

Armadillo-640 製品マニュアル

A6400-U00Z

A6400-D00Z

A6400-B00Z

A6400-N00Z

Version 4.16.0

2024/11/27

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 © 2023-2024 Atmark Techno, Inc.

Version 4.16.0
2024/11/27

目次

1. はじめに	22
1.1. 本書について	22
1.1.1. 本書で扱うこと	22
1.1.2. 本書で扱わないこと	22
1.1.3. 本書で必要となる知識と想定する読者	23
1.1.4. 本書の構成	23
1.1.5. フォント	24
1.1.6. コマンド入力例	24
1.1.7. アイコン	24
1.1.8. ユーザー限定コンテンツ	25
1.1.9. 本書および関連ファイルのバージョンについて	25
1.2. 注意事項	25
1.2.1. 安全に関する注意事項	25
1.2.2. 取扱い上の注意事項	26
1.2.3. 製品の保管について	28
1.2.4. ソフトウェア使用に関する注意事項	28
1.2.5. 本製品を廃棄する場合について	29
1.2.6. 電波障害について	30
1.2.7. 保証について	30
1.2.8. 輸出について	30
1.2.9. 商標について	30
1.2.10. 無線モジュールの安全規制について	31
1.3. 謝辞	31
2. 製品概要	32
2.1. 製品の特長	32
2.1.1. Armadillo とは	32
2.1.2. Armadillo-640 とは	32
2.1.3. Armadillo Base OS とは	34
2.1.4. Armadillo Base OS のメンテナンスポリシーとアップデートの推奨	36
2.1.5. Armadillo Twin とは	36
2.2. 製品ラインアップ	38
2.2.1. Armadillo-640 ベーシックモデル開発セット	38
2.2.2. Armadillo-640 量産ボード	39
2.3. 仕様	39
2.4. インターフェースレイアウト	40
2.5. ブロック図	40
2.6. 使用可能なストレージデバイス	41
2.7. ストレージデバイスのパーティション構成	42
2.8. ソフトウェアのライセンス	42
3. 開発編	44
3.1. 開発の準備	44
3.1.1. 準備するもの	44
3.1.2. 仮想環境のセットアップ	45
3.1.3. VS Code のセットアップ	51
3.1.4. Armadillo の初期化と ABOS のアップデート	53
3.1.5. Armadillo に初期設定をインストールする	55
3.1.6. Python アプリケーションで動作確認する	65
3.1.7. シリアルコンソールを使用する	73
3.1.8. ユーザー登録	83
3.2. アプリケーション開発の流れ	84

3.3. 開発前に知っておくべき Armadillo Base OS の機能・特徴	86
3.3.1. 一般的な Linux OS 搭載組み込み機器との違い	86
3.3.2. Armadillo Base OS 搭載機器のソフトウェア開発手法	87
3.3.3. アップデート機能について	87
3.3.4. ファイルの取り扱いについて	93
3.3.5. インストールディスクについて	95
3.4. ハードウェアの設計	97
3.4.1. 信頼性試験データについて	97
3.4.2. 放射ノイズ	97
3.4.3. ESD/雷サージ	98
3.4.4. 放熱	98
3.4.5. 周辺装置との接続	98
3.4.6. 拡張ボードの設計	99
3.4.7. 電氣的仕様	101
3.4.8. 形状図	106
3.4.9. オプション品	108
3.5. Device Tree をカスタマイズする	108
3.5.1. Linux カーネルソースコードの取得	108
3.5.2. at-dtweb のインストール	108
3.5.3. at-dtweb の起動	109
3.5.4. Device Tree をカスタマイズ	110
3.5.5. DT overlay によるカスタマイズ	117
3.6. インターフェースの使用方法和デバイスの接続方法	118
3.6.1. SD カードを使用する	119
3.6.2. Ethernet を使用する	123
3.6.3. UART を使用する	125
3.6.4. USB デバイスを使用する	127
3.6.5. WLAN を使用する	132
3.6.6. BT デバイスを使用する	133
3.6.7. Thread デバイスを扱う	137
3.6.8. 音声出力を行う	138
3.6.9. GPIO を制御する	138
3.6.10. I2C デバイスを使用する	141
3.6.11. SPI デバイスを使用する	143
3.6.12. CAN デバイスを使用する	143
3.6.13. PWM を使用する	144
3.6.14. JTAG デバッガを使用する	145
3.6.15. LCD を使用する	146
3.6.16. 電源を入力する	151
3.6.17. 起動デバイスを変更する	153
3.6.18. ユーザースイッチを使用する	154
3.6.19. LED を使用する	155
3.6.20. RTC を使用する	157
3.6.21. Wi-SUN デバイスを扱う	158
3.6.22. EnOcean デバイスを扱う	159
3.7. ソフトウェアの設計	159
3.7.1. 開発者が開発するもの、開発しなくていいもの	159
3.7.2. ユーザーアプリケーションの設計	160
3.7.3. ログの設計	160
3.7.4. ウォッチドッグタイマー	161
3.7.5. コンテナに Armadillo の情報を渡す方法	162
3.7.6. Armadillo Base OS のデフォルトで開放しているポート	162
3.8. ネットワーク設定	164

3.8.1. ABOS Web とは	164
3.8.2. ABOS Web へのアクセス	165
3.8.3. ABOS Web のパスワード登録	168
3.8.4. ABOS Web のパスワード変更	171
3.8.5. ABOS Web の設定操作	172
3.8.6. ログアウト	172
3.8.7. WWAN 設定	172
3.8.8. WLAN 設定	174
3.8.9. 各接続設定 (各ネットワークインターフェースの設定)	178
3.8.10. DHCP サーバー設定	179
3.8.11. NAT 設定	180
3.8.12. VPN 設定	182
3.8.13. 状態一覧	184
3.9. ABOS Web をカスタマイズする	184
3.10. Network Time Protocol (NTP, ネットワーク・タイム・プロトコル) の設定	187
3.11. Armadillo Twin を体験する	188
3.12. ABOSDE によるアプリケーションの開発	188
3.12.1. ABOSDE の対応言語	189
3.12.2. 参照する開発手順の章の選択	189
3.13. CUI アプリケーションの開発	190
3.13.1. CUI アプリケーション開発の流れ	190
3.13.2. ATDE 上でのセットアップ	190
3.13.3. アプリケーション開発	191
3.13.4. コンテナのディストリビューション	195
3.13.5. Armadillo に転送するディレクトリ及びファイル	195
3.13.6. コンテナ内のファイル一覧表示	195
3.13.7. Armadillo 上でのセットアップ	207
3.13.8. SBOM 生成に関する設定	212
3.13.9. リリース版のビルド	212
3.13.10. 製品への書き込み	212
3.13.11. Armadillo 上のコンテナイメージの削除	212
3.14. C 言語によるアプリケーションの開発	212
3.14.1. C 言語によるアプリケーション開発の流れ	213
3.14.2. ATDE 上でのセットアップ	213
3.14.3. アプリケーション開発	214
3.14.4. コンテナのディストリビューション	218
3.14.5. コンテナ内のファイル一覧表示	218
3.14.6. Armadillo に転送するディレクトリ及びファイル	230
3.14.7. Armadillo 上でのセットアップ	230
3.14.8. SBOM 生成に関する設定	235
3.14.9. リリース版のビルド	235
3.14.10. 製品への書き込み	235
3.14.11. Armadillo 上のコンテナイメージの削除	235
3.15. SBOM 生成に関わる設定を行う	235
3.15.1. SBOM 生成に必要なファイルを確認する	236
3.16. システムのテストを行う	237
3.16.1. ランニングテスト	237
3.16.2. 異常系における挙動のテスト	238
3.17. ユーザー設定とユーザーデータを一括削除する	238
4. 量産編	240
4.1. 概略	240
4.1.1. Armadillo Twin を契約する	240
4.1.2. リードタイムと在庫	241

4.1.3. Armadillo 納品後の製造・量産作業	241
4.2. BTO サービスを使わない場合と使う場合の違い	242
4.2.1. BTO サービスを利用しない(標準ラインアップ品)	242
4.2.2. BTO サービスを利用する	242
4.3. 量産時のイメージ書き込み手法	242
4.4. インストールディスクを用いてイメージ書き込みする	243
4.4.1. /etc/swupdate_preserve_file への追記	243
4.4.2. Armadillo Base OS の更新	244
4.4.3. パスワードの確認と変更	244
4.4.4. 開発したシステムをインストールディスクにする	245
4.4.5. VS Code を使用して生成する	245
4.4.6. インストールディスクの動作確認を行う	249
4.4.7. コマンドラインから生成する	249
4.4.8. インストールの実行	255
4.5. SWUpdate を用いてイメージ書き込みする	256
4.5.1. SWU イメージの準備	256
4.5.2. desc ファイルの記述	256
4.6. イメージ書き込み後の動作確認	257
4.7. 量産時の組み立て	257
4.7.1. オプションの組み付け	257
5. 運用編	258
5.1. Armadillo Twin に Armadillo を登録する	258
5.1.1. Armadillo の設置前に登録する場合	258
5.1.2. Armadillo の設置後に登録する場合	258
5.2. Armadillo を設置する	258
5.2.1. 設置場所	258
5.2.2. ケーブルの取り回し	258
5.2.3. サージ対策	258
5.2.4. Armadillo の状態を表すインジケータ	258
5.2.5. 個体識別情報の取得	259
5.2.6. 電源を切る	261
5.3. ABOSDE で開発したアプリケーションをアップデートする	261
5.3.1. アプリケーションのアップデート手順	261
5.4. Armadillo のソフトウェアをアップデートする	262
5.4.1. SWU イメージの作成	262
5.4.2. mkswu の desc ファイルを作成する	262
5.4.3. desc ファイルから SWU イメージを生成する	264
5.4.4. イメージのインストール	264
5.5. Armadillo Twin から複数の Armadillo をアップデートする	264
5.6. eMMC の寿命を確認する	265
5.6.1. eMMC について	265
5.6.2. eMMC 予備領域の確認方法	265
5.7. Armadillo の部品変更情報を知る	265
5.8. Armadillo を廃棄する	266
6. 応用編	267
6.1. persist_file について	267
6.2. コンテナ	269
6.2.1. Podman - コンテナ仮想化ソフトウェアとは	269
6.2.2. コンテナの基本的な操作	269
6.2.3. コンテナとコンテナに関連するデータを削除する	284
6.2.4. コンテナ起動設定ファイルを作成する	286
6.2.5. アットマークテクノが提供するイメージを使う	294
6.2.6. alpine のコンテナイメージをインストールする	296

6.2.7. コンテナのネットワークを扱う	297
6.2.8. コンテナ内にサーバを構築する	299
6.2.9. 画面表示を行う	302
6.2.10. パワーマネジメント機能を使う	304
6.2.11. コンテナからの poweroff 及び reboot	307
6.2.12. 異常検知	307
6.3. swupdate を使用してアップデートする	308
6.3.1. swupdate で可能なアップデート	308
6.3.2. コンテナのアップデート、ユーザーデータディレクトリや Armadillo Base OS の 差分アップデート	309
6.3.3. Armadillo Base OS の一括アップデート	312
6.3.4. ブートローダーのアップデート	316
6.3.5. swupdate がエラーする場合の対処	316
6.4. mkswu の .desc ファイルを編集する	316
6.4.1. インストールバージョンを指定する	316
6.4.2. Armadillo へファイルを転送する	318
6.4.3. Armadillo 上で任意のコマンドを実行する	319
6.4.4. Armadillo にファイルを転送し、そのファイルをコマンド内で使用する	319
6.4.5. 動作中の環境でのコマンドの実行	319
6.4.6. Armadillo にコンテナイメージを転送する	320
6.4.7. Armadillo のブートローダーを更新する	320
6.4.8. SWU イメージの設定関連	320
6.4.9. Armadillo 上のコンテナイメージと自動起動用 conf ファイルを削除する	321
6.4.10. SWUpdate 実行中/完了後の挙動を指定する	321
6.4.11. desc ファイル設定例	321
6.5. swupdate_preserve_files について	323
6.6. SWU イメージの内容の確認	324
6.7. SWUpdate と暗号化について	324
6.8. SWUpdate の署名鍵と証明書の更新	325
6.8.1. 署名鍵と証明書の追加	325
6.8.2. 署名鍵と証明書の削除	326
6.9. Web UI から Armadillo をセットアップする (ABOS Web)	327
6.9.1. ABOS Web ではできないこと	327
6.9.2. ABOS Web の設定機能一覧と設定手順	327
6.9.3. コンテナ管理	328
6.9.4. SWU インストール	328
6.9.5. 時刻設定	330
6.9.6. アプリケーション向けのインターフェース (Rest API)	332
6.9.7. カスタマイズ	351
6.9.8. ユーザー設定とユーザーデータの削除	351
6.9.9. ABOS Web を停止する	351
6.9.10. ABOS Web を起動する	352
6.10. ABOSDE から ABOS Web の機能を使用する	352
6.10.1. Armadillo の SWU バージョンを取得する	354
6.10.2. Armadillo のコンテナの情報を取得する	354
6.10.3. Armadillo のコンテナを起動・停止する	355
6.10.4. Armadillo のコンテナのログを取得する	357
6.10.5. Armadillo に SWU をインストールする	357
6.11. ssh 経由で Armadillo Base OS にアクセスする	358
6.12. コマンドラインからネットワーク設定をする	358
6.12.1. 接続可能なネットワーク	359
6.12.2. IP アドレスの確認方法	359
6.12.3. ネットワークの設定方法	359

6.12.4. nmcli の基本的な使い方	360
6.12.5. 有線 LAN	363
6.12.6. ファイアウォールの設定方法	364
6.13. ストレージの操作	365
6.13.1. ストレージ内にアクセスする	365
6.13.2. ストレージを安全に取り外す	366
6.13.3. ストレージのパーティション変更とフォーマット	366
6.14. ボタンやキーを扱う	368
6.14.1. SW1 の短押しと長押しの対応	368
6.14.2. USB キーボードの対応	369
6.14.3. Armadillo 起動時にのみボタンに反応する方法	370
6.15. 動作中の Armadillo の温度を測定する	370
6.15.1. 温度測定的重要性	370
6.15.2. atmark-thermal-profiler をインストールする	371
6.15.3. atmark-thermal-profiler を実行・停止する	371
6.15.4. atmark-thermal-profiler が出力するログファイルを確認する	371
6.15.5. 温度測定結果の分析	372
6.15.6. Armadillo Twin から Armadillo の温度を確認する	374
6.15.7. 温度センサーの仕様	374
6.16. Armadillo Base OS をアップデートする	374
6.17. ロールバック状態を確認する	374
6.18. Armadillo 起動時にコンテナの外でスクリプトを実行する	375
6.19. u-boot の環境変数の設定	376
6.19.1. u-boot の環境変数の変更を制限する	378
6.20. SD ブートの活用	379
6.20.1. ブートディスクの作成	379
6.20.2. SD ブートの実行	381
6.21. Armadillo のソフトウェアをビルドする	382
6.21.1. ブートローダーをビルドする	382
6.21.2. Linux カーネルをビルドする	383
6.21.3. Alpine Linux ルートファイルシステムをビルドする	386
6.22. SBOM の提供	390
6.22.1. SBOM について	390
6.22.2. SBOM の利点	390
6.22.3. ビルドしたルートファイルシステムの SBOM を作成する	390
6.22.4. SWU イメージと同時に SBOM を作成する	391
6.23. eMMC の GPP(General Purpose Partition) を利用する	392
6.23.1. squashfs イメージを作成する	392
6.23.2. squashfs イメージを書き込む	393
6.23.3. GPP への書き込みを制限する	393
6.23.4. 起動時に squashfs イメージをマウントされるようにする	394
6.24. 動作ログ	394
6.24.1. 動作ログについて	394
6.24.2. 動作ログを取り出す	394
6.24.3. ログファイルのフォーマット	395
6.24.4. ログ用パーティションについて	395
6.24.5. /var/log/ 配下のログに関して	395
6.25. シリアル通信ソフトウェア(minicom)のセットアップ	396
6.26. vi エディタを使用する	399
6.26.1. vi の起動	399
6.26.2. 文字の入力	399
6.26.3. カーソルの移動	400
6.26.4. 文字の削除	400

6.26.5. 保存と終了	401
6.27. eFuse を変更する	401
6.27.1. ブートモードとジャンパーピン	402
6.27.2. eFuse の書き換え	403
6.27.3. Boot From Fuses モード	404
6.28. オプション品	406
6.28.1. USB シリアル変換アダプタ(Armadillo-640 用)	406
6.28.2. Armadillo-600 シリーズ オプションケース(樹脂製)	408
6.28.3. Armadillo-600 シリーズ オプションケース(金属製)	411
6.28.4. LCD オプションセット(7 インチタッチパネル WVGA 液晶)	413
6.28.5. Armadillo-600 シリーズ RTC オプションモジュール	414
6.28.6. Armadillo-600 シリーズ WLAN コンボ、BT/TH オプションモジュール	420
6.28.7. 無線 LAN 用 外付けアンテナセット 08	430
6.28.8. 外付けアンテナ固定金具 00	433

目次

1.1. 製品化までのロードマップ	23
1.2. WLAN+BT コンボモジュール: Sterling LWB5+ 認証マーク	31
1.3. EYSKBNZWB 認証マーク	31
2.1. Armadillo-640 とは	33
2.2. Armadillo Base OS とは	34
2.3. コンテナによるアプリケーションの運用	35
2.4. ロールバックの仕組み	35
2.5. Armadillo Twin とは	37
2.6. インターフェースレイアウト	40
2.7. Armadillo-640 ブロック図	41
3.1. GNOME 端末の起動	47
3.2. GNOME 端末のウィンドウ	47
3.3. ソフトウェアをアップデートする	48
3.4. ATDE にデバイスを接続する	48
3.5. 共有フォルダー設定を開く	49
3.6. 共有フォルダー設定	50
3.7. 共有フォルダーの追加	50
3.8. 「ファイル」に表示される共有フォルダー	51
3.9. VS Code を起動する	51
3.10. VS Code に開発用エクステンションをインストールする	52
3.11. Armadillo-640 を初期化する接続	54
3.12. ジャンパピンの位置	55
3.13. initial_setup.swu を作成する	56
3.14. initial_setup.swu 初回生成時の各種設定	56
3.15. ABOS にアクセスするための接続	58
3.16. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする	59
3.17. ABOSDE に表示されている Armadillo を更新する	59
3.18. ABOSDE を使用して ABOS Web を開く	60
3.19. パスワード登録画面	61
3.20. パスワード登録完了画面	61
3.21. ログイン画面	62
3.22. トップページ	62
3.23. SWU インストール	63
3.24. SWU インストールに成功した画面	63
3.25. プロジェクトを作成する	65
3.26. プロジェクト名を入力する	65
3.27. VS Code で初期設定を行う	66
3.28. VS Code のターミナル	66
3.29. SSH 用の鍵を生成する	66
3.30. VS Code でコンテナイメージの作成を行う	67
3.31. コンテナイメージの作成完了	68
3.32. ABOSDE で Armadillo に SWU をインストール	68
3.33. ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する	69
3.34. ssh_config を編集する	69
3.35. Armadillo 上でアプリケーションを実行する	70
3.36. 実行時に表示されるメッセージ	70
3.37. アプリケーションを終了する	71
3.38. Armadillo 上のコンテナイメージを削除する	72
3.39. スライドスイッチの設定	73
3.40. CON9-USB シリアル変換アダプタ接続図	74

3.41. シリアルコンソールを使用する配線例	75
3.42. COM7 が競合している状態	76
3.43. Bluetooth に割当の COM を変更した状態	76
3.44. minicom 起動方法	77
3.45. minicom 終了確認	82
3.46. アプリケーション開発の流れ	85
3.47. persist_file コマンド実行例	94
3.48. chattr によって copy-on-write を無効化する例	95
3.49. Armadillo-640 の接続例	98
3.50. 電源回路の構成	103
3.51. 電源シーケンス	104
3.52. 基板形状および固定穴寸法	106
3.53. コネクタ中心寸法(A 面側)	106
3.54. コネクタ中心寸法(B 面側)	107
3.55. コネクタ穴寸法	107
3.56. at-dtweb の起動開始	109
3.57. ボード選択画面	109
3.58. Linux カーネルディレクトリ選択画面	110
3.59. at-dtweb 起動画面	110
3.60. UART1 (RXD/TXD) のドラッグ	111
3.61. CON9 3/5 ピンへのドロップ	111
3.62. 信号名の確認	112
3.63. プロパティの設定	113
3.64. プロパティの保存	113
3.65. 全ての機能の削除	114
3.66. UART1 (RXD/TXD) の削除	115
3.67. DTS/DTB の生成	116
3.68. dtbo/desc の生成完了	116
3.69. /boot/overlays.txt の変更例	117
3.70. Armadillo-640 のインターフェース	118
3.71. カバーのロックを解除する	120
3.72. カバーを開ける	120
3.73. microSD カードの挿抜	121
3.74. カードマークの確認	121
3.75. カバーを閉める	121
3.76. カバーをロックする	122
3.77. 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	127
3.78. 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	127
3.79. USB OTG2 の接続先の変更	128
3.80. USB ホストインターフェースの電源制御	128
3.81. USB シリアルデバイスを扱うためのコンテナ作成例	129
3.82. setserial コマンドによる USB シリアルデバイス設定の確認例	129
3.83. USB カメラを扱うためのコンテナ作成例	130
3.84. USB メモリをホスト OS 側でマウントする例	130
3.85. 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	130
3.86. USB メモリに保存されているデータの確認例	131
3.87. USB メモリをマウントするためのコンテナ作成例	131
3.88. コンテナ内から USB メモリをマウントする例	131
3.89. コンソールを CON3 に移動(WLAN)	132
3.90. DT overlay の設定(WLAN)	133
3.91. コンソールを CON3 に移動(BT/TH)	134
3.92. DT overlay の設定(BT/TH)	134
3.93. BT/TH オプションモジュールの BT 機能を有効化する	135

3.94. Sterling LWB5+ の BT 機能を無効化する	135
3.95. Bluetooth を扱うコンテナの作成例	135
3.96. Bluetooth を起動する実行例	136
3.97. bluetoothctl コマンドによるスキャンとペアリングの例	136
3.98. BT/TH オプションモジュールの Thread 機能を有効化する	137
3.99. Thread デバイスを扱うためのコンテナ作成例	137
3.100. 音声出力を行うためのコンテナ作成例	138
3.101. alsactl コマンドによる音声出力を行う例	138
3.102. GPIO を扱うためのコンテナ作成例	139
3.103. コンテナ内からコマンドで GPIO を操作する例	139
3.104. gpiodetect コマンドの実行	140
3.105. gpioinfo コマンドの実行	140
3.106. I2C を扱うためのコンテナ作成例	142
3.107. i2cdetect コマンドによる確認例	142
3.108. SPI を扱うためのコンテナ作成例	143
3.109. spi-config コマンドによる確認例	143
3.110. CAN を扱うためのコンテナ作成例	144
3.111. CAN の設定例	144
3.112. PWM を扱うためのコンテナ作成例	145
3.113. PWM の動作設定例	145
3.114. リセットシーケンス	145
3.115. LCD の接続方法	150
3.116. フレキシブルフラットケーブルの形状	150
3.117. AC アダプタの極性マーク	151
3.118. バックアップ電源供給	152
3.119. 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	154
3.120. evtest コマンドによる確認例	155
3.121. LED を扱うためのコンテナ作成例	156
3.122. LED の点灯/消灯の実行例	156
3.123. LED の状態を表示する	156
3.124. 対応している LED トリガを表示	157
3.125. LED のトリガに heartbeat を指定する	157
3.126. RTC を扱うためのコンテナ作成例	158
3.127. hwclock コマンドによる RTC の時刻表示と設定例	158
3.128. Wi-SUN デバイスを扱うためのコンテナ作成例	158
3.129. EnOcean デバイスを扱うためのコンテナ作成例	159
3.130. 開発者が開発するもの、開発しなくていいもの	160
3.131. 現在の面の確認方法	161
3.132. add_args を用いてコンテナに情報を渡すための書き方	162
3.133. add_args を用いてコンテナに情報を渡す例	162
3.134. avahi-daemon を停止する	163
3.135. avahi-daemon を起動する	163
3.136. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする	166
3.137. ABOSDE を使用して ABOS Web を開く	167
3.138. ABOSDE に表示されている Armadillo を更新する	167
3.139. パスワード登録画面	168
3.140. パスワード登録完了画面	169
3.141. ログイン画面	170
3.142. トップページ	171
3.143. ログイン画面	171
3.144. WWAN 設定画面	173
3.145. WLAN クライアント設定画面	175
3.146. WLAN アクセスポイント設定画面	177

3.147. 現在の接続情報画面	178
3.148. LAN 接続設定で固定 IP アドレスに設定した画面	179
3.149. eth0 に対する DHCP サーバー設定	180
3.150. LTE を宛先インターフェースに指定した設定	181
3.151. 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	182
3.152. VPN 設定	183
3.153. ABOS Web のカスタマイズ設定	185
3.154. メニュー変更画面 (一部)	187
3.155. chronydc のコンフィグの変更例	188
3.156. 参照する開発手順の章を選択する流れ	189
3.157. CUI アプリケーション開発の流れ	190
3.158. プロジェクトを作成する	191
3.159. プロジェクト名を入力する	191
3.160. VS Code で my_project を起動する	192
3.161. 初期設定を行う	192
3.162. VS Code で初期設定を行う	193
3.163. VS Code のターミナル	193
3.164. SSH 用の鍵を生成する	193
3.165. VS Code でコンテナイメージの作成を行う	194
3.166. コンテナイメージの作成完了	194
3.167. BLE パッケージをインストールする	195
3.168. コンテナ内のファイル一覧を表示するタブ	196
3.169. コンテナ内のファイル一覧の例	196
3.170. resources ディレクトリ	197
3.171. コンテナ内のファイル一覧を再表示するボタン	198
3.172. container/resources 下にファイルを追加するボタン	199
3.173. ファイル名を入力	199
3.174. 追加されたファイルの表示	200
3.175. container/resources 下にフォルダーを追加するボタン	201
3.176. container/resources 下にあるファイルを開くボタン	202
3.177. container/resources 下にあるファイルを削除するボタン	203
3.178. コンテナ内のファイルを container/resources 下に保存するボタン	204
3.179. 編集前のファイルを示すマーク	205
3.180. 編集後のファイルを示すマーク	206
3.181. コンテナ内にコピーされないことを示すマーク	207
3.182. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする	208
3.183. ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する	209
3.184. ABOSDE に表示されている Armadillo を更新する	210
3.185. ssh_config を編集する	210
3.186. Armadillo 上でアプリケーションを実行する	211
3.187. 実行時に表示されるメッセージ	211
3.188. アプリケーションを終了する	211
3.189. リリース版をビルドする	212
3.190. C 言語によるアプリケーション開発の流れ	213
3.191. プロジェクトを作成する	214
3.192. プロジェクト名を入力する	214
3.193. VS Code で my_project を起動する	215
3.194. 初期設定を行う	215
3.195. VS Code で初期設定を行う	216
3.196. VS Code のターミナル	216
3.197. SSH 用の鍵を生成する	216
3.198. C 言語による開発における packages.txt の書き方	217
3.199. VS Code でコンテナイメージの作成を行う	218

3.200. コンテナイメージの作成完了	218
3.201. コンテナ内のファイル一覧を表示するタブ	219
3.202. コンテナ内のファイル一覧の例	219
3.203. resources ディレクトリ	220
3.204. コンテナ内のファイル一覧を再表示するボタン	221
3.205. container/resources 下にファイルを追加するボタン	222
3.206. ファイル名を入力	222
3.207. 追加されたファイルの表示	223
3.208. container/resources 下にフォルダーを追加するボタン	224
3.209. container/resources 下にあるファイルを開くボタン	225
3.210. container/resources 下にあるファイルを削除するボタン	226
3.211. コンテナ内のファイルを container/resources 下に保存するボタン	227
3.212. 編集前のファイルを示すマーク	228
3.213. 編集後のファイルを示すマーク	229
3.214. コンテナ内にコピーされないことを示すマーク	230
3.215. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする	231
3.216. ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する	232
3.217. ABOSDE に表示されている Armadillo を更新する	233
3.218. ssh_config を編集する	233
3.219. Armadillo 上でアプリケーションを実行する	234
3.220. 実行時に表示されるメッセージ	234
3.221. アプリケーションを終了する	234
3.222. リリース版をビルドする	235
3.223. mkswu バージョン確認コマンド	236
3.224. mkswu のインストール・アップデートコマンド	236
3.225. make_sbom.sh 実行確認コマンド	236
3.226. python3-make-sbom のインストールコマンド	236
3.227. メモリの空き容量の確認方法	237
3.228. 削除されるユーザー設定とユーザーデータを確認	238
3.229. 実際にユーザー設定とユーザーデータを削除する	239
4.1. Armadillo 量産時の概略図	240
4.2. BTO サービスで対応する範囲	242
4.3. 任意のファイルパスを/etc/swupdate_preserve_files に追記する	244
4.4. Armadillo Base OS をアップデートする	244
4.5. パスワードを変更する	244
4.6. make-installer.swu を作成する	246
4.7. 対象製品を選択する	246
4.8. make-installer.swu 生成時のログ	246
4.9. make-installer.swu インストール時のログ	247
4.10. JTAG と SD ブートを無効化する	249
4.11. JTAG と SD ブートの設定値を確認する	250
4.12. JTAG と SD ブートの設定値をリセットする	250
4.13. U-Boot のコマンドプロンプトを無効化する	250
4.14. U-Boot のコマンドプロンプトの設定値を確認する	251
4.15. 開発完了後のシステムをインストールディスクイメージにする	251
4.16. ip_config.txt の内容	253
4.17. IP アドレスの確認	254
4.18. allocated_ips.csv の内容	254
4.19. インストールログを保存する	255
4.20. インストールログの中身	255
4.21. Armadillo に書き込みたいソフトウェアを ATDE に配置	256
4.22. desc ファイルの記述例	256
5.1. 個体番号の取得方法 (device-info)	259

5.2. device-info のインストール方法	259
5.3. 個体番号の取得方法 (get-board-info)	260
5.4. 個体番号の環境変数を conf ファイルに追記	260
5.5. コンテナ上で個体番号を確認する方法	260
5.6. MAC アドレスの確認方法	260
5.7. 出荷時の Ethernet MAC アドレスの確認方法	261
5.8. VS Code を起動	261
5.9. desc ファイルから Armadillo へ SWU イメージをインストールする流れ	263
5.10. コンテナイメージアーカイブ作成例	263
5.11. sample_container_update.desc の内容	264
5.12. sample_container_update.desc の内容	264
5.13. eMMC の予備領域使用率を確認する	265
6.1. persist_file のヘルプ	267
6.2. persist_file 保存・削除手順例	268
6.3. 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	268
6.4. persist_file 変更ファイルの一覧表示例	268
6.5. persist_file でのパッケージインストール手順例	269
6.6. コンテナを作成する実行例	270
6.7. イメージ一覧の表示実行例	271
6.8. podman images --help の実行例	271
6.9. コンテナ一覧の表示実行例	271
6.10. podman ps --help の実行例	272
6.11. コンテナを起動する実行例	272
6.12. コンテナを起動する実行例(a オプション付与)	272
6.13. podman start --help 実行例	273
6.14. コンテナを停止する実行例	273
6.15. podman stop --help 実行例	273
6.16. my_container を保存する例	273
6.17. podman build の実行例	274
6.18. podman build でのアップデートの実行例	274
6.19. コンテナを削除する実行例	275
6.20. イメージを削除する実行例	276
6.21. podman rmi --help 実行例	276
6.22. Read-Only のイメージを削除する実行例	276
6.23. コンテナ内部のシェルを起動する実行例	277
6.24. コンテナ内部のシェルから抜ける実行例	277
6.25. podman exec --help 実行例	277
6.26. コンテナを作成する実行例	278
6.27. コンテナの IP アドレスを確認する実行例	278
6.28. ping コマンドによるコンテナ間の疎通確認実行例	278
6.29. pod を使うコンテナを自動起動するための設定例	279
6.30. network を使うコンテナを自動起動するための設定例	279
6.31. abos-ctrl podman-rw の実行例	281
6.32. abos-ctrl podman-storage のイメージコピー例	282
6.33. Armadillo 上のコンテナイメージを削除する	285
6.34. abos-ctrl container-clear 実行例	286
6.35. コンテナを自動起動するための設定例	286
6.36. ボリュームを shared でサブマウントを共有する例	288
6.37. /proc/devices の内容例	289
6.38. add_armadillo_env で設定した環境変数の確認方法	290
6.39. 上記の例でエラーを発生させた際の起動ログ	293
6.40. インストール用のプロジェクトを作成する	295
6.41. at-debian-image のコンテナイメージをインストールする SWU ファイルを作成する	295

6.42. Docker ファイルによるイメージのビルドの実行例	296
6.43. ビルド済みイメージを load する実行例	296
6.44. alpine のコンテナイメージをインストールする SWU ファイルを作成する	297
6.45. コンテナの IP アドレス確認例	297
6.46. ip コマンドを用いたコンテナの IP アドレス確認例	298
6.47. ユーザ定義のネットワーク作成例	298
6.48. IP アドレス固定のコンテナ作成例	298
6.49. コンテナの IP アドレス確認例	299
6.50. コンテナに Apache をインストールする例	299
6.51. コンテナに lighttpd をインストールする例	300
6.52. コンテナに vsftpd をインストールする例	300
6.53. ユーザを追加する例	300
6.54. 設定ファイルの編集例	301
6.55. vsftpd の起動例	301
6.56. コンテナに samba をインストールする例	301
6.57. ユーザを追加する例	301
6.58. samba の起動例	302
6.59. コンテナに sqlite をインストールする例	302
6.60. sqlite の実行例	302
6.61. X Window System を扱うためのコンテナ起動例	302
6.62. コンテナ内で X Window System を起動する実行例	303
6.63. 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	303
6.64. フレームバッファに直接描画する実行例	304
6.65. タッチパネルを扱うためのコンテナ作成例	304
6.66. パワーマネジメント機能を使うためのコンテナ作成例	304
6.67. サスペンド状態にする実行例	305
6.68. サスペンド状態にする実行例、rtc で起こす	306
6.69. 180 秒後に RTC(NR3225SA)で起床する	306
6.70. コンテナから shutdown を行う	307
6.71. 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	307
6.72. 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	308
6.73. 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	308
6.74. 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	308
6.75. Armadillo Base OS を B 面にコピー	310
6.76. desc ファイルに記述した swudesc_* コマンドを実行	310
6.77. アップデート完了後の挙動	311
6.78. B 面への切り替え	312
6.79. Armadillo Base OS とファイルを B 面にコピー	313
6.80. desc ファイルに記述した swudesc_* コマンドを実行	314
6.81. アップデート完了後の挙動	315
6.82. B 面への切り替え (component=base_os)	315
6.83. mkswu --genkey で署名鍵と証明書を追加する	325
6.84. mkswu --genkey により mkswu.conf に追加された内容	325
6.85. 新しい証明書が Armadillo に追加されていることを確認する	326
6.86. 署名鍵と証明書を削除する設定	326
6.87. 証明書がインストールされていることを確認する	326
6.88. コンテナ管理	328
6.89. SWU インストール	329
6.90. SWU 管理対象ソフトウェアコンポーネントの一覧表示	330
6.91. ネットワークタイムサーバーと同期されている場合の状況確認画面	330
6.92. ネットワークタイムサーバーと同期されていない場合の状況確認画面	331
6.93. ネットワークタイムサーバーの設定項目	331
6.94. タイムゾーンの設定項目	331

6.95. 設定管理の Rest API トークン一覧表示	332
6.96. ユーザ名とパスワード認証の例	347
6.97. 証明書認証の例	347
6.98. ABOS Web を停止する	352
6.99. ABOS Web を起動する	352
6.100. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする	353
6.101. ABOSDE の ABOS Web パスワード入力画面	354
6.102. ABOSDE で Armadillo の SWU バージョンを取得	354
6.103. ABOSDE で Armadillo のコンテナ情報を取得	355
6.104. ABOSDE で Armadillo のコンテナを起動	356
6.105. ABOSDE で Armadillo のコンテナを停止	356
6.106. ABOSDE で Armadillo のコンテナのログを取得	357
6.107. ABOSDE で Armadillo に SWU をインストール	358
6.108. IP アドレスの確認	359
6.109. IP アドレス(eth0)の確認	359
6.110. nmcli のコマンド書式	360
6.111. コネクションの一覧	360
6.112. コネクションの有効化	360
6.113. コネクションの無効化	360
6.114. コネクションの作成	361
6.115. コネクションファイルの永続化	361
6.116. コネクションの削除	361
6.117. コネクションファイル削除時の永続化	362
6.118. 固定 IP アドレス設定	362
6.119. DNS サーバーの指定	362
6.120. DHCP の設定	362
6.121. コネクションの修正の反映	363
6.122. デバイスの一覧	363
6.123. デバイスの接続	363
6.124. デバイスの切断	363
6.125. 有線 LAN の PING 確認	364
6.126. 特定のポートに対する IP アドレスのフィルタリング	364
6.127. 特定のポートに対する IP アドレスのフィルタリングの設定を削除	365
6.128. mount コマンド書式	366
6.129. ストレージのマウント	366
6.130. ストレージのアンマウント	366
6.131. fdisk コマンドによるパーティション変更	367
6.132. EXT4 ファイルシステムの構築	367
6.133. buttdond で SW1 を扱う	368
6.134. buttdond で USB キーボードのイベントを確認する	369
6.135. buttdond で USB キーボードを扱う	369
6.136. buttdond で SW1 を Armadillo 起動時のみ受け付ける設定例	370
6.137. atmark-thermal-profiler をインストールする	371
6.138. atmark-thermal-profiler を実行する	371
6.139. atmark-thermal-profiler を停止する	371
6.140. ログファイルの内容例	371
6.141. 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)	372
6.142. Armadillo-IoT ゲートウェイ G4 で取得した温度のグラフ	373
6.143. abos-ctrl status の例	374
6.144. /var/at-log/atlog の内容の例	375
6.145. local サービスの実行例	376
6.146. uboot_env.d のコンフィグファイルの例	376
6.147. 自動マウントされた microSD カードのアンマウント	379

6.148. デフォルトコンフィギュレーションの適用	382
6.149. Linux カーネルを SWU でインストールする方法	385
6.150. Linux カーネルを build_rootfs でインストールする方法	386
6.151. desc ファイルの追加例	392
6.152. squashfs イメージの作成	392
6.153. mmc-utils のインストール	393
6.154. eMMC の GPP に Temporary Write Protection をかける	393
6.155. 動作ログのフォーマット	395
6.156. minicom の設定の起動	396
6.157. minicom の設定	396
6.158. minicom のシリアルポートの設定	397
6.159. 例. シリアルポート接続時のログ	397
6.160. minicom のシリアルポートのパラメータの設定	398
6.161. minicom シリアルポートの設定値	398
6.162. vi の起動	399
6.163. 入力モードに移行するコマンドの説明	399
6.164. 文字を削除するコマンドの説明	400
6.165. USB シリアル変換アダプタの配線	407
6.166. Armadillo-640 のシリアル信号線	407
6.167. Armadillo-600 シリーズ オプションケース(樹脂製)の組み立て	409
6.168. 樹脂ケース形状図	410
6.169. Armadillo-600 シリーズ オプションケース(金属製)の組み立て	411
6.170. 金属ケース(上板)寸法図	412
6.171. 金属ケース(下板)寸法図	413
6.172. RTC オプションモジュールのブロック図	415
6.173. RTC オプションモジュールのインターフェース	415
6.174. Armadillo-640 CON5 上段と RTC オプションモジュール CON3 は排他利用	417
6.175. RTC オプションモジュールの組み立て	419
6.176. 電池ホルダに電池を取り付ける	419
6.177. 電池ホルダから電池を取り外す	420
6.178. RTC オプションモジュール形状	420
6.179. 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	421
6.180. WLAN コンボ、BT/TH オプションモジュールのブロック図	423
6.181. 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	423
6.182. WLAN コンボ、BT/TH オプションモジュール形状図	425
6.183. WLAN コンボ、BT/TH オプションモジュールの組み立て	426
6.184. 外付けアンテナの組み立て(OP-MNT-ANT-MET-00 使用)	427
6.185. 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)	427
6.186. ケースの組み立て	428
6.187. 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	429
6.188. CON4 をシリアルコンソールとして使用する場合の接続例	429
6.189. CON3 をシリアルコンソールとして使用する場合の接続例	430
6.190. Sterling LWB5+上のコネクタの位置	431
6.191. 挿抜治具によるアンテナケーブルコネクタの嵌合	431
6.192. アンテナケーブルコネクタのセット	432
6.193. アンテナケーブルコネクタの嵌合	432
6.194. 挿抜治具によるアンテナケーブルコネクタの抜去	432
6.195. 無線 LAN 用 外付けアンテナ 08 形状図	433
6.196. 無線 LAN 用 外付けアンテナケーブル 08 形状図	433
6.197. 無線 LAN 用 外付けアンテナ 取り付け穴寸法	433
6.198. 外付けアンテナ固定金具 00 の外観	433
6.199. Armadillo-640 へのアンテナ固定金具の固定	434

