LAN 上に複数あると、ABOS Web に接続する URL のホスト名部分 (armadillo.local) は、2 台めでは armadillo-2.local、3 台めでは armadillo-3.local のように、違うものが自動的に割り当てられます。どのホスト名が、どの Armadillo のものなのかを判別するのが難しいので、複数台の Armadillo で同時に ABOS Web を動かすときは、LAN に固定 IP アドレスを設定して、IP アドレスで指定できるようにするのがよいでしょう。

まずは、ABOS Web のパスワードを設定して、現在のネットワーク設定状態を見てみましょう。現在のネットワーク設定状態を見る手順は、「9.3.3. 各接続設定 (各ネットワークインターフェースの設定)」をご覧ください。

#### 9.1.1. ABOS Web のパスワード登録

工場出荷状態 (インストールディスクでの初期化直後) の Armadillo では、ABOS Web のパスワードが登録されておらず、パスワードを登録しないとログインできません。最初に ABOS Web に接続した時は、「初回ログイン」のパスワード登録画面が表示されますので、パスワードを設定してください。



The image shows a dark blue login screen for Armadillo Base OS. At the top left is a circular logo with a stylized armadillo head. To its right, the text "Armadillo Base OS" is displayed in white. Below the logo and text, the title "初回ログイン" (Initial Login) is centered. Underneath the title, the instruction "登録するパスワードを入力してください" (Please enter the password to register) is shown. There are two input fields: the first is labeled "パスワード" (Password) and the second is labeled "パスワード(確認)" (Password (Confirmation)). Both fields contain six dots to represent masked characters. At the bottom center, there is a teal button with the text "登録" (Register).

図 9.1 パスワード登録画面

"初回ログイン"のパスワード登録画面で、"パスワード" フィールドと "パスワード(確認)" フィールドに、登録したいパスワードを入力してから、"登録" ボタンをクリックしてください。パスワード登録完了画面が表示されたら、パスワード登録の完了です。



図 9.2 パスワード登録完了画面

パスワード登録完了画面にある "ログインページ" というリンクをクリックすると、ログイン画面が表示されますので、先ほど登録したパスワードを入力して "ログイン" ボタンをクリックしてください。





図 9.3 ログイン画面

ログインに成功すると、ABOS Web のトップページが表示されます。



図 9.4 トップページ



複数台の Armadillo を続けてセットアップする場合、URL が同じなために、2 台目以降の Armadillo では、初回の接続で"初回ログイン"のパスワード登録画面が表示されず、パスワード入力画面が表示される場合があります。その場合は、Web ブラウザの更新ボタンを押してみてください。更新ボタンを何度か押しても"初回ログイン"のパスワード登録画面が表示されない場合は、Web ブラウザのキャッシュをクリアしてから、更新ボタンを押してみてください。

### 9.1.2. ABOS Web の設定操作

ABOS Web で Armadillo の動作設定を行うには、ログイン後に表示されるトップページで、設定したい機能へのリンクをクリックしてください。リンクをクリックすると、リンク先の設定画面が表示されますので、設定画面で、現在の設定内容の確認や設定変更を行ってください。現在の設定内容を確認するには、「各接続設定」をクリックしてください。各機能の設定ページへのリンクは、それぞれの設定ページでも、左端にサイドメニュー形式で表示されます。以後、サイドメニュー形式で表示されたリンクをクリックすることを、「サイドメニューから xxx を選択する」と表記します。ログイン後に表示されるトップページと、それぞれの設定ページには、左端のサイドメニューに加え、上端右側に、現在の接続状態が表示されます。現在の接続状態は、WWAN、WLAN、LAN、のそれぞれについて表示されます。WWAN と WLAN は、それらの通信モジュールが Armadillo に搭載されていないと、表示されません。

### 9.1.3. ログアウト

ABOS Web で必要なセットアップを行ったら、サイドメニューから "ログアウト" を選択してログアウトしてください。ログアウトすると、ログイン画面が再び表示されますので、ABOS Web をすぐに使わないのであれば、Web ブラウザを閉じてください。

## 9.2. ABOS Web ではできないこと

ABOS Web は、ABOS の詳細や Linux のコマンドシェルの操作に詳しくない方でも、簡単に Armadillo のセットアップを行なえることを目的としています。そのための、Armadillo の動作設定を行う機能ですから、動作設定以外のこと、たとえば、Armadillo の動作状態を監視したりすることは、できません。さらに、Armadillo をインターネットから設定操作する、リモート操作もできません。セキュリティの観点から、ABOS Web は、同じ LAN 内からの接続しか受け付けられないように実装しています。

ABOS Web でできる Armadillo の設定については、「9.3. ABOS Web の設定機能一覧と設定手順」をご覧ください。なお、ABOS Web は OSS で提供していますので、現在の ABOS Web に無い設定機能を、ご自分で実装して機能追加することも可能です。

## 9.3. ABOS Web の設定機能一覧と設定手順

現在の ABOS Web で設定できるのは以下の通りで、主にネットワーク動作に関するものです。今後のアップデートで設定機能を追加した際に、このマニュアルも更新します。

- ・ WWAN 設定
- ・ WLAN 設定
- ・ 各接続設定（各ネットワークインターフェースの設定）
- ・ DHCP サーバー設定

- ・ NAT 設定
- ・ VPN 設定
- ・ コンテナ管理
- ・ SWU インストール

これらの設定については、サイドメニューから"状態一覧"を選択して開く画面で、現在の設定状態を見ることができます。

LTE モジュールを搭載した Armadillo をお使いで、LTE モジュールによる WWAN 接続でインターネットをアクセスする場合に、Armadillo に LAN で接続した機器から Armadillo をルーターとして利用したい場合には、NAT 設定機能が役に立つでしょう。LTE モジュールによる WWAN 通信でクラウドサービスに接続し、WLAN や LAN で接続した機器から集めたデータをクラウドサービスに転送したり、それらの機器を、クラウドサービスから Armadillo 経由で遠隔制御するようなシステムを構成する場合にご利用ください。

コンテナ管理は、ABOS 上のコンテナの一覧表示と、それぞれのコンテナの起動と停止を行う機能です。SWU インストールは、PC 上の SWU イメージや HTTP サーバー上の SWU イメージを Armadillo にインストールする機能です。Armadillo のソフトウェアのアップデート用に最初に行う設定で作成する、initial\_setup.swu を Armadillo にインストールする際にご利用ください。さらに、VSCode の ABOSDE (Armadillo Base OS Development Environment [<https://marketplace.visualstudio.com/items?itemName=atmark-techno.armadillo-base-os-development-environment>]) を使って作成した、アプリケーションの SWU イメージをインストールするのも役立ちます。

以下に、それぞれの設定機能を説明します。

### 9.3.1. WWAN 設定

LTE をはじめとする WWAN 通信モジュールを搭載した Armadillo の、WWAN 設定を行います。この設定画面では、WWAN 接続設定の登録と、WWAN 接続の状態（現在のアドレス情報）の表示、登録済み WWAN 接続設定の削除を行うことができます。設定項目のうち、"MCC/MNC" は、通常は空欄にしてください。MCC/MNC 以外の項目を正しく設定しても WWAN 通信が動作しない場合、特に SIM カードがマルチキャリア SIM の場合は、ご契約の通信事業者に MCC/MNC を問い合わせ、通信事業者から提示された MCC/MNC の値を設定してください。それぞれの入力フィールドに設定値を入力して"設定" ボタンをクリックすると、WWAN 接続の設定を登録して、WWAN 接続動作を実行します。WWAN 通信設定が行われ、ネットワーク接続が確立した状態では、割当てられている IP アドレスなどを"現在の WWAN 接続情報"に表示します。「図 9.5. WWAN 設定画面」に、WWAN 設定を行った状態を示します。



図 9.5 WWAN 設定画面

### 9.3.2. WLAN 設定

無線 LAN モジュールを搭載した Armadillo の、WLAN（無線 LAN）設定を行います。この設定画面では、WLAN クライアント（子機）としての設定または、WLAN アクセスポイントとしての設定を行うことができます。クライアントとアクセスポイントのどちらか一方について、接続設定の登録と接続の状態の表示、登録済み設定の削除を行なえます。クライアントとアクセスポイントのどちらに設定するかは、「動作モード選択」欄で指定します。

クライアント設定とアクセスポイント設定の、それぞれについて、以下に説明します。

### 9.3.2.1. WLAN 設定 (クライアントとしての設定)

"動作モード選択"欄で"クライアントとして使用する"を選択すると、クライアント設定画面が表示されます。もしアクセスポイントに設定済みの場合は、アクセスポイントの設定を削除してください。そうしないと、動作モードをクライアントに切り替えることができません。設定項目のうち、ネットワーク名(SSID)は、リストから選択してください。WLAN アクセスポイントを Armadillo が何も検出できない場合は、このリストが空になります。セキュリティ方式も、リストから選択してください。DHCPと固定は、DHCP を選択すると DHCP サーバーから IP アドレスを取得します。固定 を選択すると、固定 IP アドレス設定用の入力フィールドを表示します。それぞれの入力フィールドに設定値を入力して "設定" ボタンをクリックすると、WLAN クライアント設定を登録して、WLAN 接続動作を実行します。WLAN 接続設定が行われ、WLAN 接続が確立した状態では、割当てられている IP アドレスなどを "現在の WLAN 接続情報" に表示します。

「図 9.6. WLAN クライアント設定画面」に、WLAN クライアント設定を行った状態を示します。



図 9.6 WLAN クライアント設定画面

登録済み WLAN クライアント設定を削除して、WLAN アクセスポイントとの接続を切断するには、"設定を削除" ボタンをクリックしてください。

### 9.3.2.2. WLAN 設定 (アクセスポイントとしての設定)

"動作モード選択"欄で"アクセスポイントとして使用する"を選択すると、アクセスポイント設定画面が表示されます。もしクライアントに設定済みの場合は、クライアントの設定を削除してください。そうしないと、動作モードをアクセスポイントに切り替えることができません。設定項目のうち、ブリッジアドレスは、WLAN アクセスポイントに割り当てる IP アドレスを入力してください。サブネットマスクは、アクセスポイントのサブネットのものを入力してください。使用周波数は、5GHz と 2.4GHz のうち使用するものを選択してください。両方の周波数を同時に使用することはできません。使用チャンネルは、リストから選択してください。SSID と パスワード に入力した値は、アクセスポイントに設定した Armadillo に WLAN 子機を接続する際に使用します。

それぞれの入力フィールドに設定値を入力して "設定" ボタンをクリックすると、WLAN アクセスポイント設定を登録して、WLAN アクセスポイント動作を開始します。WLAN アクセスポイント設定が行われ、アクセスポイント動作中の状態では、"現在のアクセスポイント情報" に設定内容を表示します。

「図 9.7. WLAN アクセスポイント設定画面」に、WLAN アクセスポイント設定を行った状態を示します。



図 9.7 WLAN アクセスポイント設定画面





アクセスポイントモードのセキュリティ方式は、WPA2 を使用します。

### 9.3.3. 各接続設定（各ネットワークインターフェースの設定）

設定されたネットワーク接続の一覧を表示します。表示した接続のそれぞれについて、接続の有効化（「接続」）や無効化（「切断」）、および接続設定内容の編集や削除を行うことができます。接続の操作を行う時は、操作したい接続をラジオボタンで選択してください。

現在の接続情報

接続名	接続状態	接続タイプ	インターフェース
<input type="radio"/> podman0	activated	bridge	podman0
<input type="radio"/> Wired connection 1	activated	ethernet	eth0
<input checked="" type="radio"/> abos_web_wwan	activated	gsm	ttyCommModem
<input type="radio"/> veth0	activated	ethernet	veth0
<input type="radio"/> Wired connection 2		ethernet	--


図 9.8 現在の接続情報画面

ここで、「ネットワーク接続」は、Linux のネットワーク接続管理機能（NetworkManager）が管理するコネクションです。ネットワーク接続に対する設定項目の詳細は、NetworkManager のリファレンス（<https://developer-old.gnome.org/NetworkManager/stable/nmcli.html>）をご覧ください。接続設定内容を編集したい接続を選択して "設定を編集" ボタンをクリックすると、設定内容の編集画面を表示します。LAN の接続以外、つまり、WWAN と WLAN の接続に対する設定は、"WWAN 設定" や "WLAN 設定" の設定画面をお使いいただくのが簡単です。

それぞれの接続設定画面では、IPv4 と IPv6 のそれぞれについて、IP アドレスを自動割り当てするかまたは固定 IP アドレスにするかを選択して設定できます。IP アドレスの割り当ては、デフォルトでは自動割り当てです。Armadillo を接続した LAN や WLAN で、Armadillo を DHCP サーバーとして運用する場合は、それらのネットワーク接続を固定 IP アドレスに設定してください。

### 9.3.3.1. LAN 接続設定

LAN 接続の接続名は、デフォルトでは "Wired connection 1" です。LAN ポートを二つ搭載した Armadillo では、二つめの LAN ポートに対応する "Wired connection 2" も有効です。Armadillo を LAN と WWAN との間で IPv4 ルーターとして運用する場合は、LAN 接続の設定で IPv4 アドレスを固定 IP アドレスに設定して下さい。「図 9.9. LAN 接続設定で固定 IP アドレスに設定した画面」に、LAN 接続の設定編集画面で固定 IP アドレスに設定した状態を示します。



接続設定

接続名 (connection.id)  
Wired connection 1

インターフェース (connection.interface-name)  
eth0

IPv4 取得モード (ipv4.method)  
manual

IPv4 アドレス (ipv4.addresses)  
172.16.69.123/16

IPv4 ゲートウェイ (ipv4.gateway)  
172.16.0.1

IPv4 DNS (ipv4.dns)  
192.168.10.1,192.168.10.2

IPv4 スタティックルート (ipv4.routes)

IPv4 ルーティングメトリック (ipv4.route-metric)  
-1

IPv6 取得モード (ipv6.method)  
auto

IPv6 ルーティングメトリック (ipv6.route-metric)  
-1

自動コネクต์ (connection.autoconnect)  
yes

詳細を表示

リセット 保存

図 9.9 LAN 接続設定で固定 IP アドレスに設定した画面

### 9.3.3.2. WWAN 接続設定

WWAN 接続の接続名は、デフォルトでは "abos\_web\_wwan" です。

### 9.3.3.3. WLAN 接続設定

WLAN 接続の接続名は、デフォルトでは、クライアントモードが "abos\_web\_wlan"、アクセスポイントモードが "abos\_web\_br\_ap" です。

### 9.3.4. DHCP サーバー設定

ネットワークインターフェースごとに、接続したネットワーク上で Armadillo を DHCP サーバーとして動作させる設定を行うことができます。接続済みの DHCP サーバー情報を、画面上部の"現在の DHCP 情報"に表示します。DHCP サーバーの設定を登録する場合は、"DHCP 情報入力"欄に設定内容を入力して"設定"ボタンをクリックしてください。「図 9.10. eth0 に対する DHCP サーバー設定」に、一つの LAN ポート (eth0) に対する設定を行った状態を示します。

IPアドレス	サブネットマスク	DHCPリース範囲	インターフェース
--------	----------	-----------	----------

削除

#### DHCP情報入力

インターフェース  
eth0 172.16.1.128/24

DHCPリース範囲  
172.16.1.10  
~  
172.16.1.254

DHCPリース時間  
24h

時間の場合はh、分の場合はmをつけてください(例: 24h、30m)

設定

図 9.10 eth0 に対する DHCP サーバー設定

たとえば、LAN ポートが二つある Armadillo で、それぞれの LAN ポートを異なる LAN に接続して、それぞれの LAN 上で Armadillo を DHCP サーバーとして運用する場合は、eth0 と eth1 に対して DHCP サーバー設定を行ってください。DHCP サーバー設定を削除するには、"現在の DHCP 情報"の一覧で削除したい設定を選択して、"削除"ボタンをクリックしてください。

### 9.3.5. NAT 設定

この設定画面では、ルーター機能での宛先インターフェース設定と、Armadillo を接続した LAN 上の機器用のポートフォワーディング設定を行うことができます。Armadillo を LAN や WLAN と WWAN との間でルーターとして運用する場合は、NAT 設定の宛先インターフェースを WWAN のインターフェー

スに設定してください。そして、LAN や WLAN 上の機器を、WWAN 接続したインターネットにサーバーとして公開したい場合は、ポートフォワーディング設定を使ってください。

### 9.3.5.1. NAT 設定

ルーター機能での宛先インターフェース設定を行なえます。「図 9.11. LTE を宛先インターフェースに指定した設定」に、宛先インターフェースに ppp0 を指定した場合の画面を示します。

現在のNAT設定情報

宛先インターフェース	
<input checked="" type="radio"/>	ppp0

削除

NAT情報入力

宛先インターフェースを選択してください

インターフェース

ppp0

設定

図 9.11 LTE を宛先インターフェースに指定した設定

### 9.3.5.2. ポートフォワーディング設定

受信インターフェースごとに、ポートフォワーディング設定を登録できます。「図 9.12. LTE からの受信パケットに対するポートフォワーディング設定」に、受信インターフェース ppp0 について、ポート 8080 番宛の tcp パケットをポートフォワーディングする設定を行った状態を示します。



図 9.12 LTE からの受信パケットに対するポートフォワーディング設定

### 9.3.6. VPN 設定

Armadillo の VPN 接続設定を行います。この設定画面では、認証方式や VPN クライアントの設定ファイル、認証用の証明書と鍵の設定を行うことができます。VPN 接続を設定していれば、現在の接続状態も表示します。現在の接続状態表示欄にある "接続" ボタンと "切断" ボタンで、VPN 接続の接続と切断を行なえます。VPN クライアントは、現在 OpenVPN [<https://openvpn.net/community/>] をサポートしています。

「図 9.13. VPN 設定」に、VPN 接続設定を行った状態を示します。



図 9.13 VPN 設定

認証方式は、"ユーザ名とパスワード" と "証明書" のどちらか一方を選択できます。認証方式が "証明書" の場合、.ovpn ファイルに証明書や鍵が埋め込まれていれば、それらのファイルを指定する必要はありません。

ABOS Web は、VPN 接続を設定する際に `abos_web_openvpn` という名前のコンテナを自動的に作成し、このコンテナで VPN クライアントを実行します。VPN 接続動作を実行する時には、進行状況を示すログを表示します。

登録済み VPN 設定を削除して、VPN 接続を切断するには、「設定を削除」ボタンをクリックしてください。

### 9.3.7. コンテナ管理

Armadillo 上のコンテナを一覧表示して、コンテナごとに起動・停止を行うことができます。この画面では、ABOS 上にあるコンテナ全てについて、イメージ名やコンテナ名、現在状態を一覧表示します。コンテナの一覧表示欄で選択したコンテナに対し、起動と停止、および、コンテナから出力されたログの表示を行うことができます。

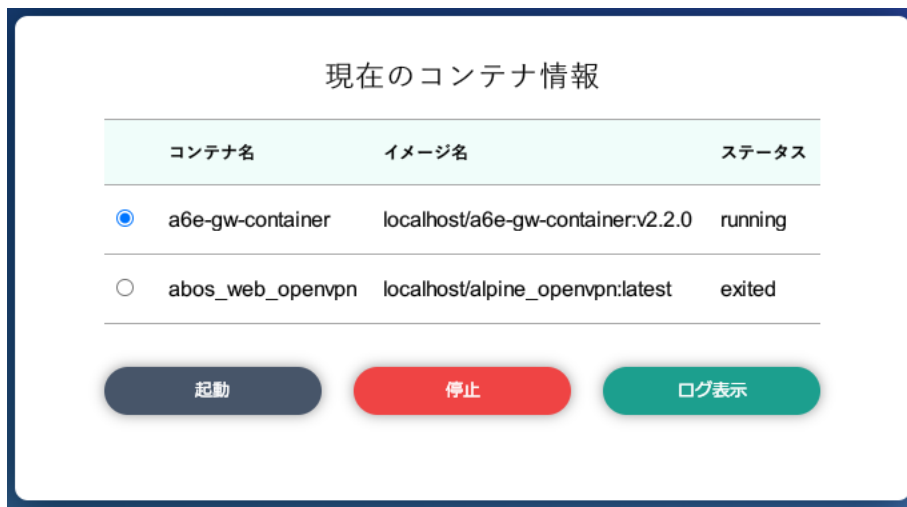


図 9.14 コンテナ管理



「9.3.6. VPN 設定」に記載のとおり、VPN 接続を設定すると、`abos_web_openvpn` のコンテナが作成されます。VPN 接続中は、このコンテナが動作状態になっており、このコンテナをコンテナ管理画面で停止すると、VPN 接続が切断されます。

### 9.3.8. SWU インストール

SWU イメージのインストールを行うことができます。SWU イメージの説明は、「10.7.1. SWU イメージとは」をご覧ください。この画面では、PC 上の SWU イメージファイルまたは、HTTP サーバー上の SWU イメージファイルの URL を指定して、Armadillo にインストールすることができます。Armadillo のソフトウェアのアップデート用に最初に行う設定で作成する `initial_setup.swu` が、まだ Armadillo にインストールされていない場合は、「`mkswu --init` で作成した `initial_setup.swu` をインストールしてください。」というメッセージを画面上部に表示します。

mkswu --init で作成した initial\_setup.swu をインストールしてください。

### SWU ファイル入力

SWU ファイル

選択されていません

---

### SWU URL 入力

SWU URL

図 9.15 SWU インストール

SWU イメージのインストール動作を実行する時には、進行状況を示すログを表示します。"現在の SWU で管理されているバージョン" 欄には、ABOS の各ソフトウェアコンポーネントの名前とバージョン情報を一覧表示します。



現在の SWU で管理されているバージョン

コンポーネント	バージョン
base_os	3.18.2-at.0.20230723
boot	2020.4-at14
extra_os.a6e-gw-container	2.2

最新のインストールログ取得

図 9.16 SWU 管理対象ソフトウェアコンポーネントの一覧表示

### 9.3.9. 状態一覧

各設定画面で行った設定の現在状態を、設定ごとに区切って一覧表示します。

# 10. Howto

## 10.1. CUI アプリケーションを開発する

ここではシェルスクリプトおよび Python を使った CUI アプリケーションの開発方法を紹介します。開発手順としてはシェルスクリプトと Python で同じであるため、シェルスクリプトの場合の例で説明します。

### 10.1.1. CUI アプリケーション開発の流れ

Armadillo 向けに CUI アプリケーションを開発する場合の流れは以下のようになります。

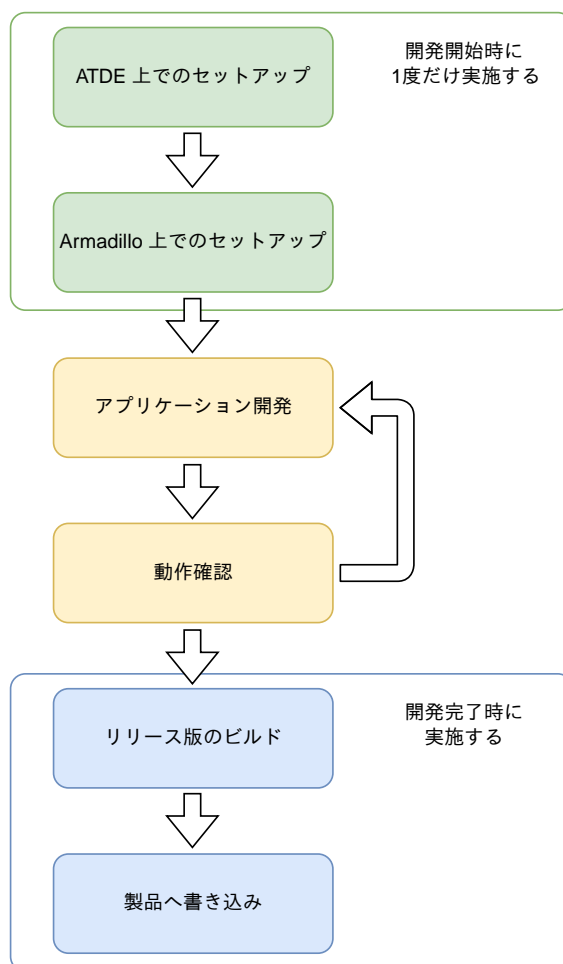


図 10.1 CUI アプリケーション開発の流れ

### 10.1.2. ATDE 上でのセットアップ

ここでは、開発開始時の ATDE 上でのセットアップ手順について説明します。ATDE をお使いでない場合は、先に「4.2.1. ATDE のセットアップ」を参照して ATDE のセットアップを完了してください。

### 10.1.2.1. ソフトウェアのアップデート

ATDE のバージョン v20230123 以上には、VSCode がインストール済みのため新規にインストールする必要はありませんが、使用する前には最新版へのアップデートを行ってください。

```
[ATDE ~]$ sudo apt update  
[ATDE ~]$ sudo apt upgrade
```

図 10.2 ソフトウェアをアップデートする

VSCode を起動するには `code` コマンドを実行します。

```
[ATDE ~]$ code
```

図 10.3 VSCode を起動する



VSCode を起動すると、日本語化エクステンションのインストールを提案してることがあります。その時に表示されるダイアログに従ってインストールを行うと VSCode を日本語化できます。

### 10.1.2.2. VSCode に開発用エクステンションをインストールする

VSCode 上でアプリケーションを開発するためのエクステンションをインストールします。

エクステンションはマーケットプレイスからインストールすることができます。VSCode を起動し、左サイドバーのエクステンションを選択して、検索フォームに「abos」と入力してください。

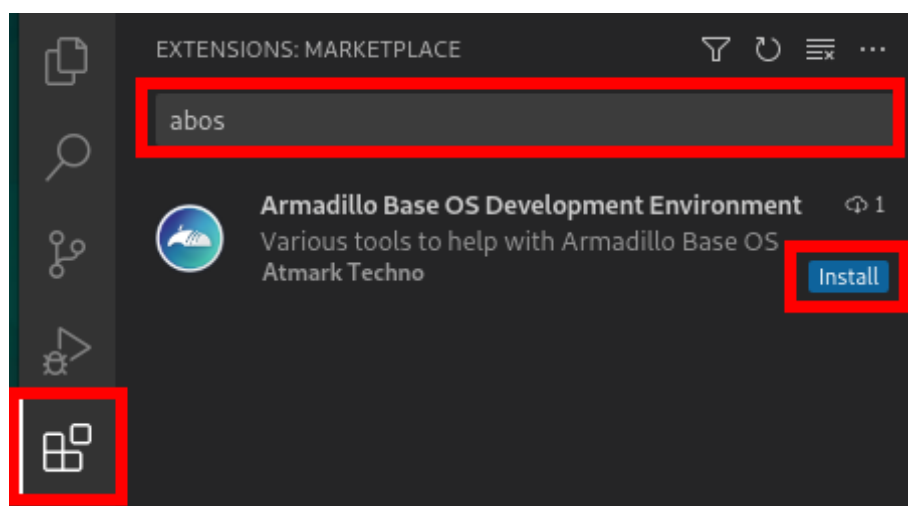


図 10.4 VSCode に開発用エクステンションをインストールする

表示された「Armadillo Base OS Development Environment」の「Install」ボタンを押すとインストールは完了します。

### 10.1.2.3. プロジェクトの作成

VSCoDe の左ペインの [A600] から [Shell New Project] を実行し、表示されるディレクトリ選択画面からプロジェクトを保存するディレクトリを選択してください。実行するためには右に表示されている三角形ボタンを押してください。Python の場合は [Python New Project] を実行してください。保存先を選択すると、プロジェクト名を入力するダイアログが表示されるので、任意のプロジェクト名を入力してエンターキーを押してください。ここでは、ホームディレクトリ直下に `my_project` として保存しています。

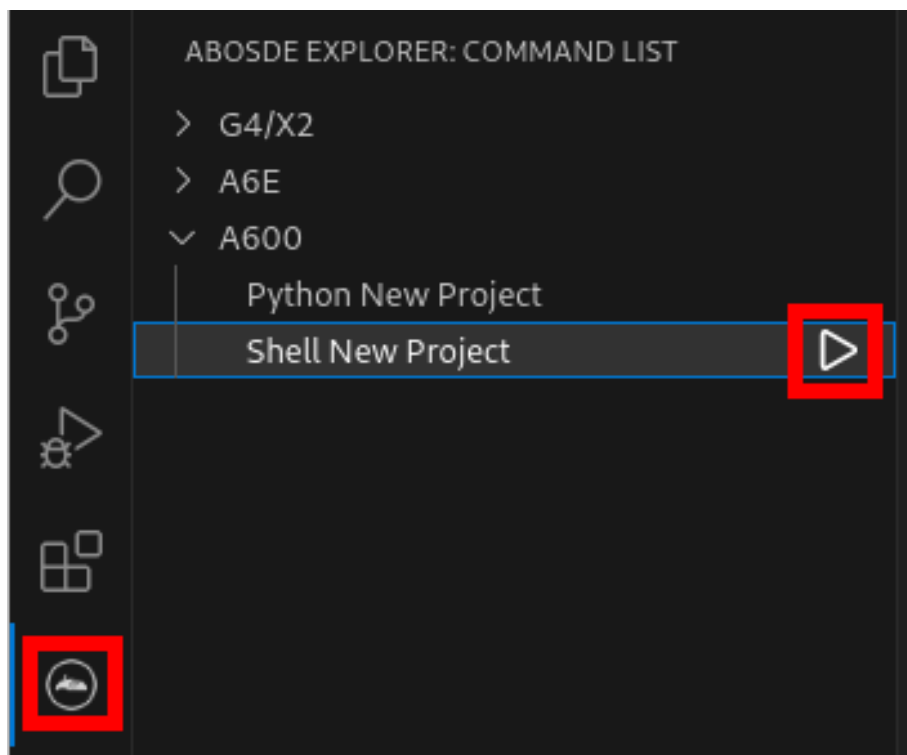


図 10.5 プロジェクトを作成する

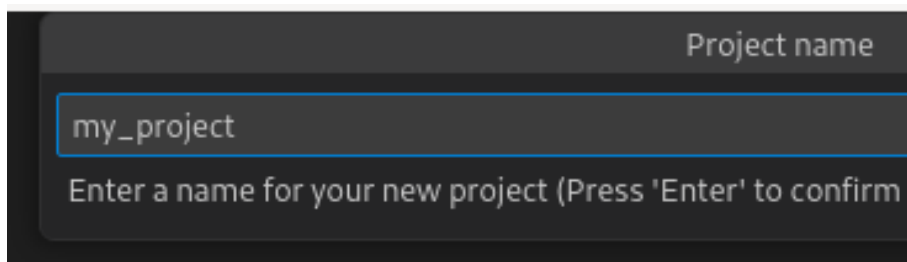


図 10.6 プロジェクト名を入力する

### 10.1.2.4. 初期設定

初期設定では主に Armadillo と SSH で接続するための秘密鍵と公開鍵の生成を行います。

作成したプロジェクトディレクトリへ移動して VSCode を起動してください。

```
[ATDE ~]$ cd my_project
[ATDE ~/my_project]$ code ./
```

図 10.7 初期設定を行う

VSCoDe の左ペインの [my\_project] から [Setup environment] を実行します。

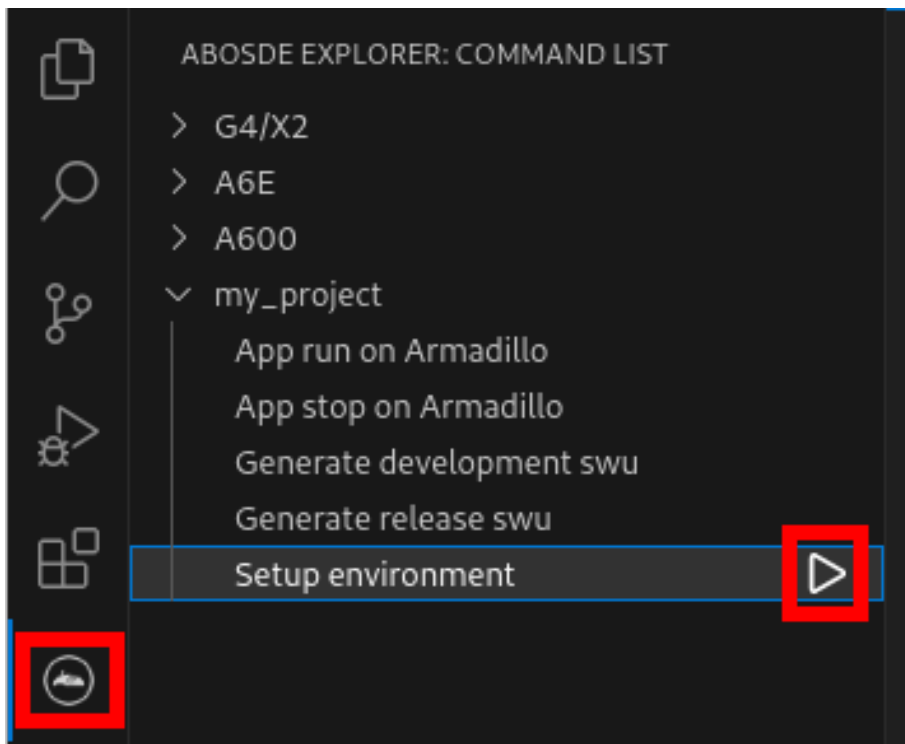


図 10.8 VSCode で初期設定を行う

選択すると、VSCode の下部に以下のようなターミナルが表示されます。

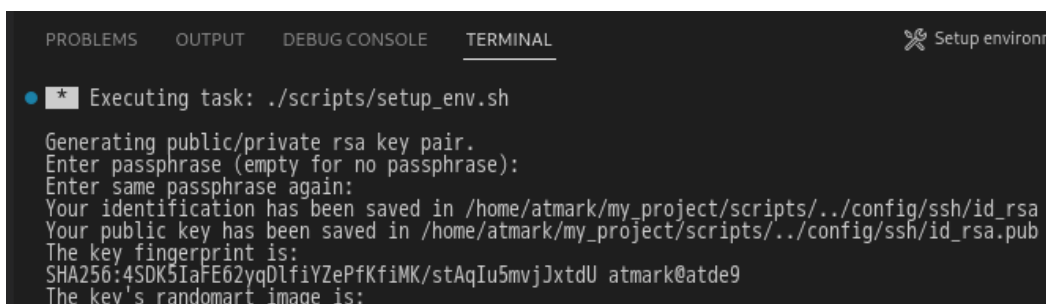


図 10.9 VSCode のターミナル

このターミナル上で以下のように入力してください。

```
* Executing task: ./scripts/setup_env.sh

Generating public/private ed25519 key pair.
Enter passphrase (empty for no passphrase): ❶
```

```
Enter same passphrase again: ❷  
Your identification has been saved in /home/atmark/.ssh/id_ed25519_vscode  
:(省略)  
  
* Terminal will be reused by tasks, press any key to close it. ❸
```

### 図 10.10 SSH 用の鍵を生成する

- ❶ パスフレーズを設定します。設定しない場合は何も入力せず Enter を押します。
- ❷ 1 でパスフレーズを設定した場合は、確認のため再度入力してください。
- ❸ ここで何か任意のキーを押すとターミナルが閉じます。

パスフレーズを設定した場合は、アプリケーションを Armadillo へ転送する時にパスフレーズの入力を求められることがあります。



ssh の鍵は `$HOME/.ssh/id_ed25519_vscode` (と `id_ed25519_vscode.pub`) に保存されていますので、プロジェクトをバックアップする時は `$HOME/.ssh` も保存してください。

#### 10.1.2.5. アプリケーション実行用コンテナイメージの作成

Armadillo 上でアプリケーションを実行するためのコンテナイメージを作成します。ここで作成したコンテナイメージは SWU イメージを使用して Armadillo へインストールするため、事前に `mkswu` を参照して SWU の初期設定を行ってください。

コンテナイメージの作成および SWU イメージの作成も VSCode で行います。VSCode の左ペインの `[my_project]` から `[Generate development swu]` を実行します。

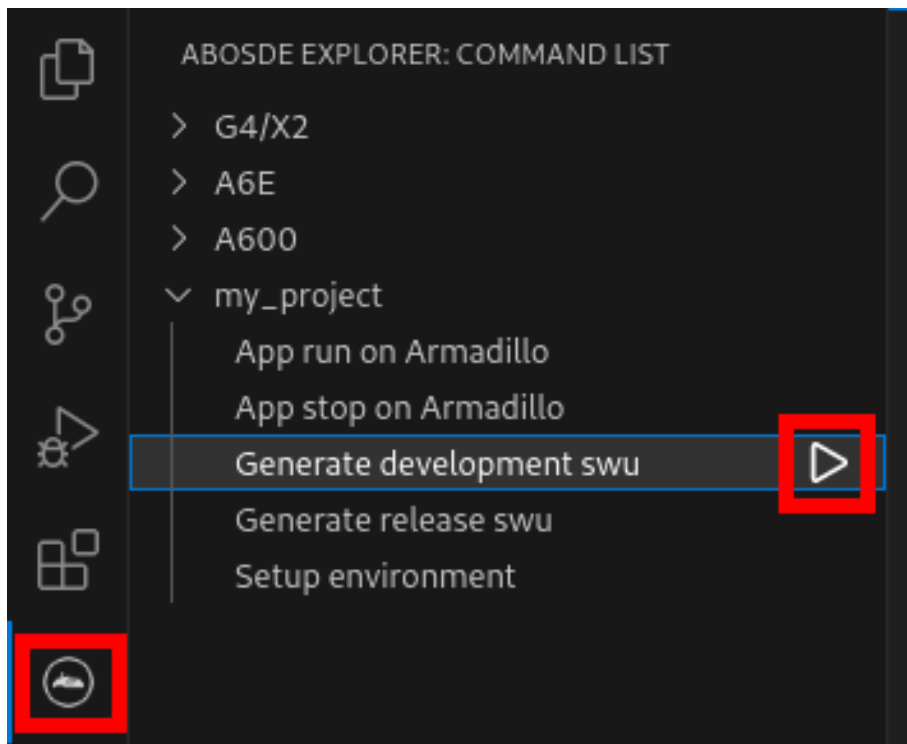


図 10.11 VSCode でコンテナイメージの作成を行う

コンテナイメージの作成にはしばらく時間がかかります。VSCode のターミナルに以下のように表示されるとコンテナイメージの作成は完了です。

```
localhost/shell_armv7_app_image:latest はすでに存在しますが、リビルドしますか？（app の更新だけの場
合には不要です） [y/N] ①
コンテナイメージを ./swu/shell_armv7_app_image.tar に保存しました。
./swu/shell_app.desc のバージョンを 1 から 2 に変更しました。
./swu/dev.swu を作成しました。
次は Armadillo に ./swu/dev.swu をインストールしてください。
* Terminal will be reused by tasks, press any key to close it.
```

↵

図 10.12 コンテナイメージの作成完了

- ① Dockerfile やパッケージリストを変更した場合のみにコンテナを再作成してください。

作成した SWU イメージは my\_project/swu ディレクトリ下に dev.swu というファイル名で保存されています。

### 10.1.3. Armadillo 上でのセットアップ

#### 10.1.3.1. アプリケーション実行用コンテナイメージのインストール

「10.1.2.5. アプリケーション実行用コンテナイメージの作成」で作成した dev.swu を「10.7.3. イメージのインストール」を参照して Armadillo にインストールしてください。

インストール後に自動で Armadillo が再起動します。

## 10.1.4. アプリケーション開発

### 10.1.4.1. VSCode の起動

ここでは、実際に Armadillo 上でサンプルアプリケーションを起動する場合の手順を説明します。プロジェクトディレクトリへ移動し VSCode を起動します。

```
[ATDE ~]$ code ./my_project
```

図 10.13 VSCode で my\_project を起動する

### 10.1.4.2. ディレクトリ構成

プロジェクト内のディレクトリ構成を説明します。

- ・ app: アプリケーションのソースです。Armadillo では /var/app/rollback/volumes/shell\_app または python\_app にそのままコピーされます。
- ・ config: 開発モードのための設定ファイルです。「10.1.4.3. ssh\_config の準備」を参照してください。
- ・ container: スクリプトを実行するコンテナの設定ファイルです。packages.txt に記載されているパッケージがインストールされます。Dockerfile を直接編集することも可能です。
- ・ scripts: VSCode のコマンドに使われているスクリプト類です。編集した場合はサポート対象外となります。
- ・ swu: mkswu のためのテンプレートとコンテナの設定ファイルがあります。shell\_app または python\_app のディレクトリの内容はそのまま SWU に組み込まれます。その中の etc/atmark/containers/shell\_app.conf または python\_app.conf に使われているボリュームやデバイス等が記載されていますので必要に応じて編集してください。

デフォルトのコンテナの設定では app の src/main.sh または Python の場合に src/main.py を実行しますので、リネームが必要な場合はコンテナの設定も修正してください。

このサンプルアプリケーションは、SOC の温度を /vol\_data/log/temp.txt に出力し、ユーザー LED(緑) を点滅させます。

### 10.1.4.3. ssh\_config の準備

プロジェクトディレクトリに入っている config/ssh\_config ファイルを編集して IP アドレスを書き換えてください。

```
[ATDE ~/my_project]$ code config/ssh_config
Host Armadillo
    Hostname 0.0.0.0 ❶
    User root
    IdentityFile ${HOME}/.ssh/id_ed25519_vscode
    UserKnownHostsFile config/ssh_known_hosts
    StrictHostKeyChecking accept-new
```

図 10.14 ssh\_config を編集する



- 1 Armadillo の IP アドレスに置き換えてください。



Armadillo を初期化した場合や、プロジェクトを実行する Armadillo を変えた場合は、プロジェクトの config/ssh\_known\_hosts に保存されている公開鍵で Armadillo を認識できなくなります。その場合はファイルを削除するか、「Setup environment」タスクを再実行してください。

#### 10.1.4.4. アプリケーションの実行

VSCode の左ペインの [my\_project] から [App run on Armadillo] を実行すると、アプリケーションが Armadillo へ転送されて起動します。

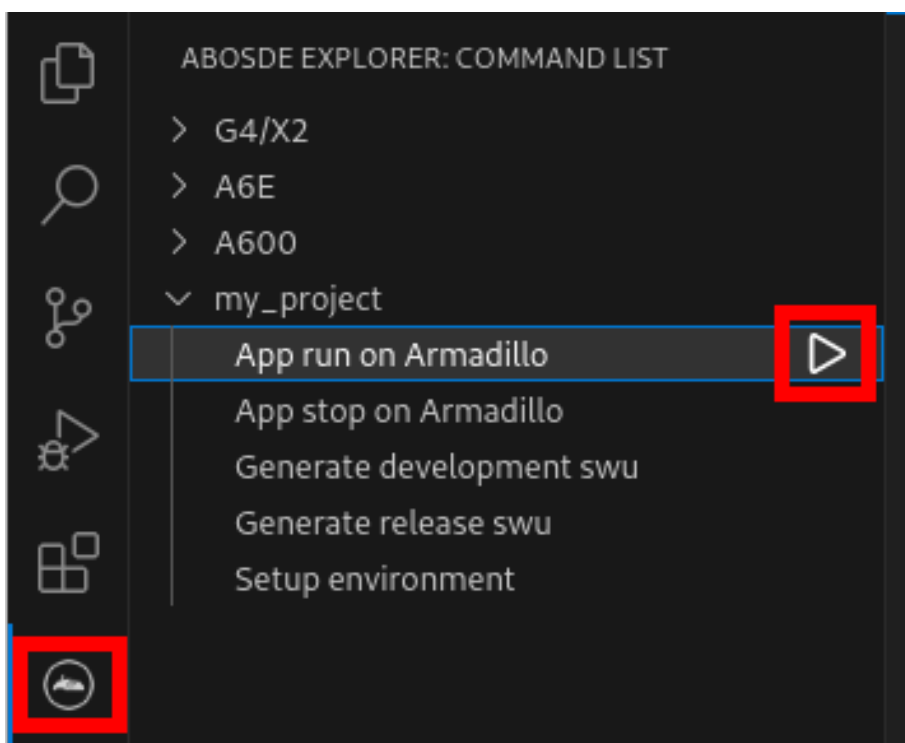


図 10.15 Armadillo 上でアプリケーションを実行する

VSCode のターミナルに以下のメッセージが表示されることがあります。これが表示された場合は yes と入力して下さい。

```
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

図 10.16 実行時に表示されるメッセージ

アプリケーションを終了するには VSCode の左ペインの [my\_project] から [App stop on Armadillo] を実行してください。

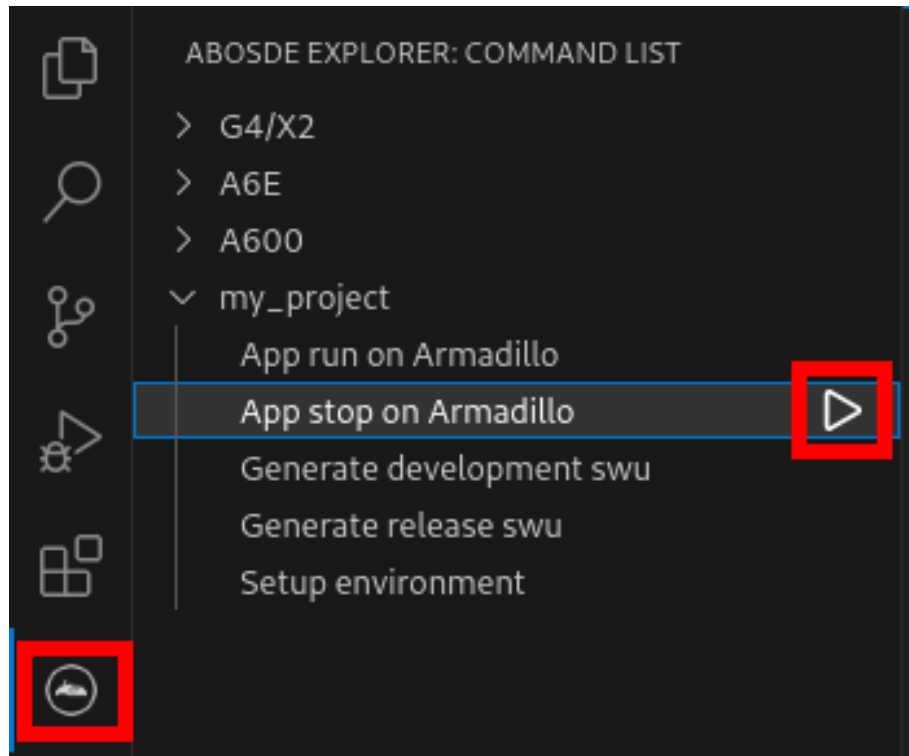


図 10.17 アプリケーションを終了する

### 10.1.5. リリース版のビルド

ここでは完成したアプリケーションをリリース版としてビルドする場合の手順について説明します。

VSCode の左ペインの [my\_project] から [Generate release swu] を実行すると、リリース版のアプリケーションを含んだ SWU イメージが作成されます。事前に「10.7.2. SWU イメージの作成」を参照して SWU の初期設定を行ってください。

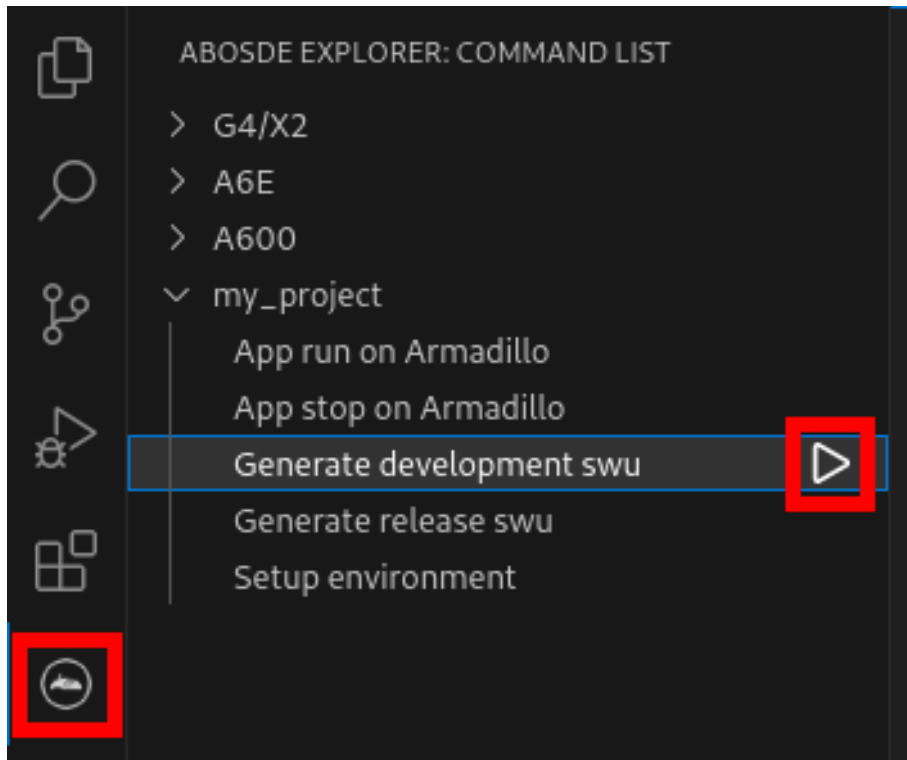


図 10.18 リリース版をビルドする

### 10.1.6. 製品への書き込み

作成した SWU イメージは `my_project/swu` ディレクトリ下に `release.swu` というファイル名で保存されています。

この SWU イメージを「10.7.3. イメージのインストール」を参照して Armadillo へインストールすると、Armadillo 起動時にアプリケーションも自動起動します。

## 10.2. アプリケーションをコンテナで実行する

### 10.2.1. Podman - コンテナ仮想化ソフトウェア

#### 10.2.1.1. Podman - コンテナ仮想化ソフトウェアとは

コンテナとはホスト OS 上に展開される仮想的なユーザ空間のことです。コンテナを使用することで複数の Armadillo-640 でも同一の環境がすぐに再現できます。ゲスト OS を必要としない仮想化であるため、アプリケーションの起動が素早いという特徴があります。

Podman とはこのようなコンテナを管理するためのソフトウェアであり、使用方法はコンテナ管理ソフトウェアの 1 つである Docker と互換性があります。

#### 10.2.2. コンテナを操作する

この章では、コンテナ仮想化ソフトウェアの 1 つである Podman の基本的な使い方について説明します。Armadillo-640 で実行させたいアプリケーションとその実行環境自体を 1 つの Podman イメージとして扱うことで、複数の Armadillo-640 がある場合でも、全てのボード上で同一の環境を再現させることが可能となります。

この章全体を通して、イメージの公開・共有サービスである Docker Hub [https://hub.docker.com] から取得した、Alpine Linux のイメージを使って説明します。

### 10.2.2.1. イメージからコンテナを作成する

イメージからコンテナを作成するためには、`podman start` コマンドを実行します。podman や docker にすでに詳しいかたは `podman run` コマンドでも実行できますが、ここでは「10.3. コンテナの運用」で紹介するコンテナの自動起動の準備も重ねて `podman start` を使います。イメージは Docker Hub [https://hub.docker.com] から自動的に取得されます。ここでは、簡単な例として `ls /` コマンドを実行するコンテナを作成します。

```
[armadillo ~]# vi /etc/atmark/containers/my_container.conf ❶
set_image docker.io/alpine
set_command ls /
[armadillo ~]# podman pull docker.io/alpine ❷
Trying to pull docker.io/library/alpine:latest...
Getting image source signatures
: (省略)
Writing manifest to image destination
Storing signatures
a6215f271958c760a2975a6765016044115dbae4b90f414eba3a448a6a26b4f6
[armadillo ~]# podman_start my_container ❸
Starting 'my_container'
b141e899b5ef7c9ec5434bda8f6a83d3e6bfc94f74bfb5dcef2a22041c71fdbf
[armadillo ~]# podman logs my_container ❹
bin
dev
: (省略)
usr
var
[armadillo ~]#
```

図 10.19 コンテナを作成する実行例

- ❶ コンテナのコンフィグを作成します。このファイルでは、コンテナのイメージやコマンド、デバイスへのアクセス権限を設定します。詳しい設定の説明には「10.3.1. コンテナの自動起動」を参照ください。
- ❷ コンテナのイメージを取得します。イメージが Armadillo に置いてない場合は「Error: docker.io/alpine: image not known」の様なエラーで失敗します。
- ❸ コンテナを起動します。これは Armadillo 起動時に自動的に起動されるコンテナと同じものになります。自動起動が不要な場合には `set_autostart no` で無効化できます。
- ❹ `podman logs` コマンドで出力を確認します。

"ls /" を実行するだけの "my\_container" という名前のコンテナが作成されました。コンテナが作成されると同時に "ls /" が実行され、その結果がログに残ります。ここで表示されているのは、コンテナ内部の "/" ディレクトリのフォルダの一覧です。




コンフィグファイルの直接な変更と `podman pull` によるコンテナの取得はデフォルト状態ではメモリ上でしか保存されません。

ファイルは `persist_file` で必ず保存し、コンテナイメージは `abos-ctrl podman-storage --disk` で `podman` のストレージを eMMC に切り替えるか `abos-ctrl podman-rw` で一時的に eMMC に保存してください。

運用中の Armadillo には直接に変更をせず、`swupdate` でアップデートしてください。

コンフィグファイルを保存して、`set_autostart no` を設定しない場合は自動起動します。



`podman_start` でコンテナが正しく起動できない場合は `podman_start -v <my_container>` で `podman run` のコマンドを確認し、`podman logs <my_container>` で出力を確認してください。

### 10.2.2.2. イメージ一覧を表示する

コンテナを作成するためのイメージは、イメージ一覧を表示する `podman images` コマンドで確認できます。

```
[armadillo ~]# podman images
REPOSITORY          TAG       IMAGE ID       CREATED        SIZE
docker.io/library/alpine latest    9c74a18b2325  2 weeks ago   4.09 MB
```

図 10.20 イメージ一覧の表示実行例

`podman images` コマンドの詳細は `--help` オプションで確認できます。

```
[armadillo ~]# podman images --help
```

図 10.21 podman images --help の実行例

### 10.2.2.3. コンテナ一覧を表示する

作成済みコンテナ一覧を表示するためには `podman ps` コマンドを実行します。

```
[armadillo ~]# podman ps -a
CONTAINER ID  IMAGE                                COMMAND      CREATED        STATUS
PORTS        NAMES
d6de5881b5fb  docker.io/library/alpine:latest    ls /        12 minutes ago Exited (0) 11 minutes ago
ago          my_container
```

図 10.22 コンテナ一覧の表示実行例

一覧表示により、コンテナ名やコンテナ ID を確認することができます。-a オプションを付けない場合は、動作中のコンテナのみ表示されます。`podman ps` コマンドの詳細は `--help` オプションで確認できます。

```
[armadillo ~]# podman ps --help
```

図 10.23 podman ps --help の実行例

#### 10.2.2.4. コンテナを起動する

作成済みのコンテナを起動するためには podman start コマンドを実行します。

```
[armadillo ~]# podman start my_container
podman start my_container
[ 3119.081068] IPv6: ADDRCONF(NETDEV_CHANGE): vethe172e161: link becomes ready
[ 3119.088214] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[ 3119.094812] cni-podman0: port 1(ve172e161) entered blocking state
[ 3119.101231] cni-podman0: port 1(ve172e161) entered disabled state
[ 3119.107745] device vethe172e161 entered promiscuous mode
[ 3119.113185] cni-podman0: port 1(ve172e161) entered blocking state
[ 3119.119546] cni-podman0: port 1(ve172e161) entered forwarding state
my_container
[ 3119.620731] cni-podman0: port 1(ve172e161) entered disabled state
[ 3119.627696] device vethe172e161 left promiscuous mode
[ 3119.632762] cni-podman0: port 1(ve172e161) entered disabled state
```

図 10.24 コンテナを起動する実行例

-a オプションを与えると、コンテナ内で実行されたアプリケーションの出力を確認できます。

```
[armadillo ~]# podman start -a my_container
[ 3150.303962] IPv6: ADDRCONF(NETDEV_CHANGE): vetha9ef8f8e: link becomes ready
[ 3150.311106] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[ 3150.317703] cni-podman0: port 1(vetha9ef8f8e) entered blocking state
[ 3150.324139] cni-podman0: port 1(vetha9ef8f8e) entered disabled state
[ 3150.330687] device vetha9ef8f8e entered promiscuous mode
[ 3150.336085] cni-podman0: port 1(vetha9ef8f8e) entered blocking state
[ 3150.342443] cni-podman0: port 1(vetha9ef8f8e) entered forwarding state
bin  etc  lib  mnt  proc  run  srv  tmp  var
dev  home media opt  root  sbin sys  usr
[ 3150.804164] cni-podman0: port 1(vetha9ef8f8e) entered disabled state
[ 3150.811249] device vetha9ef8f8e left promiscuous mode
[ 3150.816349] cni-podman0: port 1(vetha9ef8f8e) entered disabled state
```

図 10.25 コンテナを起動する実行例(a オプション付与)

ここで起動している my\_container は、起動時に "ls /" を実行するようになっているので、その結果が出力されます。podman start コマンドの詳細は --help オプションで確認できます。

```
[armadillo ~]# podman start --help
```

図 10.26 podman start --help 実行例

### 10.2.2.5. コンテナを停止する

動作中のコンテナを停止するためには `podman stop` コマンドを実行します。

```
[armadillo ~]# podman stop my_container
my_container
```

図 10.27 コンテナを停止する実行例

`podman stop` コマンドの詳細は `--help` オプションで確認できます。

```
[armadillo ~]# podman stop --help
```

図 10.28 `podman stop --help` 実行例

### 10.2.2.6. コンテナの変更を保存する

コンテナに対して変更が行われた状態で、そのままコンテナを停止してしまうと変更が失われてしまいます。

変更を保存するには二つの方法があります。

1. `podman commit` コマンドで保存する。

```
[armadillo ~]# podman commit my_container image_name:latest
Getting image source signatures
Copying blob f4ff586c6680 skipped: already exists
Copying blob 3ae0874b0177 skipped: already exists
Copying blob ea59ffe27343 done
Copying config 9ca3c55246 done
Writing manifest to image destination
Storing signatures
9ca3c55246eaac267a71731bad6bfe4b0124afcdd2b80c4f730c46aae17a88f3
```

図 10.29 `my_container` を保存する例

`podman commit` で保存する度に、変更が行なわれた差分が保存されます。繰り返し差分を保存すると、イメージサイズが大きくなってしまいます。ストレージ容量が不足する場合は、ベースとなる OS のイメージから作り直してください。

2. ボリュームを使用する。

`podman start` の `add_volumes` コマンドでコンテナに Armadillo Base OS のディレクトリをコンテナで使うことができます。

保存するデータの性質によって、保存先を選択してください。

1. `/var/app/volumes/myvolume`: アップデートした場合はコピーされません。ログやデータベースなど、アプリケーションが作成し続けるようなデータの保存に向いています。

2. myvolume が /var/app/rollback/volumes/myvolume: アップデートの際にコピーしてアップデートを行うので、アップデート中でも安全に使いつづけます。アプリケーションと一緒にアップデートするようなデータの保存に向いています。



コンテナを前のバージョンに戻した場合(ロールバック)、/var/app/rollback/volumes/ のデータの前のバージョンに戻ります。

その為、アプリケーションのバージョンに依存するようなデータは /var/app/rollback/volumes/ に入れることを推奨します。

mkswu の swdesc\_files (--extra-os 無し) と podman\_start` の add\_volumes では、相対パスはそのディレクトリをベースにします。/var/app/rollback/volumes/myvolume は myvolume で簡潔に指定できます。



Copy-on-Write (CoW) について。

この二つの volumes ディレクトリは btrfs と呼ばれるファイルシステムに保存されています。btrfs ではデータは Copy on Write (CoW) を使ってデータ完全性を保証しますが、その保証にはコストがあります。

数百 MB のファイルに小さな変更を頻繁に行う場合 CoW を無効化することを推奨します。CoW を無効化されたファイルにチェックサムが入らなくなりますので、極端な場合以外に残してください。

```
[armadillo ~]# cd /var/app/volumes/
[armadillo /var/app/volumes]# mkdir database
[armadillo /var/app/volumes]# chattr +C database ❶
[armadillo /var/app/volumes]# echo example data > database/
example
[armadillo /var/app/volumes]# lsattr database/ ❷
-----C----- database/example
```



図 10.30 chattr によって copy-on-write を無効化する例

- ❶ chattr +C でディレクトリに NoCow を設定します。これから作成されるファイルが NoCow で作成されます。すでに存在していたファイルに影響ないのでご注意ください。
- ❷ lsattr 確認します。リストの C の字があればファイルが「no cow」です。

### 10.2.2.7. コンテナの自動作成やアップデート

podman run, podman commit でコンテナを作成できますが、定期的にアップデートをする際にはコンテナの作成やアップデートを自動化できると便利です。



これを実現するために、Dockerfile と podman build を使います。この手順は Armadillo で実行可能です。

## 1. イメージを docker.io のイメージから作りなおします

```
[armadillo ~/podman-build]# cat Dockerfile
FROM docker.io/arm32v7/alpine:latest

# update & install dependencies (example: usbutils)
RUN apk upgrade && apk add usbutils && rm -f /var/cache/apk/*

# copy our application and set it to run on start
COPY my_application /my_application
ENTRYPOINT /my_application

[armadillo ~/podman-build]# podman build -t my_image:1 -t my_image:latest .
STEP 1: FROM docker.io/arm32v7/alpine:latest
STEP 2: RUN apk upgrade && apk add usbutils && rm -f /var/cache/apk/*
--> 234bf79175e
STEP 3: COPY my_application /my_application
--> 05ab31bb278
STEP 4: ENTRYPOINT /my_application
STEP 5: COMMIT my_image:latest
--> 590e3ba6d55
Successfully tagged localhost/my_image:1
Successfully tagged localhost/my_image:latest
590e3ba6d55f3e29bdef158d7283e9c4f7515567b2d3f978cfab2510dc02376b

[armadillo ~/podman-build]# podman save my_image:latest -o my_image_1.tar
```

### 図 10.31 podman build の実行例

## 2. イメージを前のバージョンからアップデートします

```
[armadillo ~/podman-build-update]# cat Dockerfile
FROM localhost/my_image:latest

# update OS packages
RUN apk upgrade --no-cache

# update application
COPY my_application /my_application
[armadillo ~/podman-build-update]# podman build -t my_image:2 -t my_image:latest .
STEP 1: FROM localhost/my_image:latest
STEP 2: RUN apk upgrade --no-cache
--> cf1dc0d7296
STEP 3: COPY my_application /my_application
STEP 4: COMMIT my_image:latest
--> 9e9d9366072
Successfully tagged localhost/my_image:2
Successfully tagged localhost/my_image:latest
9e9d9366072751007b2e70544d76c46b95a7a5a02df658ef0fa3f7dccc8850a
```

```
[armadillo ~/podman-build-update]# podman save -o my_image_2.tar my_image:2
```

### 図 10.32 podman build でのアップデートの実行例

この場合、`podman_partial_image` コマンドを使って、差分だけをインストールすることもできます。

```
[armadillo ~/podman-build-update]# podman_partial_image -b my_image:1 \
-o my_image_2_partial.tar my_image:2

[armadillo ~/podman-build-update]# ls -lh
-rw-r--r-- 1 root root 88 Dec 21 15:24 Dockerfile
-rw-r--r-- 1 root root 9.4M Dec 21 15:26 my_image_1.tar
-rw-r--r-- 1 root root 9.4M Dec 21 15:26 my_image_2.tar
-rw-r--r-- 1 root root 51K Dec 21 15:26 my_image_2_partial.tar
```

作成した .tar アーカイブは「10.7.6. mkswu の desc ファイル」の `swdesc_embed_container` と `swdesc_usb_container` で使えます。

### 10.2.2.8. コンテナを削除する

作成済みコンテナを削除する場合は `podman rm` コマンドを実行します。

```
[armadillo ~]# podman rm my_container
d6de5881b5fb973227b84d1d74abf269ac3183aad7e18b7a9d85208632641d94
[armadillo ~]# podman ps -a
CONTAINER ID  IMAGE  COMMAND  CREATED  STATUS
PORTS  NAMES
```

### 図 10.33 コンテナを削除する実行例

`podman ps` コマンドの出力結果より、コンテナが削除されていることが確認できます。`podman rm` コマンドの詳細は `--help` オプションで確認できます。

```
[armadillo ~]# podman rm --help
```

### 図 10.34 \$ podman rm --help 実行例

### 10.2.2.9. イメージを削除する

`podman` のイメージを削除するには `podman rmi` コマンドを実行します。イメージを削除するためには、そのイメージから作成したコンテナを先に削除しておく必要があります。`podman rmi` コマンドにはイメージ ID を指定する必要があるため、`podman images` コマンドで確認します。

```
[armadillo ~]# podman rm my_container
[armadillo ~]# podman images
REPOSITORY  TAG  IMAGE ID  CREATED  SIZE
docker.io/library/alpine  latest  02480aeb44d7  2 weeks ago  5.62 MB
[armadillo ~]# podman rmi 02480aeb44d7
```

```

Untagged: docker.io/library/alpine:latest
Deleted: 02480aeb44d78f1a44b8791af7edf7d6e1b18707397a1dfb3ff4f21c5ce4a44f
[armadillo ~]# podman images
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
    
```

図 10.35 イメージを削除する実行例

podman images コマンドの出力結果より、コンテナが削除されていることが確認できます。podman rmi コマンドの詳細は --help オプションで確認できます。

```
[armadillo ~]# podman rmi --help
```

図 10.36 podman rmi --help 実行例



SWU で転送されたイメージは podman images で Read-Only として表示されますので、podman rmi を実行するとエラーとなります。その場合は abos-ctrl podman-rw rmi をご使用ください。abos-ctrl podman-rw については「10.3.5.2. イメージを eMMC に保存する方法」を参照してください。