6.200. Armadillo-640 へのアンテナ固定金具の固定	435
6.201. アンテナ固定金具へのアンテナの固定	435
6.202. アンテナ固定金具 形状図	436

表目次

1.1. 使用しているフォント	24
1.2. 表示プロンプトと実行環境の関係	24
1.3. コマンド入力例での省略表記	24
1.4. 推奨温湿度環境について	28
1.5. WLAN+BT コンボモジュール: Sterling LWB5+ 適合証明情報	31
1.6. EYSKBNZWB 適合証明情報	31
2.1. Armadillo-640 ラインアップ	38
2.2. 仕様	39
2.3. インターフェース内容	40
2.4. ストレージデバイス	41
2.5. eMMC の GPP の用途	42
2.6. eMMC メモリマップ	42
2.7. eMMC ブートパーティション構成	42
2.8. eMMC GPP 構成	42
3.1. ユーザー名とパスワード	46
3.2. 動作確認に使用する取り外し可能デバイス	77
3.3. 電源を切っても保持されるディレクトリ(ユーザーデータディレクトリ)	94
3.4. CON8 信号配列	100
3.5. CON9 信号配列	100
3.6. CON14 信号配列	101
3.7. 絶対最大定格	101
3.8. 推奨動作条件	102
3.9. 入出カインターフェース(電源)の電氣的仕様	102
3.10. 入出カインターフェースの電氣的仕様(OVDD = VCC_3.3V)	102
3.11. Armadillo-640 インターフェース一覧	118
3.12. CON1 信号配列	120
3.13. CON2 信号配列	123
3.14. CON7 信号配列	124
3.15. LAN LED の動作	124
3.16. CON3 信号配列	126
3.17. CON4 信号配列	126
3.18. CON5 信号配列	128
3.19. CON9 ピンと GPIO の対応	141
3.20. I2C デバイス	142
3.21. CON10 信号配列	146
3.22. CON11 信号配列	147
3.23. CON13 信号配列	151
3.24. ジャンパの設定と起動デバイス	153
3.25. JP1 信号配列	153
3.26. JP2 信号配列	154
3.27. SW1 信号配列	154
3.28. キーコード	154
3.29. LED3、LED4、LED5	155
3.30. LED トリガの種類	156
3.31. Armadillo Base OS のデフォルトで開放しているポート	163
3.32. 用意する favicon 画像	186
3.33. ABOSDE の対応言語	189
4.1. インストールディスクと SWUpdate によるソフトウェア書き込みの比較	243
4.2. インストール中に実行される関数	252
5.1. EXT_CSD_PRE_EOL_INFO の値の意味	265

6.1. add_hotplugs オプションに指定できる主要な文字列	289
6.2. add_armadillo_env で追加される環境変数	290
6.3. 対応するパワーマネジメント状態	305
6.4. swudesc_* コマンドの種類	311
6.5. アップデート完了後の挙動の種類	312
6.6. swudesc_* コマンドの種類	314
6.7. アップデート完了後の挙動の種類	315
6.8. ネットワークとネットワークデバイス	359
6.9. 固定 IP アドレス設定例	362
6.10. thermal_profile.csv の各列の説明	372
6.11. rollback-status の出力と意味	375
6.12. rollback-status 追加情報の出力と意味	375
6.13. u-boot の主要な環境変数	377
6.14. microSD カードのパーティション構成	380
6.15. build-rootfs のファイル説明	387
6.16. desc ファイルの設定項目	392
6.17. /var/log/ 配下のログ	395
6.18. シリアル通信設定	396
6.19. 入力モードに移行するコマンド	399
6.20. カーソルの移動コマンド	400
6.21. 文字の削除コマンド	400
6.22. 保存・終了コマンド	401
6.23. GPIO override と eFuse	402
6.24. ブートデバイスと eFuse	403
6.25. オンボード eMMC のスペック	404
6.26. Armadillo-640 関連のオプション品	406
6.27. 各ピンに対応する UART コントローラ	407
6.28. 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	408
6.29. Armadillo-600 シリーズ オプションケース(樹脂製)について	408
6.30. Armadillo-600 シリーズ オプションケース(樹脂製)の仕様	408
6.31. Armadillo-600 シリーズ オプションケース(金属製)について	411
6.32. 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	413
6.33. LCD の仕様	413
6.34. Armadillo-600 シリーズ RTC オプションモジュールについて	414
6.35. RTC オプションモジュールの仕様	414
6.36. RTC オプションモジュール インターフェース一覧	416
6.37. CON1 信号配列	416
6.38. CON3 信号配列	417
6.39. 対応バッテリー例	417
6.40. CON4、CON5、CON6 信号配列	417
6.41. 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	421
6.42. WLAN コンボ、BT/TH オプションモジュールの同梱物	421
6.43. 推奨外付けアンテナ	421
6.44. WLAN コンボ、BT/TH オプションモジュールの仕様	422
6.45. 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	424
6.46. CON1 信号配列	424
6.47. CON2 信号配列	425
6.48. CON2 搭載コネクタ例	425
6.49. 無線 LAN 用 外付けアンテナセット 08 について	430
6.50. 外付けアンテナ固定金具 00 について	434
6.51. 外付けアンテナ固定金具 00 の仕様	434

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二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. 本書について

1.1.1. 本書で扱うこと

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. 本書で扱わないこと

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。

1.1.3. 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 は組み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。

1.1.4. 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。

本書の章構成は「図 1.1. 製品化までのロードマップ」に示す流れを想定したものとなっています。

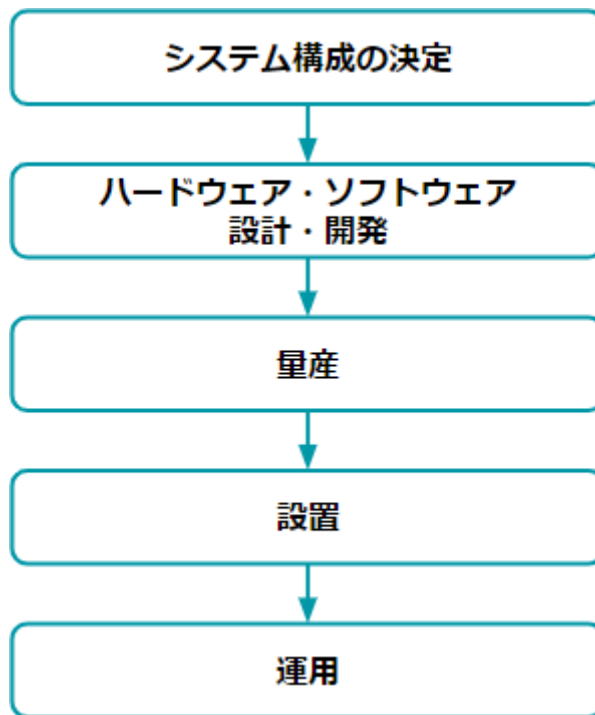


図 1.1 製品化までのロードマップ

- ・「システム構成の決定」、「ハードウェア・ソフトウェア設計・開発」

システムが必要とする要件から使用するクラウド、デバイス、ソフトウェア仕様を決定および、ハードウェア・ソフトウェアの開発時に必要な情報について、「3. 開発編」で紹介します。

- ・「量産」

開発完了後の製品を量産する方法について、「4. 量産編」で紹介します。

・「設置」、「運用」

設置時の勘所や、量産した Armadillo を含めたハードウェアを設置し、運用する際に利用できる情報について、「5. 運用編」で紹介します。

また、本書についての概要を「1. はじめに」に、Armadillo-640 についての概要を「2. 製品概要」に、開発～運用までの一連の流れの中で説明しきれなかった機能についてを、「6. 応用編」で紹介します。

1.1.5. フォント

本書では以下のような意味でフォントを使っています。

表 1.1 使用しているフォント

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

1.1.6. 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

1.1.7. 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。

1.1.8. 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、「3.1.8. ユーザー登録」を参照してください。

1.1.9. 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>

1.2. 注意事項

1.2.1. 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。

1.2.2. 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 ^[1] 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 ^[2] 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) ^[3] 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、静電気を運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。
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、電池の性能が十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。

2.4DS4/OF4

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。

^[1]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。

^[2]改造やコネクタを増設する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。

^[3]別途、活線挿抜を禁止している場合を除く

2.4FH4/XX8

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。

IEEE802.11a/n/ac

~~W52~~ | **W52** | **W53** | **W56**

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。

1.2.3. 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 1.4 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 ^[a] ^[b]
---------	---

^[a]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。

^[b]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。

1.2.4. 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)

パスワードの設定について

本製品にはシリアルコンソールや SSH (標準ソフトウェアでは無効)、Web UI からパスワードを使用してログインできる機構が備わっています。

基本的には初回ログイン時または、専用の開発環境での開発開始時に機器のパスワードの再設定を求められます。この時、推測されやすい単純なパスワードを使用すると外部からの不正アクセスや機器の乗っ取りの原因になりますので、必ず推測されにくい複雑なパスワードを設定してください。パスワードの設定については、「3.1.7.3. ログイン」や「3.1.5. Armadillo に初期設定をインストールする」、「3.8.3. ABOS Web のパスワード登録」を参照してください。

ソフトウェアのアップデートについて

弊社は Armadillo のソフトウェアのアップデートを定期的実施します。詳細は「2.1.4. Armadillo Base OS のメンテナンスポリシーとアップデートの推奨」を参照してください。

本製品のソフトウェアは常に最新版にアップデートした上でご使用ください。本製品を購入後、開発前に最新版のソフトウェアをインストールする方法については「3.1.4. Armadillo の初期化と ABOS のアップデート」を、運用時に本製品をアップデートする方法については「5.4. 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。

お客様がアップデートを適用せずに運用した場合、セキュリティインシデント、利用できる機能の制限、サポートの制限が発生することがあります。

1.2.5. 本製品を廃棄する場合について

本製品を廃棄する場合は、必ず「5.8. Armadillo を廃棄する」に記載の注意事項をご一読の上、機器内に保存されているログや個人情報などのデータを正しく全て削除した上で廃棄してください。

情報資産が機器内に残留したまま廃棄することは、個人情報や機密情報が第三者に漏洩する可能性があります。また、法的な問題や責任を引き起こす可能性があります。データの削除手順について不明点がある場合は、弊社までお問い合わせください。

1.2.6. 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

1.2.7. 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。

また、製品を安心して長い期間ご利用いただくために、保証期間を 2 年または 3 年間に延長できる「延長保証サービス」をオプションで提供しています。詳細は「製品保証サービス」を参照ください。

Armadillo サイト - 製品保証サービス

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

Armadillo サイト - 製品保証規定

<https://armadillo.atmark-techno.com/support/warranty/policy>

1.2.8. 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、必要な手続き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。

1.2.9. 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。



1.2.10. 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 1.5 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

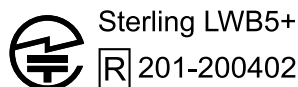


図 1.2 WLAN+BT コンボモジュール: Sterling LWB5+ 認証マーク

表 1.6 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

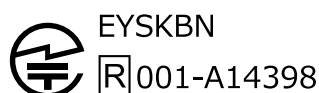


図 1.3 EYSKBNZWB 認証マーク

1.3. 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. 製品概要

2.1. 製品の特長

2.1.1. 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。

2.1.2. 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。

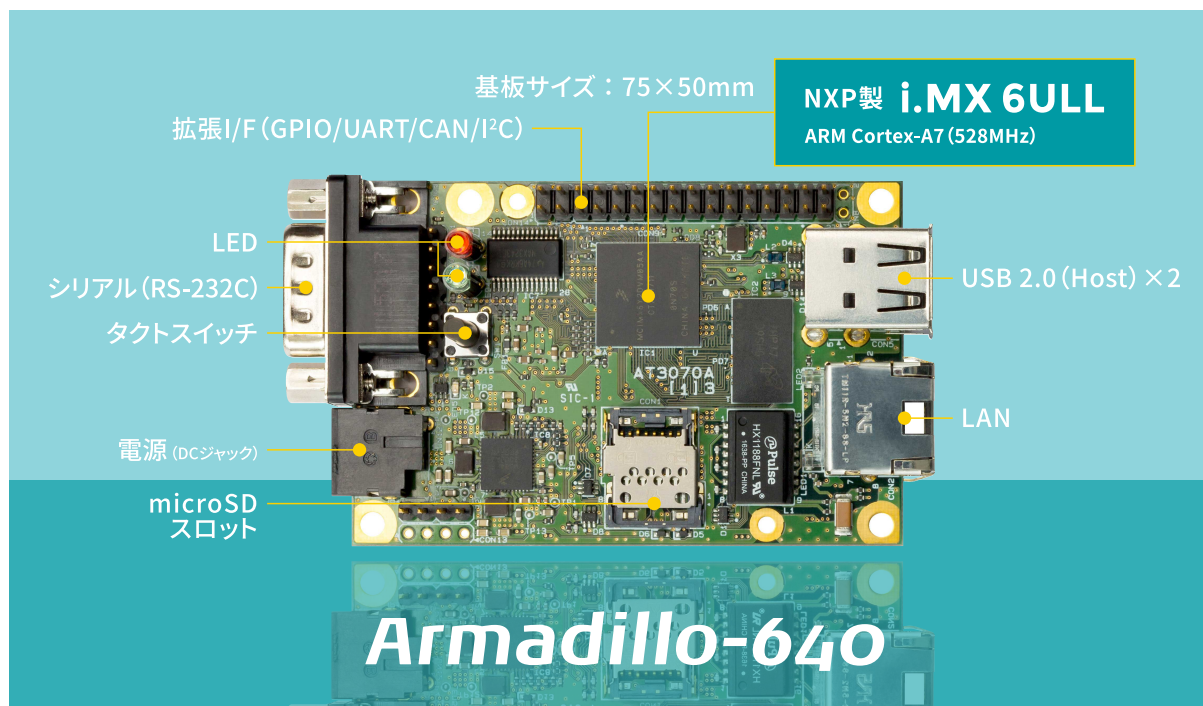


図 2.1 Armadillo-640 とは

- ・ 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。

- ・セキュアエレメント搭載

NXP Semiconductors 製のセキュアエレメント「SE050」を標準搭載しています。これを使用することで、ハードウェア Root of Trust による高いセキュリティを実現できます。

2.1.3. 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。

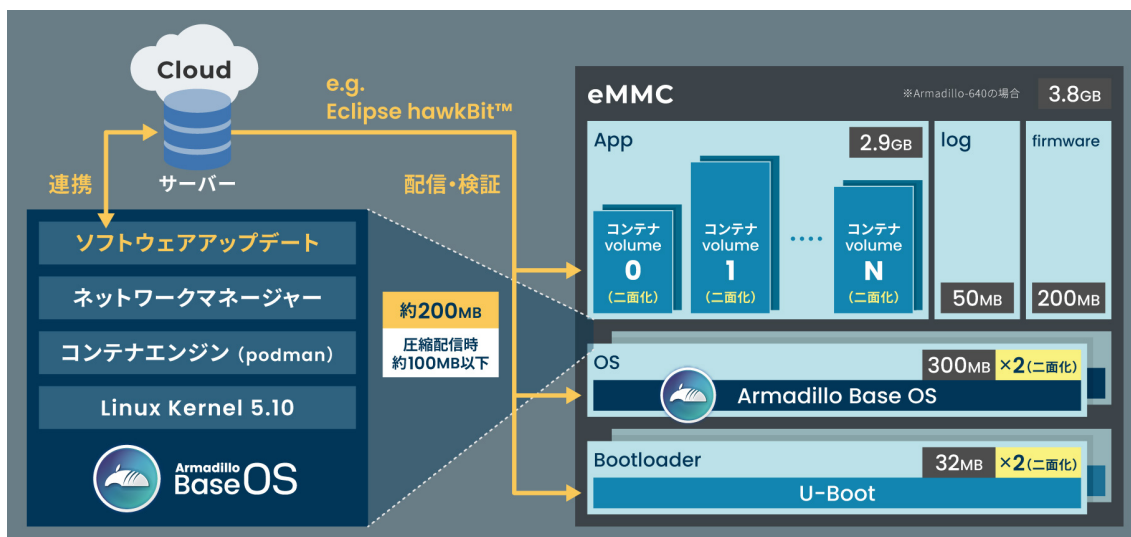


図 2.2 Armadillo Base OS とは

- ・ 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。

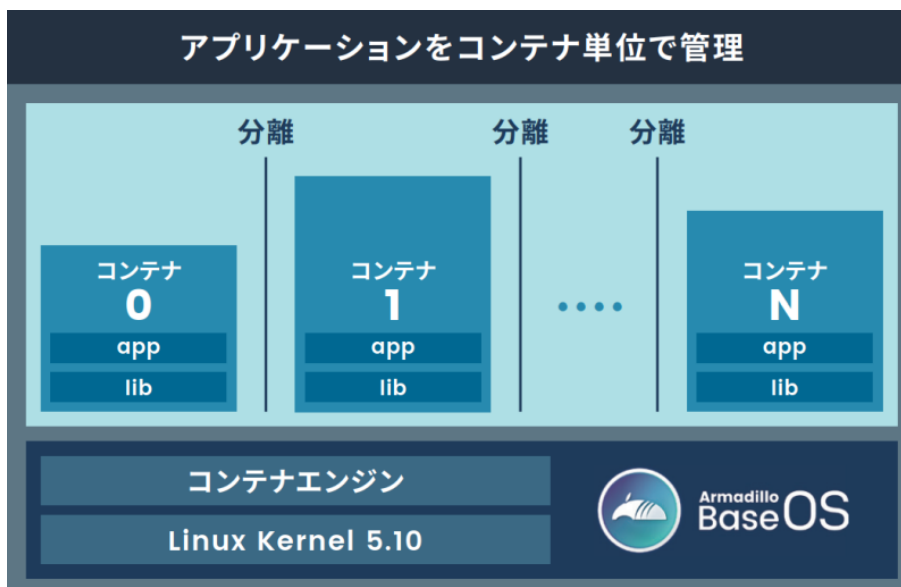


図 2.3 コンテナによるアプリケーションの運用

- ・ 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、Armadillo Twin 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。

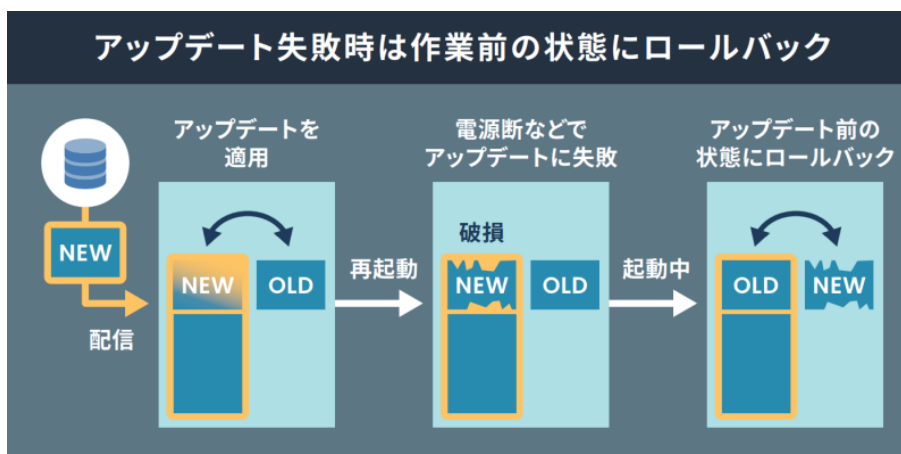


図 2.4 ロールバックの仕組み

- ・ 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耗を抑制するなど、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。デバイス証明に利用できるセキュアエレメントを搭載するほか、セキュア環境「OP-TEE」を利用可能な状態で提供しています。

2.1.4. Armadillo Base OS のメンテナンスポリシーとアップデートの推奨

Armadillo Base OS は Armadillo Base OS 搭載製品のサポート期限 [https://armadillo.atmark-techno.com/abos-support-timeline]にしたがって、アットマークテクノがセキュリティアップデートの提供、既存機能のバグ修正、今はない便利な機能の追加を継続的に行い、ユーザービリティの向上に努めます。緊急時を除き月末に "製品アップデート" としてこれらをリリースをし、Armadillo サイトから通知、変更内容の公開を行います。ユーザー登録を行うことで通知をメールで受け取ることもできます。

Armadillo を IoT 機器としてネットワークに接続し長期に運用を行う場合、継続的に最新バージョンを使用することを強く推奨いたします。最新バージョンを使用しない場合の注意点については「1.2.4. ソフトウェア使用に関する注意事項」の「ソフトウェアのアップデートについて」を参照してください。

Armadillo サイト 製品アップデート

<https://armadillo.atmark-techno.com/news/software-updates>

2.1.4.1. 後方互換性について

Armadillo Base OS は、原則、abos-ctrl コマンド等の各種機能や、sysfs ノード、コンテナ制御をするための podman コマンド等の API 後方互換を維持します。また、Armadillo Base OS とコンテナ間でサンドボックス化されていることもあり、互いの libc 等のライブラリや、各種パッケージなどの組み合わせによって互換性の問題は発生しません。このため、Armadillo Base OS をアップデートしても、これまで利用していたアプリケーションコンテナは原則的にそのまま起動・動作させることができます。

しかし、Armadillo Base OS 内の Linux-Kernel や alpine パッケージ変更によって、細かな動作タイミングが変更になる場合があるため、タイミングに大きく依存するようなアプリケーションをコンテナ内部に組み込んでいた場合に、動作に影響を与える可能性があります。まずは、テスト環境で Armadillo Base OS 更新を行い、アプリケーションコンテナと組み合わせた評価を行った後、市場で動作している Armadillo に対してアップデートを行うことを推奨します。

製品開発を開始するにあたり、Armadillo Base OS に関してより詳細な情報が必要な場合は、「3.3. 開発前に知っておくべき Armadillo Base OS の機能・特徴」を参照してください。

2.1.5. Armadillo Twin とは

Armadillo Twin は、アットマークテクノが提供する Armadillo Base OS 搭載のデバイスをリモートから運用管理することができるクラウドサービスです。様々なタスクをリモートから実行できるようになり、OS アップデートもサービス画面からの操作で行えるため、稼働中のデバイスは常に最新の状態を維持することができます。また、バグ修正やセキュリティ対策などのメンテナンスのほか、機能追加や設定変更、アプリケーションのアップデートなども行えるため、デバイスの設置現場に出向くことなく、計画的で効率的な DevOps を実現することができます。

本書では、開発・量産・運用の各フェーズにおける Armadillo Twin の利用について記載しています。



図 2.5 Armadillo Twin とは

2.1.5.1. サービスの特徴

- ・ソフトウェアアップデート (OTA)

遠隔からデバイスのソフトウェアアップデートをすることで、長期的にセキュリティ性の高いシステムを保つと共に、新たな機能を提供することも可能です。本サービスで管理するデバイスに搭載されている Armadillo Base OS は、不正なソフトウェアへのアップデートを行わせない署名検証機能や、アップデートが失敗した際に自動で元の状態に戻るロールバック機能を備えています。そのため、安心してソフトウェアアップデートを利用することができます。

- ・遠隔稼働監視

登録されたデバイスの死活監視をはじめ、CPU の使用率や温度、メモリの使用量、モバイル回線の電波状況、ストレージの空き容量や寿命を監視することができます。各値にはアラートの設定を行うことができ、異常を検知した場合はアラートメールを管理者に送信します。メールを受けた管理者は本サービスの遠隔操作機能を利用し、即座に対応を行うことができるため、システムの安定運用を行うことができます。そのほか、本サービスに登録したデバイスは、自由にラベル名を付けたりグループを作成して管理することができるため、どのデバイスをどの場所に設置したか画面上で把握することが容易になります。また、デバイス本体に搭載されているセキュアエレメントを利用した個体認証により、不正なデバイスの登録を防ぎます。

- ・遠隔操作

画面上で入力した任意のコマンドをデバイス上で実行することができます。本サービスは遠隔操作で一般的に使われる SSH(Secure Shell) のように固定グローバル IP アドレスの設定は不要です。そのため、通信回線の契約料金を安くできるだけではなく、インターネット上からのサイバー攻撃のリスクを抑制する効果も期待できます。任意のコマンドは単一のデバイスだけではなく、グループ単位、また複数のデバイスを選択して一括して実行したり、時刻を指定するスケジュール実行にも対応しています。

2.1.5.2. 提供する機能一覧

Armadillo Twin は、下記の機能を提供します。

遠隔稼働監視	死活監視、アプリケーションコンテナ稼働状況、CPU 使用率・温度/メモリ使用率、ストレージ寿命、モバイル回線電波強度、モバイル回線基地局の位置情報 ^[a] 、アラートメール
遠隔操作	ソフトウェアアップデート(OTA)、任意コマンド実行、ソフトウェアバージョン確認、設定変更、グループ一括実行、スケジュール実行 ^[a]
個体管理	デバイス登録(デバイス証明書を利用)、ラベル付け、デバイスグループ化機能
ユーザ管理	ユーザーの追加/削除、ユーザー権限の設定
お知らせ	セキュリティアップデート、システム障害通知

^[a]サービス開始時には非対応の機能です。今後のアップデートで対応予定です。

Armadillo Twin

サービス利用料金など、その他詳細については下記概要ページをご覧ください。

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



Armadillo-640 は シリアルナンバー：009C01490001 以降の個体で Armadillo Twin を利用することができます。

2.2. 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 2.1 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

2.2.1. 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 ^[1]
- ・ 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

^[1]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。

2.2.2. 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。

2.3. 仕様

Armadillo-640 の主な仕様は次のとおりです。

表 2.2 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 [a]
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 [a] 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 [a] S/PDIF 最大 1 ポート拡張可能 [a]
GPIO	最大 61 bit 拡張可能 [a]
I2C	最大 3 ポート拡張可能 [a]
SPI	最大 4 ポート拡張可能 [a]
CAN	最大 2 ポート拡張可能 [a]
PWM	最大 8 ポート拡張可能 [a]
カレンダー時計	SoC 内蔵リアルタイムクロック [b] I2C 拡張可能
スイッチ	ユーザースイッチ x 1
LED	ユーザー LED x 3
セキュアエレメント	NXP Semiconductors SE050 [c]
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) [d]
使用温度範囲	-20~+70°C(結露なきこと) [e]
外形サイズ	75x50mm(突起部を除く)

[a]i.MX6ULL のピンマルチプレクスの設定で、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。

[b]電池は付属していません。

[c]シリアルナンバー：009C01490001 以降の Armadillo-640 に搭載

[d]外部接続機器の消費分は含みません。

[e]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。

2.4. 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。

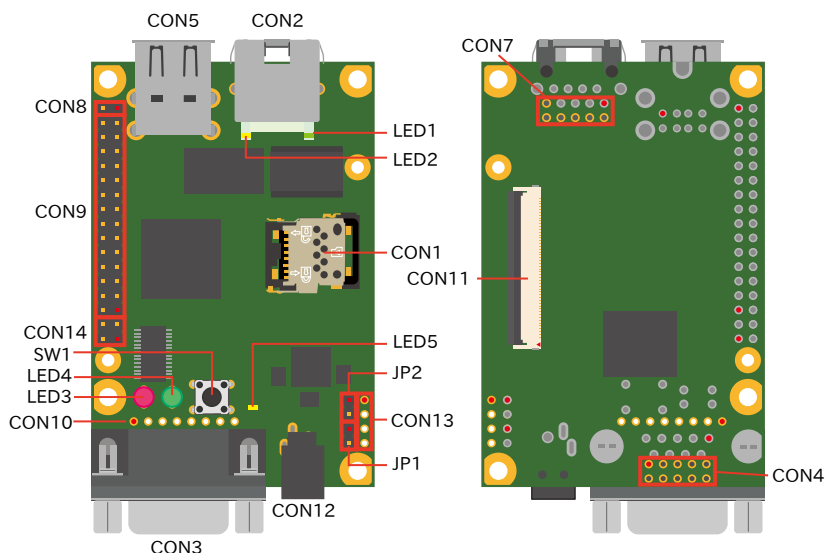


図 2.6 インターフェースレイアウト

表 2.3 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 ^[a]
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)	

^[a]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。

2.5. ブロック図

Armadillo-640 のブロック図は次のとおりです。

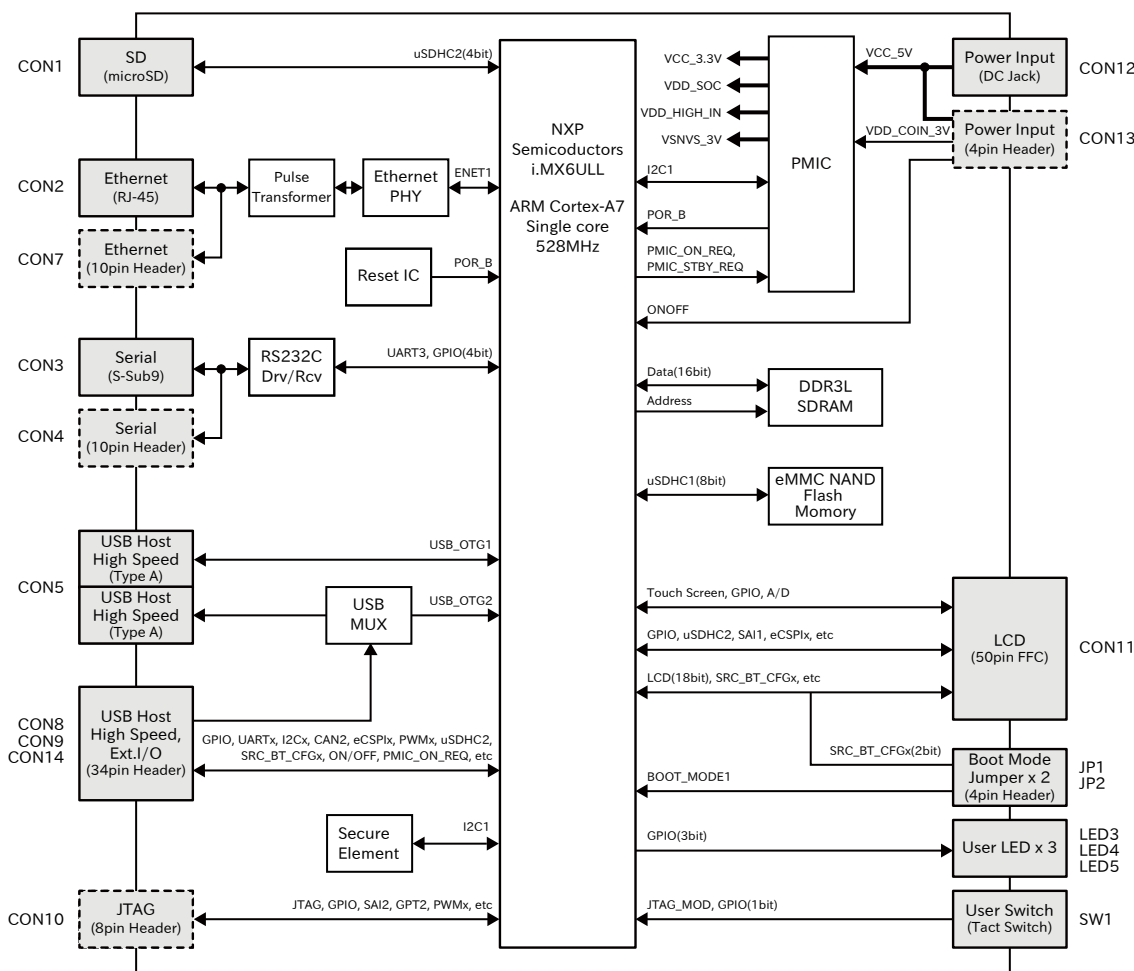


図 2.7 Armadillo-640 ブロック図

2.6. 使用可能なストレージデバイス

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 2.4 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* ^[a]	/dev/sd*1	USB ホストインターフェース (CON5)

^[a]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 2.5. eMMC の GPP の用途」に示します。

表 2.5 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 ^[a]
/dev/mmcblk0gp3	ユーザー領域

^[a]詳細は「6.24.4. ログ用パーティションについて」を参照ください。

2.7. 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 2.6. eMMC メモリマップ」に示します。

表 2.6 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 2.7. eMMC ブートパーティション構成」に示します。

表 2.7 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 2.8. eMMC GPP 構成」に示します。

表 2.8 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 ^[a]
/dev/mmcblk0gp3	8 MiB	ユーザー領域

^[a]詳細は「6.24.4. ログ用パーティションについて」を参照ください。

2.8. ソフトウェアのライセンス

Armadillo Base OS に含まれるソフトウェアのライセンスは、Armadillo にログイン後に特定のコマンドを実行することで参照できます。

手順について、詳細は以下の Howto を参照してください。

Armadillo サイト - Howto インストール済みのパッケージのライセンスを確認する

https://armadillo.atmark-techno.com/howto_software-license-confirmation

3. 開発編

Armadillo-640 では基本的に ATDE という Armadillo 専用開発環境と、Visual Studio Code (以降 VS Code と記載します) 向け Armadillo 開発用エクステンションを用いてアプリケーション開発を行っています。

3.1. 開発の準備

この節では、アプリケーション開発のために、はじめに開発環境のセットアップを行います。本節を完了すると、Armadillo を用いた製品の開発に即座に取り組むことができる状態になります。

開発環境のセットアップは、作業用 PC と Armadillo の両方に対して行います。本節では初めに作業用 PC についてのセットアップを行い、その後に Armadillo についてのセットアップを行います。そのため、新たに Armadillo を用意した場合や、Armadillo のセットアップをやり直したい方は本節の途中から行うことができます。後半では Armadillo による開発方法の勝手を大まかに把握したい方を想定して、Python アプリケーションによる LED 点滅の動作確認を行う項を用意しています。不要な方はこの項をスキップしてください。その後、Armadillo のシリアルコンソールのセットアップ・操作方法について解説します。

3.1.1. 準備するもの

開発環境をセットアップする上で、まずは次のものを用意してください。

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。
Armadillo-640 開発セット一式	詳しくは Armadillo-640 ベーシックモデル開発セットのセット内容をご参照ください。
1 GB 以上の microSD カード	Armadillo の初期化・ABOS のアップデートの際に使用します。
プラスドライバー	ドライバーはケースを取り外す際に使用しますので、ケースが無い場合は不要です。
ネットワーク環境	仮想化ソフトウェアや Armadillo の初期化インストールディスクイメージなどを作業用 PC にダウンロードする手順があります。また、「3.1.5. Armadillo に初期設定をインストールする」の手順から Armadillo と作業用 PC をネットワーク通信ができるようにする必要があります。

3.1.2. 仮想環境のセットアップ

作業用 PC をセットアップします。アットマークテクノでは、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、Oracle VM VirtualBox 仮想マシンのデータイメージを提供しています。こ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 VirtualBox を使用します。



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 ^[1]
- ・ VMware 形式の仮想ディスク(.vmdk)ファイルに対応している

3.1.2.1. VirtualBox のインストール

ATDE を使用するために、作業用 PC に VirtualBox をインストールします。VirtualBox の Web ページ(<https://www.virtualbox.org/>) を参照してインストールしてください。

また、ホスト OS が Linux の場合、デフォルトでは VirtualBox で USB デバイスを使用することができません。ホスト OS (Linux) で以下のコマンドを実行後、ホスト OS を再起動してください。

```
[PC ~]$ sudo usermod -a -G vboxusers $USER
```

ホスト OS が Windows の場合はこの操作は必要ありません。

3.1.2.2. ATDE のアーカイブを取得

ATDE のアーカイブ(.ova ファイル)を Armadillo サイト(<https://armadillo.atmark-techno.com/resources/software/atde/atde-v9>)からダウンロードします。



アットマークテクノ製品の種類ごとに対応している ATDE のバージョンが異なります。本製品に対応している ATDE のバージョンは以下のとおりです。

VirtualBox	ATDE9 v20240925 以降
VMware	ATDE9 v20230328 から ATDE9 v20240826

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。



作業用 PC の動作環境(ハードウェア、VirtualBox、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。VirtualBox の Web ページ(<https://www.virtualbox.org/>) から、使用し

[1]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。

ている VirtualBox 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。

3.1.2.3. ATDE のインポート

1. VirtualBox を起動し、[ファイル]-[仮想アプライアンスのインポート]を選択します。
2. [ソース]の項目で、ダウンロードした ATDE のアーカイブ(.ova ファイル)を選択します。
3. [設定]の項目で、[ハードドライブを VDI としてインポート]のチェックを外します。
4. [完了]をクリックします。インポートの処理が完了するまで数分程要します。
5. インポートの処理が完了したら、ホスト OS の環境に合わせた設定に更新するため仮想マシンを選択して[設定]をクリックした後に[OK]をクリックします。



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、VirtualBox の Web ページ (<https://www.virtualbox.org/>) から、VirtualBox のドキュメントなどを参照してください。

3.1.2.4. ATDE の起動

1. 仮想マシンを選択して[起動]をクリックしてください。
2. ATDE のログイン画面が表示されます。

ATDE にログイン可能なユーザーを、「表 3.1. ユーザー名とパスワード」に示します [2]。

表 3.1 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

3.1.2.5. コマンドライン端末(GNOME 端末)の起動

Armadillo を利用した開発では、CUI (Character-based User Interface)環境を提供するコマンドライン端末を通じて、Armadillo や ATDE に対して操作を行う場面が多々あります。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 3.1. 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。

[2]特権ユーザーで GUI ログインを行うことはできません



図 3.1 GNOME 端末の起動

「図 3.2. GNOME 端末のウィンドウ」のようにウィンドウが開きます。

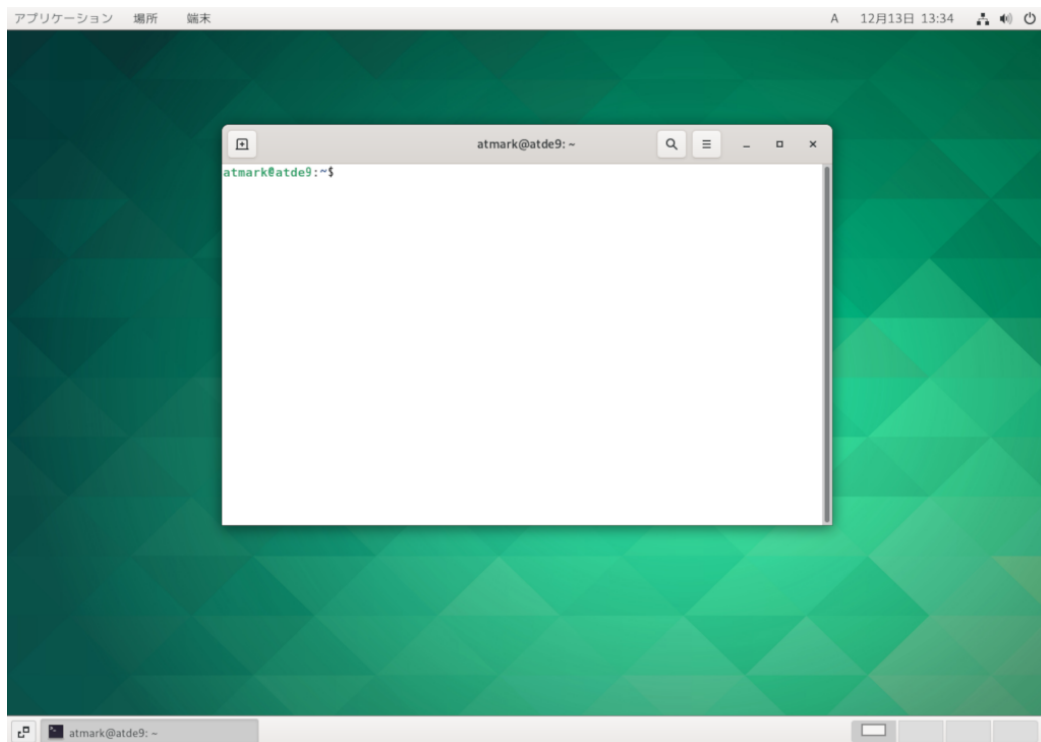


図 3.2 GNOME 端末のウィンドウ

3.1.2.6. ソフトウェアのアップデート

コマンドライン端末から次の操作を行い、ソフトウェアを最新版へアップデートしてください。

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

図 3.3 ソフトウェアをアップデートする

3.1.2.7. 取り外し可能デバイスの使用

VirtualBox 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 (VirtualBox を起動している OS)と ATDE 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 ATDE で使用するためには、ATDE にデバイスを接続する「図 3.4. ATDE にデバイスを接続する」の操作が必要になります。