```

[armadillo ~]# podman images
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE          R/O
docker.io/library/alpine  latest      02480aeb44d7    2 weeks ago    5.62
MB      true
[armadillo ~]# podman rmi docker.io/alpine
Error: cannot remove read-only image
"02480aeb44d78f1a44b8791af7edf7d6e1b18707397a1dfb3ff4f21c5ce4a44f"
[armadillo ~]# abos-ctrl podman-rw rmi docker.io/alpine
Untagged: docker.io/library/alpine:latest
Deleted:
02480aeb44d78f1a44b8791af7edf7d6e1b18707397a1dfb3ff4f21c5ce4a44f
[armadillo ~]# podman images
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
    
```

図 10.37 Read-Only のイメージを削除する実行例

### 10.2.2.10. コンテナとコンテナに関連するデータを削除する

abos-ctrl container-clear を使用すると、コンテナ、コンテナイメージ、コンテナに関するデータを削除することができます。



全てのコンテナとコンテナイメージ、コンテナに関するデータが削除されるため、十分に注意して使用してください。

abos-ctrl container-clear は以下の通り動作します。

- ・ 以下のファイル、ディレクトリ配下のファイルを削除
  - ・ /var/app/rollback/volumes/
  - ・ /var/app/volumes/
  - ・ /etc/atmark/containers/\*.conf
- ・ 以下のファイルで container を含む行を削除
  - ・ /etc/sw-versions
  - ・ /etc/swupdate.watch

```
[armadillo ~]# abos-ctrl container-clear
This command will remove all containers and related data.
- The following file and directories will be removed:
  - /var/app/rollback/volumes/
  - /var/app/volumes/
  - /etc/atmark/containers/*.conf
- Lines containing the word "container" will be deleted from the following files:
  - /etc/sw-versions
  - /etc/swupdate.watch
Continue? [y/N]
y
Remove all container data succeeded
```

図 10.38 abos-ctrl container-clear 実行例

### 10.2.2.11. 実行中のコンテナに接続する

実行中のコンテナに接続し、コンテナ内で指定したコマンドを実行するには `podman exec` コマンドを実行します。`podman exec` コマンドでコンテナ内部のシェルを起動すると、コンテナ内部を操作できるようになります。ここでは、`sleep infinity` コマンドを実行して待ち続けるだけのコンテナを作成し、そのコンテナに対して `podman exec` コマンドでシェルを起動する例を示します。

```
[armadillo ~]# vi /etc/atmark/containers/sleep_container.conf
set_image docker.io/alpine
set_command sleep infinity
[armadillo ~]# podman_start sleep_container
Starting 'test'
f62e7a666d7156d261905c8406c72fc271534fa29e69771c76f4f6660a2da41a
[armadillo ~]# podman exec -it sleep_container sh
[container ~]# ps
PID  USER  TIME  COMMAND
  1  root   0:00 /run/podman-init -- sleep infinity
  2  root   0:00 sleep infinity
  3  root   0:00 sh
  4  root   0:00 ps
```

図 10.39 コンテナ内部のシェルを起動する実行例

`podman_start` コマンドでコンテナを作成し、その後作成したコンテナ内で `sh` を実行しています。`sh` を実行すると、コンテナ内のプロンプトが表示されコンテナ内部を操作できるようになります。上記で

はコンテナ内で、ps コマンドを実行しています。コンテナ作成時に実行した sleep と podman exec で実行した sh がプロセスとして存在していることが確認できます。

コンテナ内のシェルから抜ける時は exit コマンドを実行します。

```
[container ~]# exit
```

#### 図 10.40 コンテナ内部のシェルから抜ける実行例

podman exec コマンドから抜けても、コンテナがまだ実行中です。コンテナを停止したい場合は podman stop sleep\_container か podman kill sleep\_container で停止して podman rm sleep\_container でそのコンテナを削除してください。

podman exec コマンドの詳細は --help オプションで確認できます。

```
[armadillo ~]# podman exec --help
```

#### 図 10.41 podman exec --help 実行例

### 10.2.2.12. コンテナ間で通信をする

複数のコンテナを実行している環境で、それらのコンテナ間で通信を行う方法を示します。これにより、例えば SQL サーバを実行しているコンテナに対し別のコンテナから接続するといった使い方ができます。

コンテナには作成した時点でローカル IP アドレスが割り当てられるので、コンテナの名前かその IP アドレスで通信を行うことができます。

準備として、2 つのコンテナを作成します。

```
[armadillo ~]# vi /etc/atmark/containers/my_container_1.conf
set_image docker.io/alpine
set_command sleep infinity
[armadillo ~]# vi /etc/atmark/containers/my_container_2.conf
set_image docker.io/alpine
set_command sleep infinity
[armadillo ~]# podman_start my_container_1 my_container_2
Starting 'my_container_1'
cbe0802f4e2d2fec88f4e300dabeba3b48865359dc02cbd99375b1b38c2c28eb
Starting 'my_container_2'
5e645f5e40fc096ad0bea323a00bebebbda4bd825a5e8d12103f752d8868692e
```

#### 図 10.42 コンテナを作成する実行例

コンテナに割り当てられた IP アドレスを確認するには podman inspect コマンドを実行します。

```
[armadillo ~]# podman inspect --format='{{.NetworkSettings.IPAddress}}' my_container_1
10.88.0.108
```

```
[armadillo ~]# podman inspect --format='{{.NetworkSettings.IPAddress}}' my_container_2
10.88.0.109
```

### 図 10.43 コンテナの IP アドレスを確認する実行例

これらの IP アドレスを使って、一方のコンテナからもう一方のコンテナへ対し ping コマンドで疎通確認を行うことができます。

```
[armadillo ~]# podman exec -it my_container_1 sh
[container ~]# ping -c 2 my_container_2
PING my_container_2 (10.88.0.109): 56 data bytes
64 bytes from 10.88.0.109: seq=0 ttl=42 time=0.144 ms
64 bytes from 10.88.0.109: seq=1 ttl=42 time=0.210 ms

--- my_container_2 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.144/0.177/0.210 ms
[container ~]# ping -c 2 10.88.0.109
PING 10.88.0.109 (10.88.0.109): 56 data bytes
64 bytes from 10.88.0.109: seq=0 ttl=42 time=0.140 ms
64 bytes from 10.88.0.109: seq=1 ttl=42 time=0.138 ms

--- 10.88.0.109 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.138/0.139/0.140 ms
```

### 図 10.44 ping コマンドによるコンテナ間の疎通確認実行例

このように、my\_container\_1(10.88.0.108) から my\_container\_2(10.88.0.109) への通信が確認できます。

#### 10.2.2.13. 開発時に有用な—privileged オプション

コンテナに、全権限と全てのデバイスへのアクセスを許可するオプション `--privileged` があります。このオプションを利用すると、コンテナに与えるべき最小の権限を洗い出す必要が無いため、開発時に有用です。

実運用の際、このオプションを利用することはセキュリティー上問題がある為、開発時にのみご利用ください。コンテナに必要な最低限の権限を与えることをおすすめします。

#### 10.2.3. アットマークテクノが提供するイメージを使う

アットマークテクノは、動作確認環境として使用できる Debian ベースのイメージを提供しています。ここでは、Docker ファイルからイメージをビルドする方法と、すでにビルド済みのイメージを使う方法の 2 つについて説明します。

##### 10.2.3.1. Docker ファイルからイメージをビルドする

Armadillo-640 コンテナ [https://armadillo.atmark-techno.com/resources/software/armadillo-640/container] から「Debian [VERSION] サンプル Dockerfile」ファイル (at-debian-image-dockerfile-armv7-[VERSION].tar.gz) をダウンロードします。その後 podman build コマンドを実行します。

```
[armadillo ~]# tar xzf at-debian-image-dockerfile-armv7-[VERSION].tar.gz
[armadillo ~]# cd at-debian-image-dockerfile-armv7-[VERSION]
[armadillo ~]# abos-ctrl podman-storage --disk
[armadillo ~]# podman build -t at-debian-image-armv7:latest .
:
: (省略)
:
[armadillo ~]# podman images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
localhost/at-debian-image-armv7	latest	9df263360ab3	2 minutes ago	123 MB
docker.io/arm32v7/debian	bullseye	971ca2764db3	10 days ago	105 MB

図 10.45 Docker ファイルによるイメージのビルドの実行例

podman images コマンドにより at-debian-image-armv7 がビルドされたことが確認できます。docker.io/arm32v7/debian イメージはベースとなっている Debian イメージです。

### 10.2.3.2. ビルド済みのイメージを使用する

Armadillo-640 コンテナ [<https://armadillo.atmark-techno.com/resources/software/armadillo-640/container>] から「Debian [VERSION] サンプルコンテナイメージ」ファイル (at-debian-image-armv7-[VERSION].tar) をダウンロードします。その後 podman load コマンドを実行します。

```
[armadillo ~]# abos-ctrl podman-storage --disk
[armadillo ~]# podman load -i at-debian-image-armv7-[VERSION].tar
:
: (省略)
:
[armadillo ~]# podman images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
localhost/at-debian-image-armv7	latest	a947a7449be5	5 days ago	123 MB
localhost/at-debian-image-armv7	[VERSION]	a947a7449be5	5 days ago	123 MB

図 10.46 ビルド済みイメージを load する実行例

podman images コマンドにより at-debian-image-armv7 がビルドされたことが確認できます。

## 10.2.4. 入出力デバイスを扱う

この章では、コンテナ内で動作するアプリケーションから GPIO や I2C などの入出力デバイスを扱う方法について示します。基本的に、コンテナ内のアプリケーションからホスト OS 側のデバイスへアクセスすることはできません。このため、コンテナ内のアプリケーションからデバイスを扱うためには、コンテナ作成時に扱いたいデバイスを指定する必要があります。ここで示す方法は、扱いたいデバイスに関するデバイスツリーファイルが適切に設定されていることを前提としています。デバイスツリーファイルを設定していない場合は、適切に設定してください。

### 10.2.4.1. GPIO を扱う

コンテナ内で動作するアプリケーションから GPIO を扱うためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/gpiochipN を渡す必要があります。以下は、/dev/gpiochip0 を渡して alpine イメージからコンテナを作成する例です。/dev/gpiochipN を渡すと、GPION+1 を操作することができます。

```
[armadillo ~]# vi /etc/atmark/containers/gpio_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/gpiochip0
[armadillo ~]# podman_start gpio_example
Starting 'gpio_example'
956a0fecc48d5ea1210069910f7bb48b9e90b2dadbd12895064d9776dae0360b5
```

図 10.47 GPIO を扱うためのコンテナ作成例

コンテナ内に入ってコマンドで GPIO を操作する例を以下に示します。この例では GPIO1\_IO22(CON9 1 ピン) を操作しています。

```
[armadillo ~]# podman exec -it gpio_example sh
[container ~]# apk upgrade
[container ~]# apk add libgpiod
[container ~]# gpioget gpiochip0 22 ❶
1 ❷
[container ~]# gpioset gpiochip0 22=0 ❸
```

図 10.48 コンテナ内からコマンドで GPIO を操作する例

- ❶ GPIO 番号 22 の値を取得します。
- ❷ 取得した値を表示します。
- ❸ GPIO 番号 22 に 0(Low) を設定します。

他にも、`gpiodetect` コマンドで認識している `gpiochip` をリスト表示できます。以下の例では、コンテナを作成する際に渡した `/dev/gpiochip0` が認識されていることが確認できます。

```
[container ~]# gpiodetect
gpiochip0 [209c000.gpio] (32 lines)
```

図 10.49 `gpiodetect` コマンドの実行

`gpioinfo` コマンドでは、指定した `gpiochip` の詳細な情報を表示することができます。

```
[container ~]# gpioinfo gpiochip0
gpiochip0 - 32 lines:
    line 0:      unnamed      "dtr"  output  active-low [used]
    line 1:      unnamed      unused  input   active-high
    line 2:      unnamed      unused  input   active-high
    line 3:      unnamed      unused  input   active-high
    line 4:      unnamed      unused  input   active-high
    line 5:      unnamed      "red"   output  active-high [used]
    line 6:      unnamed      unused  input   active-high
    line 7:      unnamed      unused  input   active-high
    line 8:      unnamed      "green" output  active-high [used]
    line 9:      unnamed      unused  output  active-high
    line 10:     unnamed      "SW1"  input   active-low [used]
```



```

line 11:    unnamed      unused   input   active-high
line 12:    unnamed      unused   input   active-high
line 13:    unnamed      unused   input   active-high
line 14:    unnamed      unused   input   active-high
line 15:    unnamed      unused   input   active-high
line 16:    unnamed      unused   input   active-high
line 17:    unnamed      unused   input   active-high
line 18:    unnamed      "yellow" output  active-high [used]
line 19:    unnamed      "regulators:regulator-usbotg1vbu" output active-high [used]
line 20:    unnamed      unused   input   active-high
line 21:    unnamed      unused   input   active-high
line 22:    unnamed      unused   input   active-high
line 23:    unnamed      unused   input   active-high
line 24:    unnamed      unused   input   active-high
line 25:    unnamed      unused   input   active-high
line 26:    unnamed      unused   input   active-high
line 27:    unnamed      unused   input   active-high
line 28:    unnamed      unused   input   active-high
line 29:    unnamed      unused   input   active-high
line 30:    unnamed      unused   input   active-high
line 31:    unnamed      unused   input   active-high
    
```

図 10.50 gpioinfo コマンドの実行

CON9 のピン番号と GPIO の対応を次に示します。

表 10.1 CON9 ピンと GPIO の対応

ピン番号	GPIO
1	GPIO1_IO22
2	GPIO1_IO23
3	GPIO1_IO17
4	GPIO1_IO31
5	GPIO1_IO16
6	GPIO1_IO30
13	GPIO3_IO23
14	GPIO3_IO24
15	GPIO3_IO25
16	GPIO3_IO26
17	GPIO3_IO27
18	GPIO3_IO28
25	GPIO4_IO06
26	GPIO4_IO07
27	GPIO4_IO08
28	GPIO4_IO09



「表 10.1. CON9 ピンと GPIO の対応」の CON9 1, 3~6 ピンは初期出荷状態では GPIO として利用することができません。これらのピンを GPIO として利用する場合は、at-dtweb を用います。

at-dtweb の利用方法については「10.9. Device Tree をカスタマイズする」を参照してください。

C 言語プログラムから操作する場合は、GPIO 操作ライブラリである libgpiod を使用することができます。

### 10.2.4.2. I2C を扱う

コンテナ内で動作するアプリケーションから I2C を扱うためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/i2c-N を渡す必要があります。以下は、/dev/i2c-4 を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# vi /etc/atmark/containers/i2c_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/i2c-4
[armadillo ~]# podman_start i2c_example
Starting 'i2c_example'
efa1eb129c1f036a709755f0d53b21a0f2a39307ecae32b24aac98c0b6567bf0
```

図 10.51 I2C を扱うためのコンテナ作成例

コンテナ内に入り、i2c-tools に含まれる i2cdetect コマンドを使ってスレーブアドレスを確認することができます。

```
[armadillo ~]# podman exec -it i2c_example sh
[container ~]# apk upgrade
[container ~]# apk add i2c-tools
[container ~]# i2cdetect -y 4
   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
20:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
40:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
50:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
60:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
70:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
```

図 10.52 i2cdetect コマンドによる確認例

### 10.2.4.3. SPI を扱う

コンテナ内で動作するアプリケーションから SPI を扱うためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/spidevN.N を渡す必要があります。以下は、/dev/spidev1.0 を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# vi /etc/atmark/containers/spi_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/spidev1.0
[armadillo ~]# podman_start spi_example
```

```
Starting 'spi_example'
45302bc9f95eef0e25c5d98acf198d96fc5bec1f83e791018cbe4221cc1f4523
```

図 10.53 SPI を扱うためのコンテナ作成例

コンテナ内に入り、spi-tools に含まれる spi-config コマンドを使って現在の設定を確認することができます。

```
[armadillo ~]# podman exec -it spi_example sh
[container ~]# apk upgrade
[container ~]# apk add spi-tools
[container ~]# spi-config --device=/dev/spidev1.0 -q
/dev/spidev1.0: mode=0, lsb=0, bits=8, speed=500000, spiready=0
```

図 10.54 spi-config コマンドによる確認例

#### 10.2.4.4. CAN を扱う

コンテナ内で動作するアプリケーションから CAN 通信を行うためには、Podman のイメージからコンテナを作成する際に、コンテナを実行するネットワークとして host を、権限として NET\_ADMIN を指定する必要があります。以下は、ネットワークとして host を、権限として NET\_ADMIN を指定して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# vi /etc/atmark/containers/can_example.conf
set_image dockage.io/alpine
set_command sleep infinity
set_network host
add_args --cap-add=NET_ADMIN
[armadillo ~]# podman start can_example
Starting 'can_example'
73e7dbcce86e84eef337bbc5c580a747948b94b87015bb34143da341b8301c16a
```

図 10.55 CAN を扱うためのコンテナ作成例

コンテナ内に入り、ip コマンドで CAN を有効にすることができます。以下に、設定例を示します。

```
[armadillo ~]# podman exec -it can_example sh
[container ~]# apk upgrade
[container ~]# apk add iproute2 ❶
[container ~]# ip link set can0 type can bitrate 125000 ❷
[container ~]# ip link set can0 up ❸
[container ~]# ip -s link show can0 ❹
4: can0: <NOARP,UP,LOWER_UP,ECHO> mtu 16 qdisc pfifo_fast state UP mode DEFAULT
group default qlen 10
  link/can
  RX: bytes  packets  errors  dropped missed  mcast
     0         0        0       0       0       0
  TX: bytes  packets  errors  dropped carrier collsns
     0         0        0       0       0       0
```

図 10.56 CAN の設定例

- ① CAN の設定のために必要な iproute2 をインストールします。すでにインストール済みの場合は不要です。
- ② CAN の通信速度を 125000 kbps に設定します。
- ③ can0 インターフェースを起動します。
- ④ can0 インターフェースの現在の使用状況を表示します。

#### 10.2.4.5. PWM を扱う

コンテナ内で動作するアプリケーションから PWM を扱うためには、Podman のイメージからコンテナを作成する際にホスト OS 側の /sys ディレクトリを渡す必要があります。デフォルト状態でもマウントされていますが、読み取り専用になって使えませんのでご注意ください。以下は、/sys を渡して alpine イメージからコンテナを作成する例です。ここで渡された /sys ディレクトリはコンテナ内の同じ /sys にマウントされます。

```
[armadillo ~]# vi /etc/atmark/containers/pwm_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_volumes /sys
[armadillo ~]# podman_start pwm_example
Starting 'pwm_example'
212127a8885e106e0ef7453545db3c473aef5438f000acf4b33a44d75dcd9e28
```

図 10.57 PWM を扱うためのコンテナ作成例

コンテナ内に入り、/sys/class/pwm/pwmchipN ディレクトリ内の export ファイルに 0 を書き込むことで扱えるようになります。以下に、/sys/class/pwm/pwmchip0 を扱う場合の動作設定例を示します。

```
[armadillo ~]# podman exec -it pwm_example sh
[container ~]# echo 0 > /sys/class/pwm/pwmchip0/export ①
[container ~]# echo 1000000000 > /sys/class/pwm/pwmchip0/pwm0/period ②
[container ~]# echo 500000000 > /sys/class/pwm/pwmchip0/pwm0/duty_cycle ③
[container ~]# echo 1 > /sys/class/pwm/pwmchip0/pwm0/enable ④
```

図 10.58 PWM の動作設定例

- ① pwmchip0 を export します。
- ② 周期を 1 秒にします。単位はナノ秒です。
- ③ PWM の ON 時間を 0.5 秒にします。
- ④ PWM 出力を有効にします。

#### 10.2.4.6. シリアルインターフェースを扱う

コンテナ内で動作するアプリケーションから RS-232C や RS-485 などのシリアル通信を行うためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/ttymxn を渡す必要があります。以下は、/dev/ttymx2 を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# vi /etc/atmark/containers/serial_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/ttyxc2
[armadillo ~]# podman_start serial_example
Starting 'serial_example'
3999f09d51253371cacffd68967c90fdd5250770888a82f59d7810b54fcc873e
```

図 10.59 シリアルインターフェースを扱うためのコンテナ作成例

コンテナ内に入り、setserial コマンドを使って現在の設定を確認することができます。

```
[armadillo ~]# podman exec -it serial_example sh
[container ~]# setserial -a /dev/ttyxc2
/dev/ttyxc2, Line 2, UART: undefined, Port: 0x0000, IRQ: 52
    Baud_base: 5000000, close_delay: 50, divisor: 0
    closing_wait: 3000
    Flags: spd_normal
```

図 10.60 setserial コマンドによるシリアルインターフェイス設定の確認例

#### 10.2.4.7. USB を扱う

コンテナ内で動作するアプリケーションから USB 接続のデバイスを扱うための方法について示します。

- ・ USB シリアルデバイスを扱う

USB シリアルデバイスをコンテナ内から扱う場合には、Podman のイメージからコンテナを作成する際にホスト OS 側の /dev/ttyUSB $N$  を渡す必要があります。以下は、/dev/ttyUSB0 を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# vi /etc/atmark/containers/usb_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/ttyUSB0
[armadillo ~]# podman_start usb_example
Starting 'usb example'
34cb0e60d6274ac1df87aed58a461bcf56d0c117c4d377af130605ea399e0950
```

図 10.61 USB シリアルデバイスを扱うためのコンテナ作成例

コンテナ内に入り、setserial コマンドを使って現在の設定を確認することができます。

```
[armadillo ~]# podman exec -it usb_example sh
[container ~]# setserial -a /dev/ttyUSB0
/dev/ttyUSB0, Line 0, UART: unknown, Port: 0x0000, IRQ: 0
    Baud_base: 24000000, close_delay: 0, divisor: 0
    closing_wait: infinite
    Flags: spd_normal
```

図 10.62 setserial コマンドによる USB シリアルデバイス設定の確認例

- ・ USB カメラを扱う

USB カメラをコンテナ内から扱う場合には、Podman のイメージからコンテナを作成する際にホスト OS 側の `/dev/videoN` を渡す必要があります。以下は、`/dev/video0` を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# vi /etc/atmark/containers/usbcam_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/video0
[armadillo ~]# podman_start usbcam_example
Starting 'usbcam_example'
ffe06090b45826cc0b1c7710e9e850ba9521d36b70de4288d0dfe1fe91a35632
[armadillo ~]# podman exec -it usbcam_example sh
[container ~]# ls /dev/video0
/dev/video0
```

図 10.63 USB カメラを扱うためのコンテナ作成例

GStreamer などのマルチメディアフレームワークと組み合わせることで、USB カメラからの映像のキャプチャが可能となります。

- ・ USB メモリを扱う

ここでは、USB メモリを扱う方法について 2 つの例を示します。

- ・ ホスト OS 側でマウントした USB メモリをコンテナから扱う

あらかじめホスト OS 側でマウントしてある USB メモリをコンテナから扱う場合には、Podman のイメージからコンテナを作成する際にホスト OS 側で USB メモリをマウントしてるディレクトリを渡す必要があります。

```
[armadillo ~]# mount -t vfat /dev/sda1 /mnt
[armadillo ~]# echo test >> /mnt/sample.txt
[armadillo ~]# ls /mnt
sample.txt
```

図 10.64 USB メモリをホスト OS 側でマウントする例

上記の例では、USB メモリを `/mnt` にマウントしました。以下は、`/mnt` を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# vi /etc/atmark/containers/usbmem_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_volumes /mnt
[armadillo ~]# podman_start usbmem_example
Starting 'usbmem_example'
ef77d4bfd5b04f3b8b5ddcb5bfac321304fa64219a4b88c3130e45e5a14e1b3e
```

図 10.65 ホスト OS 側でマウント済みの USB メモリを扱うためのコンテナ作成例

ホスト OS 側の /mnt ディレクトリをコンテナ内の /mnt にマウントしています。これにより、コンテナ内からも /mnt ディレクトリを通して USB メモリを扱うことができます。

```
[armadillo ~]# podman exec -it usbmem_example sh
[container ~]# ls /mnt
sample.txt
[container ~]# cat /mnt/sample.txt
test
```

図 10.66 USB メモリに保存されているデータの確認例

- ・ USB メモリをコンテナ内からマウントする

USB メモリをコンテナ内からマウントして扱う場合には、Podman のイメージからコンテナを作成する際にホスト OS 側の /dev ディレクトリを渡すと同時に、適切な権限も渡す必要があります。以下は、/dev を渡して alpine イメージからコンテナを作成する例です。権限として SYS\_ADMIN を渡しています。

```
[armadillo ~]# vi /etc/atmark/containers/usbmem_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_args --cap-add=SYS_ADMIN
add_devices /dev/sda1
[armadillo ~]# podman start usbmem_example
Starting 'usbmem_example'
387a2256530e9b35b5361ca681a99fba8f46d78b6a6cb8ecd60096246b9198a8
```

図 10.67 USB メモリをマウントするためのコンテナ作成例

コンテナ内に入り、mount コマンドで USB メモリを /mnt にマウントし、保存されているデータを確認することができます。

```
[armadillo ~]# podman exec -it usbmem_example sh
[container ~]# mount /dev/sda1 /mnt
[container ~]# ls /mnt
sample.txt
[container ~]# cat /mnt/sample.txt
test
```

図 10.68 コンテナ内から USB メモリをマウントする例

#### 10.2.4.8. RTC を扱う

コンテナ内から RTC を扱うためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/rtcN を渡すと同時に、RTC への時刻の設定を行うための権限も渡す必要があります。以下は、/dev/rtc0 を渡して alpine イメージからコンテナを作成する例です。権限として SYS\_TIME も渡しています。

```
[armadillo ~]# vi /etc/atmark/containers/rtc_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_args --cap-add=SYS_TIME
```

```
add_devices /dev/rtc0
[armadillo ~]# podman_start rtc_example
Starting 'rtc_example'
025209e0d96f43c2911239a8397b7002c3eaab057e031d8abb765df5707d75bd
```

図 10.69 RTC を扱うためのコンテナ作成例

コンテナ内に入り、hwclock コマンドで RTC の時刻表示と設定ができます。

```
[armadillo ~]# podman exec -it rtc_example sh
[container ~]# hwclock ❶
Thu Feb 18 05:14:37 2021 0.000000 seconds
[container ~]# date --set "2021-04-01 09:00:00" ❷
Thu Apr 1 09:00:00 UTC 2021
[container ~]# hwclock --systohc ❸
[container ~]# hwclock ❹
Thu Apr 1 09:00:28 2021 0.000000 seconds
```

図 10.70 hwclock コマンドによる RTC の時刻表示と設定例

- ❶ RTC に設定されている現在時刻を表示します。
- ❷ システム時刻を 2021 年 4 月 1 日 9 時 0 分 0 秒に設定します。
- ❸ システム時刻を RTC に反映させます。
- ❹ RTC に設定されている時刻が変更されていることを確認します。

#### 10.2.4.9. 音声出力を行う

Armadillo-640 に接続したスピーカーなどの音声出力デバイスへコンテナ内から音声を出力するためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/snd を渡す必要があります。以下は、/dev/snd を渡して debian イメージからコンテナを作成する例です。

```
[armadillo ~]# vi /etc/atmark/containers/snd_example.conf
set_image localhost/at-debian-image-armv7
set_command sleep infinity
add_devices /dev/snd
[armadillo ~]# podman_start snd_example
Starting 'snd_example'
b921856b504e9f0a3de2532485d7bd9adb1ff63c2e10bfdaccd1153fd36a3c1d
```

図 10.71 音声出力を行うためのコンテナ作成例

コンテナ内に入り、alsa-utils などのソフトウェアで音声出力を行えます。

```
[armadillo ~]# podman exec -it snd_example /bin/bash
[container ~]# apt update && apt upgrade
[container ~]# apt install alsa-utils ❶
```



```
[container ~]# /etc/init.d/alsa-utils start ❷
[container ~]# aplay -D hw:N,M [ファイル名] ❸
```

図 10.72 alsa-utils による音声出力を行う例

- ❶ alsa-utils をインストールします。
- ❷ alsa-utils を起動します。
- ❸ 指定したファイル名の音声ファイルを再生します。

aplay の引数にある、M は音声を出力したい CARD 番号、N はデバイス番号を表しています。CARD 番号とデバイス番号は、aplay コマンドに -l オプションを与えることで確認できます。

#### 10.2.4.10. ユーザースイッチのイベントを取得する

Armadillo-640 にはユーザースイッチが実装されています。これらのスイッチのプッシュ/リリースイベントを取得するためには、Podman のイメージからコンテナを作成する際にホスト OS 側の /dev/input ディレクトリを渡す必要があります。以下は、/dev/input を渡して alpine イメージからコンテナを作成する例です。ここで渡された /dev/input ディレクトリはコンテナ内の /dev/input にマウントされます。

```
[armadillo ~]# vi /etc/atmark/containers/sw_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/input
[armadillo ~]# podman_start sw_example
Starting 'sw_example'
c0cd8b801883266197a3c20552b0e8b6c7dd473bb0b24e05bf3ecdb581c822b9
```

図 10.73 ユーザースイッチのイベントを取得するためのコンテナ作成例

コンテナ内に入り、evtest コマンドでイベントを確認できます。

```
[armadillo ~]# podman exec -it sw_example sh
[container ~]# apk upgrade
[container ~]# apk add evtest
[container ~]# evtest /dev/input/event0
Input driver version is 1.0.1
Input device ID: bus 0x19 vendor 0x1 product 0x1 version 0x100
Input device name: "gpio-keys"
Supported events:
  Event type 0 (EV_SYN)
  Event type 1 (EV_KEY)
    Event code 28 (KEY_ENTER)
Properties:
Testing ... (interrupt to exit)
Event: time 1685517999.767274, type 1 (EV_KEY), code 28 (KEY_ENTER), value 0 ❶
Event: time 1685517999.767274, ----- SYN_REPORT -----
Event: time 1685517999.907279, type 1 (EV_KEY), code 28 (KEY_ENTER), value 1 ❷
Event: time 1685517999.907279, ----- SYN_REPORT -----
```

図 10.74 evtest コマンドによる確認例

- ❶ SW1 のボタン **プッシュ** イベントを検出したときの表示
- ❷ SW1 のボタン **リリース** イベントを検出したときの表示

### 10.2.4.11. LED を扱う

Armadillo-640 には LED が実装されています。これらの LED を扱うためには、Podman のイメージからコンテナを作成する際にホスト OS 側の /sys ディレクトリを渡す必要があります。以下は、/sys を渡して alpine イメージからコンテナを作成する例です。ここで渡された /sys ディレクトリはコンテナ内の /sys にマウントされます。

```
[armadillo ~]# vi /etc/atmark/containers/led_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_volumes /sys
[armadillo ~]# podman_start led_example
Starting 'led_example'
c770f76d7714f4cceb1229be2240382bded236c2c51bb6b866bc0098c2cb987a
```

図 10.75 LED を扱うためのコンテナ作成例

コンテナ内に入り、brightness ファイルに値を書き込むことで LED の点灯/消灯を行うことができます。0 を書き込むと消灯、0 以外の値 (1~255) を書き込むと点灯します。

```
[armadillo ~]# podman exec -it led_example sh
[container ~]# echo 0 > /sys/class/leds/red/brightness
[container ~]# echo 1 > /sys/class/leds/red/brightness
```

図 10.76 LED の点灯/消灯の実行例

LED クラスディレクトリと LED の対応は、「表 7.5. LED クラスディレクトリと LED の対応」を参照してください。

## 10.2.5. 近距離通信を行う

この章では、コンテナ内から近距離通信デバイスを扱う方法について示します。

### 10.2.5.1. Bluetooth デバイスを扱う

コンテナ内から Bluetooth を扱うには、コンテナ作成時にホストネットワークを使用するために、NET\_ADMIN の権限を渡す必要があります。「図 10.77. Bluetooth デバイスを扱うためのコンテナ作成例」、alpine イメージから Bluetooth を扱うコンテナを作成する例を示します。

```
[armadillo ~]# vi /etc/atmark/containers/bt_example.conf
set_image docker.io/alpine
set_command sleep infinity
set_network host
add_volumes /var/run/dbus/
add_args --cap-add=NET_ADMIN
[armadillo ~]# podman_start bt_example
```

```
Starting 'bt_example'
45fe1eb6b25529f0c84cd4b97ca1aef8451785fc9a87a67d54873c1ed45b70a4
```

### 図 10.77 Bluetooth デバイスを扱うためのコンテナ作成例

コンテナ内で必要なソフトウェアをインストールして、Bluetooth を起動します。

```
[armadillo ~]# podman exec -it bt_example sh
[container ~]# apk upgrade
[container ~]# apk add bluez dbus
[container ~]# dbus-daemon --system
[container ~]# /usr/lib/bluetooth/bluetoothd &
```

### 図 10.78 Bluetooth を起動する実行例

これにより、bluetoothctl で Bluetooth 機器のスキャンやペアリングなどが行えるようになります。以下に、bluetoothctl コマンドで周辺機器をスキャンしてペアリングを行う例を示します。

```
[container ~]# bluetoothctl
Agent registerd
[..CHG..] Controller XX:XX:XX:XX:XX:XX Pairable: yes
[bluetooth]# power on ❶
Changing power on succeeded
[..CHG..] Controller XX:XX:XX:XX:XX:XX Powered: yes
[bluetooth]# scan on ❷
Discovery started
[..CHG..] Controller XX:XX:XX:XX:XX:XX Discovering: yes
[..NEW..] Device AA:AA:AA:AA:AA:AA AA-AA-AA-AA-AA-AA
[..NEW..] Device BB:BB:BB:BB:BB:BB BB-BB-BB-BB-BB-BB
[..NEW..] Device CC:CC:CC:CC:CC:CC CC-CC-CC-CC-CC-CC
[..NEW..] Device DD:DD:DD:DD:DD:DD DD-DD-DD-DD-DD-DD
[..NEW..] Device EE:EE:EE:EE:EE:EE EE-EE-EE-EE-EE-EE
[bluetooth]# pair AA:AA:AA:AA:AA:AA ❸
[bluetooth]# exit ❹
[container ~]#
```

### 図 10.79 bluetoothctl コマンドによるスキャンとペアリングの例

- ❶ コントローラを起動します。
- ❷ 周辺機器をスキャンします。
- ❸ ペアリングしたい機器の MAC アドレスを指定してペアリングします。
- ❹ exit で bluetoothctl のプロンプトを終了します。

#### 10.2.5.2. Wi-SUN デバイスを扱う

ここでは、Wi-SUN デバイスが UART で接続されている場合の例を示します。この場合、コンテナ内で動作するアプリケーションから Wi-SUN デバイスで通信を行うためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/ttyxcN のうち、Wi-SUN と対応するものを渡す必要があります。以下は、/dev/ttyxc1 を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# vi /etc/atmark/containers/wisun_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/ttymxcl
[armadillo ~]# podman start wisun_example
Starting 'wisun_example'
ef9a5a2f7eee4236cb28c1fbf5090a6f0db9d6dfe7f3a34573867e0833dd3122
[armadillo ~]# podman exec -it wisun_example sh
[container ~]# ls /dev/ttymxcl
/dev/ttymxcl
```

図 10.80 Wi-SUN デバイスを扱うためのコンテナ作成例

コンテナ内から、`/dev/ttymxcl` を使って Wi-SUN データの送受信ができるようになります。

### 10.2.5.3. EnOcean デバイスを扱う

ここでは、EnOcean デバイスが UART で接続されている場合の例を示します。この場合、コンテナ内で動作するアプリケーションから EnOcean デバイスで通信を行うためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル `/dev/ttymxcl` のうち、EnOcean と対応するものを渡す必要があります。以下は、`/dev/ttymxcl` を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# vi /etc/atmark/containers/enocean_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/ttymxcl
[armadillo ~]# podman start enocean_example
Starting 'enocean_example'
a808b491a100f9078d8c72a7b36966d9182614f3657fe054fb8d7f87b0d4b31c
[armadillo ~]# podman exec -it enocean_example sh
[container ~]# ls /dev/ttymxcl
/dev/ttymxcl
```

図 10.81 EnOcean デバイスを扱うためのコンテナ作成例

コンテナ内から、`/dev/ttymxcl` を使って EnOcean データの送受信ができるようになります。

### 10.2.5.4. Thread デバイスを扱う

ここでは、Thread デバイスが UART で接続されている場合の例を示します。この場合、コンテナ内で動作するアプリケーションから Thread デバイスで通信を行うためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル `/dev/ttyACM0` や `/dev/ttymxcl` のうち、Thread と対応するものを渡す必要があります。以下は、`/dev/ttyACM0` を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# vi /etc/atmark/containers/thread_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/ttyACM0
[armadillo ~]# podman start thread_example
Starting 'thread_example'
a808b491a100f9078d8c72a7b36966d9182614f3657fe054fb8d7f87b0d4b31c
```

```
[armadillo ~]# podman exec -it thread_example sh
[container ~]# ls /dev/ttyACM0
/dev/ttyACM0
```

図 10.82 Thread デバイスを扱うためのコンテナ作成例

コンテナ内から、/dev/ttyACM0 を使って Thread データの送受信ができるようになります。

## 10.2.6. ネットワークを扱う

この章では、コンテナ内のネットワークを扱う方法について示します。

### 10.2.6.1. コンテナの IP アドレスを確認する

基本的にコンテナの IP アドレスは Podman イメージからコンテナを作成したときに自動的に割り振られます。コンテナに割り振られている IP アドレスはホスト OS 側からは podman inspect コマンドを用いて、以下のように確認することができます。

```
[armadillo ~]# vi /etc/atmark/containers/net_example.conf
set_image docker.io/alpine
set_command sleep infinity
[armadillo ~]# podman_start net_example
Starting 'net_example'
48ae479af65445674323567c17c5418dd4624292351e061bd2bd8a0add4cf150
[armadillo ~]# podman inspect --format '{{ .NetworkSettings.IPAddress }}' net_example
10.88.0.17
```

図 10.83 コンテナの IP アドレス確認例

コンテナ内の ip コマンドを用いて確認することもできます。

```
[armadillo ~]# podman exec net_example ip addr show eth0
2: eth0@if5: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP qlen 1000
    link/ether c6:13:67:83:d5:77 brd ff:ff:ff:ff:ff:ff
    inet 10.88.0.2/16 brd 10.88.255.255 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::c413:67ff:fe83:d577/64 scope link
        valid_lft forever preferred_lft forever
```

図 10.84 ip コマンドを用いたコンテナの IP アドレス確認例

### 10.2.6.2. コンテナに固定 IP アドレスを設定する



podman はデフォルトで 10.88.0.0/16 を使います。

他に使用している IP アドレスと被った場合等はコンテナに別の IP アドレスを設定してください。

コンテナに固定 IP アドレスを設定するためには、最初にユーザ定義のネットワークを作成する必要があります。以下に 192.0.2.0/24 にユーザ定義のネットワークを作成する例を示します。

```
[armadillo ~]# vi /etc/atmark/containers/my_network.conf
set_type network
set_subnet 192.0.2.0/24
[armadillo ~]# podman_start my_network
Creating network 'my_network'
my_network
```

図 10.85 ユーザ定義のネットワーク作成例

コンテナを作成する際に、上記で作成したネットワークと設定したい IP アドレスを渡すことで、コンテナの IP アドレスを固定することができます。以下の例では、IP アドレスを 192.0.2.1 に固定します。

```
[armadillo ~]# vi /etc/atmark/containers/network_example.conf
set_image docker.io/alpine
set_command sleep infinity
set_network my_network
set_ip 192.0.2.1
[armadillo ~]# podman_start network_example
Starting 'network_example'
3ea8c9031bf833228908bd73d8929b1d543b189b436c218e0634e0d39409e100
```

図 10.86 IP アドレス固定のコンテナ作成例

コンテナの IP アドレスが、192.0.2.1 に設定されていることが確認できます。

```
[armadillo ~]# podman inspect --format '{{ .NetworkSettings.Networks.my_network.IPAddress }}'
network_example
192.0.2.1
```

↩

図 10.87 コンテナの IP アドレス確認例

## 10.2.7. サーバを構築する

この章では、コンテナ内で様々なサーバを構築する方法について示します。この章で取り上げているサーバは alpine の apk コマンドでインストールすることが可能です。

### 10.2.7.1. HTTP サーバを構築する

ここでは、HTTP サーバとして Apache と lighttpd の 2 種類を使用する場合について説明します。

- ・ Apache を使用する

alpine イメージからコンテナを作成し、そのコンテナ内に Apache をインストールします。コンテナ作成の際に、ホスト OS の 8080 番ポートをコンテナ内の 80 番ポートに転送する指定を行っています。

```
[armadillo ~]# vi /etc/atmark/containers/apache_example.conf
set_image docker.io/alpine
set_command sleep infinity
```

```

add_ports 8080:80
[armadillo ~]# podman_start apache_example
Starting 'apache_example'
ea0a1ed9c2fe170a6db02e480300467510f4e844900efb35c7a24cc1a8653af2
[armadillo ~]# podman exec -it apache_example sh
[container ~]# apk upgrade && apk add apache2
[container ~]# httpd
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using
10.88.0.2. Set the 'ServerName' directive globally to suppress this message

```

↳

### 図 10.88 コンテナに Apache をインストールする例

他の PC などの Web ブラウザから、ホスト OS の IP アドレスの 8080 番ポートに接続すると、動作確認用ページが表示されます。デフォルトでは、/var/www/localhost/htdocs ディレクトリにファイルを置くことで Web ブラウザから閲覧できます。Apache の詳細な設定は、/etc/apache2 ディレクトリにある設定ファイルを編集することで変更可能です。

- ・ lighttpd を使用する

alpine イメージからコンテナを作成し、そのコンテナ内に lighttpd をインストールします。コンテナ作成の際に、ホスト OS の 8080 番ポートをコンテナ内の 80 番ポートに転送する指定を行っています。

```

[armadillo ~]# vi /etc/atmark/containers/lighttpd_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_ports 8080:80
[armadillo ~]# podman_start lighttpd_example
Starting 'lighttpd_example'
fd7ea338d09c5e8962654ed54bba17fb6a9ed4fca1b344e350bbf8f943d2f12b
[armadillo ~]# podman exec -it lighttpd_example sh
[container ~]# apk upgrade && apk add lighttpd
[container ~]# echo "<html><body>It works!</body></html>" > /var/www/localhost/htdocs/index.html
[container ~]# lighttpd -f /etc/lighttpd/lighttpd.conf

```

### 図 10.89 コンテナに lighttpd をインストールする例

lighttpd はデフォルトでは動作確認用ページが用意されていないため、上記の手順では簡単なページを /var/www/localhost/htdocs ディレクトリの下に配置しています。他の PC などの Web ブラウザから、ホスト OS の IP アドレスの 8080 番ポートに接続すると表示されます。lighttpd の詳細な設定は、/etc/lighttpd ディレクトリにある設定ファイルを編集することで変更可能です。

#### 10.2.7.2. FTP サーバを構築する

ここでは、FTP サーバとして vsftpd を使用する場合について説明します。alpine イメージからコンテナを作成し、そのコンテナ内に vsftpd をインストールします。コンテナ作成の際に、FTP 通信で使用するポートについてホスト OS 側からコンテナ内のポートに転送する指定と、コンテナ内の環境変数として PASV\_ADDRESS にホスト OS 側の IP アドレスの指定を行っています。

```

[armadillo ~]# vi /etc/atmark/containers/ftp_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_ports 21:21 21100-21110:21100-21110
add_args --env=PASV_ADDRESS=<ホストの IP アドレス>

```

```
[armadillo ~]# podman_start ftp_example
Starting 'ftp_example'
efcf1ba752c2db9ae1a33ac11af3be71d95ac7b737ce9734730ebca602e57796
[armadillo ~]# podman exec -it ftp_example sh
[container ~]# apk upgrade && apk add vsftpd
```

図 10.90 コンテナに vsftpd をインストールする例

コンテナ内にユーザアカウントを作成し、このユーザで ftp ログインできるようにします。

```
[container ~]# adduser atmark
Changing password for atmark
New password: (パスワードを入力)
Retype password: (パスワードを入力)
passwd: password for atmark changed by root
```

図 10.91 ユーザを追加する例

作成したユーザで ftp ログインできるように、vsftpd の設定ファイルを編集します。

```
[container ~]# sed -i -e 's/anonymous_enable=YES/#anonymous_enable=YES/g' /etc/vsftpd/vsftpd.conf
[container ~]# sed -i -e 's/#local_enable=YES/local_enable=YES/g' /etc/vsftpd/vsftpd.conf
[container ~]# sed -i -e 's/#write_enable=YES/write_enable=YES/g' /etc/vsftpd/vsftpd.conf
[container ~]# echo "pasv_enable=YES" >> /etc/vsftpd/vsftpd.conf
[container ~]# echo "pasv_min_port=21100" >> /etc/vsftpd/vsftpd.conf
[container ~]# echo "pasv_max_port=21110" >> /etc/vsftpd/vsftpd.conf
[container ~]# echo "pasv_address=$PASV_ADDRESS" >> /etc/vsftpd/vsftpd.conf
```

図 10.92 設定ファイルの編集例

編集した設定ファイルを指定して vsftpd を起動することにより、ftp 接続可能となります。ftp ログイン時のアカウントは前述の手順で作成したものを使用します。

```
[container ~]# vsftpd /etc/vsftpd/vsftpd.conf
```

図 10.93 vsftpd の起動例

### 10.2.7.3. Samba サーバを構築する

ここでは、Samba サーバの構築方法について説明します。alpine イメージからコンテナを作成し、そのコンテナ内に samba をインストールします。コンテナ作成の際に、samba で使用するポートについてホスト OS 側からコンテナ内のポートに転送する指定を行っています。

```
[armadillo ~]# vi /etc/atmark/containers/smb_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_ports 139:139 445:445
[armadillo ~]# podman_start smb_example
Starting 'smb_example'
6d81c01fe27b5a92ee6ea69de2f9a8dbb569d420c2f5f630ece1966c81824a1f
```



```
[armadillo ~]# podman exec -it smb_example sh
[container ~]# apk upgrade && apk add samba
```

### 図 10.94 コンテナに samba をインストールする例

コンテナ内にユーザアカウントを作成し、このユーザで samba にログインできるようにします。

```
[container ~]# adduser atmark
Changing password for atmark
New password: (パスワードを入力)
Retype password: (パスワードを入力)
passwd: password for atmark changed by root
[container ~]# pdbedit -a atmark
new password: (パスワードを入力)
retype new password: (パスワードを入力)
```

### 図 10.95 ユーザを追加する例

samba を起動すると、前述の手順で作成したユーザアカウントで他の PC などからログインすることができます。

```
[container ~]# smbd
```

### 図 10.96 samba の起動例

共有するディレクトリの指定などの詳細設定は /etc/samba/smb.conf ファイルを編集することで変更可能です。

## 10.2.7.4. SQL サーバを構築する

ここでは、RDMS として sqlite を使用する場合について説明します。alpine イメージからコンテナを作成し、そのコンテナ内に sqlite をインストールします。

```
[armadillo ~]# vi /etc/atmark/containers/sqlite_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_volumes /var/app/volumes/sqlite_db:/db
[armadillo ~]# podman_start sqlite_example
Starting 'sqlite_example'
114c5f1dbb7e81293dcb8fbe0c600b861626375b14cfe4023761acaa84fdcad1
[armadillo ~]# podman exec -it sqlite_example sh
[container ~]# apk upgrade && apk add sqlite
```

### 図 10.97 コンテナに sqlite をインストールする例

コンテナ内に入り、sqlite3 コマンドを実行すると sqlite のプロンプトが表示されデータベースの操作ができるようになります。

```
[container ~]# sqlite3 /db/mydb.sqlite
SQLite version 3.34.1 2021-01-20 14:10:07
```

```
Enter ".help" for usage hints.
sqlite>
```

図 10.98 sqlite の実行例

## 10.2.8. 画面表示を行う

この章では、コンテナ内で動作するアプリケーションから Armadillo-640 に接続されたディスプレイに出力を行う方法について示します。

### 10.2.8.1. X Window System を扱う

コンテナ内から、X Window System を起動し画面表示を行う例を示します。ここではアットマークテクノが提供するイメージからコンテナを作成します。このイメージに関しては「10.2.3. アットマークテクノが提供するイメージを使う」を参照してください。

```
[armadillo ~]# vi /etc/atmark/containers/x_example.conf
set_image at-debian-image-armv7
set_command sleep infinity
add_devices /dev/tty7 ❶
add_devices /dev/fb0 ❷
add_devices /dev/input ❸
add_volumes /run/udev:/run/udev:ro ❹
add_args --cap-add=SYS_ADMIN ❺
[armadillo ~]# podman start x_example
Starting 'x_example'
26847e21bd519f99466af32fdf0d809e2216d3e8ddf05c185e5428fe46e6a09b
```

図 10.99 X Window System を扱うためのコンテナ起動例

- ❶ X Window System に必要な tty を設定します。どこからも使われていない tty とします。
- ❷ 画面描画先となるフレームバッファを設定します。
- ❸ キーボードやマウスなどを使用可能にするためのデバイスを設定します。
- ❹ ホスト OS 側の /run/udev をコンテナ内からマウントするように設定します。
- ❺ X Window System の動作に必要な権限を設定します。

次に、以下のように X Window System を起動します。オプションである vt に設定する値は、コンテナ作成時に渡した tty の数字にします。

```
[armadillo ~]# podman exec -ti x_example bash
[container ~]# apt install xorg
[container ~]# X vt7 -retro &

X.Org X Server 1.20.11
X Protocol Version 11, Revision 0
Build Operating System: linux Debian
Current Operating System: Linux 2ae393cd5b2d 5.10.180-1-at #2-Alpine Wed Jun 7 06:53:04 UTC 2023
armv7l
Kernel command line: console=ttymx0,115200 root=/dev/mmcblk0p1 rootwait ro quiet
Build Date: 23 March 2023 10:25:56AM
```

↩

```
xorg-server 2:1.20.11-1+deb11u6 (https://www.debian.org/support)
Current version of pixman: 0.40.0
  Before reporting problems, check http://wiki.x.org
  to make sure that you have the latest version.
Markers: (--) probed, (**) from config file, (==) default setting,
  (++) from command line, (!!) notice, (II) informational,
  (WW) warning, (EE) error, (NI) not implemented, (??) unknown.
(==) Log file: "/var/log/Xorg.0.log", Time: Fri Jun  9 04:51:10 2023
(==) Using system config directory "/usr/share/X11/xorg.conf.d"
```

### 図 10.100 コンテナ内で X Window System を起動する実行例

Armadillo-640 に接続しているディスプレイ上に、デスクトップ画面が表示されます。

#### 10.2.8.2. フレームバッファに直接描画する

コンテナ内で動作するアプリケーションからフレームバッファに直接描画するためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/fbN を渡す必要があります。以下は、/dev/fb0 を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# vi /etc/atmark/containers/fb_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/fb0
[armadillo ~]# podman_start fb_example
Starting 'fb_example'
e8a874e922d047d5935350cd7411682dbbeb90fa828cef94af36acfb6d77476e
```

### 図 10.101 フレームバッファに直接描画するためのコンテナ作成例

コンテナ内に入って、ランダムデータをフレームバッファに描画する例を以下に示します。これにより、接続しているディスプレイ上の表示が変化します。

```
[armadillo ~]# podman exec -it fb_example sh
[container ~]# cat /dev/urandom > /dev/fb0
cat: write error: No space left on device
```

### 図 10.102 フレームバッファに直接描画する実行例

#### 10.2.8.3. タッチパネルを扱う

タッチパネルが組み込まれているディスプレイを接続している環境で、コンテナ内からタッチイベントを取得するためには、Podman のイメージからコンテナを作成する際にホスト OS 側の /dev/input を渡す必要があります。

```
[armadillo ~]# vi /etc/atmark/containers/touch_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/input
[armadillo ~]# podman_start touch_example
```

```
Starting 'touch_example'
cde71165076a413d198864899b64ff9c5fecdae222d9ee6646e189b5e976d94a
```

図 10.103 タッチパネルを扱うためのコンテナ作成例

X Window System などの GUI 環境と組み合わせて使うことで、タッチパネルを利用した GUI アプリケーションの操作が可能となります。

## 10.2.9. パワーマネジメント機能を使う

この章では、コンテナ内からパワーマネジメント機能を使う方法について示します。

### 10.2.9.1. サスペンド状態にする

パワーマネジメント機能を使ってサスペンド状態にするには、Podman のイメージからコンテナを作成する際にホスト OS 側の /sys ディレクトリを渡す必要があります。以下は、/sys を渡して alpine イメージからコンテナを作成する例です。ここで渡された /sys ディレクトリはコンテナ内の /sys にマウントされます。

```
[armadillo ~]# vi /etc/atmark/containers/pm_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_volumes /sys
[armadillo ~]# podman_start pm_example
Starting 'pm_example'
ab656f08a6cba2dc5919dbc32f8a6209782ba04baa0c6c21232a52046a21337e
```

図 10.104 パワーマネジメント機能を使うためのコンテナ作成例

コンテナ内から、/sys/power/state に次の文字列を書き込むことにより、サスペンド状態にすることができます。

表 10.2 対応するパワーマネジメント状態

パワーマネジメント状態	文字列	説明
Suspend-to-RAM	mem	最も消費電力を抑えることができる
Power-On Suspend	standby	Suspend-to-RAM よりも短時間で復帰することができ、Suspend-to-Idle よりも消費電力を抑えることができる
Suspend-to-Idle	freeze	最も短時間で復帰することができる



サスペンド状態を 128 秒以上継続する場合は、Suspend-to-RAM か +Power-On Suspend+を利用してください。

+Suspend-to-Idle+を利用している状態で 128 秒経過すると再起動してしまいます。

```
[armadillo ~]# podman exec -it pm_example sh
[container ~]# echo mem > /sys/power/state
```

図 10.105 サスペンド状態にする実行例

### 10.2.9.2. 起床要因を有効化する

サスペンド状態から起床要因として、利用可能なデバイスを以下に示します。

UART1 (CON9)      起床要因      データ受信

有効化

```
[container ~]# echo enabled > /sys/bus/platform/drivers/imx-uart/2020000.serial/tty/ttymxc0/power/wakeup
```



USB OTG1 (下段)      起床要因      USB デバイスの挿抜

有効化

```
[container ~]# echo enabled > /sys/bus/platform/devices/2184000.usb/power/wakeup
[container ~]# echo enabled > /sys/bus/platform/drivers/ci_hdrc/ci_hdrc.0/power/wakeup
[container ~]# echo enabled > /sys/bus/platform/drivers/ci_hdrc/ci_hdrc.0/usb1/power/wakeup
```



USB OTG2 (上段)      起床要因      USB デバイスの挿抜

有効化

```
[container ~]# echo enabled > /sys/bus/platform/devices/2184200.usb/power/wakeup
[container ~]# echo enabled > /sys/bus/platform/drivers/ci_hdrc/ci_hdrc.1/power/wakeup
[container ~]# echo enabled > /sys/bus/platform/drivers/ci_hdrc/ci_hdrc.1/usb2/power/wakeup
```



RTC(i.MX6ULL)      起床要因      アラーム割り込み

有効化

```
[container ~]# echo enabled > /sys/bus/platform/devices/20cc000.snvs%:snvs-rtc-lp/power/wakeup
```



実行例

```
[armadillo ~]# vi /etc/atmark/containers/rtc_pm_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_volumes /sys
add_devices /dev/rtc0
[armadillo ~]# podman start rtc_pm_example
Starting 'rtc_pm_example'
```

```
8fbef3edda3b7fcea5b1f8cbf960cf469b7e82c4d1ecd35477076e81fc24e3
9f
[armadillo ~]# podman exec -ti rtc_pm_example sh
[container ~]# apk add util-linux
[container ~]# rtcwake -m mem -s 5
: (省略)
[ 572.720300] printk: Suspending console(s) (use
no_console_suspend to debug)
<ここで5秒を待つ>
[ 573.010663] OOM killer enabled.
...
```

図 10.106 サスペンド状態にする実行例、rtc で起こす

## 10.2.10. コンテナからの poweroff か reboot

Armadillo Base OS は busybox init で shutdown と reboot を対応します。

busybox init で PID 1 に signal を送ることで shutdown や reboot となります。コンテナから signal を送るように、pid namespace を共有する必要がありますが、共有されたら kill で実行できます。

```
[armadillo ~]# vi /etc/atmark/containers/shutdown_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_args --pid=host
[armadillo ~]# podman_start shutdown_example
Starting 'shutdown_example'
c8e3b9b418fc72395db9f3c22b1eb69eb41eaaf790d3b7151047ef066cc4c8ff
[armadillo ~]# podman exec -ti shutdown_example sh
[container ~]# kill -USR2 1 (poweroff)
[container ~]# kill -TERM 1 (reboot)
```

図 10.107 コンテナから shutdown を行う

## 10.2.11. 異常検知

この章では、コンテナ内で動作しているアプリケーションに何らかの異常が発生し停止してしまった際に、ソフトウェアウォッチドッグタイマーを使って、システムを再起動する方法について示します。

### 10.2.11.1. ソフトウェアウォッチドッグタイマーを扱う

コンテナ内で動作するアプリケーションからソフトウェアウォッチドッグタイマーを扱うためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/watchdogN を渡す必要があります。以下は、/dev/watchdog0 を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# vi /etc/atmark/containers/watchdog_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/watchdog0
[armadillo ~]# podman_start watchdog_example
```

```
Starting 'watchdog_example'  
a5d329cca49d60423ce4155d72a119b8049a03dbd1d0277817a253e96dce7bc7
```

### 図 10.108 ソフトウェアウォッチドッグタイマーを使うためのコンテナ作成例

ソフトウェアウォッチドッグタイマーは、プログラム内からデバイスファイル `/dev/watchdog0` を `open` した時点で起動します。コンテナ内に入ってソフトウェアウォッチドッグタイマーを `echo` コマンドで起動する例を以下に示します。

```
[armadillo ~]# podman exec -it watchdog_example sh  
[container ~]# echo > /dev/watchdog0
```

### 図 10.109 コンテナ内からソフトウェアウォッチドッグタイマーを起動する実行例

ソフトウェアウォッチドッグタイマーを起動した後、`/dev/watchdog0` に任意の文字を書き込むことでソフトウェアウォッチドッグタイマーをリセットすることができます。10 秒間任意の文字の書き込みがない場合は、システムが再起動します。

```
[armadillo ~]# podman exec -it watchdog_example sh  
[container ~]# echo a > /dev/watchdog0
```

### 図 10.110 ソフトウェアウォッチドッグタイマーをリセットする実行例

ソフトウェアウォッチドッグタイマーを停止したい場合は、`/dev/watchdog0` に `V` を書き込みます。

```
[armadillo ~]# podman exec -it watchdog_example sh  
[container ~]# echo V > /dev/watchdog0
```

### 図 10.111 ソフトウェアウォッチドッグタイマーを停止する実行例

## 10.3. コンテナの運用

### 10.3.1. コンテナの自動起動

Armadillo Base OS では、`/etc/atmark/containers/*.conf` ファイルに指定されているコンテナがブート時に自動的に起動します。`nginx.conf` の記載例を以下に示します。

```
[armadillo ~]# cat /etc/atmark/containers/nginx.conf  
set_image docker.io/library/nginx:alpine  
set_readonly no  
add_ports 80:80
```

### 図 10.112 コンテナを自動起動するための設定例

`.conf` ファイルは以下のパラメータを設定できます。

- ・ コンテナイメージの選択：`set_image` [イメージ名]

イメージの名前を設定できます。

例: `set_image docker.io/debian:latest, set_image localhost/myimage`

イメージを rootfs として扱う場合に `--rootfs` オプションで指定できます。

例: `set_image --rootfs /var/app/volumes/debian`

・ ポート転送 : **add\_ports** [ホストポート]:[コンテナポート]

設定したポートで外部からコンテナへのアクセスが可能となります。

デフォルトは TCP で、UDP も `/udp` を付けて使えます。スペースで分けて複数のポートを設定することができます。

以下の例では、ポート 80、443(web)、UDP の 69(tftp)にアクセスすることができ、コンテナのポート 22(ssh)にはポート 2222 からアクセスすることができます。

例: `add_ports 80:80 443:443 2222:22 69:69/udp`



pod を使う場合、このオプションは pod の設定にしないと有効になりませんのでご注意ください。

・ デバイスファイル作成 : **add\_devices** [ホストパス]:[コンテナパス]

コンテナでデバイスを作成して、使用可能となります。

コンテナパスを設定しない場合はホストと同じパスを使います。

複数のデバイスを作成したい場合はスペースで分けて設定してください。

例: `add_devices /dev/ttyxc2 /dev/v4l/by-id/usb-046d_HD_Pro_Webcam_C920_78DA8CAF-video-index0:/dev/video3`

ホストパスに「:」を含む場合は `add_device "[ホストパス]" "[コンテナパス]"` で追加できます。

例: `add_device "/dev/v4l/by-path/platform-xhci-hcd.1.auto-usb-0:1.1:1.0-video-index1" "/dev/video3"`

コンテナパスに「:」を含むようなパスは設定できません。

・ ボリュームマウント : **add\_volumes** [ホストパス]:[コンテナパス]:[オプション]

指定するパスをコンテナ内でマウントして、データの保存や共有することができます。

ホストパスは以下のどちらかを指定してください。

- ・ `/var/app/rollback/volumes/<folder>` か `<folder>`:

アップデートの際に新しくコピー (snapshot) した場合、コピー先のみ変更しますので、アップデート中でもこのデータを使うことができます。途中で電源が落ちた場合でも、このデータに影響はありません。

SWUpdate でアップデートするデータに向いています。

- ・ `/var/app/volumes/<folder>`: app パーティションに書きます。



アップデートの際にコピーされませんので、アップデート中の新たな変更は更新されたコンテナ内のアプリケーションで見れます。

ログやデータベースに向いています。

- ・ /tmp/<folder>: 複数のコンテナでメモリファイルシステムを共有したい場合に使ってください。
- ・ /opt/firmware: 学習能力に必要なファームウェアライブラリーのパス。

コンテナパスを設定しない場合はホストパスと同じパスを使います。

オプションは podman run の --volume のオプションになりますので、ro (read-only), nodev, nosuid, noexec, shared, slave 等を設定できます。

**例** : add\_volumes /var/app/volumes/database:/database: ロールバックされないデータを /database で保存します。

**例**: add\_volumes assets:/assets:ro,nodev,nosuid /opt/firmware: アプリケーションのデータを /assets で読み取り、/opt/firmware のファームウェアを使えます。

「:」はホスト側のパスとコンテナのパスを別ける意味があるため、ファイル名やデバイス名に「:」を使うことはできません。



複数のコンテナでマウントコマンドを実行することがあれば、shared のフラグで起動後のマウントを共有することができます。

```
[armadillo ~]# cat /etc/atmark/containers/mounter.conf
set_image docker.io/alpine
add_args -ti
add_volumes /tmp/mnt:/mnt:shared ❶
add_args --cap-add SYS_ADMIN
add_device /dev/sda1
[armadillo ~]# cat /etc/atmark/containers/client.conf
set_image docker.io/alpine
add_volumes /tmp/mnt:/mnt:slave ❷
add_args -ti
[armadillo ~]# podman exec mounter mount /dev/sda1 /mnt ❸
[armadillo ~]# podman exec client ls /mnt ❹
file_on_usb
```

#### 図 10.113 ボリュームを shared でサブマウントを共有する例

- ❶ マウントを行うコンテナに shared の設定とマウント権限 (SYS\_ADMIN) を与えます。
- ❷ マウントを使うコンテナに slave だけを設定すれば一方にしか共有されません。
- ❸ USB デバイスをマウントします。
- ❹ マウントされたことを確認します。

- ・ ホットプラグデバイスの追加 : **add\_hotplugs [デバイスタイプ]**

コンテナ起動後に挿抜を行なっても認識される(ホットプラグ)デバイスを設定できます。

通常、コンテナ内からデバイスを扱うためには、あらかじめ Armadillo 本体に当該のデバイスを接続した状態で、コンテナを起動する必要がありますが、 `add_hotplugs` を使用することでホットプラグに対応できます。

例: `add_hotplugs input`

`add_hotplugs` に指定できる主要な文字列とデバイスファイルの対応について、「表 10.3. `add_hotplugs` オプションに指定できる主要な文字列」に示します。

表 10.3 `add_hotplugs` オプションに指定できる主要な文字列

文字列	引数の説明	対象のデバイスファイル
<code>input</code>	マウスやキーボードなどの入力デバイス	<code>/dev/input/mouse0</code> , <code>/dev/input/event0</code> など
<code>video4linux</code>	USB カメラなどの <code>video4linux</code> デバイスファイル	<code>/dev/video0</code> など
<code>sd</code>	USB メモリなどの SCSI ディスクデバイスファイル	<code>/dev/sda1</code> など

「表 10.3. `add_hotplugs` オプションに指定できる主要な文字列」に示した文字列の他にも、`/proc/devices` の数字から始まる行に記載されている文字列を指定することができます。「図 10.114. `/proc/devices` の内容例」に示す状態の場合、デバイスタイプを示す文字列としては、各行の先頭の数字を除いた `mem` や `tty` などを指定できることがわかります。

```
[armadillo ~]# cat /proc/devices
1 mem
4 /dev/vc/0
4 tty
5 /dev/tty
5 /dev/console
5 /dev/ptmx
7 vcs
10 misc
13 input
29 fb
89 i2c
108 ppp
116 alsa
: (省略)
```

図 10.114 `/proc/devices` の内容例

デバイスタイプと実際のデバイスファイルの対応については、カーネルドキュメント: `devices.txt`(Github) [<https://github.com/torvalds/linux/blob/master/Documentation/admin-guide/devices.txt>] を参照してください。

複数のデバイスタイプを指定したい場合はスペースで分けて設定してください。

例: `add_hotplugs input usb sd`

- ・ pod の選択: `set_pod [ポッド名]`

「10.3.2. pod の作成」で作成した pod の名前を入れてコンテナを pod 内で起動します。

例: `set_pod mypod`

- ・ ネットワークの選択 : **set\_network** [ネットワーク名]

この設定に「10.3.3. network の作成」で作成したネットワーク以外に `none` と `host` の特殊な設定も選べます。

`none` の場合、コンテナに `localhost` しかない名前空間に入ります。

`host` の場合は OS の名前空間をそのまま使います。

例: `set_network mynetwork`

- ・ IP アドレスの設定 : **set\_ip** [アドレス]

コンテナの IP アドレスを設定することができます。

例: `set_ip 10.88.0.100`



コンテナ間の接続が目的であれば、`pod` を使って `localhost` か `pod` の名前前でアクセスすることができます。

- ・ 読み取り専用設定 : **set\_readonly yes**

コンテナ内からのファイルシステムへの書き込み許可を設定します。

デフォルトで書き込み可能となっています。

コンテナ内からのファイルシステムへの書き込みを禁止することで、`tmpfs` として使うメモリの消費を明示的に抑えることができますが、アプリケーションによっては読み込み専用のファイルシステムでは動作しない可能性もあります。

- ・ イメージの自動ダウンロード設定 : **set\_pull** [設定]

この設定を `missing` にすると、イメージが見つからない場合にイメージを自動的にダウンロードします。

`always` にすると、イメージがすでにダウンロード済みでも起動前に必ず更新の確認を取ります。

デフォルトでは `never` で、イメージが見つからない場合にエラーを表示します。

例 : `set_pull missing` か `set_pull always`

- ・ コンテナのリスタート設定 : **set\_restart** [設定]

コンテナが停止した時にリスタートさせます。

`podman kill` か `podman stop` で停止する場合、この設定と関係なくリスタートしません。

デフォルトで `on-failure` になっています。

例: `set_restart always` か `set_restart no`

- ・ 信号を受信するサービスの無効化: **set\_init no**

コンテナのメインプロセスが PID 1 で起動していますが、その場合のデフォルトの信号の扱いが変わります: SIGTERM などのデフォルトハンドラが無効です。

そのため、init 以外のコマンドを `set_command` で設定する場合は `podman-init` のプロセスを PID 1 として立ち上げて、設定したコマンドをその子プロセスとして起動します。

例: `set_init no`

- ・ 自動起動の無効化: **set\_autostart no**

手動かまたは別の手段で操作するコンテナがある場合、Armadillo の起動時に自動起動しないようにします。

その場合、`podman_start <name>` で起動させることができます。



コンフィグに記載していないイメージはアップデートの際に削除されますので、そういったイメージに対して設定してください。

- ・ 実行コマンドの設定: **set\_command [コマンド]**

コンテナを起動するときのコマンド。設定されなかった場合、コンテナイメージのデフォルトを使います。

例: `set_command /bin/sh -c "echo bad example"`

- ・ `podman run` に引数を渡す設定: **add\_args [引数]**

ここまでで説明した設定項目以外の設定を行いたい場合は、この設定で `podman run` に直接引数を渡すことができます。

例: `add_args --cap-add=SYS_TTY_CONFIG --env=XDG_RUNTIME_DIR=/run/xdg_home`

## 10.3.2. pod の作成

`podman_start` で pod 機能を使うことができます。

pod を使うことで、複数のコンテナが同じネットワークネームスペースを共有することができます。同じ pod の中のコンテナが IP の場合 `localhost` で、unix socket の場合 `abstract path` で相互に接続することができます。

```
[armadillo ~]# cat /etc/atmark/containers/mypod.conf
set_type pod
add_ports 80:80

[armadillo ~]# cat /etc/atmark/containers/nginx.conf
set_image docker.io/library/nginx:alpine
set_readonly no
set_pod mypod
```

```
[armadillo ~]# podman ps
CONTAINER ID  IMAGE                                COMMAND                                CREATED      STATUS
PORTS        NAMES
0cdb0597b610 localhost/podman-pause:4.3.1-1683096588  2 hours ago Up 2 hours ago
0.0.0.0:80->80/tcp  5ba7d996f673-infra
3292e5e714a2 docker.io/library/nginx:alpine  nginx -g daemon o...  2 hours ago Up 2 hours ago
0.0.0.0:80->80/tcp  nginx
```

図 10.115 pod を使うコンテナを自動起動するための設定例

コンテナと同じく、`/etc/atmark/containers/[NAME].conf` ファイルを作って、`set_type pod` を設定することで pod を作成します。

pod を使う時にコンテナの設定ファイルに `set_pod [NAME]` の設定を追加します。

ネットワークネームスペースは pod を作成するときに必要なため、`ports`、`network` と `ip` の設定は pod のコンフィグファイルに入れなければなりません。

必要であれば、他の `podman pod create` のオプションを `add_args` で設定することができます。

### 10.3.3. network の作成

`podman_start` で `podman` の `network` も作成ことができます。

デフォルトの `10.88.0.0/16` が使えない場合、あるいはコンテナ同士で接続できないようにしたい場合は使ってください。

```
[armadillo ~]# cat /etc/atmark/containers/mynetwork.conf
set_type network
set_subnet 192.168.100.0/24

[armadillo ~]# cat /etc/atmark/containers/nginx.conf
set_image docker.io/library/nginx:alpine
add_ports 80:80
set_ip 192.168.100.10
set_network mynetwork

[armadillo ~]# podman ps
CONTAINER ID  IMAGE                                COMMAND                                CREATED      STATUS
PORTS        NAMES
3292e5e714a2 docker.io/library/nginx:alpine  nginx -g daemon o...  2 hours ago Up 2 hours ago
0.0.0.0:80->80/tcp  nginx
```

図 10.116 network を使うコンテナを自動起動するための設定例

コンテナと同じく、`/etc/atmark/containers/[NAME].conf` ファイルを作って、`set_type network` を設定することで `network` を作成します。

そのネットワークを使う時にコンテナの設定ファイルに `set_network [NAME]` の設定をいれます。

ネットワークのサブネットは `set_subnet [SUBNET]` で設定します。この設定は `set_type network` の後しか使えませんので、`set_type` はファイルの最初のところに使ってください

他の `podman network create` のオプションが必要であれば、`add_args` で設定することができます。

### 10.3.4. コンテナからのコンテナ管理

podman では REST API による管理アクセスも可能です。

自分のコンテナから他のコンテナの管理が必要な場合に、ホストの podman サービスを有効にして、コンテナに /run/podman をボリュームマウントすれば podman --remote で管理できます。

podman\_start をインストールすればそちらも --remote で使えます。

このオプションは Armadillo のホスト側の udev rules からコンテナを扱う時にも必要です。

### 10.3.5. コンテナの配布



コンテナの作成は「10.2. アプリケーションをコンテナで実行する」を参考にしてください。

コンテナのイメージを配布する方法は大きく分けて二つあります：

1. インターネット上のリポジトリ (dockerhub 等) で登録してそこから配布する
2. SWUpdate のアップデートイメージを配布する



Podman のイメージをインストールする時に、一時データを大量に保存する必要があります。

swu イメージ内で組み込む時は 3 倍、pull や USB ドライブで分けてインストールすると転送するデータ量の 2 倍の空き容量が app パーティションに必要です。

アップデート時にアップデート前のコンテナが使われているのでご注意ください。

#### 10.3.5.1. リモートリポジトリにコンテナを送信する方法

1. イメージをリモートリポジトリに送信する：

```
[armadillo ~]$ podman image push <localimage> docker://<registry>/<remoteimage>:<tag>
```

2. set\_pull always を設定しないかぎり、SWUpdate でダウンロードの命令を送らないとアップデートを行いません。

(mkswu については「10.7. Armadillo のソフトウェアをアップデートする」を参考にしてください)

```
[ATDE ~/mkswu]$ cp /usr/share/mkswu/examples/pull_container_nginx.desc .  
[ATDE ~/mkswu]$ cp -r /usr/share/mkswu/examples/nginx_start .
```

```
[ATDE ~/mkswu]$ cat pull_container_nginx.desc
swdesc_option version=1

swdesc_pull_container "docker.io/nginx:alpine"
swdesc_files --extra-os nginx_start
[ATDE ~/mkswu]$ mkswu pull_container_nginx.desc
Enter pass phrase for /home/atmark/mkswu/swupdate.key:
pull_container_nginx.swu を作成しました。
```

### 10.3.5.2. イメージを eMMC に保存する方法

Armadillo Base OS のデフォルトでは、Podman のデータは tmpfs に保存されます。

起動時にコンテナを起動するにはイメージを eMMC に書き込む必要があります。開発が終わって運用の場合は「10.3.5.3. イメージを SWUpdate で転送する方法」でコンテナのイメージを転送します。この場合は読み取り専用の app パーティションのサブボリュームに展開します。

開発の時に以下の `abos-ctrl podman-rw` か `abos-ctrl podman-storage --disk` のコマンドを使って直接にイメージを編集することができます。



ここで紹介する内容はコンテナのイメージの管理の説明です。データベース等のコンテナから書き込みが必要な場合には「10.2.2.6. コンテナの変更を保存する」にあるボリュームの説明を参照してください。

- ・ `abos-ctrl podman-rw`

`abos-ctrl podman-rw` を使えば、`read-only` になっているイメージを扱う事ができます。

```
[armadillo ~]# podman images
REPOSITORY          TAG          IMAGE ID      CREATED      SIZE      R/O
[armadillo ~]# mount /dev/sda1 /mnt
[armadillo ~]# abos-ctrl podman-rw load -i /mnt/at-debian-image.tar
Getting image source signatures
Copying blob 63c098a71e7b done
Copying blob 837e73dd4d20 done
Copying blob a25086e65f63 done
Copying config b5a30f8581 done
Writing manifest to image destination
Storing signatures
Loaded image(s): localhost/at-debian-image:latest
[armadillo ~]# podman image list
REPOSITORY          TAG          IMAGE ID      CREATED      SIZE      R/O
localhost/at-debian-image  latest      b5a30f8581cc  2 hours ago  233 MB    true
```

### 図 10.117 abos-ctrl podman-rw の実行例

- ・ `abos-ctrl podman-storage`

`abos-ctrl podman-storage` はメモリとディスクの切り替えの他に、読み書きストレージから読み取り専用ストレージへのコピーもできます。

```
[armadillo ~]# podman pull docker.io/alpine ❶
Trying to pull docker.io/library/alpine:latest...
Getting image source signatures
Copying blob f97344484467 done
Copying config 3d81c46cd8 done
Writing manifest to image destination
Storing signatures
3d81c46cd8756ddb6db9ec36fa06a6fb71c287fb265232ba516739dc67a5f07d
[armadillo ~]# abos-ctrl podman-storage ❷
List of images configured on development storage:
REPOSITORY          TAG          IMAGE ID      CREATED      SIZE
docker.io/library/alpine latest      3d81c46cd875 3 days ago  5.56 MB

What should we do? ([C]opy (default), [N]othing, [D]elete)
copy ❸
Create a snapshot of '/mnt/boot_1/containers_storage' in '/mnt/new_storage'
Getting image source signatures
Copying blob 8ec3165d6e61 done
Copying config 4a49b68e7c done
Writing manifest to image destination
Storing signatures
Delete subvolume (no-commit): '/mnt/new_storage'
Merging development images to readonly storage succeeded
Feel free to adjust the result with abos-ctrl podman-rw commands

Now freeing up original data...
Podman is in tmpfs mode ❹
[armadillo ~]# podman image list ❺
REPOSITORY          TAG          IMAGE ID      CREATED      SIZE      R/O
docker.io/library/alpine latest      3d81c46cd875 3 days ago  5.56 MB    true
```

図 10.118 abos-ctrl podman-storage のイメージコピー例

- ❶ イメージを書き込み可能ストレージに取得します。
- ❷ abos-ctrl podman-storage をオプション無しで実行します。
- ❸ 書き込み可能ストレージにイメージがある場合に対応を聞かれます。今回はコピー (copy) します。
- ❹ abos-ctrl podman-storage にオプションを指定しなかったため、ストレージが tmpfs のままになります。すでに --disk で切り替えた場合にディスクのままでも可能です。
- ❺ コピーの確認します。イメージが読み取り専用 (R/O, Read only) になりました。



podman が壊れやすいので、デフォルトの「abos-ctrl podman-storage --tmpfs」で運用することを推奨しますが、tmpfs の容量が小さくてイメージの操作には向いてません。

開発時には「abos-ctrl podman-storage --disk」の状態で行い、運用時には「abos-ctrl podman-storage --tmpfs」に戻してください。戻る際に「copy」を選択肢する場合は一時的なストレージをそのまま使いつづけますので、すべての変更が残ります。





SWUpdate でアップデートをインストールする際には、`/var/lib/containers/storage_readonly` ディレクトリの不要になったイメージを自動的に削除します。

自動起動させる予定がなくても、「10.3.1. コンテナの自動起動」を参考に、`/etc/atmark/containers/*.conf` を使ってください。`set_autostart no` を設定することで自動実行されません。

### 10.3.5.3. イメージを SWUpdate で転送する方法

1. イメージをファイルに保存する：

```
[armadillo ~]$ podman image save -o <myimage>.tar <localimage>
```

2. ファイルを SWUpdate のイメージに入れる。

二つのやり方があります：

- a. swu イメージ内に組み込む

```
[ATDE ~/mkswu]$ cp /usr/share/mkswu/examples/embed_container_nginx.desc .
[ATDE ~/mkswu]$ cp -r /usr/share/mkswu/examples/nginx_start .
[ATDE ~/mkswu]$ cat embed_container_nginx.desc
swdesc_option version=1

swdesc_embed_container "nginx_alpine.tar"
swdesc_files --extra-os nginx_start
[ATDE ~/mkswu]$ podman pull --arch arm --variant v7 docker.io/nginx:alpine
[ATDE ~/mkswu]$ podman run --rm docker.io/nginx:alpine uname -m
armv7l
[ATDE ~/mkswu]$ podman save docker.io/nginx:alpine > nginx_alpine.tar
[ATDE ~/mkswu]$ mkswu embed_container_nginx.desc
Enter pass phrase for /home/atmark/mkswu/swupdate.key:
embed_container_nginx.swu を作成しました
```

- b. USB ドライブに保存する

```
[ATDE ~/mkswu]$ cp /usr/share/mkswu/examples/usb_container_nginx.desc .
[ATDE ~/mkswu]$ cp -r /usr/share/mkswu/examples/nginx_start .
[ATDE ~/mkswu]$ cat usb_container_nginx.desc
swdesc_option version=1

swdesc_usb_container "nginx_alpine.tar"
swdesc_files --extra-os nginx_start
[ATDE ~/mkswu]$ podman pull --arch arm --variant v7 docker.io/nginx:alpine
[ATDE ~/mkswu]$ podman run --rm docker.io/nginx:alpine uname -m
armv7l
[ATDE ~/mkswu]$ podman save docker.io/nginx:alpine > nginx_alpine.tar
[ATDE ~/mkswu]$ mkswu -o usb_container_nginx.swu usb_container_nginx.desc
Enter pass phrase for /home/atmark/mkswu/swupdate.key:
```

```
以下のファイルを USB メモリにコピーしてください :
'/home/atmark/mkswu/usb_container_nginx.swu'
'/home/atmark/mkswu/nginx_alpine.tar'
'/home/atmark/mkswu/.usb_container_nginx/nginx_alpine.tar.sig'

usb_container_nginx.swu を作成しました。
```

## 10.4. Armadillo のソフトウェアをビルドする

ここでは、Armadillo-640 で使用するソフトウェアのビルド方法を説明します。

### 10.4.1. ブートローダーをビルドする

ここでは、Armadillo-640 向けのブートローダーイメージをビルドする方法を説明します。

#### 1. ソースコードの取得

Armadillo Base OS 対応 Armadillo-640 ブートローダー [<https://armadillo.atmark-techno.com/resources/software/armadillo-640/abos-boot-loader>] から「ブートローダーソース」ファイル (uboot-[VERSION].tar.gz) を次のようにダウンロードします。

```
[ATDE ~]$ https://armadillo.atmark-techno.com/files/downloads/armadillo-640/source/u-boot-
[VERSION].tar.gz
[ATDE ~]$ tar xf uboot-[VERSION].tar.gz
[ATDE ~]$ cd uboot-[VERSION]
```

↩

#### 2. デフォルトコンフィギュレーションの適用

「図 10.119. デフォルトコンフィギュレーションの適用」に示すコマンドを実行します。

```
[ATDE ~/u-boot-[VERSION]]$ make ARCH=arm armadillo-640_defconfig
HOSTCC scripts/basic/fixdep
HOSTCC scripts/kconfig/conf.o
YACC scripts/kconfig/zconf.tab.c
LEX scripts/kconfig/zconf.lex.c
HOSTCC scripts/kconfig/zconf.tab.o
HOSTLD scripts/kconfig/conf
#
# configuration written to .config
#
```

図 10.119 デフォルトコンフィギュレーションの適用

#### 3. ビルド

次のコマンドを実行します。

```
[ATDE ~/u-boot-[VERSION]]$ make CROSS_COMPILE=arm-linux-gnueabihf-
:
: (省略)
:
LD u-boot
```

```
OBJCOPY u-boot-nodtb.bin
CAT      u-boot-dtb.bin
MKIMAGE u-boot-dtb.imx
OBJCOPY u-boot.srec
COPY     u-boot.bin
SYM      u-boot.sym
CFGCHK   u-boot.cfg
```

#### 4. インストール

ビルドしたブートローダーは、以下に示すどちらかの方法でインストールしてください。

- ・ swupdate でインストールする

mkswu の初期化を行った後に 提供されているスクリプトを使って SWU イメージを作成してください。

```
[ATDE ~/u-boot-[VERSION]]$ echo 'swdesc_boot u-boot-dtb.imx' > boot.desc
[ATDE ~/u-boot-[VERSION]]$ mkswu boot.desc
boot.swu を作成しました。
```

作成された boot.swu のインストールについては 「10.7.3. イメージのインストール」 を参照ください。

- ・ 「10.5.1. ブートディスクの作成」 でインストールする

手順を参考にして、ビルドされた u-boot-dtb.imx を使ってください。

### 10.4.2. Linux カーネルをビルドする

ここでは、Armadillo-640 向けの Linux カーネルイメージをビルドする方法を説明します。



Armadillo-640 では、基本的には Linux カーネルイメージをビルドする必要はありません。「10.4.3. Alpine Linux ルートファイルシステムをビルドする」の手順を実施することで、標準の Linux カーネルイメージがルートファイルシステムに組み込まれます。

標準の Linux カーネルイメージは、アットマークテクノが提供する linux-at という Alpine Linux 用のパッケージに含まれています。

カスタマイズした Linux カーネルイメージを利用する場合は、以下に示す手順を参照してください。

#### 1. ソースコードの取得

Armadillo Base OS 対応 Armadillo-640 Linux カーネル [<https://armadillo.atmark-techno.com/resources/software/armadillo-640/abos-linux-kernel>] から 「Linux カーネル」ファイル (linux-at-a6-[VERSION].tar) をダウンロードして、次のコマンドを実行します。

```
[ATDE ~]$ tar xf linux-at-a6-[VERSION].tar
[ATDE ~]$ tar xf linux-at-a6-[VERSION]/linux-[VERSION].tar.gz
[ATDE ~]$ cd linux-[VERSION]
```

## 2. デフォルトコンフィギュレーションの適用

次のコマンドを実行します。

```
[ATDE ~/linux-[VERSION]]$ make ARCH=arm armadillo-640_defconfig
```

## 3. カーネルコンフィギュレーションの変更

次のコマンドを実行します。カーネルコンフィギュレーションの変更を行わない場合はこの手順は不要です。

```
[ATDE ~]$ make ARCH=arm menuconfig
```

コマンドを実行するとカーネルコンフィギュレーション設定画面が表示されます。カーネルコンフィギュレーションを変更後、"Exit"を選択して「Do you wish to save your new kernel configuration? (Press <ESC><ESC> to continue kernel configuration.)」で"Yes"とし、カーネルコンフィギュレーションを確定します。

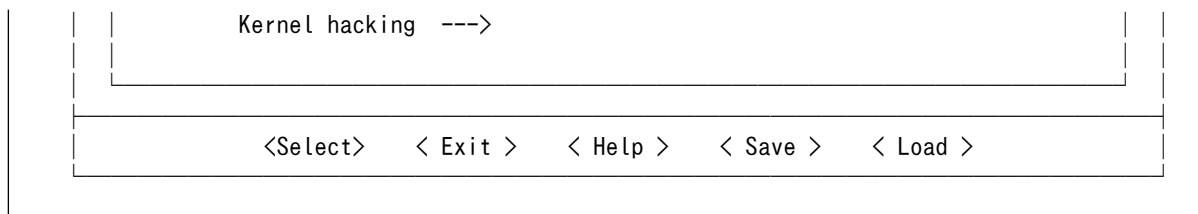
```
.config - Linux/arm 5.10.145 Kernel Configuration
```


```

Linux/arm 5.10.145 Kernel Configuration
-----
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus
----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M>
modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search.
Legend: [*] built-in [ ] excluded <M> module < > module capable

  General setup --->
  System Type --->
  Bus support --->
  Kernel Features --->
  Boot options --->
  CPU Power Management --->
  Floating point emulation --->
  Power management options --->
  Firmware Drivers --->
  [ ] ARM Accelerated Cryptographic Algorithms ----
  General architecture-dependent options --->
  [*] Enable loadable module support --->
  [*] Enable the block layer --->
  IO Schedulers --->
  Executable file formats --->
  Memory Management options --->
  [*] Networking support --->
  Device Drivers --->
  File systems --->
  Security options --->
  -* Cryptographic API --->
  Library routines --->

```





Linux Kernel Configuration メニューで"/"キーを押下すると、カーネルコンフィギュレーションの検索を行うことができます。カーネルコンフィギュレーションのシンボル名(の一部)を入力して"Ok"を選択すると、部分一致するシンボル名を持つカーネルコンフィギュレーションの情報が一覧されます。

#### 4. ビルド

次のコマンドを実行します。

```
[ATDE ~/linux-[VERSION]]$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
[ATDE ~/linux-[VERSION]]$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
LOADADDR=0x82000000 uImage
```

#### 5. インストール

ビルドしたカーネルは、以下に示すどちらかの方法でインストールしてください。

- ・ swupdate でインストールする

mkswu の初期化を行った後に 提供されているスクリプトを使って SWU イメージを作成してください。

```
[ATDE ~/linux-[VERSION]]$ /usr/share/mkswu/examples/kernel_update_plain.install.sh ~/
mkswu/kernel.desc
Installing kernel in /home/atmark/mkswu/kernel ...
'arch/arm/boot/uImage' -> '/home/atmark/mkswu/kernel/uImage'
'arch/arm/boot/dts/armadillo-640-at-dtweb.dtb' -> '/home/atmark/mkswu/kernel/
armadillo-610-at-dtweb.dtb'
: (省略)
INSTALL arch/arm/crypto/chacha-neon.ko
INSTALL arch/arm/crypto/curve25519-neon.ko
: (省略)
DEPMOD [VERSION]
Updated /home/atmark/mkswu/kernel.desc version from [PREV_VERSION] to [VERSION]
Done installing kernel, run `mkswu "/home/atmark/mkswu/kernel.desc"` next.
[ATDE ~/linux-[VERSION]]$ mkswu ~/mkswu/kernel.desc
/home/atmark/mkswu/kernel.swu を作成しました
```

図 10.120 Linux カーネルを SWU でインストールする方法

作成された kernel.swu のインストールについては 「10.7.3. イメージのインストール」 を参照ください。



この kernel.swu をインストールする際は /etc/swupdate\_preserve\_files の更新例の様に /boot と /lib/modules を維持するように追加します。カーネルをインストールした後に Armadillo Base OS を更新しても、この kernel.swu のカーネルが維持されます。

標準のカーネルに戻りたいか、以下の「図 10.121. Linux カーネルを build\_rootfs でインストールする方法」で Armadillo Base OS の更新のカーネルを使用したい場合は /etc/swupdate\_preserve\_files から /boot と /lib/modules の行を削除してください。

- ・ build\_rootfs で新しいルートファイルシステムをビルドする場合は build\_rootfs を展開した後以下コマンドでインストールしてください。

```
[ATDE ~/linux-[VERSION]]$ BROOTFS=$HOME/build-rootfs-[VERSION] ❶
[ATDE ~/linux-[VERSION]]$ sed -i -e '/^linux-at-a6/d' "$BROOTFS/a600/packages" ❷
[ATDE ~/linux-[VERSION]]$ cp -v arch/arm/boot/uImage "$BROOTFS/a600/resources/boot/"
'arch/arm/boot/uImage' -> '/home/atmark/build-rootfs-v3.17-at.7/a600/resources/boot/
uImage'
[ATDE ~/linux-[VERSION]]$ cp -v arch/arm/boot/dts/armadillo*. {dtb,dtbo} "$BROOTFS/a600/
resources/boot/"
'arch/arm/boot/dts/armadillo-640-at-dtweb.dtb' -> '/home/atmark/build-rootfs-v3.17-at.7/
a600/resources/boot/armadillo-640-at-dtweb.dtb'
: (省略)
[ATDE ~/linux-[VERSION]]$ rm -rfv "$BROOTFS/a600/resources/lib/modules" ❸
[ATDE ~/linux-[VERSION]]$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
INSTALL_MOD_PATH="$BROOTFS/a600/resources" -j5 modules_install
INSTALL arch/arm/crypto/chacha-neon.ko
INSTALL arch/arm/crypto/curve25519-neon.ko
: (省略)
DEPMOD [VERSION]
```

図 10.121 Linux カーネルを build\_rootfs でインストールする方法

- ❶ build\_rootfs のディレクトリ名を設定します。これによって、長いディレクトリ名を何度も入力する必要がなくなります。
- ❷ アットマークテクノが提供するカーネルをインストールしない様に、linux-at-a6@atmark と記載された行を削除します。
- ❸ 別のカーネルをすでにインストールしている場合は、新しいモジュールをインストールする前に古いモジュールを削除する必要があります。

### 10.4.3. Alpine Linux ルートファイルシステムをビルドする

ここでは、alpine/build-rootfs を使って、Alpine Linux ルートファイルシステムを構築する方法を説明します。

alpine/build-rootfs は、ATDE 上で Armadillo-640 用の Alpine Linux ルートファイルシステムを構築することができるツールです。

### 1. ルートファイルシステムのビルドに必要な Podman のインストール

次のコマンドを実行します。

```
[ATDE ~]$ sudo apt install podman btrfs-progs xxhash
```


### 2. alpine/build-rootfs の入手

Armadillo Base OS 対応 Armadillo-640 開発用ツール [https://armadillo.atmark-techno.com/resources/software/armadillo-640/abos-tools] から「Alpine Linux ルートファイルシステムビルドツール」ファイル (build-rootfs-[VERSION].tar.gz) を次のようにダウンロードします。

```
[ATDE ~/$ wget https://armadillo.atmark-techno.com/files/downloads/armadillo-640/tool/build-rootfs-latest.tar.gz
[ATDE ~/$ tar xf build-rootfs-latest.tar.gz
[ATDE ~/$ cd build-rootfs-[VERSION]
```

### 3. Alpine Linux ルートファイルシステムの変更

a600 ディレクトリ以下のファイルを変更することで、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。



common と a600 ディレクトリ直下にある fixup や packages などの同名ファイルは、それぞれのファイルを連結して利用されます。パッケージの削除などを行う場合は、common ディレクトリ以下のファイルも確認してください。

common と a600 内のサブディレクトリにある同名ファイルは、a600 のファイルが利用されます。

build-rootfs に含まれるファイルの説明は次の通りです。

**表 10.4 build-rootfs のファイル説明**

ファイル	説明
a600/resources/*	配置した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直下にコピーされます。ファイルを追加する場合は、このディレクトリに入れてください。
a600/packages	このファイルに記載されているパッケージはルートファイルシステムにインストールされます。パッケージを追加する場合はこのファイルに追加してください。
a600/fixup	このファイルに記載されているコマンドはパッケージのインストールが完了した後に実行されます。
a600/image_firstboot/*	配置したファイルやディレクトリは、「10.5.1. ブートディスクの作成」や「10.6.1.1. 初期化インストールディスクの作成」の手順のようにブートディスクイメージを作成する際、そのままルートファイルシステム直下にコピーされます。
a600/image_installer/*	配置したファイルやディレクトリは、「10.6.1.1. 初期化インストールディスクの作成」の手順のようにインストールディスクイメージを作成する際、そのままインストーラにコピーされます。ルートファイルシステムに影響はありません。
a600/image_common/*	配置したファイルやディレクトリは、ブートディスクイメージおよびインストールディスクイメージを作成する際、ルートファイルシステム、インストーラにそれぞれコピーされます。



利用可能なパッケージは以下のページで検索することができます。

**Alpine Linux Packages** <https://pkgs.alpinelinux.org/packages>

Alpine Linux ルートファイルシステムを起動している Armadillo でも検索することができます。

```
[armadillo ~]# apk update
[armadillo ~]# apk search ruby
ruby-test-unit-rr-1.0.5-r0
ruby-rmagick-5.1.0-r0
ruby-public_suffix-5.0.0-r0
:
: (省略)
:
ruby-mustache-1.1.1-r5
ruby-nokogiri-1.13.10-r0
```

#### 4. ビルド

次のコマンドを実行します。

パッケージをインターネット上から取得するため回線速度に依存しますが、ビルドには数分かかります。

```
[ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_rootfs.sh -b a600
use default(outdir=/home/atmark/git/build-rootfs)
use default(output=baseos-640-ATVERSION.tar.zst)
:
: (略)
:
> Creating rootfs archive
-rw-r--r--  1 root   root   231700480 Nov 26 07:18 rootfs.tar
ERROR: No such package: .make-alpine-make-rootfs
=====
footprint[byte]  tarball[byte]  packages
                229904000      74942331  alpine-base coreutils chrony ... (省略)
=====
done.
```



リリース時にバージョンに日付を含めたくないときは `--release` を引数に追加してください。





インターネットに接続できない環境か、テスト済みのソフトウェアのみをインストールしたい場合は Armadillo Base OS 対応 Armadillo-640 開発用ツール [<https://armadillo.atmark-techno.com/resources/software/armadillo-640/abos-tools>] からキャッシュアーカイブもダウンロードして、`build_rootfs.sh --cache baseos-640-[version].cache.tar` で使ってください。



任意のパス、ファイル名で結果を出力することもできます。

```
[ATDE ~/build-rootfs-[VERSION]]$ ./build_rootfs.sh -b a600 ~/
alpine.tar.zst
:
: (略)
:
[ATDE ~/build-rootfs-[VERSION]]$ ls ~/alpine.tar.zst
~/alpine.tar.zst
```



## 5. インストール

ビルドしたルートファイルシステムは、以下に示すどちらかの方法でインストールしてください。

- ・ `swupdate` でインストールする

`mkswu` の初期化を行った後に 提供されているスクリプトを使って SWU イメージを作成してください。

```
[ATDE ~/build-rootfs-[VERSION]]$ vi OS_update.desc
swdesc_tar --version base_os [VERSION] ¥
--preserve-attributes baseos-640-[VERSION].tar.zst
[ATDE ~/build-rootfs-[VERSION]]$ mkswu OS_update.desc
OS_update.swu を作成しました。
```

作成された `OS_update.swu` のインストールについては「10.7.3. イメージのインストール」を参照ください。

- ・ 「10.5.1. ブートディスクの作成」 でインストールする

手順を実行すると、ビルドされた `baseos-640-[VERSION].tar.zst` が自動的に利用されます。

## 10.5. SD ブートの活用

本章では、microSD カードから直接起動(以降「SD ブート」と表記します)する手順を示します。SD ブートを活用すると、microSD カードを取り替えることでシステムイメージを変更することができます。本章に示す手順を実行するためには、容量が 8Gbyte 以上の microSD カードを必要とします。



SD ブートをを行った場合、ブートローダーの設定は **microSD カード** に保存されます。

## 10.5.1. ブートディスクの作成

### 1. ブートディスクイメージのビルドします

「10.4.3. Alpine Linux ルートファイルシステムをビルドする」で説明されているソースツリー `alpine/build-rootfs` にあるスクリプト `build_image` と 「10.4.1. ブートローダーをビルドする」でビルドした `u-boot-dtb.imx` を利用します。

```
[ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_image.sh --board a600 ¥
--boot ~/u-boot-[VERSION]/u-boot-dtb.imx
: (省略)
[ATDE ~/build-rootfs-[VERSION]]$ ls baseos-640*img
baseos-640-[VERSION].img
```

### 2. ATDE に microSD カードを接続します。詳しくは「4.2.2. 取り外し可能デバイスの使用」を参考にしてください。

### 3. microSD カードのデバイス名を確認します

```
[ATDE ~]$ ls /dev/sd?
/dev/sda /dev/sdb
[ATDE ~]$ sudo fdisk -l /dev/sdb
Disk /dev/sdb: 7.22 GiB, 7751073792 bytes, 15138816 sectors
Disk model: SD/MMC
: (省略)
```

### 4. microSD カードがマウントされている場合、アンマウントします。

```
[ATDE ~]$ mount
: (省略)
/dev/sdb1 on /media/52E6-5897 type ext2
(rw,nosuid,nodev,relatime,uid=1000,gid=1000,fmask=0022,dmask=0077,codepage=cp437,ioccharset=utf8,shortname=mixed,showexec=utf8,flush,errors=remount-ro,uhelper=udisks)
[ATDE ~]$ sudo umount /dev/sdb1
```

図 10.122 自動マウントされた microSD カードのアンマウント

### 5. ブートディスクイメージの書き込み

```
[ATDE ~]$ sudo dd if=~/build-rootfs-[VERSION]/baseos-640-[VERSION].img ¥
of=/dev/sdb bs=1M oflag=direct status=progress
```

microSD カードの性能にもよりますが、書き込みには 5 分程度かかります。



microSD カードのパーティション構成は次のようになっています。

表 10.5 microSD カードのパーティション構成

パーティション	オフセット	サイズ	説明
-	0	10MiB	ブートローダー
1	10MiB	300MiB	A/B アップデートの A 面パーティション
2	310MiB	300MiB	A/B アップデートの B 面パーティション
3	610MiB	50MiB	ログ用パーティション
4	660MiB	200MiB	ファームウェア
5	860MiB	残り	アプリケーション用パーティション

gdisk で確認すると次のようになります。

```
[ATDE ~]$ sudo gdisk -l /dev/sdb
GPT fdisk (gdisk) version 1.0.8

Partition table scan:
  MBR: protective
  BSD: not present
  APM: not present
  GPT: present

Found valid GPT with protective MBR; using GPT.
Disk /dev/mmcblk1: 15319040 sectors, 7.3 GiB
Sector size (logical/physical): 512/512 bytes
Disk identifier (GUID): 309AD967-470D-4FB2-835E-7963578102A4
Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
First usable sector is 34, last usable sector is 15319006
Partitions will be aligned on 2048-sector boundaries
Total free space is 20446 sectors (10.0 MiB)

Number  Start (sector)    End (sector)  Size      Code  Name
   1            20480             634879    300.0 MiB   8300  rootfs_0
   2            634880           1249279    300.0 MiB   8300  rootfs_1
   3           1249280           1351679     50.0 MiB   8300  logs
   4           1351680           1761279    200.0 MiB   8300  firm
   5           1761280          60485632   28.0 GiB   8300  app
```

## 10.5.2. SD ブートの実行

「10.5.1. ブートディスクの作成」で作成したブートディスクから起動する方法を説明します。

1. Armadillo-640 に電源を投入する前に、ブートディスクを CON1(microSD スロット)に挿入します。また、JP1 と JP2 を共にジャンパでショートします。
2. 電源を投入します。

```
U-Boot 2020.04-at15 (Jun 09 2023 - 18:46:32 +0900)

CPU:   i.MX6ULL rev1.1 at 396 MHz
Model: Atmark Techno Armadillo-640
DRAM:  512 MiB
setup_rtc_disarm_alarm: Can't find bus
WDT:   Started with servicing (10s timeout)
PMIC:  PFUZE3000 DEV_ID=0x30 REV_ID=0x11
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
Loading Environment from MMC... OK
In:    mxc_serial
Out:   mxc_serial
Err:   mxc_serial
switch to partitions #0, OK
mmc1 is current device
flash target is MMC:1
Net:   eth0: ethernet@2188000
Fastboot: Normal
Saving Environment to MMC... Writing to MMC(1)... OK
Normal Boot
Hit any key to stop autoboot:  0
switch to partitions #0, OK
mmc1 is current device
Cannot lookup file boot/boot.scr
6859976 bytes read in 1420 ms (4.6 MiB/s)
Booting from mmc ...
37363 bytes read in 93 ms (391.6 KiB/s)
Loading fdt boot/armadillo.dtb
Cannot lookup file boot/overlays.txt
## Booting kernel from Legacy Image at 80800000 ...
   Image Name:   Linux-5.10.180-2-at
   Created:      2023-06-09  9:48:24 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    6859912 Bytes = 6.5 MiB
   Load Address: 82000000
   Entry Point:  82000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 83500000
   Booting using the fdt blob at 0x83500000
   Loading Kernel Image
   Loading Device Tree to 9ef1d000, end 9ef49fff ... OK

Starting kernel ...

... 中略...

Welcome to Alpine Linux 3.17
Kernel 5.10.180-2-at on an armv7l (/dev/ttyMxc0)

armadillo login:
```

## 10.6. Armadillo のソフトウェアの初期化

microSD カードを使用し、Armadillo Base OS の初期化を行えます。



初期化を行っても、ファームウェアパーティション(mmcbk0p4)は変更されません。

## 10.6.1. インストールディスクの作成

インストールディスクは二つの種類があります：

- ・ 初期化インストールディスク。Armadillo Base OS 対応 Armadillo-640 インストールディスクイメージ [https://armadillo.atmark-techno.com/resources/software/armadillo-640/abos-disc-image] にある標準のイメージです。
- ・ 開発が完了した Armadillo-640 をクローンするためのインストールディスク。

### 10.6.1.1. 初期化インストールディスクの作成

1. 512 MB 以上の microSD カードを用意してください。
2. 標準のインストールディスクイメージを使用する場合は、Armadillo Base OS 対応 Armadillo-640 インストールディスクイメージ [https://armadillo.atmark-techno.com/resources/software/armadillo-640/abos-disc-image] から「Armadillo Base OS」をダウンロードしてください。

「10.4. Armadillo のソフトウェアをビルドする」でビルドしたイメージを使用してインストールディスクを作成したい場合は、以下のコマンドを実行して、インストールディスクイメージを作成してください。

```
[ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_image.sh ¥
--firmware ~/at-imxlibpackage/imx_lib.img
: (省略)
[ATDE ~/build-rootfs-[VERSION]]$ ls baseos-a640*img
baseos-640-[VERSION].img
[ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_image.sh --board a600 ¥
--boot ~/u-boot-[VERSION]/u-boot-dtb.img ¥
--installer ./baseos-640-[VERSION].img
```

コマンドの実行が完了すると、baseos-640-[VERSION]-installer.img というファイルが作成されていますので、こちらを使用してください。

3. ATDE に microSD カードを接続します。詳しくは「4.2.2. 取り外し可能デバイスの使用」を参考にしてください。
4. microSD カードのデバイス名を確認します

```
[ATDE ~]$ ls /dev/sd?
/dev/sda /dev/sdb
[ATDE ~]$ sudo fdisk -l /dev/sdb
Disk /dev/sdb: 7.22 GiB, 7751073792 bytes, 15138816 sectors
Disk model: SD/MMC
: (省略)
```

- microSD カードがマウントされている場合、アンマウントします。

```
[ATDE ~]$ mount
: (省略)
/dev/sdb1 on /media/52E6-5897 type ext2
(rw,nosuid,nodev,relatime,uid=1000,gid=1000,mask=0022,dmask=0077,codepage=cp437,ioccharset
=utf8,shortname=mixed,showexec=utf8,flush,errors=remount-ro,uhelper=udisks)
[ATDE ~]$ sudo umount /dev/sdb1
```



- ダウンロードしたファイルを展開し、img ファイルを microSD カードに書き込んでください。  
Linux PC の場合、以下のように microSD カードに書き込むことができます。

```
[ATDE ~]$ unzip baseos-600-installer-[VERSION].zip
[ATDE ~]$ sudo dd if=baseos-600-installer-[VERSION].img ¥
of=/dev/sdb bs=1M oflag=direct status=progress
```

また、Windows の場合、エクスプローラー等で Zip ファイルから img ファイルを取り出し、「Win32 Disk Imager」などを使用して microSD カードに書き込むことができます。

### 10.6.1.2. 開発が完了した Armadillo をクローンするインストールディスクの作成

- microSD カードを用意してください。Armadillo-640 にインストールされてるソフトウェアをコピーしますので、場合によって 8GB 以上のカードが必要です。
- 初期化インストールディスクをベースとしますので、「10.6.1.1. 初期化インストールディスクの作成」でビルドした SD カードを使用できますが、用意されていない場合は次のステップで自動的にダウンロードされます。
- abos-ctrl make-installer を実行してください

```
[armadillo ~]# abos-ctrl make-installer
It looks like your SD card does not contain an installer image
Download base SD card image from https://armadillo.atmark-techno.com (~200MB) ? [y/N]
WARNING: it will overwrite your sd card!!
y
Downloading installer image
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
   Dload  Upload  Total   Spent    Left   Speed
100 167M  100 167M    0     0  104M      0  0:00:01  0:00:01  --:--:-- 104M
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
   Dload  Upload  Total   Spent    Left   Speed
100   70  100   70    0     0  1441      0  --:--:--  --:--:--  --:--:-- 1458
Writing baseos-600-installer-3.17.3-at.7.img to SD card (442M)
439353344 bytes (439 MB, 419 MiB) copied, 134 s, 3.3 MB/s
421+0 records in
421+0 records out
441450496 bytes (441 MB, 421 MiB) copied, 134.685 s, 3.3 MB/s
Verifying written image is correct
436207616 bytes (436 MB, 416 MiB) copied, 46 s, 9.5 MB/s
421+0 records in
421+0 records out
441450496 bytes (441 MB, 421 MiB) copied, 46.8462 s, 9.4 MB/s
Checking and growing installer main partition
```

```
GPT data structures destroyed! You may now partition the disk using fdisk or
other utilities.
Setting name!
partNum is 0
The operation has completed successfully.
e2fsck 1.46.4 (18-Aug-2021)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
rootfs_0: 2822/102400 files (0.5% non-contiguous), 352391/409600 blocks
(1/1) Installing e2fsprogs-extra (1.46.4-r0)
Executing busybox-1.34.1-r5.trigger
OK: 202 MiB in 197 packages
resize2fs 1.46.4 (18-Aug-2021)
Resizing the filesystem on /dev/mmcblk1p1 to 15547884 (1k) blocks.
The filesystem on /dev/mmcblk1p1 is now 15547884 (1k) blocks long.

Currently booted on /dev/mmcblk0p1
Copying boot image
Copying rootfs
301989888 bytes (302 MB, 288 MiB) copied, 10 s, 30.1 MB/s
300+0 records in
300+0 records out
314572800 bytes (315 MB, 300 MiB) copied, 10.3915 s, 30.3 MB/s
Copying /opt/firmware filesystem
Copying appfs
At subvol app/snapshots/volumes
At subvol app/snapshots/boot_volumes
At subvol app/snapshots/boot_containers_storage
Cleaning up and syncing changes to disk...
Installer updated successfully!
```

## 10.6.2. インストールディスクを使用する

1. JP1 と JP2 を共にジャンパーでショート(SD ブートに設定)し、microSD カードを CON1 に挿入します。
2. 電源を投入すると、1 分程度で eMMC のソフトウェアの初期化が完了します。
3. 完了すると電源が切れます(全ての LED が消灯、コンソールに reboot: Power down が表示)。
4. 電源を取り外し、続いて JP1 と JP ジャンパーと microSD カードを外してください。
5. 10 秒以上待ってから再び電源を入れると、初回起動時と同じ状態になります。

## 10.7. Armadillo のソフトウェアをアップデートする

Armadillo-640 では、開発・製造・運用それぞれに適した複数のソフトウェアアップデート方法を用意しています。本章では、それぞれのソフトウェアアップデート方法について説明します。

ソフトウェアアップデートを実現するソフトウェアの概要や仕様、用語については「13. ソフトウェア仕様」を参照してください。

## 10.7.1. SWU イメージとは

Armadillo Base OS ではソフトウェアアップデートのために OS やコンテナ等を格納するために SWU というイメージ形式を使います。

SWU イメージは `swupdate` (<https://sbabic.github.io/swupdate/swupdate.html>) によって Armadillo Base OS 上で検証とインストールが実行されます。SWU イメージを Armadillo に転送するための方法は、用途や状況に合わせて様々な方法を用意しています。例えば、USB メモリから読み取る、ウェブサーバーからダウンロードする、hawkBit という Web アプリケーションを使うなどです。

## 10.7.2. SWU イメージの作成

SWU イメージの作成には、`mkswu` というツールを使います。

`mkswu` に含まれる `mkswu` を実行すると、アップデート対象やバージョン等の情報を記載した `.desc` ファイルに含まれる命令を順次実行してイメージを作り上げます。

詳しくは「10.7.6. `mkswu` の `desc` ファイル」を参考にしてください。

### 1. `mkswu` の取得

```
[ATDE ~]$ sudo apt update && sudo apt install mkswu
```

インストール済みの場合は、以下のコマンドを実行し最新版への更新を行ってください。

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



git のバージョンからアップデートする場合、`mkswu --import` で以前使っていたコンフィグをロードしてください。

```
[ATDE ~/swupdate-mkimage]$ mkswu --import
コンフィグファイルを更新しました: /home/atmark/swupdate-mkimage/
mkswu.conf
/home/atmark/swupdate-mkimage/mkswu.conf のコンフィグファイルとそ
の鍵を
/home/atmark/mkswu にコピーします。
mkdir: ディレクトリ '/home/atmark/mkswu' を作成しました
'/home/atmark/swupdate-mkimage/swupdate.key' -> '/home/atmark/
mkswu/swupdate.key'
'/home/atmark/swupdate-mkimage/swupdate.pem' -> '/home/atmark/
mkswu/swupdate.pem'
/home/atmark/swupdate-mkimage/mkswu.conf のコンフィグファイルを
/home/atmark/mkswu/mkswu.conf にコピーしました。
mkswu でイメージ作成を試してから前のディレクトリを消してください。
```

### 2. 最初に行う設定

`mkswu --init` を実行して鍵や最初の書き込み用のイメージを生成します。作成する鍵は、SWU パッケージを署名するために使用します。



過去に本手順を行っている場合、再度初回アップデート作業を行う必要はありません。再度アップデートを行う際には、Armadillo に配置した公開鍵に対応する秘密鍵でアップデートを行いますので、「10.7.6. mkswu の desc ファイル」を参考にしてください。

```
[ATDE ~]$ mkswu --init
mkdir: ディレクトリ '/home/atmark/mkswu' を作成しました
コンフィグファイルを更新しました: /home/atmark/mkswu/mkswu.conf
証明書の Common name を入力してください: [COMMON_NAME] ❶
証明書の鍵のパスワードを入力ください (4-1024 文字) ❷
証明書の鍵のパスワード (確認):
Generating an EC private key
writing new private key to '/home/atmark/mkswu/swupdate.key'
-----
アップデートイメージを暗号化しますか? (N/y) ❸
アットマークテクノが作成したイメージをインストール可能にしますか? (Y/n) ❹
root パスワード: ❺
root パスワード (確認):
atmark ユーザのパスワード (空の場合はアカウントをロックします): ❻
atmark ユーザのパスワード (確認):
BaseOS イメージの armadillo.atmark-techno.com サーバーからの自動アップデートを行いますか?
(y/N) ❼
/home/atmark/mkswu/initial_setup.swu を作成しました。

"/home/atmark/mkswu/initial_setup.swu" をそのまま使うことができますが、
モジュールを追加してイメージを再構築する場合は次のコマンドで作成してください:
mkswu "/home/atmark/mkswu/initial_setup.desc" [他の.desc ファイル]

インストール後は、このディレクトリを削除しないように注意してください。
鍵を失うと新たなアップデートはデバイスの /etc/swupdate.pem
を修正しないとインストールできなくなります。

[ATDE ~]$ ls ~/mkswu
initial_setup.desc  initial_setup.swu  mkswu.conf
swupdate.aes-key    swupdate.key        swupdate.pem ❽
```

- ❶ COMMON\_NAME には証明鍵の「common name」として会社や製品が分かるような任意の名称を入力してください。
- ❷ 証明鍵を保護するパスフレーズを 2 回入力します。
- ❸ swu イメージ自体を暗号化する場合に「y」を入力します。詳細は「10.7.9. SWUpdate と暗号化について」を参考にしてください。
- ❹ アットマークテクノのアップデートをインストールしない場合は「n」を入力します。
- ❺ root のパスワードを 2 回入力します。
- ❻ atmark ユーザーのパスワードを 2 回入力します。何も入力しない場合はユーザーをロックします。
- ❼ 自動アップデートを無効のままに進みます。ここで「y」を入れると、定期的にアットマークテクノのサーバーからアップデートの有無を確認し、自動的にインストールします。
- ❽ 作成したファイルを確認します。「swupdate.aes-key」は暗号化の場合にのみ作成されます。

このイメージは初回インストール用の署名鍵を使って、作成した鍵とユーザーのパスワードを設定します。

インストール後にコンフィグの `mkswu.conf` と鍵の `swupdate.*` をなくさないようにしてください。



このイメージに他の変更も入れれます。他の `/usr/share/mkswu/examples/` ディレクトリにある `.desc` ファイルや「10.7.6. mkswu の `desc` ファイル」を参考にして、以下の例のように同じ `swu` にいくつかの `.desc` を組み込めます。

例えば、`openssh` を有効にします。

```
[ATDE ~/mkswu]$ cp -rv /usr/share/mkswu/examples/enable_sshd* .
: (省略)
'/usr/share/mkswu/examples/enable_sshd/root/.ssh/
authorized_keys'
-> './enable_sshd/root/.ssh/authorized_keys'
'/usr/share/mkswu/examples/enable_sshd.desc' -> './
enable_sshd.desc'
[ATDE ~/mkswu]$ cp ~/.ssh/id_rsa.pub ¥
enable_sshd/root/.ssh/authorized_keys
[ATDE ~/mkswu]$ mkswu initial_setup.desc enable_sshd.desc
enable_sshd.desc を組み込みました。
initial_setup.swu を作成しました。
```



### 3. イメージのインストール

「10.7.3. イメージのインストール」を参考に、作成したイメージをインストールしてください。

### 4. 次回以降のアップデート

次回以降のアップデートは作成した証明鍵を使用して Armadillo-640 の SWU イメージを作成します。

`.desc` ファイルの内容は `/usr/share/mkswu/examples/` のディレクトリや「10.7.6. mkswu の `desc` ファイル」を参考にしてください。

## 10.7.3. イメージのインストール

イメージをインストールする方法として以下に示すような方法があります。もし、作成した SWU イメージのインストールに失敗する場合は、「10.7.4. `swupdate` がエラーする場合の対処」をご覧ください。

- ・ USB メモリまたは SD カードからの自動インストール

Armadillo-640 に USB メモリまたは SD カードを接続すると自動的にアップデートが始まります。アップデート終了後に Armadillo-640 は自動で再起動します。

USB メモリや SD カードを `vfat` もしくは `ext4` 形式でフォーマットし、作成した `swu` のファイルをディレクトリを作らずに配置してください。



ATDE 上で USB メモリ/microSD カードのパーティションを作成・フォーマットする方法

<https://armadillo.atmark-techno.com/howto/atde-partition-howto>

```
[ATDE ~/mkswu]$ df -h
Filesystem      Size  Used Avail Use% Mounted on
: (省略)
/dev/sda1       15G  5.6G  9.1G  39% /media/USBDRIVE ❶
[ATDE ~/mkswu]$ cp initial_setup.swu /media/USBDRIVE/ ❷
[ATDE ~/mkswu]$ umount /media/USBDRIVE ❸
```

- ❶ USB メモリがマウントされている場所を確認します。
- ❷ ファイルをコピーします。
- ❸ /media/USBDRIVE をアンマウントします。コマンド終了後に USB メモリを取り外してください。

エラーの場合、/var/log/message に保存されます。例えば、コンソールで証明の間違ったイメージのエラーを表示します：

```
[armadillo ~]# tail /var/log/messages
Nov 19 10:48:42 user.notice swupdate-auto-update: Mounting sda0 on /mnt
Nov 19 10:48:42 user.notice swupdate-auto-update: Trying update /mnt/initial_setup.swu
Nov 19 10:48:42 user.info swupdate: START Software Update started !
Nov 19 10:48:42 user.err swupdate: FAILURE ERROR : Signature verification failed ❶
Nov 19 10:48:42 user.err swupdate: FAILURE ERROR : Compatible SW not found
Nov 19 10:48:42 user.err swupdate: FATAL_FAILURE Image invalid or corrupted. Not installing ...
```

- ❶ 証明が間違ったメッセージ。

#### ・ 外部記憶装置からイメージのインストール (手動)

USB メモリや microSD カード等の外部記憶装置のルートディレクトリ以外に swu イメージを保存して、イメージのインストールを行います。ルートディレクトリに保存すると自動アップデートが行われますので、/var/log/messages を確認してください。

以下は外部記憶装置が/dev/mmcblk1p1 (microSD カード) として認識された場合に、イメージのインストールを行う例です。

```
[armadillo ~]# mount /dev/mmcblk1p1 /mnt
[armadillo ~]# swupdate -i /mnt/swu/initial_setup.swu
SWUpdate v5f2d8be-dirty

Licensed under GPLv2. See source distribution for detailed copyright notices.

[INFO ] : SWUPDATE running : [main] : Running on AGX4500 Revision at1
[INFO ] : SWUPDATE started : Software Update started !
[INFO ] : SWUPDATE running : [read_lines_notify] : No base os update: copying current os over
[INFO ] : SWUPDATE running : [read_lines_notify] : Removing unused containers
[INFO ] : SWUPDATE running : [read_lines_notify] : swupdate triggering reboot!
Killed
```

・ ウェブサーバーからイメージのインストール (手動)

swu イメージをウェブサーバーにアップロードして、イメージのインストールを行います。以下は、`http://server/initial_setup.swu` のイメージをインストールする例です。

```
[armadillo ~]# swupdate -d '-u http://server/initial_setup.swu'
SWUpdate v5f2d8be-dirty

Licensed under GPLv2. See source distribution for detailed copyright notices.

[INFO ] : SWUPDATE running : [main] : Running on AGX4500 Revision at1
[INFO ] : SWUPDATE running : [channel_get_file] : Total download size is 25 kB.
[INFO ] : SWUPDATE started : Software Update started !
[INFO ] : SWUPDATE running : [read_lines_notify] : No base os update: copying current os over
[INFO ] : SWUPDATE running : [read_lines_notify] : Removing unused containers
[INFO ] : SWUPDATE running : [read_lines_notify] : swupdate triggering reboot!
Killed
```


・ ウェブサーバーからの定期的な自動インストール

`swupdate-url` を有効にしたら、定期的にチェックしてインストールします。以下はサービスの有効化とタイミングの設定の例です。

```
[armadillo ~]# rc-update add swupdate-url ❶
[armadillo ~]# persist_file /etc/runlevels/default/swupdate-url ❷
[armadillo ~]#
    echo https://download.atmark-techno.com/armadillo-640/image/baseos-640-latest.swu ¥
    > /etc/swupdate.watch ❸
[armadillo ~]# echo 'schedule="0 tomorrow"' > /etc/conf.d/swupdate-url
[armadillo ~]# echo 'rdelay="21600"' >> /etc/conf.d/swupdate-url ❹
[armadillo ~]# persist_file /etc/swupdate.watch /etc/conf.d/swupdate-url ❺
```

- ❶ swupdate-url サービスを有効します。
- ❷ サービスの有効化を保存します。
- ❸ イメージの URL を登録します。一行ごとにイメージの URL を設定することができ、複数行にイメージの URL を設定することができます。
- ❹ チェックやインストールのスケジュールを設定します。
- ❺ 変更した設定ファイルを保存します。

USB メモリからのアップデートと同様に、ログは `/var/log/messages` に保存されます。



`initial_setup` のイメージを作成の際に `/usr/share/mkswu/examples/enable_swupdate_url.desc` を入れると有効にすることができます。

・ hawkBit を使用した自動インストール

hawkBit で Armadillo-640 を複数台管理してアップデートすることができます。「10.7.5. hawkBit サーバーから複数の Armadillo に配信する」を参考にしてください。

## 10.7.4. swupdate がエラーする場合の対処

SWU イメージのインストール動作は、「10.7.1. SWU イメージとは」で述べたように swupdate が実行します。mkswu で作成した SWU イメージの内容が適切でなかったり、あるいは、ストレージの空き容量が不足していたりするなど、いくつかの理由で swupdate のインストール動作が失敗することがあります。インストールに失敗すると、swupdate は /var/log/messages にエラーメッセージのログを残しますので、エラーメッセージを見ると、エラーの内容・原因が分かります。

エラーの原因ごとに、エラーメッセージとエラーの内容および対処方法を記した FAQ ページ (<https://armadillo.atmark-techno.com/faq/swupdate-troubleshooting-abos>) を公開しています。SWU イメージのインストールに失敗して対処法が分からないときは、この FAQ ページをご覧ください。

## 10.7.5. hawkBit サーバーから複数の Armadillo に配信する

hawkBit サーバーを利用することで複数の Armadillo のソフトウェアをまとめてアップデートすることができます。

手順は次のとおりです。

### 1. コンテナ環境の準備

Docker を利用すると簡単にサーバーを準備できます。Docker の準備については <https://docs.docker.com/get-docker/> を参照してください。

Docker の準備ができたら、要件に合わせてコンテナの設定を行います。

#### ・ ATDE の場合

- ・ `apt update && apt install mkswu` で最新のバージョンを確認してください。
- ・ ポート転送も必要です。一番シンプルな、プロキシを使用しない場合は 8080、TLS を使う場合は 443 を転送してください。

vmware を使う場合は vmware の NAT モードのネットワークを使用している仮想マシン上で Web サーバを構成する [<https://kb.vmware.com/s/article/2006955?lang=ja>] ページを参考にしてください。

- ・ ホスト PC の IP アドレスを控えておいてください。

#### ・ ATDE 以外の場合

- ・ Armadillo Base OS 対応 Armadillo-640 開発用ツール [<https://armadillo.atmark-techno.com/resources/software/armadillo-640/abos-tools>] から「Hawkbit docker-compose コンテナ」をダウンロードして展開してください。この場合、以下に `/usr/share/mkswu/hawkbit-compose` を使う際に展開先のディレクトリとして扱ってください。
- ・ docker がアクセスできるホスト名前やアドレスを控えておいてください。

### 2. hawkBit サーバーの準備

`/usr/share/mkswu/hawkbit-compose/setup_container.sh` を実行して、質問に答えてください。

以下に簡単な (TLS を有効にしない) テスト用の場合と、TLS を有効にした場合の例を示します。

setup\_container.sh を一度実行した場合はデータのディレクトリにある setup\_container.sh のリンクを実行して、ユーザーの追加等のオプション変更を行うこともできます。詳細は`--help`を参考にしてください。

```
[ATDE ~]$ /usr/share/mkswu/hawkbit-compose/setup_container.sh
docker-compose の設定ファイルと hawkBit のデータをどこに保存しますか? [/home/atmark/hawkbit-
compose] ❶
setup_container.sh へのリンクを /home/atmark/hawkbit-compose に作ります。
docker サービスに接続できませんでした。sudo でもう一度試します。
[sudo] atmark のパスワード: ❷
OK!
Hawkbit admin user name [admin] ❸
admin ユーザーのパスワード: ❹
パスワードを再入力してください:
追加の管理人アカウントのユーザーネーム (空にすると追加しません) ❺
hawkBit の「device」ユーザーを登録しますか? (自動登録用) [Y/n] ❻
device ユーザーのパスワード:
パスワードを再入力してください:
hawkBit の「mkswu」ユーザーを登録しますか? (swu のアップロード用) [Y/n] ❼
ユーザーにロールアウトの権限を与えますか? (インストール要求を出すこと) [Y/n] ❽
mkswu ユーザーのパスワード:
パスワードを再入力してください:
Setup TLS reverse proxy? [y/N] ❾

コンテナの設定が完了しました。docker-compose コマンドでコンテナの管理が可能です。
/home/atmark/hawkbit-compose/setup_container.sh を再び実行すると設定の変更が可能です。
hawkBit コンテナを起動しますか? [Y/n] ❿
Creating network "hawkbit-compose_default" with the default driver
Pulling mysql (mysql:5.7)...
: (省略)
Creating hawkbit-compose_hawkbit_1 ... done
Creating hawkbit-compose_mysql_1 ... done
```

図 10.123 hawkBit コンテナの TLS なしの場合 (テスト用) の実行例

- ❶ コンテナのコンフィグレーションとデータベースの場所を設定します。
- ❷ docker の設定によって sudo が必要な場合もあります。
- ❸ admin ユーザーのユーザー名を入力します。
- ❹ admin ユーザーのパスワードを二回入力します。
- ❺ 追加のユーザーが必要な場合に追加できます。
- ❻ examples/hawkbit\_register.desc で armadillo を登録する場合に作っておいてください。詳細は「10.7.5.2. SWU で hawkBit を登録する」を参考にしてください。
- ❼ hawkbit\_push\_update でアップデートを CLI で扱う場合は、「Y」を入力してください。詳細は「10.7.5.1. hawkBit のアップデート管理を CLI で行う」を参照してください。
- ❽ hawkbit\_push\_update でアップデートを実行する場合は、「Y」を入力してください。
- ❾ ここでは http でテストのコンテナを作成するので、「N」のままで進みます。
- ❿ コンテナを起動します。初期化が終わったら <IP>:8080 でアクセス可能になります。

```
[ATDE ~]$ /usr/share/mkswu/hawkbit-compose/setup_container.sh
docker-compose の設定ファイルと hawkBit のデータをどこに保存しますか? [/home/atmark/hawkbit-
compose]
setup_container.sh へのリンクを /home/atmark/hawkbit-compose に作ります。
docker サービスに接続できませんでした。sudo でもう一度試します。
OK!
Hawkbit admin user name [admin]
admin ユーザーのパスワード:
パスワードを再入力してください:
パスワードが一致しません。
admin ユーザーのパスワード:
パスワードを再入力してください:
追加の管理人アカウントのユーザーネーム (空にすると追加しません)
hawkBit の「device」ユーザーを登録しますか? (自動登録用) [Y/n]
device ユーザーのパスワード:
パスワードを再入力してください:
hawkBit の「mkswu」ユーザーを登録しますか? (swu のアップロード用) [Y/n]
ユーザーにロールアウトの権限を与えますか? (インストール要求を出すこと) [Y/n]
mkswu ユーザーのパスワード:
パスワードを再入力してください:
Setup TLS reverse proxy? [y/N] y ❶
lighttpd が起動中で、リバースプロキシ設定と競合しています。
lighttpd サービスを停止しますか? [Y/n] ❷
Synchronizing state of lighttpd.service with SysV service script with /lib/systemd/systemd-
sysv-install.
Executing: /lib/systemd/systemd-sysv-install disable lighttpd
Removed /etc/systemd/system/multi-user.target.wants/lighttpd.service.
リバースプロキシの設定に証明書の domain name が必要です。
この domain はこのままデバイスからアクセスできる名前にしてください。
例えば、https://hawkbit.domain.tld でアクセスしたら hawkbit.domain.tld、
https://10.1.1.1 でしたら 10.1.1.1 にしてください。
証明書の domain name: 10.1.1.1 ❸
証明書の有効期限を指定する必要があります。Let's encrypt を使用する場合、
この値は新しい証明書が生成されるまでしか使用されないの、デフォルトの値
のままにしておくことができます。Let's encrypt を使用しない場合、
数年ごとに証明書を新しくすることが最も好まれます。
証明書の有効期間は何日間になりますか? [3650] ❹
クライアントの TLS 認証を設定するために CA が必要です。
署名 CA のファイルパス (空にするとクライアント TLS 認証を無効になります) [] ❺
サーバーが直接インターネットにアクセス可能であれば、Let's Encrypt の証明書
を設定することができます。TOS への同意を意味します。
https://letsencrypt.org/documents/LE-SA-v1.2-November-15-2017.pdf
certbot コンテナを設定しますか? [y/N] ❻

/home/atmark/hawkbit-compose/data/nginx_certs/proxy.crt を /usr/local/share/ca-
certificates/ にコピーして、 update-ca-certificates を実行する必要があります。
この base64 でエンコードされたコピーを examples/hawkbit_register.sh の
SSL_CA_BASE64 に指定する手順が推奨されます。

LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSB0tLS0tCk1JSUJlekNDQVNHZ0F3SUJBZ0lVQTByZ0cwcTJF
SFNnampb0tUZWg3aGlaSVVVd0NnWUllLb1pJemowRUF3SxcKRXpFuk1B0EdBMVVFQXd3SU1UQXVn
UzR4TGpFd0hoY05Nak13TXpJMU1EVXh0VFU0V2hjTk16SXdkNek15TURVeAp0VFU0V2pBVE1SRXdE
d1lEVlFRERBz3hNQzR4TGpFdU1UQlPnQk1HQnlxR1NNNDlBZ0VH0Q0Nxr1NNNDlBd0VICkEwSUFc
SDFFREhBN3NOTlFJUDlTdLhLUhNmWj12dVVFVWkRkMVE2TzViriLV2RTh4UjUwUjBCLzNlajMzd0VI
NEoKYmZqb296bEpXaExlSG5SbGZsaHExVDlKdm5TaLV6QlJNQjBHQTFVZERnUVdCQlFBUmYvSkdT
dkVJek5xZ2JMNQpQamY2VGRpSk1EQWZCZ05WSFNRRUdEQVdnQlFBUmYvSkdTdkVJek5xZ2JMNVBq
```

↩

↩

↩