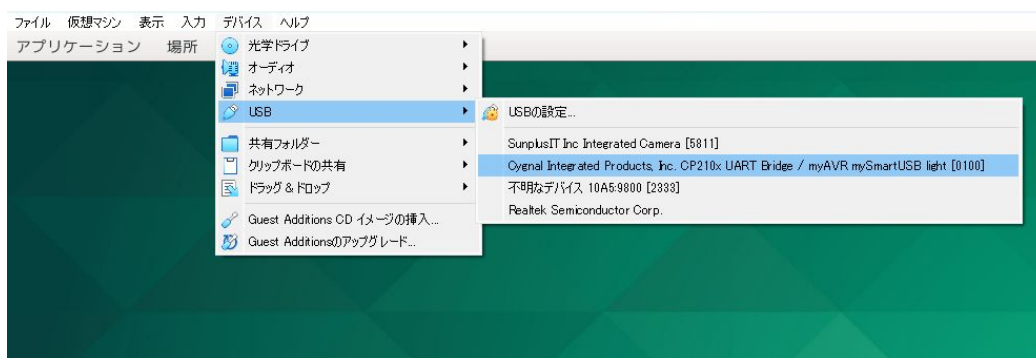


図 3.4 ATDE にデバイスを接続する

3.1.2.8. VirtualBox Guest Additions の再インストール

ATDE は VirtualBox 仮想マシン用ソフトである VirtualBox Guest Additions があらかじめインストールされた状態で配布されています。

Guest Additions のバージョンは VirtualBox 自体のバージョンと連動しているため、お使いの VirtualBox のバージョンと ATDE にインストール済みの Guest Additions のバージョンが異なる場合があります。

VirtualBox と Guest Additions のバージョンが異なることによって問題が起こる可能性は低いですが、これに起因すると思われる不具合 (ATDE の画面・共有フォルダー・クリップボード等の不調) が発生した場合は、以下の手順を参考に Guest Additions を再インストールしてください。(実行前に ATDE のスナップショットを作成しておくことを推奨します)

1. ATDE を起動後、上部バーの[ツール]-[Guest Additions CD イメージの挿入]を選択してください。
2. お使いの VirtualBox と同じバージョンの VBox_GAs_[VERSION] が「ファイル」上に表示されます。
3. VBox_GAs_[VERSION] をマウントするために、「ファイル」から VBox_GAs_[VERSION] を押下してください。
4. インストールする前に、以下のコマンドで既にインストール済みの Guest Additions をアンインストールします。


```
sudo /opt/VBoxGuestAdditions-[VERSION]/uninstall.sh
```

5. 以下のコマンドでお使いの VirtualBox のバージョンに合った Guest Additions がインストールされます。

```
cd /media/cdrom0  
sudo sh ./autorun.sh
```

3.1.2.9. 共有フォルダーの作成

ホスト OS と ATDE 間でファイルを受け渡す手段として、共有フォルダーがあると大変便利です。ここでは、ホスト OS と ATDE 間の共有フォルダーを作成する手順を紹介しますが、不要な方はこの手順をスキップしてください。

1. VirtualBox の上部バーから[デバイス]-[共有フォルダー]-[共有フォルダー設定]を選択します。（「図 3.5. 共有フォルダー設定を開く」）
2. 「図 3.6. 共有フォルダー設定」 の赤枠で示したアイコンをクリックします。
3. 「図 3.7. 共有フォルダーの追加」 のように、[フォルダーのパス]-[その他]を選択して、共有フォルダーに設定したいホスト OS 上のフォルダーを選択します。
4. 「図 3.7. 共有フォルダーの追加」 のように、[自動マウント]と[永続化する]にチェックを入れます。
5. [OK]をクリックして共有フォルダーを追加します。

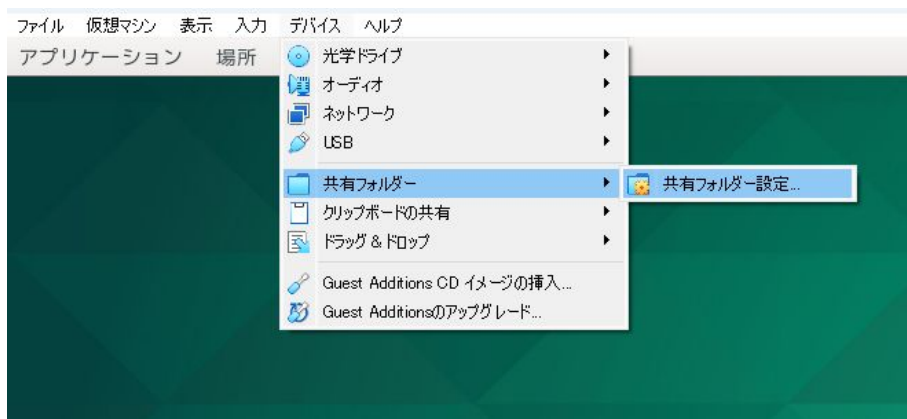


図 3.5 共有フォルダー設定を開く

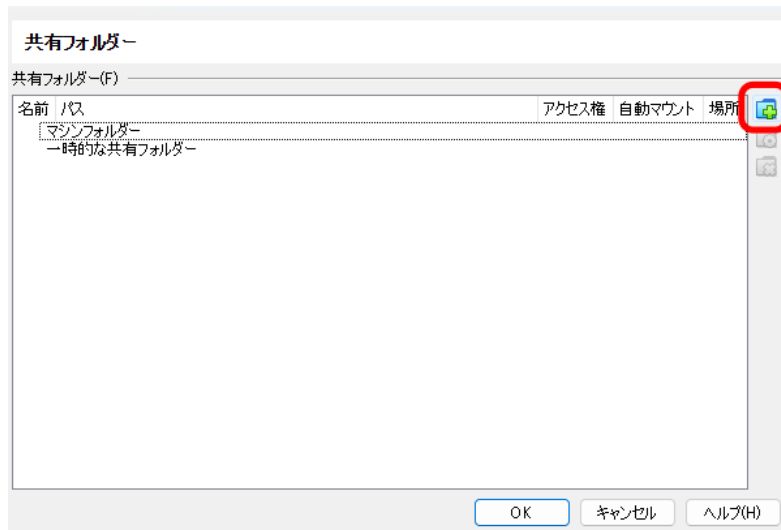


図 3.6 共有フォルダー設定

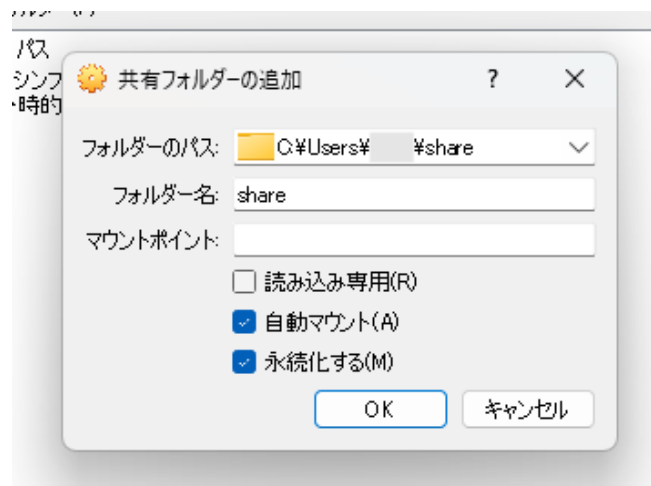


図 3.7 共有フォルダーの追加



図 3.8 「ファイル」に表示される共有フォルダー

追加した共有フォルダーは、「図 3.8. 「ファイル」に表示される共有フォルダー」のように「ファイル」からアクセスするか、または /media/sf_share (共有フォルダー名) からアクセスできます。(share というフォルダー名で作成すると、ATDE 上では sf_share として表示されます。)

3.1.3. VS Code のセットアップ

作業用 PC のセットアップです。Armadillo-640 の開発には、VS Code を使用します。ATDE のバージョン v20230123 以上には、VS Code 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「図 3.3. ソフトウェアをアップデートする」にしたがって最新版へのアップデートを行ってください。

以下の手順は全て ATDE 上で実施します。

3.1.3.1. VS Code を起動する

VS Code を起動するために code コマンドを実行するか、「アプリケーション」の中から「Visual Studio Code」を探して起動してください。

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

図 3.9 VS Code を起動する



VS Code 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 VS Code を日本語化できます。

3.1.3.2. VS Code に開発用エクステンションをインストールする

VS Code 上でアプリケーションを開発するために、ABOSDE (Armadillo Base OS Development Environment) という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。VS Code 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。

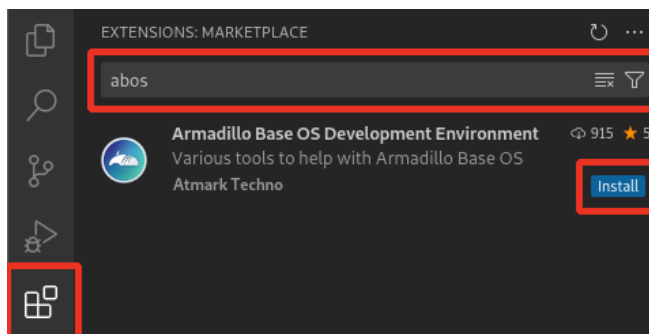


図 3.10 VS Code に開発用エクステンションをインストールする

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。

3.1.4. Armadillo の初期化と ABOS のアップデート

Armadillo をセットアップします。まずは、お手元の Armadillo に搭載されている Armadillo Base OS (ABOS) を最新版にします。ABOS のバージョンが古い場合、本マニュアルで紹介されている重要な機能を使用できない可能性があります。そのため、以下の手順に従って、ABOS のアップデートを兼ねた Armadillo の初期化を行ってください。

3.1.4.1. 初期化インストールディスクの作成

- 1 GB 以上の microSD カードを用意してください。
- 標準のインストールディスクイメージをダウンロードします。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 インストールディスクイメージ」を ATDE にダウンロードしてください。



標準のインストールディスクイメージは、シリアルコンソールの出力に CON9 を使用します。オプションモジュール等を使用しシリアルコンソールの出力を CON3 へ変更したい場合は、「Armadillo Base OS インストールディスクイメージ(UART3 コンソール)」を使用するか、`build_image.sh` 実行時のオプションに `--uboot-env console=ttymxc2,115200` を追加してください。

ABOS 3.19.1-at.3 以降のバージョンで `abos-ctrl make-installer` を実行してインストールディスクイメージを作成した場合は、現在の設定を自動的に取り込みます。

インストールディスクの出力だけではなく、インストールされた後にも適用されます。

- ATDE に microSD カードを接続します。詳しくは「3.1.2.7. 取り外し可能デバイスの使用」を参考にしてください。
- 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。

3.1.4.2. インストールディスクを使用する

以下の手順に沿って、「図 3.11. Armadillo-640 を初期化する接続」のとおり接続します。

- Armadillo に AC アダプタが接続されている場合は、これを取り外します。
- JP1 と JP2 をジャンパーソケットでショート (SD ブートに設定) させ、microSD カードを CON1 に挿入します。(方法は「3.6.1.2. microSD カードの挿抜方法」を参考にしてください)
- AC アダプタを接続して電源を投入すると、数秒経過後に LED が点灯します。
- 1 分程度で eMMC のソフトウェアの初期化が完了し、電源が切れます (LED が消灯)。
- LED が消灯したら、AC アダプタを取り外し、続いて JP1 と JP2 のジャンパーソケットと microSD カードを外してください。

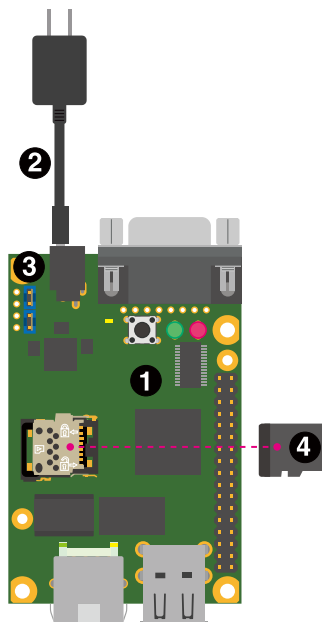


図 3.11 Armadillo-640 を初期化する接続

- 1 Armadillo-640
- 2 AC アダプタ (5V/2.0A)
- 3 ジャンパーソケット

4 microSD カード



ジャンパの設定について

この手順ではジャンパの設定が必要になります。

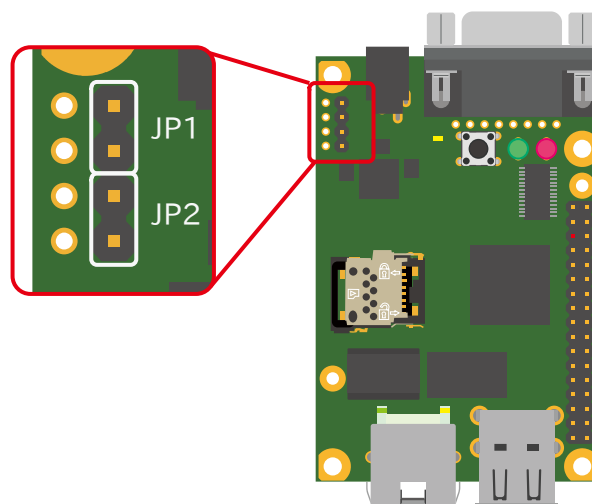




図 3.12 ジャンパピンの位置

ジャンパの設定は「図 3.12. ジャンパピンの位置」で行い、次のように「オープン」・「ショート」で設定を切り替えられます。

 「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。

ジャンパの状態を変更することで、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「3.6.17. 起動デバイスを変更する」を参照してください。

3.1.5. Armadillo に初期設定をインストールする

次に、Armadillo に初期設定（`initial_setup.swu`）をインストールします。`initial_setup.swu` はログインパスワードやユーザー固有の証明書などの初期設定を Armadillo にインストールするためのファイルです。`initial_setup.swu` でインストールされるユーザー固有の証明書がない場合、ユーザーが開発したアプリケーションをインストール、またはアップデートすることができません。このため開発前に、初期化された Armadillo に `initial_setup.swu` をインストールする必要があります。初期化された Armadillo に対してユーザーが開発したアプリケーションのインストール・アップデートを行うために必須の手順になりますので、必ず行ってください。

ここでは、initial_setup.swu を VS Code で作成し、ABOS Web で Armadillo にインストールします。

3.1.5.1. initial_setup.swu の作成

「[図 3.13. initial_setup.swu を作成する](#)」に示すように、VS Code の左ペインの [COMMON PROJECT COMMAND] から [Generate Initial Setup Swu] を実行してください。

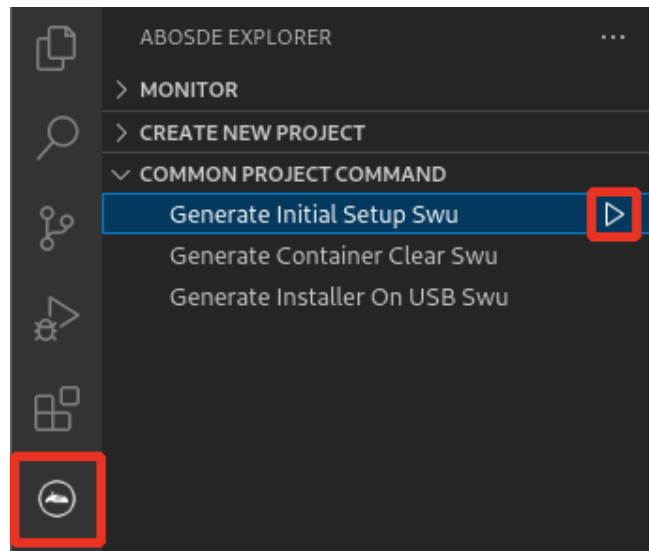


図 3.13 initial_setup.swu を作成する

初回実行時には各種設定の入力を求められます。入力する設定の内容を「[図 3.14. initial_setup.swu 初回生成時の各種設定](#)」に示します。

なお、この後の Python アプリケーションによる動作確認では ABOS Web を使用した手順を記載しています。この後の手順通りに動作確認を行いたい場合は、ABOS Web のパスワードを設定してください。

```
Executing task: ./scripts/generate_initial_setup_swu.sh
```

```
mkdir: ディレクトリ '/home/atmark/mkswu' を作成しました
設定ファイルを更新しました: /home/atmark/mkswu/mkswu.conf
```

```
証明書の共通ネーム(一般名)を入力してください: [COMMON_NAME] ①
```

```
証明書の鍵のパスワードを入力ください (4-1024 文字) ②
```

```
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.tmp'
```

```
-----
```

```
アップデートイメージを暗号化しますか? (N/y) ③
```

```
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) ④
```

```
root パスワード: ⑤
```

```
root のパスワード (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): ⑥
```

```
atmark のパスワード (確認):
```

```
BaseOS/プリインストールコンテナを armadillo.atmark-techno.com サーバーから自動アップデートしますか?
```

```
(N/y) ⑦
```

```
abos-web のパスワードを設定してください。
```

↩


```

abos-web のパスワード (空の場合はサービスを無効にします) : ⑧
abos-web のパスワード (確認) :
/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。
* Terminal will be reused by tasks, press any key to close it.

[ATDE ~]$ ls ~/mkswu
initial_setup.desc  initial_setup.swu  mkswu.conf
swupdate.aes-key   swupdate.key       swupdate.pem ⑨

```

図 3.14 initial_setup.swu 初回生成時の各種設定

- ① 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。
- ② 証明鍵を保護するパスフレーズを 2 回入力します。
- ③ 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「6.7. SWUpdate と暗号化について」を参考にしてください。
- ④ 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。
- ⑤ root のパスワードを 2 回入力します。使用するパスワードは以下のルールに従ってください。
 - ・ 辞書に載っている言葉を使用しない
 - ・ 単調な文字列を使用しない
 - ・ 8 文字以上のパスワード長にする
- ⑥ 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。使用できるパスワードの制限は root と同様です。
- ⑦ 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。
- ⑧ abos-web を使用する場合はパスワードを設定してください。ここで設定したパスワードは abos-web から変更できます。使用できるパスワードの制限は root と同様です。詳細は「3.8.4. ABOS Web のパスワード変更」を参考にしてください。
- ⑨ 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。

ファイルは ~/mkswu/initial_setup.swu に保存されます。

3.1.5.2. initial_setup.swu を Armadillo にインストール

上の手順で作成した SWU イメージ (initial_setup.swu) を Armadillo へインストールします。インストール方法は様々ありますが (「3.3.3.6. SWU イメージのインストール」)、ここでは ABOS Web を使用した手動インストールを行います。

ABOS には ABOS Web という機能が含まれています。この機能を活用することで、Web ブラウザからネットワークの設定や、SWU イメージのインストールなどを簡単に行うことができます。(ただし、Armadillo と作業用 PC が同一 LAN 内に存在している必要があります)

以下の手順に沿って、ABOS Web へアクセスし、initial_setup.swu のインストールを行ってください。

まず、「図 3.15. ABOS にアクセスするための接続」のとおり Armadillo に配線を行い、電源を入れてください。なお、JP1 をオープンにし、JP2 のみをショートさせることに注意してください。

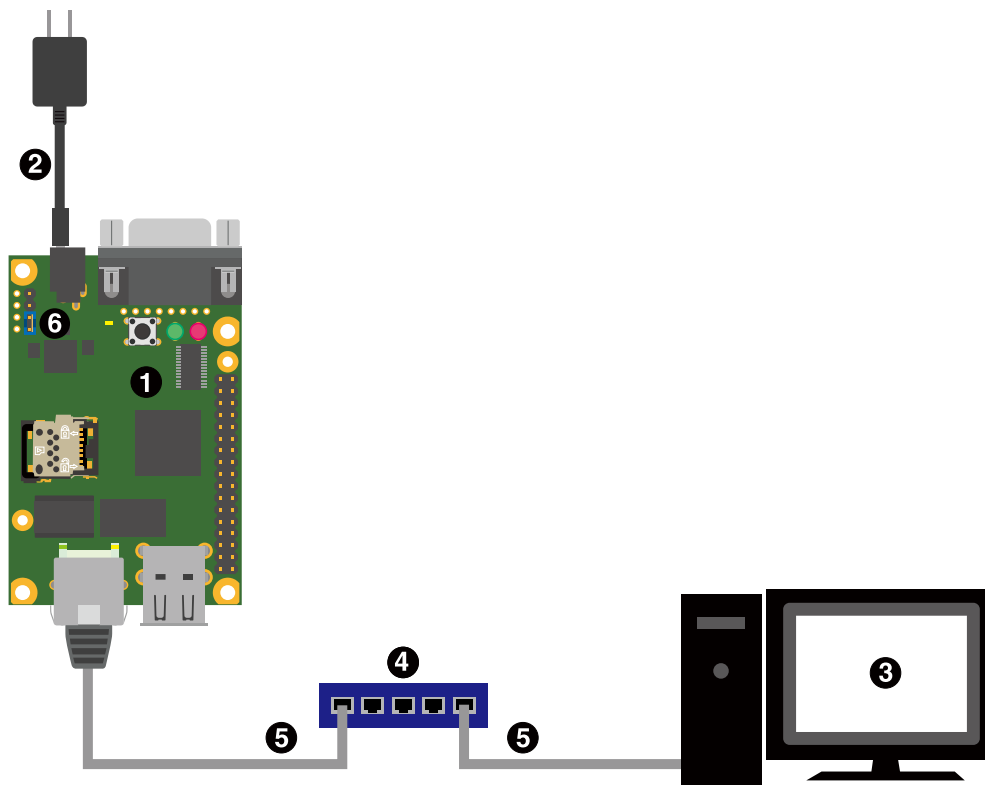


図 3.15 ABOS にアクセスするための接続

- ❶ Armadillo-640
- ❷ AC アダプタ (5V/2.0A)
- ❸ 作業用 PC
- ❹ LAN HUB
- ❺ Ethernet ケーブル
- ❻ ジャンパーソケット

1 分ほど待機して、ABOSDE でローカルネットワーク上の Armadillo をスキャンします。「図 3.16. ABOSDE でローカルネットワーク上の Armadillo をスキャンする」の赤枠で囲われているボタンをクリックしてください。

Armadillo が正常に起動していた場合、「図 3.17. ABOSDE に表示されている Armadillo を更新する」の一覧に起動した Armadillo が armadillo.local という名称で表示されます。表示されない場合は 1 分

ほど待機してから「図 3.17. ABOSDE に表示されている Armadillo を更新する」の赤枠で囲われているマークをクリックしてスキャンを再度試みてください。

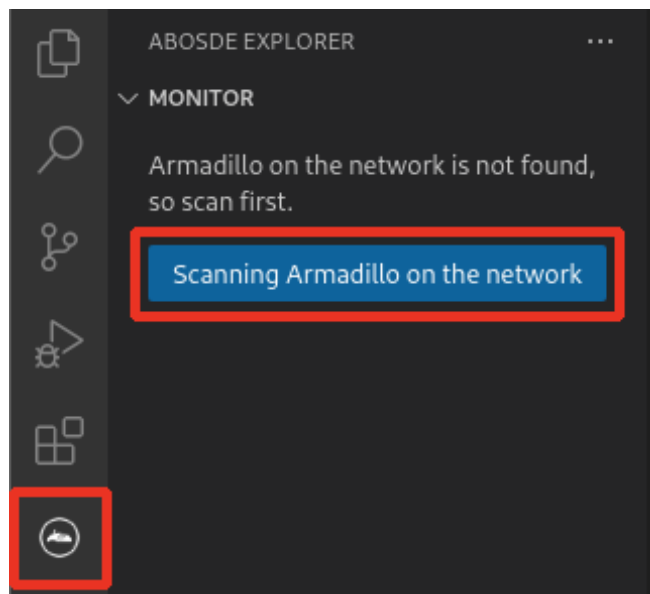


図 3.16 ABOSDE で ローカルネットワーク上の Armadillo をスキャンする

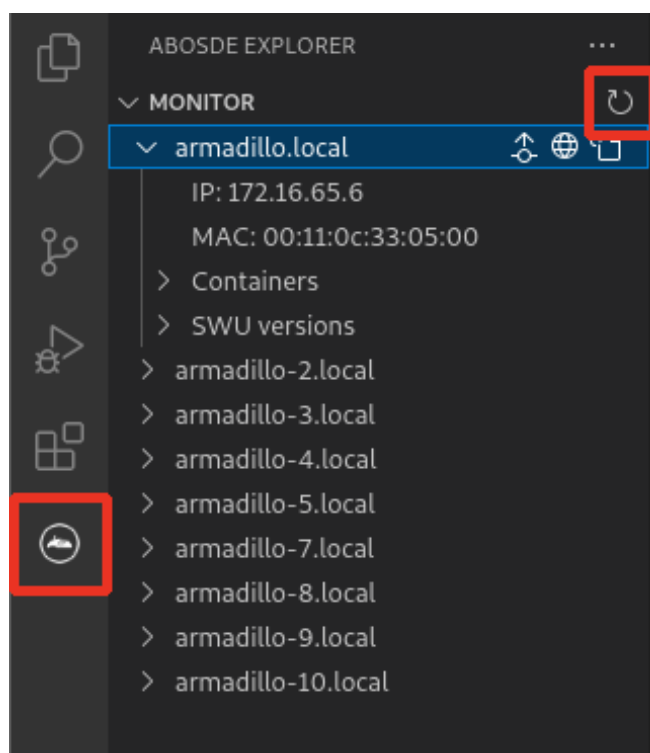


図 3.17 ABOSDE に表示されている Armadillo を更新する

ただし、ATDE のネットワークをブリッジ接続以外に設定している場合は Armadillo がリストに表示されない場合があります。表示するためには ATDE のネットワークをブリッジ接続に設定してください。また、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) が、2 台目では armadillo-2.local、3 台目では armadillo-3.local の

ように、違うものが自動的に割り当てられます。目的の Armadillo がどのホスト名なのか不明な場合には、Armadillo のラベルに記載されている MAC アドレスと一致するもの（「図 3.18. ABOSDE を使用して ABOS Web を開く」の赤枠に表示されます）を探してください。

続いて、「図 3.18. ABOSDE を使用して ABOS Web を開く」の赤枠で囲われているマークをクリックして、ABOS Web を Web ブラウザで開きます。

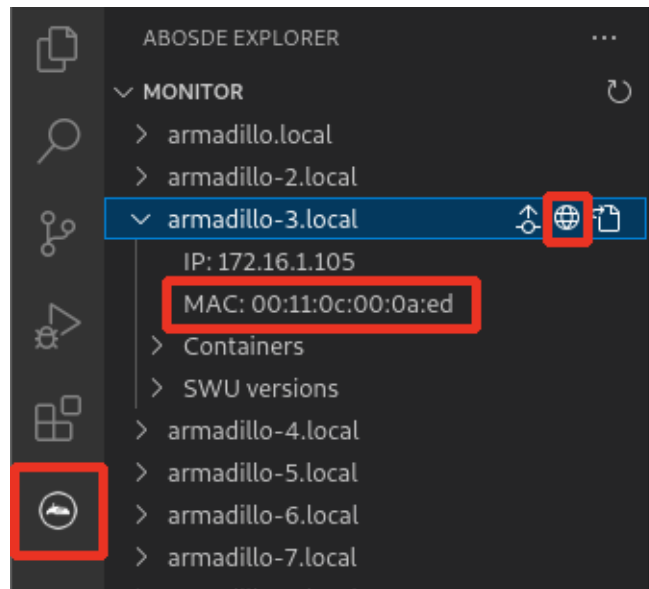


図 3.18 ABOSDE を使用して ABOS Web を開く

3.1.5.3. ABOS Web へアクセス

ABOS Web が正常に起動していれば、Web ブラウザに パスワード登録画面（「図 3.19. パスワード登録画面」）が表示されます。initial_setup.swu を作成する手順で設定したパスワードを入力して、ABOS Web のログイン用パスワードを設定します。



図 3.19 パスワード登録画面

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 3.20 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。

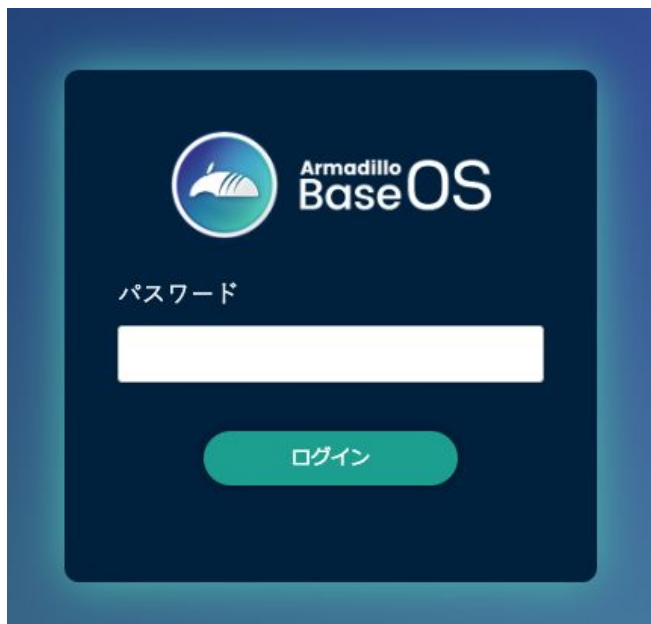


図 3.21 ログイン画面

ログインに成功すると、ABOS Web の設定画面（「図 3.22. トップページ」）に表示が変わり、設定操作を行うことができます。これで、ABOS Web へのアクセスが完了しました。

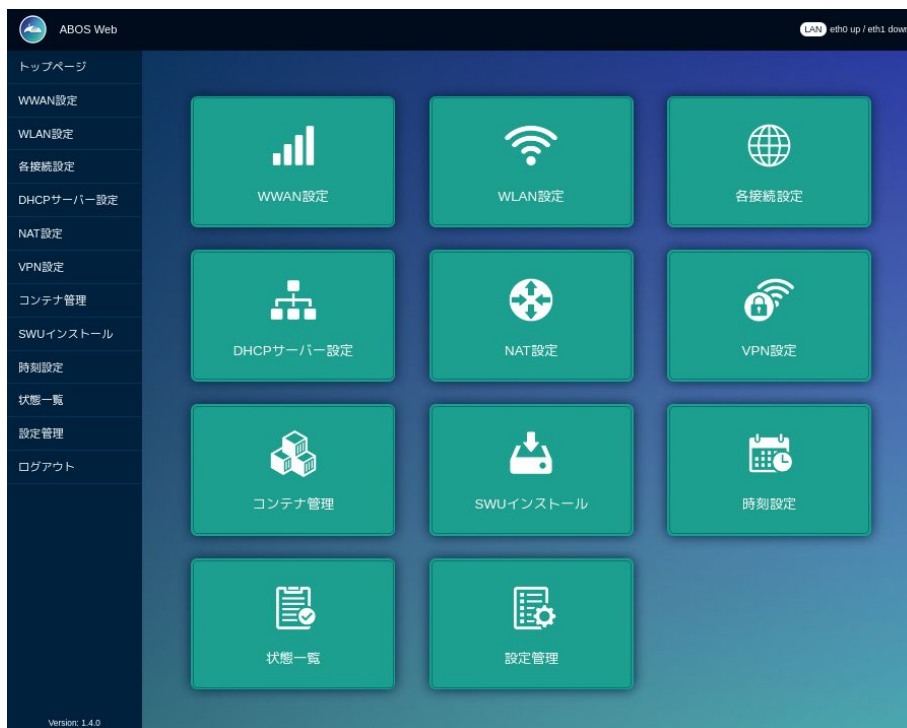


図 3.22 トップページ

3.1.5.4. ABOS Web から initial_setup.swu をインストール

ABOS Web のトップページから"SWU インストール"をクリックして、「図 3.23. SWU インストール」の画面に遷移します。



図 3.23 SWU インストール

"参照..."から `~/mkswu/initial_setup.swu` を選択し、"インストール"をクリックしてください。数分ほど待機すると「図 3.24. SWU インストールに成功した画面」のように"インストールが成功しました。"と表示され、Armadillo が再起動します。(ABOS Web も再起動されるので、再起動完了後にページを更新するとログイン画面に戻ります)



図 3.24 SWU インストールに成功した画面

これで Armadillo に初期設定をインストールする手順が完了です。インストール完了後に ~/mkswu ディレクトリ以下にある mkswu.conf と、鍵ファイルの swupdate.* をなくさないようにしてください。



ABOS Web にブラウザから直接アクセスする

ABOSDE を使わずに、直接 Web ブラウザのアドレスバーに ABOS Web の URL を入力することでも ABOS Web にアクセスできます。ATDE で Web ブラウザを起動した後、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>

複数台の Armadillo が接続されている場合には、armadillo.local の部分が armadillo-2.local や armadillo-3.local となっている可能性があります。これらは ABOSDE のリストに表示されているホスト名と同名ですので、目的の Armadillo と一致するホスト名を入力してください。

また、Web ブラウザから直接アクセスする方法では、ホスト名ではなく IP アドレスを指定することもできます。例えば、Armadillo の（ネットワークコネクタの）IP アドレスが 192.0.2.80 である場合は、次の URL を入力してください：<https://192.0.2.80:58080>

IP アドレスを固定している場合は IP アドレスを指定する方法が便利になる場面もあります。また、IP アドレスを指定する方法は ATDE のネットワークを NAT に設定している場合でも有効です。



ABOS Web からログアウトする

ログアウトを行う場合は、サイドメニューから "ログアウト" を選択してください。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。

3.1.6. Python アプリケーションで動作確認する

本項では LED を点滅させる Python のサンプルアプリケーションを使用して、Armadillo による開発方法の勝手を大まかに把握したい方を想定した簡単な動作確認を行います。なお、開発環境のセットアップに直接関わる手順ではないので、この動作確認が不要な方は本項をスキップしてください。

3.1.6.1. プロジェクトの作成

Armadillo でのアプリケーションの開発には ABOSDE を使用します。

VS Code の左ペインの [A600] から [Python New Project] を実行（右に表示されている三角形ボタン）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。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。この操作により、選択した保存先に、入力したプロジェクト名と同名のディレクトリが作成されます。

また、ここでは次のように設定しています。

- ・ 保存先：ホームディレクトリ
- ・ プロジェクト名：my_project

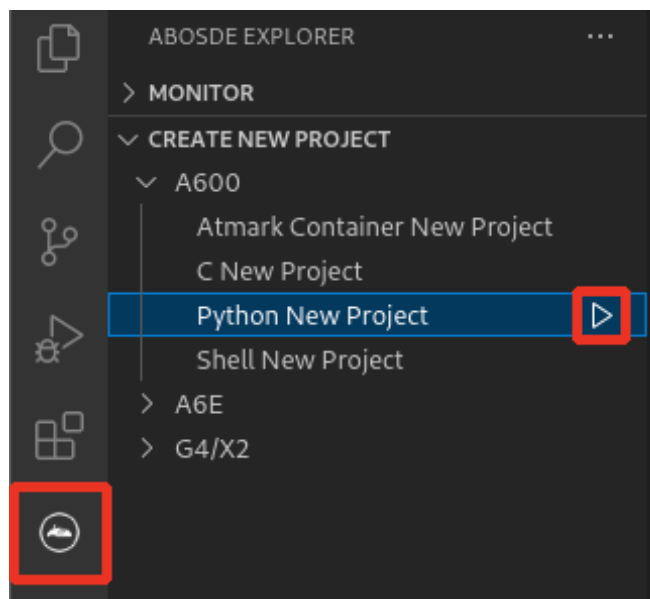


図 3.25 プロジェクトを作成する

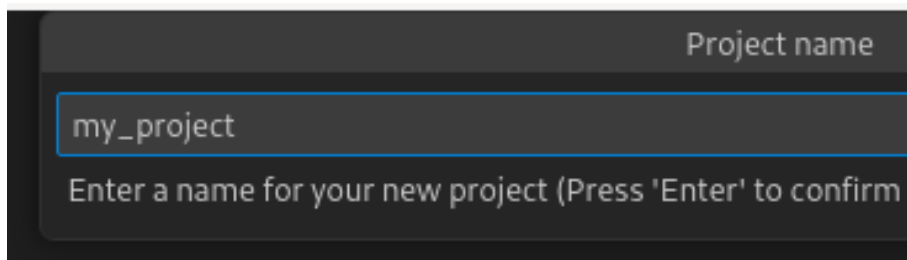


図 3.26 プロジェクト名を入力する

プロジェクトを作成したら、VS Code で my_project のディレクトリを開いてください。

3.1.6.2. 初期設定

プロジェクトを作成する度に、初期設定を行う必要があります。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。以下の手順を実施してください。

VS Code の左ペインの [my_project] から [Setup environment] を実行します。

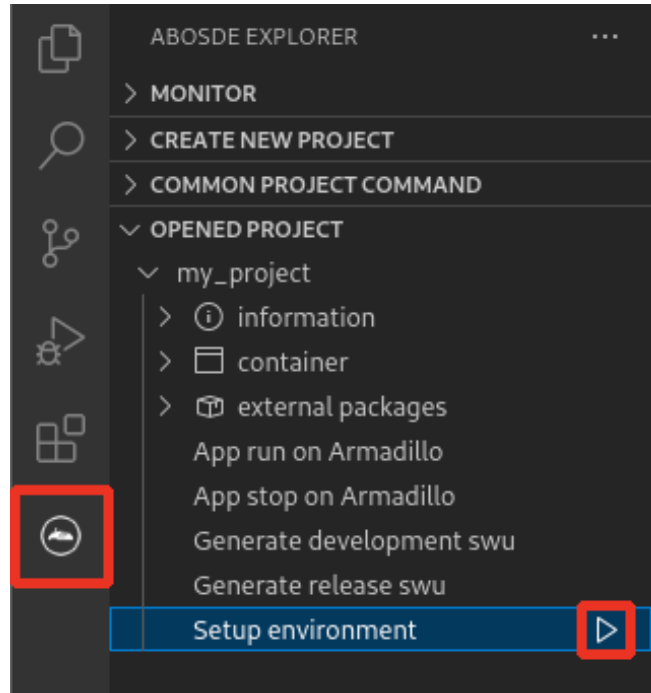


図 3.27 VS Code で初期設定を行う

選択すると、VS Code の下部に以下のようなターミナルが表示されます。

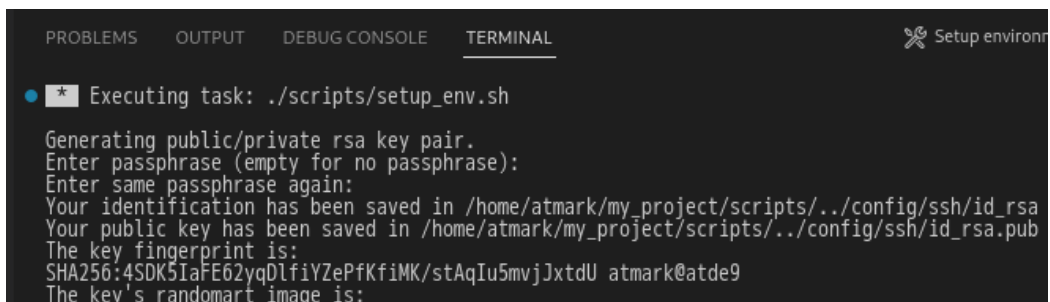


図 3.28 VS Code のターミナル

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): ①
Enter same passphrase again: ②
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. ③

図 3.29 SSH 用の鍵を生成する

- ① 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。
- ② 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。
- ③ 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。

3.1.6.3. 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 へインストールします。

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

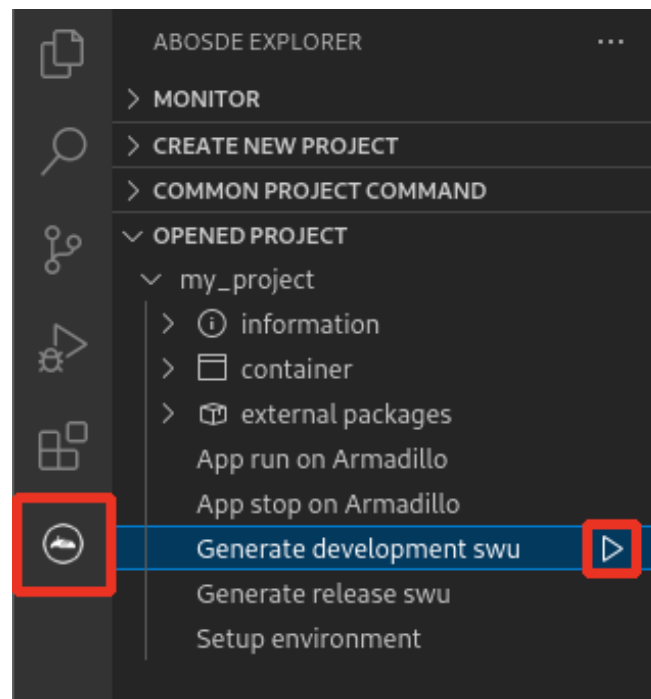


図 3.30 VS Code でコンテナイメージの作成を行う

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。VS Code 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。

```
コンテナイメージを ./swu/my_project.tar に保存しました。
./swu/app.desc のバージョンを 1 から 2 に変更しました。
./development.swu を作成しました。
次は Armadillo に ./development.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 3.31 コンテナイメージの作成完了

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

3.1.6.4. 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

上で作成した development.swu を Armadillo へインストールします。initial_setup.swu をインストールしたときと同様に ABOS Web からインストールさせることも可能ですが、ここでは ABOSDE を使用してインストールする手順をご紹介します。

「図 3.32. ABOSDE で Armadillo に SWU をインストール」のように、目的の Armadillo の隣にある赤枠で囲まれているボタンをクリックしてください。パスワードの入力を要求されますので、ABOS Web のパスワードを入力してください。その後、~/my_project/development.swu を選択してインストールを開始します。

インストールが成功すると、VS Code のターミナルに Successfully installed SWU と表示されます。

インストール後に自動で Armadillo が再起動し、1 分ほど待機すると LED が点滅します。

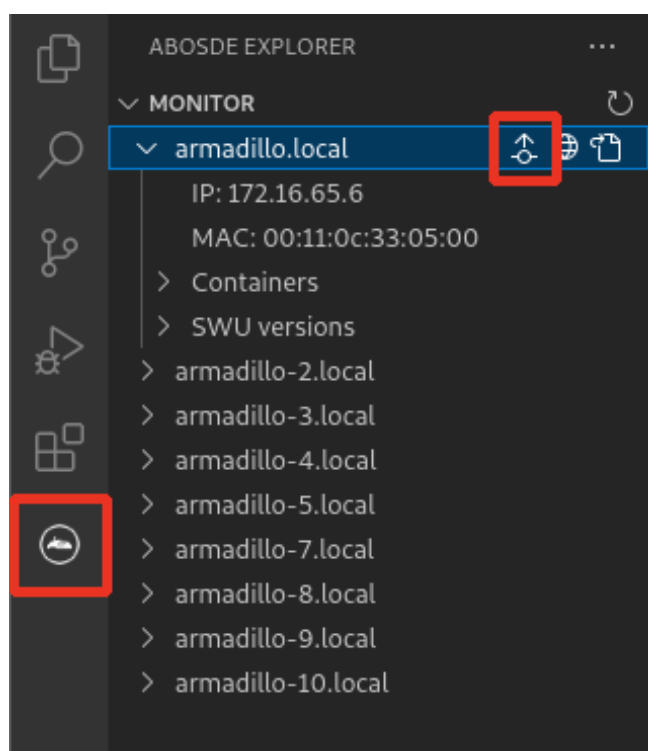


図 3.32 ABOSDE で Armadillo に SWU をインストール

3.1.6.5. ssh 接続に使用する IP アドレスの設定

以下の手順にしたがい、ABOS Web が動作している Armadillo の一覧を確認し、ssh 接続に使用する Armadillo の IP アドレスを指定してください。なお、この手順は Armadillo の IP アドレス が変更される度に行う必要があります。

「図 3.16. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする」の赤枠で囲われているボタン、または「図 3.17. ABOSDE に表示されている Armadillo を更新する」の赤枠で囲われているマークをクリックして、ローカルネットワーク上で ABOS Web が実行されている Armadillo をスキャンしてください。

その後、目的の Armadillo について、「図 3.33. ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する」の赤枠で囲われているマークをクリックしてください。

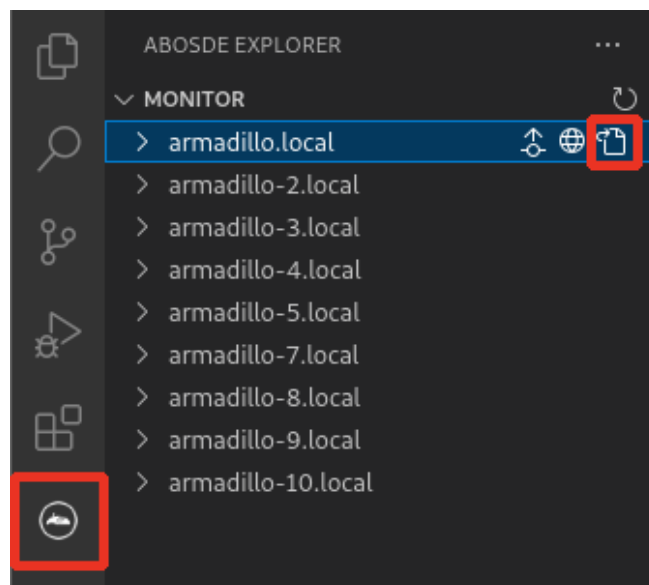


図 3.33 ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する

これにより、指定した Armadillo の IP アドレスを ssh 接続に使用する IP アドレスに設定することができます。また、プロジェクトディレクトリ内の config/ssh_config ファイルに指定した Armadillo の IP アドレスが記載されます。ATDE のネットワークを NAT に設定している場合や、ABOS Web を起動していない場合等、ABOSDE のリストに Armadillo が表示されない場合は、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 x.x.x.x ❶
  User root
  IdentityFile ${HOME}/.ssh/id_ed25519_vscode
  UserKnownHostsFile config/ssh_known_hosts
  StrictHostKeyChecking accept-new
```

図 3.34 ssh_config を編集する

- ❶ 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。

3.1.6.6. アプリケーションの実行

VS Code 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。

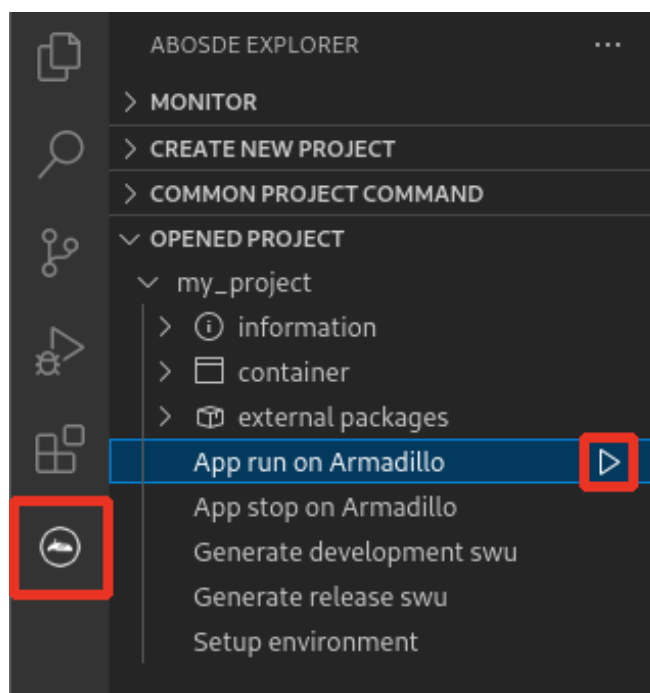


図 3.35 Armadillo 上でアプリケーションを実行する

VS Code 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 3.36 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 VS Code の左ペインの [my_project] から [App stop on Armadillo] を実行してください。

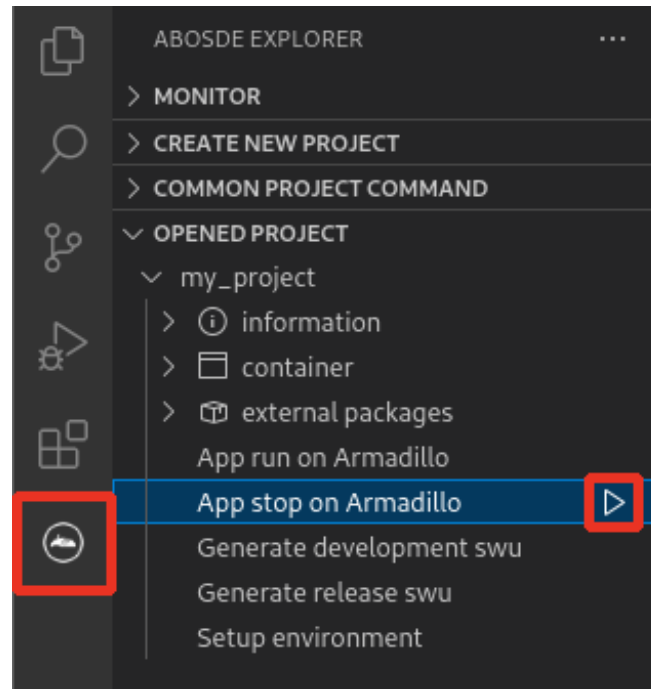


図 3.37 アプリケーションを終了する

3.1.6.7. アプリケーションの削除

動作確認として使用した Python アプリケーションを削除します。ABOSDE から Armadillo のコンテナイメージを全て削除する SWU イメージを作成します。この方法はコンテナイメージを全て削除する方法ですので、開発中に複数のコンテナイメージを使用している場合などはそれらも削除されることに注意してください。

VS Code の左ペインの [COMMON PROJECT COMMAND] から [Generate Container Clear Swu] を実行すると、SWU イメージが作成されます。SWU イメージは `~/mkswu/container_clear.swu` に保存されます。

この SWU イメージを Armadillo へインストールします。

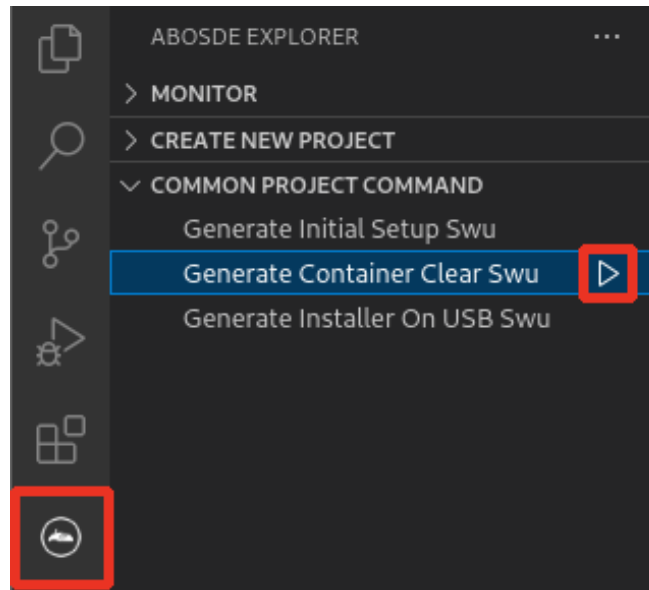


図 3.38 Armadillo 上のコンテナイメージを削除する

「図 3.32. ABOSDE で Armadillo に SWU をインストール」のように、目的の Armadillo の隣にある赤枠で囲まれているボタンをクリックしてください。パスワードの入力を要求されますので、ABOS Web のパスワードを入力してください。その後、`~/mkswu/container_clear.swu` を選択してインストールを開始します。

インストール後に自動で Armadillo が再起動し、LED が点滅しなくなります。

3.1.7. シリアルコンソールを使用する

Armadillo ではシリアルコンソールを通じて Linux コマンドを直接実行することができます。シリアルコンソールを活用することで、ABOS Web や ABOSDE からではできない多くのことが可能になるため、より応用的な開発やメンテナンス・デバッグの際に重宝します。また、この章以降ではシリアルコンソールを使用した手順が多々登場します。

本項ではシリアル通信ソフトウェア(minicom)を使用したシリアルコンソールの操作方法について記載しています。



minicom のセットアップについて

ATDE9 v20240925 以降の ATDE では Armadillo 用の minicom の通信設定を既に済ませた状態で配布しています。これより前のバージョンの場合は、「6.25. シリアル通信ソフトウェア(minicom)のセットアップ」を参照して minicom のセットアップを行ってください。

3.1.7.1. Armadillo と開発用 PC を接続

Armadillo-640 のシリアルコンソールを使用するために、「図 3.41. シリアルコンソールを使用する配線例」のとおり配線を行ってください。この配線図は Armadillo-640 のシリアルコンソールを使用するための最低限の配線ですので、これに加えて他のインターフェースを接続しても問題ありません。



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。

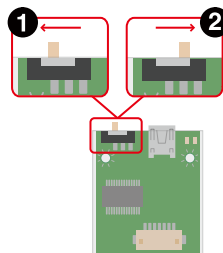


図 3.39 スライドスイッチの設定

- ① 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。
- ② 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はオートブートモードとなり、Linux が起動します。



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目印にして、下図のように接続してください。

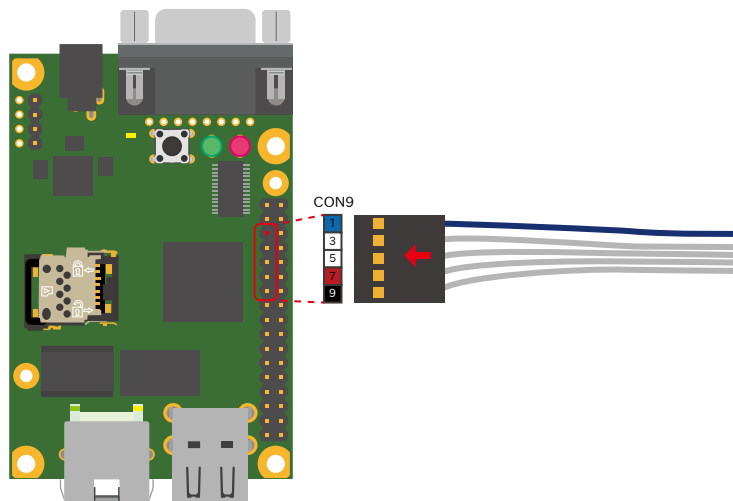


図 3.40 CON9-USB シリアル変換アダプタ接続図

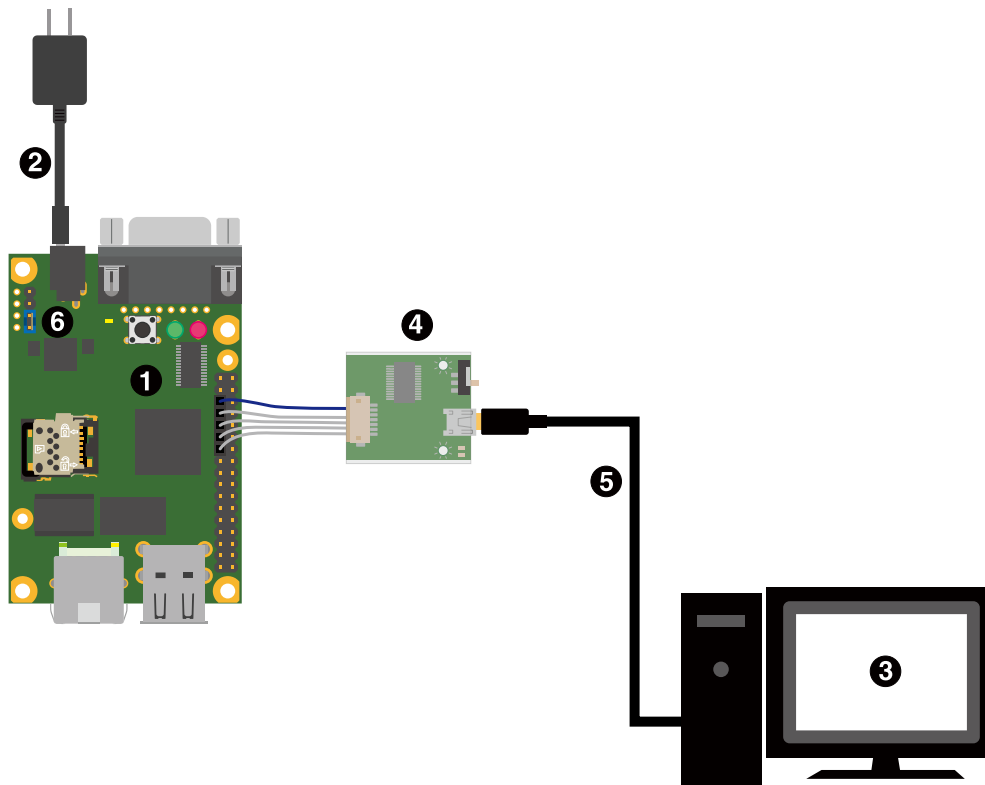


図 3.41 シリアルコンソールを使用する配線例

- ❶ Armadillo-640
- ❷ AC アダプタ (5V/2.0A)
- 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)
- ❻ ジャンパーソケット



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。

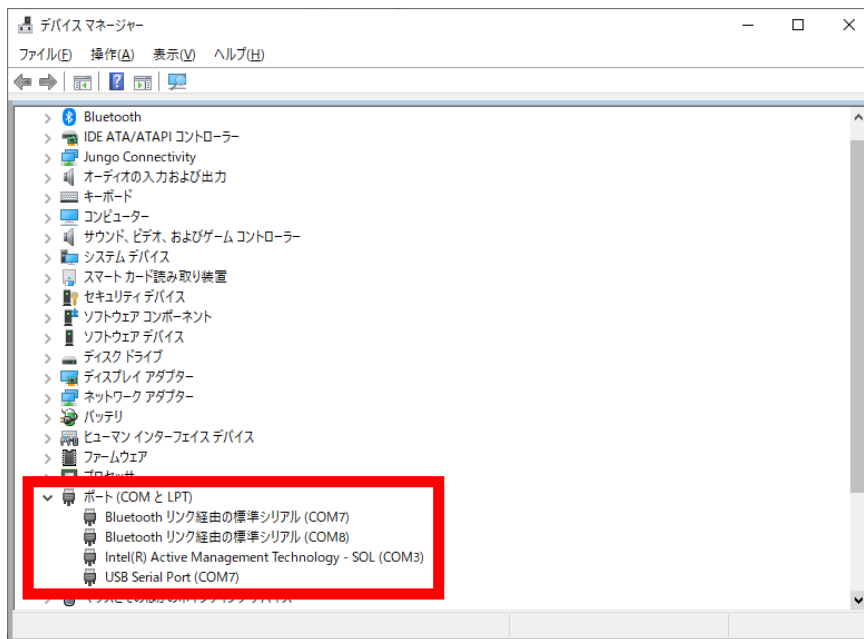


図 3.42 COM7 が競合している状態

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。

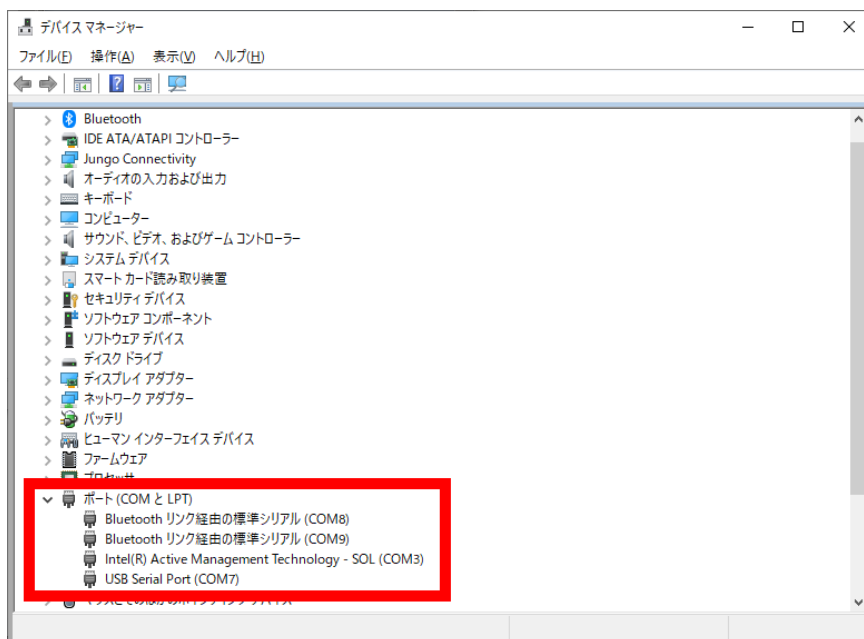


図 3.43 Bluetooth に割当の COM を変更した状態

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。

3.1.7.2. minicom の起動

minicom を起動する前に、Armadillo からのログを表示させるため、「表 3.2. 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 に接続してください。

表 3.2 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

「図 3.44. minicom 起動方法」のようにして、minicom を起動してください。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。

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

図 3.44 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
```

既に電源が接続されていて起動している場合は、Enter を 1 回押してください。次のようなログインプロンプトが表示されます。（「3.1.7.3. ログイン」）

```
armadillo login:
```

電源が接続されていない場合は、電源入力インターフェースに電源を接続して Armadillo-640 を起動してください。

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!
buttd        | * 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:
```

3.1.7.3. ログイン

起動が完了するとログインプロンプトが表示されます。初期状態では「root」ユーザーと、一般ユーザーである「atmark」ユーザーが存在しますが、「atmark」ユーザーは初期状態ではロックされていますので、「root」ユーザーでログインしてください。initial_setup.swu をインストールしていない場合、「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。

「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することで、「atmark」ユーザーのロックが解除されます。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

「atmark」ユーザーは初期状態ではロックされています。そのため、「root」ユーザーでログイン後に「atmark」ユーザーのパスワードを設定してから「atmark」ユーザーでログインします。

```
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: ❸
Welcome to Alpine!
```

- ❶ atmark ユーザーのパスワード変更コマンドです。「5.4.1. SWU イメージの作成」を使用した場合には不要です。
- ❷ パスワードファイルを永続化します。
- ❸ 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「6.1. persist_file について」を参照してください。

3.1.7.4. Armadillo の終了方法

eMMC や USB メモリ等へ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
```



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。



電源を再投入する際は、コンデンサに蓄えられた電荷を抜くため、電源を切断後、一定時間以上待つ必要があります。開発セット付属の AC アダプタの場合に必要な時間は以下のとおりです。

- ・ DC プラグ側で電源を切断した場合：約 5 秒
- ・ AC プラグ側で電源を切断した場合：約 1 分

コンデンサに蓄えられた電荷が抜ける前に電源を再投入した場合、電源シーケンスが守られず、起動しない等の動作不具合の原因となります。

3.1.7.5. minicom の終了

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 3.45 minicom 終了確認

3.1.8. 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/>

3.1.8.1. 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

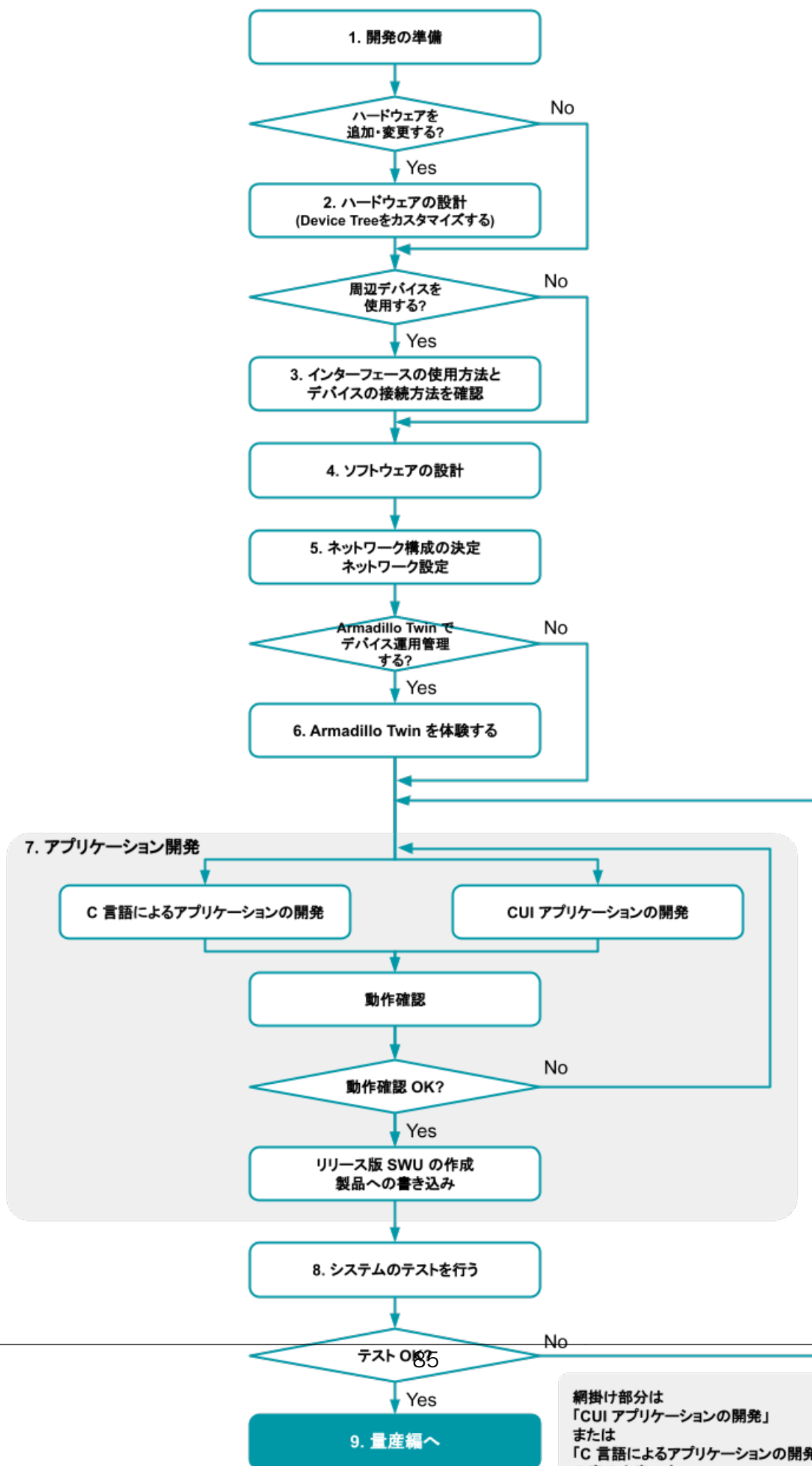
<https://armadillo.atmark-techno.com/armadillo-640/register>

以上で開発環境のセットアップと動作確認の手順は終了です。

3.2. アプリケーション開発の流れ

基本的な Armadillo-640 でのアプリケーション開発の流れを「図 3.46. アプリケーション開発の流れ」に示します。

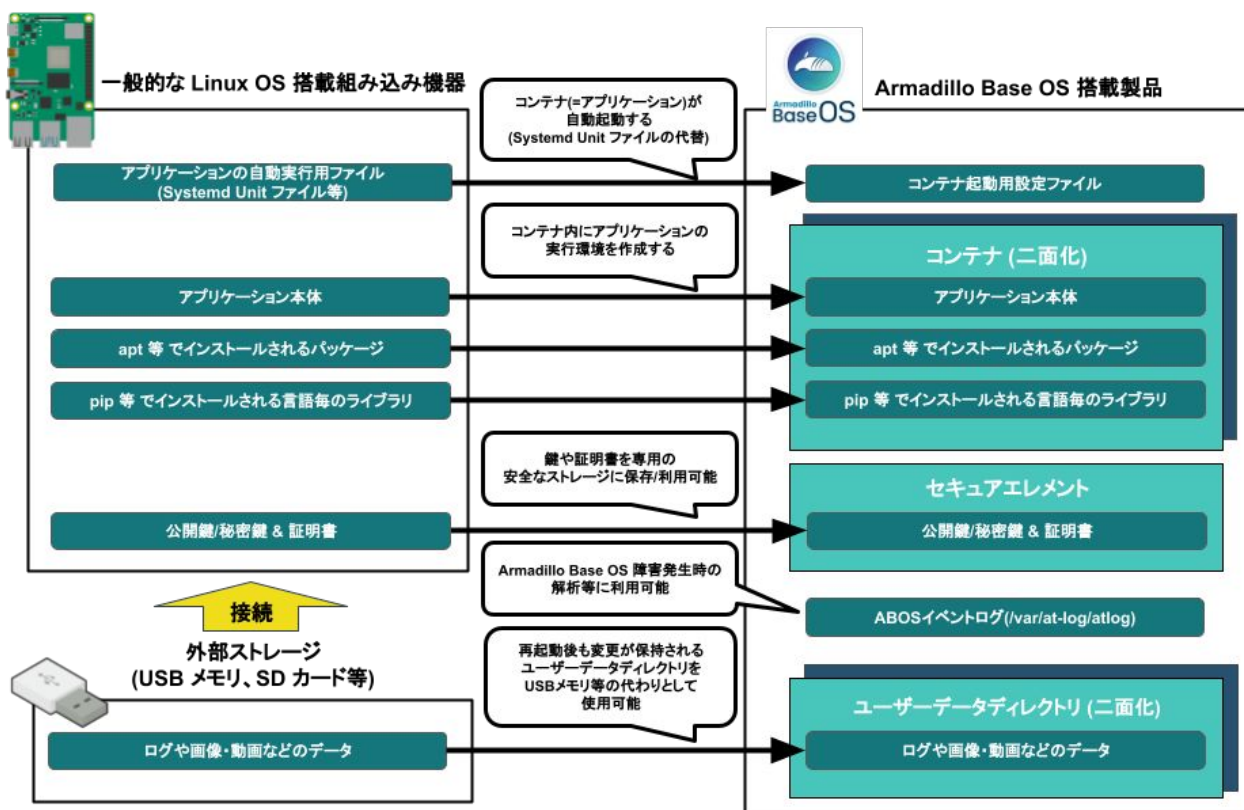
本章では、「図 3.46. アプリケーション開発の流れ」に示す開発時の流れに沿って手順を紹介していきます。



3.3. 開発前に知っておくべき Armadillo Base OS の機能・特徴

「2.1.3. Armadillo Base OS とは」にて Armadillo Base OS についての概要を紹介しましたが、開発に入るにあたってもう少し詳細な概要について紹介します。

3.3.1. 一般的な Linux OS 搭載組み込み機器との違い

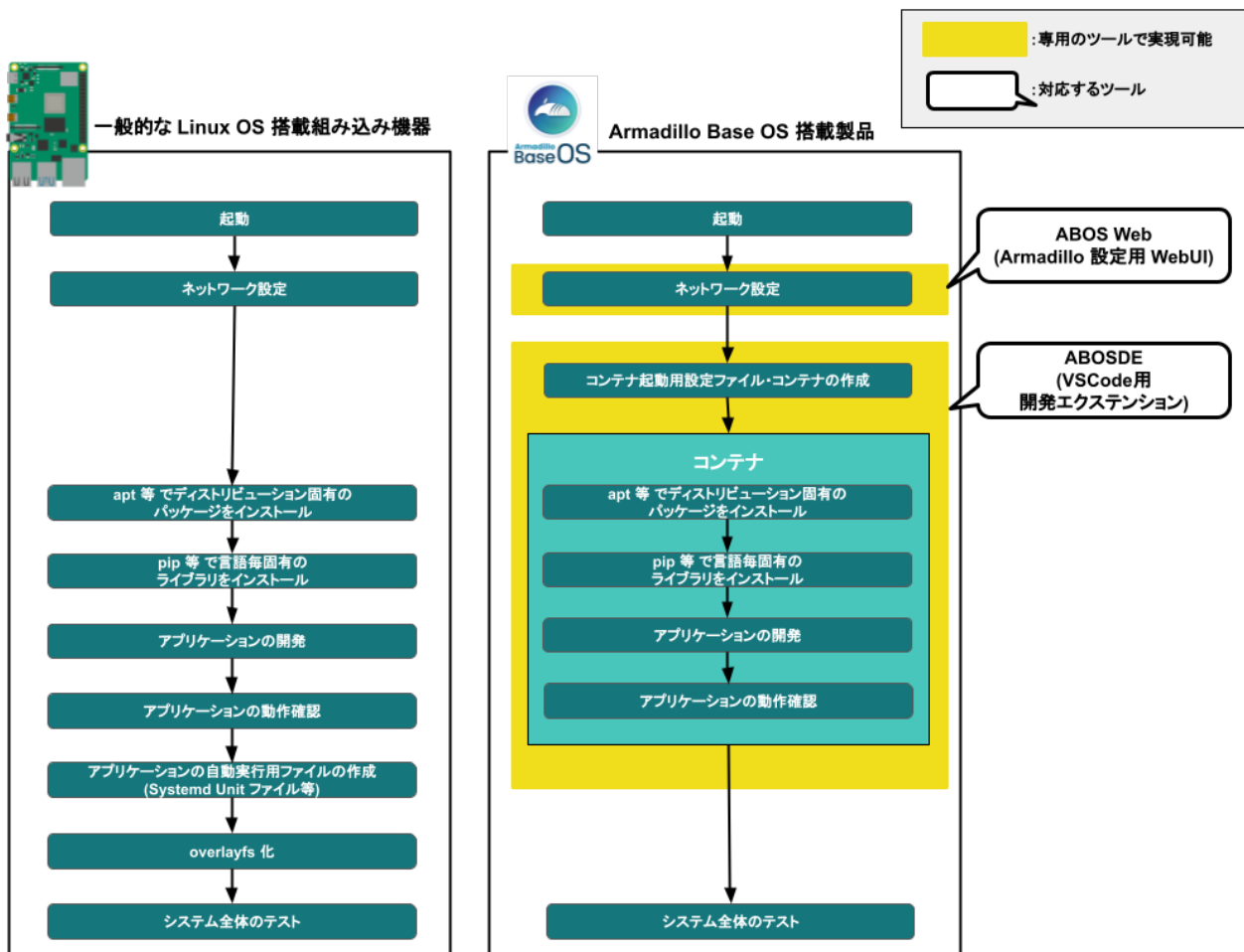


Linux OS 搭載組み込み機器ではアプリケーションの実行環境をユーザーランド上に直接用意し、Systemdなどでアプリケーションを自動実行させるのが一般的です。Armadillo Base OS 搭載機器では、アプリケーションの実行環境をコンテナ内に用意して、コンテナ起動用設定ファイルを所定の場所に配置することでコンテナ(=アプリケーション)を自動実行させます。

また、Linux OS 搭載組み込み機器では、ストレージの保護のために overlaysfs で運用するのが一般的です。そのため、アプリケーションが出力するログや画像などのデータは、USBメモリなどの外部デバイスに保存する必要があります。Armadillo Base OS 搭載機器もルートファイルシステムが overlaysfs 化されていますが、内部に USBメモリなどと同じように使用できるユーザーデータディレクトリを持っており、別途外部記録デバイスを用意しておく必要はありません。

Armadillo Base OS 搭載機器は、標準でセキュアエレメントを搭載しており、対応した暗号化方式の認証鍵や証明書を安全に保存・利用することが可能です。

3.3.2. Armadillo Base OS 搭載機器のソフトウェア開発手法



Armadillo Base OS 搭載機器上で動作するソフトウェアの開発は、基本的に作業用 PC 上で行います。

ネットワークの設定は ABOS Web という機能で、コマンドを直接打たずとも設定可能です。

開発環境として、ATDE (Atmark Techno Development Environment) という仮想マシンイメージを提供しています。その中で、ABOSDE (Armadillo Base OS Development Environment) という、Visual Studio Code にインストールできる開発用エクステンションを利用してソフトウェア開発を行います。

ABOSDE を使用することで、コンテナ及びコンテナ自動起動用設定ファイルの作成、コンテナ内におけるパッケージのインストール、コンテナ内で動作するアプリケーション本体の開発をすべて VS Code 内で行うことができます。

3.3.3. アップデート機能について

Armadillo-640 では、開発・製造・運用時にソフトウェアを書き込む際に、SWUpdate という仕組みを利用します。

3.3.3.1. 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機能を実現しています。

- ・ 機密性、完全性、真正性の担保
- ・ A/B アップデート(アップデートの二面化)
- ・ リカバリーモード
- ・ ソフトウェアの圧縮、暗号化、署名付与
- ・ Armadillo Twin でのリモートアップデート対応
- ・ Web サーバーでのリモートアップデート対応
- ・ ダウングレードの禁止



2024 年 2 月までは、hawkBit の WebUI を利用したアップデートも紹介していましたが、hawkBit は 2024 年 3 月 22 日に行われたバージョン 0.5.0 へのアップデートで、これまで採用していた Web UI を廃止しました。これに伴い、今後 OTA によるアップデートを行いたい場合は、Armadillo Twin [<https://armadillo.atmark-techno.com/guide/armadillo-twin/>] の利用を推奨します。

なお、hawkBit 0.4.1 の配布は継続していますので、こちらを利用する場合は Armadillo-640 開発用ツール [<https://armadillo.atmark-techno.com/resources/software/armadillo-640/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。

hawkBit に関する詳細な情報は hawkBit 公式サイト [<https://eclipse.dev/hawkbit/>] を参照してください。

3.3.3.2. SWU イメージとは

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...

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などです。

3.3.3.3. 機密性、完全性、真正性の担保

ユーザーは SWU イメージをネットワーク/ストレージ経由で Armadillo にインストールします。

インターネットを通じて Armadillo にインストールする場合、以下の脅威が存在することが考えられます。

- ・ 攻撃者が正規のユーザーを偽りデータをインストールする（なりすまし）
- ・ データの一部を悪意のあるコードに書き換えられる（改ざん）
- ・ データを盗み見される（盗聴）

Armadillo Base OS では暗号化技術、SHA-256 によるハッシュ化、デジタル署名を駆使することで、インストールするデータに対する機密性、完全性、真正性を保証します。

それらの機能は SWUpdate によって実現しています。SWUpdate は以下の対策を提供します。

- ・ SWU イメージ内の Armadillo にインストールするデータを暗号化する
- ・ デジタル署名により正規の SWU イメージであることを保証する
- ・ 復号したデータに対してもチェックサムを計算して、インストールするデータが正しいことを保証する

これらの対策により、たとえ攻撃者が不正な SWU イメージを Armadillo に送信したとしてもデジタル署名により正規の SWU イメージでないことがわかります。

攻撃者がインターネット上で SWU イメージ内のデータを書き換えたとしても、インストール前にそのデータに対してチェックサムが正しいかを確認します。そのため、不正なデータが Armadillo にインストールされることはありません。

また、攻撃者がネットワーク上で SWU イメージのデータを盗み見たとしても暗号化されているので、重要なデータが漏洩することはありません。

3.3.3.4. A/B アップデート(アップデートの二面化)

A/B アップデートは、Flash メモリにパーティションを二面確保し、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

3.3.3.5. ロールバック(リカバリー)

アップデート直後に起動に失敗した場合、起動可能な状態へ復帰するためアップデート前の状態にロールバックします。

ロールバック状態の確認は「6.17. ロールバック状態を確認する」を参照してください。

自動ロールバックが動作する条件は以下の通りです：

- ・ アップデート直後の再起動、または「abos-ctrl rollback-clone」コマンドを実行した後(アップデートが成功した後では古いバージョンに戻りません)
- ・ 以下のどちらかに該当した場合：
 - ・ rootfs にブートに必要なファイルが存在しない (/boot/ulimage, /boot/armadillo.dtb)
 - ・ 起動を 3 回試みて、Linux ユーザーランドの「reset_bootcount」サービスの起動まで至らなかった

また、ユーザースクリプト等で「abos-ctrl rollback」コマンドを実行した場合にもロールバック可能となります。このコマンドで「--allow-downgrade」オプションを設定すると古いバージョンに戻すことも可能です。

いずれの場合でもロールバックが実行されると /var/at-log/atlog にログが残ります。



Armadillo Base OS 3.19.1-at.4 以前のバージョンではアップデート直後の条件が存在しなかったため、古いバージョンに戻る事ができる問題がありました。

最新の Armadillo Base OS へのアップデートを推奨しますが、上記バージョン以前の Armadillo Base OS をご利用でダウングレードを防ぎたい場合は、以下のコマンドを入力することで回避可能です：