```
ZjZUZG1KtURBUEJnTLZIUk1CckFm0EVCVEFEQVFIL01Bb0dDQ3FHU0000UJBTUNBMGdBTUVVQ0LD
Nis3ZzJlZk1SRXl0RVk5WDhDNC8vUEw1U1kKWUlgZHUxVFZiUEZrSlV0SUFpRUE4bm1VSnVQSF1z
SHg2N2ErZFRwSXZ1QmJUSG1KbWd6dU13bTJ2RXppRnZRPQotLS0tLUVORCBDRVJUSUZJQ0FURS0t
LS0tCg== ⑦
```

Let's encrypt の設定は後で足したい場合に `setup_container.sh` を `--letsencrypt` で実行してください。

コンテナの設定が完了しました。 `docker-compose` コマンドでコンテナの管理が可能です。  
`/home/atmark/hawkbit-compose/setup_container.sh` を再び実行すると設定の変更が可能です。  
hawkBit コンテナを起動しますか? [Y/n]

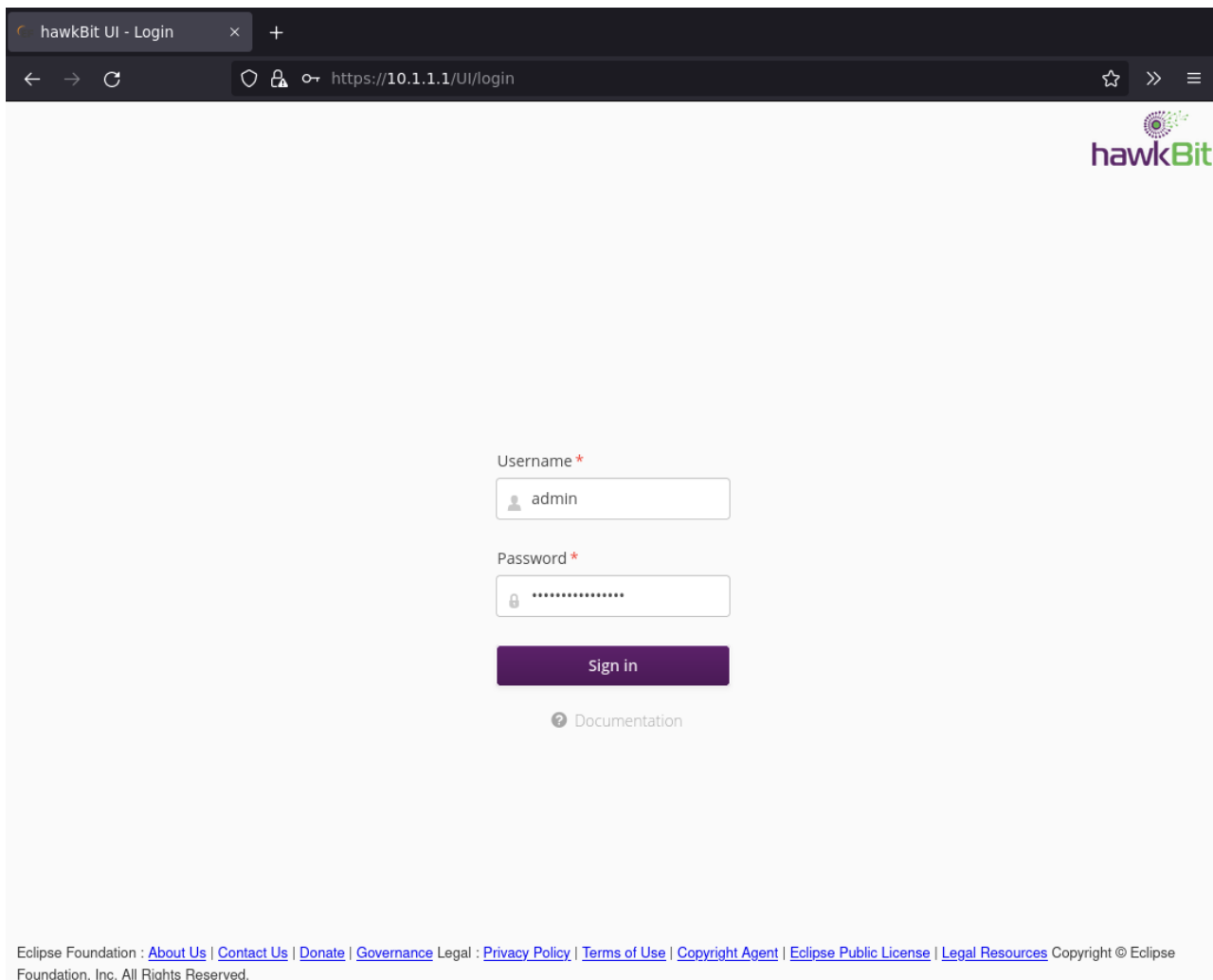
### 図 10.124 hawkBit コンテナの TLS ありの場合の実行例

- ① 今回は TLS を有効にするので、「y」を入力します。
- ② `lighttpd` サービスが起動している場合に聞かれます。不要なので、停止します。
- ③ 証明書の `common name` を入力してください。ATDE の場合、ポート転送によってホストの IP アドレスで接続しますのでそのアドレスを入力します。Let's encrypt を使用する場合には外部からアクセス可能な DNS を入力してください。
- ④ 証明書の有効期間を設定します。デフォルトでは 10 年になっています。Let's encrypt を使用する場合には使われていません。
- ⑤ クライアント側では x509 証明書で認証をとることができますが、この例では使用しません。
- ⑥ Let's encrypt による証明書を作成できます。ATDE の場合は外部からのアクセスが難しいので、この例では使用しません。
- ⑦ 自己署名証明書を作成したので、Armadillo に設置する必要があります。この証明書の取扱いは「10.7.5.2. SWU で hawkBit を登録する」を参照してください。

### 3. hawkBit へのログイン

作成したコンテナによって `http://<サーバーの IP アドレス>:8080` か `https://<サーバーのアドレス>` にアクセスすると、ログイン画面が表示されます。



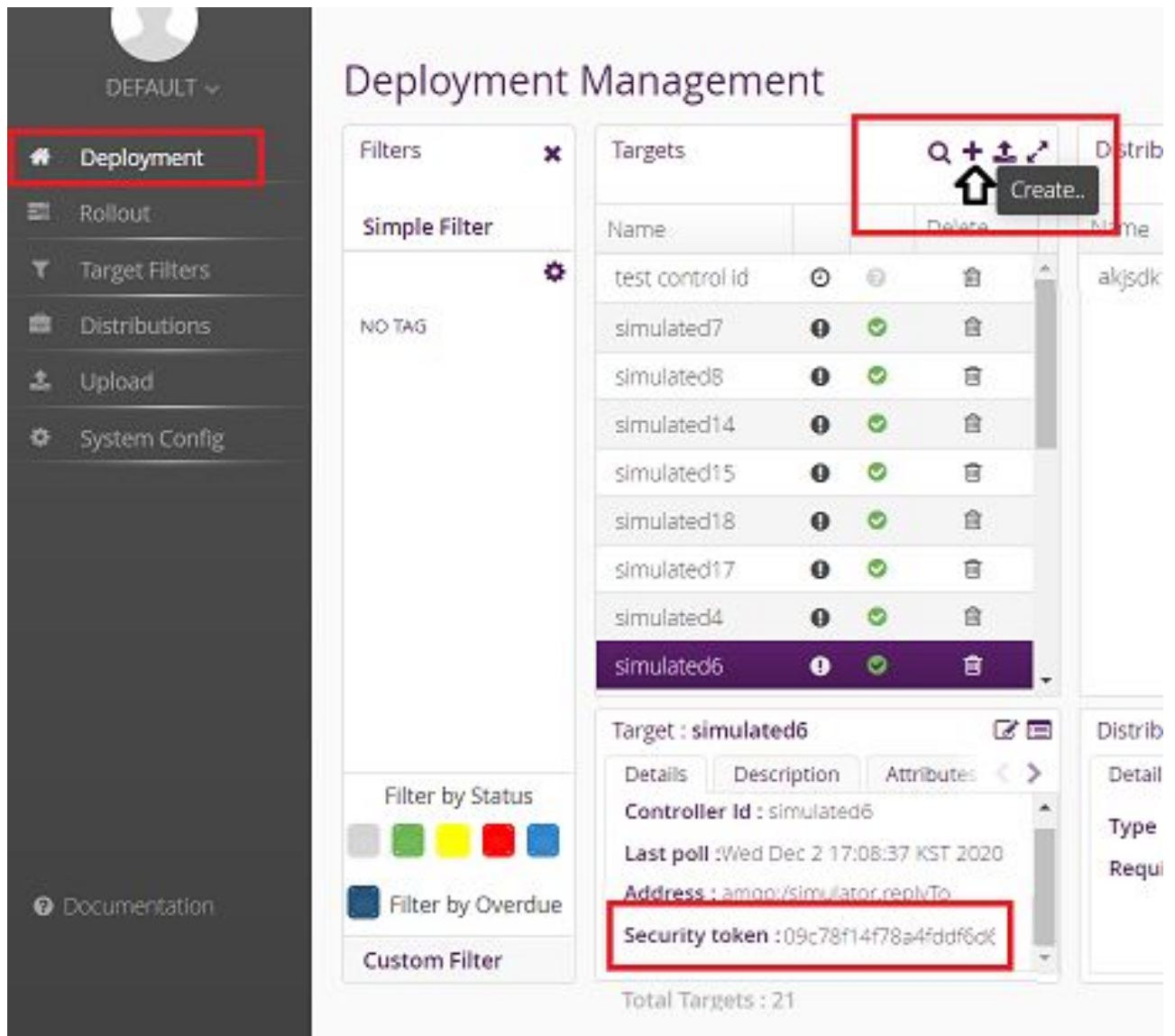


デフォルトでは次のアカウントでログインできます。

ユーザー	admin
パスワード	admin

#### 4. Armadillo を Target に登録する

左側のメニューから Deployment をクリックして、Deployment の画面に移ります。



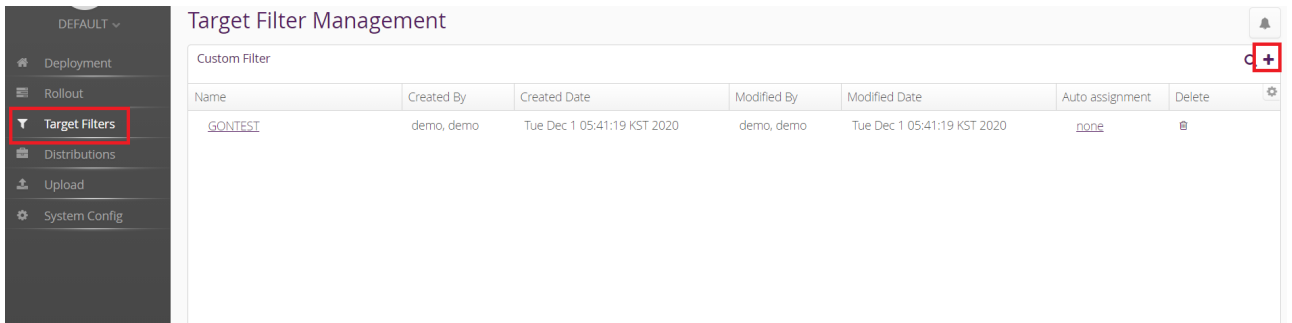
"+"をクリックして Target を作成します。

作成したターゲットをクリックすると、下のペインに "Security token:<文字列>" と表示されるので、<文字列>の部分メモします。

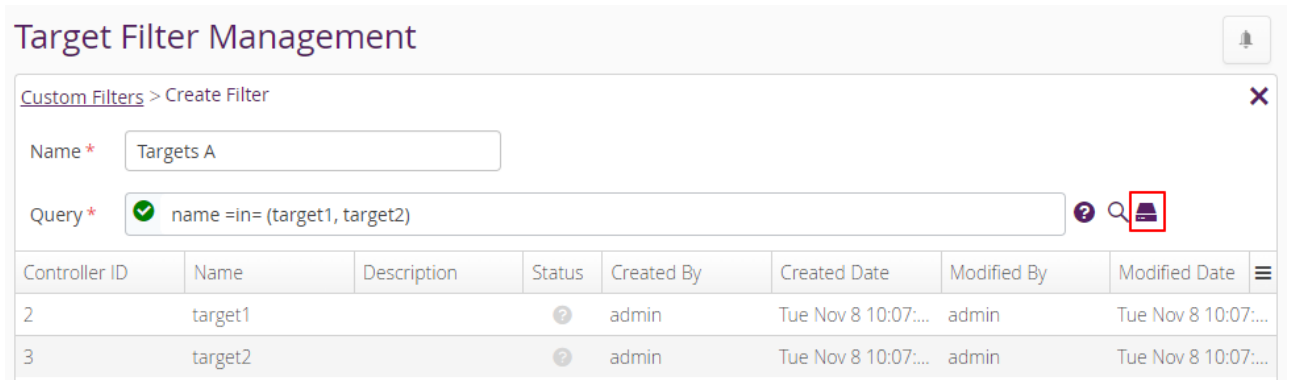
メモした<文字列>を Armadillo の /etc/swupdate.cfg に設定すると Hawkbit への接続認証が通るようになります。

## 5. Target Filter を作成する

左側のメニューから"Target Filters"をクリックして、Target Filters の画面に移ります。



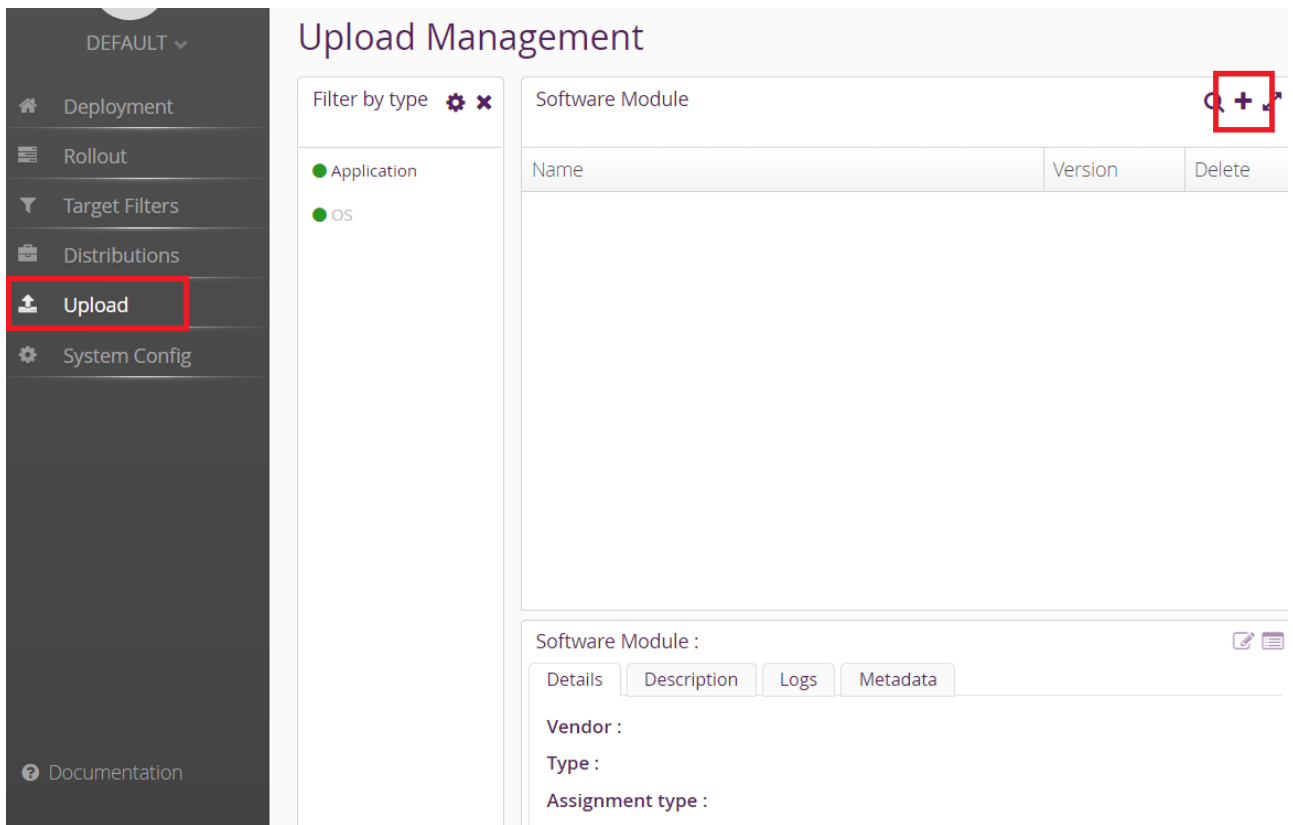
"+" をクリックして新規に Target Filter を作成します。



Filter name と フィルタリング条件を入力して保存します。

## 6. Software module を作成する

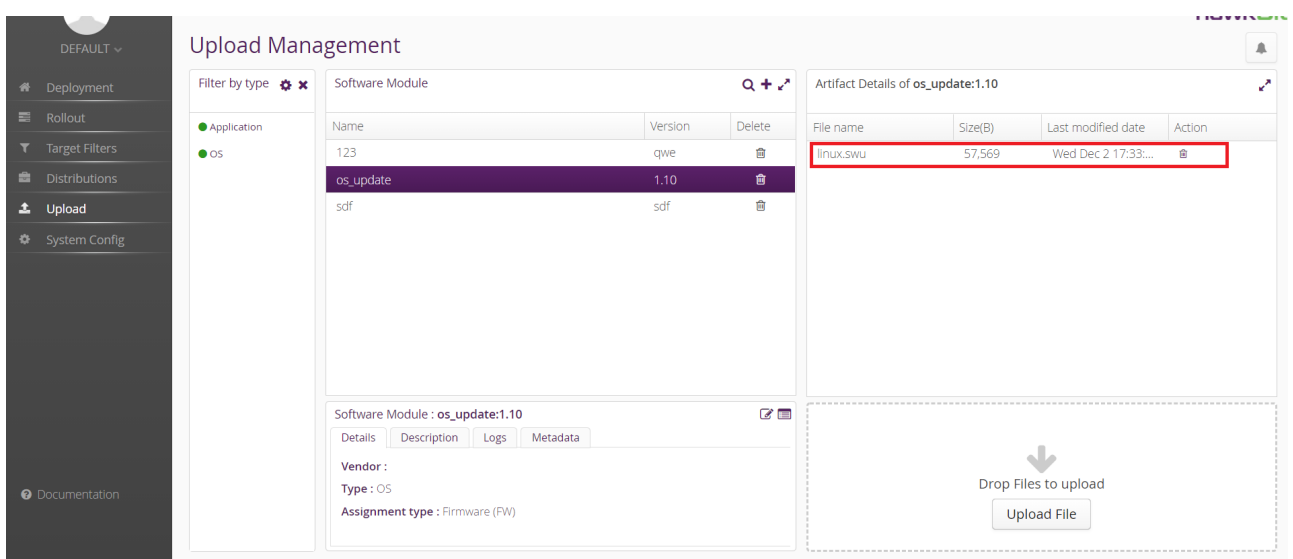
左側のメニューから"Upload"をクリックして、Upload Management の画面に移ります。



"+" をクリックして Software module を作成します。type には OS/Application、version には任意の文字列を指定します。

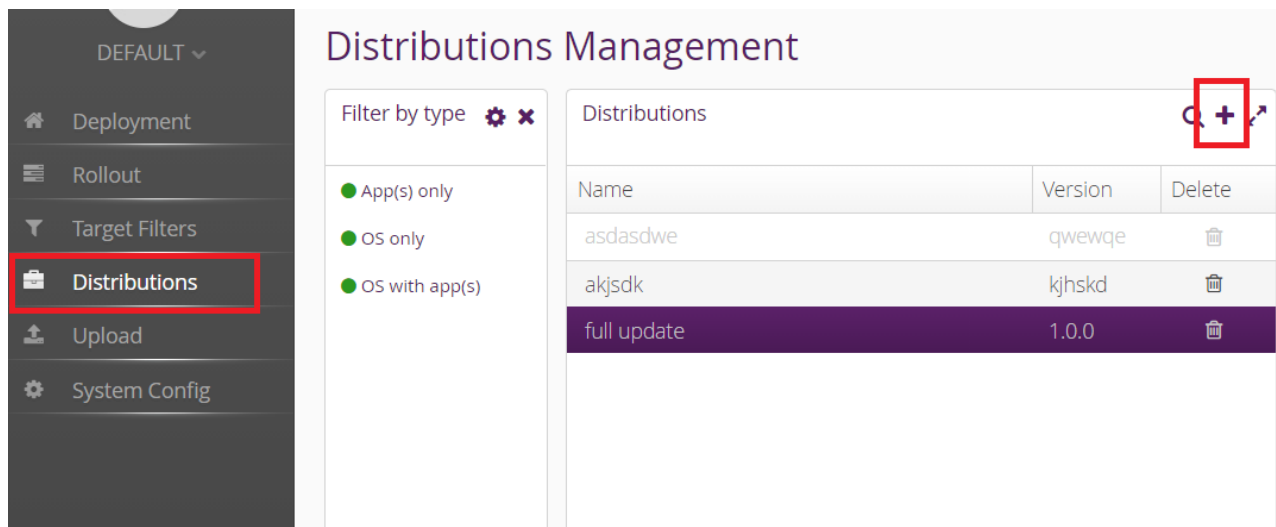
7. swu パッケージをアップロードして Software module に関連付ける

先程作成した Software module を選択して、ハイライトされた状態で、"Upload File"ボタンをクリックするか、ファイルをドラッグアンドドロップしてアップロードします。



8. Distribution を作成して Software module を関連付ける

左側のメニューから"Distribution"をクリックして、Distribution Management の画面に移ります。



The screenshot shows the 'Distributions Management' interface. On the left, a sidebar menu lists 'Deployment', 'Rollout', 'Target Filters', 'Distributions' (highlighted with a red box), 'Upload', and 'System Config'. The main content area is titled 'Distributions Management' and contains a table of distributions. The table has columns for 'Name', 'Version', and 'Delete'. The table contains three rows: 'asdasdwe' with version 'qwewqe', 'akjsdk' with version 'kjhsdk', and 'full update' with version '1.0.0'. A red box highlights the '+' icon in the top right corner of the table, indicating the 'Add' button.

Name	Version	Delete
asdasdwe	qwewqe	
akjsdk	kjhsdk	
full update	1.0.0	

"+" をクリックして Distribution を作成します。type には OS/OSwithApp/Apps、version には任意の文字列を指定します。

### Create new Distribution ✕

Select Type  ▼

Name \*

Version \*

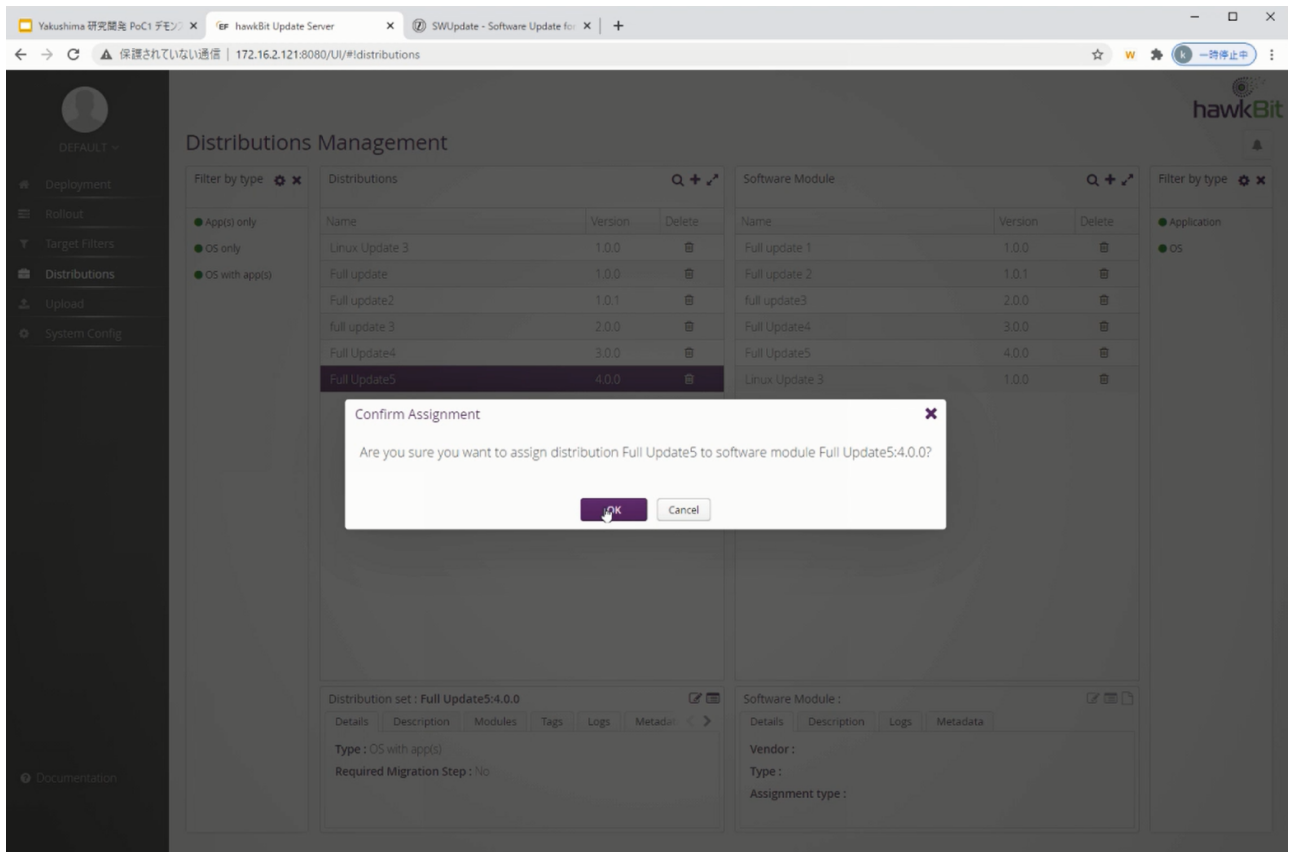
Description

Required Migration Step

\* Mandatory Field

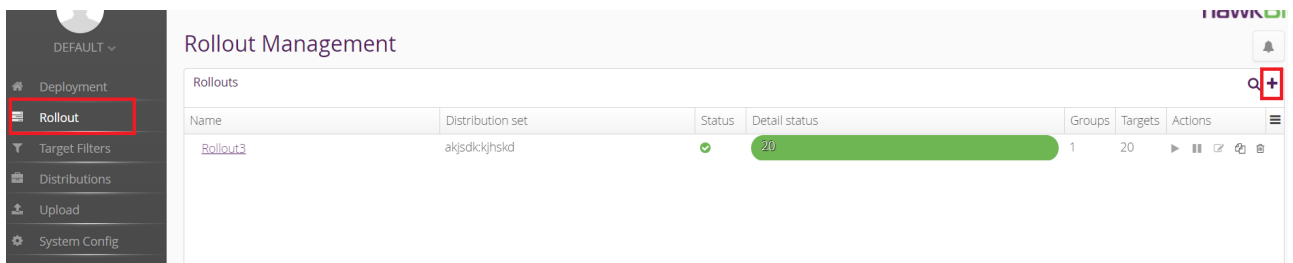
 Save  Cancel

"Software module"のペインから先程作成した Software をドラッグして、作成した Distribution の上にドロップします。



### 9. Rollout を作成してアップデートを開始する

左側のメニューから"Rollout"をクリックして、Rollout Management の画面に移ります。



"+"をクリックして Rollout を作成します。

Create new Rollout ✕

Name \*  \*

Distribution set \*  ▼


Custom Target Filter \*  ▼

Description

Action type \*  ⚡ Forced  ⏸ Soft  ⌚ Time Forced   ⬇️ Download Only

Start type \*  📁 Manual  ▶ Auto  ⌚ Scheduled

Number of Groups  Advanced Group definition



Total Targets : 20  
20 in Group 1

---

Generate the groups automatically with the specified thresholds.

Number of groups \*  Targets per group :20

Trigger threshold \*  %

Error threshold \*   %  Count

\* Mandatory Field

📁 Save ✕ Cancel ?

項目	説明
Name	任意の文字列を設定します。
Distribution Set	先程作成した Distribution を選択します。
Custom Target Filter	先程作成した Target Filter を選択します。
Action Type	アップデート処理をどのように行うかを設定します。 ・ Forced/Soft: 通常のアップデート ・ Time Forced: 指定した時刻までにアップデートする ・ Download only: ダウンロードのみ行う
Start Type	Rollout の実行をどのように始めるかを設定します。 ・ Manual: 後で手動で開始する ・ Auto: Target からのハートビートで開始する ・ Scheduled: 決まった時間から開始する

### 10. アップデートの状態を確認する

Rollout Management の画面の Detail Status で、各 Rollout のアップデートの状態を確認できます。

アップデート中は黄色、アップデートが正常に完了すると緑色になります。



### 10.7.5.1. hawkBit のアップデート管理を CLI で行う

一つのアップデートを登録するには、hawkBit の Web UI で必要な手順が長いので CLI で行うことで効率よく実行できます。

サーバーの設定の段階では、「mkswu」のユーザーを作成する必要があります。作成していない場合は `setup_container.sh --add-user mkswu` で作成してください。

#### 1. hawkbit\_push\_update の実行例

```
[ATDE ~/mkswu]$ ls enable_sshd.swu ❶
enable_sshd.swu

[ATDE ~/mkswu]$ hawkbit_push_update --help
Usage: /usr/bin/hawkbit_push_update [options] file.swu

rollout creation:
  --no-rollout: only upload the file without creating a rollout ❷
  --new: create new rollout even if there already is an existing one ❸
  --failed: Apply rollout only to nodes that previously failed update ❹

post action:
  --start: start rollout immediately after creation ❺

[ATDE ~/mkswu]$ hawkbit_push_update --start enable_sshd.swu ❻
Uploaded (or checked) image extra_os.sshd 1 successfully
Created rollout extra_os.sshd 1 successfully
Started extra_os.sshd 1 successfully
```

- ❶ この例ではあらかじめ作成されている `enable_sshd.swu` を hawkBit に登録します。
- ❷ `--no-rollout` を使う場合に SWU を「distribution」として登録します。デフォルトでは rollout も作成します。テストする際、デバイスがまだ登録されていなければ rollout の段階で失敗します。
- ❸ 同じ SWU で rollout を二回作成した場合にエラーが出ます。もう一度作成する場合は `--new` を使ってください。
- ❹ 一度 rollout をスタートして、Armadillo で失敗した場合には失敗したデバイスだけに対応した rollout を作れます。
- ❺ 作成した rollout をすぐ実行します。このオプションには追加の権限を許可する必要があります。
- ❻ スタートまで行う実行例です。実行結果は Web UI で表示されます。

### 10.7.5.2. SWU で hawkBit を登録する

デバイスが多い場合は、SWU を一度作って armadillo を自己登録させることができます。

サーバーの設定の段階では、「device」のユーザーを作成する必要があります。作成していない場合は `setup_container.sh --add-user device` で作成してください。

#### 1. hawkbit\_register.desc で hawkBit の自己登録を行う例

```
[ATDE ~]$ cd mkswu/
```

```
[ATDE ~/mkswu]$ cp /usr/share/mkswu/examples/hawkbit_register.* . ❶

[ATDE ~/mkswu]$ vi hawkbit_register.sh ❷
# Script configuration: edit this if required!
# user given here must have CREATE_TARGET, READ_TARGET_SECURITY_TOKEN permissions
HAWKBIT_USER=device
HAWKBIT_PASSWORD="CS=wC, zJmrQeeKT.3" ❸
HAWKBIT_URL=https://10.1.1.1 ❹
HAWKBIT_TENANT=default
# set custom options for suricatta block or in general in the config
CUSTOM_SWUPDATE_SURICATTA_CFG="" # e.g. "polldelay = 86400;"
CUSTOM_SWUPDATE_CFG=""
# set to non-empty if server certificate is invalid
SSL_NO_CHECK_CERT=
# or set to cafile that must have been updated first
SSL_CAFILE=
# ... or paste here base64 encoded crt content
SSL_CA_BASE64="
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUJlakNDQVNHZ0F3SUJBZ0lVYTMvYXpNSHZ0
bFFnaFZnZDhIZWhMaEwxNm5Bd0NnWUllb1pJemowRUF3SXcKRXpFuk1BOEdBMVVFQXd3SU1UQXVn
UzR4TGpFd0hoY05Nakl3TWpFNE1EVTFNakV6V2hjTk16SXdNakUyTURVMQpNakV6V2pBVE1SRXdE
d1lEVlFRREBz3hNzR4TGpFdU1UQlNk1HQnlxR1NNNDLBZ0VHQ0NzR1NNNDLB0VICKwSUFc
RFJGcnJV3hHNnBhdWVoejRkRzVqYkVWtm5scHUwYXBHT1c3UUVBPUY4cWp1ZzJWYjk2UHNScWJY
Sk8KbEFdVVo20StaMhk3c1BqeDJHYnhDNms0czFHaLV6QlJNqjBHQTFVZERnUvdCQlJtZzhxL2FV
OURRc3EvTGE1TgpaWFdkTHROUmNEQWZCZ05WSFNRRUdEQVdnQlJtZzhxL2FVOURRc3EvTGE1TlpY
V2RMdE5SY0RBUEJnTlZlUk1CCkFm0EVCVEFEQVFIL01Bb0dDQ3FHU000UJBTUNBMGNBTUVRQ0LB
ZTRCQ0xKREpWZnFTQVdRcVBqNTFmMjJvQkYKRmVBbVlGY2VBMU45dE8rN0FpQXVvUEV1VGFxWjhH
UFYyRUg1UWd0MFRKS05SckJDOEtpNkZwcFlkRUowYWc9PQotLS0tLUVORCBDRVJUSUZJQ0FURS0t
LS0tCg== ❺
"

# ... or add your own options if required
CURLLOPT=-s

: (省略)

[ATDE ~/mkswu]$ cat hawkbit_register.desc ❻
: (省略)
swdesc_script hawkbit_register.sh --version extra_os.hawkbit 1

[ATDE ~/mkswu]$ mkswu hawkbit_register.desc ❼
hawkbit_register.swu を作成しました。

[ATDE ~/mkswu]$ mkswu initial_setup.desc hawkbit_register.desc ❽
hawkbit_register.desc を組み込みました。
initial_setup.swu を作成しました。
```

- ❶ hawkbit\_register.sh と .desc ファイルをカレントディレクトリにコピーします。
- ❷ hawkbit\_register.sh を編集して、設定を記載します。
- ❸ hawkBit の設定の時に入力した「device」ユーザーのパスワードを入力します。この例のパスワードは使用しないでください。
- ❹ hawkBit サーバーの URL を入力します。
- ❺ TLS を使用の場合に、コンテナ作成の時の証明書を base64 で入力します。

- ⑥ hawkbit\_register.desc の中身を確認します。hawkbit\_register.sh を実行するだけです。
- ⑦ SWU を作成して、initial\_setup がすでにインストール済みの Armadillo にインストールできます。
- ⑧ または、initial\_setup.desc と合わせて hawkbit\_register を含んだ initial\_setup.swu を作成します。

## 10.7.6. mkswu の desc ファイル

.desc ファイルを編集すると、いくつかのコマンドが使えます。

例

```
[ATDE ~/mkswu]$ tree /usr/share/mkswu/examples/nginx_start
/usr/share/mkswu/examples/nginx_start
├── etc
│   └── atmark
│       └── containers
│           └── nginx.conf
└──
```

```
[ATDE ~/mkswu]$ cat /usr/share/mkswu/examples/usb_container_nginx.desc
swdesc_option version=1

swdesc_usb_container "nginx_alpine.tar" ①
swdesc_files --extra-os "nginx_start" ②
```

- ① nginx\_alpine.tar ファイルに保存されたコンテナをインストールします。
- ② nginx\_start ディレクトリの中身を転送します。

コマンドは書かれた順番でインストールされます。インストールするかどうかの判断はバージョンで行います:

```
swdesc_option component=<component>
swdesc_option version=<version>
か
swdesc_xxx --version <component> <version> [options]
```

・ <component>は以下のどれかにしてください (デフォルトでは .desc ファイルのファイル名を使います)

1. base\_os: rootfs (Armadillo Base OS)を最初から書き込む時に使います。現在のファイルシステムは保存されていない。

この場合、/etc/swupdate\_preserve\_files に載ってるファイルのみをコピーして新しい base OS を展開します。

この component がないと現在の rootfs のすべてがコピーされます。

2. extra\_os.<文字列>: rootfs の変更を行う時に使います。<文字列> には任意の文字列を指定します。

rootfs を変更を行う時に使います。swdesc\_\* コマンドに --extra-os オプションを追加すると、component に自動的に extra\_os. を足します。

3. <文字列> (コンテナの名前などの任意の文字列) : rootfs の変更がないときに使います。

この component を使うと rootfs の変更ができませんのでご注意ください。

- ・ アップデートを行う際にこのバージョンと現在のバージョンを比べてアップデートの判断を行います。

<component> がまだインストールされてなかった時や <version> が上がる時にインストールします。

デフォルトではダウングレードはできませんが、 --install-if=different オプションを追加することで <version> が変わる際にインストール可能になります。

アップデートの一部をインストールすることもありますので、複数の component で管理し、いくつかの古いバージョンに対応するアップデートも作成可能です。

以下のコマンドから使ってください

- ・ swdesc\_tar と swdesc\_files でファイルを転送します。

```
swdesc_tar [--dest <dest>] <tar_file>
swdesc_files [--dest <dest>] [--basedir <basedir>] ¥
<file> [<more files>]
```

swdesc\_tar の場合、予め用意されてある tar アーカイブをこのままデバイスで展開します。

--dest <dest> で展開先を選ぶことができます。デフォルトは / (--extra-os を含め、バージョンの component は base\_os か extra\_os.\* の場合) か /var/app/rollback/volumes/ (それ以外の component)。後者の場合は /var/app/volumes と /var/app/rollback/volumes 以外は書けないので必要な場合に --extra-os を使ってください。

swdesc\_files の場合、mkswu がアーカイブを作ってくれますが同じ仕組みです。

--basedir <basedir> でアーカイブ内のパスをどこで切るかを決めます。

- ・ 例えば、swdesc\_files --extra-os --basedir /dir /dir/subdir/file ではデバイスに /subdir/file を作成します。
- ・ デフォルトは <file> から設定されます。ディレクトリであればそのまま basedir として使います。それ以外であれば親ディレクトリを使います。
- ・ swdesc\_command や swdesc\_script でコマンドを実行する

```
swdesc_command <command> [<more commands>]
swdesc_script <script>
```

アップデート先の環境でコマンドやスクリプトファイルを走らせます。

バージョンの component は base\_os と extra\_os 以外の場合、 /var/app/volumes と /var/app/rollback/volumes 以外は変更できないのでご注意ください。

コマンドの実行が失敗した場合、アップデートも失敗します。

- ・ swdesc\_exec でファイルを配ってコマンドでそのファイルを使う

```
swdesc_exec <file> <command>
```

swdesc\_command と同じくコマンドを走らせますが、<file> を先に転送してコマンド内で"\$1"として使えます。

- swdesc\_command\_nochroot, swdesc\_script\_nochroot, swdesc\_exec\_nochroot で起動中のシステム上でコマンドを実行します。

このコマンドは nochroot なしのバージョンと同じ使い方で、現在起動中のシステムに変更や確認が必要な場合にのみ使用してください。



nochroot コマンドは確認を一切しないため、Armadillo が起動できない状態になる可能性もあります。充分にご注意ください。

例が必要な場合は /usr/share/mkswu/examples/firmware\_update.desc を参考にしてください。

- swdesc\_embed\_container, swdesc\_usb\_container, swdesc\_pull\_container で予め作成したコンテナを転送します。

```
swdesc_embed_container <container_archive>
swdesc_usb_container <container_archive>
swdesc_pull_container <container_url>
```

例は「10.3.5. コンテナの配布」を参考にしてください。

- swdesc\_boot で u-boot を更新します。

```
swdesc_boot <boot image>
```

このコマンドだけにバージョンは自動的に設定されます。

コマンドの他には、設定変数もあります。以下の設定は /home/atmark/mkswu/mkswu.conf に設定できます。

- DESCRIPTION="<text>": イメージの説明、ログに残ります。
- PRIVKEY=<path>, PUBKEY=<path>: 署名鍵と証明書
- PRIVKEY\_PASS=<val>: 鍵のパスワード（自動用）

openssl の Pass Phrase をそのまま使いますので、pass:password, env:var や file:pathname のどれかを使えます。pass や env の場合他のプロセスに見られる恐れがありますので file をおすすめします。

- ENCRYPT\_KEYFILE=<path>: 暗号化の鍵

以下のオプションも `mkswu.conf` に設定できますが、`.desc` ファイルにも設定可能です。`swdesc_option` で指定することで、誤った使い方した場合 `mkswu` の段階でエラーを出力しますので、必要な場合は使用してください。

- `swdesc_option CONTAINER_CLEAR`: インストールされたあるコンテナと `/etc/atmark/containers/*.conf` をすべて削除します。

このオプションは簡単な初期化と考えてください。通常の運用では、不要になったイメージは自動的に削除されますのでこのオプションを設定する必要はありません。

以下のオプションは Armadillo 上の `/etc/atmark/baseos.conf` に、例えば `MKSWU_POST_ACTION=xxx` として設定することができます。

その場合に `swu` に設定されなければ `/etc` の設定で実行されますので、アットマークテクノが用意している Base OS のアップデートでも動作の変更は可能です。`swu` に特定のオプションが設定された場合は設定されたオプションが優先されますので、一時的な変更も可能です。

- `swdesc_option POST_ACTION=container`: コンテナのみのアップデート後に再起動を行いません。コンテナの中身だけをアップデートする場合、Armadillo-640 を再起動せずにコンテナだけを再起動させます。
- `swdesc_option POST_ACTION=poweroff`: アップデート後にシャットダウンを行います。
- `swdesc_option POST_ACTION=wait`: アップデート後に自動的に再起動は行われず、次回起動時にアップデートが適用されます。
- `swdesc_option POST_ACTION=reboot`: デフォルトの状態に戻します。アップデートの後に再起動します。
- `swdesc_option NOTIFY_STARTING_CMD="command"`, `swdesc_option NOTIFY_SUCCESS_CMD="command"`, `swdesc_option NOTIFY_FAIL_CMD="command"`: アップデートをインストール中、成功した場合と失敗した場合に実行されるコマンドです。

コマンドを実行する事で、アプリケーションやユーザーにアップデートを知らせることができます。

LED で知らせる例を `/usr/share/mkswu/examples/enable_notify_led.desc` に用意してあります。

### 10.7.6.1. 例: sshd を有効にする

`/usr/share/mkswu/examples/enable_sshd.desc` を参考にします。

`desc` ファイルを編集する必要がありませんが自分の公開鍵を指定された場所に配置してください。

```
[ATDE ~/mkswu]$ cp -r /usr/share/mkswu/examples/enable_sshd* .
[ATDE ~/mkswu]$ cat enable_sshd.desc
swdesc_option component=extra_os.sshd version=1

# add your public key in enable_sshd/root/.ssh/authorized_keys
if [ -z "$SWDESC_TEST" ]; then
    grep -qE '^ssh-' enable_sshd/root/.ssh/authorized_keys ¥
    || error "Add your keys in enable_sshd/root/.ssh/authorized_keys"
fi
swdesc_files --dest /root enable_sshd/root ❶

swdesc_command "ssh-keygen -A" ¥ ❷
```

```

"rc-update add sshd" ❸
[ATDE ~/mkswu]$ cp ~/.ssh/id_rsa.pub ¥
enable_sshd/root/.ssh/authorized_keys ❹
[ATDE ~/mkswu]$ mkswu enable_sshd.desc ❺
Enter pass phrase for /home/atmark/mkswu/swupdate.key:
enable_sshd.swu を作成しました。

```

- ❶ 自分の公開鍵を転送します。デフォルトのオプションなので enable\_sshd/root ディレクトリの中身をそのまま /root に転送されます。
- ❷ 再起動する度に新しいサーバーの鍵が変わらないように、アップデートの時に一回作成します。
- ❸ サービスを有効にします。
- ❹ 自分の公開鍵を指定された場所に配置します。
- ❺ イメージを作成します。パスワードは証明鍵のパスワードです。

### 10.7.6.2. 例: Armadillo Base OS アップデート

ここでは、「10.4. Armadillo のソフトウェアをビルドする」でメインシステム向けのビルドで作成したファイルを使用します。

/usr/share/mkswu/examples/OS\_update.desc を参考にします。

```

[ATDE ~/mkswu]$ cp /usr/share/mkswu/examples/OS_update.desc update-[VERSION].desc
[ATDE ~/mkswu]$ vi update-[VERSION].desc
# uboot image can be generated with atmark u-boot script
swdesc_uboot u-boot-dtb.imx ❶

# base OS is a tar that will be extracted on a blank filesystem,
# after copying just a few key config files.
#
# OS updates are only installed if version is greater than previous update
# so if you install your own updates atmark-techno provided Armadillo Base OS
# updates might not get installed
swdesc_tar "baseos-640-[VERSION].tar.zst" ¥ ❷
--version base_os [VERSION] ❸
[ATDE ~/mkswu]$ mkswu update-[VERSION].desc ❹
Enter pass phrase for /home/atmark/mkswu/swupdate.key:
update-[VERSION].swu を作成しました。

```

- ❶ 「10.4.1. ブートローダーをビルドする」でビルドしたイメージを使います。
- ❷ build-rootfs でビルドしたイメージを使います。
- ❸ バージョンが上がるときにしかインストールされませんので、現在の/etc/sw-versionsを確認して適切に設定してください。
- ❹ イメージを作成します。パスワードは証明鍵の時のパスワードです。

### 10.7.6.3. 例: swupdate\_preserve\_files で Linux カーネル以外の Armadillo-640 向けのイメージをインストールする方法

Armadillo-640 向けのアップデートイメージに Linux カーネルが含まれています。

swupdate\_preserve\_files を使って、以下のコマンドでインストール後に現在のカーネルをコピーして更新させないようにします。

```
[armadillo ~]# echo 'POST /boot' >> /etc/swupdate_preserve_files
[armadillo ~]# echo 'POST /lib/modules' >> /etc/swupdate_preserve_files ❶
[armadillo ~]# persist_file /etc/swupdate_preserve_files ❷
```

- ❶ swupdate\_preserve\_files に /boot と /lib/modules を保存するように追加します。
- ❷ 変更した設定ファイルを保存します



/usr/share/mkswu/examples/kernel\_update\*.desc のように update\_preserve\_files.sh のヘルパーで、パスを自動的に /etc/swupdate\_preserve\_files に追加することができます。

```
[ATDE ~/mkswu]$ cat example.desc
swdesc_script "$SCRIPT_DIR/examples/update_preserve_files.sh" -- ¥ ❶
"POST /boot" ¥
"POST /lib/modules"
```

- ❶ スクリプトの内容を確認する場合は /usr/share/mkswu/examples/update\_preserve\_files.sh を参照してください。



Armadillo Base OS のカーネルを再び使用したい場合は同じスクリプトの --del オプションで行を削除することができます。

```
[ATDE ~/mkswu]$ cat example.desc
swdesc_script "$SCRIPT_DIR/examples/update_preserve_files.sh" -- ¥
--del "POST /boot" "POST /lib/modules"
```

### 10.7.7. swupdate\_preserve\_files について

extra\_os のアップデートで rootfs にファイルを配置することができますが、次の OS アップデートの際に削除される可能性があります。

デフォルトでは、/etc/atmark と、swupdate、sshd やネットワークの設定を保存しますがそれ以外はコピーされてません。

そうでないファイルを更新する際には /etc/swupdate\_preserve\_files に記載します。「10.7.6.3. 例: swupdate\_preserve\_files で Linux カーネル以外の Armadillo-640 向けのイメージをインストールする方法」を参考にしてください。

コピーのタイミングによって、以下のどれかを使用してください：

1. 単にファイルを記載する



この場合、アップデートする前にファイルをコピーします。baseos のイメージと同じ swu にアップデートしたいファイルを記載していても、このファイルが Armadillo Base OS に含まれないのであれば問題なくアップデートできます。

例: `echo "/root/.profile" >> /etc/swupdate_preserve_files`

## 2. POST のキーワードの後に記載する

この場合、アップデートの最後でコピーします。Armadillo Base OS に含まれてるファイルであれば、インストール前にコピーしても保存されないのでコピーのタイミングをずらします。

そのコピーが最後に行われるので、同じアップデートでファイルの変更ができません。アップデートを別けて、baseos のイメージをインストールしてからこのファイルを更新することができます。

例: `echo "POST /etc/conf.d/podman-atmark" >> /etc/swupdate_preserve_files`

## 10.7.8. SWU イメージの内容の確認

`mkswu --show [file.swu]` で SWU イメージの内容を確認することができます。

出力は desc ファイルに似ていますが、そのまま desc ファイルとして利用できませんので確認用としてお使いください。

```
[ATDE ~/mkswu]$ mkswu --show enable_sshd.swu
enable_sshd.swu

# built with mkswu 4.1

swdesc_files --dest /root enable_sshd/root
--version extra_os.sshd 1
(encrypted)

swdesc_command ssh-keygen -A && rc-update add sshd default
--version extra_os.sshd 1
```

## 10.7.9. SWUpdate と暗号化について

`mkswu --init` の時に暗号化を有効にする場合は AES でファイルを暗号化します。

現在使われてる SWUpdate の暗号化はコマンドやメタデータを含む sw-description ファイルは暗号化されてません。そのため、通信の暗号化 (HTTPS で送信するなど) を使うことを推奨します。

## 10.8. Armadillo Base OS の操作

Armadillo Base OS は Alpine Linux をベースとして作られています。

このセクションでは Armadillo Base OS の機能を紹介します。

### 10.8.1. アップデート

Armadillo Base OS は SWUpdate によってアップデートすることができます。

アップデートする際には、rootfs ファイルシステムにインストールされたファイルをすべて消して、アップデートの中身と /etc/swupdate\_preserve\_files に記載されているファイルで新しい rootfs を作ります。「10.7.7. swupdate\_preserve\_files について」を参照してください。

アップデートでファイルを削除してしまった場合に `abos-ctrl mount-old` で前のシステムを read-only でマウントして、削除されたファイルをコピーすることもできます。

## 10.8.2. overlaysfs と persist\_file について

Armadillo BaseOS ではルートファイルシステムに overlaysfs を採用しています。

そのため、ファイルを変更した後 Armadillo の電源を切ると変更内容は保持されません。開発中などに rootfs の変更内容を保持するには、変更したファイルに対して `persist_file` コマンドを使用します。

開発以外の時は安全のため、ソフトウェアアップデートによる更新を実行してください。アップデート手順に関しては「10.7. Armadillo のソフトウェアをアップデートする」を参照してください。

rootfs の内容を変更しても、ソフトウェアアップデートを実施した際に変更した内容が保持されない可能性があります。ソフトウェアアップデート実施後も変更内容を保持する手順に関しては「10.7.7. swupdate\_preserve\_files について」を参照してください。

`persist_file` コマンドの概要を「[図 10.125. persist\\_file のヘルプ](#)」に示します。

```
[armadillo ~]# persist_file -h
Usage: /usr/bin/persist_file [options] file [more files...]

Mode selection:
  (none) single entry copy
  -d, --delete    delete file
  -l, --list      list content of overlay
  -a, --apk       apk mode: pass any argument after that to apk on rootfs
  -R, --revert    revert change: only delete from overlay, making it
                  look like the file was reverted back to original state

Copy options:
  -r, --recurse  recursive copy (note this also removes files!)
  -p, --preserve make the copy persist through baseos upgrade
                  by adding entries to /etc/swupdate_preserve_files
  -P, --preserve-post same, but copy after upgrade (POST)

Delete options:
  -r, --recurse  recursively delete files

Common options:
  -v, --verbose  verbose mode for all underlying commands

Note this directly manipulates overlaysfs lower directories
so might need a reboot to take effect
```

図 10.125 `persist_file` のヘルプ

### 1. ファイルの保存・削除手順例

```
[armadillo ~]# echo test > test
[armadillo ~]# persist_file -rv /root
```

```

'/root/test' -> '/mnt/root/test' ❶
'/root/.ash_history' -> '/mnt/root/.ash_history'
[armadillo ~]# rm -f test
[armadillo ~]# persist_file -rv /root
removed '/mnt/root/test' ❷
removed '/mnt/root/.ash_history' ❸
'/root/.ash_history' -> '/mnt/root/.ash_history'

```

図 10.126 persist\_file 保存・削除手順例

- ❶ 追加・変更したファイルを rootfs へコピーします。
- ❷ -r を指定すると、ひとつ前の rm -f コマンドで削除したファイルが rootfs からも削除されますのでご注意ください。
- ❸ すでに rootfs に存在するファイルも一度削除してからコピーするため、このようなメッセージが表示されます。

## 2. ソフトウェアアップデート後も変更を維持する手順例

```

[armadillo ~]# vi /etc/conf.d/podman-atmark ❶
[armadillo ~]# persist_file -P /etc/conf.d/podman-atmark ❷
[armadillo ~]# tail -n 2 /etc/swupdate_preserve_files ❸
# persist_file 20211216
POST /etc/conf.d/podman-atmark

```

図 10.127 persist\_file ソフトウェアアップデート後も変更を維持する手順例

- ❶ 何らかのファイルの内容を変更します。
- ❷ -P オプションを付与して persist\_file を実行します。
- ❸ swupdate\_preserve\_files に追加されたことを確認します。

## 3. 変更ファイルの一覧表示例

```

[armadillo ~]# mkdir dir
[armadillo ~]# persist_file -l
directory          /
directory          /root
opaque directory   /root/dir ❶
whiteout           /root/test ❷
regular file       /root/.ash_history
directory          /etc
regular file       /etc/resolv.conf
directory          /var
symbolic link     /var/lock
: (省略)

```

図 10.128 persist\_file 変更ファイルの一覧表示例

- ❶ rootfs のファイルを見せないディレクトリは opaque directory と表示されます。

- ② 削除したファイルは whiteout と表示されます。
4. パッケージをインストールする時は apk コマンドを使用してメモリ上にインストールできますが、persist\_file コマンドで rootfs に直接インストールすることも可能です。

```
[armadillo ~]# persist_file -a add strace
(1/3) Installing fts (1.2.7-r1)
(2/3) Installing libelf (0.185-r0)
(3/3) Installing strace (5.14-r0)
Executing busybox-1.34.1-r3.trigger
OK: 251 MiB in 188 packages
Install succeeded, but might not work in the running system
Please reboot if installed program does not work ❶
[armadillo ~]# strace ls
: (省略)
exit_group(0) = ?
+++ exited with 0 +++
```

図 10.129 persist\_file でのパッケージインストール手順例

- ❶ この例では Armadillo を再起動せずにインストールしたコマンドを使用できましたが、Armadillo の再起動が必要となるパッケージもありますので、その場合は Armadillo を再起動してください。

### 10.8.3. ロールバック状態の確認

Armadillo Base OS の ルートファイルシステムが壊れて起動できなくなった場合に自動的に前のバージョンで再起動します。

自分で確認する必要がある場合に abos-ctrl status でロールバックされてるかどうかの確認ができます。

必要な場合（例えば、自分のアプリケーションがアップデート直後に問題があった場合）、abos-ctrl rollback で手動のロールバックも可能です。ロールバックにエラーがなければ、再起動してロールバックを完了します。

なお、/var/at-log/atlog に切り替えの際に必ずログを書きますので、調査の時に使ってください。

```
[armadillo ~]# cat /var/at-log/atlog
Mar 17 14:51:35 armadillo NOTICE swupdate: Installed update to /dev/mmcblk0p2: ¥
extra_os.sshd: unset -> 1, extra_os.initial_setup: unset -> 1
Mar 17 16:48:52 armadillo NOTICE swupdate: Installed update to /dev/mmcblk0p1: ¥
boot: 2020.04-at5 -> 2020.04-at6, base_os: 3.15.0-at.3 -> 3.15.0-at.4
Mar 17 17:42:15 armadillo NOTICE swupdate: Installed update to /dev/mmcblk0p2: ¥
other_boot: 2020.04-at5 -> 2020.04-at6, container: unset -> 1, extra_os.container: unset -> 1
```

図 10.130 /var/at-log/atlog の内容の例

### 10.8.4. ボタンやキーを扱う

自分のアプリケーションで直接入力の処理ができない場合に Base OS から簡単な処理ができます。

buttd サービスで指定されたイベントでコマンドを実行します。

/etc/atmark/buttond.conf に BUTTOND\_ARGS を指定することで、動作を指定することができます:

- ・ -s <key> -a "command" : 短押しの設定。キーを 1 秒以内に離せば短押しと認識し "command" を実行します。認識する時間は -t <time\_ms> オプションで変更可能です。
- ・ -l <key> -s "command" : 長押しの設定。キーを 5 秒押し続けたタイミングで "command" を実行します。長押しと認識する時間は -t <time\_ms> オプションで変更可能です。
- ・ 1 つのキーに対して複数の設定が可能です。長押しの設定が複数ある場合、押したままの状態だと一番長い時間に設定されている "command" を実行します。途中でキーを離れた場合は、キーを離れた時間に応じた "command" を実行します。(例: buttond -s <key> -a "cmd1" -l <key> -t 2000 -a "cmd2" -l <key> -t 10000 -a "cmd3" <file> を実行した場合、1 秒以内に離すと "cmd1"、2 秒以上 10 秒以内に離すと "cmd2"、10 秒を越えたら "cmd3" を実行します)。
- ・ 短押し設定を複数指定する場合、時間の短い設定を先に指定してください。0.5 秒、1 秒を設定したい場合、1 秒 → 0.5 秒の順番で指定すると 0.5 秒が無視されます。

以下にデフォルトを維持したままで SW1 の早押しと長押しにそれぞれの場合にコマンドを実行させます。

```
[armadillo ~]# vi /etc/atmark/buttond.conf ❶
BUTTOND_ARGS="$BUTTOND_ARGS -s prog1 -a 'date >> /tmp/shortpress'"
BUTTOND_ARGS="$BUTTOND_ARGS -l prog1 -t 5000 -a 'date >> /tmp/longpress'"
[armadillo ~]# persist_file /etc/atmark/buttond.conf ❷
[armadillo ~]# rc-service buttond restart ❸
buttond          | * Stopping button watching daemon ...           [ ok ]
buttond          | * Starting button watching daemon ...           [ ok ]
[armadillo ~]# cat /tmp/shortpress ❹
Tue Mar 22 17:16:42 JST 2022
Tue Mar 22 17:16:43 JST 2022
[armadillo ~]# cat /tmp/longpress
Tue Mar 22 17:16:48 JST 2022
```

### 図 10.131 buttond で SW1 を扱う

- ❶ buttond の設定ファイルを編集します。この例では、短押しの場合 /tmp/shotpress に、5 秒以上の長押しの場合 /tmp/longpress に日付を出力します。
- ❷ 設定ファイルを保存します。
- ❸ buttond サービスを再起動させます。ここでは再起動後短押しを 2 回、長押しを 1 回行ったとします。
- ❹ 押された回数を確認します。

USB キーボードや他の入力デバイスにも対応できます。

1. デバイスを接続してから、buttond でデバイス名とキーコードを確認します。

```
[armadillo ~]# buttond -vvv /dev/input/* /dev/input/by-*/ * ❶
Skipping directory /dev/input/by-id
Skipping directory /dev/input/by-path
[78972.042] /dev/input/event2 4 4 458976: non-keyboard event ignored
[78972.042] /dev/input/event2 LEFTCTRL (29) pressed: ignored ❷
```

```
[78972.042] /dev/input/by-id/usb-0566_3029-event-kbd 4 4 458976: non-keyboard event ignored
[78972.042] /dev/input/by-id/usb-0566_3029-event-kbd LEFTCTRL (29) pressed: ignored
[78972.042] /dev/input/by-path/platform-xhci-hcd.1.auto-usb-0:1:1.0-event-kbd 4 4 458976:
non-keyboard event ignored
[78972.042] /dev/input/by-path/platform-xhci-hcd.1.auto-usb-0:1:1.0-event-kbd LEFTCTRL
(29) pressed: ignored
[78972.130] /dev/input/event2 4 4 458976: non-keyboard event ignored
[78972.130] /dev/input/event2 LEFTCTRL (29) released: ignored
[78972.130] /dev/input/by-id/usb-0566_3029-event-kbd 4 4 458976: non-keyboard event ignored
[78972.130] /dev/input/by-id/usb-0566_3029-event-kbd LEFTCTRL (29) released: ignored
[78972.130] /dev/input/by-path/platform-xhci-hcd.1.auto-usb-0:1:1.0-event-kbd 4 4 458976:
non-keyboard event ignored
[78972.130] /dev/input/by-path/platform-xhci-hcd.1.auto-usb-0:1:1.0-event-kbd LEFTCTRL
(29) released: ignored
```

- ❶ buttdond を -vvv で冗長出力にして、すべてのデバイスを指定します。
  - ❷ 希望のキーを押すと、LEFTCTRL が三つのパスで認識されました。一番安定する by-id のパスを控えておきます。
2. USB デバイスを外すこともありますので、-i (inotify) で管理されてる入力デバイスとして追加します。そうしないとデバイスを外したときに buttdond が停止します。

```
[armadillo ~]# vi /etc/atmark/buttnd.conf
BUTTND_ARGS="$BUTTND_ARGS -i /dev/input/by-id/usb-0566_3029-event-kbd"
BUTTND_ARGS="$BUTTND_ARGS -s LEFTCTRL -a 'podman_start button_pressed_container'"
[armadillo ~]# persist_file /etc/atmark/buttnd.conf
[armadillo ~]# rc-service buttnd restart
```

### 10.8.5. Armadillo Base OS 側の起動スクリプト

起動時に何かスクリプトを走らせるためにはコンテナとして実行することを推奨します。「10.3.1. コンテナの自動起動」を参照してください。

コンテナで実行不可能な場合に、「local」サービスを使うことができます:/etc/local.d ディレクトリに.start ファイルを置いておくと起動時に実行されて、.stop ファイルは終了時に実行されます。

```
[armadillo ~]# vi /etc/local.d/date_test.start ❶
#!/bin/sh

date > /tmp/boottest
[armadillo ~]# chmod +x /etc/local.d/date_test.start ❷
[armadillo ~]# persist_file /etc/local.d/date_test.start ❸
[armadillo ~]# reboot
: (省略)
[armadillo ~]# cat /tmp/boottest ❹
Tue Mar 22 16:36:12 JST 2022
```

図 10.132 local サービスの実行例

- ❶ スクリプトを作ります。
- ❷ スクリプトを実行可能にします。

- ③ スクリプトを保存して、再起動します。
- ④ 実行されたことを確認します。

### 10.8.6. u-boot の環境変数の設定

u-boot の環境変数を変更するには /boot/uboot\_env.d/ ディレクトリに環境変数が書かれた設定ファイルを配置します。

ファイルの構文は fw\_setenv が扱うことができるもので、以下のとおりです：

- ・ # で始まる行はコメントと扱われる為、無視されます。また、環境変数への代入を示す = がない場合も無視されます。
- ・ [変数]=[値] で変数を設定します。スペースや引用符を含め他の文字は有効ですので、変数の名前と値に不要な文字を入れないように注意してください。
- ・ [変数]= で変数を消します。値がない場合に変数が消去されます。

このファイルによるアップデート内容は swupdate でアップデートする際に適用されます。

実行中のシステムに影響がありませんので、設定ファイルを swupdate で転送しない場合はファイル永続化後に fw\_setenv -s /boot/uboot\_env.d/[ファイル名] で変数を書き込んでください。

swupdate でファイルを転送した場合には、変数はすぐに利用されます。

```
[armadillo ~]# vi /boot/uboot_env.d/no_prompt ①
# bootdelay を -2 に設定することで u-boot のプロンプトを無効化します
bootdelay=-2
[armadillo ~]# persist_file -v /boot/uboot_env.d/no_prompt ②
'/boot/uboot_env.d/no_prompt' -> '/mnt/boot/uboot_env.d/no_prompt'
[armadillo ~]# fw_setenv -s /boot/uboot_env.d/no_prompt ③
Environment OK, copy 0
[armadillo ~]# fw_printenv | grep bootdelay ④
bootdelay=-2
```

図 10.133 uboot\_env.d のコンフィグファイルの例


- ① コンフィグファイルを生成します。
- ② ファイルを永続化します。
- ③ 変数を書き込みます。
- ④ 書き込んだ変数を確認します。



mkswu バージョン 4.4 以降が必要です。必要な場合はアップデートしてください。

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

書き方は、 /usr/share/mkswu/examples/uboot\_env.desc を参考にしてください。



「10.4.1. ブートローダーをビルドする」の際に u-boot のデフォルトを変更した場合や、u-boot のプロンプトで「setenv」や「saveenv」を実行しても、/boot/uboot\_env.d/00\_defaults によって変更がアップデートの際にリセットされます。

00\_defaults のファイルは Base OS の一部で更新されることもありますので、変更を望む場合は別のファイルを作って設定してください。ファイルはアルファベット順で処理されます。00\_defaults にある変数を後のファイルにも設定した場合はそのファイルの値だけが残ります。

主要な u-boot の環境変数を以下に示します。

表 10.6 u-boot の主要な環境変数

環境変数	説明	デフォルト値
console	U-Boot と Linux で利用するコンソールのデバイスノードと、UART のボーレート等を指定します。	ttymxc0,115200
bootcount	起動回数を示します。初回起動時に 1 となり、起動に失敗する度にインクリメントされます。ユーザーランドの bootcount サービスが起動されると、この値はクリアされます。この値が"bootlimit"を越えた場合はロールバックします。ロールバックの詳細については、「13.1.4. ロールバック (リカバリー)」を参照してください。	1
bootlimit	"bootcount"のロールバックを行うしきい値を指定します。	3
bootdelay	保守モードに遷移するためのキー入力を待つ時間を指定します(単位: 秒)。次の値は特別な意味を持ちます。 <ul style="list-style-type: none"> <li>・ -1: キー入力の有無に関らず保守モードに遷移します。</li> <li>・ -2: キー入力の有無に関らず保守モードに遷移しません。</li> </ul>	2
image	Linux カーネルイメージファイルのパスです。"mmcdev"で指定されたデバイスの、"mmpart"で指定されたパーティションのルートディレクトリからの相対パスで指定します。	boot/Image
fdt_file	DTB ファイルのパスです。"mmcdev"で指定されたデバイスの、"mmpart"で指定されたパーティションのルートディレクトリからの相対パスで指定します。	boot/armadillo.dtb
overlays_list	DT overlay の設定ファイルのパスです。"mmcdev"で指定されたデバイスの、"mmpart"で指定されたパーティションのルートディレクトリからの相対パスで指定します。DT overlay の詳細については、「10.9.4. DT overlay によるカスタマイズ」を参照してください。	boot/overlays.txt
mmcautodetect	mmc デバイスの自動検出機能の有効/無効を指定します。yes を指定した場合のみ、u-boot が起動された mmc デバイスが自動的に mmcdev として利用されます。	yes
mmcdev	"image"や"fdt_file"で指定されたファイルが配置してある mmc デバイスのインデックスを指定します。インデックスと mmc デバイスの対応は次の通りです。 <ul style="list-style-type: none"> <li>・ 0: eMMC</li> <li>・ 1: microSD/microSDHC/microSDXC カード</li> </ul> "mmcautodetect"に yes が指定されている場合は、u-boot の起動時に上書きされます。	2



環境変数	説明	デフォルト値
mmcpart	"image"や"fdt_file"で指定されたファイルが配置してある、"mmcdev"で指定された mmc デバイスのパーティション番号を指定します。"mmcautoload"に yes が指定されている場合は、u-boot の起動時に上書きされます。	1
mmcroot	ルートファイルシステムが配置されているデバイスノードと、マウントオプションを指定します。"mmcautoload"に yes が指定されている場合は、u-boot の起動時に上書きされます。overlayfs が正しく機能しなくなる場合があるので、ro の指定は変更しないでください。	/dev/mmcblk0p1 rootwait ro
optargs	Linux カーネル起動時パラメータを指定します。"quiet"を削除すると、コンソールに起動ログが出力されるようになりますが、起動時間が長くなります。nokaslr を削除すると、KASLR(Kernel Address Space Layout Randomization)が有効となり、Linux カーネルの仮想アドレス空間がランダム化されます。	quiet
loadaddr	Linux カーネルが RAM にロードされる物理アドレスを指定します。	0x80800000
fdt_addr	DTB が RAM にロードされる物理アドレスを指定します。	0x83500000
overlay_addr	DT overlay のワーク領域として利用される RAM の物理アドレスを指定します。	0x83520000

### 10.8.7. Network Time Protocol (NTP, ネットワーク・タイム・プロトコル)

Armadillo Base OS では chronyd を使っています。

デフォルトの設定（使用するサーバーなど）は /etc/chrony/conf.d/ にあり、変更用に /etc/atmark/chrony.conf.d/ のファイルも読み込みます。/etc/atmark/chrony.conf.d ディレクトリに /etc/chrony/conf.d/ と同じファイル名の設定ファイルを置いておくことで、デフォルトのファイルを読まないようになります。

例えば、NTP サーバーの設定は servers.conf に記載されてますので、変更する際には /etc/atmark/chrony.conf.d/servers.conf のファイルに記載します：

```
[armadillo ~]# vi /etc/atmark/chrony.conf.d/servers.conf ❶
pool my.ntp.server iburst
[armadillo ~]# persist_file /etc/atmark/chrony.conf.d/servers.conf ❷
[armadillo ~]# rc-service chronyd restart ❸
chronyd          | * Stopping chronyd ... [ ok ]
chronyd          | * Starting chronyd ... [ ok ]
armadillo:~# chronyc sources ❹
MS Name/IP address          Stratum Poll Reach LastRx Last sample
=====
^? my.ntp.server            1      6      3      2    +88ms[ +88ms] +/- 173ms
```

図 10.134 chronyd のコンフィグの変更例

- ❶ コンフィグファイルを作ります。
- ❷ ファイルを保存します
- ❸ chronyd サービスを再起動します。
- ❹ chronyc で新しいサーバーが使用されていることを確認します。

## 10.9. Device Tree をカスタマイズする

at-dtweb を利用して Device Tree をカスタマイズする方法を説明します。at-dtweb では、Web ブラウザ上のマウス操作で dtbo ファイルおよび desc ファイルを生成することができます。カスタマイズの対象は拡張インターフェース(CON9/CON14)および LCD 拡張インターフェース(CON11)です。

## 10.9.1. at-dtweb のインストール

ATDE9 に at-dtweb パッケージをインストールします。

```
[ATDE ~]$ sudo apt update
[ATDE ~]$ sudo apt install at-dtweb
```

インストール済みの場合は、以下のコマンドを実行し最新版への更新を行ってください。

```
[ATDE ~]$ sudo apt update
[ATDE ~]$ sudo apt upgrade
```

## 10.9.2. at-dtweb の起動

### 1. at-dtweb の起動開始

at-dtweb の起動を開始するには、デスクトップ左上のアプリケーションの「システムツール」から「at-dtweb」を選択してください。

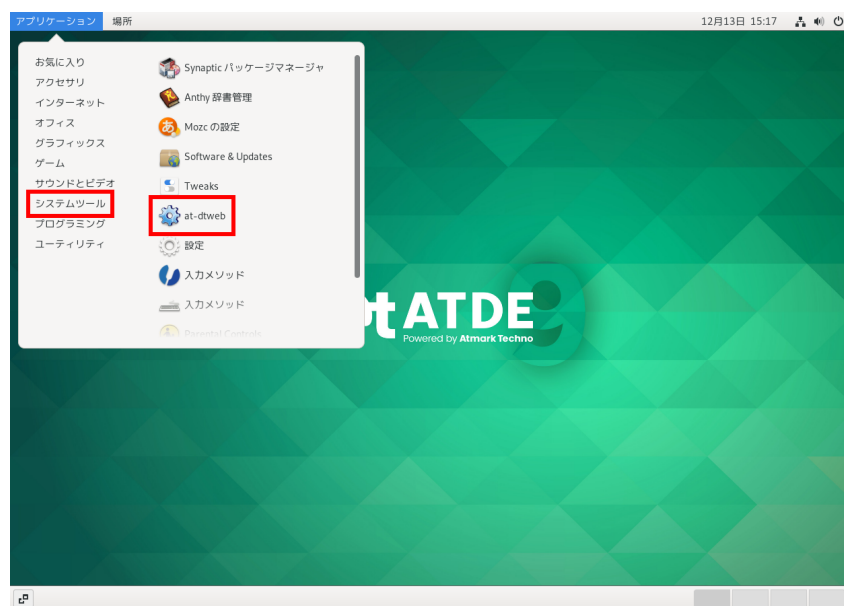


図 10.135 at-dtweb の起動開始

コマンドライン上からでも、at-dtweb コマンドで起動できます。

```
[ATDE ~]$ at-dtweb
```

### 1. ボードの選択

ボードを選択します。Armadillo-640 を選択して、「OK」をクリックします。

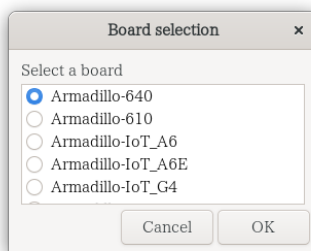


図 10.136 ボード選択画面

2. Linux カーネルディレクトリの選択

Linux カーネルディレクトリを選択します。コンフィギュレーション済みの Linux カーネルディレクトリを選択して、「OK」をクリックします。

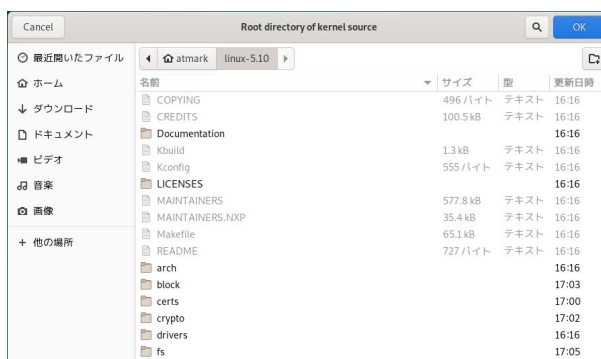


図 10.137 Linux カーネルディレクトリ選択画面

3. at-dtweb の起動完了

at-dtweb が起動し、次のように画面が表示されます。

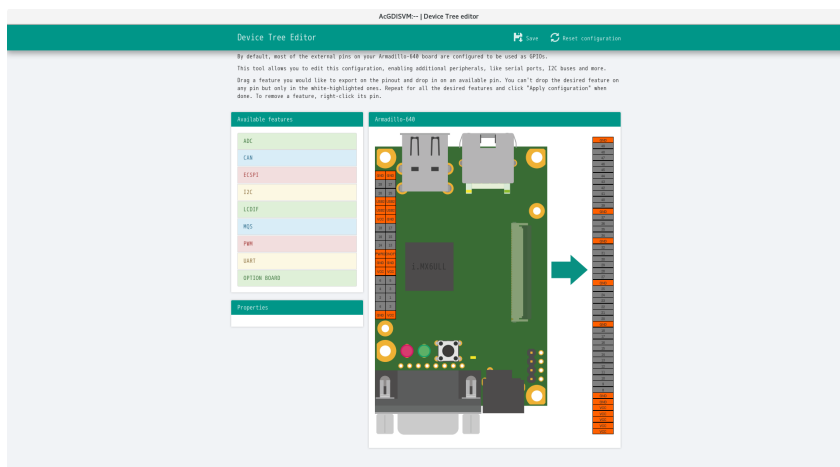



図 10.138 at-dtweb 起動画面




Linux カーネルは、事前にコンフィギュレーションされている必要があります。コンフィギュレーションの手順については「10.4. Armadillo のソフトウェアをビルドする」を参照してください。

### 10.9.3. Device Tree をカスタマイズ

#### 10.9.3.1. 機能の選択

機能の選択は、ドラッグ&ドロップで行います。画面左上の「Available features」から有効にしたい機能をドラッグし、画面右側の「Armadillo-640」の白色に変化したピンにドロップします。例として CON9 3/5 ピンを UART1 (RXD/TXD) に設定します。



何も機能が選択されていないピンには GPIO の機能が割り当てられます。

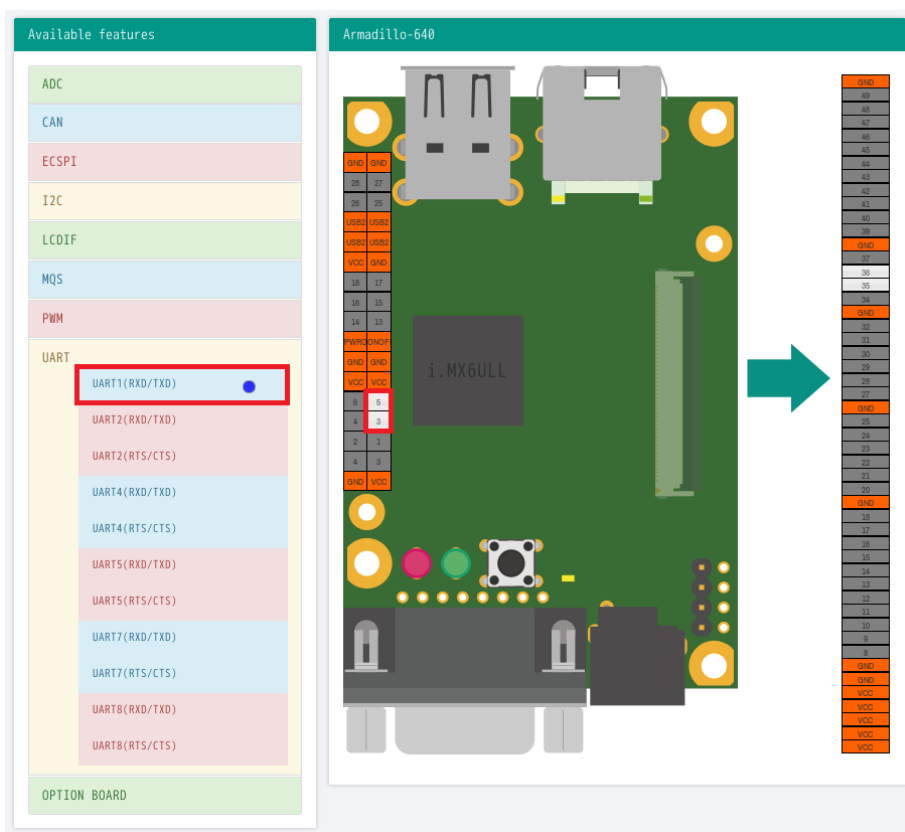


図 10.139 UART1(RXD/TXD)のドラッグ

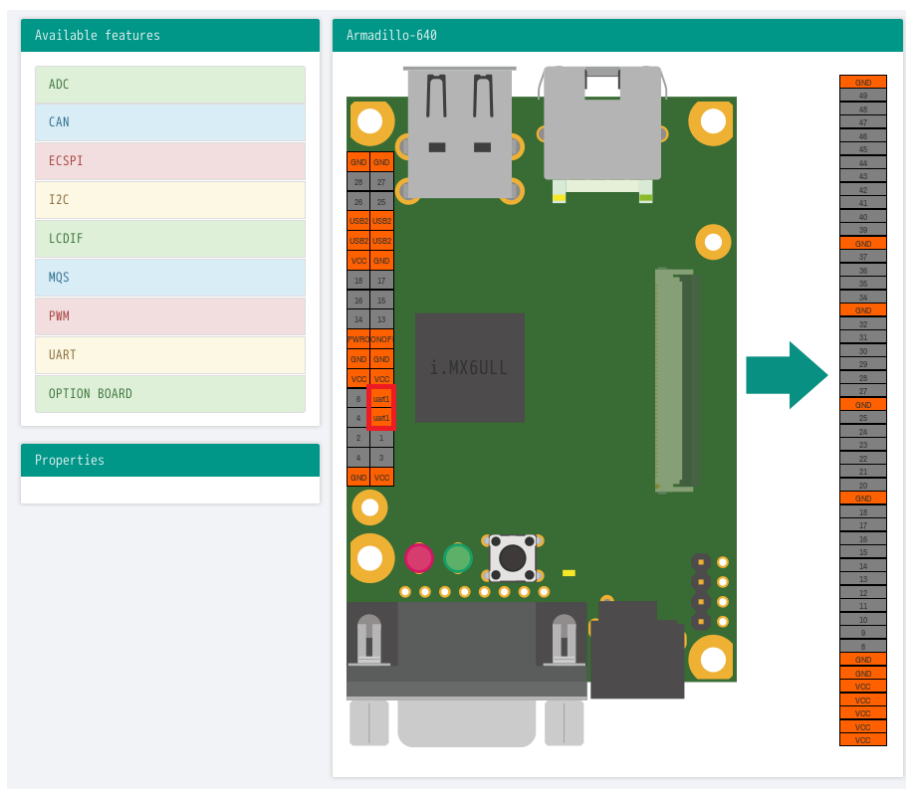


図 10.140 CON9 3/5 ピンへのドロップ

### 10.9.3.2. 信号名の確認

画面右側の「Armadillo-640」にドロップして設定したピンを左クリックすると信号名が表示されます。どのピンがどの信号に対応しているのかを確認することができます。

例として UART1 (RXD/TXD) の信号名を確認します。

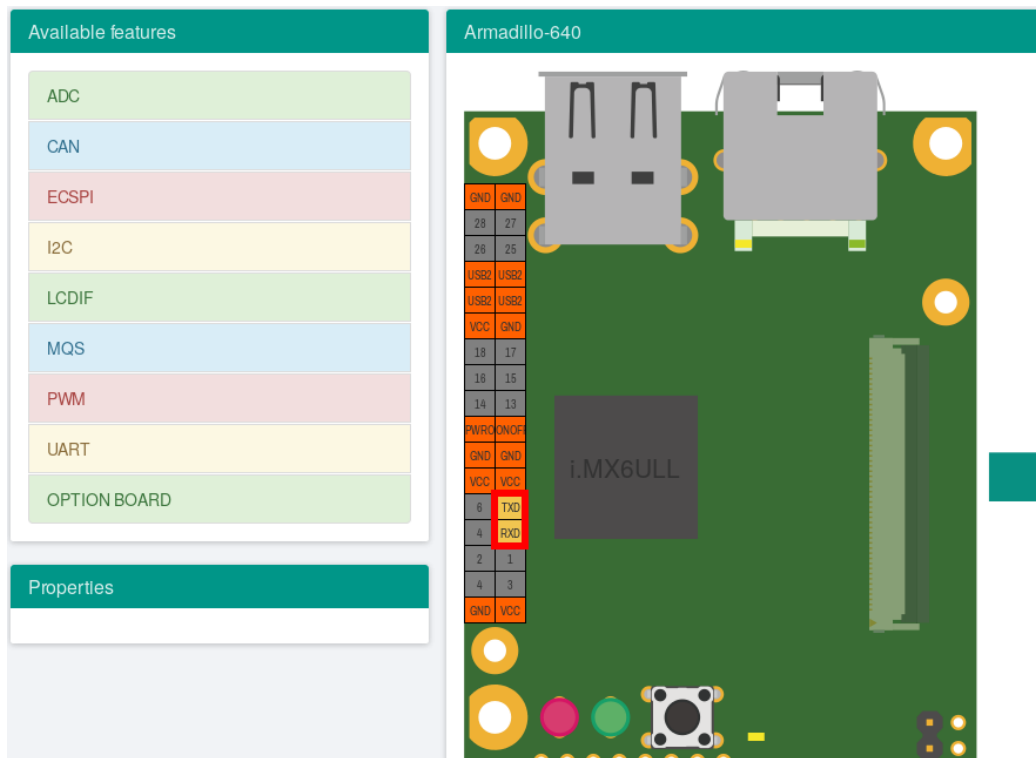



図 10.141 信号名の確認



再度ピンを左クリックすると機能名の表示に戻ります。

### 10.9.3.3. プロパティの設定

いくつかの機能にプロパティを設定することができます。画面右側の「Armadillo-640」に選択した機能を左クリックすると、画面左下の「Properties」からプロパティを選択することができます。

例として CON9 4/6 ピンの I2C2(SCL/SDA)の clock\_frequency プロパティを設定します。

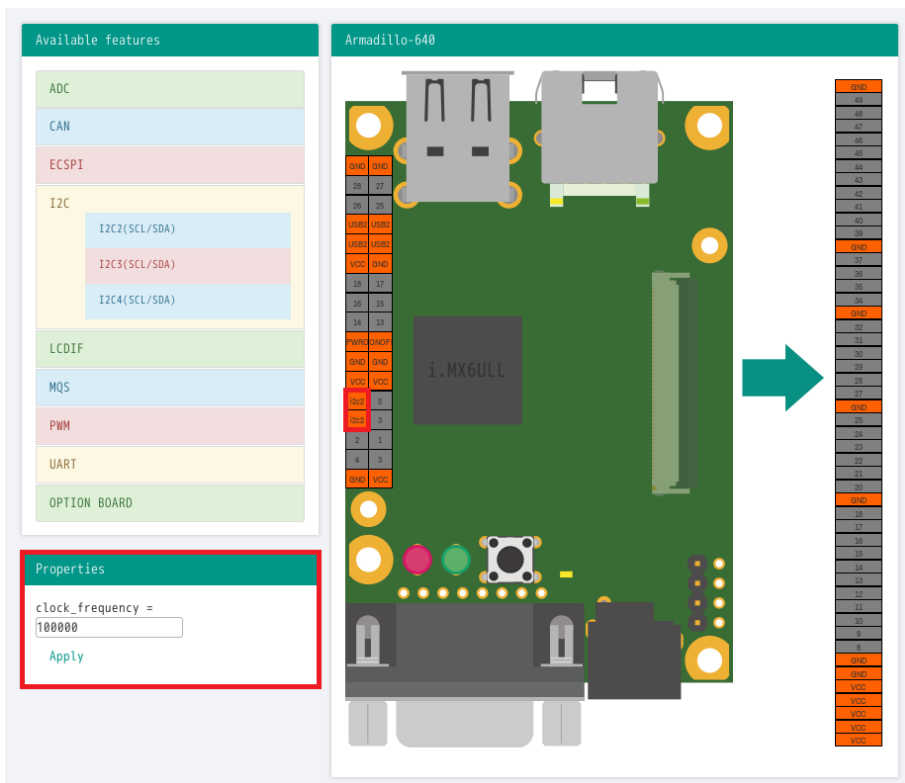


図 10.142 プロパティの設定

設定したプロパティを確定させるには「Apply」をクリックします。

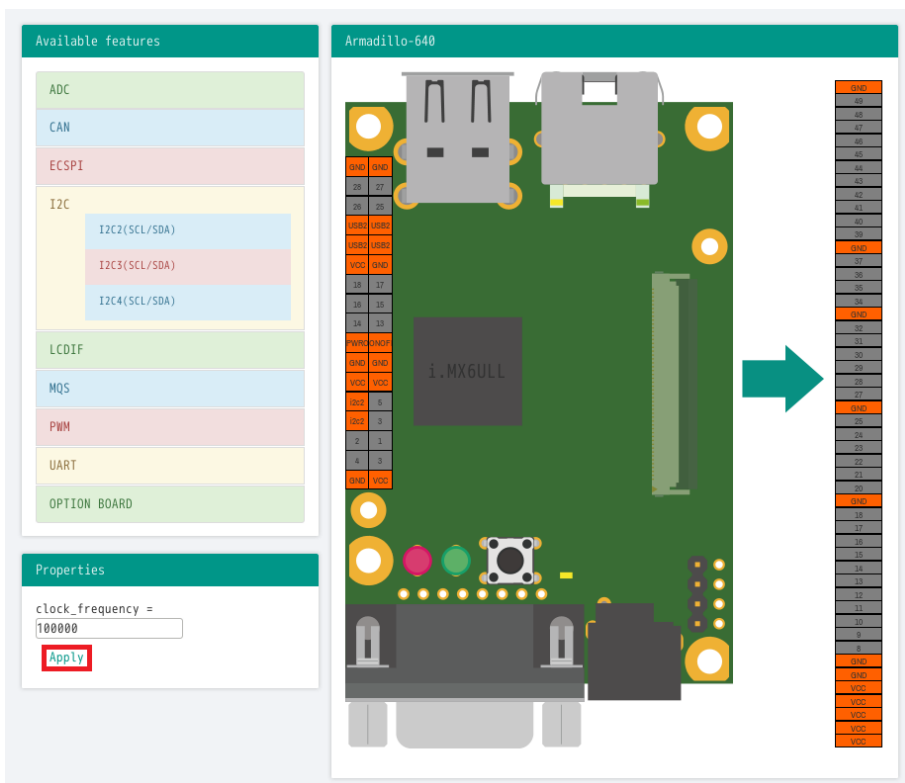


図 10.143 プロパティの保存

### 10.9.3.4. 機能の削除

全ての機能を削除する場合は、画面右上の「Reset configuration」をクリックします。機能ごとに削除する場合は、画面右側の「Armadillo-640」のピンを右クリックして「Remove」をクリックします。

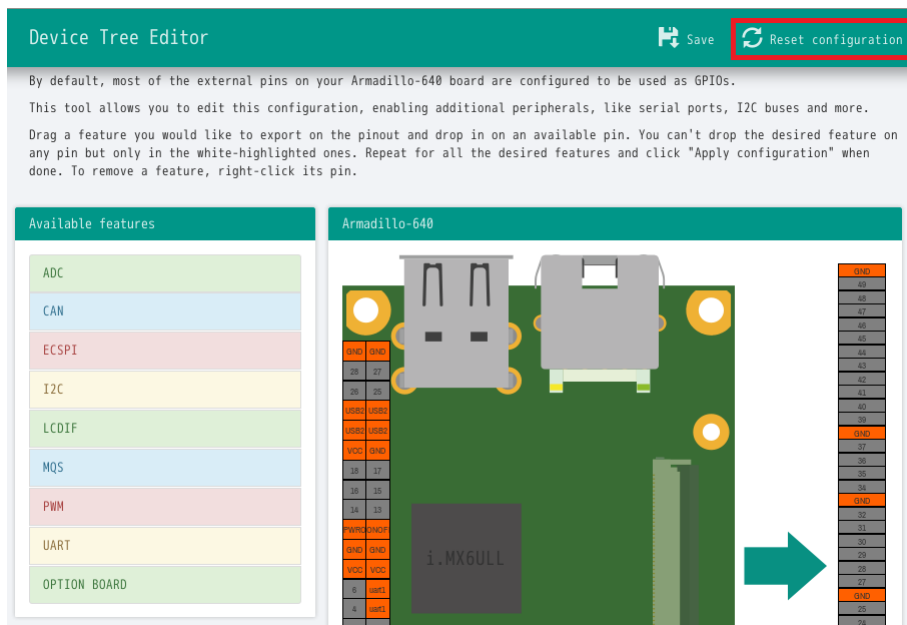


図 10.144 全ての機能の削除



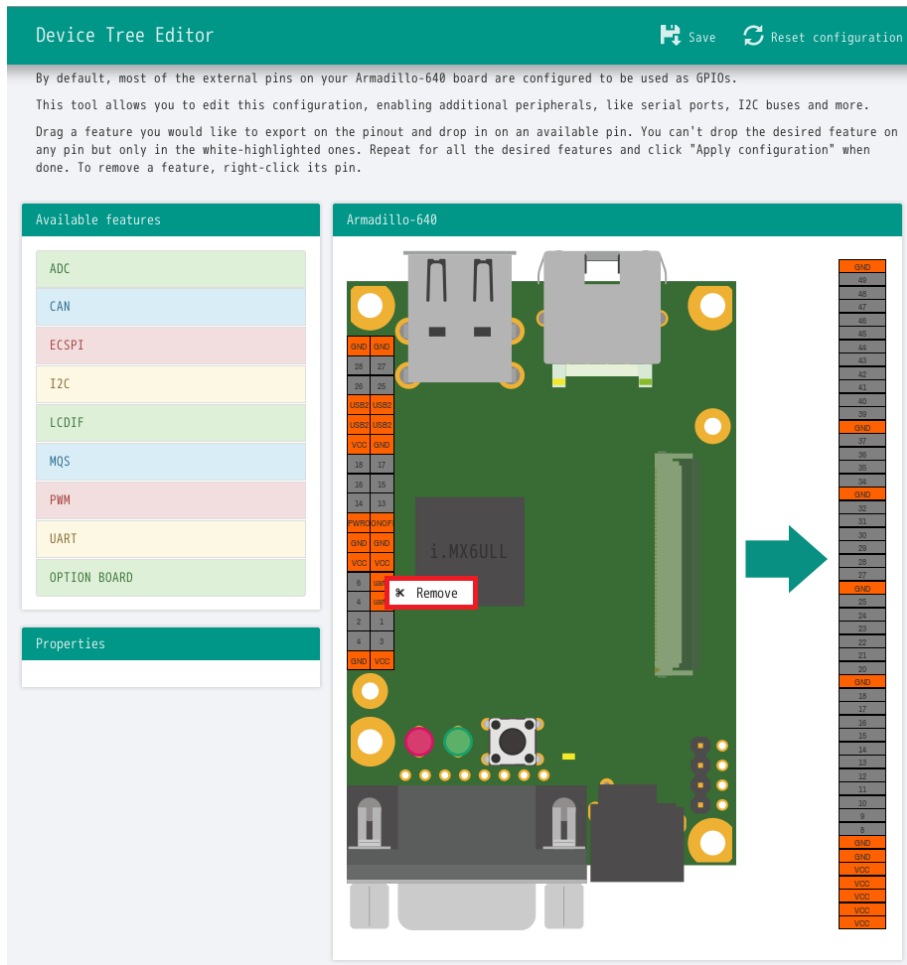


図 10.145 UART1(RXD/TXD)の削除

### 10.9.3.5. Device Tree のファイルの生成

Device Tree のファイルを生成するには、画面右上の「Save」をクリックします。

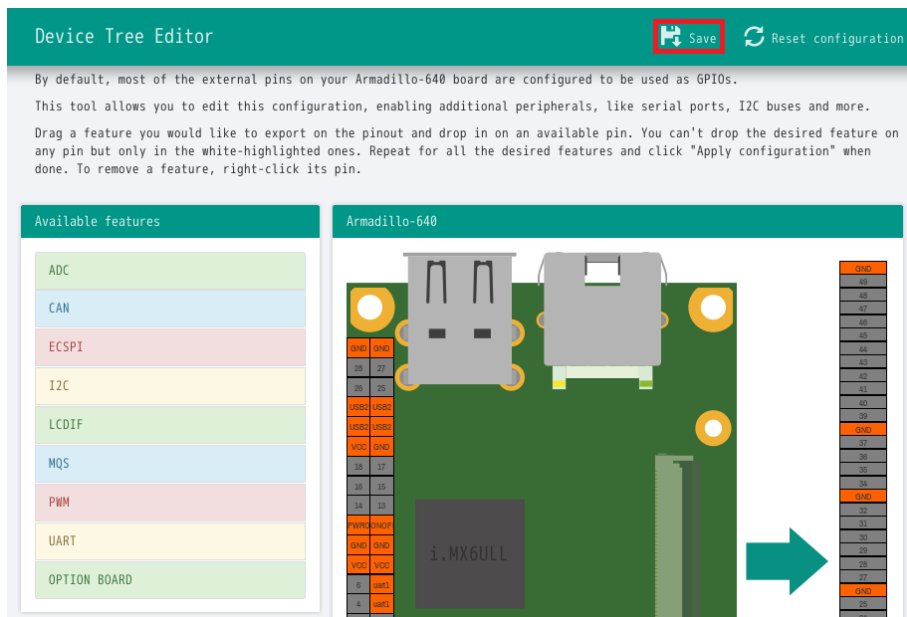


図 10.146 DTS/DTB の生成

以下の画面ようなメッセージが表示されると、dtbo ファイルおよび desc ファイルの生成は完了です。

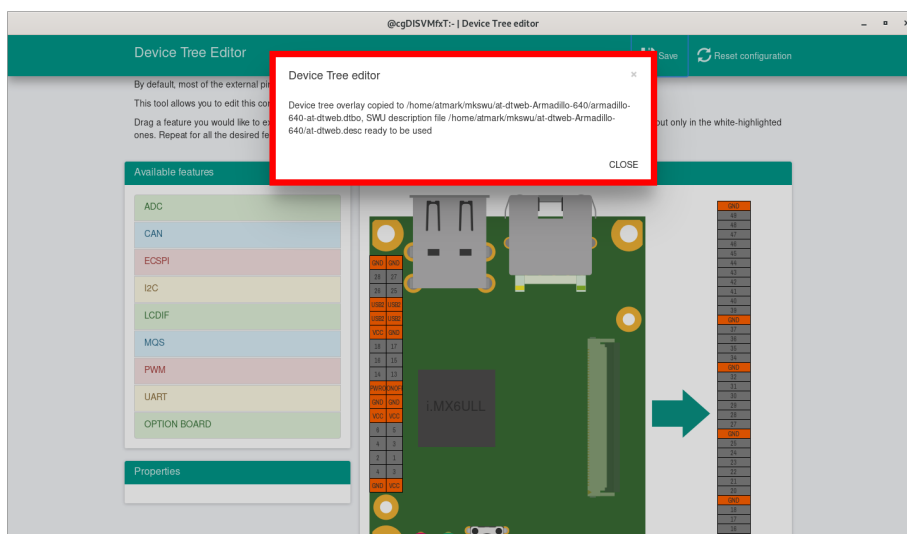


図 10.147 dtbo/desc の生成完了

ビルドが完了するとホームディレクトリ下の mkswu/at-dtweb-Armadillo-640 ディレクトリに、DT overlay ファイル(dtbo ファイル)と desc ファイルが生成されます。Armadillo-640 本体に書き込む場合は、mkswu コマンドで desc ファイルから SWU イメージを生成してアップデートしてください。

```
[ATDE ~]$ ls ~/mkswu/at-dtweb-Armadillo-640
armadillo-640-at-dtweb.dtbo  update_overlays.sh
at-dtweb.desc               update_preserve_files.sh
[ATDE ~]$ cd ~/mkswu/at-dtweb-Armadillo-640
[ATDE ~]$ mkswu at-dtweb.desc ①
Enter pass phrase for /home/atmark/mkswu/swupdate.key:
at-dtweb.swu を作成しました。
```

## ❶ SWU イメージを生成します。

SWU イメージを使ったアップデートの詳細は「10.7. Armadillo のソフトウェアをアップデートする」を参照してください。

### 10.9.4. DT overlay によるカスタマイズ

Device Tree は「DT overlay」(dtbo) を使用することでも変更できます。

DT overlay を使用することで、通常の dts の更新が自動的に入りつづける状態で dts の変更でしかできない設定を行うことができます。

/boot/overlays.txt に fdt\_overlays を dtbo 名で設定することで、u-boot が起動時にその DT overlay を通常の dtb と結合して起動します。

複数の DT overlay を使う場合は以下の例のようにスペースで別けたファイル名を記載することができます。

```
[armadillo ~]# vi /boot/overlays.txt ❶
fdt_overlays=armadillo-640-lcd70ext-l00.dtbo armadillo-640-at-dtweb.dtbo

[armadillo ~]# persist_file -vp /boot/overlays.txt ❷
'/boot/overlays.txt' -> '/mnt/boot/overlays.txt'
Added "/boot/overlays.txt" to /etc/swupdate_preserve_files

[armadillo ~]# reboot ❸
: (省略)
Applying fdt overlay: armadillo-640-lcd70ext-l00.dtbo ❹
Applying fdt overlay: armadillo-640-at-dtweb.dtbo
: (省略)
```

図 10.148 /boot/overlays.txt の変更例

- ❶ /boot/overlays.txt ファイルに「armadillo-640-at-dtweb.dtbo」を追加します。ファイルが存在しない場合は新規に作成してください。このファイルの詳細については「10.9.4. DT overlay によるカスタマイズ」を参照してください。
- ❷ /boot/overlays.txt を保存し、アップデートの場合でも保存します。
- ❸ overlay の実行のために再起動します。
- ❹ シリアルコンソールの場合に、u-boot によるメッセージを確認できます。

#### 10.9.4.1. 提供している DT overlay

以下の DT overlay を用意しています：

- ・ **armadillo-640-lcd70ext-l00.dtbo**: LCD オプションセット(7 インチタッチパネル WVGA 液晶を接続する場合にご使用ください。
- ・ **armadillo-640\_con9\_thread.dtb**: Armadillo-600 シリーズ BT/TH オプションモジュールを接続する場合にご使用ください。

- ・ **armadillo-640-con9-thread-lwb5plus.dtbo**: Armadillo-600 シリーズ WLAN コンボオプションモジュール または、Armadillo-600 シリーズ BT/TH オプションモジュール WLAN コンボ対応を接続する場合にご使用ください。

## 10.10. LCD オプションセット(7 インチタッチパネル WVGA 液晶)を利用する

この章では、「15.4. LCD オプションセット(7 インチタッチパネル WVGA 液晶)」の利用方法について説明します。

### 10.10.1. LCD オプションセット(7 インチタッチパネル WVGA 液晶)を利用する準備



デフォルトではこの作業は必要ありません。「10.9.4. DT overlay によるカスタマイズ」を行っている場合のみ必要です。

LCD オプションセット(7 インチタッチパネル WVGA 液晶)を利用するには、DT overlay の設定が必要です。

```
[armadillo~]# vi /boot/overlays.txt
fdt_overlays=armadillo-640-lcd70ext-l00.dtbo
[armadillo~]# persist_file /boot/overlays.txt
[armadillo~]# reboot
: (省略)
Applying fdt overlay: armadillo-640-lcd70ext-l00.dtbo
: (省略)
```

図 10.149 DT overlay の設定(LCD)

### 10.10.2. LCD オプションセット(7 インチタッチパネル WVGA 液晶)を利用する

「10.2.8. 画面表示を行う」を参照して画面表示を行うことができます。

## 10.11. Armadillo-600 シリーズ WLAN コンボオプションモジュールを利用する

この章では、「15.6. Armadillo-600 シリーズ WLAN コンボ、BT/TH オプションモジュール」に搭載されている Sterling LWB5+ を使って様々なネットワークを構成する例を紹介します。

### 10.11.1. 無線 LAN を利用する準備

「15.6. Armadillo-600 シリーズ WLAN コンボ、BT/TH オプションモジュール」の組み立てを行う前に、コンソールを CON3 に移動します。「図 15.25. オプションモジュールの CON2 をシリアルコンソールとして使用する場合の接続例」のように接続する場合は、この設定は不要です。

```
[armadillo ~]# vi /boot/uboot_env.d/console_con3
console=ttymxc2,115200
[armadillo ~]# persist_file -v /boot/uboot_env.d/console_con3
'/boot/uboot_env.d/console_con3' -> '/mnt/boot/uboot_env.d/console_con3'
[armadillo ~]# fw_setenv -s /boot/uboot_env.d/console_con3
Environment OK, copy 1
[armadillo ~]# fw_printenv | grep console=ttymxc
console=ttymxc2,115200
```

図 10.150 コンソールを CON3 に移動(WLAN)



Armadillo-600 シリーズ WLAN コンボオプションモジュールを利用しなくなった場合は、次のようにコンソールを CON9 に戻すことができます。

```
[armadillo ~]# vi /boot/uboot_env.d/console_con9
console=ttymxc0,115200
[armadillo ~]# persist_file -v /boot/uboot_env.d/console_con9
'/boot/uboot_env.d/console_con9' -> '/mnt/boot/uboot_env.d/console_con9'
[armadillo ~]# fw_setenv -s /boot/uboot_env.d/console_con9
Environment OK, copy 1
[armadillo ~]# fw_printenv | grep console=ttymxc
console=ttymxc0,115200
```

また、WLAN 機能を利用するには、DT overlay の設定が必要です。

```
[armadillo~]# vi /boot/overlays.txt
fdt_overlays=armadillo-640-con9-thread-lwb5plus.dtbo
[armadillo~]# persist_file /boot/overlays.txt
[armadillo~]# reboot
: (省略)
Applying fdt overlay: armadillo-640-con9-thread-lwb5plus.dtbo
: (省略)
```

図 10.151 DT overlay の設定(WLAN)

## 10.11.2. 無線 LAN アクセスポイント (AP) に接続する

Armadillo を子機として、無線 LAN AP に接続する方法を説明します。

以下は、WPA2-PSK(AES) の AP に接続する例です。ここでは、AP の ESSID を [essid]、パスワードを [passphrase] と表記します。

```
[armadillo ~]# nmcli device wifi connect [essid] password [passphrase]
[armadillo ~]# persist_file /etc/NetworkManager/system-connections/[essid].nmconnection ❶
```

図 10.152 無線 LAN AP に接続する

① 設定ファイルを永続化することで、Armadillo 起動時に自動的に AP に接続するようになります。

- ・ nmtui を使用して設定する

nmtui というツールを使用するとテキストユーザーインターフェースで設定を行うことができます。

```
[armadillo ~]# nmtui
```

図 10.153 nmtui を起動する

表示された画面上で設定が行なえます。

Activate a connection を選択します。

```

+-+ NetworkManager TUI +-+
|
| Please select an option
|
| Edit a connection
| Activate a connection
| Set system hostname
|
| Quit
|
|                                     <OK>
+-----+
    
```

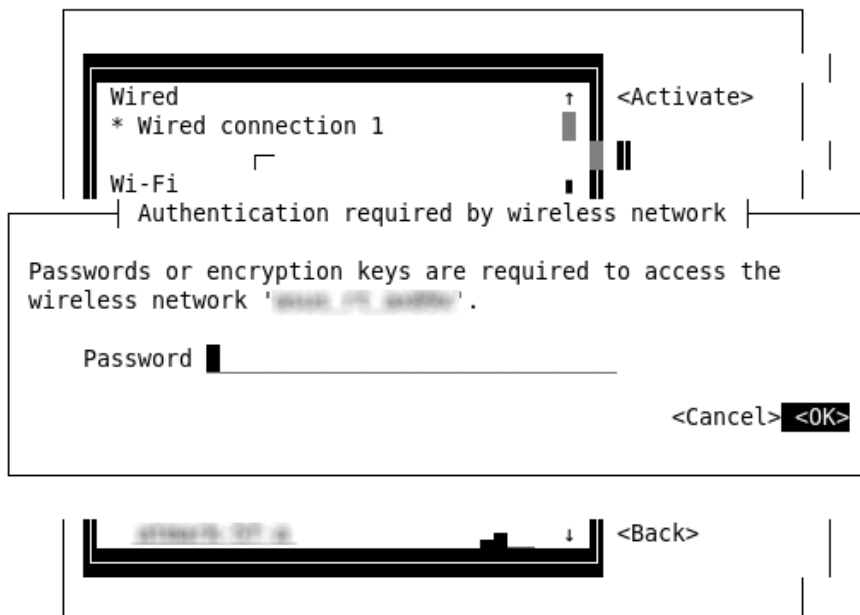
Wi-Fi の下に表示される AP 一覧の中から、接続したいものを選択します。

```

Wired
* Wired connection 1
┌───┴───┐
Wi-Fi
  avia_ft_6030
  avia_ft_6030_5G
  connect-glinet
  NETGEAR-5G
  atmark-47
  proglinet
  NETGEAR
  Intel(R) Wi-Fi 2_4G_WPA2
  atmark-47-a
  Linksys 5G
  vici-wifi
  atmark-47-a
    
```

↑ <Deactivate>  
||  
↓ <Back>

表示された画面で、接続のためのパスワードを入力します。



入力後に OK を押すと接続されます。

### 10.11.3. 無線 LAN アクセスポイント (AP) として設定する

Armadillo を無線 LAN AP として設定する方法を説明します。AP を設定するには hostapd というソフトウェアと、DNS/DHCP サーバである dnsmasq というソフトウェアを使用します。

hostapd と dnsmasq は Armadillo Base OS にデフォルトでインストール済みとなっているため、インストール作業は不要です。インストールされていない場合は、Armadillo Base OS を最新バージョンに更新してください。



アクセスポイントモード (AP) とステーションモード (STA) の同時利用はできません。Armadillo を子機として無線 LAN AP に接続しながら、hostapd を起動するような使い方は避けてください。

#### 10.11.3.1. hostapd を使用して設定する

- ・ bridge インターフェースを追加する

NetworkManager で bridge インターフェース (br0) を追加します。同時に IP アドレスも設定します。ここでは 192.0.2.1 を設定しています。

```
[armadillo ~]# nmcli con add type bridge ifname br0
[armadillo ~]# nmcli con mod bridge-br0 ipv4.method manual ipv4.address "192.0.2.1/24"
[armadillo ~]# nmcli con up bridge-br0
[armadillo ~]# persist_file /etc/NetworkManager/system-connections/bridge-br0.nmconnection ❶
```

図 10.154 bridge インターフェースを作成する

### ❶ 設定ファイルを永続化します。

また、NetworkManager のデフォルト状態では定期的に wlan0 のスキャンを行っています。スキャン中は AP の性能が落ちてしまうため wlan0 を NetworkManager の管理から外します。

```
[armadillo ~]# vi /etc/NetworkManager/conf.d/90_disable_wlan0.conf
[device_wlan0]
match-device=interface-name:wlan0
managed=0

[armadillo ~]# persist_file /etc/NetworkManager/conf.d/90_disable_wlan0.conf
[armadillo ~]# nmcli d set wlan0 managed no ❶
```

## 図 10.155 wlan0 インターフェースを NetworkManager の管理から外す

### ❶ nmcli で NetworkManager をリスタートせずに設定します。



hostapd が使用するインターフェース (wlan0) は /etc/NetworkManager/conf.d/00\_disable\_ap.conf ファイルによってデフォルトで NetworkManager の管理から外しております。

- ・ hostapd を設定する

hostapd の設定ファイルの雛形として用意してある /etc/hostapd/hostapd.conf.example をコピーして使用します。

```
[armadillo ~]# cp /etc/hostapd/hostapd.conf.example /etc/hostapd/hostapd.conf
[armadillo ~]# vi /etc/hostapd/hostapd.conf
hw_mode=a ❶
channel=44 ❷
ssid=myap ❸
wpa_passphrase=myap_pass ❹
interface=wlan0 ❺
bridge=br0
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
driver=nl80211
country_code=JP
ctrl_interface=/var/run/hostapd
ctrl_interface_group=0
disassoc_low_ack=1
preamble=1
wmm_enabled=1
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
ieee80211ac=1
ieee80211ax=0 ❻
```



```

ieee80211n=1
ieee80211d=1
ieee80211h=1
logger_syslog=-1
logger_syslog_level=2
logger_stdout=-1
logger_stdout_level=2

[armadillo ~]# persist_file /etc/hostapd/hostapd.conf ⑦
[armadillo ~]# rc-service hostapd start ⑧
[armadillo ~]# rc-update add hostapd ⑨
[armadillo ~]# persist_file /etc/runlevels/default/hostapd ⑩

```

図 10.156 hostapd.conf を編集する

- ① 5GHz であれば a を、2.4GHz であれば g を設定します。
- ② 使用するチャンネルを設定します。
- ③ 子機から接続するための任意の SSID を設定します。この例では myap を設定しています。
- ④ 子機から接続するための任意のパスワードを設定します。この例では myap\_pass を設定しています。
- ⑤ wlan0 を設定します。
- ⑥ IEEE802.11ax に非対応のため、0 を指定します。
- ⑦ 設定ファイルを永続化します。
- ⑧ hostapd を起動します。
- ⑨ Armadillo 起動時に hostapd が自動的に起動されるようにします。
- ⑩ hostapd 自動起動の設定を永続化します。

・ dnsmasq を設定する

dnsmasq の設定ファイルを以下の内容で作成し /etc/dnsmasq.d/ 下に配置します。ファイル名は任意ですが、拡張子は .conf としてください。ここでは dhcp.conf としています。

```

[armadillo ~]# vi /etc/dnsmasq.d/dhcp.conf
interface=br0
bind-dynamic
dhcp-range=192.0.2.10, 192.0.2.254, 24h ①

[armadillo ~]# persist_file /etc/dnsmasq.d/dhcp.conf ②
[armadillo ~]# rc-service dnsmasq restart ③

```

図 10.157 dnsmasq の設定ファイルを編集する

- ① 子機に割り当てる IP アドレスの範囲とリース期間を設定します。
- ② 設定ファイルを永続化します。
- ③ dnsmasq を再起動します。

hostapd と dnsmasq の起動完了後、子機から hostapd.conf で設定した SSID とパスワードで接続できます。

## 10.11.4. ルータとして設定する

Armadillo をルータとして設定して外部ネットワークに接続する方法を説明します。ここでは外部ネットワークの例として一般的なインターネットを設定しています。

### 10.11.4.1. 無線 LAN 側に接続した機器から Ethernet 経由でインターネットに接続する

Armadillo を無線 LAN AP として設定し、AP に接続した子機から、Ethernet(eth0) を経由してインターネットに接続する方法を説明します。

最初に、「10.11.3. 無線 LAN アクセスポイント (AP) として設定する」を参照して AP の設定を完了させてください。

- ・ ip\_forward を有効にする

ルータとして機能させるために、ip\_forward を有効にします。sysctl の設定ファイルを以下の内容で作成し /etc/sysctl.d/ 下に配置します。ファイル名は任意ですが、拡張子は .conf としてください。ここでは router.conf としています。

```
[armadillo ~]# vi /etc/sysctl.d/router.conf
net.ipv4.ip_forward = 1 ❶

[armadillo ~]# persist_file /etc/sysctl.d/router.conf ❷
[armadillo ~]# rc-service sysctl restart ❸
```

図 10.158 ip\_forward を有効にする

- ❶ 1 (有効) に設定します。
- ❷ 設定ファイルを永続化します。
- ❸ sysctl を再起動して設定を反映させます。

- ・ iptables コマンドで NAT を設定する

```
[armadillo ~]# iptables -t nat -A POSTROUTING -s 192.0.2.0/24 -o eth0 -j MASQUERADE ❶
[armadillo ~]# /etc/init.d/iptables save ❷
[armadillo ~]# rc-update add iptables ❸
[armadillo ~]# persist_file /etc/iptables/rules-save /etc/runlevels/default/iptables ❹
```

図 10.159 NAT を設定する

- ❶ ここで設定する IP アドレスのネットワーク部は AP に設定したものと合わせてください。
- ❷ iptables の設定を保存します。
- ❸ サービスを有効にします
- ❹ 保存した設定ファイルを永続化します。

設定完了後、AP を起動して子機を接続すると子機からインターネットに接続することができます。

#### 10.11.4.2. Ethernet 側に接続した機器から無線 LAN 経由でインターネットに接続する

Armadillo の Ethernet(eth0) に接続した機器から、無線 LAN を経由してインターネットに接続する方法を説明します。

- ・ ip\_forward を有効にする

「[図 10.158. ip\\_forward を有効にする](#)」と同様の手順で設定します。

- ・ bridge インターフェースを追加して eth0 を割り当てる

```
[armadillo ~]# nmcli con down "Wired connection 1" ❶
[armadillo ~]# nmcli con add type bridge ifname br0 ❷
[armadillo ~]# nmcli con mod bridge-br0 ipv4.method manual ipv4.address "192.0.2.1/24" bridge.stp
off ❸
[armadillo ~]# nmcli con add type ethernet slave-type bridge ifname eth0 master bridge-br0 ❹
[armadillo ~]# nmcli con up bridge-br0
[armadillo ~]# nmcli con up bridge-slave-eth0
[armadillo ~]# persist_file /etc/NetworkManager/system-connections/bridge-br0.nmconnection ❺
[armadillo ~]# persist_file /etc/NetworkManager/system-connections/bridge-slave-eth0.nmconnection
```

↩

図 10.160 bridge に eth0 を割り当てる

- ❶ デフォルトで存在しているコネクションを down します。
- ❷ bridge インターフェースを作成します。
- ❸ 作成した bridge-br0 に任意の IP アドレスを設定し STP を無効にします。
- ❹ eth0 を bridge-br0 に割り当てます。
- ❺ それぞれの設定ファイルを永続化します。

- ・ dnsmasq を設定する

dnsmasq の設定ファイルを以下の内容で作成し /etc/dnsmasq.d/ 下に配置します。ファイル名は任意ですが、拡張子は .conf としてください。ここでは dhcp.conf としています。

```
[armadillo ~]# vi /etc/dnsmasq.d/dhcp.conf
interface=br0
bind-dynamic
dhcp-range=192.0.2.10, 192.0.2.254, 24h ❶

[armadillo ~]# persist_file /etc/dnsmasq.d/dhcp.conf
[armadillo ~]# rc-service dnsmasq restart ❷
```

図 10.161 dnsmasq の設定ファイルを編集する

- ❶ 接続した機器に割り当てる IP アドレスの範囲とリース期間を設定します。
- ❷ dnsmasq を再起動します。

- ・ iptables コマンドで NAT を設定する

```
[armadillo ~]# iptables -t nat -A POSTROUTING -s 192.0.2.0/24 -o wlan0 -j MASQUERADE ❶  
[armadillo ~]# /etc/init.d/iptables save ❷  
[armadillo ~]# rc-update add iptables ❸  
[armadillo ~]# persist_file /etc/iptables/rules-save /etc/runlevels/default/iptables ❹
```

図 10.162 NAT を設定する

- ❶ ここで設定する IP アドレスのネットワーク部は bridge-br0 に設定したものと合わせてください。
- ❷ iptables の設定を保存します。
- ❸ サービスを有効にします
- ❹ 保存した設定ファイルを永続化します。

- ・ 無線 LAN AP に接続する

「10.11.2. 無線 LAN アクセスポイント (AP) に接続する」と同様の手順で、無線 LAN アクセスポイントに接続します。

設定完了後、eth0 ポートに機器を接続すると無線 LAN 経由でインターネットに接続することができます。

## 10.12. Armadillo-600 シリーズ BT/TH オプションモジュールを利用する

この章では、「15.6. Armadillo-600 シリーズ WLAN コンボ、BT/TH オプションモジュール」に搭載されている EYSKBNZWB の利用方法について説明します。

EYSKBNZWB は、BT または Thread 機能を選択して利用することができます。

### 10.12.1. Armadillo-600 シリーズ BT/TH オプションモジュールを利用する準備

「15.6. Armadillo-600 シリーズ WLAN コンボ、BT/TH オプションモジュール」の組み立てを行う前に、コンソールを CON3 に移動します。「図 15.25. オプションモジュールの CON2 をシリアルコンソールとして使用する場合の接続例」のように接続する場合は、この設定は不要です。

```
[armadillo ~]# vi /boot/uboot_env.d/console_con3  
console=ttymxc2,115200  
[armadillo ~]# persist_file -v /boot/uboot_env.d/console_con3  
'/boot/uboot_env.d/console_con3' -> '/mnt/boot/uboot_env.d/console_con3'  
[armadillo ~]# fw_setenv -s /boot/uboot_env.d/console_con3  
Environment OK, copy 1  
[armadillo ~]# fw_printenv | grep console=ttymxc  
console=ttymxc2,115200
```

図 10.163 コンソールを CON3 に移動(BT/TH)



Armadillo-600 シリーズ BT/TH オプションモジュールを利用しなくなった場合は、次のようにコンソールを CON9 に戻すことができます。

```
[armadillo ~]# vi /boot/u-boot_env.d/console_con9
console=ttymxc0,115200
[armadillo ~]# persist_file -v /boot/u-boot_env.d/console_con9
'/boot/u-boot_env.d/console_con9' -> '/mnt/boot/u-boot_env.d/console_con9'
[armadillo ~]# fw_setenv -s /boot/u-boot_env.d/console_con9
Environment OK, copy 1
[armadillo ~]# fw_printenv | grep console=ttymxc
console=ttymxc0,115200
```

また、Armadillo-600 シリーズ BT/TH オプションモジュールを利用するには、DT overlay の設定が必要です。

```
[armadillo~]# vi /boot/overlays.txt
fdt_overlays=armadillo-640-con9-thread-lwb5plus.dtbo
[armadillo~]# persist_file /boot/overlays.txt
[armadillo~]# reboot
: (省略)
Applying fdt overlay: armadillo-640-con9-thread-lwb5plus.dtbo
: (省略)
```

図 10.164 DT overlay の設定(BT/TH)

## 10.12.2. BT 機能を利用する

BT 機能を有効化するためのコンテナを、Armadillo-640 コンテナ [<https://armadillo.atmark-techno.com/resources/software/armadillo-640/container>] からダウンロードします。

BT 機能を有効化するには、以下のコマンドを入力します。

```
[armadillo ~]# curl -s https://armadillo.atmark-techno.com/files/downloads/armadillo-640/container/
firmware-at-bt-writer-latest.tar | podman load
Getting image source signatures
Writing manifest to image destination
Storing signatures
Loaded image: localhost/firmware-at-bt-writer:latest
[armadillo ~]# podman run --privileged --cap-add=CAP_SYS_RWIO localhost/firmware-at-bt-writer
Updating firmware...
Updating firmware has been successful!
[armadillo ~]# podman rmi localhost/firmware-at-bt-writer
```



図 10.165 BT/TH オプションモジュールの BT 機能を有効化する



一度 BT 機能を有効化すると、再起動後も BT 機能が有効化された状態を維持します。再起動する度に BT 機能を有効化する必要はありません。



Bluetooth® version 5.0 以降で追加された Coded PHY(Long Range)などの機能は、この章に記載の手順では利用することができません。これは、BlueZ が非対応の為です。

EYSKBNZWB と Sterling LWB5+ の BT 機能は同時に利用することができません。デフォルトでは EYSKBNZWB の BT 機能が利用できないようになっています。次のように、idVendor と idProduct の値を変更して再起動してください。

```
[armadillo ~]# vi /lib/udev/rules.d/80-bluetooth.rules
: (省略)
ACTION=="add", SUBSYSTEM=="usb", DRIVER=="usb", \
  ATTRS{idVendor}=="04b4", ATTRS{idProduct}=="640c", ATTR{authorized}="0"
[armadillo ~]# persist_file /lib/udev/rules.d/80-bluetooth.rules
[armadillo ~]# reboot
```

### 図 10.166 Sterling LWB5+ の BT 機能を無効化する

これで EYSKBNZWB の BT 機能を利用する準備は完了です。「10.2.5.1. Bluetooth デバイスを扱う」を参照して BT 機能を利用できます。

## 10.12.3. Thread 機能を利用する

Thread 機能を有効化するためのコンテナを、Armadillo-640 コンテナ [<https://armadillo.atmark-techno.com/resources/software/armadillo-640/container>] からダウンロードします。

Thread 機能を有効化するには、以下のコマンドを入力します。

```
[armadillo ~]# curl -s https://armadillo.atmark-techno.com/files/downloads/armadillo-640/container/
firmware-at-thread-writer-latest.tar | podman load
Getting image source signatures
Writing manifest to image destination
Storing signatures
Loaded image: localhost/firmware-at-thread-writer:latest
[armadillo ~]# podman run --privileged --cap-add=CAP_SYS_RWIO localhost/firmware-at-thread-writer
Updating firmware...
Updating firmware has been successful!
[armadillo ~]# podman rmi localhost/firmware-at-thread-writer
```



### 図 10.167 BT/TH オプションモジュールの Thread 機能を有効化する



一度 Thread 機能を有効化すると、再起動後も Thread 機能が有効化された状態を維持します。再起動する度に Thread 機能を有効化する必要はありません。

これで EYSKBNZWB の Thread 機能を利用する準備は完了です。「10.2.5.4. Thread デバイスを扱う」を参照して Thread 機能を利用できます。デバイスファイルは `/dev/ttyACM0` です。



TTY デバイスは検出された順番にインデックスが割り振られます。USB シリアルデバイスなどを接続してしている場合は、デバイスファイルのインデックスが異なる可能性があります。

## 10.13. 動作中の Armadillo の温度を測定する

この章では、Armadillo Base OS 搭載製品を組み込んだユーザー製品の熱設計時に役立つ温度プロファイラツールである「atmark-thermal-profiler」について紹介します。

### 10.13.1. 温度測定的重要性

Armadillo は製品ごとに動作温度範囲が設定されていますが、それらはいくまでも標準筐体に放熱材と共に取り付けて使用した場合の目安であり、実運用時には自作の筐体の使用や放熱の有無などで記載のスペック通りにならない場合があります。また、Armadillo には CPU または SoC が特定の温度以上になると、自動的にシャットダウンするサーマルシャットダウン機能が搭載されています。そのため、現実的には Armadillo を組み込んだ製品を運用時と同等の環境で動作させつつ、実際に温度を計測して実運用時の CPU 及び SoC 温度がどの程度まで上がるか、サーマルシャットダウンは起こらないかを確認する必要があります。

Armadillo Base OS 搭載製品では、動作中の Armadillo の各種温度等を取得し CSV 形式で出力する atmark-thermal-profiler を利用することができますので、温度測定に役立てることができます。

### 10.13.2. atmark-thermal-profiler をインストールする

atmark-thermal-profiler は apk パッケージで公開されていますので、apk add コマンドでインストールすることが可能です。

```
[armadillo ~]# apk upgrade
[armadillo ~]# apk add atmark-thermal-profiler
```

図 10.168 atmark-thermal-profiler をインストールする



atmark-thermal-profiler はデバッグ(開発)用途で温度情報を収集及び解析するツールです。atmark-thermal-profiler は、他の apk パッケージと同様に `persist_file -a` コマンドで永続的にインストールしておくことが可能ですが、ログの保存のために Armadillo が起動している間 eMMC への

書き込みを続けるので、Armadillo を組み込んだ製品の運用時に動かしたままにしておくことは推奨しません。

atmark-thermal-profiler を永続的にインストールする場合は、運用時には必ず削除してください。

### 10.13.3. atmark-thermal-profiler を実行・停止する

「[図 10.169. atmark-thermal-profiler を実行する](#)」に示すコマンドを実行することで、atmark-thermal-profiler が動作を開始します。

```
[armadillo ~]# rc-service atmark-thermal-profiler start
```

**図 10.169 atmark-thermal-profiler を実行する**

「[図 10.170. atmark-thermal-profiler を停止する](#)」に示すコマンドを実行することで、atmark-thermal-profiler が動作を停止します。

```
[armadillo ~]# rc-service atmark-thermal-profiler stop
```

**図 10.170 atmark-thermal-profiler を停止する**

### 10.13.4. atmark-thermal-profiler が出力するログファイルを確認する

atmark-thermal-profiler は、インストール直後から自動的に温度や CPU 負荷率、Load Average などの情報を 30 秒に 1 度の周期で集め、/var/log/thermal\_profile.csv に追記していきます。

```
[armadillo ~]# head /var/log/thermal_profile.csv
DATE, ONESHOT, CPU_TMEP, SOC_TEMP, LOAD_AVE, CPU_1, CPU_2, CPU_3, CPU_4, CPU_5, USE_1, USE_2, USE_3, USE_4, USE_5
2022-11-30T11:11:05+09:00, 0, 54, 57, 0.24, /usr/sbin/rngd -b -p /run/rngd.pid -q -0 jitter:buffer_size:
4133 -0 jitter:refill_thresh:4133 -0 jitter:thread_count:1, /usr/sbin/chronyd -f /etc/chrony/
chrony.conf, [kworker/1:3H-kb], podman network inspect podman, /usr/sbin/NetworkManager -n, 22, 2, 2, 0, 0,
: (省略)
```

↵  
↵  
↵

**図 10.171 ログファイルの内容例**

thermal\_profile.csv の 1 行目はヘッダ行です。各列についての説明を「[表 10.7. thermal\\_profile.csv の各列の説明](#)」に記載します。

**表 10.7 thermal\_profile.csv の各列の説明**

ヘッダ	説明
DATE	その行のデータ取得日時です。"年-月-日 T 時:分:秒+タイムゾーン" の形式で出力されます。
ONESHOT	この列が 1 の行のデータは、サーマルシャットダウンを含むシャットダウンが実行された時に取得されたことを示します。
CPU_TEMP	計測時点の CPU 温度を示します。単位は°Cです。
SOC_TEMP	計測時点の SoC 温度を示します。単位は°Cです。製品によっては非対応で、その場合は空白になります。



ヘッダ	説明
LOAD_AVE	計測時点から直近 1 分間の Load Average です。
CPU_1	計測時点の CPU 使用率 1 位のプロセスです。
CPU_2	計測時点の CPU 使用率 2 位のプロセスです。
CPU_3	計測時点の CPU 使用率 3 位のプロセスです。
CPU_4	計測時点の CPU 使用率 4 位のプロセスです。
CPU_5	計測時点の CPU 使用率 5 位のプロセスです。
USE_1	計測時点の CPU 使用率 1 位のプロセスの CPU 使用率です。
USE_2	計測時点の CPU 使用率 2 位のプロセスの CPU 使用率です。
USE_3	計測時点の CPU 使用率 3 位のプロセスの CPU 使用率です。
USE_4	計測時点の CPU 使用率 4 位のプロセスの CPU 使用率です。
USE_5	計測時点の CPU 使用率 5 位のプロセスの CPU 使用率です。

### 10.13.5. 温度測定結果の分析

atmark-thermal-profiler を使用して得られたログファイルの内容を分析してみます。

#### 10.13.5.1. サーマルシャットダウン温度の確認

予め、使用している Armadillo が何°Cでサーマルシャットダウンするか確認しておきます。ここでは、Armadillo Base OS を搭載している Armadillo-IoT ゲートウェイ G4 を例とします。他の製品では得られる結果が異なる場合があることに注意してください。