```
[armadillo ~]# sed -i -e 's/fw_setenv bootcount/& ¥&¥& fw_setenv
upgrade_available/' /etc/init.d/reset_bootcount
[armadillo ~]# tail -n 3 /etc/init.d/reset_bootcount
fw_setenv bootcount && fw_setenv upgrade_available
end $? "Could not set bootloader env"
}
[armadillo ~]# persist_file -v /etc/init.d/reset_bootcount
'/mnt/etc/init.d/reset_bootcount' -> '/target/etc/init.d/
reset_bootcount'
```

3.3.3.6. SWU 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。

- ・ 手元でイメージをインストールする方法
 - ・ ABOS Web を使用した手動インストール
 - ・ ABOSDE から ABOS Web を使用した手動インストール

- ・ USB メモリまたは microSD カードからの自動インストール
- ・ 外部記憶装置からイメージのインストール (手動)
- ・ リモートでイメージをインストールする方法
 - ・ Armadillo Twin を使用した自動インストール
 - ・ ウェブサーバーからイメージのインストール (手動)
 - ・ ウェブサーバーからの定期的な自動インストール

それぞれのインストール方法の詳細については、以下に記載しております。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、「6.3.5. swupdate がエラーする場合の対処」をご覧ください。

- ・ ABOS Web を使用した手動インストール

Armadillo-640 で動作している Web アプリケーションの ABOS Web を使用してアップデートすることができます。「6.9.4. SWU インストール」を参考にしてください。

- ・ ABOSDE から ABOS Web を使用した手動インストール

VS Code 拡張機能の ABOSDE を使用することで、Armadillo-640 で動作している ABOS Web 経由でアップデートすることができます。「6.10.5. Armadillo に SWU をインストールする」を参考にしてください。

- ・ USB メモリまたは microSD カードからの自動インストール

Armadillo-640 に USB メモリ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 microSD 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。

- ③ /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、コンソールで証明書が間違っているイメージのエラーは以下の様に表示されます。

```
[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 証明書エラーのメッセージ。

- 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
```

- Armadillo Twin を使用した自動インストール

Armadillo Twin 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。「5.5. Armadillo Twin から複数の Armadillo をアップデートする」を参考にしてください。

- 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-600-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。

3.3.4. ファイルの取り扱いについて

Armadillo Base OS 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。

```
[armadillo ~]# echo test > test
[armadillo ~]# persist_file -v test
'/root/test' -> '/mnt/root/test'
```

図 3.47 persist_file コマンド実行例

persist_file コマンドの詳細については、「6.1. persist_file について」を参照してください。

また、SWUpdate によってルートファイルシステム上に配置されたファイルについては、persist_file を実行しなくても保持されます。開発以外の時は安全のため、persist_file コマンドではなく SWUpdate による更新を実行するようにしてください。

3.3.4.1. 電源を切っても保持されるディレクトリ(ユーザーデータディレクトリ)

「3.3.4. ファイルの取り扱いについて」にて、Armadillo Base OS 上のファイルは通常、persist_file コマンドを実行せずに電源を切ると変更内容が保存されないと紹介しましたが、「表 3.3. 電源を切っても保持されるディレクトリ(ユーザーデータディレクトリ)」に示すディレクトリ内にあるファイルはこの限りではありません。

表 3.3 電源を切っても保持されるディレクトリ(ユーザーデータディレクトリ)

ディレクトリ	備考
/var/app/volumes	SWUpdate の最中や後も保持され続けます。ロールバックが発生しても、アップデート前の状態には戻りません。ログやデータベースなど、アプリケーションが動作中に作成し続けるようなデータはこのディレクトリに保存してください。
/var/app/rollback/volumes	SWUpdate の最中や後も保持され続けます。ロールバックが発生すると、アップデート前の状態に戻ります。コンフィグファイルなど、アプリケーションのバージョンに追従してアップデートするようなデータはこのディレクトリに保存してください。



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 3.48 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。

3.3.5. インストールディスクについて

インストールディスクは、Armadillo の eMMC の中身をまとめて書き換えることのできる microSD カードを指します。インストールディスクは、インストールディスクイメージを microSD カードに書き込むことで作成できます。

インストールディスクには以下の 2 つの種類があります。

- ・ 初期化インストールディスク

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>] にある標準イメージです。Armadillo を初期化する際に使用します。

- ・ 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。

量産時など、特定の Armadillo を複製する際に使用されます。詳しくは、「4. 量産編」で説明します。

3.3.5.1. 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-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 ATDE にダウンロードしてください。

「6.21. 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 --board a600
: (省略)
[ATDE ~/build-rootfs-[VERSION]]$ ls baseos-600*img
baseos-600-[VERSION].img
[ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_image.sh --board a600 ¥
```

```
--boot ~/imx-boot-[VERSION]/imx-boot_armadillo_600 ¥
--installer ./baseos-600-[VERSION].img
```

コマンドの実行が完了すると、baseos-600-[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。



標準のインストールディスクイメージは、シリアルコンソールの出力に CON9 を使用します。オプションモジュール等を使用しシリアルコンソールの出力を CON3 へ変更したい場合は、「Armadillo Base OS インストールディスクイメージ(UART3 コンソール)」を使用するか、build_image.sh 実行時のオプションに --uboot-env console=ttyxc2,115200 を追加してください。

ABOS 3.19.1-at.3 以降のバージョンで abos-ctrl make-installer を実行してインストールディスクイメージを作成した場合は、現在の設定を自動的に取り込みます。

インストールディスクの出力だけではなく、インストールされた後にも適用されます。

1. ATDE に microSD カードを接続します。詳しくは「3.1.2.7. 取り外し可能デバイスの使用」を参考にしてください。
2. 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)
```

3. 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
```

4. 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。



インストールディスク作成時に SBOM を作成する場合は `build_image.sh` の引数に `--sbom` を渡してください。ライセンス情報等を記載するためのコンフィグファイルはデフォルトは `baseos_sbom.yaml` となっています。コンフィグファイルを変更する場合は `--sbom-config <config>` に引数を入れてください。また、コンテナイメージを含める場合等に外部の SBOM を入れる必要がある場合は `--sbom-external <sbom>` に引数を入れてください。SBOM のライセンス情報やコンフィグファイルの設定方法については「6.22.3. ビルドしたルートファイルシステムの SBOM を作成する」をご覧ください。

3.3.5.2. インストールディスクを使用する

1. JP1 と JP2 をジャンパーピンでショート(SD ブートに設定)させ、microSD カードを microSD スロットに挿入します。
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(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 JP2 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。

3.4. ハードウェアの設計

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。

3.4.1. 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。

3.4.2. 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

3.4.3. 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)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には、以下の対策が効果的です。

- ・ 通信対向機との GND 接続を強化する
- ・ シールド付きのケーブルを使用する

3.4.4. 放熱

Armadillo Base OS には標準で、CPU や SoC の温度をプロファイリングするソフトウェアが搭載されているので、温度設計にお役立てください。詳細は「6.15. 動作中の Armadillo の温度を測定する」を参照してください。

3.4.5. 周辺装置との接続

Armadillo-640 開発セットと周辺装置の接続例を「図 3.49. Armadillo-640 の接続例」に示します。

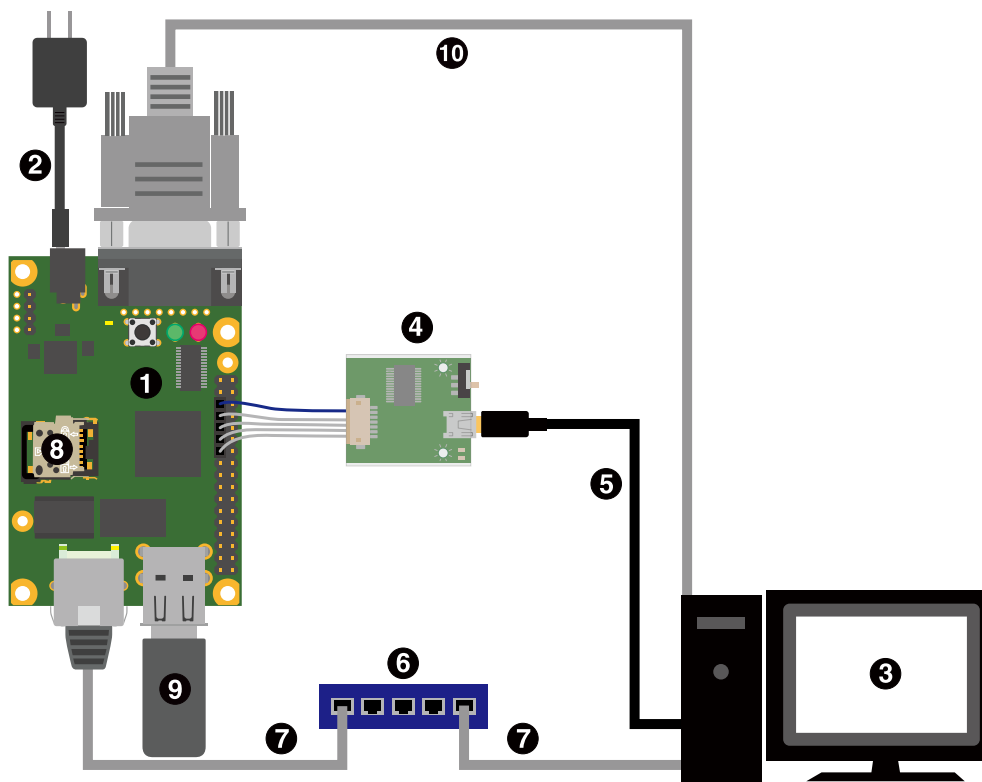


図 3.49 Armadillo-640 の接続例

- 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

3.4.6. 拡張ボードの設計

Armadillo-640 の拡張インターフェース(CON8、CON9、CON14)には、複数の機能をもった信号線が接続されており、様々な機能拡張が可能です。

拡張インターフェースに接続する基板を設計する際の制限事項について、説明します。

3.4.6.1. ピンアサイン

Armadillo-640 では、「表 2.2. 仕様」の拡張インターフェースの欄にあるとおりの機能が拡張できます。ただし、ここに記載の拡張数は、優先的に機能を割り当てた場合の最大数ですので、必要な機能がすべて実現できるかは、『Armadillo-640 マルチプレクス表』で検討する必要があります。

マルチプレクス表では、各ピンに割り当て可能な機能の他に、リセット後の信号状態、プルアップ/ダウン抵抗の有無等の情報を確認することができます。

各機能の詳細な仕様が必要な場合は、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』をご確認ください。Armadillo-640 固有の情報を除いて、回路設計に必要な情報はこれらのマニュアルに、すべて記載されています。検索しやすいように、マルチプレクス表や「3.4.6.2. CON8、CON9、CON14(拡張インターフェース)」に i.MX 6ULL のピン名やコントローラー名を記載しておりますので、是非ご活用ください。



Armadillo-640 マルチプレクス表は「Armadillo-640 マルチプレクス表」
[<https://armadillo.atmark-techno.com/resources/documents/armadillo-640/manual-multiplex>]からダウンロードしてください。

3.4.6.2. 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。



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 3.4 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 3.5 CON9 信号配列

ピン番号	ピン名	I/O	説明
1	GPIO1_I022	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_I023	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_I017	In/Out	拡張入出力、i.MX6ULL の UART1_RX_DATA ピンに接続
4	GPIO1_I031	In/Out	拡張入出力、i.MX6ULL の UART5_RX_DATA ピンに接続
5	GPIO1_I016	In/Out	拡張入出力、i.MX6ULL の UART1_TX_DATA ピンに接続
6	GPIO1_I030	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_SNVS_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_SNVS_3V)されています。
13	GPIO3_I023	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_I024	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_I025	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_I026	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_I027	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_I028	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

ピン番号	ピン名	I/O	説明
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、基板上のプルダウン抵抗で 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 3.6 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

3.4.7. 電氣的仕様

3.4.7.1. 絶対最大定格

表 3.7 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。

3.4.7.2. 推奨動作条件

表 3.8 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 [a]	-	3.3	V	Topr=+25°C
使用温度範囲	Ta	-20	25	70	V	結露なきこと

[a]S/N: 009C00010001~009C00060102 の Armadillo-640 では、下限電圧 2.95V です。

3.4.7.3. 入出インターフェースの電氣的仕様

表 3.9 入出インターフェース(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 3.10 入出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[a]	VIH	0.7×OVDD	OVDD	V	-
ローレベル入力電圧[a]	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	-

[a]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。

3.4.7.4. 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。

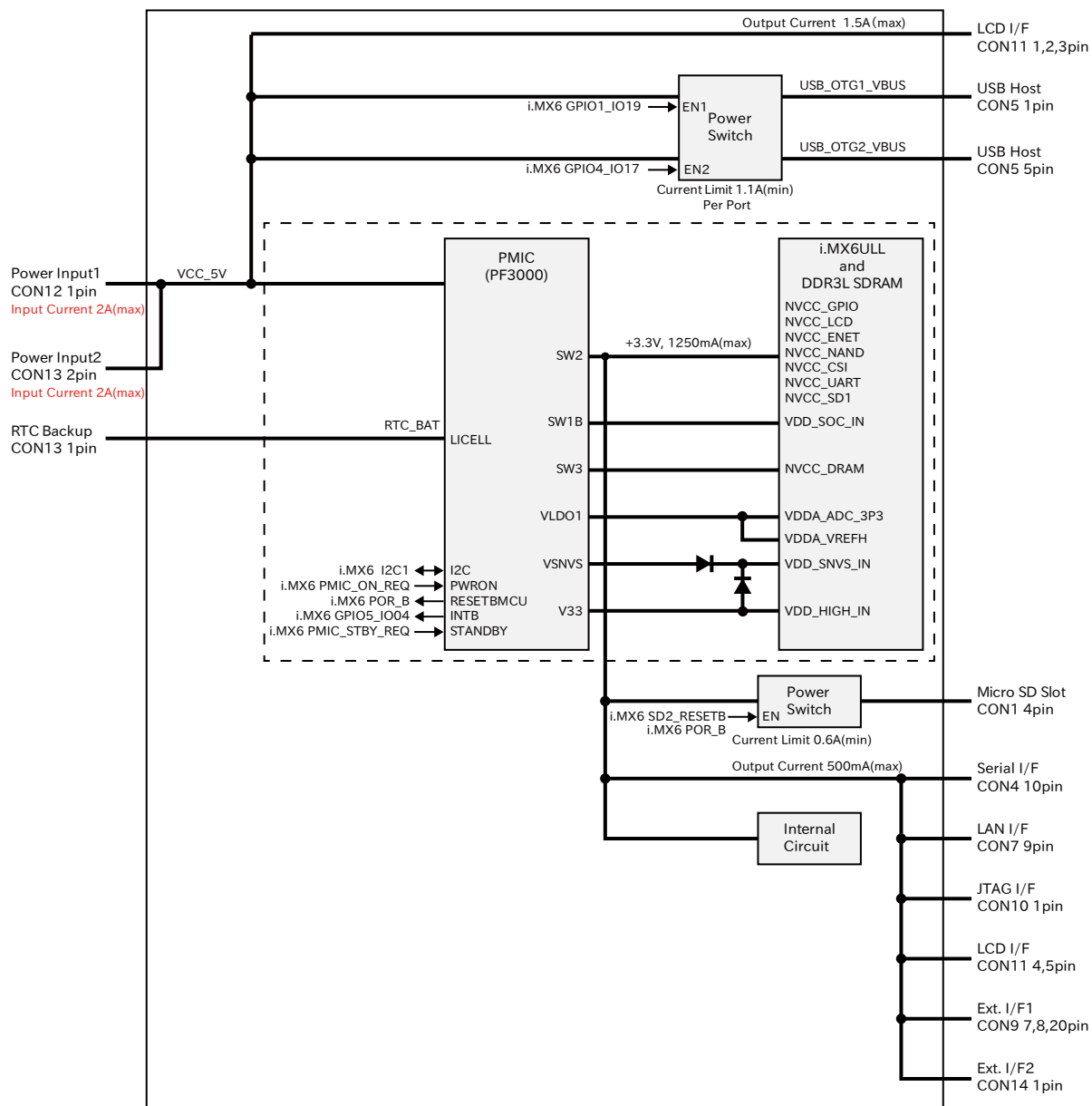


図 3.50 電源回路の構成

電源シーケンスは次のとおりです。

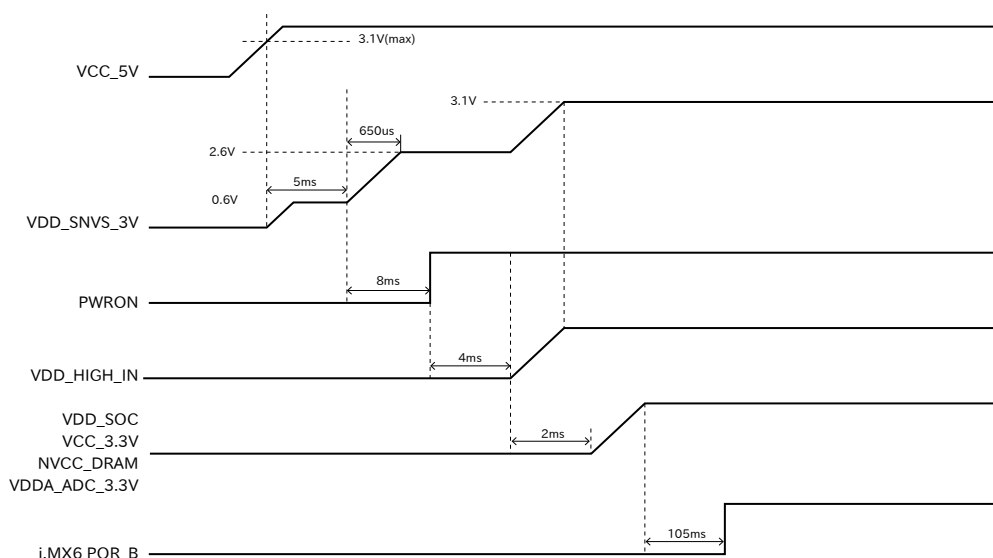


図 3.51 電源シーケンス

3.4.7.5. 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
- ・ 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。

・ 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。

・ 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。

3.4.8. 形状図

3.4.8.1. 基板形状図

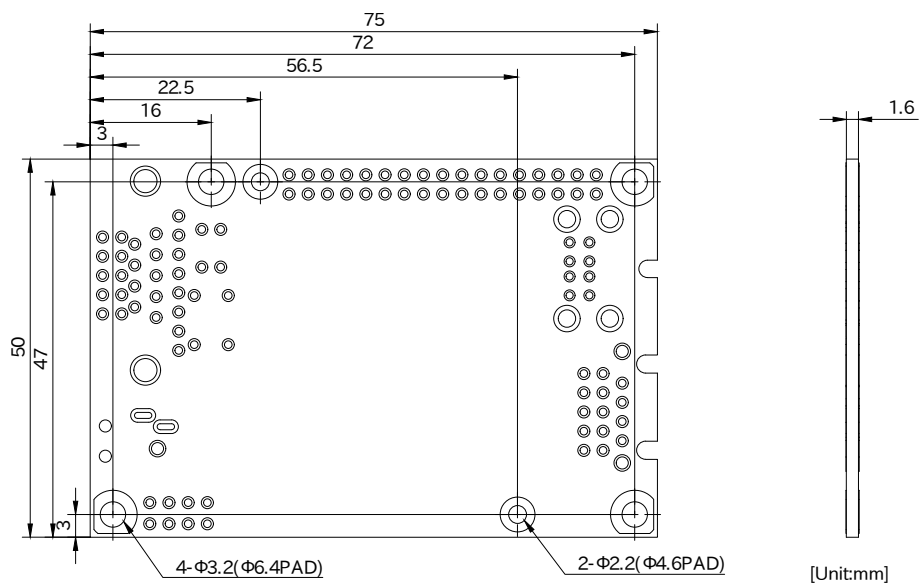


図 3.52 基板形状および固定穴寸法

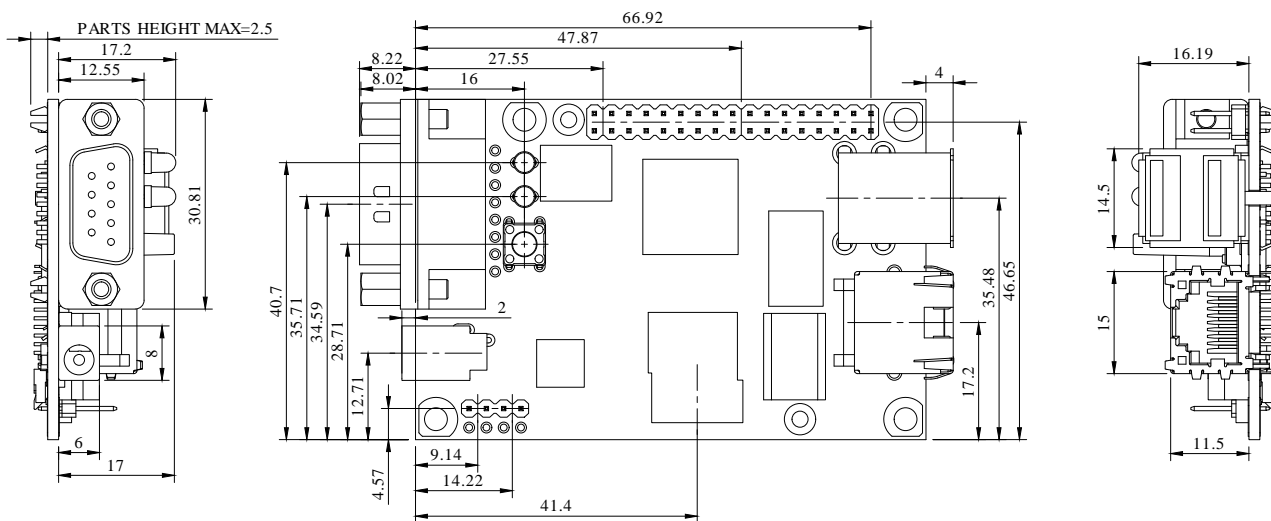


図 3.53 コネクタ中心寸法(A 面側)

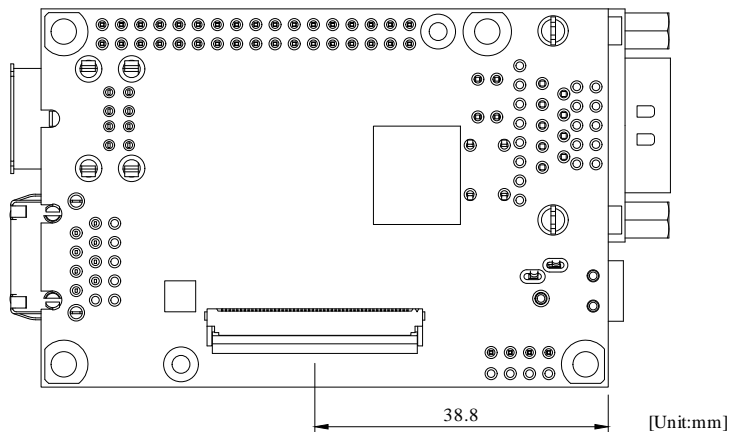


図 3.54 コネクタ中心寸法(B 面側)

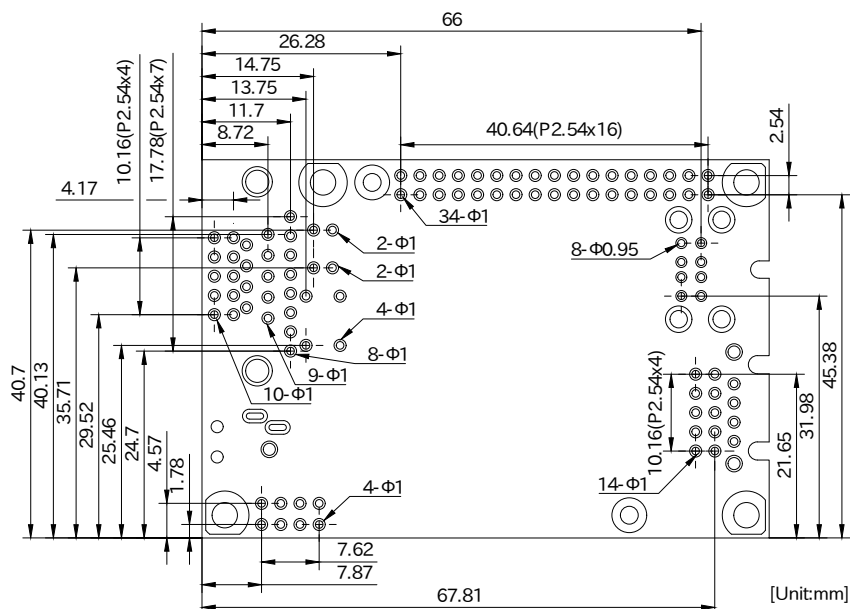


図 3.55 コネクタ穴寸法



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。

3.4.9. オプション品

Armadillo-640 のオプション品については、「6.28. オプション品」を参照してください。

3.5. 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。

3.5.1. Linux カーネルソースコードの取得

at-dtweb を使用するためには、予め Linux カーネルのソースコードを用意しておく必要があります。



at-dtweb が必要とするのは Linux カーネルソースコード内の dts(Device Tree Source)ファイルと Makefile であり、Linux カーネルイメージのビルドをする必要はありません。そのため、ここでは Linux カーネルのビルドは行いません。

Linux カーネルのビルド手順については、「6.21.2. Linux カーネルをビルドする」を参照してください。

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]
```

次のコマンドを実行して、デフォルトコンフィギュレーションを適用しておきます。

```
[ATDE ~/linux-[VERSION]]$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- armadillo-640_defconfig
```

3.5.2. 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
```