```
[armadillo ~]# cat /sys/class/thermal/thermal_zone0/trip_point_1_temp
105000 ❶
[armadillo ~]# cat /sys/class/thermal/thermal_zone1/trip_point_1_temp
105000 ❷
```

図 10.172 サーマルシャットダウン温度の確認(Armadillo-IoT ゲートウェイ G4 を例に)

- ❶ CPU のサーマルシャットダウン温度です。ミリ°Cで表記されているので、105°Cでサーマルシャットダウンすることがわかります。
- ❷ SoC のサーマルシャットダウン温度です。ミリ°Cで表記されているので、105°Cでサーマルシャットダウンすることがわかります。

#### 10.13.5.2. 温度測定結果のグラフ化

atmark-thermal-profiler が出力するログ(thermal\_profile.csv)は CSV ファイルなので、各種表計算ソフトでインポートしてグラフ化することが可能です。これにより Armadillo 動作中の温度の変化が可視化され、得られる情報が見やすくなります。

「図 10.173. Armadillo-IoT ゲートウェイ G4 で取得した温度のグラフ」は Armadillo-IoT ゲートウェイ G4 上で一定期間 atmark-thermal-profiler を実行して取得した thermal\_profile.csv を Google スプレッドシートでグラフ化したものです。例のために、途中で stress-ng コマンドを実行して CPU に負荷を与えた後、stress-ng コマンドを停止して CPU と SoC の温度が下がるのを待った際のデータです。

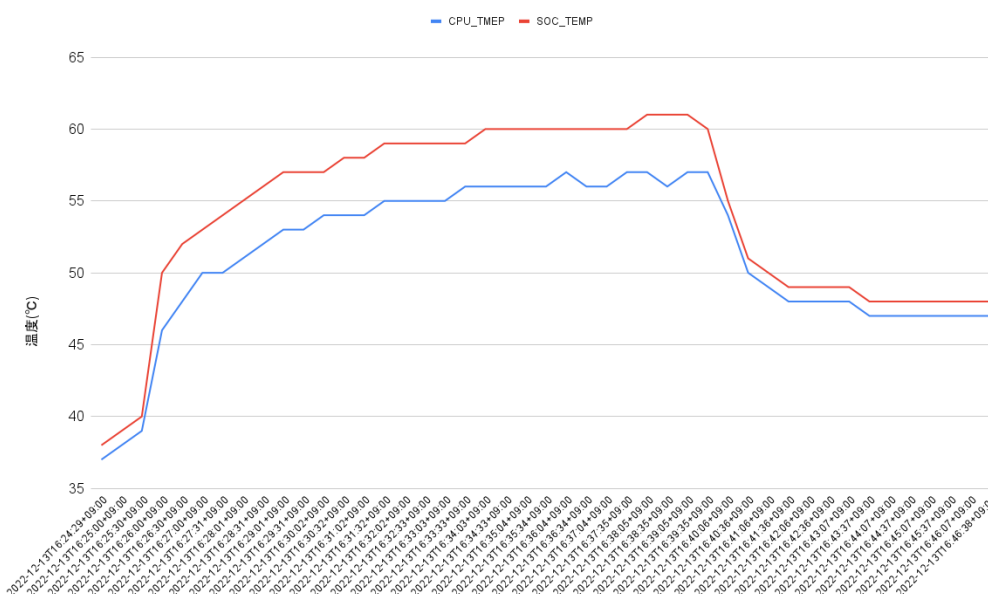


図 10.173 Armadillo-IoT ゲートウェイ G4 で取得した温度のグラフ

グラフの縦軸は温度(°C)で、横軸は時間です。青い線は CPU の温度、赤い線は SoC の温度を表しています。このグラフと、「10.13.5.1. サーマルシャットダウン温度の確認」で得たサーマルシャットダウン温度を見比べると、CPU に負荷をかけた際であっても SoC の温度は 60°C 前後ほどまでしか上がらず、この条件で動く Armadillo が温度的にどれほど余裕を持っているかをひと目で確認できます。

### 10.13.5.3. CPU 使用率の確認

atmark-thermal-profiler は、時間毎の温度だけでなく CPU 使用率と CPU 使用率の高いプロセスについても取得して記録します。CPU 使用率については thermal\_profile.csv の CPU\_1~CPU\_5 列と、USE\_1~USE\_5 列を参照してください。各列については「表 10.7. thermal\_profile.csv の各列の説明」にまとまっています。

一般的に CPU 使用率が高くなると、CPU 周辺の温度も高くなります。そのため、測定した温度が高い場合は、CPU 使用率の高いプロセスに注目して、CPU を無駄に使用している意図しない処理が行なわれていないかなどを確認することをおすすめします。

## 10.14. eMMC の GPP(General Purpose Partition) を利用する

GPP に squashfs イメージを書き込み、Armadillo の起動時に自動的にマウントする方法を紹介します。

### 10.14.1. squashfs イメージを作成する

この作業は ATDE 上で行います。

squashfs-tools パッケージに含まれている mkfsquashfs コマンドを使用して squashfs イメージを作成します。

```
[ATDE]$ mkdir sample
[ATDE]$ echo "complete mounting squashfs on eMMC(GPP)" > sample/README
[ATDE]$ mksquashfs sample squashfs.img
```

図 10.174 squashfs イメージの作成

## 10.14.2. squashfs イメージを書き込む

以降の作業は Armadillo 上で行います。

「10.14.1. squashfs イメージを作成する」で作成した squashfs イメージを、USB メモリ利用するなどして Armadillo-640 にコピーし、GPP に書き込みます。



ユーザー領域として使用可能な GPP は /dev/mmcblk0gp3 のみです。

GPP への書き込みを行う際は、誤って /dev/mmcblk0gp0 などに書き込みを行わないよう、十分に注意してください。

```
[armadillo]# mount /dev/sda1 /mnt
[armadillo]# dd if=/mnt/squashfs.img of=/dev/mmcblk0gp3 conv=fsync
[armadillo]# umount /mnt
```

## 10.14.3. GPP への書き込みを制限する

GPP の全ブロックに対して Temporary Write Protection をかけることにより、GPP への書き込みを制限することができます。Temporary Write Protection は電源を切断しても解除されません。

Temporary Write Protection をかけるには、mmc-utils パッケージに含まれている mmc コマンドを使用します。

```
[armadillo]# apk add mmc-utils
```

図 10.175 mmc-utils のインストール

GPP の全ブロックに対して Temporary Write Protection をかけるには、次のようにコマンドを実行します。

```
[armadillo]# mmc writeprotect user get /dev/mmcblk0gp3 ❶
Write Protect Group size in blocks/bytes: 16384/8388608
Write Protect Groups 0-0 (Blocks 0-16383), No Write Protection
[armadillo]# mmc writeprotect user set temp 0 16384 /dev/mmcblk0gp3 ❷
```

図 10.176 eMMC の GPP に Temporary Write Protection をかける

- ❶ /dev/mmcblk0gp3 のブロック数を確認します。コマンドの出力を見ると /dev/mmcblk0gp3 が 16384 ブロックあることがわかります。

- ② /dev/mmcblk0gp3 の全ブロックに Temporary Write Protection をかけます。



Temporary Write Protection を解除するには、次のコマンド実行します。

```
[armadillo]# mmc writeprotect user set none 0 16384 /dev/mmcblk0gp3
```

#### 10.14.4. 起動時に squashfs イメージをマウントされるようにする

/etc/fstab を変更し、起動時に squashfs イメージがマウントされるようにします。

```
[armadillo]# mkdir -p /opt/sample ①
[armadillo]# persist_file /opt/sample/
[armadillo]# vi /etc/fstab
:
:(省略)
:
/dev/mmcblk0gp3 /opt/sample squashfs defaults,nofail 0 0 ②
[armadillo]# persist_file /etc/fstab
```

- ① squashfs イメージをマウントするディレクトリを作成します
- ② 最終行にこの行を追加します。これで、/dev/mmcblk0gp3 が /opt/sample にマウントされるようになります。

Armadillo の再起動後、/opt/sample/README の内容が正しければ完了です。

```
[armadillo]# reboot
:
:(省略)
:
[armadillo]# ls /opt/sample
README
[armadillo]# cat /opt/sample/README
complete mounting squashfs on eMMC(GPP)
```

### 10.15. eFuse を変更する

Armadillo-610 で採用している CPU (i.MX6ULL) には、一度しか書き込むことのできない eFuse が搭載されています。eFuse には、CPU がブートする時の設定や MAC アドレスなどが書かれます。Armadillo-610 は組み込み機器を作り込むエンジニアを対象にした製品ですので、eFuse もユーザーに開放し、細かな制御を可能にしています。しかし eFuse はその性質上、一度書き間違えると直すことができません。十分に注意してください。



eFUSE は一度書き込むと元に戻すことができません。eFUSE の設定によっては Armadillo-610 が正常に動作しなくなる可能性がありますので、

書き込みを行う際には細心の注意を払うようお願いいたします。eFUSE の設定によって異常が起こった場合は保証対象外となります。

MAC アドレスは Armadillo-610 の出荷時に書き込まれているので、新たに書き込む必要はありません。この章では U-Boot を使って eFuse の書き換えを行い、ブートモードを制御する方法を説明します。

eFuse を変更する場合は、必ず「i.MX 6ULL Applications Processor Reference Manual [https://www.nxp.com/docs/en/reference-manual/IMX6ULLRM.pdf]」を参照してください。重要な章は、以下の 4 つです。

- ・ Chapter 5: Fusemap
- ・ Chapter 8: System Boot
- ・ Chapter 37: On-Chip OTP Controller
- ・ Chapter 58: Ultra Secured Digital Host Controller

以降、本章では i.MX 6ULL Applications Processor Reference Manual を「リファレンスマニュアル」と呼びます。



章番号や章タイトルは、i.MX 6ULL Applications Processor Reference Manual Rev. 1, 11/2017 現在の情報です。異なるリビジョンのリファレンスマニュアルでは、章番号およびタイトルが異なる場合があります。

## 10.15.1. ブートモード

i.MX6ULL にはブートモードを決める `BOOT_MODE0` と `BOOT_MODE1` というピンがあります。Armadillo-610 では、`BOOT_MODE0` は 0、`BOOT_MODE1` は 1 となるよう回路が設計されており、ブートモードは必ず **Internal Boot** モードとなります。

### 10.15.1.1. Internal Boot モード

Internal Boot モードでは、on-chip boot ROM に書き込まれているコードが実行し、ブート可能なデバイスを検索します。リファレンスマニュアル「8.5 Boot devices (internal boot)」に、i.MX6ULL がブートできるデバイスの一覧が記載されています。Armadillo-610 では、そのうちオンボード eMMC と microSD カードに対応しています。

Internal Boot モードでは、GPIO によって eFuse の設定を上書き (override) できるようになっています。この機能は eFuse の `BT_FUSE_SEL` が 0 の場合のみ有効となります。eFuse の設定とは異なり何度も再設定できる点では便利ですが、override に対応したピンには i.MX6ULL の電源投入時に決まった信号を入力しておかなければいけないため、ハードウェア設計上は不便になります。

Armadillo-610 では、GPIO による override を利用することで、仕様が確定していない段階ではブートデバイスを自由に何度も切り替えることを可能にしつつ、`BT_FUSE_SEL` を 1 にして GPIO による override を無効化することで、仕様が確定した段階では自由なハードウェア設計が可能になるよう配慮しています。また、GPIO による override を無効化することで、フィールドに出した製品が悪意ある人によって意図していないブートをし、被害が出ることを防ぐことができます。(もちろん、ブート後に root アカウントを乗っ取られるような作りでは、意味がありませんが…)

## 10.15.2. ブートデバイス

Internal Boot モードでは、GPIO によって eFuse の設定を上書き (override) できるようになってると紹介しましたが、Armadillo-610 では、Armadillo-610 拡張ボードの JP1 はまさにこの機能を使っています。JP1 は BJP1(Armadillo-610 CON2\_42 ピン) に接続されており、LCD1\_DATA05 と LCD1\_DATA11 の制御をしていますが、これらのピンはそれぞれ BOOT\_CFG1[5] と BOOT\_CFG2[3] を override しています。「8.3.2 GPIO boot overrides」の表「8-3. GPIO override contact assignments」を確認してください。

ややこしい事に、この BOOT\_CFG で始まる eFUSE は、リファレンスマニュアルの中では eFuse のアドレスでも表記されています。BOOT\_CFG1 は eFuse のアドレスで言うと 0x450 の下位 8 bit つまり 0x450[7:0] であり、BOOT\_CFG2 は上位 8 bit つまり 0x450[15:8] にあたります。これは「5.1 Boot Fusemap」の表「5-5. SD/eSD Boot Fusemap」または表「5-6. MMC/eMMC Boot Fusemap」を確認することでわかります。

さらにややこしい事に、eFuse を書き込む場合にはこれら全ての値が使えず、On-Chip OTP Controller の bank と word の値が必要になります。これらの値は リファレンスマニュアルの「On-Chip OTP Controller」を参照してください。後で出てきますが Boot From Fuses で使用する BT\_FUSE\_SEL という eFuse のように GPIO による override ができないものもあります。

表 10.8 GPIO override と eFuse

信号名	eFuse 名	eFuse アドレス	OCOTP 名	Bank	Word
LCD1_DATA05	BOOT_CFG1[5]	0x450[5]	OCOTP_CFG4	0	5
LCD1_DATA11	BOOT_CFG2[3]	0x450[11]	OCOTP_CFG4	0	5
N/A	BT_FUSE_SEL	0x460[4]	OCOTP_CFG5	0	6

Armadillo-610 では SD カード または eMMC からのブートになるので、ブートデバイスを選択する eFuse BOOT\_CFG1[7:4] は、010x または 011x になります。

リファレンスマニュアル「8.5.3.1 Expansion device eFUSE configuration」には、さらに詳しく SD/MMC デバイスの設定について記載されています。テーブル「8-15. USDHC boot eFUSE descriptions」によれば、eFuse の 0x450[7:6] が 01 の場合に SD/MMC デバイスからブートすることを決めています。さらに 0x450[5] が 0 なら SD が、0x450[5] が 1 なら MMC が選択されます。つまり、4 から 7 bit までの間で 5 bit 目だけが MMC か SD かを決めています。BOOT\_CFG1[5] が 0 の場合はコントローラーは SD デバイスが繋がっている前提で、BOOT\_CFG1[5] が 1 の場合は MMC デバイスが繋がっている前提で動作します。

i.MX6ULL には、SD/MMC のコントローラーである uSDHC が 2 つ搭載されています。Armadillo-610 では、eMMC が uSDHC1 に、microSD カードが uSDHC2 に接続されています。ブート時にどちらのコントローラーからブートするかを決めている eFuse が 0x450[12:11] です。0x450[12:11] が 00 であれば uSDHC1 つまりオンボード eMMC から、01 であれば uSDHC2 つまり microSD カードからブートします。言い換えると Armadillo-610 でオンボード eMMC からブートしたい場合は、0x450[5] を 1 に、0x450[12:11] を 00 にします。逆に microSD カードから起動したい場合は 0x450[5] を 0 に、0x450[12:11] を 01 にします。

表 10.9 ブートデバイスと eFuse

ブートデバイス	eFuse 0x450[5]	0x450[12:11]
オンボード eMMC	1	00
microSD カード	0	01

## 10.15.3. eFuse の書き換え

Armadillo-610 では、U-Boot のコマンドによって eFuse の書き換えをサポートしています。「4.6. スライドスイッチの設定について」を参照して U-Boot を保守モードで起動してください。

eFuse の書き換えは、 fuse コマンドを使います。



U-Boot の fuse コマンドのソースコードは、以下の 2 つです。

- ・ cmd/fuse.c
- ・ drivers/misc/mxc\_ocotp.c

```
=> help fuse
fuse - Fuse sub-system

Usage:
fuse read <bank> <word> [<cnt>] - read 1 or 'cnt' fuse words,
    starting at 'word'
fuse sense <bank> <word> [<cnt>] - sense 1 or 'cnt' fuse words,
    starting at 'word'
fuse prog [-y] <bank> <word> <hexval> [<hexval>...] - program 1 or
    several fuse words, starting at 'word' (PERMANENT)
fuse override <bank> <word> <hexval> [<hexval>...] - override 1 or
    several fuse words, starting at 'word'
=>
```

fuse read            eFuse の値を Shadow Register から読み出します。i.MX6ULL の eFuse は、すべて Shadow Register を持ち、起動時に eFuse から Shadow Register に値がコピーされます。詳しくはリファレンスマニュアル「37.3.1.1 Shadow Register Reload」を確認してください。

fuse sense           eFuse の値を eFuse から読み出します

fuse prog            eFuse の値を書き換えます

fuse コマンドは、 bank 、 word 、 cnt 、 hexval を引数に取ります。

bank                eFuse のバンク番号

word                eFuse のワード番号

cnt                 eFuse を読み出す個数

hexval             書き込む値

## 10.15.4. eFuse の設定によるブートデバイスの選択

### 10.15.4.1. BT\_FUSE\_SEL

eFuse の設定によるブートデバイスの選択を可能にするには、 eFuse に書き込んだ値が正しいことを i.MX6ULL に教える必要があります。そのための eFuse が BT\_FUSE\_SEL (0x460[4]) です。Armadillo-610 では、このビットが 1 であれば、GPIO による override が無効になり eFuse の設定にしたがってブートデバイスが選択されるようになります。

### 10.15.4.2. eMMC からのブートに固定

オンボード eMMC からだけブートさせたい場合は、ブートデバイスの種類で MMC と、コントローラで uSDHC1 を選択することで可能です。忘れずに BT\_FUSE\_SEL を 1 にします。

オンボード eMMC のスペックは、以下の通りです。リファレンスマニュアル 8.5.3 Expansion device および 表「5-6. MMC/eMMC Boot Fusemap」を確認してください。「可変」列が「不」となっている値は、変更しないでください。例えば、オンボード eMMC は 1.8 V に対応していません。bit 9 の SD Voltage Selection で 1 の 1.8 V では動作しません。

表 10.10 オンボード eMMC のスペック

名前	Bit	eFuse	値	bit 列	可変
BOOT_CFG2	[15:13]	Bus Width	8 bit	010	不
	[12:11]	Port Select	uSDHC1	00	不
	[10]	Boot Frequencies	500 / 400 MHz	00	可
	[9]	SD Voltage Selection	3.3 V	0	不
	[8]	-	-	0	-
BOOT_CFG1	[7:5]	eMMC	-	011	不
	[4]	Fast Boot	Regular	0	可
	[3]	SD/MMC Speed	High	0	不
	[2]	Fast Boot Acknowledge Disable	Enabled	0	可
	[1]	SD Power Cycle Enable	Enabled	1	可
	[0]	SD Loopback Clock Source Sel	SD Pad	0	不

値を見易いように、BOOT\_CFG2 を上にしてあります。BOOT\_CFG1 と BOOT\_CFG2 は、OC0TP\_CFG4 にマップされており Bank 0 Word 5 です。つまり 010000000 01100010 の 16 bit (0x4062) を Bank 0 Word 5 に書き込めば良いことが分ります。BOOT\_CFG3 と BOOT\_CFG4 はここでは無視します。

BT\_FUSE\_SEL は Bank 0 Word 6 の 4 bit 目になるので 0x10 を書き込みます。

```

=> fuse read 0 5
Reading bank 0:

Word 0x00000005: 00000000
=> fuse prog 0 5 0x4060
Programming bank 0 word 0x00000005 to 0x00004060...
Warning: Programming fuses is an irreversible operation!
        This may brick your system.
        Use this command only if you are sure of what you are doing!

Really perform this fuse programming? <y/N>
y
=> fuse read 0 6
Reading bank 0:

Word 0x00000006: 00000000
=> fuse prog -y 0 6 0x10
Programming bank 0 word 0x00000006 to 0x00000010...
=> fuse read 0 6
    
```



Reading bank 0:

Word 0x00000006: 00000010

(電源入れなおしても、SD からブートしない)



fuse prog にオプション `-y` を付けると 「Really perform this fuse programming? <y/N>」 と聞かれません。

これで eMMC からしか起動しない Armadillo-610 ができあがりました。



eMMC からしか起動しないので、あやまって eMMC に書き込まれている U-Boot を消してしまうと、二度と起動しないようになります。注意してください。



eMMC Fast Boot 機能を使う場合や Power Cycle を Enable にする場合は、当該ビットを 1 に変更してください。

同じ要領で、SD からだけしかブートしないようにすることも可能です。しかし eFuse によるブートデバイスの固定は、意図しないブートを防ぐことが目的です。Armadillo-610 で microSD からのブートに固定することは可能ですが、別の microSD カードを挿入されてしまうと、その別の microSD カードからブートしてしまうので目的を達成できません。理解してお使いください。

#### 10.15.4.3. eFuse のロック

書き込んだ eFuse の値を変更されてしまっては、Boot From Fuse モードにしている意味がありません。i.MX6ULL では eFuse を変更できなくするビットも用意されています。

リファレンスマニュアル「5.3 Fusemap Descriptions Table」を確認してください。

# 11. 動作ログ

## 11.1. 動作ログについて

Armadillo-640 ではシステムが出力するログの一部は、一般的な /var/log ディレクトリではなく、/var/at-log ディレクトリに出力されます。/var/at-log は、ルートファイルシステムとは別のパーティションになっているので、ルートファイルシステムに障害が発生した場合でも、/var/at-log のパーティションが無事であれば、ログファイルを取り出して、不具合等の解析に利用することができます。

## 11.2. 動作ログを取り出す

ログファイルは /var/at-log ディレクトリ内に atlog というファイル名で作成されているので、これを任意のディレクトリにコピーすることで取り出せます。もし、eMMC 上のルートファイルシステムが壊れてしまい起動できない場合は、microSD カードから起動することでログファイルを取り出すことができます。

## 11.3. ログファイルのフォーマット

ログファイルの内容はテキストデータであり、以下のようなフォーマットになっています。

```
日時 armadillo ログレベル 機能: メッセージ
```

図 11.1 動作ログのフォーマット

## 11.4. ログ用パーティションについて

ログ出力先である /var/at-log ディレクトリには、GPP である /dev/mmcblk0gp1 パーティションがマウントされています。このパーティションに論理的な障害が発生した場合は、/dev/mmcblk0gp1 のデータを /dev/mmcblk0gp2 にコピーし、/dev/mmcblk0gp1 は FAT ファイルシステムでフォーマットされます。このパーティションの障害チェックはシステム起動時に自動的に実行されます。

# 12. 製品機能

本章では、Armadillo-640 で利用できる各種機能の仕様について説明します。

## 12.1. UART

Armadillo-640 のシリアルは、i.MX6ULL の UART (Universal Asynchronous Receiver/Transmitter) を利用しています。Armadillo-640 の標準状態では、UART1 (CON9) をコンソールとして利用しています。

- フォーマット
- ・ データビット長: 7 or 8 ビット
  - ・ ストップビット長: 1 or 2 ビット
  - ・ パリティ: 偶数 or 奇数 or なし
  - ・ フロー制御: CTS/RTS or XON/XOFF or なし
  - ・ 最大ボーレート: 230.4kbps

- 関連するソースコード
- ・ drivers/tty/serial/imx.c
  - ・ drivers/tty/serial/imx\_earlycon.c

- Device Tree ドキュメント
- ・ Documentation/devicetree/bindings/serial/fsl-imx-uart.yaml

デバイスファイル

シリアルインターフェース	デバイスファイル
UART1	/dev/ttymx0
UART3	/dev/ttymx2
UART5	/dev/ttymx4

カーネルコンフィギュレーション

```

Device Drivers --->
Character devices --->
  [*] Enable TTY <TTY>
      Serial drivers --->
        <*> IMX serial port support <SERIAL_IMX>
        <*> Console on IMX serial port <SERIAL_IMX_CONSOLE>
        [*] Earlycon on IMX serial port <SERIAL_IMX_EARLYCON>
    
```

## 12.2. Ethernet

Armadillo-640 の Ethernet (LAN) は、i.MX6ULL の ENET(10/100-Mbps Ethernet MAC)を利用しています。

Armadillo-640 では、LAN インターフェース(CON7)が ENET を利用しています。

- 機能
- ・ 通信速度: 100Mbps (100BASE-TX), 10Mbps (10BASE-T)
  - ・ 通信モード: Full-Duplex (全二重), Half-Duplex (半二重)

- ・ Auto Negotiation サポート
  - ・ キャリア検知サポート
  - ・ リンク検出サポート
- 関連するソースコード
- ・ drivers/net/ethernet/freescale/fec\_main.c
  - ・ drivers/net/phy/smsc.c
  - ・ drivers/net/mdio/of\_mdio.c
- Device Tree ドキュメント
- ・ Documentation/devicetree/bindings/net/fsl-fec.txt
  - ・ Documentation/devicetree/bindings/net/mdio-mux.txt
  - ・ Documentation/devicetree/bindings/net/ethernet-phy.yaml
  - ・ Documentation/devicetree/bindings/net/smsc-lan87xx.txt
- ネットワークデバイス
- ・ eth0
- カーネルコンフィギュレーション

```

Device Drivers --->
[*] Network device support ---> <NETDEVICES>
  [*] Ethernet driver support ---> <ETHERNET>
    [*] Freescale devices <NET_VENDOR_FREESCALE>
    <*> FEC ethernet controller (of ColdFire and some i.MX CPUs)
        <FEC>
  -*- PHY Device support and infrastructure ---> <PHYLIB>
    {*} SMSC PHYs <SMSC_PHY>
    
```

## 12.3. SD ホスト

Armadillo-640 の SD ホストは、i.MX6ULL の uSDHC (Ultra Secured Digital Host Controller) を利用しています。Armadillo-640 では、オンボード microSD コネクタ (CON1) が uSDHC2 を利用しています。

- 機能
- ・ カードタイプ: microSD/microSDHC/microSDXC/microSDIO
  - ・ バス幅: 1bit or 4bit
  - ・ スピードモード: Default Speed (24.75MHz), High Speed (49.5MHz)
  - ・ カードディテクトサポート
- デバイスファイル
- ・ /dev/mmcblk1
- 関連するソースコード
- ・ drivers/mmc/host/sdhci-esdhc-imx.c
  - ・ drivers/mmc/host/sdhci-of-esdhc.c
- Device Tree ドキュメント
- ・ Documentation/devicetree/bindings/mmc/fsl-imx-esdhc.yaml
  - ・ Documentation/devicetree/bindings/mmc/mmc-controller.yaml

カーネルコンフィギュレーション

```

Device Drivers --->
<*> MMC/SD/SDIO card support ---> <MMC>
<*> MMC block device driver <MMC_BLOCK>
(8) Number of minors per block device <MMC_BLOCK_MINORS>
*** MMC/SD/SDIO Host Controller Drivers ***
<*> Secure Digital Host Controller Interface support <MMC_SDHCI>
<*> SDHCI platform and OF driver helper <MMC_SDHCI_PLTFM>
    
```

## 12.4. USB ホスト

Armadillo-640 の USB ホストは、i.MX6ULL の USB-PHY (Universal Serial Bus 2.0 Integrated PHY) および USB (Universal Serial Bus Controller) を利用しています。Armadillo-640 では、USB ホストインターフェース (CON5) が OTG1 (下段) と OTG2 (上段) を利用しています。OTG2 は CON5 と CON9 と排他利用になっており、外部からの信号で切り替えることができるようになっています。詳しくは「14.2.5. CON5(USB ホストインターフェース)」を参照してください。

- |                    |  |
|--------------------|--|
| 機能                 | <ul style="list-style-type: none"> <li>・ Universal Serial Bus Specification Revision 2.0 準拠</li> <li>・ Enhanced Host Controller Interface (EHCI) 準拠</li> <li>・ 転送レート: USB2.0 High-Speed (480Mbps), Full-Speed (12Mbps), Low-Speed (1.5Mbps)</li> </ul> |
| デバイスファイル           | <ul style="list-style-type: none"> <li>・ メモリデバイスの場合は、デバイスを認識した順番で/dev/sdN (N は 'a' からの連番)となります。</li> <li>・ I/O デバイスの場合は、ファンクションに応じたデバイスファイルとなります。</li> </ul>   |
| 関連するソースコード         | <ul style="list-style-type: none"> <li>・ drivers/usb/chipidea/</li> <li>・ drivers/usb/host/ehci-hcd.c</li> <li>・ drivers/usb/phy/of.c</li> <li>・ drivers/usb/phy/phy-generic.c</li> <li>・ drivers/usb/phy/phy.c</li> </ul>                             |
| Device Tree ドキュメント | <ul style="list-style-type: none"> <li>・ Documentation/devicetree/bindings/usb/ci-hdrc-usb2.txt</li> <li>・ Documentation/devicetree/bindings/phy/mxs-usb-phy.txt</li> </ul>  |

カーネルコンフィギュレーション

```

Device Drivers --->
[*] USB support ---> <USB_SUPPORT>
<*> Support for Host-side USB <USB>
*** USB Host Controller Drivers ***
<*> EHCI HCD (USB 2.0) support <USB_EHCI_HCD>
<*> ChipIdea Highspeed Dual Role Controller <USB_CHIPIDEA>
[*] ChipIdea device controller <USB_CHIPIDEA_UDC>
[*] ChipIdea host controller <USB_CHIPIDEA_HOST>
    
```

```

USB Physical Layer drivers --->
<*> Freescale MXS USB PHY support <USB_MXS_PHY>
    
```

## 12.5. リアルタイムクロック

Armadillo-640 のリアルタイムクロックは、i.MX6ULL の RTC 機能を利用しています。

- 機能
  - ・ アラーム割り込みサポート
- デバイスファイル
  - ・ /dev/rtc ( /dev/rtc0 へのシンボリックリンク)
  - ・ /dev/rtc0
- 関連するソースコード
  - ・ drivers/rtc/rtc-snvs.c
- Device Tree ドキュメント
  - ・ Documentation/devicetree/bindings/crypto/fsl-sec4.txt
- カーネルコンフィギュレーション
 

```

Device Drivers --->
[*] Real Time Clock ---> <RTC_CLASS>
<*> Freescale SNVS RTC support <RTC_DRV_SNVS>
                
```

アラーム割り込みは、デバイスファイル経由で利用することができます。

詳細な情報については、Linux カーネルのソースコードに含まれているドキュメント (Documentation/admin-guide/rtc.rst) やサンプルプログラム (tools/testing/selftests/rtc/rtctest.c) を参照してください。

## 12.6. LED

Armadillo-640 に搭載されているユーザー LED には、GPIO が接続されています。

Linux では、GPIO 接続用 LED ドライバ (leds-gpio) で制御することができます。

- sysfs LED クラスディレクトリ
  - ・ /sys/class/leds/red
  - ・ /sys/class/leds/green
  - ・ /sys/class/leds/yellow
- 関連するソースコード
  - ・ drivers/leds/leds-gpio.c
- Device Tree ドキュメント
  - ・ Documentation/devicetree/bindings/leds/leds-gpio.yaml
- カーネルコンフィギュレーション
 

```

Device Drivers --->
[*] LED Support ---> <NEW_LEDS>
<*> LED Support for GPIO connected LEDs <LEDS_GPIO>
                
```

## 12.7. ユーザースイッチとイベント信号

Armadillo-640 に搭載されているユーザースイッチには、GPIO が接続されています。

Linux では、ユーザー空間でイベント(Press/Release)を検出することができます。Linux では、GPIO 接続用キーボードドライバ(gpio-keys)で制御することができます。

ユーザースイッチと信号には、次に示すキーコードが割り当てられています。

表 12.1 キーコード

ユーザースイッチ	キーコード	イベントコード	X11 キーコード
SW1	KEY_ENTER	28	Return

- デバイスファイル            ・ /dev/input/by-path/platform-gpio-keys-event <sup>[1]</sup>
- 関連するソースコード      ・ drivers/input/keyboard/gpio\_keys.c
- Device Tree ドキュメント   ・ Documentation/devicetree/bindings/input/gpio-keys.yaml
- カーネルコンフィギュレーション

```

Device Drivers --->
Input device support --->
  *- Generic input layer (needed for keyboard, mouse, ...)
                                     <INPUT>
  [*] Keyboards --->                                     <INPUT_KEYBOARD>
  <*- GPIO Buttons                                     <KEYBOARD_GPIO>
    
```

## 12.8. I2C

Armadillo-640 の I2C インターフェースは、i.MX6ULL の I2C(I2C Controller) および GPIO を利用した I2C バスドライバ(i2c-gpio)を利用します。また、i2c-gpio を利用することで、I2C バスを追加することができます。

Armadillo-640 で利用している I2C バスと、接続される I2C デバイスを次に示します。

表 12.2 I2C デバイス

I2C バス	I2C デバイス	
	アドレス	デバイス名
0(I2C1)	0x08	PF3000
4(I2C-GPIO)	0x51	GT800X480A-1013P <sup>[a]</sup>
5(I2C-GPIO1)	0x32	NR3225SA <sup>[b]</sup>

<sup>[a]</sup>CON11 に LCD オプションセット(7 インチタッチパネル WVGA 液晶)を接続した場合。

<sup>[b]</sup>CON9 に Armadillo-600 シリーズ WLAN オプションモジュールまたは Armadillo-600 シリーズ RTC オプションモジュールを接続した場合。

Armadillo-640 の標準状態では、CONFIG\_I2C\_CHARDEV が有効となっているためユーザードライバで I2C デバイスを制御することができます。ユーザードライバを利用する場合は、Linux カーネルで I2C デバイスに対応するデバイスドライバを無効にする必要があります。

- 機能                            ・ 最大転送レート: 400kbps
- デバイスファイル            ・ /dev/i2c-0 (I2C1) <sup>[2]</sup>
- ・ /dev/i2c-4 (I2C-GPIO)

<sup>[1]</sup>USB キーボードなどを接続してインプットデバイスを追加している場合は、番号が異なる可能性があります

<sup>[2]</sup>Armadillo-640 の標準状態ではデバイスファイルが作成されません。

- 関連するソースコード      ・ drivers/i2c/busses/i2c-imx.c
- Device Tree ドキュメント    ・ Documentation/devicetree/bindings/i2c/i2c-gpio.yaml
- ・ Documentation/devicetree/bindings/i2c/i2c-imx.yaml

カーネルコンフィギュレーション

```

Device Drivers --->
I2C support --->
  *- I2C support                                     <I2C>
    <*> I2C device interface                         <I2C_CHARDEV>
  I2C Hardware Bus support --->
    <*> GPIO-based bitbanging I2C                   <I2C_GPIO>
    <*> IMX I2C interface                           <I2C_IMX>
    
```

## 12.9. パワーマネジメント

Armadillo-640 のパワーマネジメント機能は、Linux の SPM(System Power Management)および DPM(Device Power Management)を利用しています。パワーマネジメント状態を省電力モードに遷移させることにより、Armadillo-640 の消費電力を抑えることができます。

パワーマネジメント状態を省電力モードに遷移させると、アプリケーションの実行は一時停止し、Linux カーネルはサスペンド状態となります。起床要因が発生すると、Linux カーネルのリジューム処理が行われた後、アプリケーションの実行を再開します。

sysfs ファイル                ・ /sys/power/state

関連するソースコード    ・ kernel/power/

カーネルコンフィギュレーション


```

Power management options --->
[*] Suspend to RAM and standby                       <SUSPEND>
-* Device power management core functionality       <PM>
    
```

Armadillo-640 が対応するパワーマネジメント状態と、/sys/power/state に書き込む文字列の対応を次に示します。

表 12.3 対応するパワーマネジメント状態

パワーマネジメント状態	文字列	説明
Suspend-to-RAM	mem	Suspend-to-Idle よりも消費電力を抑えることができる
Power-On Suspend	standby	Suspend-to-RAM よりも短時間で復帰することができ、Suspend-to-Idle よりも消費電力を抑えることができる
Suspend-to-Idle	freeze	suspend-to-ram よりも短時間で復帰することができる



サスペンド状態を 128 秒以上継続する場合は、Suspend-to-RAM か +Power-On Suspend+を利用してください。

+Suspend-to-Idle+を利用している状態で 128 秒経過すると再起動してしまいます。



起床要因として利用可能なデバイスは次の通りです。

UART1 (CON9)	起床要因	データ受信	
	有効化		<pre>[armadillo ~]# echo enabled &gt; /sys/class/tty/ttymx0/power/wakeup</pre>
USB OTG1 (下段)	起床要因	USB デバイスの挿抜	
	有効化		<pre>[armadillo ~]# echo enabled &gt; /sys/bus/platform/devices/2184000.usb/power/wakeup [armadillo ~]# echo enabled &gt; /sys/bus/platform/drivers/ci_hdrc/ci_hdrc.0/power/wakeup [armadillo ~]# echo enabled &gt; /sys/bus/platform/drivers/ci_hdrc/ci_hdrc.0/usb1/power/wakeup</pre>
USB OTG2 (上段)	起床要因	USB デバイスの挿抜	
	有効化		<pre>[armadillo ~]# echo enabled &gt; /sys/bus/platform/devices/2184200.usb/power/wakeup [armadillo ~]# echo enabled &gt; /sys/bus/platform/drivers/ci_hdrc/ci_hdrc.1/power/wakeup [armadillo ~]# echo enabled &gt; /sys/bus/platform/drivers/ci_hdrc/ci_hdrc.1/usb2/power/wakeup</pre>
RTC(i.MX6ULL)	起床要因	アラーム割り込み	
	有効化		デフォルトで有効化されています

## 12.10. GPIO

Armadillo-640 の GPIO は、i.MX6ULL の GPIO(General Purpose Input/Output)を利用しています。

- 関連するソースコード      ・ drivers/gpio/gpio-mxc.c
- Device Tree ドキュメント    ・ Documentation/devicetree/bindings/gpio/fsl-imx-gpio.yaml

デバイスファイル

デバイスファイル	GPIO 番号
/dev/gpiochip0	0~31(GPIO1_IO00~GPIO1_IO31)
/dev/gpiochip1	32~63(GPIO2_IO00~GPIO2_IO31)
/dev/gpiochip2	64~95(GPIO3_IO00~GPIO3_IO31)
/dev/gpiochip3	96~127(GPIO4_IO00~GPIO4_IO31)
/dev/gpiochip4	128~159(GPIO5_IO00~GPIO5_IO31)

sysfs GPIO クラスディレクトリ `· /sys/class/gpio/`



sysfs GPIO クラスは旧バージョンの Linux カーネルとの互換性維持の為に残っています。新しくアプリケーションを開発する際の利用はおすすめしません。

新しくアプリケーションを開発する場合は、libgpiod パッケージに含まれるアプリケーションまたは Linux カーネルのソースコードに含まれているサンプル(`tools/gpio/`)を参考にしてください。

カーネルコンフィギュレーション

```
Device Drivers --->
[*] GPIO Support ---> <GPIOLIB>
    (512) Maximum number of GPIOs for fast path
                                <GPIOLIB_FASTPATH_LIMIT>

Memory mapped GPIO drivers --->
    -*- i.MX GPIO support <GPIO_MXC>
```

## 12.11. 温度センサー

Armadillo-640 の温度センサーは、i.MX6ULL の TEMPMON(Temperature Monitor)を利用しています。

起動直後の設定では、ARM または SoC の測定温度が 105°C 以上になった場合、Linux カーネルはシステムを停止します。

- 機能 · 測定温度範囲: -40~+105°C
- sysfs thermal クラスディレクトリ · `/sys/class/thermal/thermal_zone0`
- 関連するソースコード · `drivers/thermal/imx_thermal.c`
- Device Tree ドキュメント · `Documentation/devicetree/bindings/thermal/imx-thermal.yaml`

カーネルコンフィギュレーション

```
Device Drivers --->
    -*- Thermal drivers ---> <THERMAL>
        <*> Temperature sensor driver for Freescale i.MX SoCs

<IMX_THERMAL>
```



## 12.12. ウォッチドッグタイマー

Armadillo-640 のウォッチドッグタイマーは、i.MX6ULL の WDOG(Watchdog Timer)を利用しています。


ウォッチドッグタイマーは、U-Boot によって有効化されます。標準状態でタイムアウト時間は 10 秒に設定されます。

何らかの要因でウォッチドッグタイマーのキックができなくなりタイムアウトすると、システムリセットが発生します。

- 関連するソースコード      ・ `drivers/watchdog/imx2_wdt.c`
- Device Tree ドキュメント    ・ `Documentation/devicetree/bindings/watchdog/fsl-imx-wdt.yaml`
- カーネルコンフィギュレーション

```
Device Drivers --->
[*] Watchdog Timer Support --->                                <WATCHDOG>
<*>  IMX2+ Watchdog                                           <IMX2_WDT>
```

ウォッチドッグタイマーの設定変更は、`ioctl` システムコール経由で行うことができます。詳細な情報については、Linux カーネルのソースコードに含まれているドキュメント (`Documentation/watchdog/watchdog-api.rst`) を参照してください。



ウォッチドッグタイマーを停止することはできません。


## 12.13. WLAN

「Armadillo-600 シリーズ WLAN コンボオプションモジュール」および「Armadillo-600 シリーズ BT/TH オプションモジュール WLAN コンボ対応」には、Laird Connectivity 製 Sterling LWB5+ が搭載されています。Sterling LWB5+ の WLAN は、USB2422 を介して「12.4. USB ホスト」に示す OTG2 に接続されています。


- 機能
  - ・ IEEE 802.11a/b/g/n/ac 準拠
  - ・ 最大通信速度(2.4GHz): 150Mbps(理論値)
  - ・ 最大通信速度(5GHz): 433.3Mbps(理論値)
  - ・ 動作モード: インフラストラクチャモード(STA/AP), アドホックモード
  - ・ チャンネル(2.4GHz): 1-13
  - ・ チャンネル(5GHz): 36-48(W52), 52-64(W53), 100-140(W56)
- ネットワークデバイス      ・ `wlan0`
- 関連するソースコード      ・ `drivers/net/wireless/broadcom/brcm80211/brcmfmac/*`
- Device Tree ドキュメント    ・ `Documentation/devicetree/bindings/net/wireless/brcm,bcm43xx-fmac.txt`

```
Device Drivers --->
[*] Network device support --->                                <NETDEVICES>
[*]  Wireless LAN --->                                         <WLAN>
```

```
[*] Broadcom devices <WLAN_VENDOR_BROADCOM>
[*] Broadcom FullMAC WLAN driver <BRCMFMAC>
[*] SDIO bus interface support for FullMAC driver <BRCMFMAC_SDIO>
```



Sterling LWB5+ のファームウェアは、ATDE にインストールされている firmware-brcm80211 パッケージに含まれています。ファームウェアは Linux カーネルイメージ内に改変無く配置されます。firmware-brcm80211 の著作権およびライセンス情報については、ATDE 上で /usr/share/doc/firmware-brcm80211/copyright を参照してください。



WLAN 機能を利用するには、DT overlay の設定が必要です。

```
[armadillo~]# vi /boot/overlays.txt
fdt_overlays=armadillo-640-con9-thread-lwb5plus.dtbo
[armadillo~]# persist_file /boot/overlays.txt
[armadillo~]# reboot
```

## 12.14. BT


「Armadillo-600 シリーズ WLAN コンボオプションモジュール」および「Armadillo-600 シリーズ BT/TH オプションモジュール WLAN コンボ対応」には、Laird Connectivity 製 Sterling LWB5+ が搭載されています。Sterling LWB5+ の WLAN は、USB2422 を介して「12.4. USB ホスト」に示す OTG2 に接続されています。

機能, デバイス            • hci0

関連するソースコード   • drivers/bluetooth/hci\_bcm.c

カーネルコンフィギュレーション

```
[*] Networking support ---> <NET>
[*] Bluetooth subsystem support ---> <BT>
Bluetooth device drivers --->
[*] Broadcom protocol support <BT_HCIUART_BCM>
```



BT 機能を利用するには、DT overlay の設定が必要です。

```
[armadillo~]# vi /boot/overlays.txt
fdt_overlays=armadillo-640-con9-thread-lwb5plus.dtbo
[armadillo~]# persist_file /boot/overlays.txt
[armadillo~]# reboot
```


## 12.15. LCD

Armadillo-640 の LCD ホストは、i.MX6ULL の eLCDIF(Enhanced LCD Interface)を利用しています。CON11 に LCD オプションセット(7 インチタッチパネル WVGA 液晶)を接続した場合に利用できます。

- 機能 , デバイス
  - ・ /dev/dri/card0 (DRM)
  - ・ /dev/fb0 (フレームバッファ)
- 関連するソースコード
  - ・ drivers/gpu/drm/mxsfb/
- Device Tree ドキュメント
  - ・ Documentation/devicetree/bindings/display/mxsfb.txt

カーネルコンフィギュレーション

```
Device Drivers --->
Graphics support --->
  <M> i.MX (e)LCDIF LCD controller                                <DRM_MXSFB>
```



LCD 機能を利用するには、DT overlay の設定が必要です。

```
[armadillo~]# vi /boot/overlays.txt
fdt_overlays=armadillo-640-lcd70ext-l00.dtbo
[armadillo~]# persist_file /boot/overlays.txt
[armadillo~]# reboot
```

# 13. ソフトウェア仕様

---

## 13.1. SWUpdate

### 13.1.1. SWUpdate とは

SWUpdate は、デバイス上で実行されるプログラムで、ネットワーク/ストレージ経由でデバイスのソフトウェアを更新することができます。Stefano Babic, DENX software engineering, Germany によってオープンソースで開発が進められています。

Armadillo-640 では、SWUpdate を利用することで次のような機能を実現しています。

- ・ A/B アップデート(アップデートの 2 面化)
- ・ リカバリーモード
- ・ ソフトウェアの圧縮、暗号化、署名付与
- ・ Web サーバー機能
- ・ hawkBit への対応
- ・ ダウングレードの禁止

### 13.1.2. swu パッケージ

swu パッケージは、SWUpdate 独自のソフトウェアの配布フォーマットです。SWUpdate では、1 回のアップデートは 1 つの swu パッケージで行われます。

swu パッケージには、次のような様々なものを含めることができます。

- ・ アップデート対象のイメージファイル
- ・ アップデート対象のイメージファイルのチェックサム
- ・ アップデート前後に実行するスクリプト
- ・ 書き込み先ストレージの情報
- ・ U-Boot 環境変数の書き換え情報
- ・ ソフトウェアのバージョン情報
- ・ etc...

### 13.1.3. A/B アップデート(アップデートの 2 面化)

A/B アップデートは、Flash メモリにパーティションを 2 面確保し、アップデート時には交互に利用する仕組みです。

常に使用していない方のパーティションを書き換えるため次の特徴を持ちます。

- ・ ○ アップデートによって動作中のソフトウェアは破壊されない
- ・ ○ 書き込みが電源断などで中断後しても、すぐに復帰出来る
- ・ ○ 機器が動作中に書き込みが出来る
- ・ × 使用 Flash メモリ量が増える

### 13.1.4. ロールバック (リカバリー)

システムが起動できなくなった際に、自動的にアップデート前のシステムにロールバックします。

ロールバック状態の確認は「10.8.3. ロールバック状態の確認」を参照してください。

ロールバックする条件は次の通りです:

- ・ rootfs にブートに必要なファイルが存在しない場合 (/boot/Image, /boot/armadillo.dtb)
- ・ 3 回起動を試して「bootcount」サービスが1度も起動できなかった場合は、次の起動時にロールバックします。

bootcount 機能は u-boot の「upgrade\_available」変数で管理されています。bootcount 機能を利用しないようにするには、「10.8.6. u-boot の環境変数の設定」を参照して変数を消します。

- ・ ユーザーのスクリプトなどから、「abos-ctrl rollback」コマンドを実行した場合。

ロールバックが実行されると /var/at-log/at log にログが残ります。

## 13.2. hawkBit

### 13.2.1. hawkBit とは

hawkBit は、サーバー上で実行されるプログラムで、ネットワーク経由でデバイスのソフトウェアを更新(配信)することができます。

hawkBit は次のような機能を持っています。

- ・ ソフトウェアの管理
- ・ デバイスの管理
  - ・ デバイス認証 (セキュリティトークン、証明書)
  - ・ デバイスのグループ化
- ・ アップデート処理の管理
  - ・ 進捗のモニタリング
  - ・ スケジューリング、強制アップデート
- ・ RESTful API での直接操作

### 13.2.2. データ構造

hawkBit は、配信するソフトウェアを次のデータ構造で管理します。

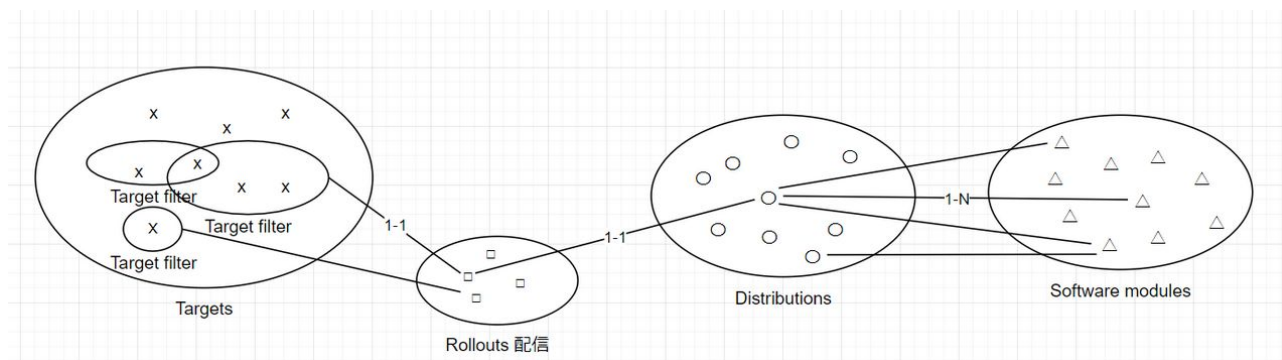


図 13.1 hawkBit が扱うソフトウェアのデータ構造



# 14. ハードウェア仕様

## 14.1. 電氣的仕様

### 14.1.1. 絶対最大定格

表 14.1 絶対最大定格

項目	記号	Min.	Max.	単位	備考
電源電圧	VCC_5V	-0.3	6.0	V	-
入出力電圧(USB 信号以外)	VI,VO	-0.3	OVDD +0.3	V	OVDD=VCC_3.3V
入力電圧(USB 信号)	VI_USB	-0.3	3.63	V	USB_OTG1_DP, USB_OTG1_DN, USB_OTG2_DP, USB_OTG2_DN
RTC バックアップ電源電圧	RTC_BAT	-0.3	3.6	V	-
使用温度範囲	Topr	-20	70	°C	結露なきこと



絶対最大定格は、あらゆる使用条件や試験状況において、瞬時でも超えてはならない値です。上記の値に対して余裕をもってご使用ください。

### 14.1.2. 推奨動作条件

表 14.2 推奨動作条件

項目	記号	Min.	Typ.	Max.	単位	備考
電源電圧	VCC_5V	4.75	5	5.25	V	-
RTC バックアップ電源電圧	RTC_BAT	2.75 <sup>[a]</sup>	-	3.3	V	Topr=+25°C
使用温度範囲	Ta	-20	25	70	V	結露なきこと

<sup>[a]</sup>S/N: 009C00010001~009C00060102 の Armadillo-640 では、下限電圧 2.95V です。

### 14.1.3. 入出力インターフェースの電氣的仕様

表 14.3 入出力インターフェース(電源)の電氣的仕様

項目	記号	Min.	Typ.	Max.	単位	備考
5V 電源電圧	VCC_5V USB_OTG1_VBUS USB_OTG2_VBUS	4.75	5	5.25	V	-
3.3V 電源電圧	VCC_3.3V	3.102	3.3	3.498	V	-

表 14.4 入出力インターフェースの電氣的仕様(OVDD = VCC\_3.3V)

項目	記号	Min.	Max.	単位	備考
ハイレベル出力電圧	VOH	OVDD-0.15	OVDD	V	IOH = -0.1mA, -1mA
ローレベル出力電圧	VOL	0	0.15	V	IOL = 0.1mA, 1mA
ハイレベル入力電圧 <sup>[a]</sup>	VIH	0.7×OVDD	OVDD	V	-

項目	記号	Min.	Max.	単位	備考
ローレベル入力電圧 <sup>[a]</sup>	VIL	0	0.3×OVDD	V	-
ローレベル入力電圧(ONOFF 信号)	VIL	0	0.9	V	-
ローレベル入力電圧 (PWRON 信号)	VIL	0	0.5	V	-
ローレベル入力電圧 (EXT_RESET_B 信号)	VIL	0	0.19	V	-
入力リーク電流(no Pull-up/ Pull-down)	IIN	-1	1	μA	-
Pull-up 抵抗(5kΩ)	-	4	6	kΩ	-
Pull-up 抵抗(47kΩ)	-	37.6	56.4	kΩ	-
Pull-up 抵抗(100kΩ)	-	80	120	kΩ	-
Pull-down 抵抗(100kΩ)	-	80	120	kΩ	-

<sup>[a]</sup>オーバーシュートとアンダーシュートは 0.6V 以下でかつ 4ns を超えないようにしてください。

### 14.1.4. 電源回路の構成

電源回路の構成は次のとおりです。電源入力インターフェース(CON12 または CON13)からの入力電圧をパワーマネジメント IC(PMIC)で各電圧に変換し、内部回路および各インターフェースに供給しています。各インターフェースやスイッチング・レギュレータの最大出力電流値を超えないように、外部機器の接続、供給電源の設計を行なってください。

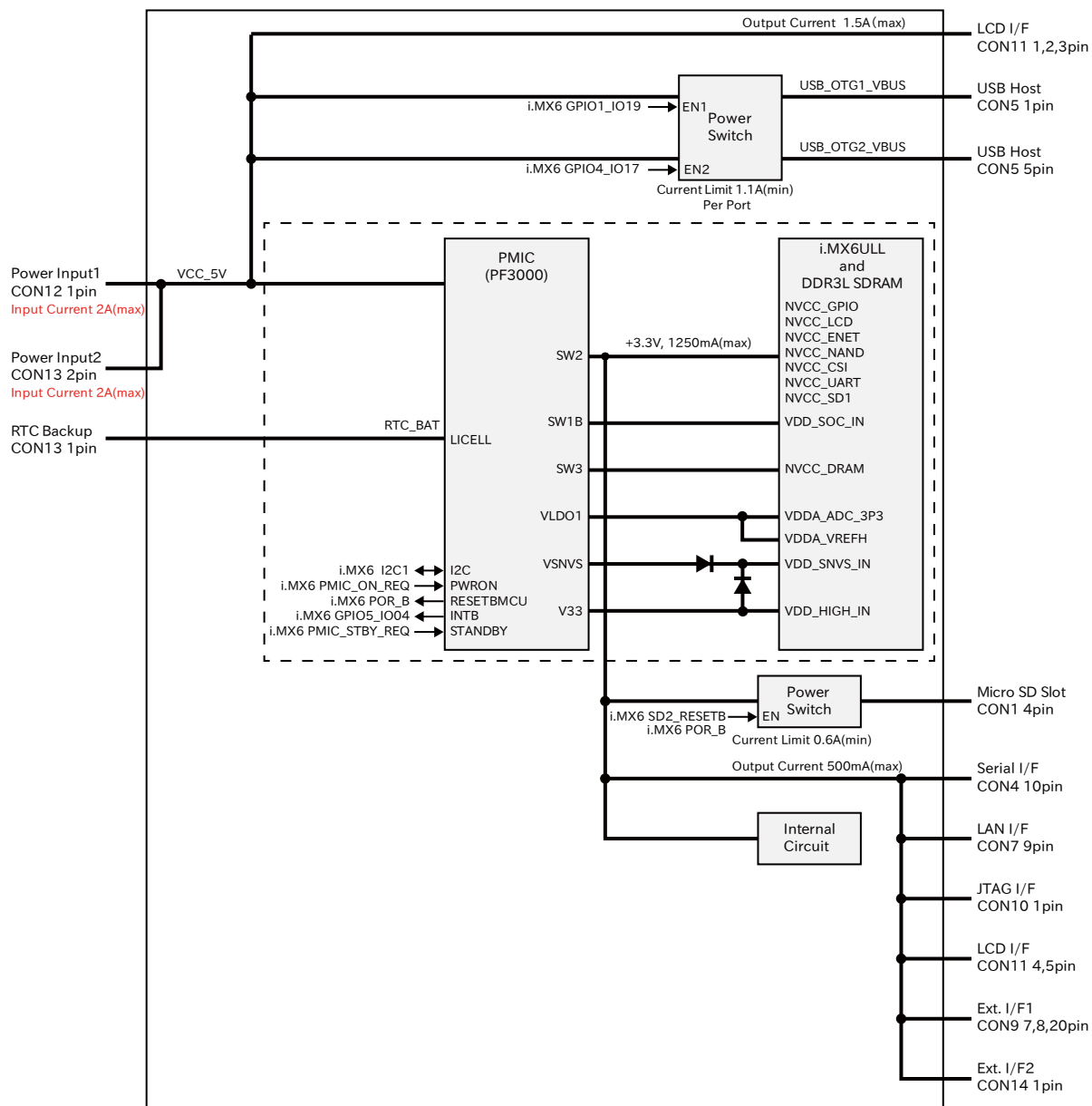


図 14.1 電源回路の構成

電源シーケンスは次のとおりです。

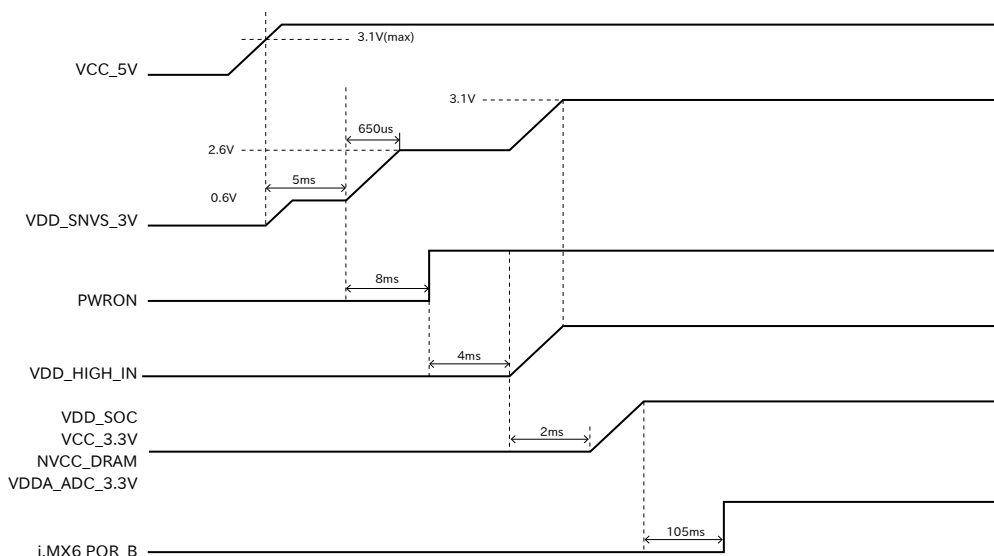


図 14.2 電源シーケンス

### 14.1.5. 外部からの電源制御

Armadillo-640 は、拡張インターフェースのピンを制御することにより電源をオンまたはオフに切り替えることができます。


ここでは、外部からの電源制御に必要な以下の項目を説明します。

- ・ ONOFF ピンの制御
- ・ PWRON ピンの制御
- ・ RTC\_BAT ピン

#### 14.1.5.1. ONOFF ピンの制御について

ONOFF ピンは、一定時間以上 GND とショートすることで、Armadillo-640 の電源をオフまたはオンすることができます。外部から ONOFF ピンを制御する場合、電圧の印加はできませんのでオープンドレインなどの出力を接続し GND とショートする回路を接続してください。

- ・ 電源オンから電源オフに切り替える方法
  - ・ ONOFF ピンを 5 秒以上 GND とショートすることで、電源がオフになります。
- ・ 電源オフから電源オンに切り替える方法
  - ・ ONOFF ピンを 500 ミリ秒以上 GND とショートすることで、電源がオンになります。



連続して電源を切り替える場合、確実に動作させるため 5 秒以上空けてから ONOFF ピンを GND とショートしてください。



電源のオンまたはオフの状況は i.MX6ULL の低消費電力ドメイン (SNVS\_LP) で保持されているため、電源オフの状態でも 5V 電源入力を切ってもしばらくは電源オフであることを保持しています。そのため、すぐに電源を再入力した場合電源が入らない状態になる可能性があります。電源オフの状態でも 5V 電源を再入力する場合は、確実に電源を入れるため、5 秒以上間隔を空けてから 5V 電源を入れてください。



電源のオンまたはオフの状況は RTC\_BAT ピンからのバックアップ電源により保持されるため、RTC\_BAT ピンにバックアップ電源を入力した状態で 5V 電源を切ったのち 5V 電源を再入力しても、5V 電源切断前の電源のオンまたはオフの状況が継続されます。

#### 14.1.5.2. PWRON ピンの制御について

PWRON ピンは、GND とショートすることで、Armadillo-640 の電源を即座にオフすることができます。外部から PWRON ピンを制御する場合、電圧の印加はできませんのでオープンドレインなどの出力を接続し GND とショートする回路を接続してください。

- ・ 電源オンから電源オフに切り替える方法
  - ・ PWRON ピンを GND とショートすることで、即座に電源がオフになります。
- ・ 電源オフから電源オンに切り替える方法
  - ・ PWRON ピンをオープンにすることで、電源がオンになります。

#### 14.1.5.3. RTC\_BAT ピンについて

RTC\_BAT ピンは、i.MX6ULL の低消費電力ドメインにある SRTC(Secure Real Time Clock)の外部バックアップインターフェースです。長時間電源が切断されても時刻データを保持させたい場合にご使用ください。

## 14.2. インターフェース仕様

Armadillo-640 のインターフェース仕様について説明します。

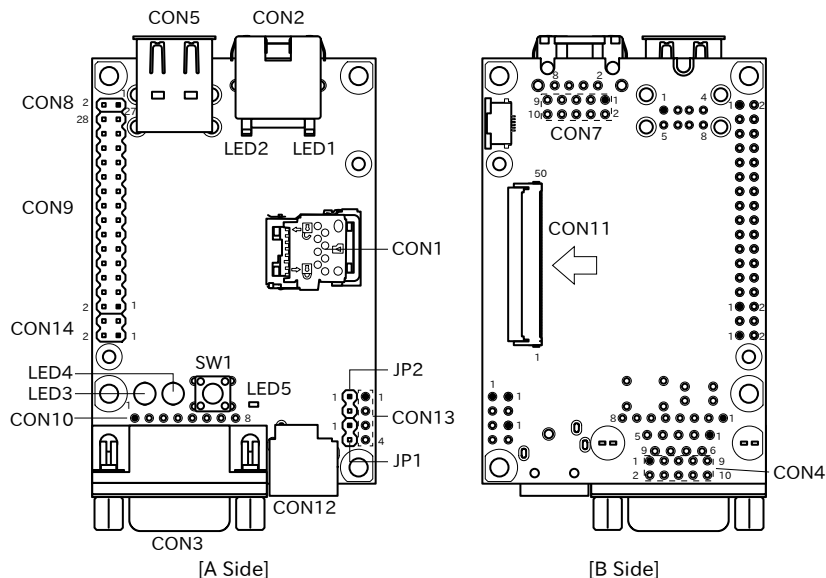


図 14.3 Armadillo-640 のインターフェース

表 14.5 Armadillo-640 インターフェース一覧 [a]


部品番号	インターフェース名	型番	メーカー
CON1	SD インターフェース	SDHK-8BNS-K-303-TB(HF)	J.S.T.Mfg.
CON2	LAN インターフェース	TM11R-5M2-88-LP	HIROSE ELECTRIC
CON7		A1-10PA-2.54DSA(71)	HIROSE ELECTRIC
CON3	シリアルインターフェース	XM2C-0942-132L	OMRON
CON4		A1-10PA-2.54DSA(71)	HIROSE ELECTRIC
CON5	USB インターフェース	UBA-4RS-D14T-4D(LF)(SN)	J.S.T.Mfg.
CON8	拡張インターフェース	A1-34PA-2.54DSA(71)	HIROSE ELECTRIC
CON9			
CON14			
CON10	JTAG インターフェース	A2-8PA-2.54DSA(71)	HIROSE ELECTRIC
CON11	LCD 拡張インターフェース	XF2M-5015-1A	OMRON
CON12	電源入力インターフェース	HEC3690-015210	HOSIDEN
CON13		A2-4PA-2.54DSA(71)	HIROSE ELECTRIC
JP1	起動デバイス設定ジャンパ	A2-4PA-2.54DSA(71)	HIROSE ELECTRIC
JP2			
LED1	LAN スピード LED	SML-310MTT86	ROHM
LED2	LAN リンクアクティビティ LED	SML-310YTT86	ROHM
LED3	ユーザー LED(赤)	SLR-342VC3F/LK-12	ROHM/MAC8
LED4	ユーザー LED(緑)	SLR-342MC3F/LK-12	ROHM/MAC8
LED5	ユーザー LED(黄)	SML-310YTT86	ROHM
SW1	ユーザースイッチ	SKHHDJA010	ALPS ELECTRIC

[a] 部品の実装、未実装を問わず、搭載可能な部品型番を記載しています。

### 14.2.1. CON1(SD インターフェース)

CON1 はハイスピード(最大クロック周波数: 49.5MHz)に対応した SD インターフェースです。信号線は i.MX6ULL の SD ホストコントローラ(uSDHC2)に接続されています。

SD カードに供給される電源は i.MX6ULL の NAND\_ALE ピン(GPIO4\_IO10)で制御が可能です。High レベル出力で電源が供給され、Low レベル出力で電源が切断されます。

 CON1 は活線挿抜に対応していません。microSD カードの挿抜は、電源を切断してから行ってください。

 SD コントローラ(uSDHC2)は CON1、CON9、CON11 で利用可能ですが、排他利用となります。

表 14.6 CON1 信号配列

ピン番号	ピン名	I/O	説明
1	DAT2	In/Out	SD データバス(bit2)、i.MX6ULL の NAND_DATA02 ピンに接続
2	CD/DAT3	In/Out	SD データバス(bit3)、i.MX6ULL の NAND_DATA03 ピンに接続
3	CMD	In/Out	SD コマンド/レスポンス、i.MX6ULL の NAND_WE_B ピンに接続
4	VDD	Power	電源(VCC_3.3V)
5	CLK	Out	SD クロック、i.MX6ULL の NAND_RE_B ピンに接続
6	VSS	Power	電源(GND)
7	DAT0	In/Out	SD データバス(bit0)、i.MX6ULL の NAND_DATA00 ピンに接続
8	DAT1	In/Out	SD データバス(bit1)、i.MX6ULL の NAND_DATA01 ピンに接続

### 14.2.1.1. microSD カードの挿抜方法

1. 上からカバーを軽く押し、約 1.2mm スライドさせて、ロックを解除します。

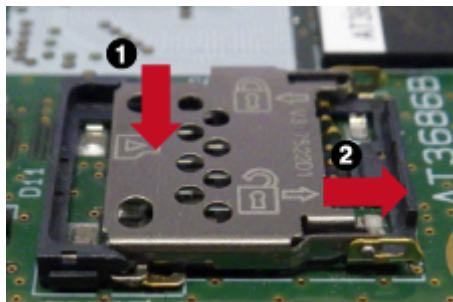


図 14.4 カバーのロックを解除する

2. カバーを開けます。

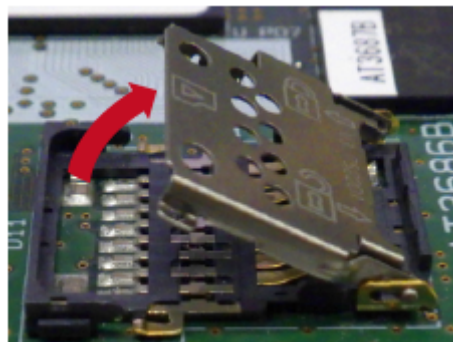


図 14.5 カバーを開ける



カバーは過度な力で回転させたり、回転方向以外の方向へ力を加えると、破損の原因となりますので、ご注意ください。

3. 任意の角度までトレイを開いた状態で、microSD カードを挿抜します。



図 14.6 microSD カードの挿抜



microSD カード挿入方向については、カバーに刻印されているカードマークを目安にしてください。



図 14.7 カードマークの確認

4. カバーを閉めます。



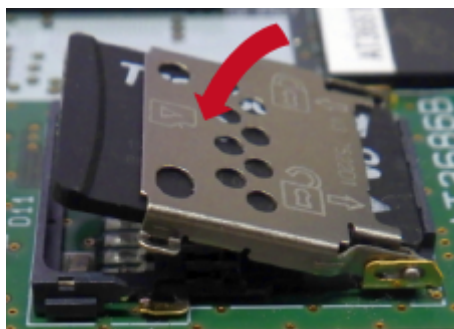


図 14.8 カバーを閉める

5. 上からカバーを軽く押し、約 1.2mm スライドさせて、ロックします。

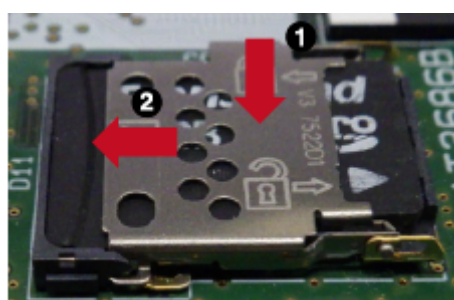


図 14.9 カバーをロックする



microSD カード装着後のカードの抜き取り手順は挿入時と同じです。

### 14.2.2. CON2、CON7(LAN インターフェース)

CON2、CON7 は 10BASE-T/100BASE-TX に対応した LAN インターフェースです。カテゴリ 5 以上の Ethernet ケーブルを接続することができます。AUTO-MDIX 機能を搭載しており、ストレートケーブルまたはクロスケーブルを自動認識して送受信端子を切り替えます。

信号線は Ethernet PHY(LAN8720AI-CP/Microchip Technology) を経由して i.MX6ULL の Ethernet コントローラ(ENET1)に接続されています。

表 14.7 CON2 信号配列

ピン番号	ピン名	I/O	説明
1	TX+	In/Out	送信データ(+), CON7 の 1 ピンと共通
2	TX-	In/Out	送信データ(-), CON7 の 4 ピンと共通
3	RX+	In/Out	受信データ(+), CON7 の 3 ピンと共通
4	-	-	CON2 の 5 ピンと接続後に 75Ω 終端、CON7 の 5 ピンと共通
5	-	-	CON2 の 4 ピンと接続後に 75Ω 終端、CON7 の 5 ピンと共通
6	RX-	In/Out	受信データ(-), CON7 の 6 ピンと共通
7	-	-	CON2 の 8 ピンと接続後に 75Ω 終端、CON7 の 7 ピンと共通

ピン番号	ピン名	I/O	説明
8	-	-	CON2 の 7 ピンと接続後に 75Ω 終端、CON7 の 7 ピンと共通

表 14.8 CON7 信号配列

ピン番号	ピン名	I/O	説明
1	TX+	In/Out	送信データ(+)
2	LINK_ACTIVITY_LED	Out	LINK/ACTIVITY 表示 (High: リンクが確立されている、Low: リンクが確立されていない、Pulse : リンクが確立されており、データを送受信している)
3	RX+	In/Out	受信データ(+)
4	TX-	In/Out	送信データ(-)
5	-	-	75Ω 終端、CON2 の 4、5 ピンと共通
6	RX-	In/Out	受信データ(-)
7	-	-	75Ω 終端、CON2 の 7、8 ピンと共通
8	SPEED_LED	Out	SPEED 表示 (High: 10Mbps または Ethernet ケーブル未接続、Low: 100Mbps)
9	VCC_3.3V	Power	電源(VCC_3.3V)
10	GND	Power	電源(GND)



CON2 と CON7 は、共通の信号が接続されており、同時に使用することはできません。どちらか一方のコネクタでのみ、ご使用ください。

### 14.2.3. LED1、LED2(LAN LED)

LED1、LED2 は LAN インターフェースのステータス LED です。CON2 の上部に表示されます。信号線は Ethernet PHY(LAN8720AI-CP/Microchip Technology)の LED ピンに接続されています。

表 14.9 LAN LED の動作

LED	名称(色)	状態	説明
LED1	LAN スピード LED(緑)	消灯	10Mbps で接続されている、もしくは Ethernet ケーブル未接続
		点灯	100Mbps で接続されている
LED2	LAN リンクアクティビティ(黄)	消灯	リンクが確立されていない
		点灯	リンクが確立されている
		点滅	リンクが確立されており、データを送受信している

### 14.2.4. CON3、CON4(シリアルインターフェース)

CON3、CON4 は非同期(調歩同期)シリアルインターフェースです。信号線は RS232C レベル変換 IC を経由して i.MX6ULL の UART コントローラ(UART3)に接続されています。

- ・ 信号入出力レベル: RS232C レベル
- ・ 最大データ転送レート: 230.4kbps
- ・ フロー制御: CTS、RTS、DTR、DSR、DCD、RI

表 14.10 CON3 信号配列

ピン番号	ピン名	I/O	説明
1	DCD	In	キャリア検出、i.MX6ULL の SNVS_TAMPER2 ピンに接続、CON4 の 1 ピンと共通
2	RXD	In	受信データ、i.MX6ULL の UART3_RX_DATA ピンに接続、CON4 の 3 ピンと共通
3	TXD	Out	送信データ、i.MX6ULL の UART3_TX_DATA ピンに接続、CON4 の 5 ピンと共通
4	DTR	Out	データ端末レディ、i.MX6ULL の GPIO1_IO00 ピンに接続、CON4 の 7 ピンと共通
5	GND	Power	電源(GND)
6	DSR	In	データセットレディ、i.MX6ULL の SNVS_TAMPER0 ピンに接続、CON4 の 2 ピンと共通
7	RTS	Out	送信要求、i.MX6ULL の UART3_CTS_B ピンに接続、CON4 の 4 ピンと共通
8	CTS	In	送信可能、i.MX6ULL の UART3_RTS_B ピンに接続、CON4 の 6 ピンと共通
9	RI	In	被呼表示、i.MX6ULL の SNVS_TAMPER1 ピンに接続、CON4 の 8 ピンと共通

表 14.11 CON4 信号配列

ピン番号	ピン名	I/O	説明
1	DCD	In	キャリア検出、i.MX6ULL の SNVS_TAMPER2 ピンに接続、CON3 の 1 ピンと共通
2	DSR	In	データセットレディ、i.MX6ULL の SNVS_TAMPER0 ピンに接続、CON3 の 6 ピンと共通
3	RXD	In	受信データ、i.MX6ULL の UART3_RX_DATA ピンに接続、CON3 の 2 ピンと共通
4	RTS	Out	送信要求、i.MX6ULL の UART3_CTS_B ピンに接続、CON3 の 7 ピンと共通
5	TXD	Out	送信データ、i.MX6ULL の UART3_TX_DATA ピンに接続、CON3 の 3 ピンと共通
6	CTS	In	送信可能、i.MX6ULL の UART3_RTS_B ピンに接続、CON3 の 8 ピンと共通
7	DTR	Out	データ端末レディ、i.MX6ULL の GPIO1_IO00 ピンに接続、CON3 の 4 ピンと共通
8	RI	In	被呼表示、i.MX6ULL の SNVS_TAMPER1 ピンに接続、CON3 の 9 ピンと共通
9	GND	Power	電源(GND)
10	VCC_3.3V	Power	電源出力(VCC_3.3V)



CON3 と CON4 は、共通の信号が接続されており、同時に使用することはできません。どちらか一方のコネクタでのみ、ご使用ください。

### 14.2.5. CON5(USB ホストインターフェース)

CON5 は USB ホストインターフェースです。2 段のコネクタを実装しており、下段の信号線は i.MX6ULL の USB コントローラ(USB OTG1)接続されています。上段の信号線はマルチプレクサを経由して、i.MX6ULL の USB コントローラ(USB OTG2)に接続されています。

マルチプレクサのセレクトピンは CON9 の 24 ピンで制御することが可能で、オープンもしくは High レベルを入力することで CON5 の上段、Low レベルを入力することで CON9 に USB OTG2 の接続先が変更されます。

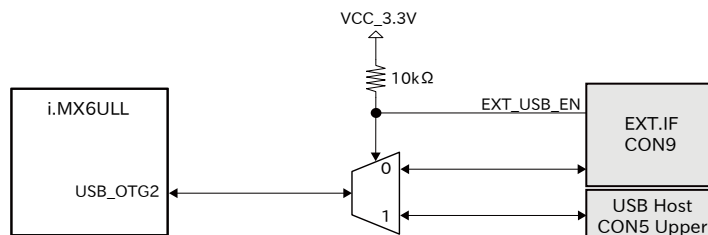


図 14.10 USB OTG2 の接続先の変更

下段に供給される電源(USB\_OTG1\_VBUS)は i.MX6ULL の UART1\_RTS\_B ピン(GPIO1\_I019)、上段に供給される電源(USB\_OTG2\_VBUS)は i.MX6ULL の CSI\_MCLK ピン(GPIO4\_I017)で制御が可能です、High レベル出力で電源が供給され、Low レベル出力で電源が切断されます。

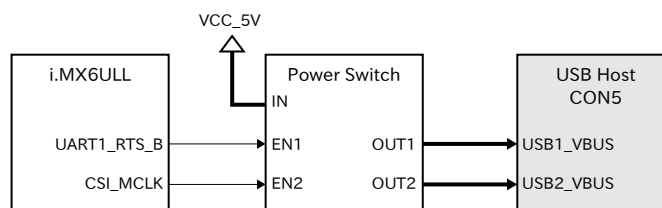


図 14.11 USB ホストインターフェースの電源制御

- ・ データ転送モード
  - ・ High Speed(480Mbps)
  - ・ Full Speed(12Mbps)
  - ・ Low Speed(1.5Mbps)

表 14.12 CON5 信号配列

ピン番号	ピン名	I/O	説明
1	USB1_VBUS	Power	電源(USB_OTG1_VBUS)、i.MX6ULL の USB_OTG1_VBUS ピンに接続
2	USB1_DN	In/Out	USB1 のマイナス側信号、i.MX6ULL の USB_OTG1_DN ピンに接続
3	USB1_DP	In/Out	USB1 のプラス側信号、i.MX6ULL の USB_OTG1_DP ピンに接続
4	GND	Power	電源(GND)
5	USB2_VBUS	Power	電源(USB2_OTG2_VBUS)、i.MX6ULL の USB_OTG2_VBUS ピンに接続
6	USB2_DN	In/Out	USB2 のマイナス側信号、i.MX6ULL の USB_OTG2_DN ピンに接続
7	USB2_DP	In/Out	USB2 のプラス側信号、i.MX6ULL の USB_OTG2_DP ピンに接続
8	GND	Power	電源(GND)

### 14.2.6. CON8、CON9、CON14(拡張インターフェース)

CON8、CON9、CON14 は機能拡張用のインターフェースです。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、「アットマークテクノ Armadillo サイト」 [<https://armadillo.atmark-techno.com/>]からダウンロードできる『Armadillo-640 マルチプレクス表』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。

表 14.13 CON8 信号配列

ピン番号	ピン名	I/O	説明
1	GND	Power	電源(GND)
2	GND	Power	電源(GND)


表 14.14 CON9 信号配列

ピン番号	ピン名	I/O	説明
1	GPIO1_IO22	In/Out	拡張入出力、i.MX6ULL の UART2_CTS_B ピンに接続(CTS/RTS 信号線を利用する際の注意点)
2	GPIO1_IO23	In/Out	拡張入出力、i.MX6ULL の UART2_RTS_B ピンに接続(CTS/RTS 信号線を利用する際の注意点)
3	GPIO1_IO17	In/Out	拡張入出力、i.MX6ULL の UART1_RX_DATA ピンに接続
4	GPIO1_IO31	In/Out	拡張入出力、i.MX6ULL の UART5_RX_DATA ピンに接続
5	GPIO1_IO16	In/Out	拡張入出力、i.MX6ULL の UART1_TX_DATA ピンに接続
6	GPIO1_IO30	In/Out	拡張入出力、i.MX6ULL の UART5_TX_DATA ピンに接続
7	VCC_3.3V	Power	電源(VCC_3.3V)
8	VCC_3.3V	Power	電源(VCC_3.3V)
9	GND	Power	電源(GND)
10	GND	Power	電源(GND)
11	ONOFF	In	i.MX6ULL の ON/OFF 信号、オープンドレイン入力、i.MX6ULL の ONOFF ピンに接続、i.MX6ULL 内部で 100kΩ プルアップ(VDD_SNV5_3V)されています。
12	PWRON	In	パワーマネジメント IC の PWRON 信号、オープンドレイン入力、パワーマネジメント IC の PWRON ピンと i.MX6ULL の PMIC_ON_REQ ピンに接続、i.MX6ULL 内部で 100kΩ プルアップ(VDD_SNV5_3V)されています。
13	GPIO3_IO23	In/Out	拡張入出力、i.MX6ULL の LCD_DATA18 ピンに接続、基板上で 10kΩ プルダウンされています。
14	GPIO3_IO24	In/Out	拡張入出力、i.MX6ULL の LCD_DATA19 ピンに接続、基板上で 10kΩ プルダウンされています。
15	GPIO3_IO25	In/Out	拡張入出力、i.MX6ULL の LCD_DATA20 ピンに接続、基板上で 10kΩ プルダウンされています。
16	GPIO3_IO26	In/Out	拡張入出力、i.MX6ULL の LCD_DATA21 ピンに接続、基板上で 10kΩ プルダウンされています。
17	GPIO3_IO27	In/Out	拡張入出力、i.MX6ULL の LCD_DATA22 ピンに接続、基板上で 10kΩ プルダウンされています。
18	GPIO3_IO28	In/Out	拡張入出力、i.MX6ULL の LCD_DATA23 ピンに接続、基板上で 10kΩ プルダウンされています。
19	GND	Power	電源(GND)
20	VCC_3.3V	Power	電源(VCC_3.3V)
21	USB2_DN	In/Out	USB マイナス側信号、i.MX6ULL の USB_OTG2_DN ピンに接続
22	USB2_DP	In/Out	USB プラス側信号、i.MX6ULL の USB_OTG2_DP ピンに接続
23	USB2_VBUS	Power	電源(USB_OTG2_VBUS)、i.MX6ULL の USB_OTG2_VBUS ピンに接続
24	USB2_EN	In	USB OTG2 の切り替え信号、(Low: USB OTG2 を CON9 で使用する、High: USB OTG2 を CON5 で使用する)、基板上で 10kΩ プルアップ(VCC_3.3V)されています。
25	GPIO4_IO06	In/Out	拡張入出力、i.MX6ULL の NAND_DATA04 ピンに接続
26	GPIO4_IO07	In/Out	拡張入出力、i.MX6ULL の NAND_DATA05 ピンに接続
27	GPIO4_IO08	In/Out	拡張入出力、i.MX6ULL の NAND_DATA06 ピンに接続
28	GPIO4_IO09	In/Out	拡張入出力、i.MX6ULL の NAND_DATA07 ピンに接続



### 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) 及び 18 ピン (GPIO3\_IO28) 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、ブートモード設定のために、基板上的プルダウン抵抗で Low レベルの状態を保持しています。意図しない動作を引き起こす原因となるため、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.MX 6ULL Applications Processor Reference Manual』をご参照ください。

表 14.15 CON14 信号配列

ピン番号	ピン名	I/O	説明
1	VCC_3.3V	Power	電源(VCC_3.3V)
2	GND	Power	電源(GND)
3	GPIO1_IO20	In/Out	拡張入出力、i.MX6ULL の UART2_TX_DATA ピンに接続
4	GPIO1_IO21	In/Out	拡張入出力、i.MX6ULL の UART2_RX_DATA ピンに接続

### 14.2.7. CON10(JTAG インターフェース)

CON10 は JTAG デバッガを接続することのできる JTAG インターフェースです。信号線は i.MX6ULL のシステム JTAG コントローラ(SJC)に接続されています。

EXT\_RESET\_B ピンからシステムリセットを行うことが可能です。システムリセットを行う際は、「図 14.12. リセットシーケンス」のとおり、20ms 以上の Low 期間を設定してください。

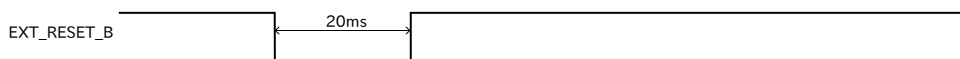



図 14.12 リセットシーケンス



CON10 に接続されている信号線は、JTAG 以外の機能でも使用可能です。詳細につきましては、「アットマークテクノ Armadillo サイト」 [<https://armadillo.atmark-techno.com/>] からダウンロードできる『Armadillo-640 マルチプレクス表』をご参照ください。



システム JTAG コントローラ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.MX 6ULL Applications Processor Reference Manual』をご参照ください。モード設定に必要な i.MX6ULL の JTAG\_MOD ピンは SW1 に接続されています。

表 14.16 CON10 信号配列

ピン番号	ピン名	I/O	説明
1	VCC_3.3V	Power	電源(VCC_3.3V)
2	JTAG_TRST_B	In	テストリセット、i.MX6ULL の JTAG_TRST_B ピンに接続、i.MX6ULL 内部で 47kΩ プルアップ(VCC_3.3V)されています。
3	JTAG_TDI	In	テストデータ入力、i.MX6ULL の JTAG_TDI ピンに接続、i.MX6ULL 内部で 47kΩ プルアップ(VCC_3.3V)されています。
4	JTAG_TMS	In	テストモード選択、i.MX6ULL の JTAG_TMS ピンに接続、i.MX6ULL 内部で 47kΩ プルアップ(VCC_3.3V)されています。
5	JTAG_TCK	In	テストクロック、i.MX6ULL の JTAG_TCK ピンに接続、i.MX6ULL 内部で 47kΩ プルアップ(VCC_3.3V)されています。
6	JTAG_TDO	Out	テストデータ出力、i.MX6ULL の JTAG_TDO ピンに接続
7	EXT_RESET_B	In	システムリセット、i.MX6ULL の POR_B ピンに接続、オープンドレイン入力
8	GND	Power	電源(GND)

### 14.2.8. CON11 (LCD 拡張インターフェース)

CON11 はデジタル RGB 入力を持つ液晶パネルモジュールなどを接続することができる、LCD 拡張インターフェースです。信号線は i.MX6ULL の LCD コントローラ等に接続されています。



CON11 に接続されている信号線は、LCD 以外の機能でも使用可能です。詳細につきましては、「アットマークテクノ Armadillo サイト」 [<https://armadillo.atmark-techno.com/>] からダウンロードできる『Armadillo-640 マルチプレクス表』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。

表 14.17 CON11 信号配列

ピン番号	ピン名	I/O	説明
1	VCC_5V	Power	電源出力(VCC_5V)
2	VCC_5V	Power	電源出力(VCC_5V)
3	VCC_5V	Power	電源出力(VCC_5V)
4	VCC_3.3V	Power	電源出力(VCC_3.3V)
5	VCC_3.3V	Power	電源出力(VCC_3.3V)
6	GND	Power	電源(GND)
7	GND	Power	電源(GND)

ピン番号	ピン名	I/O	説明
8	LCD_CLK	In/Out	拡張入出力、i.MX6ULL の LCD_CLK ピンに接続
9	LCD_HSYNC	In/Out	拡張入出力、i.MX6ULL の LCD_HSYNC ピンに接続
10	LCD_VSYNC	In/Out	拡張入出力、i.MX6ULL の LCD_VSYNC ピンに接続
11	LCD_ENABLE	In/Out	拡張入出力、i.MX6ULL の LCD_ENABLE ピンに接続
12	PWM5_OUT	In/Out	拡張入出力、i.MX6ULL の NAND_DQS ピンに接続
13	LCD_DATA00	In/Out	拡張入出力、i.MX6ULL の LCD_DATA00 ピンに接続、基板上で 10kΩ プルダウンされています。
14	LCD_DATA01	In/Out	拡張入出力、i.MX6ULL の LCD_DATA01 ピンに接続、基板上で 10kΩ プルアップ(VCC_3.3V)されています。
15	LCD_DATA02	In/Out	拡張入出力、i.MX6ULL の LCD_DATA02 ピンに接続、基板上で 10kΩ プルダウンされています。
16	LCD_DATA03	In/Out	拡張入出力、i.MX6ULL の LCD_DATA03 ピンに接続、基板上で 10kΩ プルダウンされています。
17	LCD_DATA04	In/Out	拡張入出力、i.MX6ULL の LCD_DATA04 ピンに接続、基板上で 10kΩ プルダウンされています。
18	LCD_DATA05	In/Out	拡張入出力、i.MX6ULL の LCD_DATA05 ピンに接続、JP1 がオープン時、10kΩ プルアップ(VCC_3.3V)、ショート時、10kΩ プルダウンされます。
19	GND	Power	電源(GND)
20	LCD_DATA06	In/Out	拡張入出力、i.MX6ULL の LCD_DATA06 ピンに接続、基板上で 10kΩ プルアップ(VCC_3.3V)されています。
21	LCD_DATA07	In/Out	拡張入出力、i.MX6ULL の LCD_DATA07 ピンに接続、基板上で 10kΩ プルダウンされています。
22	LCD_DATA08	In/Out	拡張入出力、i.MX6ULL の LCD_DATA08 ピンに接続、基板上で 10kΩ プルダウンされています。
23	LCD_DATA09	In/Out	拡張入出力、i.MX6ULL の LCD_DATA09 ピンに接続、基板上で 10kΩ プルダウンされています。
24	LCD_DATA10	In/Out	拡張入出力、i.MX6ULL の LCD_DATA10 ピンに接続、基板上で 10kΩ プルダウンされています。
25	LCD_DATA11	In/Out	拡張入出力、i.MX6ULL の LCD_DATA11 ピンに接続、JP1 がオープン時、10kΩ プルダウン、ショート時、10kΩ プルアップ(VCC_3.3V)されます。
26	GND	Power	電源(GND)
27	LCD_DATA12	In/Out	拡張入出力、i.MX6ULL の LCD_DATA12 ピンに接続、基板上で 10kΩ プルダウンされています。
28	LCD_DATA13	In/Out	拡張入出力、i.MX6ULL の LCD_DATA13 ピンに接続、基板上で 10kΩ プルダウンされています。
29	LCD_DATA14	In/Out	拡張入出力、i.MX6ULL の LCD_DATA14 ピンに接続、基板上で 10kΩ プルダウンされています。
30	LCD_DATA15	In/Out	拡張入出力、i.MX6ULL の LCD_DATA15 ピンに接続、基板上で 10kΩ プルダウンされています。
31	LCD_DATA16	In/Out	拡張入出力、i.MX6ULL の LCD_DATA16 ピンに接続、基板上で 10kΩ プルダウンされています。
32	LCD_DATA17	In/Out	拡張入出力、i.MX6ULL の LCD_DATA17 ピンに接続、基板上で 10kΩ プルダウンされています。
33	GND	Power	電源(GND)
34	XPUL	In/Out	拡張入出力、i.MX6ULL の GPIO1_IO04 ピンに接続、0.01uF のコンデンサが接続されています。
35	XNUR	In/Out	拡張入出力、i.MX6ULL の GPIO1_IO03 ピンに接続、0.01uF のコンデンサが接続されています。
36	YPLL	In/Out	拡張入出力、i.MX6ULL の GPIO1_IO02 ピンに接続、0.01uF のコンデンサが接続されています。
37	YNLR	In/Out	拡張入出力、i.MX6ULL の GPIO1_IO01 ピンに接続、0.01uF のコンデンサが接続されています。
38	GND	Power	電源(GND)
39	GPIO4_IO18	In/Out	拡張入出力、i.MX6ULL の CSI_PIXCLK ピンに接続
40	GPIO4_IO21	In/Out	拡張入出力、i.MX6ULL の CSI_DATA00 ピンに接続
41	GPIO4_IO24	In/Out	拡張入出力、i.MX6ULL の CSI_DATA03 ピンに接続
42	GPIO4_IO22	In/Out	拡張入出力、i.MX6ULL の CSI_DATA01 ピンに接続
43	GPIO4_IO23	In/Out	拡張入出力、i.MX6ULL の CSI_DATA02 ピンに接続



ピン番号	ピン名	I/O	説明
44	GPIO4_IO28	In/Out	拡張入出力、i.MX6ULL の CSI_DATA07 ピンに接続
45	GPIO4_IO27	In/Out	拡張入出力、i.MX6ULL の CSI_DATA06 ピンに接続
46	GPIO4_IO26	In/Out	拡張入出力、i.MX6ULL の CSI_DATA05 ピンに接続
47	GPIO4_IO25	In/Out	拡張入出力、i.MX6ULL の CSI_DATA04 ピンに接続
48	GPIO4_IO20	In/Out	拡張入出力、i.MX6ULL の CSI_HSYNC ピンに接続
49	GPIO4_IO19	In/Out	拡張入出力、i.MX6ULL の CSI_VSYNC ピンに接続
50	GND	Power	電源(GND)



### CON11 のブートモード設定ピンについて

CON11 の 13~18、20~25、27~32 ピン(LCD\_DATA00~17)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、ブートモード設定のために、基板上的プルアップ/ダウン抵抗で、High/Low レベルの状態を保持しています。意図しない動作を引き起こす原因となるため、電源投入時から U-Boot が動作するまでは、各々のピンを High/Low レベルに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.MX 6ULL Applications Processor Reference Manual』をご参照ください。

## 14.2.9. CON12、CON13(電源インターフェース)

CON12、CON13 は電源供給用インターフェースです。



CON12 と CON13 の電源(VCC\_5V)供給ラインは接続されていますので、同時に電源を供給することはできません。どちらか一方からのみ電源を供給してください。

CON12 には DC ジャックが実装されており、「図 14.13. AC アダプタの極性マーク」と同じ極性マークのある AC アダプタが使用できます。AC アダプタのジャック形状は JEITA RC-5320A 準拠(電圧区分 2)です。



図 14.13 AC アダプタの極性マーク



AC アダプタを使用する際は、AC アダプタの DC プラグを Armadillo-640 に接続してから AC プラグをコンセントに挿してください。

CON13 からは電源(VCC\_5V)供給の他、バックアップ電源(RTC\_BAT)供給、i.MX6ULL の ON/OFF 制御を行うことができます。バックアップ電源供給は、長時間電源を切断しても、i.MX6ULL の一部データ(時刻データ等)を保持したい場合にご使用ください。

表 14.18 CON13 信号配列

ピン番号	ピン名	I/O	説明
1	RTC_BAT	Power	電源(RTC_BAT)、パワーマネジメント IC の LICELL ピンに接続
2	VCC_5V	Power	電源(VCC_5V)
3	GND	Power	電源(GND)
4	ONOFF	In	i.MX6ULL の ON/OFF 用信号、i.MX6ULL の ONOFF ピンに接続、オープンドレイン入力

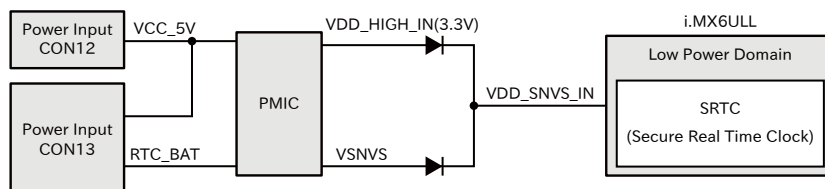






図 14.14 バックアップ電源供給

 低消費電力モードに速やかに移行するためには、バックアップ電源 (RTC\_BAT) を供給した直後に一度、電源 (VCC\_5V) を 100 ミリ秒以上供給する必要があります。

 RTC\_BAT の入力電圧範囲は 2.75V~3.3V です。内部デバイスが正常に動作しなくなる可能性がありますので、入力電圧範囲内でご使用ください。

 内蔵リアルタイムクロックの平均月差は周囲温度 25°C で ±70 秒程度 (参考値) です。時間精度は周囲温度に大きく影響を受けますので、ご使用の際は十分に特性の確認をお願いします。

 内蔵リアルタイムクロックは、一般的なリアルタイムクロック IC よりも消費電力が高いため、外付けバッテリーの消耗が早くなります。

バッテリー持続例: CR2032 の場合、約 4 か月

バッテリーの消耗が製品の運用に支障をきたす場合には、消費電力が少ないリアルタイムクロック IC を外付けすることを推奨します。CON9 (拡張インターフェース) に接続可能な Armadillo-600 シリーズ RTC オプションモジュールもありますので、ご検討ください。

### 14.2.10. LED3、LED4、LED5(ユーザー LED)

LED3、LED4、LED5 は、ユーザー側で自由に利用できる LED です。

表 14.19 LED3、LED4、LED5

部品番号	名称(色)	説明
LED3	ユーザー LED(赤)	i.MX6ULL の GPIO1_IO05 ピンに接続、(Low: 消灯、High: 点灯)
LED4	ユーザー LED(緑)	i.MX6ULL の GPIO1_IO08 ピンに接続、(Low: 消灯、High: 点灯)
LED5	ユーザー LED(黄)	i.MX6ULL の UART1_CTS_B ピンに接続、(Low: 消灯、High: 点灯)

### 14.2.11. SW1(ユーザースイッチ)

SW1 は、ユーザー側で自由に利用できる押しボタンスイッチです。

表 14.20 SW1 信号配列

部品番号	名称	説明
SW1	ユーザースイッチ	i.MX6ULL の JTAG_MOD ピンに接続、(Low: 押されていない状態、High: 押された状態)

### 14.2.12. JP1、JP2(起動デバイス設定ジャンパ)

JP1、JP2 は起動デバイス設定ジャンパです。JP1、JP2 の状態で、起動デバイスを設定することができます。

表 14.21 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	i.MX6ULL の eFUSE 設定 <sup>[a]</sup> に基づいたデバイス eFUSE が未設定の場合は eMMC
オープン	ショート	eMMC
ショート	ショート	microSD

<sup>[a]</sup>eFUSE 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.MX 6ULL Applications Processor Reference Manual』をご参照ください。



出荷時、i.MX6ULL の起動デバイスに関する eFUSE は未設定です。未設定のまま JP2 をオープン状態で使用すると、eMMC から起動します。



eFUSE は一度書き込むと元に戻すことができません。eFUSE の設定によっては Armadillo-640 が正常に動作しなくなる可能性がありますので、書き込みを行う際には細心の注意を払うようお願いいたします。eFUSE の設定によって異常が起こった場合は保証対象外となります。



JP2 をオープン状態で使用する場合、JP1 の設定は無視されます。JP1 をショート状態にすると、プルアップ抵抗により消費電流が増加するため、JP1 はオープン状態で使用することをお勧めします。

表 14.22 JP1 信号配列

ピン番号	ピン名	I/O	説明
1	JP1	In	起動デバイス設定用信号、ロジック IC を経由して i.MX6ULL の LCD_DATA05 ピン、LCD_DATA11 ピンに接続(Low: LCD_DATA05 ピンはプルアップ、LCD_DATA11 ピンはプルダウンされます。High: LCD_DATA05 ピンはプルダウン、LCD_DATA11 ピンはプルアップされます。)、基板上で 47kΩ プルダウンされています。
2	JP1_PU	Out	VCC_3.3V で 1kΩ プルアップ

表 14.23 JP2 信号配列

ピン番号	ピン名	I/O	説明
1	JP2_PU	Out	VDD_SNVS_3V で 1kΩ プルアップ
2	JP2	In	起動デバイス設定用信号、i.MX6ULL の BOOT_MODE1 ピンに接続、i.MX6ULL 内部で 100kΩ プルダウンされています。

### 14.3. 形状図

#### 14.3.1. 基板形状図

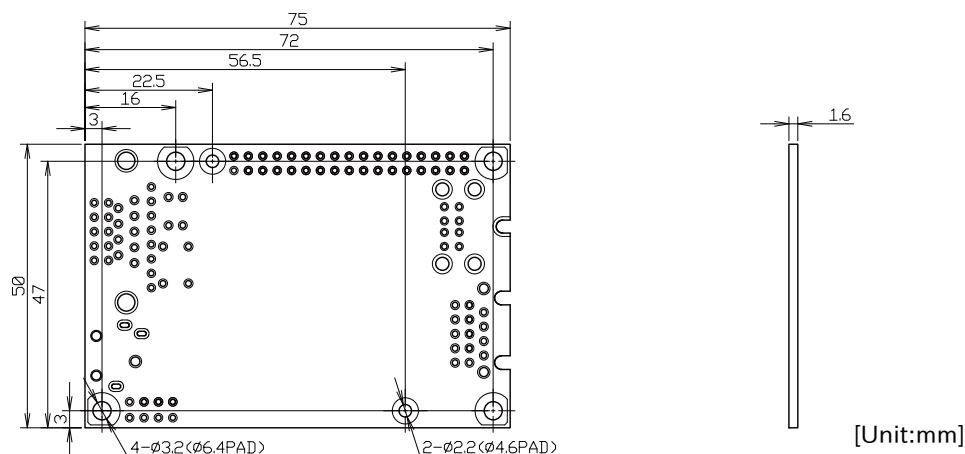


図 14.15 基板形状および固定穴寸法

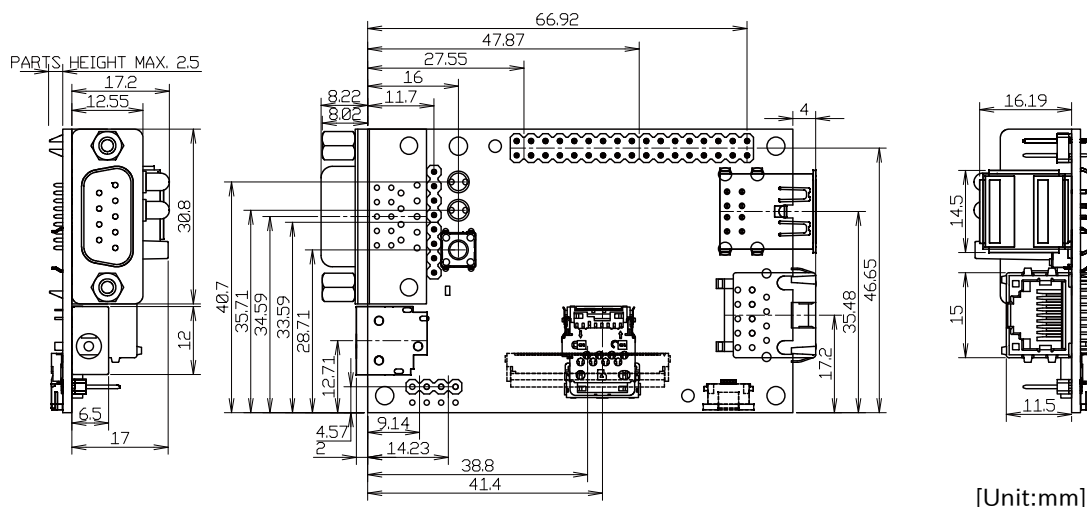
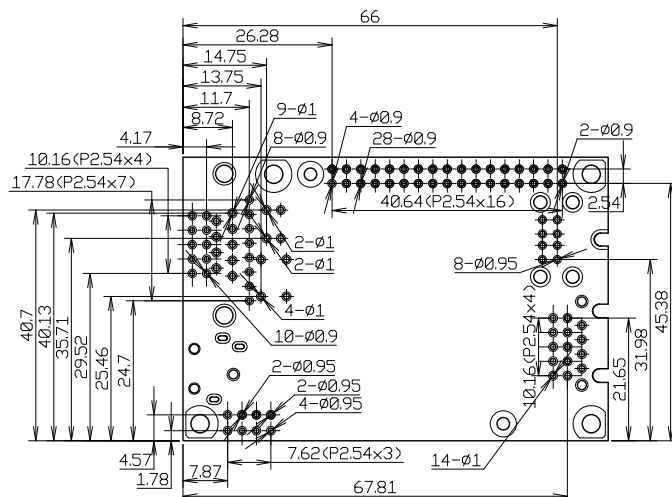


図 14.16 コネクタ中心寸法



[Unit:mm]

図 14.17 コネクタ穴寸法



基板改版や部品変更により、基板上の部品位置、高さは変更になることがあります。ケースや拡張基板を設計する場合、ある程度の余裕をもった寸法での設計をお願いいたします。