3.5.3. 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。

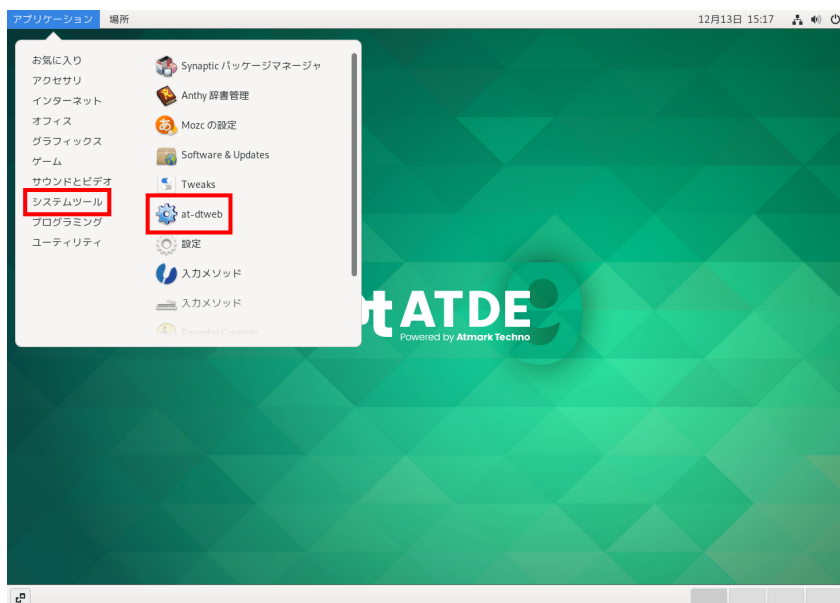


図 3.56 at-dtweb の起動開始

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。

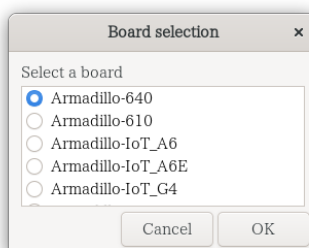


図 3.57 ボード選択画面

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。

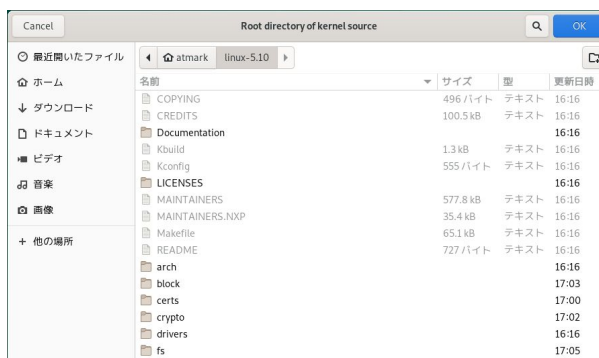


図 3.58 Linux カーネルディレクトリ選択画面

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。

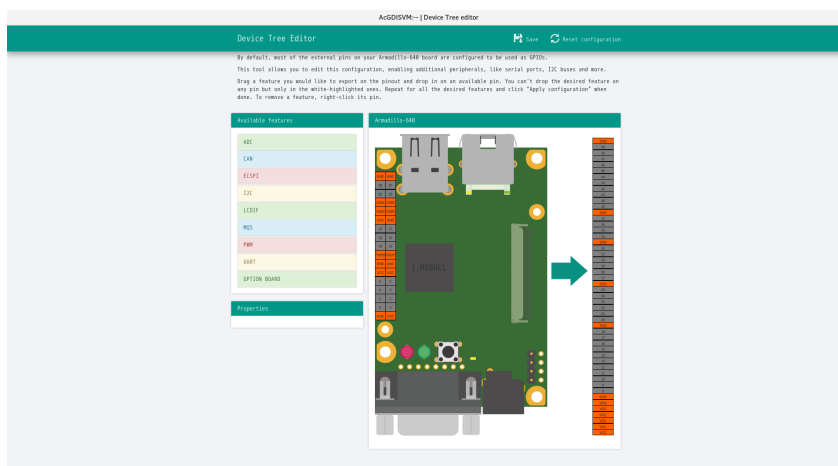



図 3.59 at-dtweb 起動画面

3.5.4. Device Tree をカスタマイズ

3.5.4.1. 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。

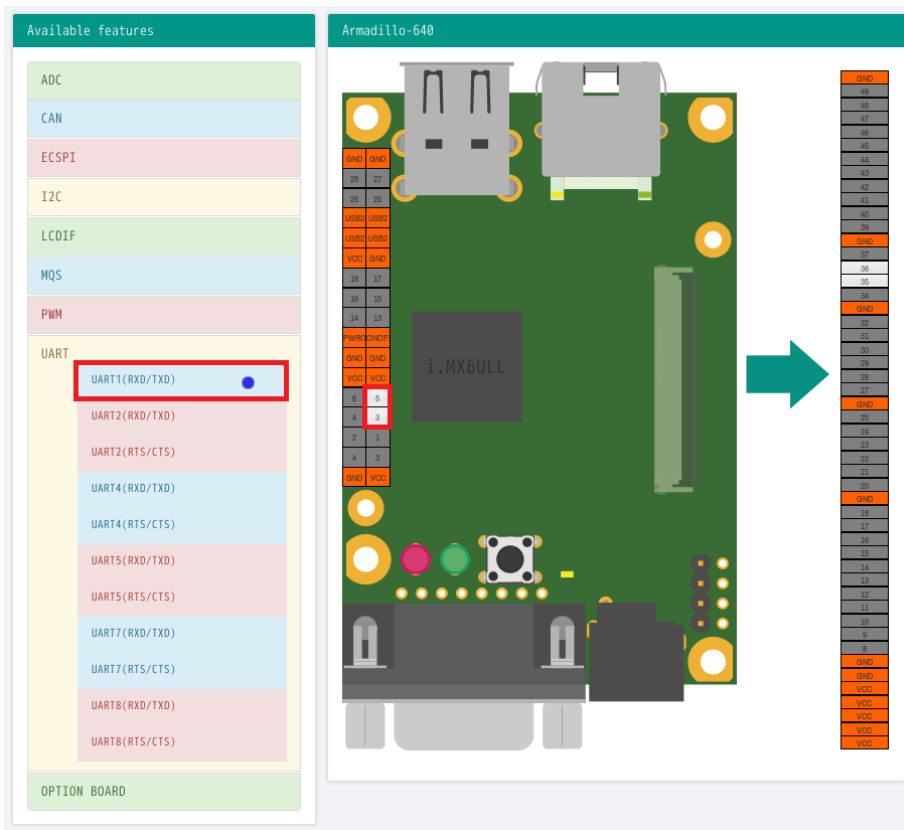


図 3.60 UART1 (RXD/TXD)のドラッグ

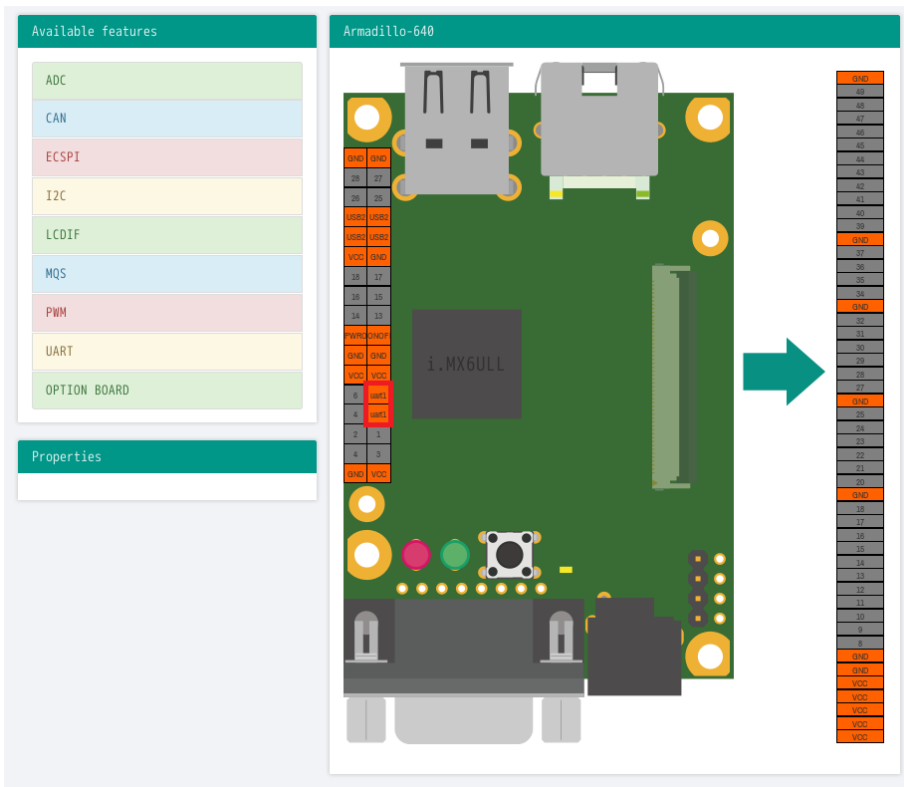


図 3.61 CON9 3/5 ピンへのドロップ

3.5.4.2. 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。

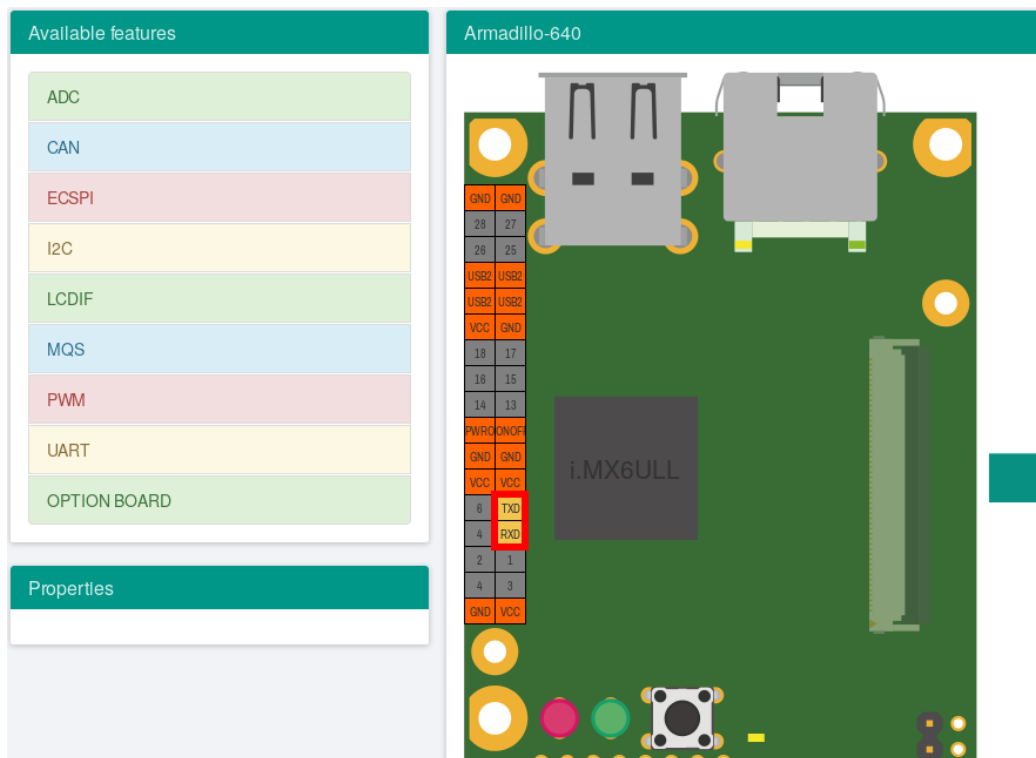


図 3.62 信号名の確認



再度ピンを左クリックすると機能名の表示に戻ります。

3.5.4.3. 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。

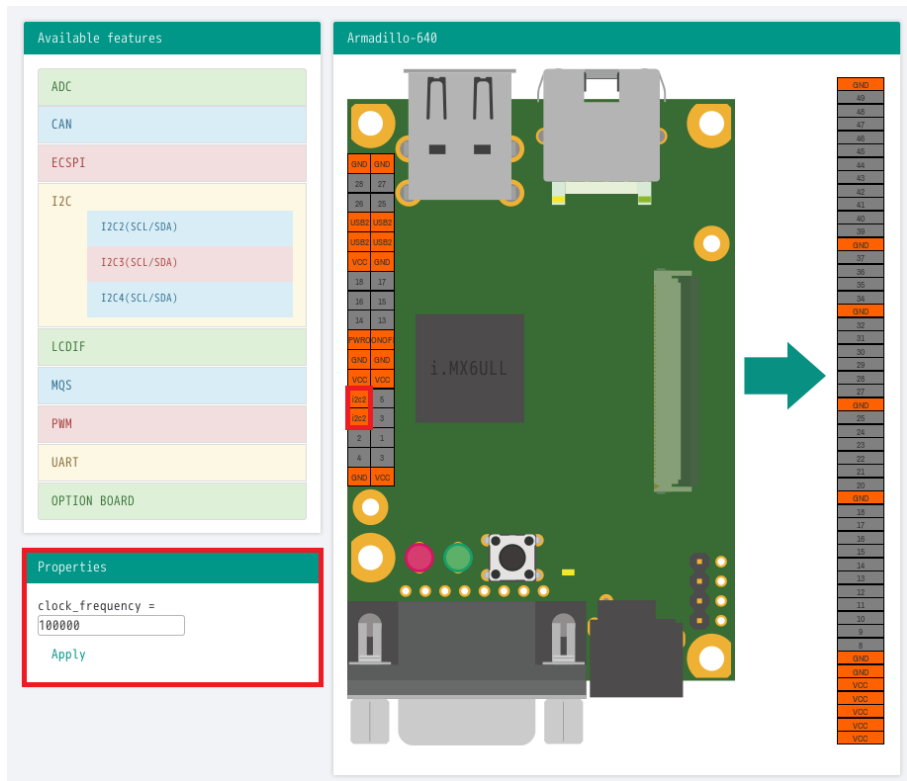


図 3.63 プロパティの設定

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。

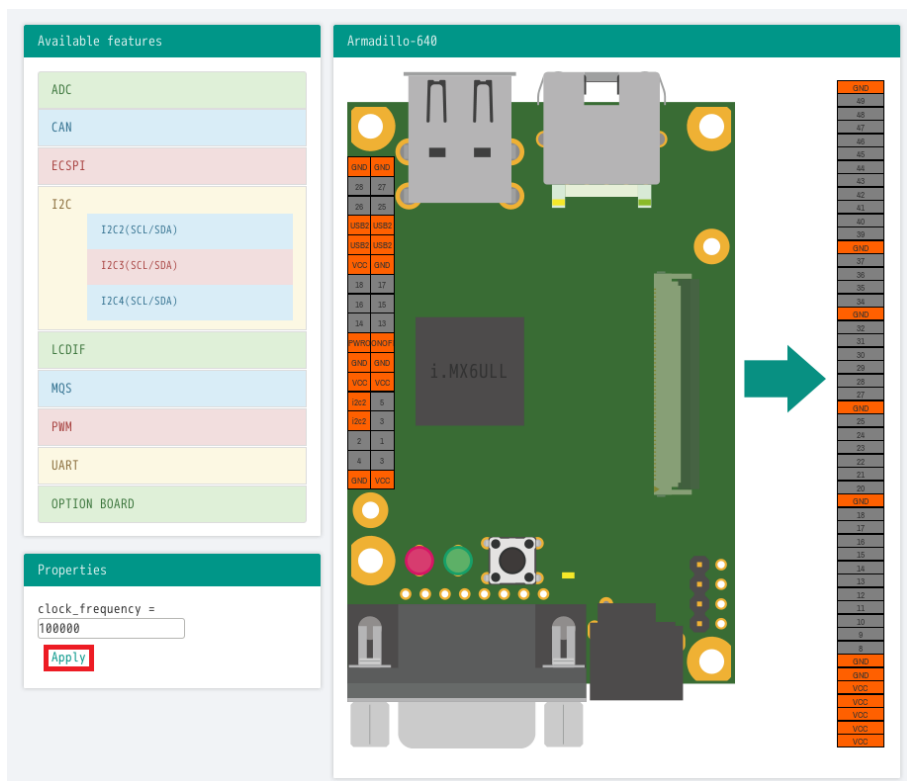


図 3.64 プロパティの保存

3.5.4.4. 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。

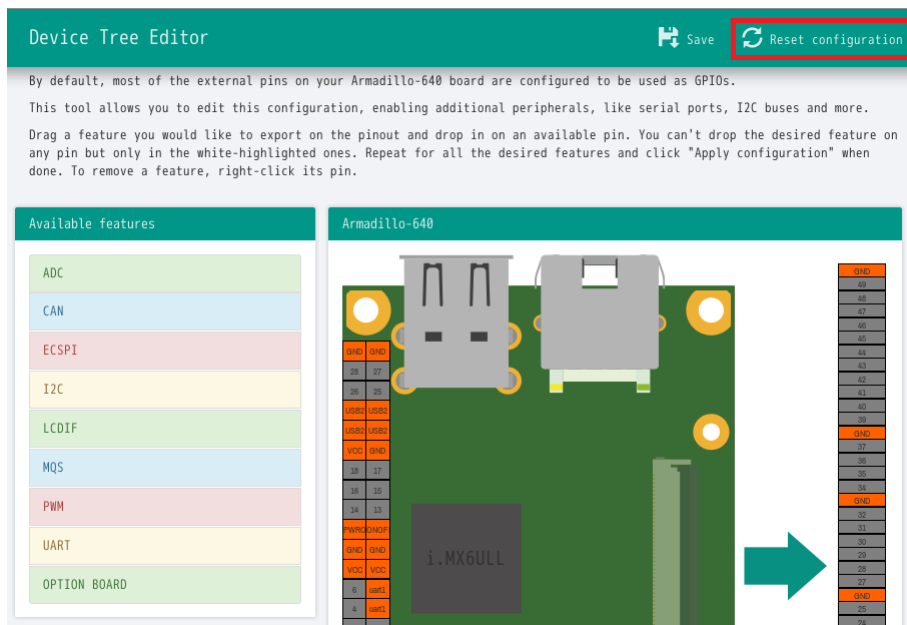


図 3.65 全ての機能の削除

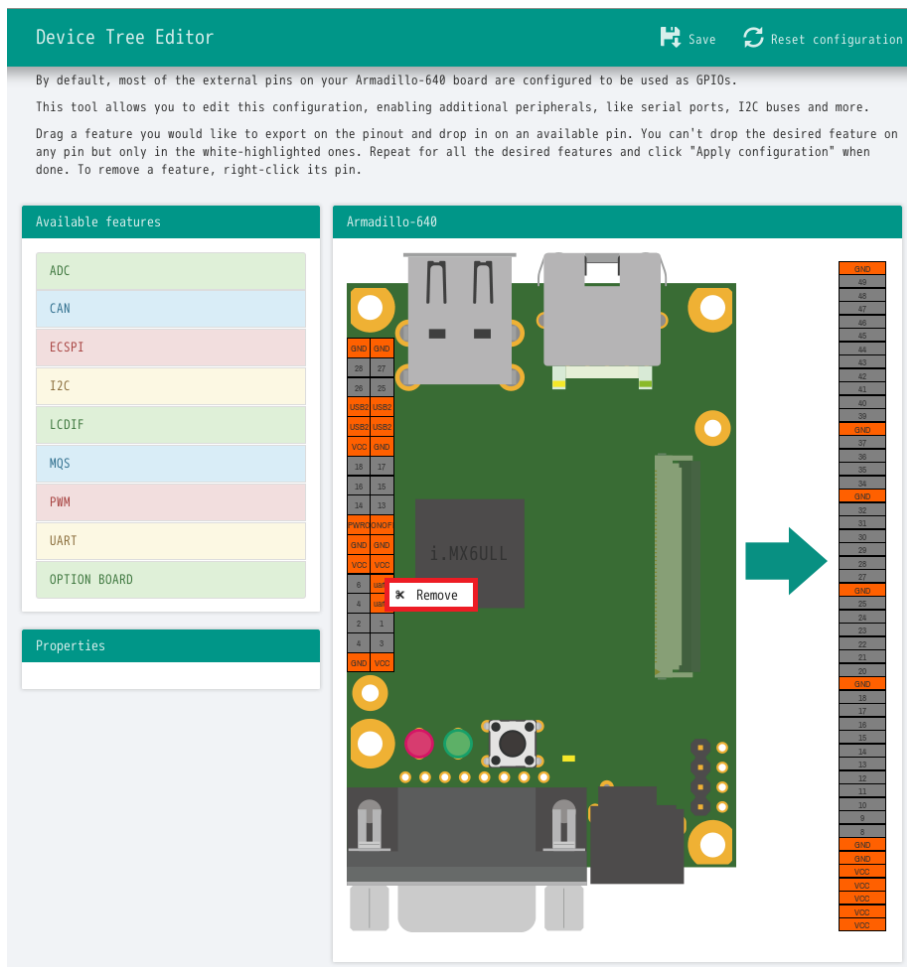


図 3.66 UART1 (RXD/TXD) の削除

3.5.4.5. 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。

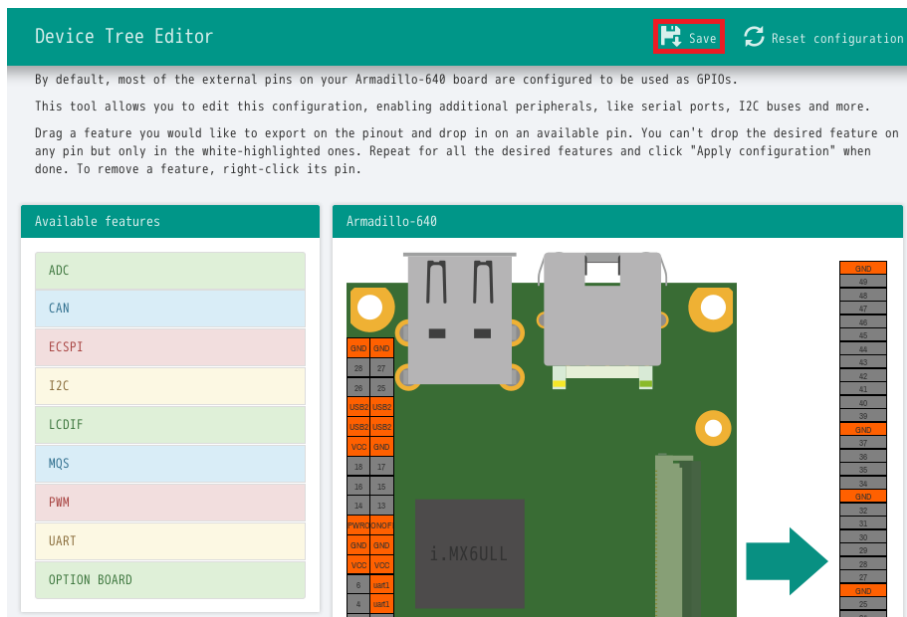


図 3.67 DTS/DTB の生成

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。

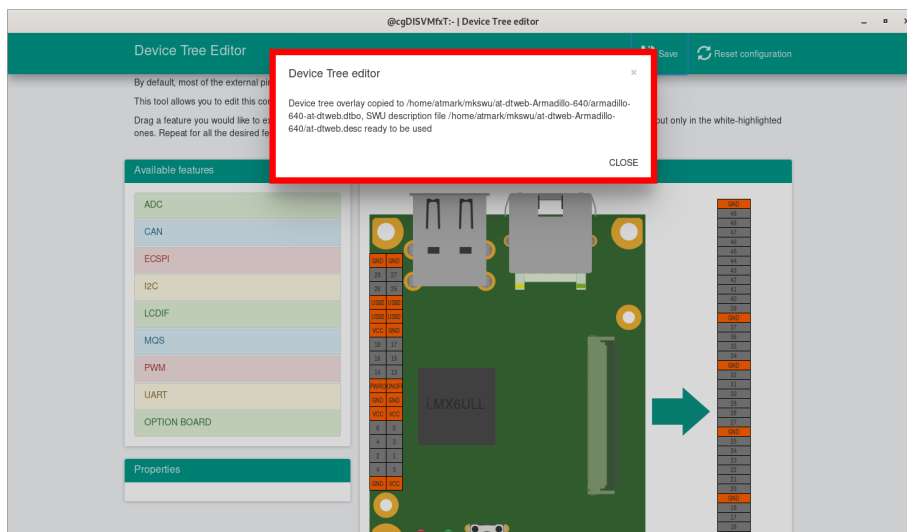


図 3.68 dtbo/desc の生成完了

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 ❶
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「3.3.3. アップデート機能について」を参照してください。

3.5.5. 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-con9-thread-lwb5plus.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-con9-thread-lwb5plus.dtbo
: (省略)
```

図 3.69 /boot/overlays.txt の変更例


- ❶ /boot/overlays.txt ファイルに「armadillo-640-con9-thread-lwb5plus.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「3.5.5. 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。

3.5.5.1. 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-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、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-600-button-enter.dtbo**: SW1 の動作を Debian 版と同じにします。SW1 の押下を ENTER キーのリリースに割り当てます。



at-dtweb で作成した armadillo-640-at-dtweb.dtbo と同時にこれらの dtbo を使用する場合、at-dtweb で設定した内容との間で設定の競合が発生し、正しく動作しない場合があります。

3.6. インターフェースの使用方法和デバイスの接続方法

Armadillo を用いた開発に入る前に、開発するシステムに接続する必要のある周辺デバイスをこのタイミングで接続しておきます。

「図 3.70. Armadillo-640 のインターフェース」に Armadillo-640 の各インターフェースの位置を、「表 3.11. Armadillo-640 インターフェース一覧」に各インターフェースの概要を示します。

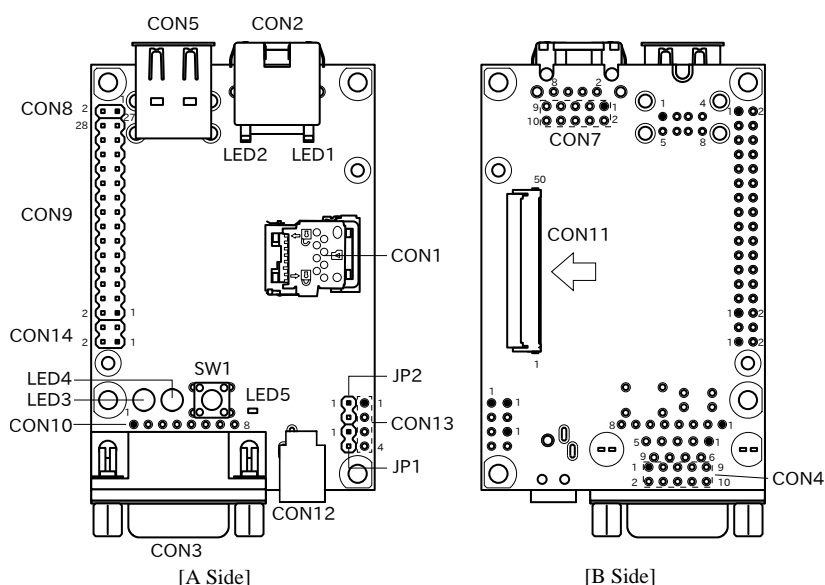


図 3.70 Armadillo-640 のインターフェース

表 3.11 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		XM2C-0942-132L	OMRON
CON4	シリアルインターフェース	A1-10PA-2.54DSA(71)	HIROSE ELECTRIC
CON5		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	HEC3600-016110	HOSIDEN
CON13		A2-4PA-2.54DSA(71)	HIROSE ELECTRIC

部品番号	インターフェース名	型番	メーカー
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。

以下では、各デバイスの接続方法、仕様及び使用方法について紹介していきます。

3.6.1. SD カードを使用する

以下の説明では、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。

3.6.1.1. ハードウェア仕様

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

インターフェース仕様 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 3.12 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

3.6.1.2. 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。

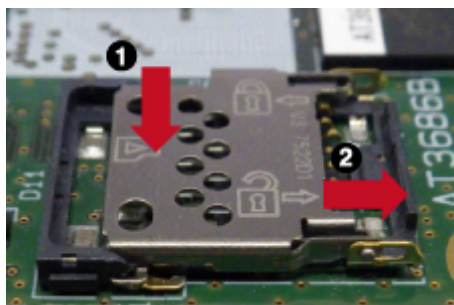


図 3.71 カバーのロックを解除する

2. カバーを開けます。

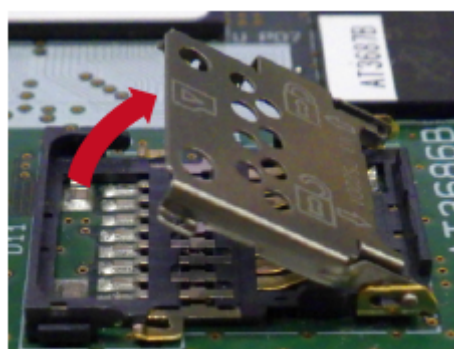


図 3.72 カバーを開ける



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 3.73 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 3.74 カードマークの確認

4. カバーを閉めます。

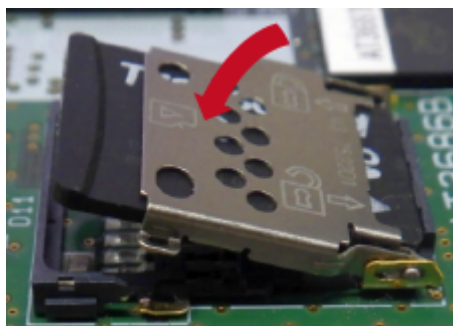


図 3.75 カバーを閉める

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。

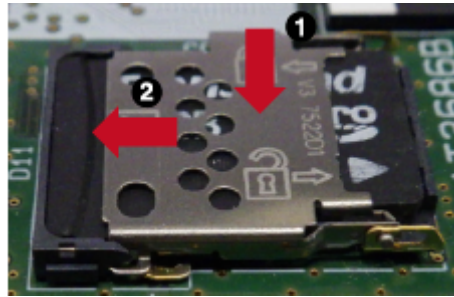


図 3.76 カバーをロックする



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。

3.6.1.3. ソフトウェア仕様

デバイスファ
イル ・ /dev/mmcblk1

3.6.1.4. 使用方法

ここでは、sd_example という名称の alpine ベースのコンテナを作成し、その中で microSD カードを使用します。必要なコンテナイメージは予め podman pull している前提で説明します。

CON1 に microSD カードを挿入してください。

/etc/atmark/containers/sd_example.conf というファイルを以下の内容で作成します。

```
set_image docker.io/alpine
add_hotplugs mmc ❶
add_args --cap-add=SYS_ADMIN ❷
set_command sleep infinity
```

- ❶ add_hotplugs に mmc を指定することで、コンテナ内で microSD カードをホットプラグで認識します
- ❷ コンテナ内で microSD カードをマウントするための権限を与えます

コンテナを起動し、コンテナの中に入ります。

```
[armadillo]# podman_start sd_example
Starting 'sd_example'
1d93ecff872276834e3c117861f610a9c6716c06eb95623fd56aa6681ae021d4

[armadillo]# podman exec -it sd_example sh
[container]#
```

コンテナ内で microSD カードは、 /dev/mmcblk1 として認識されますので /mnt にマウントします。

```
[container]# mount /dev/mmcblk1p1 /mnt
```

ストレージの使用方法については、「6.13. ストレージの操作」もあわせて参照してください。

3.6.2. Ethernet を使用する

3.6.2.1. ハードウェア仕様

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 | <ul style="list-style-type: none"> ・ 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 |
|----|--|

インターフェース仕様(CON2、CON7)	<p>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。</p>
-----------------------	--

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 3.13 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 3.14 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。

インターフェース仕様(LED1、LED2)

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 3.15 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

3.6.2.2. ソフトウェア仕様

ネットワークデ
バイス ・ eth0

3.6.2.3. 使用方法

ネットワークの設定方法については「3.8. ネットワーク設定」を参照してください。

3.6.3. UART を使用する

3.6.3.1. ハードウェア仕様

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。CON3、CON4 や拡張インターフェース(CON9、CON14)及び、LCD 拡張インターフェース(CON11)に最大 6 ポート拡張可能です。拡張インターフェースの仕様については「3.4.6.2. CON8、CON9、CON14(拡張インターフェース)」を、LCD 拡張インターフェースの仕様については「3.6.15. LCD を使用する」参照してください。

- 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: 4Mbps



UART3 (CON3 / CON4)の最大ボーレートは 230.4kbps です。これは、RS-232C レベル変換 IC の最大ボーレートです。

インターフェース仕様(CON3、CON4) 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 3.16 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 3.17 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。

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。

3.6.3.2. ソフトウェア仕様

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

3.6.3.3. 使用方法

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/ttymx2
[armadillo ~]# podman_start serial_example
Starting 'serial_example'
3999f09d51253371cacffd68967c90fdd5250770888a82f59d7810b54fcc873e
```

図 3.77 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/ttymx2
/dev/ttymx2, Line 2, UART: undefined, Port: 0x0000, IRQ: 52
    Baud_base: 5000000, close_delay: 50, divisor: 0
    closing_wait: 3000
    Flags: spd_normal
```

図 3.78 setserial コマンドによるシリアルインターフェース設定の確認例

3.6.4. USB デバイスを使用する

3.6.4.1. ハードウェア仕様

- ・ 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きます。

機能 ・ Universal Serial Bus Specification Revision 2.0 準拠

- ・ Enhanced Host Controller Interface (EHCI) 準拠
- ・ 転送レート: USB2.0 High-Speed (480Mbps), Full-Speed (12Mbps), Low-Speed (1.5Mbps)

インターフェース仕様 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。

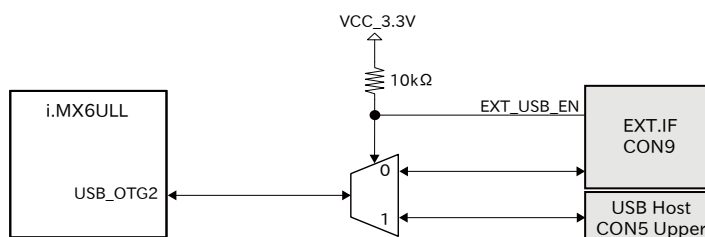


図 3.79 USB OTG2 の接続先の変更

下段に供給される電源(USB_OTG1_VBUS)は i.MX6ULL の UART1_RTS_B ピン(GPIO1_IO19)、上段に供給される電源(USB_OTG2_VBUS)は i.MX6ULL の CSI_MCLK ピン(GPIO4_IO17)で制御が可能で、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。

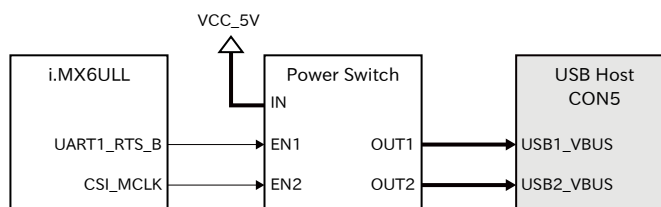


図 3.80 USB ホストインターフェースの電源制御

- ・ データ転送モード
 - ・ High Speed(480Mbps)
 - ・ Full Speed(12Mbps)
 - ・ Low Speed(1.5Mbps)

表 3.18 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

ピン番号	ピン名	I/O	説明
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)

3.6.4.2. ソフトウェア仕様

デバイスファイル ・ 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。

・ 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。

3.6.4.3. 使用方法

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 add_hotplugs に ttyUSB を設定する必要があります。この設定により、コンテナ起動後に USB シリアルデバイスを接続した場合でも正しく認識されます。以下は、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_hotplugs ttyUSB
[armadillo ~]# podman_start usb_example
Starting 'usb_example'
34cb0e60d6274ac1df87aed58a461bcf56d0c117c4d377af130605ea399e0950
```

図 3.81 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/serial/by-id/usb-067b_2303-if00-port0
/dev/serial/by-id/usb-067b_2303-if00-port0, Line 4, UART: 16654, Port: 0x0000, IRQ: 0
Baud_base: 460800, close_delay: 0, divisor: 0
closing_wait: infinite
Flags: spd_normal
```

図 3.82 setserial コマンドによる USB シリアルデバイス設定の確認例

コンテナ内からのデバイスの指定には /dev/ttyUSB N を使用することもできますが、デバイスを接続するタイミングによっては N の値が変わる可能性があります。このため上記の例のように /dev/serial/by-id/ 下にあるファイルを指定することで確実に目的のデバイスを使用することができます。

・ 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 `add_hotplugs` に `video4linux` を設定する必要があります。この設定により、コンテナ起動後に USB カメラを接続した場合でも正しく認識されます。以下は、`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_hotplugs video4linux
[armadillo ~]# podman_start usbcam_example
Starting 'usbcam_example'
ffe06090b45826cc0b1c7710e9e850ba9521d36b70de4288d0dfe1fe91a35632
[armadillo ~]# podman exec -it usbcam_example sh
[container ~]# ls /dev/v4l/by-id/usb-046d_HD_Pro_Webcam_C920_78DA8CAF-video-index0
/dev/v4l/by-id/usb-046d_HD_Pro_Webcam_C920_78DA8CAF-video-index0
```

図 3.83 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。

コンテナ内からのデバイスの指定には `/dev/videoN` を使用することもできますが、デバイスを接続するタイミングによっては `N` の値が変わる可能性があります。このため上記の例のように `/dev/v4l/by-id/` 下にあるファイルを指定することで確実に目的のデバイスを使用することができます。

・ 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 3.84 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 3.85 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 3.86 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 `add_hotplugs` に `sd` を設定する必要があります。この設定により、コンテナ起動後に USB メモリを接続した場合でも正しく認識されます。加えて、コンテナ内からマウントするためには適切な権限も設定する必要があります。以下は、`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_hotplugs sd
[armadillo ~]# podman start usbmem_example
Starting 'usbmem_example'
387a2256530e9b35b5361ca681a99fba8f46d78b6a6cb8ecd60096246b9198a8
```

図 3.87 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/disk/by-label/[MYUSBMEMORY] /mnt ❶
[container ~]# ls /mnt
sample.txt
[container ~]# cat /mnt/sample.txt
test
```

図 3.88 コンテナ内から USB メモリをマウントする例

- ❶ [MYUSBMEMORY] の部分は USB メモリに設定しているラベルに置き換えてください。

コンテナ内からマウントするデバイスの指定には `/dev/sdN` を使用することもできますが、他にもストレージデバイスを接続している場合などには `N` の値が変わることがあります。このため、USB

メモリにラベルを設定している場合は、上記の例のように /dev/disk/by-label/ 下にあるラベルと同名のファイルを指定することで確実に目的のデバイスを使用することができます。

3.6.5. WLAN を使用する

3.6.5.1. ハードウェア仕様

「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、Ezurio 製 Sterling LWB5+ が搭載されています。Sterling LWB5+ の WLAN は、USB2422 を介して「3.6.4. USB デバイスを使用する」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)

3.6.5.2. ソフトウェア仕様

- ネットワークデバイス
- ・ wlan0 (STA)

3.6.5.3. 使用方法

「6.28.6. 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、Ezurio 製 Sterling LWB5+ を用いた使用方法を紹介します。

まずはじめに、コンソールを CON3 に移動します。「図 6.187. 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 3.89 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/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、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 3.90 DT overlay の設定(WLAN)

DT overlay の設定後、Armadillo の電源を切り、「6.28.6. Armadillo-600 シリーズ WLAN コンボ、BT/TH オプションモジュール」を参考に Armadillo-600 シリーズ WLAN コンボオプションモジュールを組み付けてください。

その後のネットワークの設定方法については「3.8. ネットワーク設定」を参照してください。

3.6.6. BT デバイスを使用する

3.6.6.1. ハードウェア仕様

「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、Ezurio 製 Sterling LWB5+ が搭載されています。Sterling LWB5+ の WLAN は、USB2422 を介して「3.6.4. USB デバイスを使用する」に示す OTG2 に接続されています。

3.6.6.2. ソフトウェア仕様

デバイス ・ hci0
ス



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
```

3.6.6.3. BT を使用する準備

「6.28.6. 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 を用いた使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。

まずはじめに、コンソールを CON3 に移動します。「図 6.187. オプションモジュールの CON2 を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 3.91 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/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、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 3.92 DT overlay の設定(BT/TH)

3.6.6.4. 使用方法

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 3.93 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 3.94 Sterling LWB5+ の BT 機能を無効化する

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 3.95. Bluetooth を扱うコンテナの作成例」に、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 3.95 Bluetooth を扱うコンテナの作成例

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 ~]# mkdir /run/dbus
[container ~]# dbus-daemon --system
[container ~]# /usr/lib/bluetooth/bluetoothd &
```

図 3.96 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 registered
[..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 3.97 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。

3.6.7. Thread デバイスを扱う

3.6.7.1. Thread を使用する準備

「6.28.6. 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 を用いた使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。

準備の手順は「3.6.6.3. BT を使用する準備」と同一なので、そちらを参照してください。

3.6.7.2. 使用方法

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 3.98 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。



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。

以下は、EYSKBNZWB のデバイスファイルである `/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 3.99 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。

3.6.8. 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
set_command sleep infinity
add_devices /dev/snd
[armadillo ~]# podman_start snd_example
Starting 'snd_example'
b921856b504e9f0a3de2532485d7bd9adb1ff63c2e10bfdaccd1153fd36a3c1d
```

図 3.100 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 3.101 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。

3.6.9. GPIO を制御する

3.6.9.1. ハードウェア仕様

Armadillo-640 の GPIO は、i.MX6ULL の GPIO(General Purpose Input/Output)を利用しています。

拡張インターフェース(CON8、CON9、CON14)や、LCD 拡張インターフェース(CON11)などで GPIO を最大 61 ポート拡張することが可能です。拡張インターフェースの仕様については「3.4.6.2. CON8、CON9、CON14(拡張インターフェース)」を、LCD 拡張インターフェースの仕様については「3.6.15. LCD を使用する」参照してください。

3.6.9.2. ソフトウェア仕様

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。

3.6.9.3. 使用方法

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 3.102 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 ②
[container ~]# gpiowrite gpiochip0 22=0 ③
```

図 3.103 コンテナ内からコマンドで GPIO を操作する例

- ① GPIO 番号 22 の値を取得します。
- ② 取得した値を表示します。
- ③ 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 3.104 `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 3.105 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 3.19 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 3.19. 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「3.5. 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せん。

3.6.10. I2C デバイスを使用する

3.6.10.1. ハードウェア仕様

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。

拡張インターフェース(CON9、CON14)や、LCD 拡張インターフェース(CON11)などで I2C を最大 3 ポート拡張することが可能です。拡張インターフェースの仕様については「3.4.6.2. CON8、CON9、CON14(拡張インターフェース)」を、LCD 拡張インターフェースの仕様については「3.6.15. LCD を使用する」参照してください。

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 3.20 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 ^[a]
5(I2C-GPIO1)	0x32	NR3225SA ^[b]

^[a]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。

^[b]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。

3.6.10.2. ソフトウェア仕様

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 ・ 最大データ転送レート: 384kbps

デバイスファ ・ /dev/i2c-0 (I2C1) ^[3]
イル

 ・ /dev/i2c-4 (I2C-GPIO)

3.6.10.3. 使用方法

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 3.106 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:  -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

^[3]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。

```
60: ---
70: ---
```

図 3.107 i2cdetect コマンドによる確認例

3.6.11. SPI デバイスを使用する

3.6.11.1. ハードウェア仕様

拡張インターフェース(CON9、CON14)や、LCD 拡張インターフェース(CON11)などで SPI を最大 4 ポート拡張することが可能です。拡張インターフェースの仕様については「3.4.6.2. CON8、CON9、CON14(拡張インターフェース)」を、LCD 拡張インターフェースの仕様については「3.6.15. LCD を使用する」参照してください。

3.6.11.2. 使用方法

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 3.108 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 3.109 spi-config コマンドによる確認例

3.6.12. CAN デバイスを使用する

3.6.12.1. ハードウェア仕様

拡張インターフェース(CON9)や、LCD 拡張インターフェース(CON11)などで CAN を最大 2 ポート拡張することが可能です。拡張インターフェースの仕様については「3.4.6.2. CON8、CON9、CON14(拡張インターフェース)」を、LCD 拡張インターフェースの仕様については「3.6.15. LCD を使用する」参照してください。

3.6.12.2. 使用方法

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'
73e7dbce86e84eef337bbc5c580a747948b94b87015bb34143da341b8301c16a
```

図 3.110 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 3.111 CAN の設定例

- ❶ 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。
- ❷ CAN の通信速度を 125000 kbps に設定します。
- ❸ can0 インターフェースを起動します。
- ❹ can0 インターフェースの現在の使用状況を表示します。

3.6.13. PWM を使用する

3.6.13.1. ハードウェア仕様

拡張インターフェース(CON9)や、LCD 拡張インターフェース(CON11)などで I2C を最大 8 ポート拡張することが可能です。拡張インターフェースの仕様については「3.4.6.2. CON8、CON9、CON14(拡張インターフェース)」を、LCD 拡張インターフェースの仕様については「3.6.15. LCD を使用する」参照してください。

3.6.13.2. 使用方法

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 3.112 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 ❶
[container ~]# echo 1000000000 > /sys/class/pwm/pwmchip0/pwm0/period ❷
[container ~]# echo 500000000 > /sys/class/pwm/pwmchip0/pwm0/duty_cycle ❸
[container ~]# echo 1 > /sys/class/pwm/pwmchip0/pwm0/enable ❹
```

図 3.113 PWM の動作設定例

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

3.6.14. JTAG デバッガを使用する

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 3.114. 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。

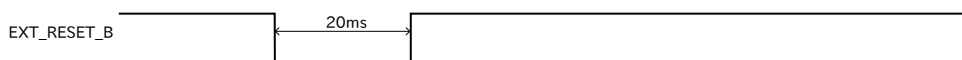


図 3.114 リセットシーケンス



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、「Armadillo-640 マルチプレクス表」 [<https://armadillo.atmark-techno.com/resources/documents/armadillo-640/manual-multiplex>] 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 3.21 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、オーブンドレイン入力
8	GND	Power	電源(GND)

3.6.15. LCD を使用する

3.6.15.1. ハードウェア仕様

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。

インターフェース仕様 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、「Armadillo-640 マルチプレクス表」 [<https://armadillo.atmark-techno.com/resources/documents/armadillo-640/manual-multiplex>] 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 3.22 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)
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。

ピン番号	ピン名	I/O	説明
25	LCD_DATA 11	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_DATA 12	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_DATA 13	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_DATA 14	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_DATA 15	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_DATA 16	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_DATA 17	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_IO1 8	In/Out	拡張入出力、i.MX6ULL の CSI_PIXCLK ピンに接続
40	GPIO4_IO2 1	In/Out	拡張入出力、i.MX6ULL の CSI_DATA00 ピンに接続
41	GPIO4_IO2 4	In/Out	拡張入出力、i.MX6ULL の CSI_DATA03 ピンに接続
42	GPIO4_IO2 2	In/Out	拡張入出力、i.MX6ULL の CSI_DATA01 ピンに接続
43	GPIO4_IO2 3	In/Out	拡張入出力、i.MX6ULL の CSI_DATA02 ピンに接続
44	GPIO4_IO2 8	In/Out	拡張入出力、i.MX6ULL の CSI_DATA07 ピンに接続
45	GPIO4_IO2 7	In/Out	拡張入出力、i.MX6ULL の CSI_DATA06 ピンに接続
46	GPIO4_IO2 6	In/Out	拡張入出力、i.MX6ULL の CSI_DATA05 ピンに接続
47	GPIO4_IO2 5	In/Out	拡張入出力、i.MX6ULL の CSI_DATA04 ピンに接続
48	GPIO4_IO2 0	In/Out	拡張入出力、i.MX6ULL の CSI_HSYNC ピンに接続
49	GPIO4_IO1 9	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、基板上のプルアップ/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。

3.6.15.2. ソフトウェア仕様

デバイスファ
イル

- ・ /dev/dri/card0 (DRM)
- ・ /dev/fb0 (フレームバッファ)

3.6.15.3. 使用方法

LCD オプションセット(7 インチタッチパネル WVGA 液晶)を例に説明します。LCD オプションセット(7 インチタッチパネル WVGA 液晶)の概要については「6.28.4. LCD オプションセット(7 インチタッチパネル WVGA 液晶)」を参照してください。

「図 3.115. 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。

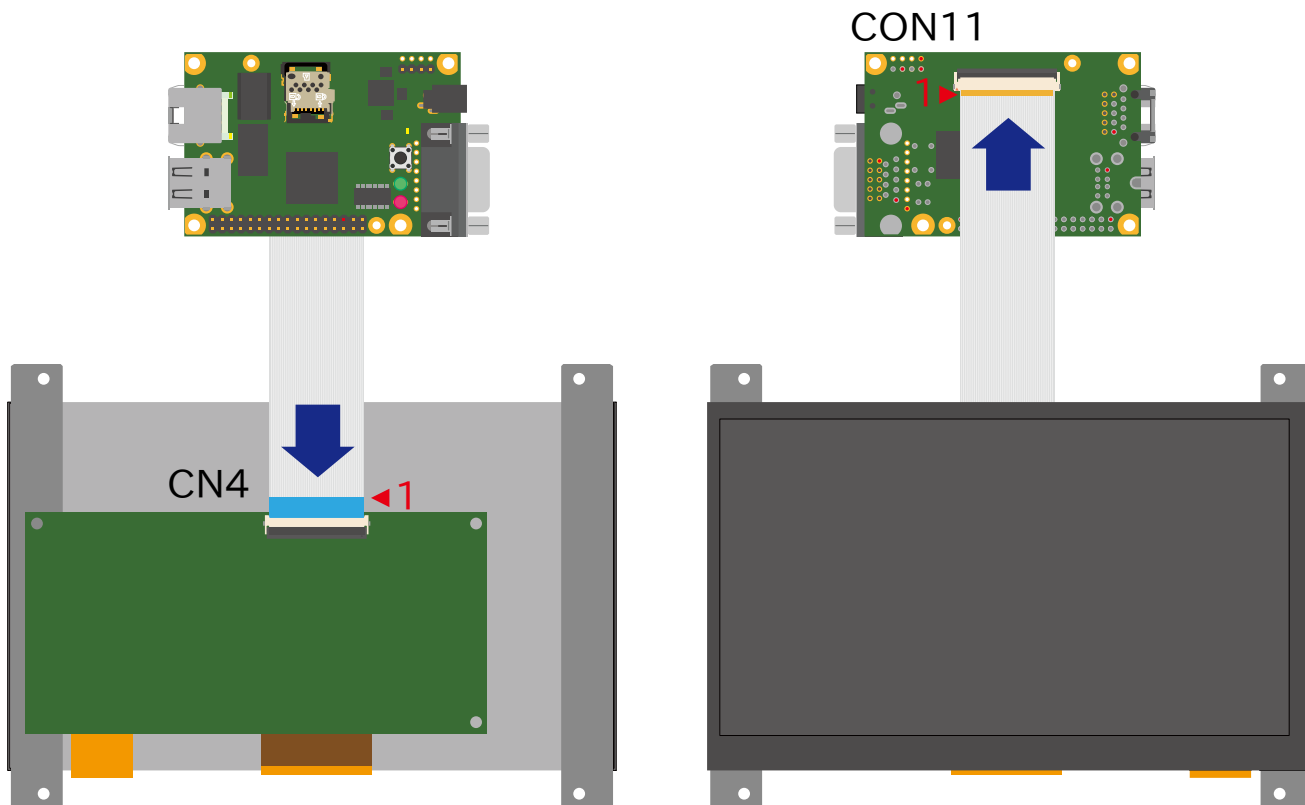


図 3.115 LCD の接続方法

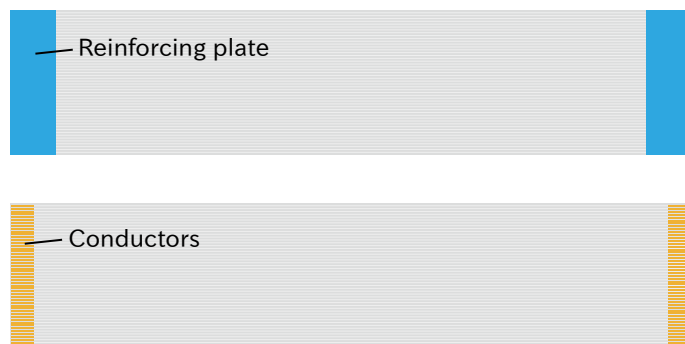




図 3.116 フレキシブルフラットケーブルの形状

 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。

DT overlay の設定を行います。


```
[armadillo~]# vi /boot/overlays.txt
fdt_overlays=armadillo-640-lcd70ext-l00.dtbo
[armadillo~]# persist_file /boot/overlays.txt
[armadillo~]# reboot
```

その後の使用方法については「6.2.9. 画面表示を行う」を参照してください。

3.6.16. 電源を入力する

3.6.16.1. ハードウェア仕様

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 3.117. 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 3.117 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。

インターフェース仕様 (CON13)

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 3.23 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

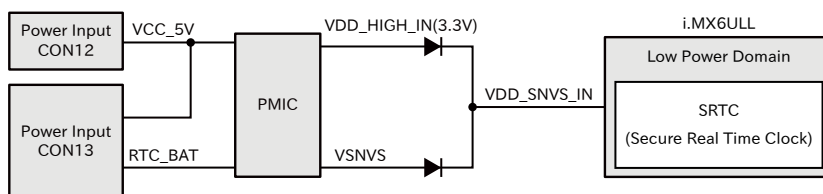


図 3.118 バックアップ電源供給



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。

3.6.17. 起動デバイスを変更する

3.6.17.1. ハードウェア仕様


機能

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 3.24 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 ^[a] 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


^[a]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。

インターフェース仕様 (JP1)

表 3.25 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

インターフェース仕様
(JP2)

表 3.26 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。

3.6.18. ユーザースイッチを使用する

3.6.18.1. ハードウェア仕様

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。

インターフェース仕様 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 3.27 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)

3.6.18.2. ソフトウェア仕様

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 3.28 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` [4]
 イル

3.6.18.3. 使用方法

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-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
```

[4]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

```
add_devices /dev/input
[armadillo ~]# podman_start sw_example
Starting 'sw_example'
c0cd8b801883266197a3c20552b0e8b6c7dd473bb0b24e05bf3ecdb581c822b9
```

図 3.119 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 3.120 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

ユーザースイッチ押下などに対して、細かく動作を指定できる buttond という機能があります。詳細は「6.14. ボタンやキーを扱う」を参照してください。

3.6.19. LED を使用する

3.6.19.1. ハードウェア仕様

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。

インターフェース仕様(LED3、LED4、LED5) LED3、LED4、LED5 は、ユーザー側で自由に利用できる LED です。

表 3.29 LED3、LED4、LED5

部品番号	名称(色)	説明
LED3	ユーザー LED(赤)	i.MX6ULL の GPIO1_I005 ピンに接続、(Low: 消灯、High: 点灯)
LED4	ユーザー LED(緑)	i.MX6ULL の GPIO1_I008 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)

3.6.19.2. ソフトウェア仕様

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
```

3.6.19.3. 使用方法

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 3.121 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 3.122 LED の点灯/消灯の実行例

brightness ファイルを読み出すことで、現在の LED の状態を参照することも可能です。

```
[container ~]# cat /sys/class/leds/red/brightness
```

図 3.123 LED の状態を表示する

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 3.30 LED トリガの種類

設定	説明
none	トリガを設定しません
mmc1	microSD スロットのアクセスランプにします

設定	説明
mmc2	eMMC のアクセスランプにします
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。

```
[container ~]# cat /sys/class/leds/red/trigger
[none] rc-feedback bluetooth-power rfkill-any rfkill-none kbd-scrolllock kbd-num
lock kbd-capslock kbd-kanalock kbd-shiftlock kbd-altgrlock kbd-ctrllock kbd-altl
ock kbd-shiftrllock kbd-shiftrlock kbd-ctrllock kbd-ctrlrlock rfkill0 rfkill1 di
sk-activity disk-read disk-write ide-disk heartbeat cpu cpu0 cpu1 cpu2 cpu3 mmc2
default-on panic mmc1
```

図 3.124 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、心拍のように点灯/消灯を行います。

```
[container ~]# echo heartbeat > /sys/class/leds/red/trigger
```

図 3.125 LED のトリガに heartbeat を指定する

3.6.20. RTC を使用する

3.6.20.1. ハードウェア仕様

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

インターフェース仕様 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。詳細は「3.6.16. 電源を入力する」を参照してください。

3.6.20.2. ソフトウェア仕様

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

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。

3.6.20.3. 使用方法

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 3.126 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 3.127 `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。

3.6.21. 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/ttyxc1
[armadillo ~]# podman_start wisun_example
Starting 'wisun_example'
ef9a5a2f7eee4236cb28c1fbf5090a6f0db9d6dfe7f3a34573867e0833dd3122
[armadillo ~]# podman exec -it wisun_example sh
```

```
[container ~]# ls /dev/ttyxc1
/dev/ttyxc1
```

図 3.128 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/ttyxc1` を使って Wi-SUN データの送受信ができるようになります。

3.6.22. 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/ttyxcN` 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/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/enoccean_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/ttyxc1
[armadillo ~]# podman_start enoccean_example
Starting 'enoccean_example'
a808b491a100f9078d8c72a7b36966d9182614f3657fe054fb8d7f87b0d4b31c
[armadillo ~]# podman exec -it enoccean_example sh
[container ~]# ls /dev/ttyxc1
/dev/ttyxc1
```

図 3.129 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/ttyxc1` を使って EnOcean データの送受信ができるようになります。

3.7. ソフトウェアの設計

Armadillo-640 を用いた製品のソフトウェア設計は、一般的な組み込み開発と基本的には変わりません。しかし、Armadillo Base OS という独自 OS を搭載しているため、ソフトウェアの設計には特有のポイントがいくつかあります。本章では、それらの設計時に考慮すべき Armadillo Base OS 特有のポイントについて紹介していきます。

3.7.1. 開発者が開発するもの、開発しなくていいもの

Armadillo Base OS では、組み込み機器において必要になる様々な機能を標準で搭載しています。

「図 3.130. 開発者が開発するもの、開発しなくていいもの」は、Armadillo Base OS 搭載製品において、開発者が開発するものと開発しなくていいものをまとめた図です。

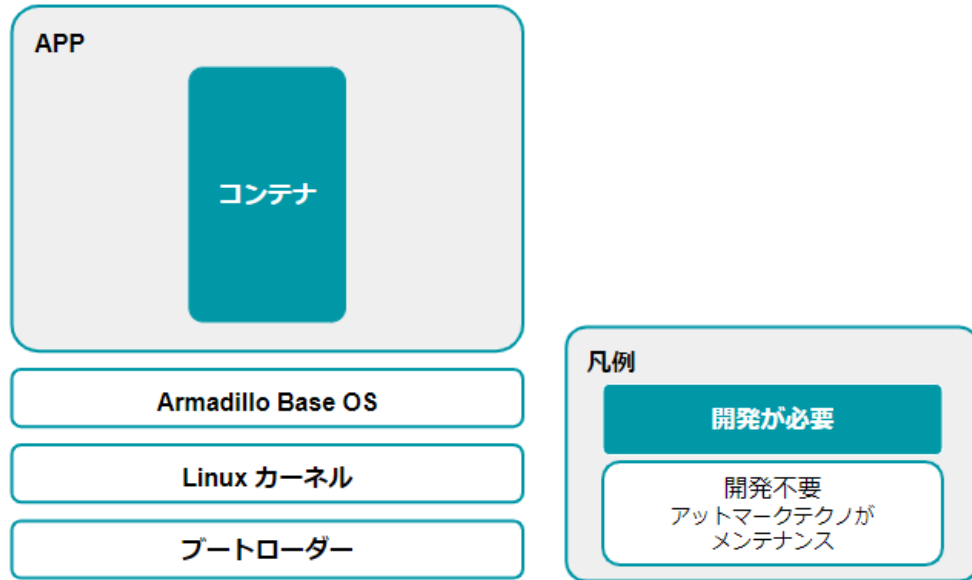


図 3.130 開発者が開発するもの、開発しなくていいもの

開発しなくていいものについては設計を考える必要はありません。開発するものに絞って設計を進めることができます。

3.7.2. ユーザーアプリケーションの設計

Armadillo Base OS では基本的にユーザーアプリケーションを Podman コンテナ上で実行します。そのため、実行環境として Armadillo Base OS を意識する必要はありません。

Podman は、同じくコンテナを扱えるソフトウェアである Docker [<https://www.docker.com/>] と基本的に互換性があります。

Docker Hub [https://hub.docker.com/search?type=image&image_filter=official] などから使い慣れたディストリビューションのコンテナイメージを取得して開発することができます。

3.7.2.1. ユーザーデータの保存場所

アプリケーションが出力するユーザーデータで保存が必要なものは、「3.3.4.1. 電源を切っても保持されるディレクトリ(ユーザーデータディレクトリ)」にも記載があるとおり、`/var/app/volumes/` 以下に配置してください。

色々な場所にデータが保存されていますと Armadillo-640 の初期化を行う際に削除の処理が煩雑になりますので、`/var/app/volumes/` 以下に集約してください。

3.7.2.2. アプリケーション設定情報の保存場所

開発したアプリケーションやコンテナがバージョンアップした際にも必要となる設定情報は、「3.3.4.1. 電源を切っても保持されるディレクトリ(ユーザーデータディレクトリ)」にも記載があるとおり、`/var/app/rollback/volumes/` 以下に保存してください。

3.7.3. ログの設計

ユーザーアプリケーションのログは、不具合発生時の原因究明の一助になるため必ず残しておくことを推奨します。

3.7.3.1. ログの保存場所

ユーザーアプリケーションが出力するログは、「3.3.4.1. 電源を切っても保持されるディレクトリ(ユーザーデータディレクトリ)」にも記載があるとおり、`/var/app/volumes/`以下に配置するのが良いです。

コンテナの中から `/var/app/volumes/` ディレクトリにアクセスすることになります。手順についての詳細は実際に開発を行う箇所にて紹介します。

3.7.3.2. 保存すべきログ

- ・ Ethernet、LTE、BT、WLAN などのネットワーク系のログ

一般に不具合発生時によく疑われる箇所なので、最低でも接続・切断情報などのログを残しておくことをおすすめします。

- ・ ソフトウェアのバージョン

`/etc/sw-versions` というファイルが Armadillo Base OS 上に存在します。これは、SWUpdate に管理されている各ソフトウェアのバージョンが記録されているファイルです。このファイルの内容をログに含めておくことで、当時のバージョンを記録することができます。

- ・ A/B 面どちらであったか

アップデート後になにか不具合があって、自動的にロールバックしてしまう場合があります。後でログを確認する際に、当時 A/B 面どちらであったかで環境が大きく変わってしまい解析に時間がかかる場合があるので、どちらの面で動作していたかをログに残しておくことをおすすめします。

「図 3.131. 現在の面の確認方法」に示すコマンドを実行することで、現在 A/B どちらの面で起動しているかを確認できます。

```
[armadillo ~]# abos-ctrl  
rollbackCurrently booted on /dev/mmcblk0p1 ❶  
: (省略)
```

図 3.131 現在の面の確認方法

- ❶ この実行結果から今の面は `/dev/mmcblk0p1` であることが分かります。

3.7.4. 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。

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。

3.7.5. コンテナに Armadillo の情報を渡す方法

Armadillo Base OS からコンテナに環境変数として情報を渡すためにコンテナ起動設定ファイルを使用します。

コンテナ起動設定ファイル (conf ファイル) に関しては「6.2.4. コンテナ起動設定ファイルを作成する」を参照してください。

- ・ アットマークテクノが提供する情報を環境変数として渡す

コンテナ起動設定ファイルに `add_armadillo_env` を使用してください。

アットマークテクノが設定した LAN1 (eth0) の MAC アドレス、個体番号などの Armadillo の情報を環境変数としてコンテナに渡します。

`add_armadillo_env` については「6.2.4.6. 個体識別情報の環境変数の追加」を参照してください。

- ・ 任意の情報を環境変数として渡す

コンテナ起動設定ファイルに `add_args` を使用してください。

`add_args` については「6.2.4.20. podman run に引数を渡す設定」を参照してください。

`add_args` を下記のように使用することでコンテナに環境変数として情報を渡すことができます。

```
add_args --env=<環境変数名>=<値> ❶
```

図 3.132 `add_args` を用いてコンテナに情報を渡すための書き方

- ❶ シェルコマンドの出力を環境変数に代入する場合は `<値>` として `$(シェルコマンド)` を使用してください。

`add_args --env` の例を示します。

```
add_args --env=MY_ENV=my_value
```

図 3.133 `add_args` を用いてコンテナに情報を渡す例

これにより、コンテナ内の環境変数 `MY_ENV` に文字列 `my_value` が設定されます。

3.7.6. Armadillo Base OS のデフォルトで開放しているポート

Armadillo Base OS では「表 3.31. Armadillo Base OS のデフォルトで開放しているポート」に示すポートをデフォルトで開放しています。

表 3.31 Armadillo Base OS のデフォルトで開放しているポート

ポート番号	プロトコル	使用目的
58080	TCP	ABOS Web
5353	UDP	avahi(mDNS)

使用していないポートを開放することは攻撃の標的になります。使用しないサービスを停止しポートを閉じてください。

ABOS Web のサービスを停止する方法は「6.9.9. ABOS Web を停止する」を、起動する方法は「6.9.10. ABOS Web を起動する」を参照してください。

「図 3.134. avahi-daemon を停止する」に avahi のサービスを停止する方法を示します。

```
[armadillo ~]# rc-update | grep avahi-daemon ❶
      avahi-daemon |      default
[armadillo ~]# rc-service avahi-daemon status ❷
* status: started
[armadillo ~]# rc-service avahi-daemon stop ❸
avahi-daemon      | * Stopping avahi-daemon ... [ ok ]
[armadillo ~]# rc-update del avahi-daemon ❹
* service avahi-daemon deleted from runlevel default
[armadillo ~]# persist_file -d /etc/runlevels/default/avahi-daemon ❺
```

図 3.134 avahi-daemon を停止する

- ❶ OpenRC に avahi のサービスが登録されていることを確認します。
- ❷ avahi のサービスが起動していることを確認します。
- ❸ avahi のサービスを停止します。
- ❹ サービスを管理している OpenRC から avahi のサービスの登録を解除します。
- ❺ サービス設定ファイルの削除を永続化します。

「図 3.135. avahi-daemon を起動する」に avahi サービスを起動する方法を示します。

```
[armadillo ~]# rc-update | grep avahi-daemon ❶
[armadillo ~]# rc-update add avahi-daemon ❷
* service avahi-daemon added to runlevel default
[armadillo ~]# rc-service avahi-daemon start ❸
avahi-daemon      | * Starting avahi-daemon ... [ ok ]
[armadillo ~]# persist_file /etc/runlevels/default/avahi-daemon ❹
```

図 3.135 avahi-daemon を起動する

- ❶ OpenRC に avahi のサービスが登録されていないことを確認します。
- ❷ サービスを管理している OpenRC に avahi のサービスを登録します。
- ❸ avahi のサービスを起動します。

- ④ サービス設定ファイルを永続化します。

3.8. ネットワーク設定

必要であれば、Armadillo のネットワークの設定を行います。

3.8.1. ABOS Web とは

Armadillo Base OS(以降、ABOS) には、Armadillo と作業用 PC が同一 LAN 内に存在していれば Web ブラウザからネットワークの設定や、SWU イメージのインストールなどを行うことが可能となる、ABOS Web という機能があります。この機能は、バージョン v3.17.4-at.7 以降の ABOS に標準で組み込まれています。

現在、ネットワークに関して ABOS Web で設定できるのは以下のものです。

- ・ 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



ABOS Web で設定できる項目はネットワーク関連以外にもありますが、それらについては「6.9. Web UI から Armadillo をセットアップする (ABOS Web)」で紹介します。



バージョン v3.17.4-at.7 以前から ABOS をアップデートした場合の注意

バージョン v3.17.4-at.7 以前からこのバージョン以降へ ABOS をアップデートすると、avahi サービスが新しく追加されます。ABOS Web にアクセスできるようにするためには、この avahi サービスが自動起動するように設定を変更する必要があります。そのため、以下の手順にしたがって設定を変更してください。(新しく追加されたサービスが自動起動することによる悪影響を防ぐため、アップデート直後では avahi サービスは自動起動しない設定になっています。)

```
[armadillo ~]# rc-update add avahi-daemon
[armadillo ~]# rc-service avahi-daemon start
[armadillo ~]# persist_file /etc/runlevels/default/avahi-daemon
```



バージョン 4.13 以前の mkswu を使用した場合の注意

バージョン v3.17.4-at.7 以降の ABOS に、バージョン 4.13 以前の mkswu の mkswu --init で作成した initial_setup.swu をインストールした場合、ABOS Web にパスワードが設定されていないため自動起動しません。そのため、以下の手順にしたがって ABOS Web のパスワードを設定してください。

```
[armadillo ~]# passwd abos-web-admin
[armadillo ~]# persist_file /etc/shadow
[armadillo ~]# rc-service abos-web restart
```

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に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。

以下では、ABOS Web を利用した各種ネットワーク設定の方法について紹介します。

3.8.2. ABOS Web へのアクセス

Armadillo と PC を有線 LAN で接続し、Armadillo の電源を入れて PC で Web ブラウザを起動した後、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 は、初期状態では同一サブネットのネットワークのみアクセス可能です。サブネット外からのアクセスを許可したい場合は、`/etc/atmark/abos_web/init.conf` を作成し、ABOS Web のサービスを再起動してください。

以下の例ではコンテナとループバックからのアクセスのみを許可します：

```
[armadillo ~]# vi /etc/atmark/abos_web/init.conf
command_args="--allowed-subnets '10.88.0.0/16 127.0.0.0/8 ::1/128'"
[armadillo ~]# persist_file -v /etc/atmark/abos_web/init.conf
'/mnt/etc/atmark/abos_web/init.conf' -> '/target/etc/atmark/abos_web/init.conf'
[armadillo ~]# rc-service abos-web restart
```



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。

また、VS Code 上で ABOSDE(Armadillo Base OS Development Environment) から、ABOS Web が動作している Armadillo の一覧を確

認し、指定した Armadillo の ABOS Web を Web ブラウザ で開くことができます。ただし、ATDE のネットワークを NAT に設定している場合は Armadillo がリストに表示されません。

「図 3.136. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする」の赤枠で囲われているボタンをクリックすることで、ローカルネットワーク上で ABOS Web が実行されている Armadillo をスキャンすることができます。

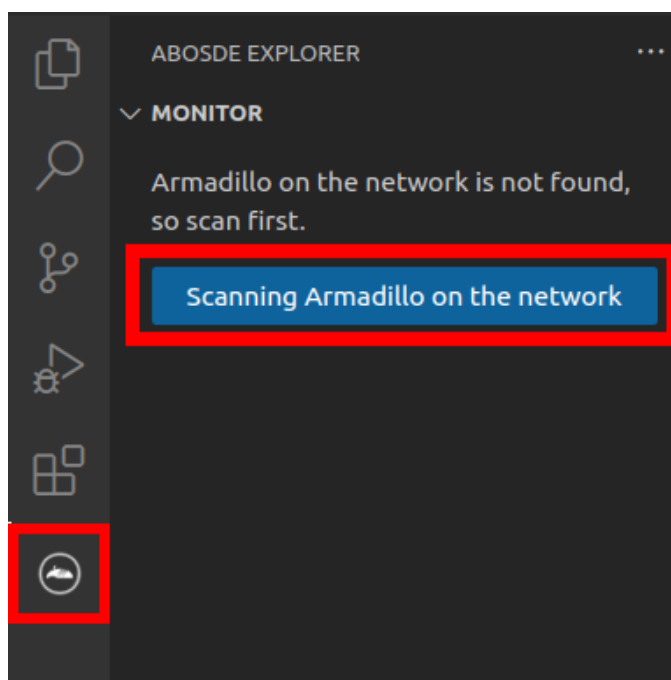


図 3.136 ABOSDE で ローカルネットワーク上の Armadillo をスキャンする

「図 3.137. ABOSDE を使用して ABOS Web を開く」の赤枠で囲われているマークをクリックすることで、ABOS Web を Web ブラウザで開くことができます。

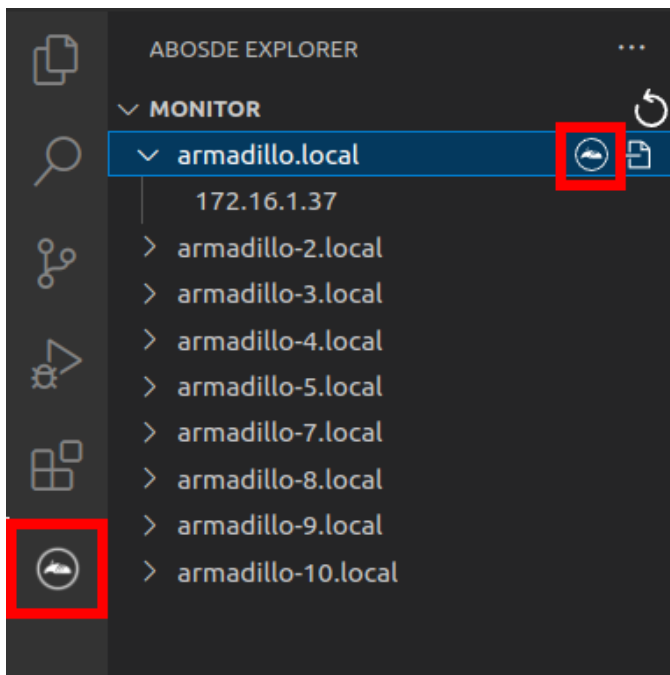


図 3.137 ABOSDE を使用して ABOS Web を開く

「図 3.138. ABOSDE に表示されている Armadillo を更新する」の赤枠で囲われているマークをクリックすることで、ABOSDE に表示されている Armadillo を更新することができます。

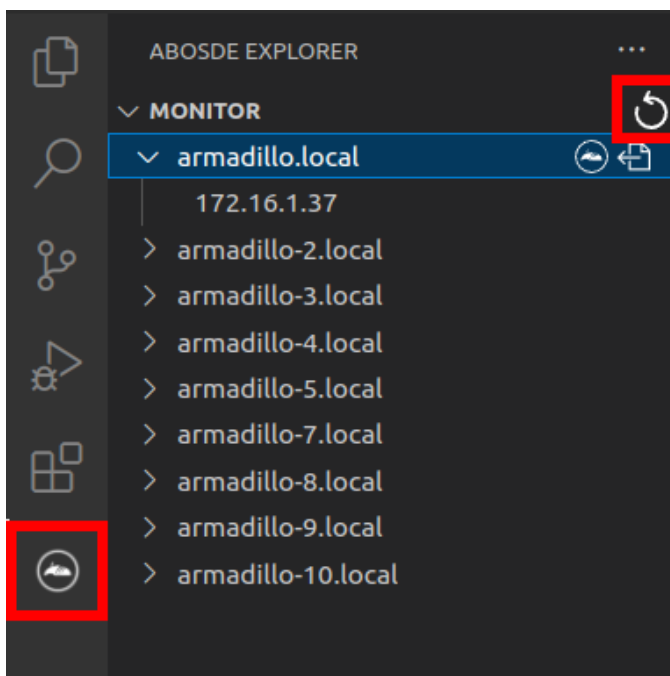


図 3.138 ABOSDE に表示されている Armadillo を更新する

3.8.3. ABOS Web のパスワード登録

「3.1.5.1. initial_setup.swu の作成」で ABOS Web のログイン用パスワードを設定していない場合、ABOS Web 初回ログイン時に、「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。



図 3.139 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 3.140 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。

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 のログイン画面が表示されます。



図 3.141 ログイン画面

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。

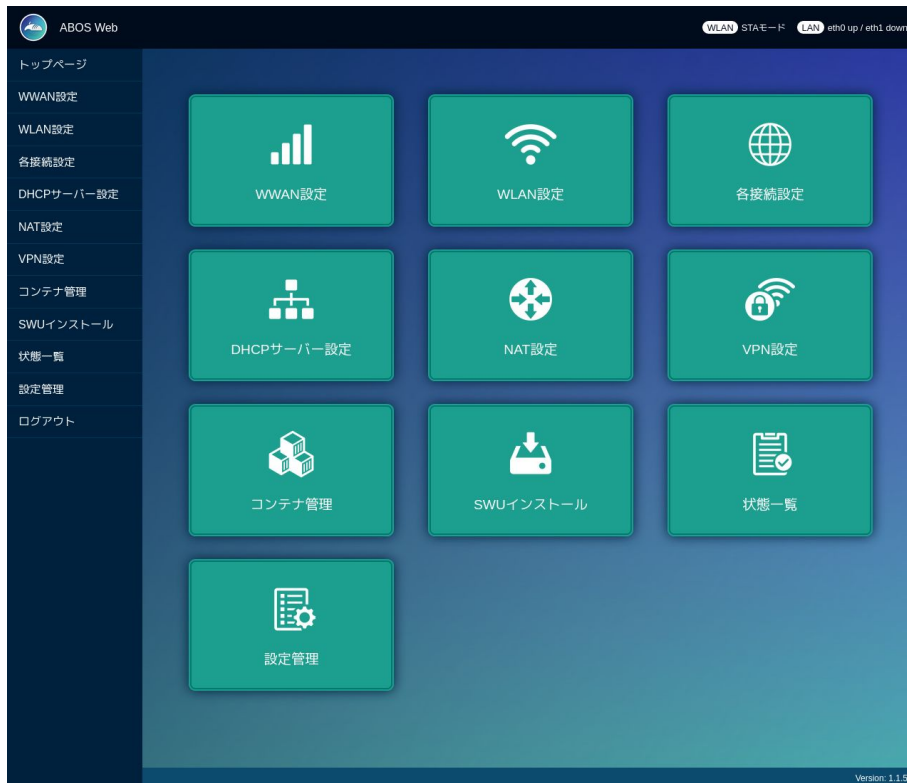


図 3.142 トップページ

3.8.4. ABOS Web のパスワード変更

登録した ABOS Web のログイン用パスワードは「設定管理」画面から変更することができます。トップページから「設定管理」をクリックすると、移動した先にパスワード変更画面が表示されますので、現在のパスワードと変更後のパスワードを入力して登録ボタンをクリックしてください。

パスワード変更

現在のパスワード

新しいパスワード(8 文字以上)

新しいパスワード(確認)

図 3.143 ログイン画面

3.8.5. 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。

3.8.6. ログアウト

ABOS Web で必要なセットアップを行なったら、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。

3.8.7. 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 接続設定の編集と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「接続して保存」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 3.144. WWAN 設定画面」に、WWAN 設定を行った状態を示します。

現在のWWAN接続情報

接続状態： 接続中

APN	ユーザ名	認証方式	MCC/MNC	IMSI
[REDACTED]	[REDACTED]	CHAP	--	[REDACTED]

IPアドレス	サブネットマスク	ゲートウェイ	インターフェース
[REDACTED]	255.255.255.255	0.0.0.0	ppp0

切断

APN	ユーザ名
<input checked="" type="radio"/> [REDACTED]	[REDACTED]

接続
設定を編集
設定を削除

WWAN接続情報入力

WWAN接続に必要な情報を入力してください

APN

ユーザー名

パスワード

認証方式

MCC/MNC

IPv6 設定

接続して保存

173
 図 3.144 WWAN 設定画面



ABOS Web のバージョン 1.3.3 以降では「IPv6 設定」を選択することができます。使用する SIM によっては IPv6 が有効だと接続できず、無効にすると接続できることがあります。その場合は、この設定を「使用しない」に設定して接続してください。

3.8.8. 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。

3.8.8.1. 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「接続して保存」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。

ABOS-WEB 上では複数のネットワーク設定を保存することが可能です。設定項目のうちネットワーク情報を入力した後、「保存」ボタンをクリックすると、入力した内容の登録のみを行い、接続は行いません。登録した設定の一覧は WLAN ページの中央にあるリストに表示されます。このリストでは WLAN 設定の接続／編集／削除を行うことができます。保存した設定に接続先を変更したい場合はリストから選択して、「接続」ボタンをクリックしてください。保存した設定を編集したい場合はリストから選択して、「設定を編集」ボタンをクリックしてください。保存した設定を削除したい場合はリストから選択して、「設定を削除」ボタンをクリックしてください。

「図 3.145. 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 3.145 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。

3.8.8.2. 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 3.146. 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。

現在のアクセスポイント情報

SSID	使用周波数	チャンネル
abos-web	5GHz	36

ブリッジアドレス	サブネットマスク	インターフェース
192.168.1.1	255.255.255.0	br_ap

設定を削除

アクセスポイント設定入力

Armadilloをアクセスポイントとして使用するために必要な設定を入力してください

ブリッジアドレス

サブネットマスク

使用周波数

使用チャンネル

SSID

パスワード

設定

図 3.146 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。

3.8.9. 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。



図 3.147 現在の接続情報画面

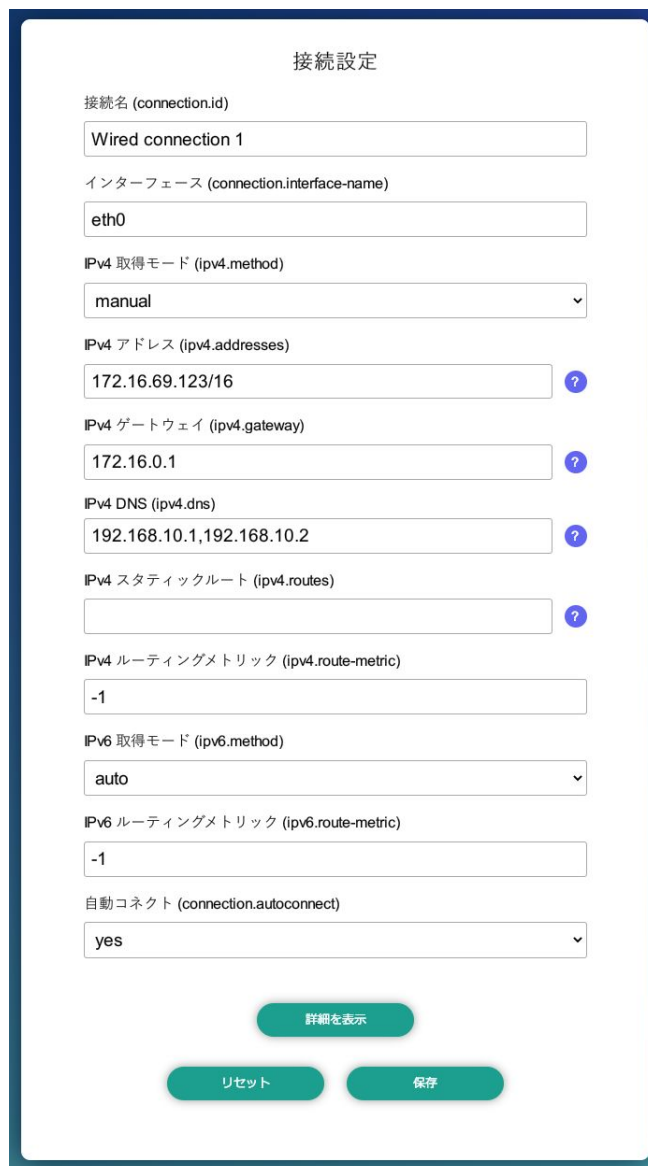
ここで、「ネットワーク接続」は、Linux のネットワーク接続管理機能（NetworkManager）が管理するコネクションです。ネットワーク接続に対する設定項目の詳細は、NetworkManager のリファレンス（<https://developer-old.gnome.org/NetworkManager/stable/nmcli.html>）をご覧ください。接続設定内容を編集したい接続を選択して「設定を編集」ボタンをクリックすると、設定内容の編集画面を表示します。LAN の接続以外、つまり、WWAN と WLAN の接続に対する設定は、「WWAN 設定」や「WLAN 設定」の設定画面を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

3.8.9.1. 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 3.148. 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。



The screenshot shows a web interface for configuring a LAN connection. The title is "接続設定" (Connection Settings). The form includes the following fields and values:

- 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): (empty)
- 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
- 自動コネクット (connection.autoconnect): yes

At the bottom of the form, there are three buttons: "詳細を表示" (Show details), "リセット" (Reset), and "保存" (Save).

図 3.148 LAN 接続設定で固定 IP アドレスに設定した画面

3.8.9.2. WWAN 接続設定

WWAN 接続の接続名は、デフォルトでは "gsm-ttyCommModem" です。

3.8.9.3. 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。

3.8.10. 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 3.149. eth0 に対する DHCP サーバー設定」に、一つめの LAN ポート (eth0) に対する設定を行った状態を示します。

現在のDHCP情報

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 3.149 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。

3.8.11. 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。

3.8.11.1. 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 3.150. 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。



The screenshot displays the NAT configuration page, divided into two main sections:

- 現在のNAT設定情報 (Current NAT Settings Information):** This section shows a table with the header "宛先インターフェース" (Destination Interface). A single entry is listed with "ppp0" as the interface name, indicated by a selected radio button.
- NAT情報入力 (NAT Information Input):** This section prompts the user to "宛先インターフェースを選択してください" (Please select the destination interface). It features a dropdown menu labeled "インターフェース" (Interface) with "ppp0" selected.

Buttons for "削除" (Delete) and "設定" (Settings) are visible at the bottom of each section.

図 3.150 LTE を宛先インターフェースに指定した設定

3.8.11.2. 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 3.151. 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。

現在のポートフォワーディング設定情報

受信インターフェース	プロトコル	変換前ポート番号	宛先アドレス	変換後ポート番号
<input checked="" type="radio"/> ppp0	tcp	8080	192.168.1.100	80

削除

ポートフォワーディング情報入力

インターフェース
veth0

プロトコル
tcp

変換前ポート番号
8080

宛先アドレス
192.168.1.100

変換後ポート番号
80

設定

図 3.151 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

3.8.12. 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 3.152. VPN 設定」に、VPN 接続設定を行った状態を示します。



図 3.152 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。

3.8.13. 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。

3.9. ABOS Web をカスタマイズする

ABOS Web では以下の要素についてお客様自身で用意したものを使用してカスタマイズすることができます。

- ・ ロゴ画像
- ・ ヘッダロゴアイコン画像
- ・ ヘッダタイトル
- ・ favicon 画像
- ・ 背景色
- ・ メニューの表示名

ABOS Web をお客様の最終製品に組み込む場合、自社のロゴ画像に変更するといったような使い方ができます。

カスタマイズは、「設定管理」で行うことができます。



カスタマイズは ABOS Web のバージョン 1.3.0 以降で対応しています。



図 3.153 ABOS Web のカスタマイズ設定

・ **ロゴ画像**

ログインページや新規パスワード設定画面で表示される画像です。「ファイルを選択」をクリックしてアップロードしたい画像ファイルを選択してください。フォーマットは PNG のみで、ファイルサイズは 3MB のものまでアップロードできます。

・ **ヘッダロゴアイコン画像**

画面左上に常に表示されている画像です。「ファイルを選択」をクリックしてアップロードしたい画像ファイルを選択してください。フォーマットは PNG のみで、ファイルサイズは 3MB のものまでアップロードできます。

・ **ヘッダタイトル**

画面左上に常に表示されている文字列です。24 文字まで入力できます。

・ **favicon 画像**

Web ブラウザのタブなどに小さく表示される画像です。favicon 画像は以下の種類を favicon ディレクトリに保存して、favicon ディレクトリごと zip 圧縮したものをアップロードしてください。

表 3.32 用意する favicon 画像

ファイル名	縦横サイズ	説明
android-chrome-192x192.png	192x192	スマートフォンのホームに Web ページを追加した時に使用されます。
android-chrome-512x512.png	512x512	Web ページを開いた時のスプラッシュ画面に使用されます。
apple-touch-icon.png	180x180	スマートフォンのホームに Web ページを追加した時に使用されます。
favicon-16x16.png	16x16	Web ブラウザで使用されます。
favicon-32x32.png	32x32	Web ブラウザで使用されます。
mstile-150x150.png	150x150	Windows でスタート画面にピン止めしたときに使用されます。

・ **背景色**

5 種類の中から選択できます。

・ **メニューの表示名**

画面左にあるメニューの表示名を変更する、または非表示にすることができます。「メニュー項目を変更する」をクリックし、変更用ページへ行ってください。

メニュー項目の変更

空欄にしたメニュー項目は非表示になります

項目名1: トップページ

項目名1の説明

項目名2: WWAN設定

項目名2の説明

図 3.154 メニュー変更画面 (一部)

各メニュー項目名と説明を変更することができます。項目名を空欄にするとそのメニューは非表示になります。入力し終わったらページ下部の「メニューを設定」をクリックしてください。

画像やメニューの変更後、すぐに Web ブラウザ画面に反映されない場合は、お使いの Web ブラウザの設定でキャッシュの削除を行ってください。

変更完了後は、「カスタマイズ機能を無効にする」をクリックするとカスタマイズ項目が非表示になりそれ以上カスタマイズできなくなります。お客様の最終製品に ABOS Web を組み込む場合に実行してください。



Armadillo 内の `/etc/atmark/abos_web/customize_disable` ファイルを削除すると、再びカスタマイズ項目が表示されるようになります。

3.10. Network Time Protocol (NTP, ネットワーク・タイム・プロトコル) の設定

Armadillo Base OS では `chronyd` を使用しています。

デフォルトの設定（使用するサーバーなど）は `/lib/chrony.conf.d/` にあり、設定変更用に `/etc/chrony/conf.d/` のファイルも読み込みます。`/etc/chrony/conf.d/` ディレクトリに `/lib/chrony.conf.d/` と同名の設定ファイルを配置することで、デフォルトのファイルを読み込まないようにします。

時刻取得に関する設定は 2 つのファイルに分かれています：

- ・ `initstepslew.conf` : `chronyd` 起動時「`initstepslew`」コマンドでサーバーと通信し時刻を取得します。
- ・ `servers.conf` : `chronyd` 起動後周期的に「`pool`」または「`server`」コマンドでサーバーと通信し時刻を補正します。

例えば、NTP サーバーを変更する際は「図 3.155. chronyd のコンフィグの変更例」に示す通り /etc/chrony/conf.d/initstepslew.conf と /etc/chrony/conf.d/servers.conf に記載します：

```
[armadillo ~]# vi /etc/chrony/conf.d/initstepslew.conf ❶
initstepslew 10 192.0.2.1
[armadillo ~]# vi /etc/chrony/conf.d/servers.conf ❷
server 192.0.2.1 iburst
[armadillo ~]# persist_file -rv /etc/chrony/conf.d ❸
'/mnt/etc/chrony/conf.d/initstepslew.conf' -> '/target/etc/chrony/conf.d/initstepslew.conf'
'/mnt/etc/chrony/conf.d/servers.conf' -> '/target/etc/chrony/conf.d/servers.conf'
[armadillo ~]# rc-service chronyd restart ❹
chronyd          | * Stopping chronyd ... [ ok ]
chronyd          | * Starting chronyd ... [ ok ]
armadillo:~# chronyc -n sources ❺
MS Name/IP address          Stratum Poll Reach LastRx Last sample
=====
^* 192.0.2.1                 2    6   17   24   +11us[ +34us] +/- 53ms
```

図 3.155 chronyd のコンフィグの変更例

- ❶ 起動時のサーバー設定です。不要な場合は空のファイルを生成してください。
- ❷ 運用時のサーバー設定です。複数の行または「pool」の設定も可能です。
- ❸ ファイルを保存します。
- ❹ chronyd サービスを再起動します。
- ❺ 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。

NTP の設定は ABOS Web や Rest API を使って行うこともできます。詳細は、「6.9.5. 時刻設定」および「6.9.6.13. Rest API : 時刻の設定」を参照してください。

3.11. Armadillo Twin を体験する

Armadillo Twin を利用したデバイス運用管理を検討する場合は、一度 Armadillo Twin をお試しください。詳しくはおすすしめします。Armadillo Twin は、無償トライアルでご登録いただくことで、3ヶ月間無償で全ての機能をご利用いただくことができます。また、トライアル中の設定内容は、有料の月額プランに申込後も引き継いで利用することができます。

詳細は Armadillo Twin ユーザーマニュアル 「アカウント・ユーザーを作成する」 [<https://manual.armadillo-twin.com/create-account-and-user/>] をご確認ください。

3.12. ABOSDE によるアプリケーションの開発

ここでは、ABOSDE(Armadillo Base OS Development Environment) によるアプリケーション開発の概要と ABOSDE で作成される各プロジェクトの違いについて説明します。

ABOSDE は Visual Studio Code にインストールできる開発用エクステンションです。ABOSDE を使用することで、コンテナ及びコンテナ自動起動用設定ファイルの作成、コンテナ内におけるパッケージのインストール、コンテナ内で動作するアプリケーション本体の開発をすべて VS Code 内で行うことができます。

ABOSDE では、以下のようなアプリケーションを開発できます。

- ・ CUI アプリケーション
- ・ C 言語アプリケーション

3.12.1. ABOSDE の対応言語

「表 3.33. ABOSDE の対応言語」に示すように、アプリケーション毎に対応している言語が異なります。

表 3.33 ABOSDE の対応言語

アプリケーションの種類	使用言語 (フレームワーク)
CUI アプリケーション	シェルスクリプト
	Python
C 言語アプリケーション	C 言語

3.12.2. 参照する開発手順の章の選択

どのようなアプリケーションを開発するかによって ABOSDE による開発手順が異なります。「図 3.156. 参照する開発手順の章を選択する流れ」を参考に、ご自身が開発するアプリケーションに適した章を参照してください。