DXF 形式の形状図を「アットマークテクノ Armadillo サイト」から「購入者向けの限定公開データ」としてダウンロード可能です。

## 14.4. 設計情報

本章では、Armadillo-640 の機能拡張や信頼性向上のための設計情報について説明します。

### 14.4.1. 信頼性試験データについて

Armadillo-640 の各種信頼性試験データを、「アットマークテクノ Armadillo サイト」から「購入者向けの限定公開データ」としてダウンロード可能ですのでご確認ください。

### 14.4.2. 放射ノイズ

CON11 (LCD 拡張インターフェース) を使用して、Armadillo-640 と拡張基板を接続すると、放射ノイズが問題になる場合があります。特に、オーディオアンプのような電力が大きく変動するデバイスを拡張基板に搭載する場合、FFC の GND 線の接続のみでは強い放射ノイズが発生する可能性があります。放射ノイズを減らすために、以下の対策が効果的です。

- ・ シールド付 FFC を使用する
- ・ 長さが余る場合は、ケーブルを折りたたむ
- ・ シールドは拡張基板の GND に接続する

- ・ Armadillo-640 の GND(固定穴等)と拡張基板の GND を太い導線や金属スペーサ等で接続する
- ・ 未使用の拡張ピンは Low レベル出力とする
- ・ 使用する拡張ピンはコンデンサ(1000pF 程度)を介して GND と接続する

### 14.4.3. ESD/雷サージ

Armadillo-640 の ESD 耐性を向上させるために、以下の対策が効果的です。

- ・ Armadillo-640 を金属筐体に組み込み、GND(固定穴等)を金属ねじ等で接続する
- ・ 金属筐体を接地する

また、Armadillo-640 に接続されたケーブルが屋外に露出するような設置環境では、ケーブルに侵入した雷サージ等のストレスによりインターフェース回路が破壊される場合があります。ストレスへの耐性を向上させるために、以下の対策が効果的です。

- ・ Armadillo-640 と通信対向機の GND 接続を強化する
- ・ シールド付きのケーブルを使用する

# 15. オプション品

本章では、Armadillo-640 のオプション品について説明します。

表 15.1 Armadillo-640 関連のオプション品

名称	型番	備考
USB シリアル変換アダプタ(Armadillo-640 用)	SA-SCUSB-10	Armadillo-640 ベーシックモデル開発セットに付属
Armadillo-600 シリーズ オプションケース(樹脂製)	OP-CASE600-PLA-00	Armadillo-640 ベーシックモデル開発セットに付属
Armadillo-600 シリーズ オプションケース(金属製)	OP-CASE600-MET-00	
LCD オプションセット(7 インチタッチパネル WVGA 液晶)	OP-LCD70EXT-L00	
Armadillo-600 シリーズ RTC オプションモジュール	OP-A600-RTCMOD-00	
Armadillo-600 シリーズ WLAN コンボオプションモジュール	OP-A600-AWLMOD-20	
Armadillo-600 シリーズ BT/TH オプションモジュール	OP-A600-BTTHMOD-20	
Armadillo-600 シリーズ BT/TH オプションモジュール WLAN コンボ対応	OP-A600-BTTHMOD-21	
D-Sub9/10 ピン シリアル変換ケーブル (ロックなし)	OP-SCDSUB-00	
AC アダプタ 05 (5V/2.0A EIAJ#2)	OP-AC5V5-00	Armadillo-640 ベーシックモデル開発セットに付属
無線 LAN 用 外付けアンテナセット 08	OP-ANT-WLAN-08K	Sterling LWB5+用
外付けアンテナ固定金具 00	OP-MNT-ANT-MET-00	

## 15.1. USB シリアル変換アダプタ(Armadillo-640 用)

### 15.1.1. 概要

FT232RL を搭載した USB-シリアル変換アダプタです。シリアル信号レベルは 3.3V CMOS です。

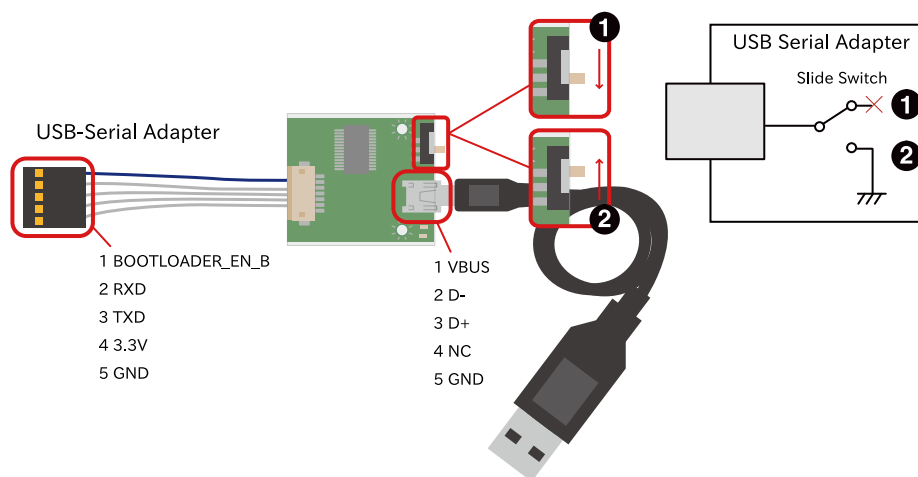


図 15.1 USB シリアル変換アダプタの配線

- ① オープン
- ② GND ショート

Armadillo-640 の CON9 の「1、3、5、7、9」または「2、4、6、8、10」ピンに接続して使用することが可能です。

各ピンに対応する UART コントローラは以下のとおりです。

表 15.2 各ピンに対応する UART コントローラ

CON9 ピン番号	UART 名
1、3、5、7、9	UART1
2、4、6、8、10	UART5

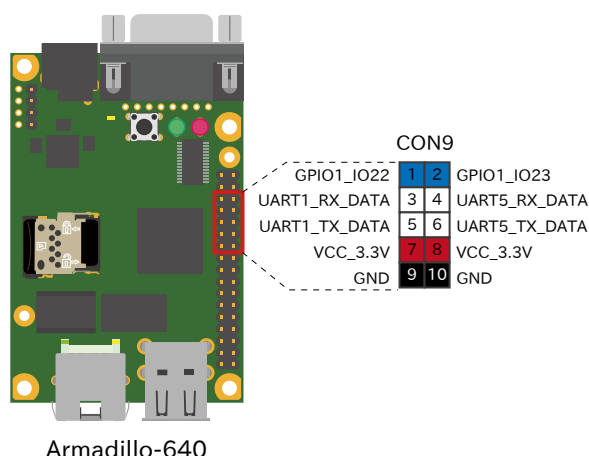



図 15.2 Armadillo-640 のシリアル信号線


ご使用の際は、USB シリアル変換アダプタの 5 ピンコネクタ(青いケーブル側)を Armadillo-640 CON9 の 1 ピンもしくは 2 ピンに合わせて接続してください。UART1 側で使用した場合、USB シリアル変換アダプタのスイッチで、電源投入時の起動モードを設定することが可能です。スライドスイッチの状態に対応した起動モードは以下のとおりです。

表 15.3 USB シリアル変換アダプタのスライドスイッチによる起動モードの設定

スライドスイッチ	起動モード
オープン	オートブートモード
GND ショート	保守モード



USB シリアル変換アダプタは、Armadillo-640 の電源を切断した状態で接続してください。故障の原因となる可能性があります。



USB シリアル変換アダプタは、試作・開発用の製品です。外観や仕様を予告なく変更する場合がありますので、ご了承ください。



## 15.2. Armadillo-600 シリーズ オプションケース(樹脂製)

### 15.2.1. 概要

Armadillo-640 用のプラスチック製小型ケースです。Armadillo-640 の基板を収めた状態で、DC ジャック、シリアルインターフェース(D-Sub9 ピン)、USB インターフェース、LAN インターフェースにアクセス可能となっています。

取り外しが可能なパーツにより、CON9(拡張インターフェース)等の機能を外部に取り出すための開口部も用意しています。



Armadillo-600 シリーズ オプションケース(樹脂製)は Armadillo-640 ベーシックモデル開発セットに付属しています。樹脂ケースのみ必要なお客様のためにオプション品として別売りもしています。

表 15.4 Armadillo-600 シリーズ オプションケース(樹脂製)について

商品名	Armadillo-600 シリーズ オプションケース(樹脂製)
型番	OP-CASE600-PLA-00
内容	樹脂ケース、ネジ、ゴム足

表 15.5 Armadillo-600 シリーズ オプションケース(樹脂製)の仕様

材質	難燃 ABS 樹脂
難燃性	UL94 V-0
色	白
使用温度範囲	-10~50°C



最高使用温度よりも高い温度で保管または使用した場合、樹脂ケースが変形する可能性があります。

## 15.2.2. 組み立て

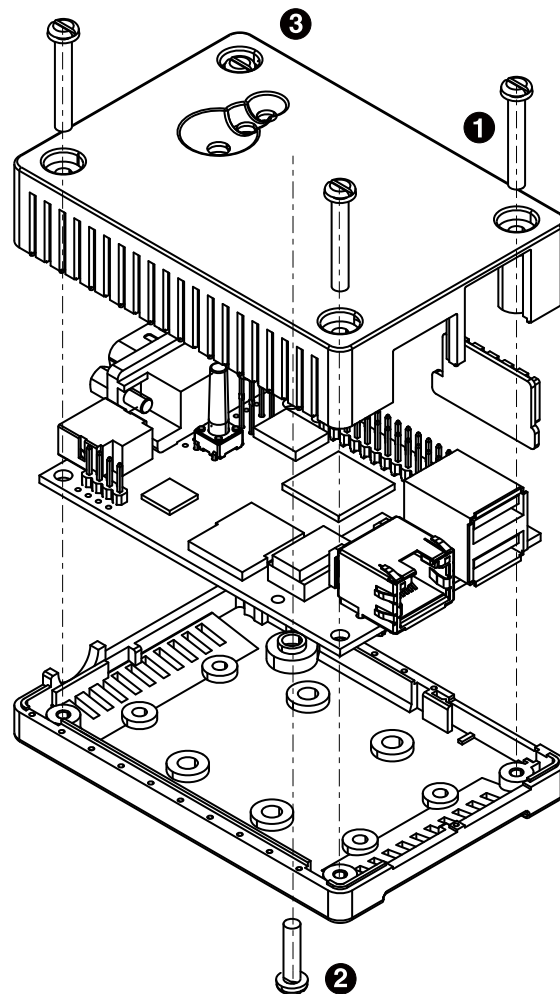


図 15.3 Armadillo-600 シリーズ オプションケース(樹脂製)の組み立て

- ❶ タッピングねじ(M2.6、L=20mm) x 3
- ❷ タッピングねじ(M3、L=12mm) x 1
- ❸ 飾りねじ x 1

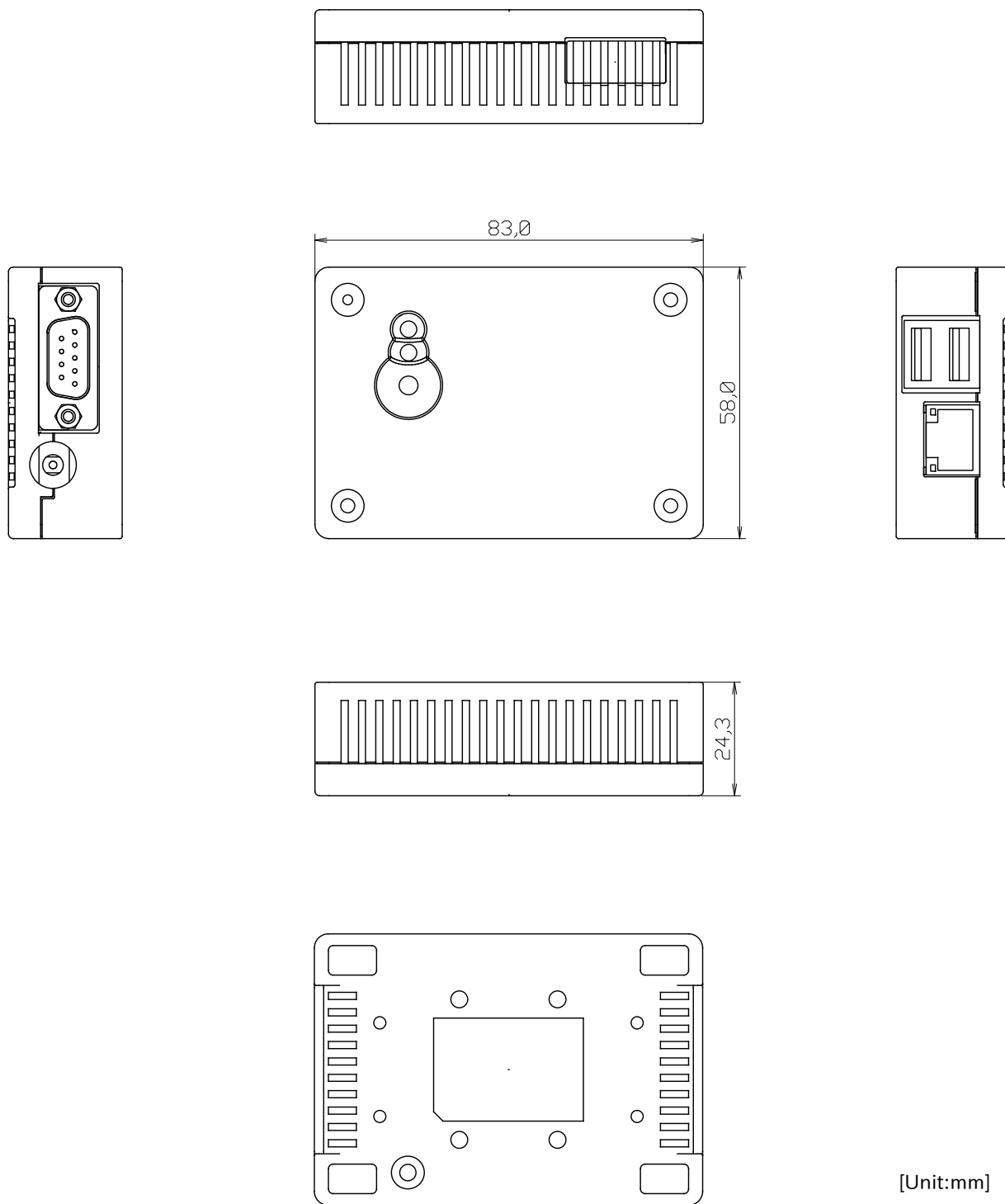


ネジをきつく締め過ぎると、ケースが破損する恐れがありますので、十分にご注意ください。



飾りネジはボンド止めされておりますので、無理に取り外さないで下さい。

### 15.2.3. 形状図



[Unit:mm]

図 15.4 樹脂ケース形状図

## 15.3. Armadillo-600 シリーズ オプションケース(金属製)

### 15.3.1. 概要

Armadillo-640 用のアルミ製小型ケースです。Armadillo-640 の基板を収めた状態で、DC ジャック、シリアルインターフェース(D-Sub9 ピン)、USB インターフェース、LAN インターフェースにアクセス可能となっています。ケース固定ネジを利用して、AC アダプタ固定用パーツおよびアース線を接続することが可能です。

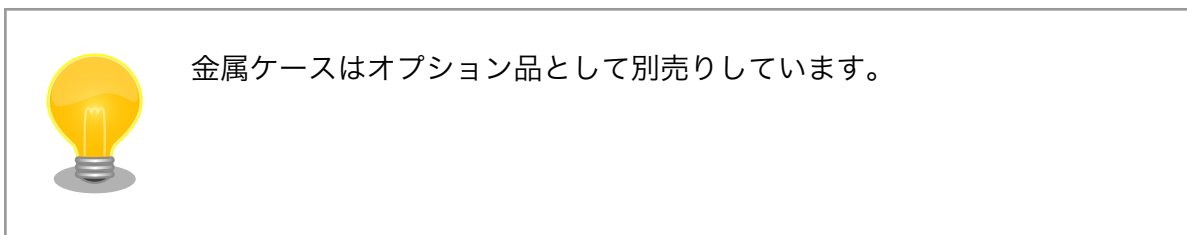


表 15.6 Armadillo-600 シリーズ オプションケース(金属製)について

商品名	Armadillo-600 シリーズ オプションケース(金属製)
型番	OP-CASE600-MET-00
内容	アルミケース、ネジ、ゴム足、AC アダプタケーブル固定用パーツ

### 15.3.2. 組み立て

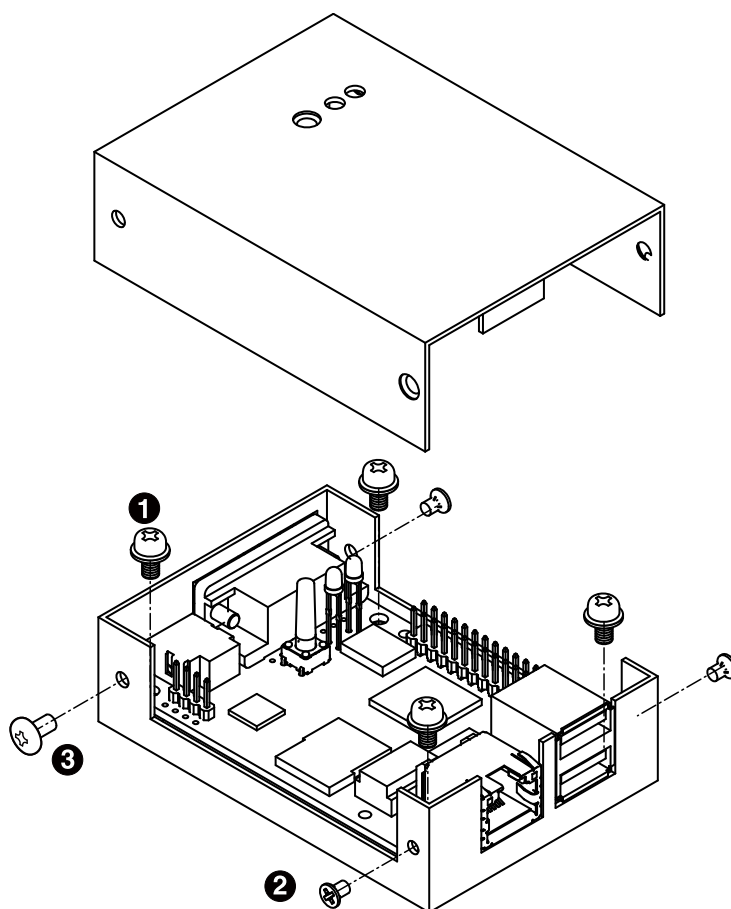


図 15.5 Armadillo-600 シリーズ オプションケース(金属製)の組み立て

- ❶ なべ小ネジ、スプリングワッシャ付き (M3、L=5mm) x 4
- ❷ 皿ネジ (M2.6、L=4mm) x 3
- ❸ トラス小ネジ (M3、L=5mm) x 1

### 15.3.3. 形状図

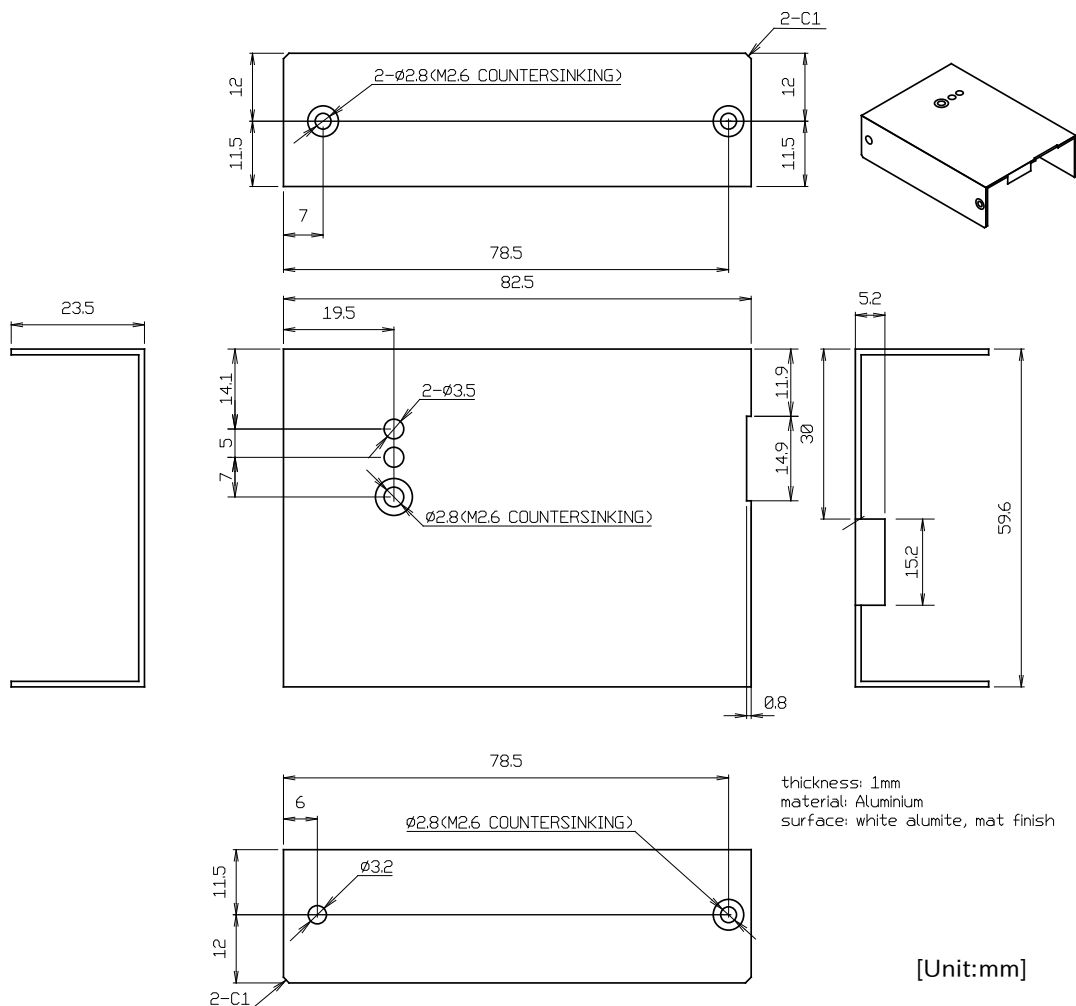


図 15.6 金属ケース(上板)寸法図

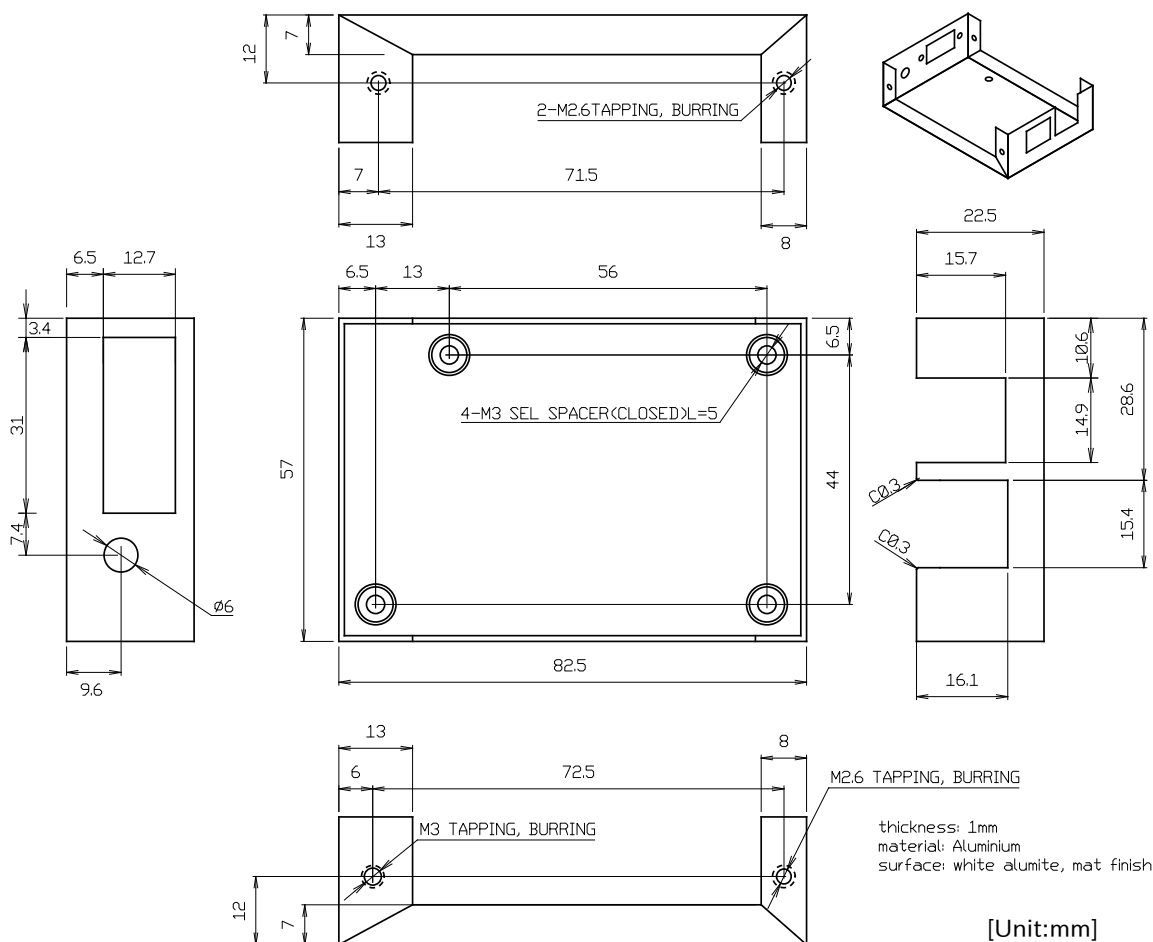


図 15.7 金属ケース(下板)寸法図

## 15.4. LCD オプションセット(7 インチタッチパネル WVGA 液晶)

### 15.4.1. 概要

ノリタケ伊勢電子製のタッチパネル LCD とフレキシブルフラットケーブル(FFC)のセットです。Armadillo-640 の CON11(LCD 拡張インターフェース)に接続して使用することが可能です。

ソフトウェアからの利用方法については、「10.10. LCD オプションセット(7 インチタッチパネル WVGA 液晶)を利用する」を参照してください。

表 15.7 LCD オプションセット(7 インチタッチパネル WVGA 液晶)について


商品名	LCD オプションセット(7 インチタッチパネル WVGA 液晶)
型番	OP-LCD70EXT-L00
内容	7インチ タッチパネル LCD、FFC

表 15.8 LCD の仕様

型番	GT800X480A-1013P
メーカー	ノリタケ伊勢電子
タイプ	TFT-LCD
表示サイズ	7 インチ
外形サイズ	164.8 x 99.8 mm

解像度	800 x 480 pixels
表示色数	約 1677 万色
使用温度範囲	-20~+70°C
輝度	850cd/m <sup>2</sup> (Typ.) 25°C
電源	DC 5V±5%/500mA (Typ.), DC 3.3V±3%/35mA (Typ)
映像入力インターフェース	RGB 平行(18bit/24bit) <sup>[a]</sup>
タッチパネルインターフェース	I2C(HID 準拠)
タッチ方式	投影型静電容量方式
マルチタッチ	最大 10 点対応

<sup>[a]</sup>Armadillo-640 は 18bit 対応です。



タッチパネル LCD をご使用になる前に、『GT800X480A-1013P 製品仕様書』にて注意事項、詳細仕様、取扱方法等をご確認ください。

『GT800X480A-1013P 製品仕様書』は「アットマークテクノ Armadillo サイト」の「[オプション] LCD オプションセット (7 インチタッチパネル WVGA 液晶) 製品仕様書」からダウンロード可能です。

### 15.4.2. 組み立て

「図 15.8. LCD の接続方法」を参考にし、タッチパネル LCD の CN4 の 1 ピンと Armadillo-640 の CON11 の 1 ピンが対応するように、FFC を接続します。FFC の向きは、タッチパネル LCD 側は電極が下、Armadillo-640 側は電極が上になるようにします。

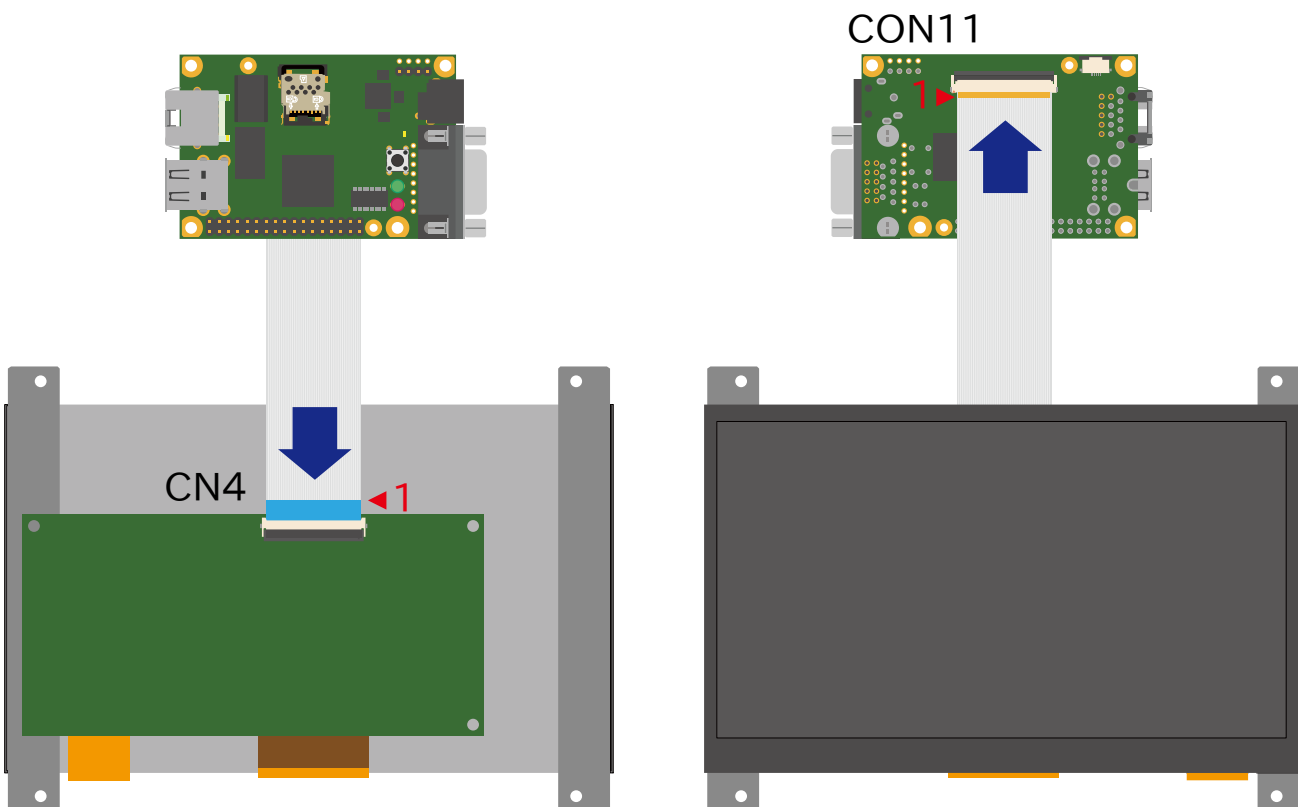


図 15.8 LCD の接続方法



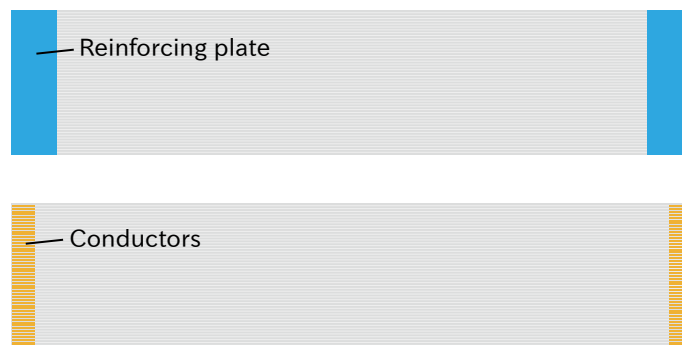




図 15.9 フレキシブルフラットケーブルの形状



必ず 1 ピンと 1 ピンが対応するように、接続してください。1 ピンと 50 ピンが対応するように接続した場合、電源と GND がショートし、故障の原因となります。



FFC の電極の上下を逆に接続した場合、Armadillo-640 の実装部品と電極が接触し、故障する可能性があります。

## 15.5. Armadillo-600 シリーズ RTC オプションモジュール

### 15.5.1. 概要

温度補償高精度リアルタイムクロック (RTC) と USB コネクタを搭載したオプションモジュールです。時刻データを保持するための電池ホルダも搭載しています。Armadillo-640 の CON9 (拡張インターフェース) に接続して使用することが可能です。



Armadillo-600 シリーズ RTC オプションモジュールの回路図、部品表は「アットマークテクノ Armadillo サイト」からダウンロード可能です。

表 15.9 Armadillo-600 シリーズ RTC オプションモジュールについて

商品名	Armadillo-600 シリーズ RTC オプションモジュール
型番	OP-A600-RTCMOD-00
内容	RTC オプションモジュール、ネジ、スペーサ


### 15.5.2. 仕様

RTC オプションモジュールの仕様は次のとおりです。

表 15.10 RTC オプションモジュールの仕様

リアルタイムクロック	型番	NR3225SA
	メーカー	日本電波工業
バックアップ	電池ホルダ、外部バッテリー接続コネクタ搭載(対応電池: CR2016、CR2032 WK11 等)	
平均月差(参考値)	約 21 秒@-20°C、約 8 秒@25°C、約 21 秒@70°C	
USB	Type A コネクタ搭載	
電源電圧	DC 3.3V±6%(RTC)、DC 2.0~3.5V(RTC バックアップ)、DC 5V±5%(USB)	
使用温度範囲	-20~+70°C(結露なきこと) [a]	
外形サイズ	42 x 50 mm	

[a]Armadillo-600 シリーズ オプションケースセット(樹脂製)を組み合わせた場合、USB デバイスの発熱量によっては、使用温度範囲が狭くなる可能性があります。



RTC の時間精度は周囲温度に大きく影響を受けますので、ご使用の際には十分に特性の確認をお願いします。

### 15.5.3. ブロック図

RTC オプションモジュールのブロック図は次のとおりです。

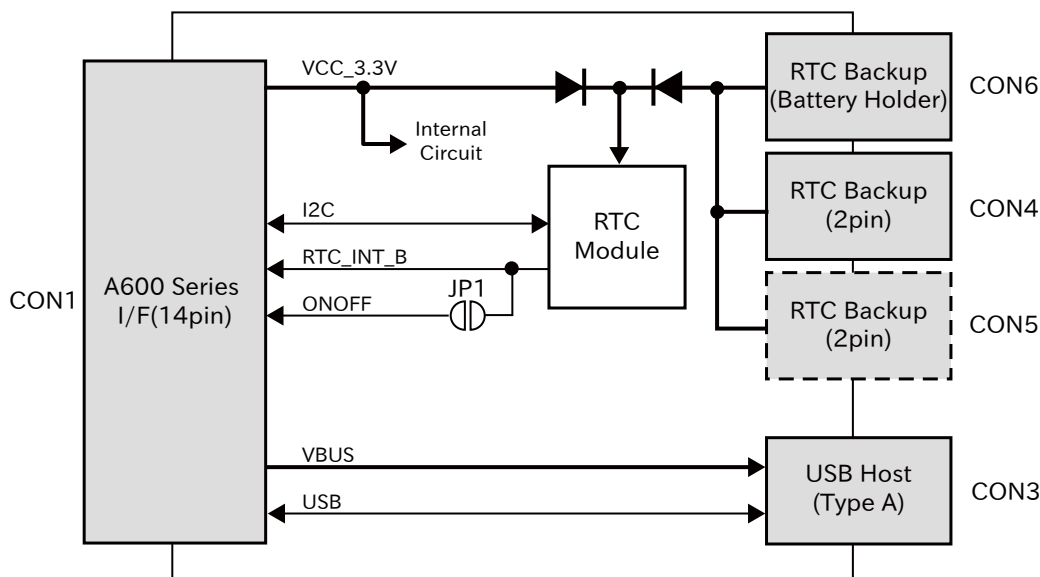


図 15.10 RTC オプションモジュールのブロック図

### 15.5.4. インターフェース仕様

RTC オプションモジュールのインターフェース仕様について説明します。

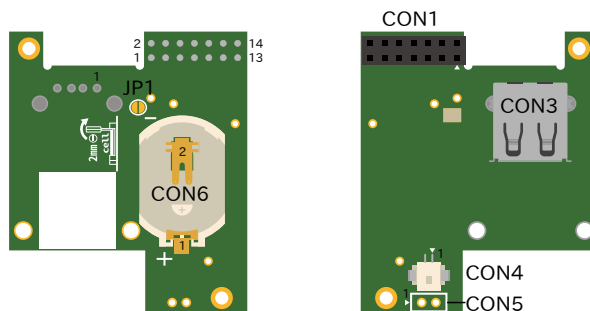


図 15.11 RTC オプションモジュールのインターフェース

表 15.11 RTC オプションモジュール インターフェース一覧 [a]

部品番号	インターフェース名	型番	メーカー
CON1	Armadillo-600 シリーズ接続インターフェース	PPPC072LFBN-RC	Sullins Connector Solutions
CON3	USB ホストインターフェース	SS-52100-001	Stewart Connector
CON4	RTC バックアップインターフェース	DF13A-2P-1.25H(21)	HIROSE ELECTRIC
CON5		B2B-EH(LF)(SN)	J.S.T.Mfg.
CON6		BLP2016SM-G	Memory Protection Devices

[a] 部品の実装、未実装を問わず、搭載可能な部品型番を記載しています。

### 15.5.4.1. CON1 (Armadillo-600 シリーズ接続インターフェース)

CON1 は Armadillo-600 シリーズの基板と接続するためのインターフェースです。Armadillo-640 の CON9(拡張インターフェース)の 11 ピンから 24 ピンに接続して使用します。

表 15.12 CON1 信号配列


ピン番号	ピン名	I/O	説明
1	ONOFF	Out	JP1 を経由して RTC の割り込みピンに接続、オープンドレイン出力 [a]
2	NC	-	未接続
3	I2C_SCL	In	RTC の I2C クロックに接続、基板上で 2kΩ プルアップ(VCC_3.3V)されています。
4	I2C_SDA	In/Out	RTC の I2C データピンに接続、基板上で 2kΩ プルアップ(VCC_3.3V)されています。
5	Reserved	-	-
6	RTC_INT_B	Out	RTC の割り込みピンに接続、オープンドレイン出力、基板上で 2kΩ プルアップ(VCC_3.3V)されています。
7	NC	-	未接続
8	USB2_PORT_EN	In	CON1 の 14 ピン(USB2_EN_B)に接続されているバッファのイネーブルピンに接続 (High: USB2_EN_B が Hi-Z、Low: USB2_EN_B が Low)
9	GND	Power	電源(GND)
10	VCC_3.3V	Power	電源(VCC_3.3V)
11	USB2_DN	In/Out	USB のマイナス側信号、CON3 の 2 ピン(D-)に接続
12	USB2_DP	In/Out	USB のプラス側信号、CON3 の 3 ピン(D+)に接続
13	USB2_VBUS	Power	電源(VBUS)、CON3 の 1 ピン(VBUS)に接続
14	USB2_EN_B	Out	バッファの出力ピンに接続、CON1 の 8 ピン(USB_PORT_EN)で設定した値が出力されます

[a] 出荷時 JP1 はオープンですので、使用する場合は JP1 をショートする必要があります。

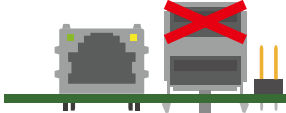
### 15.5.4.2. CON3(USB ホストインターフェース)

CON3 は USB ホストインターフェースです。


CON1 の 8 ピン(USB2\_PORT\_EN)から USB コントローラ(USB OTG2)の接続先を変更して使用します。Low レベルを入力すると RTC オプションモジュールの CON3、High レベルを入力すると Armadillo-640 の CON5 の上段に USB OTG2 が接続されます。詳細については、「14.2.5. CON5(USB ホストインターフェース)」および回路図をご確認ください。



RTC オプションモジュール CON3 と Armadillo-640 CON5 上段は同じ USB コントローラ(USB\_OTG2)に接続されており、同時に使用できません。



**図 15.12 Armadillo-640 CON5 上段と RTC オプションモジュール CON3 は排他利用**



CON3 は活線挿抜非対応となっております。ユースケースとして活線挿抜を想定していないため、突入電流に耐えるだけのコンデンサを搭載しておらず、活線挿抜した場合には Armadillo 本体の電圧が降下してしまいます。そのため、電源を切断した状態でデバイスを挿抜してください。

**表 15.13 CON3 信号配列**

ピン番号	ピン名	I/O	説明
1	VBUS	Power	電源(VBUS)
2	D-	In/Out	USB のマイナス側信号、CON1 の 11 ピンに接続
3	D+	In/Out	USB のプラス側信号、CON1 の 12 ピンに接続
4	GND	Power	電源(GND)

### 15.5.4.3. CON4、CON5、CON6(RTC バックアップインターフェース)

CON4、CON5、CON6 は RTC のバックアップ電源供給用のインターフェースです。別途バックアップ用のバッテリーを接続することで、電源(VCC\_3.3V)が切断された場合でも、時刻データを保持することが可能です。3つの形状のインターフェースがありますので、お使いのバッテリーに合わせてご使用ください。

**表 15.14 対応バッテリー例**

部品番号	対応電池	バックアップ時間(参考値)
CON4	CR2032 WK11	6.2 年
CON5 <sup>[a]</sup>	-	-
CON6	CR2016	2.5 年

<sup>[a]</sup>CON5 には部品が実装されていません。2.5mm ピッチのコネクタを実装してご使用ください。

**表 15.15 CON4、CON5、CON6 信号配列**

ピン番号	ピン名	I/O	説明
1	RTC_BAT	Power	電源(RTC_BAT)、リアルタイムクロックの電源ピンに接続
2	GND	Power	電源(GND)



CON4、CON5、CON6 は共通の端子に接続されており、同時に使用できません。



CON4、CON5、CON6 はリチウムコイン電池からの電源供給を想定したインターフェースです。リチウムコイン電池以外から電源を供給する場合、回路図、部品表にて搭載部品をご確認の上、絶対定格値を超えない範囲でご使用ください。



CON6 に搭載している電池ホルダは、大変破損しやすい部品となっております。電池の取り付け、取り外しの手順について、「15.5.5.2. 電池の取り付け、取り外し」をご確認ください。

#### 15.5.4.4. JP1 (ONOFF ピン接続ジャンパ)

JP1 は RTC のアラーム割り込み端子と Armadillo-640 の ONOFF ピン(Armadillo-640 CON9 11 ピン)を接続するジャンパです。JP1 をショートすることで、RTC のアラーム割り込みによる i.MX6ULL の電源制御が可能になります。



JP1 は、はんだジャンパになります。半分に割れたパッドになっておりますので、はんだごてでパッドを熱し、はんだを盛ってショートしてください。

### 15.5.5. 組み立て

#### 15.5.5.1. Armadillo-640 と RTC オプションモジュールの組み立て

RTC オプションモジュールは Armadillo-640 の CON9 の 11 ピンから 24 ピンに接続します。「図 15.13. RTC オプションモジュールの組み立て」のようにコネクタ接続後、金属スペーサで固定してください。



Armadillo-640 CON1 (microSD スロット) に microSD カードを挿抜する際には、RTC オプションモジュールを取り外してください。組み立て後は、RTC オプションモジュールと Armadillo-640 の間隔が狭いため、無理に挿抜した場合、microSD カードが正常に挿入されないなどの原因で、動作不良を起こす場合があります。

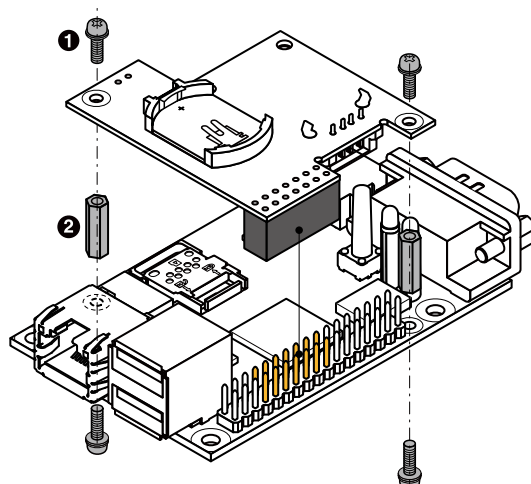


図 15.13 RTC オプションモジュールの組み立て

- ① なべ小ネジ、スプリングワッシャー、小径平ワッシャー付(M2、L=6mm) x 4
- ② 金属スペーサ(M2、L=11mm) x 2

### 15.5.5.2. 電池の取り付け、取り外し

RTC オプションモジュールの CON6(RTC バックアップインターフェース)には CR2016 等のリチウムコイン電池を搭載可能です。電池を取り付ける手順は以下のとおりです。

1. プラス端子側に電池を入れる
2. 電池ホルダのツメの下に電池を押し込む

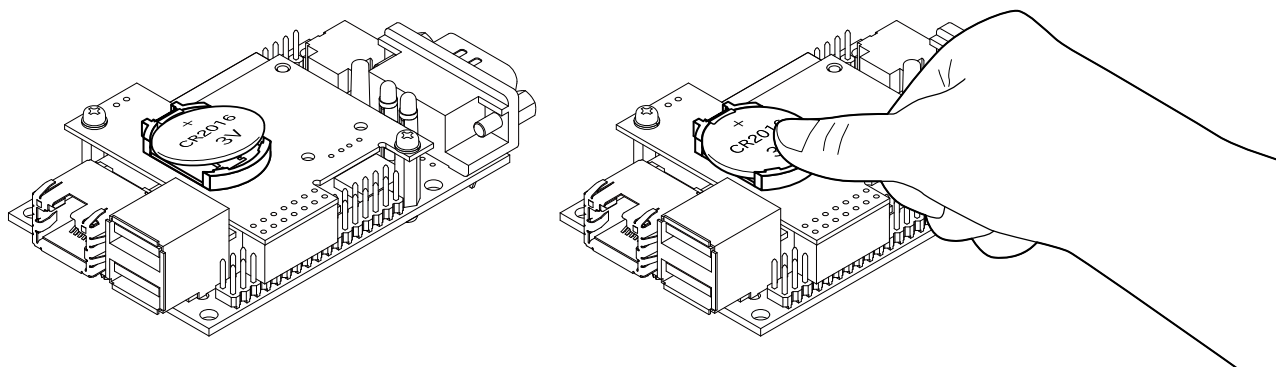


図 15.14 電池ホルダに電池を取り付ける

電池を取り外す手順は以下のとおりです。

1. プラスチック製もしくは絶縁テープを巻き付けたマイナスドライバー(2mm)を用意する
2. 電池を軽く押さえる
3. 電池ホルダの縁の中央部分と電池の間にマイナスドライバーを挿入する
4. マイナスドライバーで電池を持ち上げる

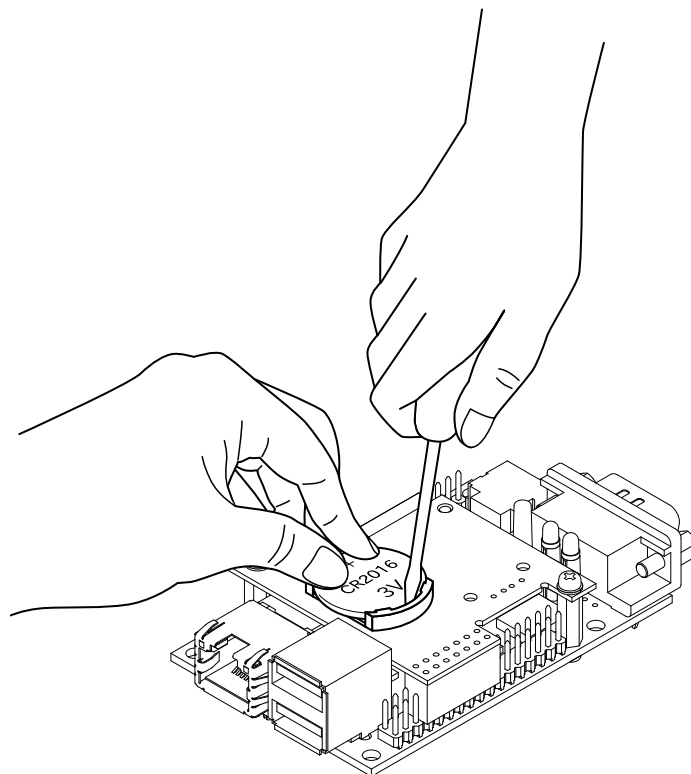


図 15.15 電池ホルダから電池を取り外す

15.5.6. 形状図

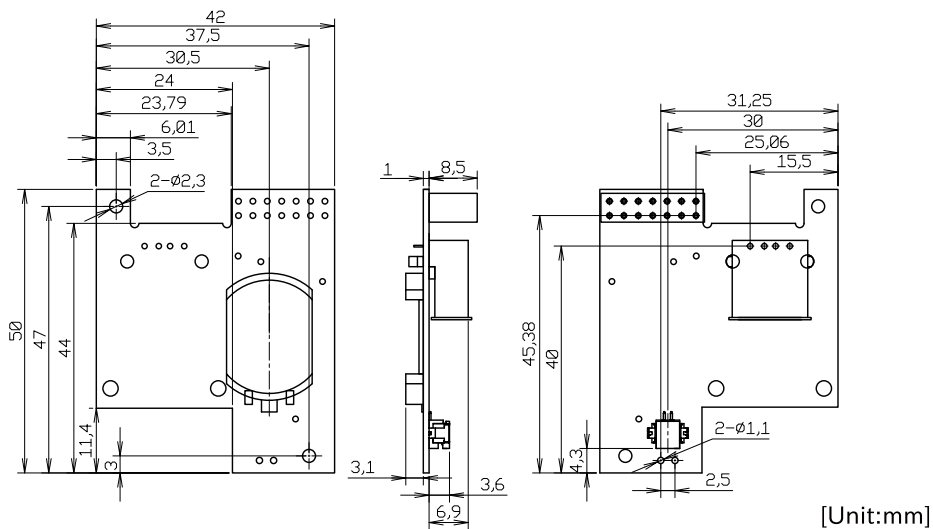


図 15.16 RTC オプションモジュール形状

## 15.6. Armadillo-600 シリーズ WLAN コンボ、BT/TH オプションモジュール

### 15.6.1. 概要

Laird Connectivity 製の無線 LAN/BT コンボモジュール Sterling LWB5+、加賀 FEI 製の BT/TH モジュール EYSKBNZWB の実装/未実装で、3 種類のオプションモジュールがあります。

表 15.16 Armadillo-640 WLAN コンボ、BT/TH オプションモジュールの搭載デバイス

名称	型番	搭載デバイス
Armadillo-600 シリーズ WLAN コンボオプションモジュール	OP-A600-AWLMOD-20	Sterling LWB5+
Armadillo-600 シリーズ BT/TH オプションモジュール	OP-A600-BTTHMOD-20	EYSKBNZWB
Armadillo-600 シリーズ BT/TH オプションモジュール WLAN コンボ対応	OP-A600-BTTHMOD-21	Sterling LWB5+および EYSKBNZWB

無線 LAN および BT 機能を使用したい場合は、Sterling LWB5+を搭載したオプションモジュール、BT 機能もしくは Thread 機能を使用したい場合は、EYSKBNZWB を搭載したオプションモジュールを選択してください。

ソフトウェアからの利用方法については、Sterling LWB5+搭載のオプションモジュールを利用している場合は「10.11. Armadillo-600 シリーズ WLAN コンボオプションモジュールを利用する」を、EYSKBNZWB 搭載のオプションモジュールを利用している場合は「10.12. Armadillo-600 シリーズ BT/TH オプションモジュールを利用する」を参照してください。

WLAN コンボ、BT/TH オプションモジュールの同梱物は「表 15.17. WLAN コンボ、BT/TH オプションモジュールの同梱物」のとおりです。Sterling LWB5+を使う場合は、外付けアンテナが必須となりますので別途ご用意ください。

表 15.17 WLAN コンボ、BT/TH オプションモジュールの同梱物

内容物	個数	用途
なべ小ネジ、スプリングワッシャー、小径平ワッシャー付(M2、L=6mm)	4	Armadillo-640 との固定
金属スペーサ(M2、L=11mm)	2	Armadillo-640 との固定
USB コネクタ用キャップ	1	Armadillo-640 CON5 上段の穴隠し用

表 15.18 推奨外付けアンテナ

商品名	無線 LAN 用 外付けアンテナセット 08
型番	OP-ANT-WLAN-08K
セット内容	アンテナ(WAND2DBI-SMA-2NB/OxfordTEC)、アンテナケーブル(ケーブル長 60mm)



Sterling LWB5+で使用可能なアンテナにつきましては、「アットマークテクノ Armadillo サイト」 [<https://armadillo.atmark-techno.com/>] で公開しているアンテナリストをご確認ください。

WLAN コンボ、BT/TH オプションモジュールは Armadillo-640 の CON14(拡張インターフェース)、CON9(拡張インターフェース)に接続して使用します。



WLAN コンボ、BT/TH オプションモジュールが Armadillo-640 の USB コントローラ(USB OTG2)を使用するため、Armadillo-640 の CON5(USB ホストインターフェース)の上段は使用できなくなります。詳細については、「14.2.5. CON5(USB ホストインターフェース)」および回路図をご確認ください。

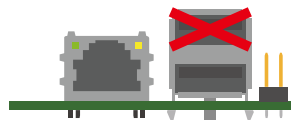


図 15.17 WLAN コンボ、BT/TH オプションモジュール接続時に Armadillo-640 CON5 上段は使用不可



Armadillo-600 シリーズ WLAN、BT/TH オプションモジュールの回路図、部品表は「アットマークテクノ Armadillo サイト」 [<https://armadillo.atmark-techno.com/>]からダウンロード可能です。

また、「15.1. USB シリアル変換アダプタ(Armadillo-640 用)」を CON9 に接続することができなくなります。シリアルコンソールが必要な場合、WLAN コンボ、BT/TH オプションモジュールの CON2(拡張インターフェース)か Armadillo-640 の CON3 もしくは CON4(シリアルインターフェース)をご使用ください。詳細につきましては、「15.6.7. シリアルコンソールの使用方法」をご確認ください。



Armadillo-640 は量産向けに、搭載するモジュールやケースの有無、部品実装の一部変更、ROM イメージの書き込みなどを選択・指定できる BTO サービスを提供しています。詳細につきましては、「アットマークテクノ BTO サービス」 [<https://armadillo.atmark-techno.com/services/customize/bto>]をご確認ください。

## 15.6.2. 仕様


WLAN コンボ、BT/TH オプションモジュールの仕様は次のとおりです。

表 15.19 WLAN コンボ、BT/TH オプションモジュールの仕様

無線 LAN/BT コンボモジュール	型番	Sterling LWB5+
	メーカー	Laird Connectivity
	無線規格	IEEE 802.11a/b/g/n/ac and Bluetooth 5.2
BT/TH モジュール	型番	EYSKBNZWB
	メーカー	加賀 FEI
	無線規格	Bluetooth 5.0 or IEEE 802.15.4(Thread) <sup>[a]</sup>
電源電圧	DC 5V±5%	
使用温度範囲(基板単体)	-20~+70°C(結露なきこと) <sup>[b]</sup>	
外形サイズ	41 x 49.7 mm	

<sup>[a]</sup>ファームウェアの書き換えにより、どちらか一方を利用することができます。

<sup>[b]</sup>Armadillo-600 シリーズ オプションケースセット(樹脂製)を組み合わせた場合の使用温度範囲は-10°C~+50°Cです。



Laird Connectivity 製 Sterling LWB5+および加賀 FEI 製 EYSKBNZWB  
の詳細な仕様については、デバイスのメーカーサイトにてご確認ください。

### 15.6.3. ブロック図

WLAN コンボ、BT/TH オプションモジュールのブロック図は次のとおりです。

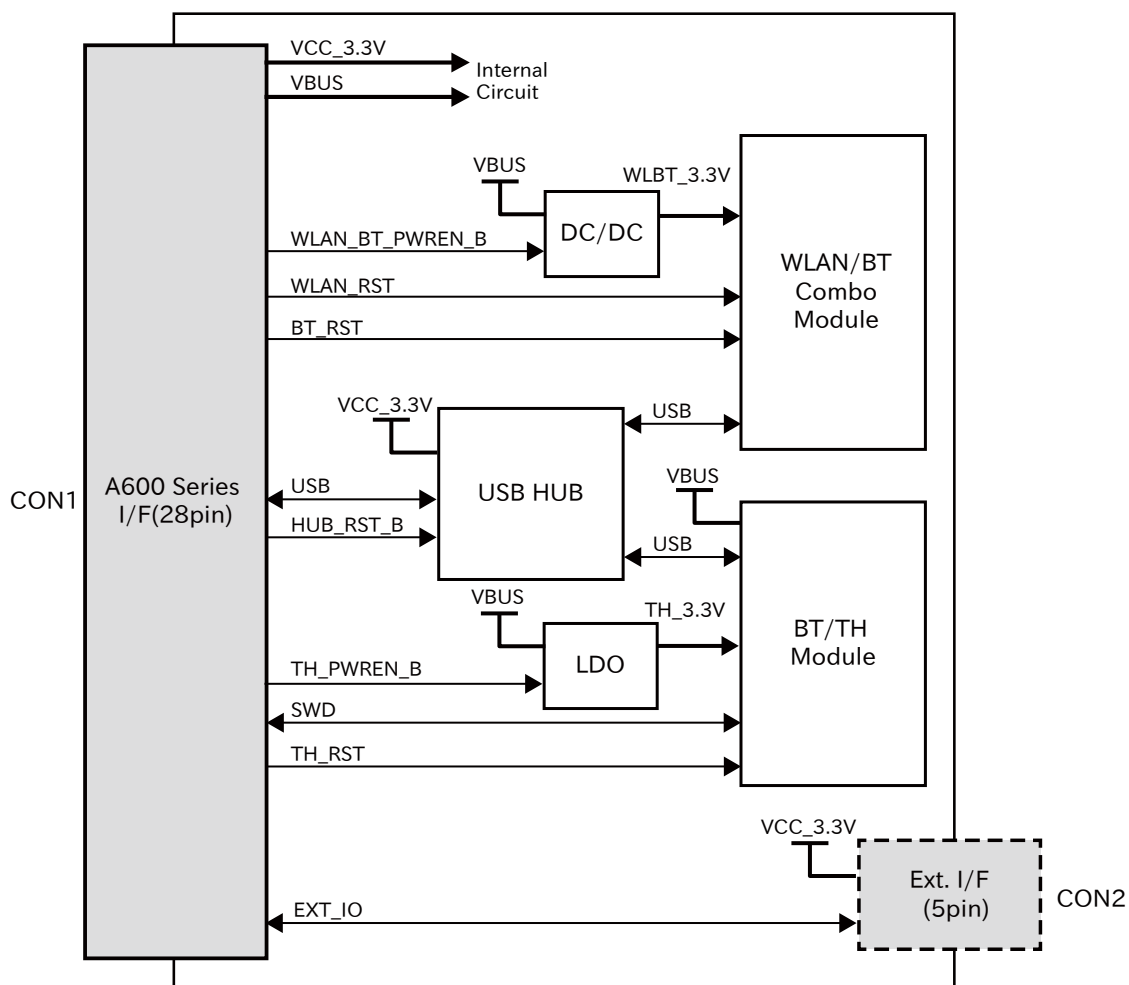


図 15.18 WLAN コンボ、BT/TH オプションモジュールのブロック図

### 15.6.4. インターフェース仕様

WLAN コンボ、BT/TH オプションモジュールのインターフェース仕様について説明します。

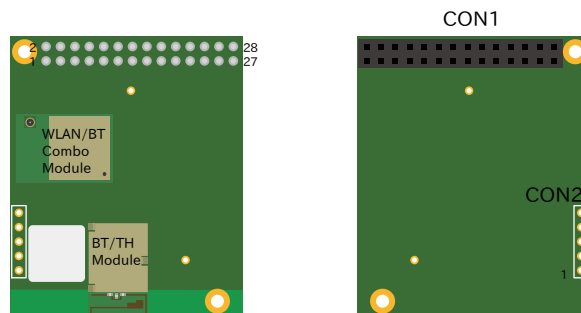


図 15.19 WLAN コンボ、BT/TH オプションモジュールのインターフェース

表 15.20 WLAN コンボ、BT/TH オプションモジュール インターフェース一覧 [a]

部品番号	インターフェース名	型番	メーカー
CON1	Armadillo-600 シリーズ接続インターフェース	PPPC142LFBN-RC	Sullins Connector Solutions
CON2	拡張インターフェース	61300511021	Würth Electronics

[a] 部品の実装、未実装を問わず、搭載可能な代表型番を記載しています。

### 15.6.4.1. CON1 (Armadillo-600 シリーズ接続インターフェース)

CON1 は Armadillo-600 シリーズの基板と接続するためのインターフェースです。

表 15.21 CON1 信号配列

ピン番号	ピン名	I/O	説明
1	VCC_3.3V	Power	電源(VCC_3.3V)
2	GND	Power	電源(GND)
3	NC	-	未接続
4	NC	-	未接続
5	UBOOT_EN_B	In/Out	CON2 の 1 ピンに接続されています。
6	SWDCLK	In	BT/TH モジュールの SWDCLK に接続されています。
7	UART1_RX	In/Out	CON2 の 2 ピンに接続されています。
8	SWDIO	In/Out	BT/TH モジュールの SWDIO に接続されています。
9	UART1_TX	In/Out	CON2 の 3 ピンに接続されています。
10	HUB_RST_B	In	USB HUB をリセットするための信号です。(High: リセット解除、Low: リセット)
11	VCC_3.3V	Power	電源(VCC_3.3V)
12	VCC_3.3V	Power	電源(VCC_3.3V)
13	GND	Power	電源(GND)
14	GND	Power	電源(GND)
15	NC	-	未接続
16	NC	-	未接続
17	WLAN_RST	In	WLAN/BT コンボモジュールの WL_REG_ON 信号に接続されています。
18	BT_RST	In	WLAN/BT コンボモジュールの BT_REG_ON 信号に接続されています。
19	WLAN_BT_PWR_EN_B	In	WLAN/BT コンボモジュールの電源を ON/OFF 制御するための信号です。(High: 電源切断、Low: 電源供給)
20	TH_RST	In	BT/TH モジュールをリセットするための信号です。(High: リセット、Low: リセット解除)
21	TH_PWREN_B	In	BT/TH モジュールの電源を ON/OFF 制御するための信号です。(High: 電源切断、Low: 電源供給)
22	NC	-	未接続
23	GND	Power	電源(GND)
24	VCC_3.3V	Power	電源(VCC_3.3V)
25	USB2_DN	In/Out	USB のマイナス側信号です。

ピン番号	ピン名	I/O	説明
26	USB2_DP	In/Out	USB のプラス側信号です。
27	USB2_VBUS	Power	電源(VBUS)
28	USB2_EN_B	Out	Armadillo-640 の USB OTG2 の接続先を切り替えるための信号です。GND に接続されています。

### 15.6.4.2. CON2(拡張インターフェース)

CON2 は機能拡張用インターフェースです。Armadillo-640 と接続した場合、UART1 の信号線が利用可能です。

表 15.22 CON2 信号配列

ピン番号	ピン名	I/O	説明	Armadillo-640 CON9 接続ピン名
1	UBOOT_EN_B	In/Out	拡張入出力	GPIO1_IO22
2	UART1_RX	In/Out	拡張入出力	GPIO1_IO17
3	UART1_TX	In/Out	拡張入出力	GPIO1_IO16
4	VCC_3.3V	Power	電源(VCC_3.3V)	-
5	GND	Power	電源(GND)	-

コネクタは実装されておりませんので、必要であればコネクタを実装してください。「表 15.23. CON2 搭載コネクタ例」に記載したコネクタ等が実装可能です。61300511021 等のピンヘッドを実装すると、「15.1. USB シリアル変換アダプタ(Armadillo-640 用)」を接続してシリアルコンソールとして使用することができます。S5B-XH-A、S5B-EH を使用する場合、実装面によっては Armadillo-600 シリーズ オプションケースセット(樹脂製)の蓋が閉まらなくなりますので、オプションケースへの収納を検討している場合はご注意ください。

表 15.23 CON2 搭載コネクタ例

型番	メーカー
61300511021	Würth Electronics
S5B-XH-A	J.S.T.Mfg.
S5B-EH	J.S.T.Mfg.

### 15.6.5. 形状図

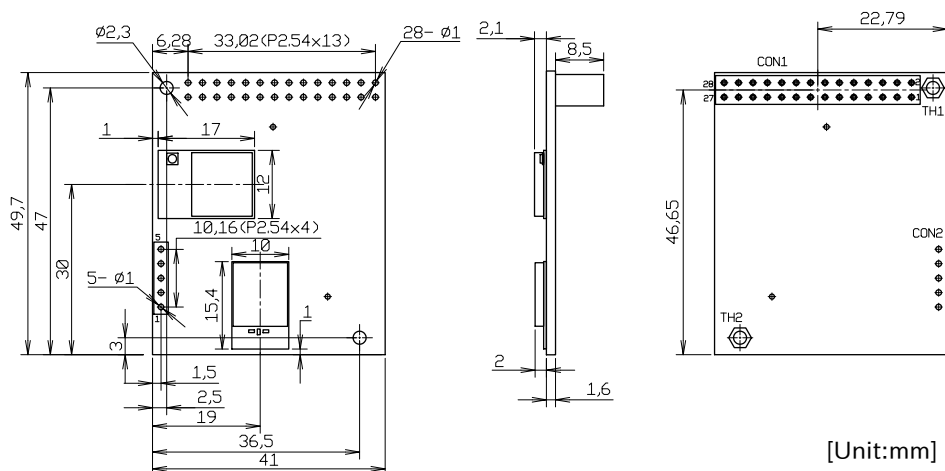


図 15.20 WLAN コンボ、BT/TH オプションモジュール形状図

## 15.6.6. 組み立て

### 15.6.6.1. オプションボードの組み立て

WLAN コンボ、BT/TH オプションモジュールは Armadillo-640 の CON14(拡張インターフェース)の 1 ピンから 4 ピン、CON9(拡張インターフェース)の 1 ピンから 24 ピンに接続します。電波強度等に影響がでますので、Armadillo-640 と WLAN コンボ、BT/TH オプションモジュールは、金属スペーサで固定してください。

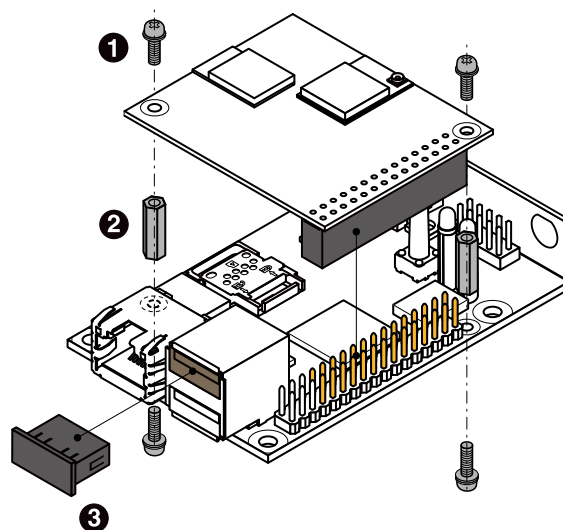


図 15.21 WLAN コンボ、BT/TH オプションモジュールの組み立て

- ❶ なべ小ネジ、スプリングワッシャー、小径平ワッシャー付(M2、L=6mm) x 4
- ❷ 金属スペーサ(M2、L=11mm) x 2
- ❸ USB コネクタキャップ



Armadillo-640 CON1(microSD スロット)に microSD カードを挿抜する際には、WLAN コンボ、BT/TH オプションモジュールを取り外すことを推奨します。組み立てたまま挿抜した場合、microSD カードが正常に挿入されないなどの原因で、動作不良を起こす可能性があります。



付属の USB コネクタキャップは一度嵌めると簡単に取り外すことができません。取り付け箇所に間違いがないか、十分にご確認の上ご使用ください。

### 15.6.6.2. アンテナの組み立て

Sterling LWB5+を使用する場合、外付けアンテナが必要になります。アンテナケーブルコネクタは Sterling LWB5+上のコネクタに接続します。

アンテナは外付けアンテナ固定金具 00(OP-MNT-ANT-MET-00)で Armadillo-640 のねじ穴に固定することが可能です。

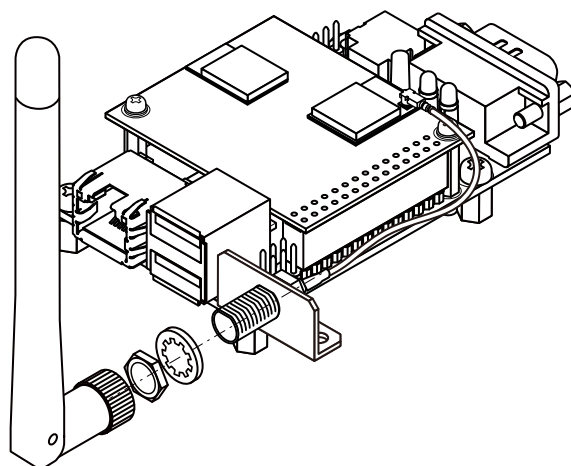


図 15.22 外付けアンテナの組み立て(OP-MNT-ANT-MET-00 使用)

CON3 に実装されている D-Sub9 ピンコネクタの代わりにオプションケース対応のアンテナ固定金具を装着して、アンテナを固定することも可能です。

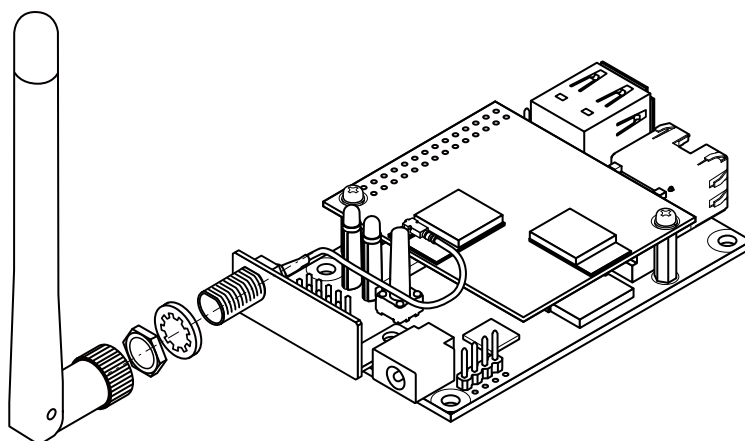


図 15.23 外付けアンテナの組み立て(オプションケース対応のアンテナ固定金具使用)



BTO サービスで、オプションケース対応のアンテナ固定金具を、D-Sub9 ピンコネクタの代わりに装着することが可能です。詳細につきましては、「アットマークテクノ BTO サービス」 [<https://armadillo.atmark-techno.com/services/customize/bto>]をご確認ください。

### 15.6.6.3. ケースの組み立て

オプションケース対応のアンテナ固定金具を使用した場合、組み立て後でも Armadillo-600 シリーズ オプションケース(樹脂製)に収めることが可能です。

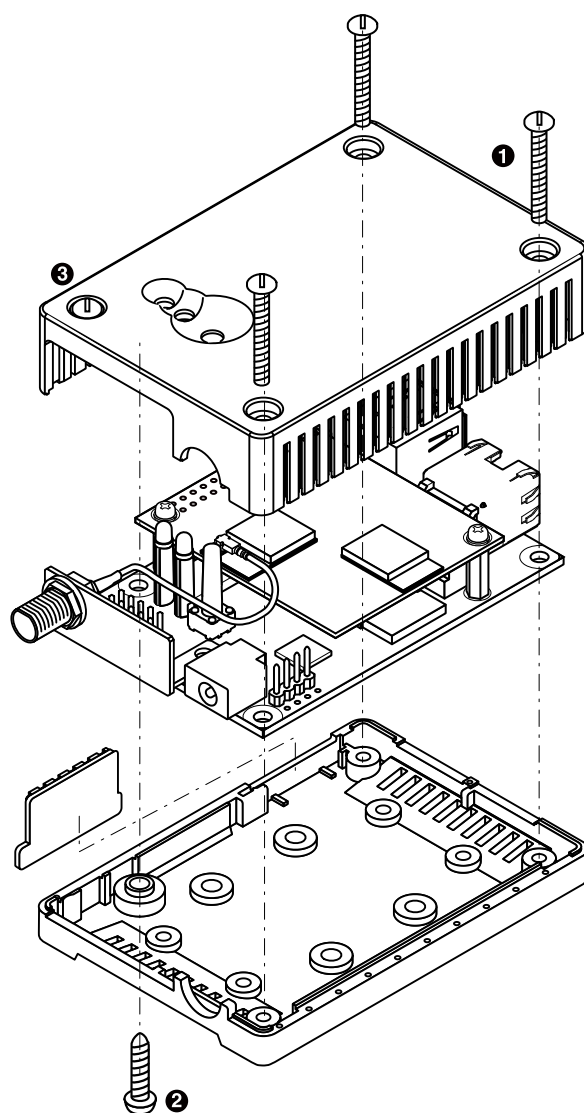


図 15.24 ケースの組み立て

- ❶ タッピングねじ(M2.6、L=20mm) x 3
- ❷ タッピングねじ(M3、L=12mm) x 1
- ❸ 飾りねじ x 1

### 15.6.7. シリアルコンソールの使用方法

シリアルコンソールが必要な場合、WLAN コンボ、BT/TH オプションモジュールを Armadillo-640 の CON9 に接続するため、「15.1. USB シリアル変換アダプタ(Armadillo-640 用)」を接続することができません。

これらの信号線は WLAN コンボ、BT/TH オプションモジュールの CON2(拡張インターフェース)から使用可能です。CON2 に 5 ピンのピンヘッドを実装することで、「15.1. USB シリアル変換アダプタ(Armadillo-640 用)」を接続することができます。

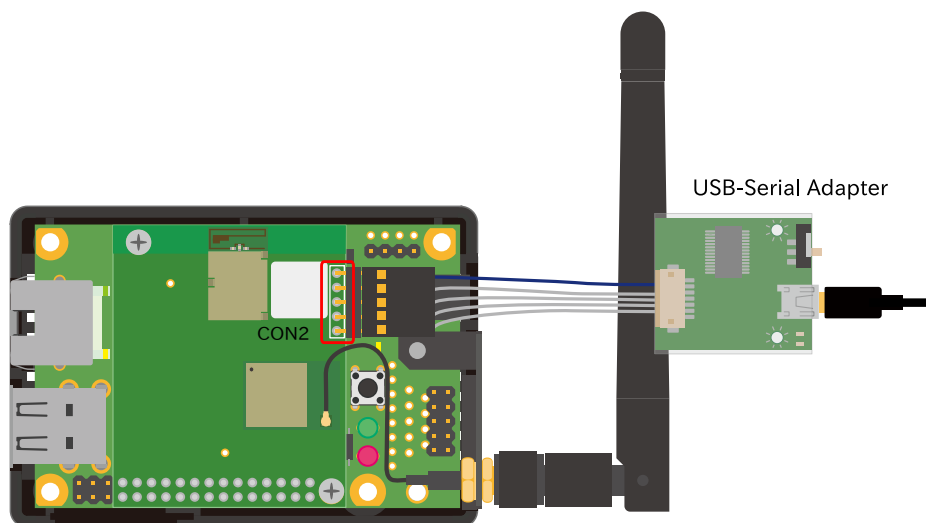


図 15.25 オプションモジュールの CON2 をシリアルコンソールとして使用する場合の接続例

CON3 の D-Sub9 ピンコネクタの代わりにオプションケース対応のアンテナ固定金具を装着している場合、CON4(シリアルインターフェース)に 10 ピンのピンヘッダを実装することで、シリアルコンソールとして使用することができます。CON4 を使用する場合は、シリアルクロスケーブルと D-Sub9/10 ピンシリアル変換ケーブルが必要になります。

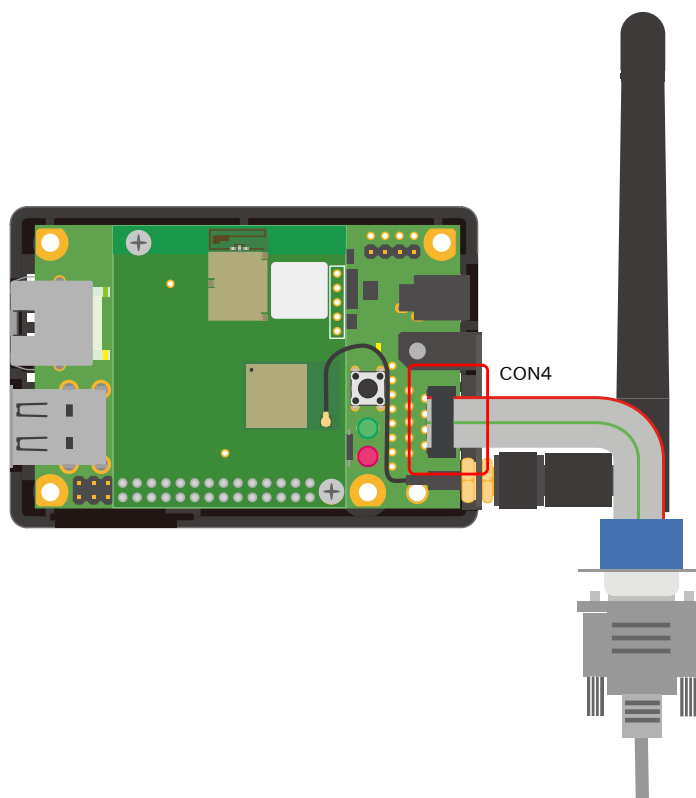


図 15.26 CON4 をシリアルコンソールとして使用する場合の接続例

アンテナの固定が不要な場合は、Armadillo-640 の CON3(シリアルインターフェース)をシリアルコンソールとして使用すると Armadillo-600 シリーズ オプションケース(樹脂製)に入れたままでも、シリ



アルコンソールを利用することが可能です。CON3 を使用する場合は、シリアルクロスケーブルが必要になります。

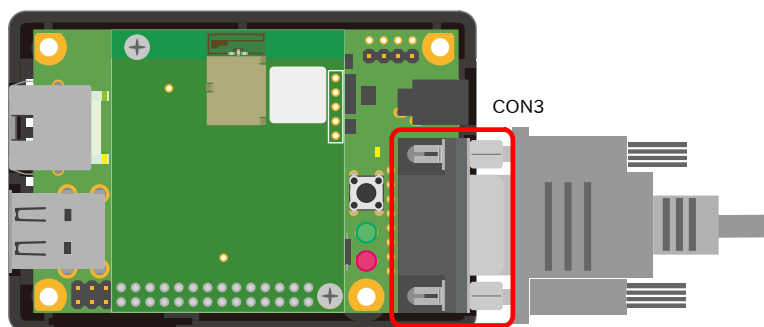


図 15.27 CON3 をシリアルコンソールとして使用する場合の接続例

## 15.7. 無線 LAN 用 外付けアンテナセット 08

### 15.7.1. 概要

無線 LAN 用 外付けアンテナセット 08 は、Armadillo-600 シリーズ WLAN コンボ、BT/TH オプションモジュールに搭載している Laird Connectivity 製の無線 LAN/BT コンボモジュール Sterling LWB5+対応のアンテナセットです。

表 15.24 無線 LAN 用 外付けアンテナセット 08 について

商品名	無線 LAN 用 外付けアンテナセット 08
型番	OP-ANT-WLAN-08K
セット内容	アンテナ(WAND2DBI-SMA-2NB/OxfordTEC)、アンテナケーブル(ケーブル長 100mm)
対応製品	OP-A600-AWLMOD-20, OP-A600-BTTHMOD-21



BTO サービスで、ケーブル長 60mm のアンテナケーブルを選択することが可能です。オプションケースおよびオプションケース対応のアンテナ固定金具を使用する場合は、ケーブルの挟み込みによる断線や、意図しないコネクタ外れを防ぐため、ケーブル長 60mm のアンテナケーブルの使用を推奨いたします。詳細につきましては、「アットマークテクノ BTO サービス」 [<https://armadillo.atmark-techno.com/services/customize/bto>]をご確認ください。

### 15.7.2. 組み立て

#### 15.7.2.1. アンテナケーブルコネクタの嵌合

アンテナケーブルコネクタは、Sterling LWB5+上のコネクタに接続します。

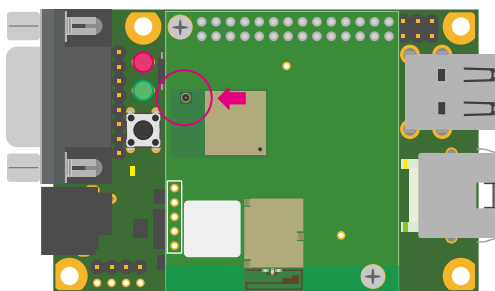


図 15.28 Sterling LWB5+上のコネクタの位置

アンテナケーブルコネクタは、挿抜治具(90609-0001/IPEX)を使用して嵌合するか、手で直接嵌合します。

#### 挿抜治具による嵌合

アンテナケーブルコネクタのストッパーに当たるまで挿抜治具をスライドさせ、コネクタ全体を抱えるようにします。アンテナケーブルコネクタが基板に対し平行になっていることを確認し、垂直に挿抜治具を押してコネクタを嵌合します。

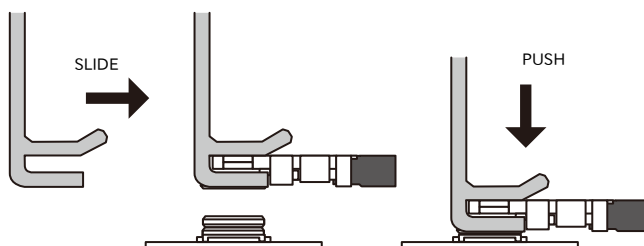


図 15.29 挿抜治具によるアンテナケーブルコネクタの嵌合



アンテナケーブルコネクタは基板に対して平行してから嵌合してください。曲がったまま嵌合すると、コネクタ破損の原因となります。

#### 手で直接嵌合

アンテナケーブルを持ち、Sterling LWB5+上のアンテナコネクタにアンテナケーブルのコネクタをセットします。セットしたら前後に軽く動かし、動かないことを確認します。

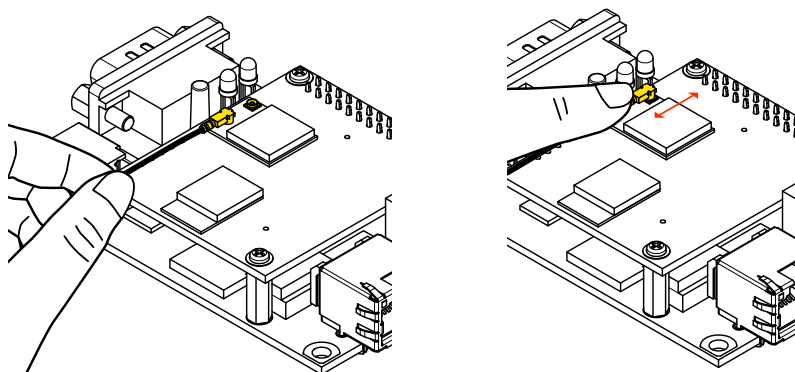


図 15.30 アンテナケーブルコネクタのセット

コネクタのセンターを真上から押し、カチッという音がすると嵌合完了です。

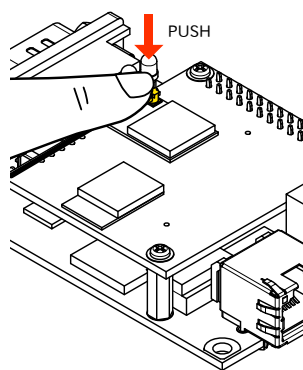


図 15.31 アンテナケーブルコネクタの嵌合

### 15.7.2.2. アンテナケーブルコネクタの抜去

アンテナのケーブルコネクタは、ケーブルコネクタ首部へのストレスを避けるため、挿抜治具 (90609-0001/IPEX) を使用して抜去してください。

ケーブルコネクタのストッパーに当たるまで挿抜治具をスライドさせ、ケーブルコネクタ全体を抱えるようにします。基板と垂直に挿抜治具を引き上げてコネクタを抜去します。

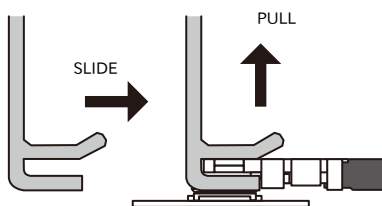


図 15.32 挿抜治具によるアンテナケーブルコネクタの抜去



挿抜治具は必ず基板と垂直に引き上げてください。ひねったり、ななめに引き上げたりした場合、コネクタ破損の原因となります。

### 15.7.3. 形状図

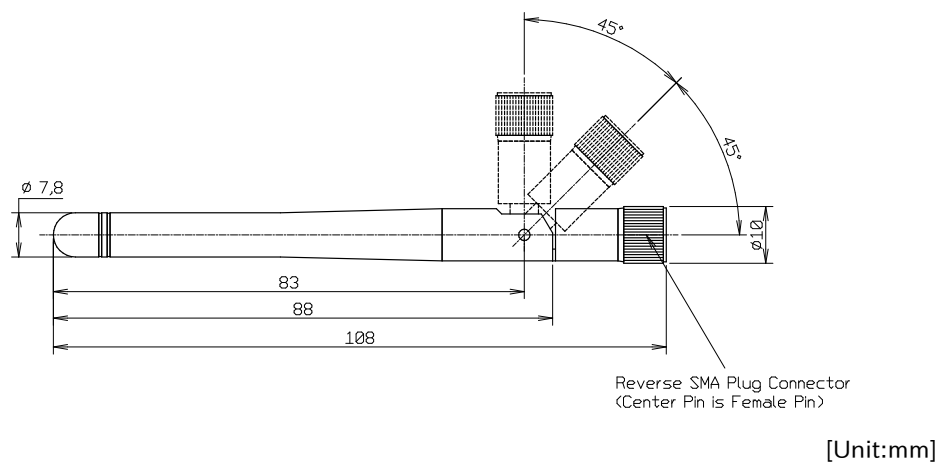


図 15.33 無線 LAN 用 外付けアンテナ 08 形状図

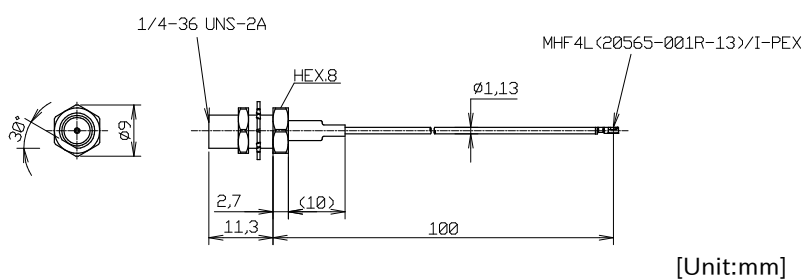


図 15.34 無線 LAN 用 外付けアンテナケーブル 08 形状図

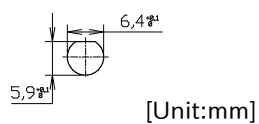


図 15.35 無線 LAN 用 外付けアンテナ 取り付け穴寸法

## 15.8. 外付けアンテナ固定金具 00

### 15.8.1. 概要

アンテナを固定する金具です。

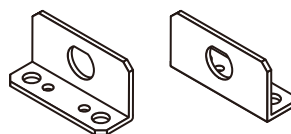


図 15.36 外付けアンテナ固定金具 00 の外観

アンテナ固定金具を使用すると、Armadillo-640 の USB コネクタ横のねじ穴にアンテナを固定することができます。

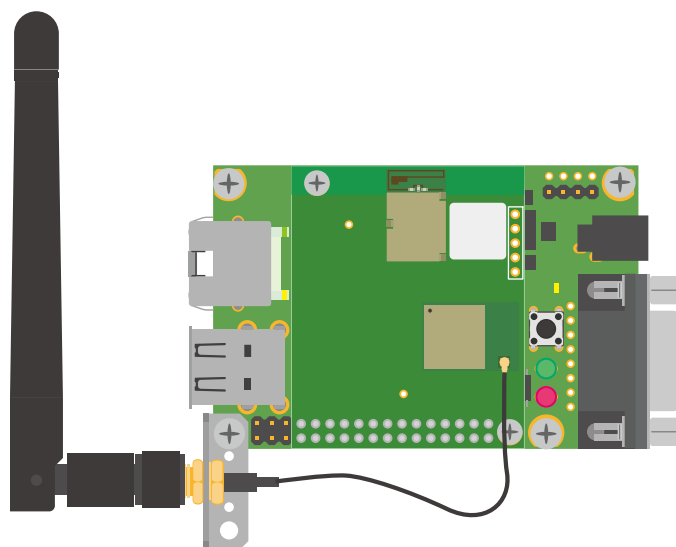


図 15.37 Armadillo-640 へのアンテナ固定金具の固定

表 15.25 外付けアンテナ固定金具 00 について

商品名	外付けアンテナ固定金具 00
型番	OP-MNT-ANT-MET-00
セット内容	アンテナ固定金具本体 <sup>[a]</sup>

<sup>[a]</sup>固定用のねじ、スペーサはセットに含まれません。別途ご準備ください。Armadillo-640 ベーシックモデル 開発セットに含まれるねじ、スペーサで固定することができます。

表 15.26 外付けアンテナ固定金具 00 の仕様

材質	鉄
板厚	1.0mm



オプションケース対応のアンテナ固定金具とは違う金具です。オプションケース対応のアンテナ固定金具が必要な場合は、BTO サービスをご利用ください。

## 15.8.2. 組み立て

「図 15.38. Armadillo-640 へのアンテナ固定金具の固定」のように M3 のねじとスペーサで Armadillo-640 のねじ穴に固定します。

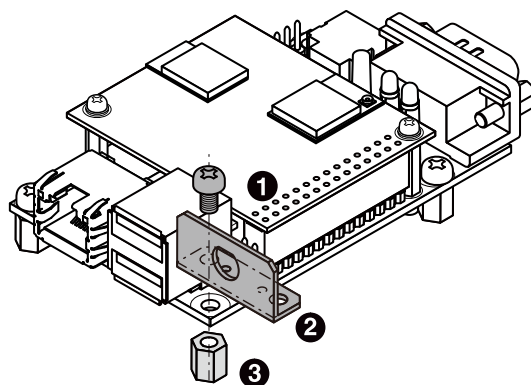


図 15.38 Armadillo-640 へのアンテナ固定金具の固定

- ❶ なべ小ねじ(M3、L=5)
- ❷ アンテナ固定金具
- ❸ スペーサ(M3、L=8mm)

アンテナは「図 15.39. アンテナ固定金具へのアンテナの固定」のように固定します。

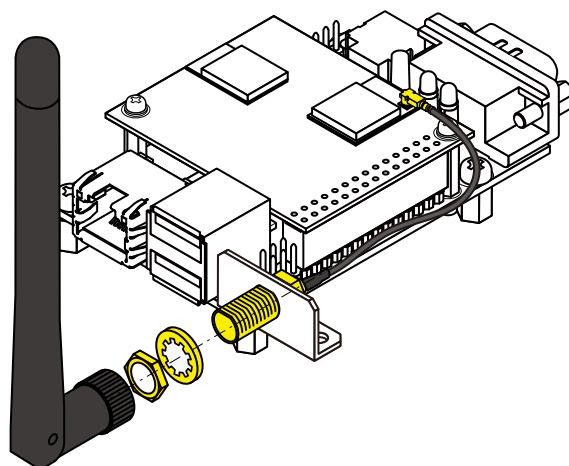


図 15.39 アンテナ固定金具へのアンテナの固定

### 15.8.3. 形状図

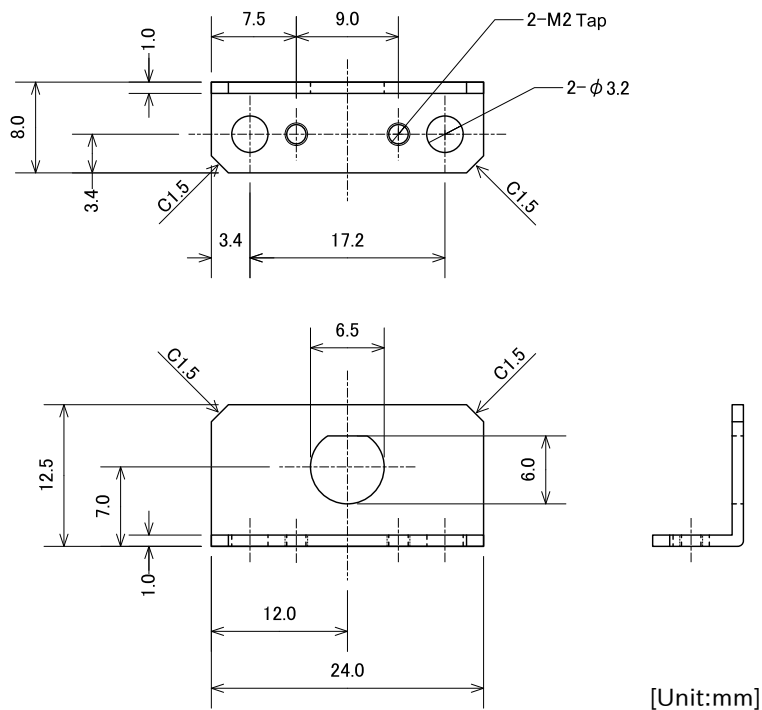


図 15.40 アンテナ固定金具 形状図

## 改訂履歴

バージョン	年月日	改訂内容
3.0.0	2023/06/29	<ul style="list-style-type: none"><li>・ 初版発行</li></ul>
3.1.0	2023/07/26	<ul style="list-style-type: none"><li>・ Armadillo-610 拡張ボード参考回路の情報を Sterling LWB5+ 向けのものに修正</li><li>・ 「9.3. ABOS Web の設定機能一覧と設定手順」 に 「9.3.6. VPN 設定」 を追加</li><li>・ 「9.3. ABOS Web の設定機能一覧と設定手順」 に 「9.3.7. コンテナ管理」 を追加</li><li>・ 「9.3. ABOS Web の設定機能一覧と設定手順」 に 「9.3.8. SWU インストール」 を追加</li><li>・ 「10.1. CUI アプリケーションを開発する」 の内容を最新のソフトウェアのものに更新</li><li>・ 「10.1. CUI アプリケーションを開発する」 に ssh_known_hosts に関する注意を追加</li><li>・ 「10.3.2. pod の作成」 の説明を set_infra_image を使わないものに変更</li><li>・ 誤記修正</li></ul>



Armadillo-640 製品マニュアル  
Version 3.1.0  
2023/07/27