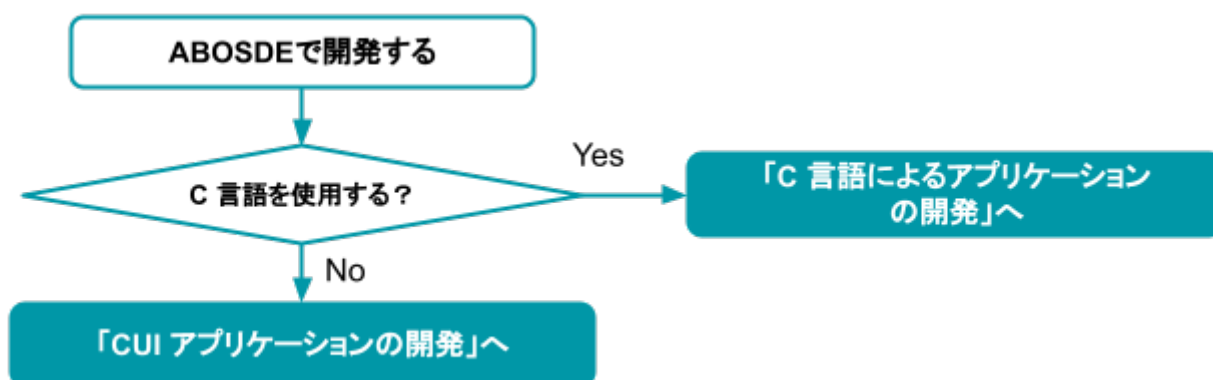


図 3.156 参照する開発手順の章を選択する流れ

- | | |
|----------------------------|---|
| <p>CUI アプリケーション</p> | <ul style="list-style-type: none"> ・ 対象ユーザー ・ 画面を使用しないアプリケーションを開発したい ・ マニュアルの参照先 ・ 「3.13. CUI アプリケーションの開発」を参照 |
| <p>C 言語アプリケーション</p> | <ul style="list-style-type: none"> ・ 対象ユーザー ・ C 言語でないと実現できないアプリケーションを開発したい ・ 既存の C 言語によって開発されたアプリケーションを Armadillo で動作させたい ・ 開発環境に制約がある ・ マニュアルの参照先 |

・「3.14. C 言語によるアプリケーションの開発」を参照

3.13. CUI アプリケーションの開発

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ご紹介します。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。

3.13.1. 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。

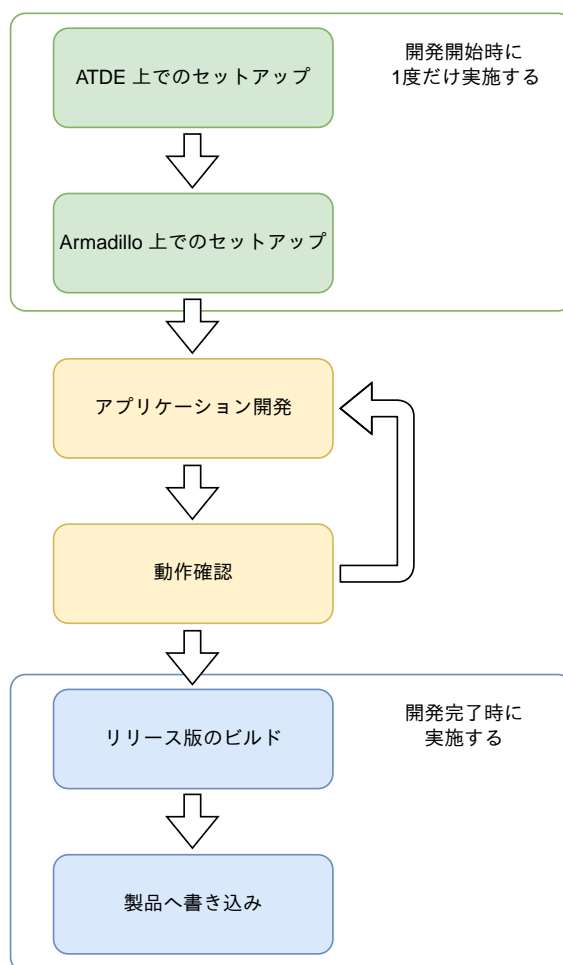


図 3.157 CUI アプリケーション開発の流れ

3.13.2. 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「3.1. 開発の準備」を参照して ATDE 及び、VS Code のセットアップを完了してください。

3.13.2.1. プロジェクトの作成

VS Code 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。この操作により、選択した保存先に、入力したプロジェクト名と同名のディレクトリが作成されます。

また、ここでは次のように設定しています。

- ・ 保存先 : ホームディレクトリ
- ・ プロジェクト名 : my_project

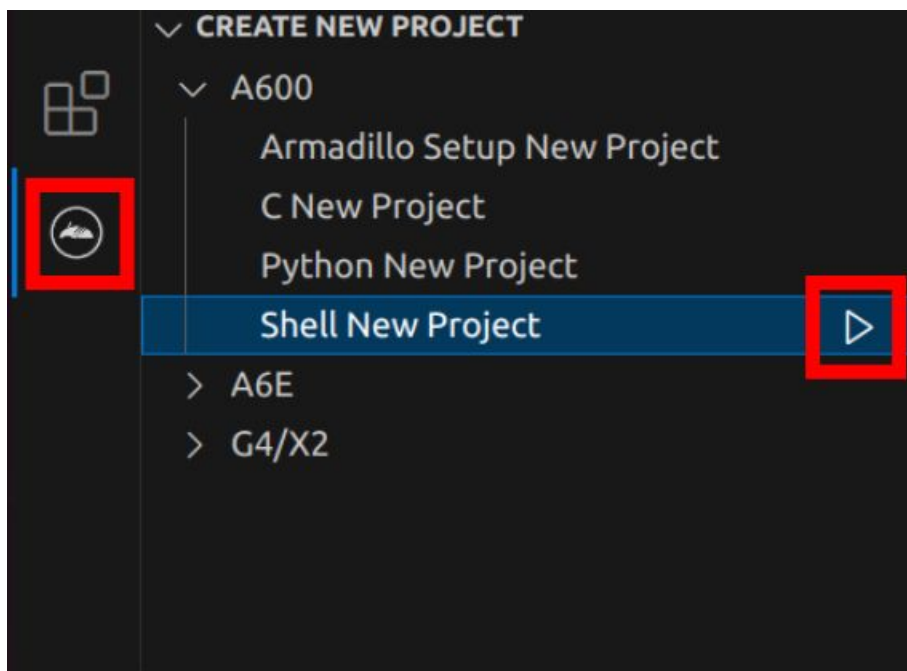


図 3.158 プロジェクトを作成する

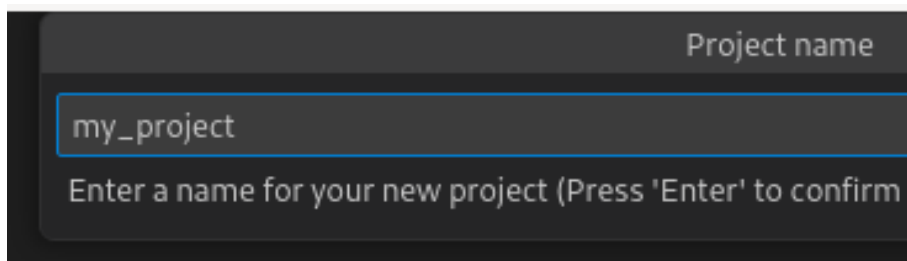


図 3.159 プロジェクト名を入力する

3.13.3. アプリケーション開発

3.13.3.1. VS Code 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 VS Code を起動します。

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

図 3.160 VS Code で my_project を起動する

3.13.3.2. ディレクトリ構成

プロジェクトには下記のディレクトリがあります。

- ・ **app** : アプリケーションのソースです。Armadillo ではビルドしたアプリケーションが `/var/app/rollback/volumes/my_project` にコピーされます。
- ・ **requirements.txt** : Python プロジェクトにのみ存在しており、このファイルに記載したパッケージは pip を使用してインストールされます。
- ・ **config** : 設定に関わるファイルが含まれるディレクトリです。
- ・ **app.conf** : コンテナのコンフィグです。記載内容については「6.2.4. コンテナ起動設定ファイルを作成する」を参照してください。
- ・ **app.desc** : SWU イメージを生成するための .desc ファイルです。記載内容については「6.4. mkswu の .desc ファイルを編集する」を参照してください。
- ・ **ssh_config** : Armadillo への ssh 接続に使用します。「3.13.7.2. ssh 接続に使用する IP アドレスの設定」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が含まれるディレクトリです。
- ・ **packages.txt** : このファイル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** : 直接編集することも可能です。

デフォルトのコンテナコンフィグ (`app.conf`) ではシェルスクリプトの場合は `app` の `src/main.sh` または Python の場合 `src/main.py` を実行しますので、リネームが必要な場合にコンテナのコンフィグも修正してください。

このサンプルアプリケーションは、CPU と SOC の温度を `/vol_data/log/temp.txt` に出力し、LED4 を点滅させます。

3.13.3.3. 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 VS Code を起動してください。

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

図 3.161 初期設定を行う

VS Code の左ペインの `[my_project]` から `[Setup environment]` を実行します。

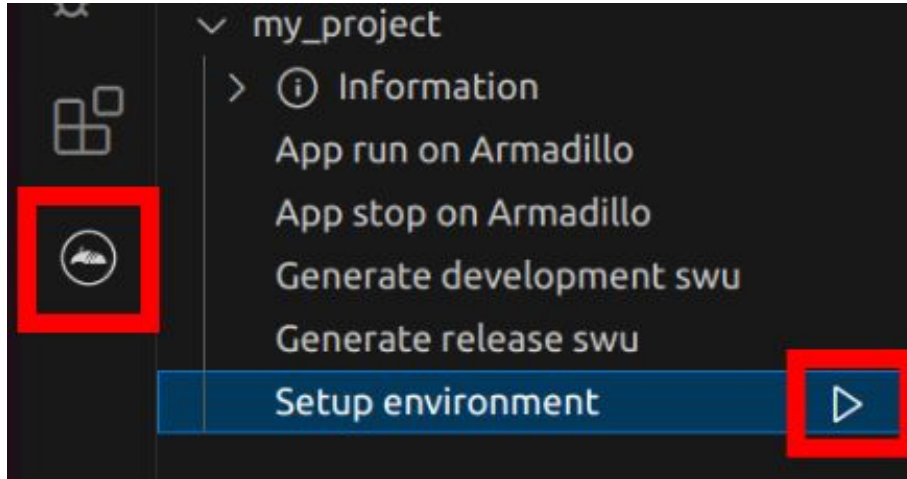


図 3.162 VS Code で初期設定を行う

選択すると、VS Code の下部に以下のようなターミナルが表示されます。

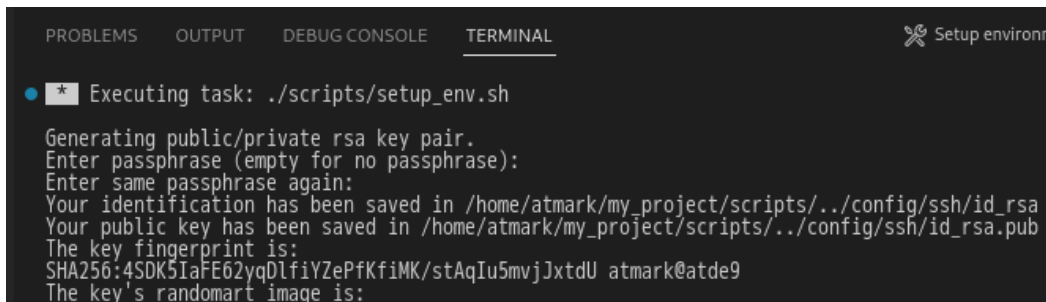


図 3.163 VS Code のターミナル

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 3.164 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 ❶ 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。

3.13.3.4. 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 ヘインストールするため、事前に「5.4.1. 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。

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

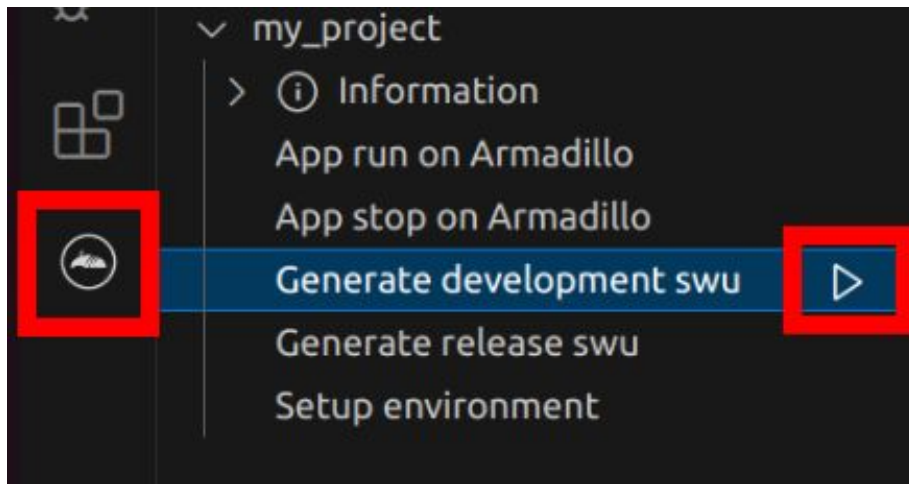


図 3.165 VS Code でコンテナイメージの作成を行う

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。VS Code 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。

```

コンテナイメージを ./swu/my_project.tar に保存しました。
./swu/app.desc のバージョンを 1 から 2 に変更しました。
./development.swu を作成しました。
次は Armadillo に ./development.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 3.166 コンテナイメージの作成完了

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

3.13.3.5. Python アプリケーションに BLE パッケージをインストールする

Python アプリケーションの場合は、アプリケーションから BLE を使用するために必要なパッケージを VS Code からインストールすることができます。

左ペインの [my_project] から [external packages] を開き [bleak] の右にある+ をクリックするとインストールされます。

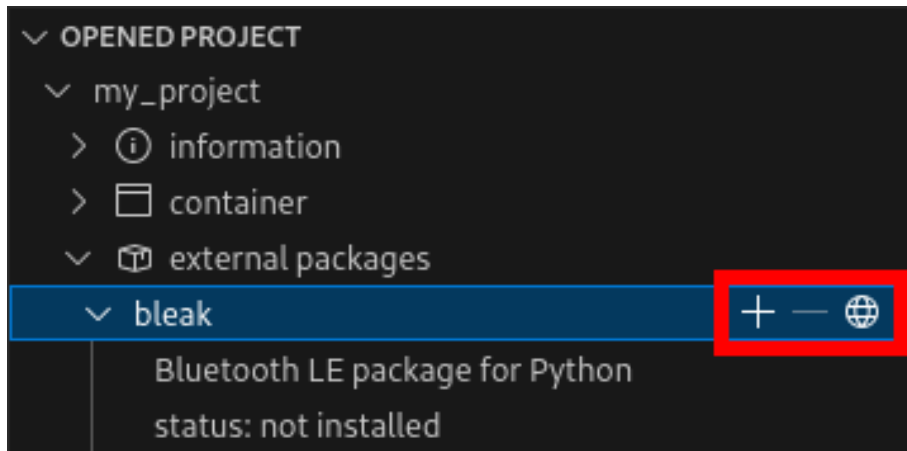


図 3.167 BLE パッケージをインストールする

すでにインストール済みの状態で - をクリックするとアンインストールされます。一番右にある丸アイコンをクリックすると Web ブラウザで bleak パッケージの API リファレンスページを開きます。



BLE パッケージのインストールは ABOSDE のバージョン 1.8.4 以降で、かつ 2024 年 7 月 24 日以降に「3.13.2.1. プロジェクトの作成」の手順で新たに作成したプロジェクトで使用できるようになります。

3.13.4. コンテナのディストリビューション

使用するコンテナのディストリビューションは以下のとおりです。

ディストリビューション ・ debian:bullseye-slim

3.13.5. Armadillo に転送するディレクトリ及びファイル

コンテナイメージ以外に、以下に示すディレクトリやファイルを Armadillo に転送します。ここでは、プロジェクト名は my_project としています。

Armadillo に転送するディレクトリ及びファイル ・ my_project/app/src

3.13.6. コンテナ内のファイル一覧表示

「図 3.168. コンテナ内のファイル一覧を表示するタブ」の赤枠で囲われているタブをクリックすることで、development.swu または「3.13.9. リリース版のビルド」で作成される release.swu に含まれるコンテナ内のファイルおよびディレクトリを表示します。

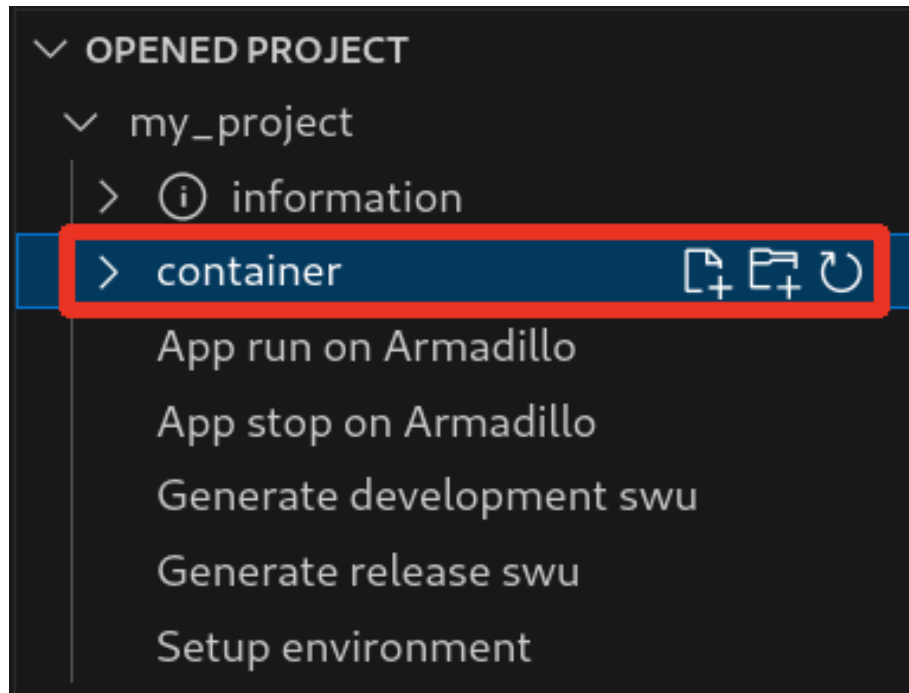


図 3.168 コンテナ内のファイル一覧を表示するタブ

クリック後の表示例を「図 3.169. コンテナ内のファイル一覧の例」に示します。

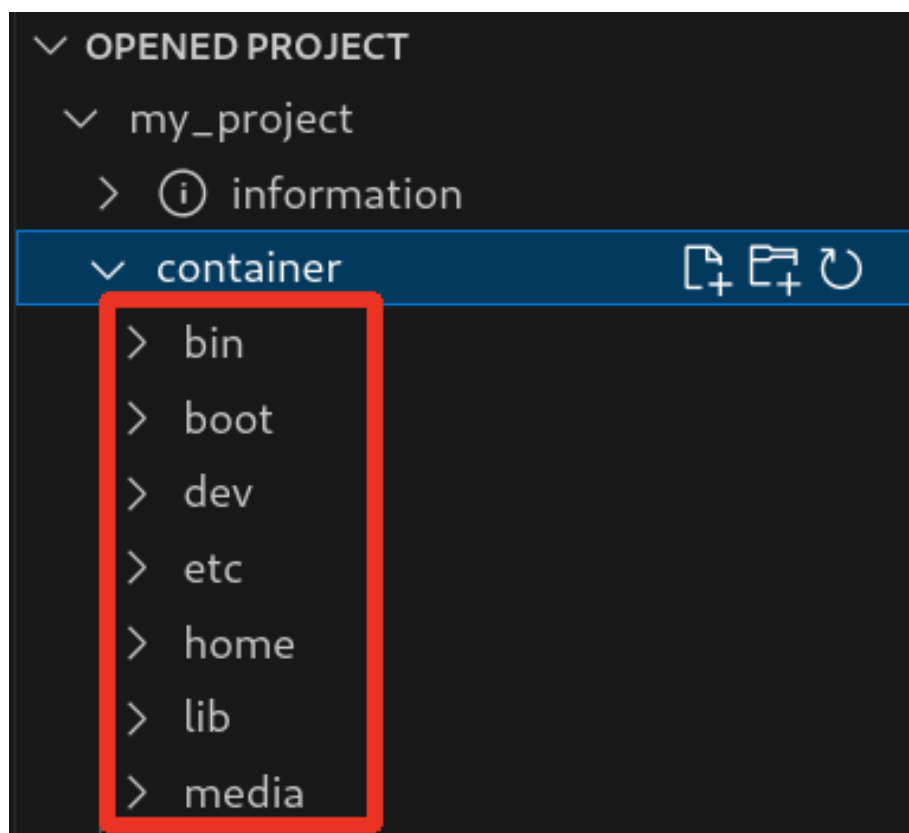


図 3.169 コンテナ内のファイル一覧の例

コンテナ内のファイル一覧は [Generate development swu] または [Generate release swu] を実行することで ATDE 上に作成されるコンテナイメージから取得しています。

そのため、[Generate development swu] または [Generate release swu] を実行していない場合はコンテナ内のファイル一覧は表示されません。その場合は [Generate development swu] または [Generate release swu] を先に実行してください。



この機能を使用するにあたり、ATDE 上でプロジェクトのコンテナイメージからコンテナを作成します。

コンテナ名は「プロジェクト名-abosde」を使用します。例えば、プロジェクト名が my_project の場合、コンテナ名は「my_project-abosde」になります。

ユーザー自身で同名のコンテナを既に作成していた場合、そのコンテナはこの機能を使用時に削除されます。



コンテナ内のファイル一覧には、ファイルおよびディレクトリのみを表示しています。シンボリックリンク、特殊デバイスファイルなどは表示していません。

3.13.6.1. resources ディレクトリについて

「図 3.170. resources ディレクトリ」に示すように ATDE 上のプロジェクトディレクトリには container/resources ディレクトリがあります。

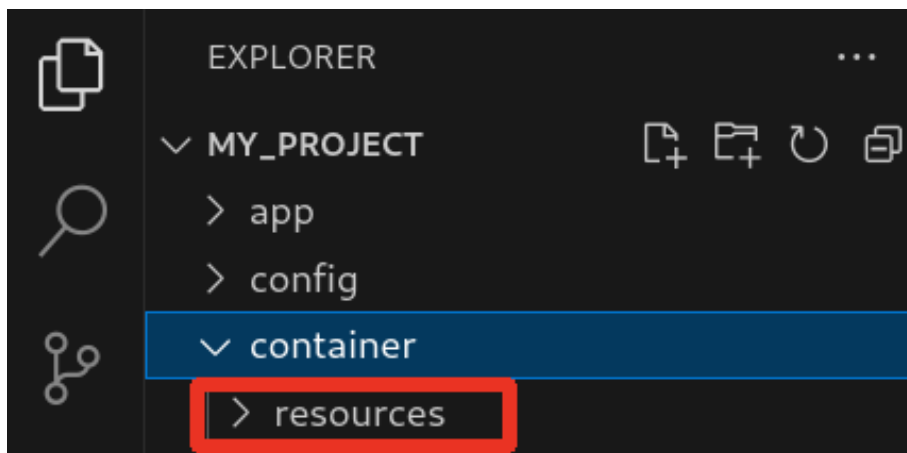


図 3.170 resources ディレクトリ

container/resources ディレクトリ下に、コンテナ内と同じパスでファイルまたはディレクトリを配置することで、それらは [Generate development swu] または [Generate release swu] を実行時にコンテナ内にコピーされます。

例えば、コンテナ内にある /etc/adduser.conf を上書きする場合は、編集した adduser.conf ファイルをプロジェクトディレクトリにある container/resources/etc/adduser.conf に配置してください。

プロジェクトディレクトリにある **container/resources** 下のファイルおよびディレクトリを操作する方法は以下の 2 通りがあります。

- ・ エクスプローラーを使用する
- ・ ABOSDE のコンテナ内のファイル一覧表示機能を使用する

ABOSDE のコンテナ内のファイル一覧表示機能を使用することで、視覚的にファイル構成や、差分があるファイルを把握しながら操作可能です。以降に詳細を説明します。

3.13.6.2. コンテナ内のファイル一覧の再表示

「図 3.168. コンテナ内のファイル一覧を表示するタブ」の赤枠で囲われているボタンをクリックすることで、コンテナ内のファイル一覧を再表示します。

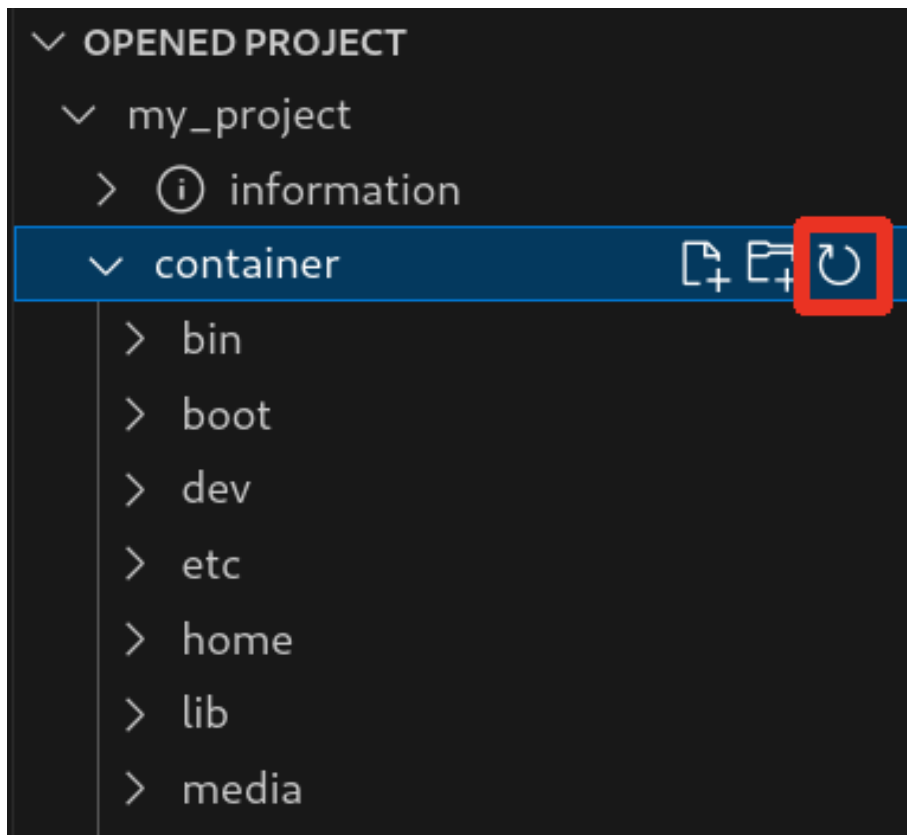


図 3.171 コンテナ内のファイル一覧を再表示するボタン

3.13.6.3. container/resources 下にファイルおよびフォルダーを作成

「図 3.172. container/resources 下にファイルを追加するボタン」の赤枠で囲われている表記のボタンをクリックすることで、プロジェクトディレクトリにある **container/resources** 下にファイルを追加することが可能です。

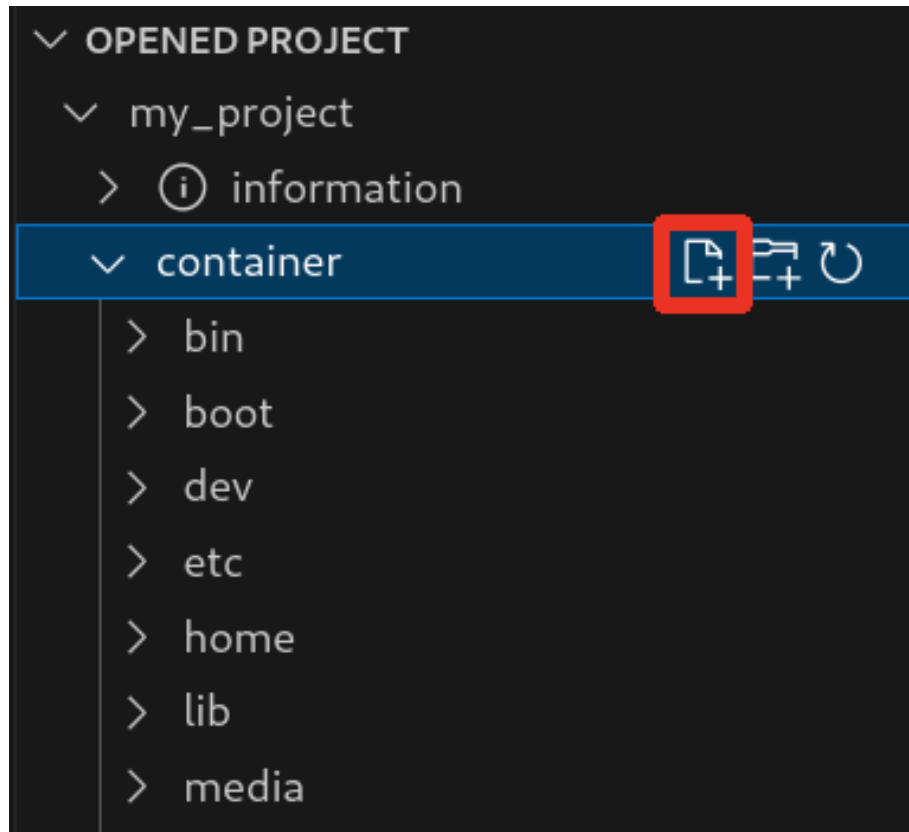


図 3.172 container/resources 下にファイルを追加するボタン

「図 3.173. ファイル名を入力」に示すように、コマンドパレットが表示されますのでファイル名を入力してください。

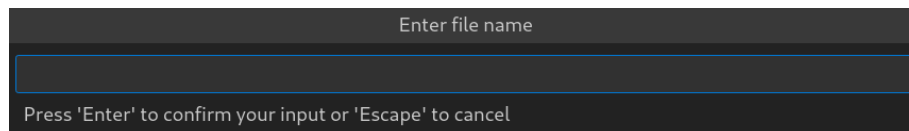


図 3.173 ファイル名を入力

例として、「add_file」というファイル名を入力したとします。

「図 3.174. 追加されたファイルの表示」に示すように、追加したファイルには「A」というマークが表示されます。

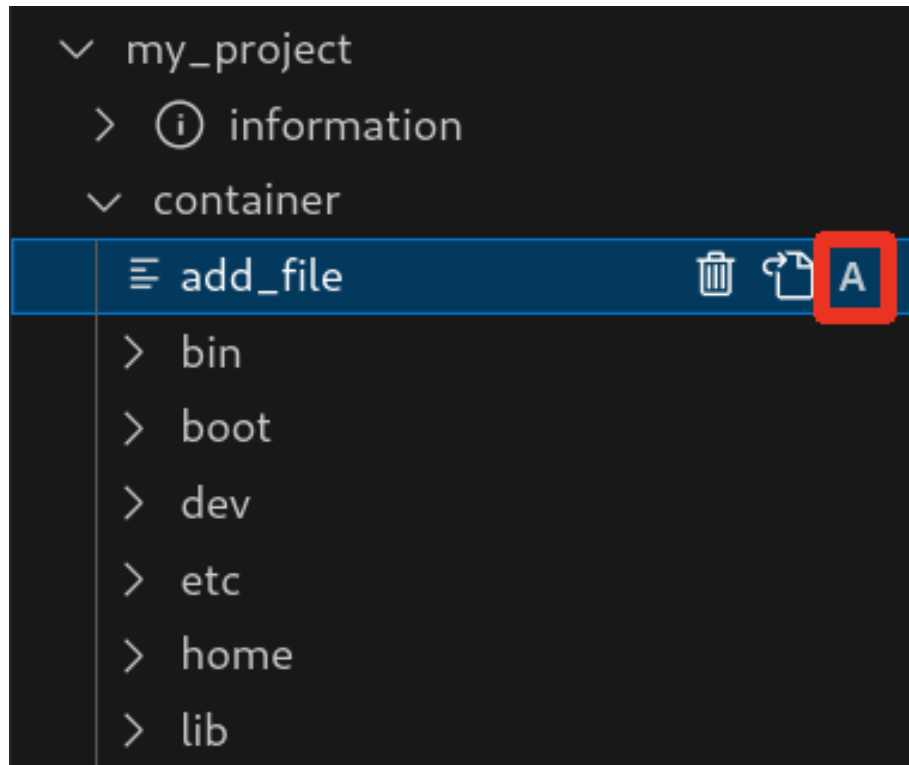


図 3.174 追加されたファイルの表示

また、「図 3.175. container/resources 下にフォルダーを追加するボタン」の赤枠で囲われている表記のボタンをクリックすることで、ファイルの追加と同様の操作でディレクトリを追加することが可能です。

追加したディレクトリも同様に "A" というマークが表示されます。

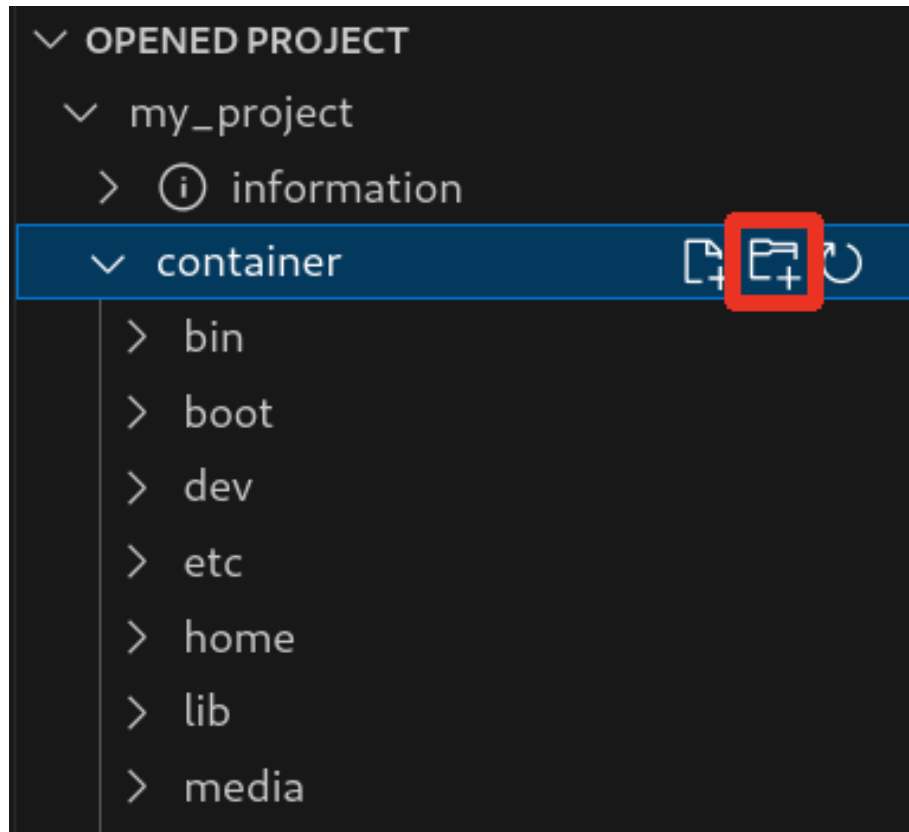


図 3.175 container/resources 下にフォルダーを追加するボタン

3.13.6.4. container/resources 下にあるファイルを開く

「図 3.176. container/resources 下にあるファイルを開くボタン」の赤枠で囲われている表記のボタンをクリックすることで、プロジェクトディレクトリにある container/resources 下のファイルをエディタに表示することができます。

この例では、プロジェクトディレクトリにある container/resources 下の add_file をエディタに表示します。

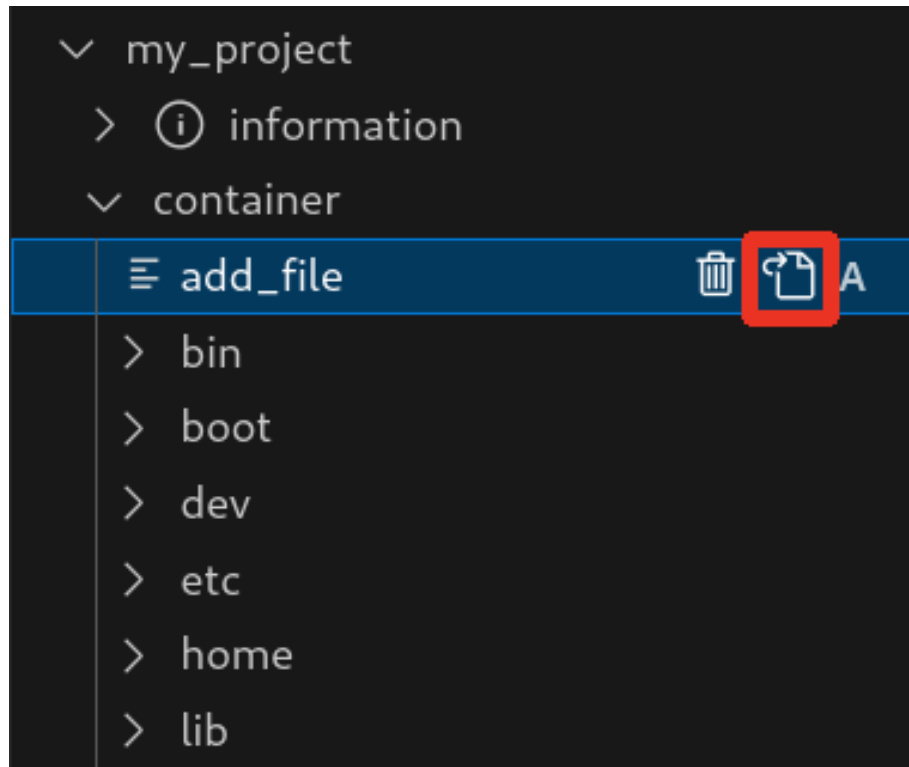


図 3.176 container/resources 下にあるファイルを開くボタン

3.13.6.5. container/resources 下にあるファイルおよびフォルダーの削除

「図 3.176. container/resources 下にあるファイルを開くボタン」の赤枠で囲われている表記のボタンをクリックすることで、container/resources 下にあるファイルを削除することができます。

この例では、プロジェクトディレクトリにある container/resources 下の add_file を削除します。

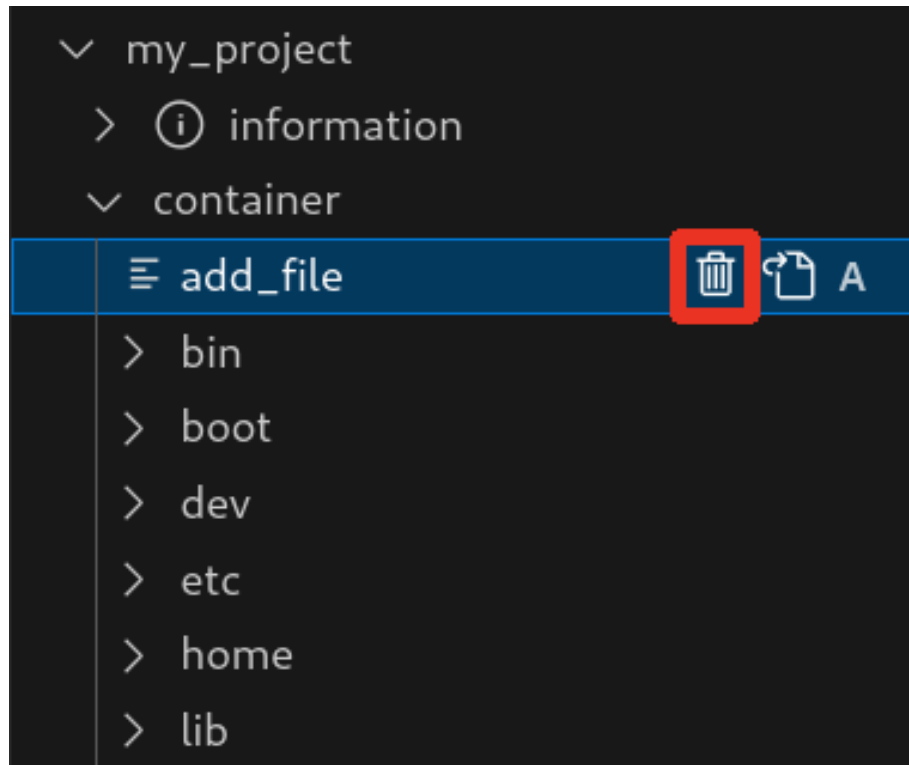


図 3.177 container/resources 下にあるファイルを削除するボタン

ディレクトリも同様に「図 3.176. container/resources 下にあるファイルを開くボタン」の赤枠で囲われている表記のボタンをクリックすることで削除することができます。

3.13.6.6. コンテナ内のファイルを container/resources 下に保存

「図 3.178. コンテナ内のファイルを container/resources 下に保存するボタン」の赤枠で囲われている表記のボタンをクリックすることで、コンテナ内にあるファイルをプロジェクトディレクトリにある container/resources 下に保存します。

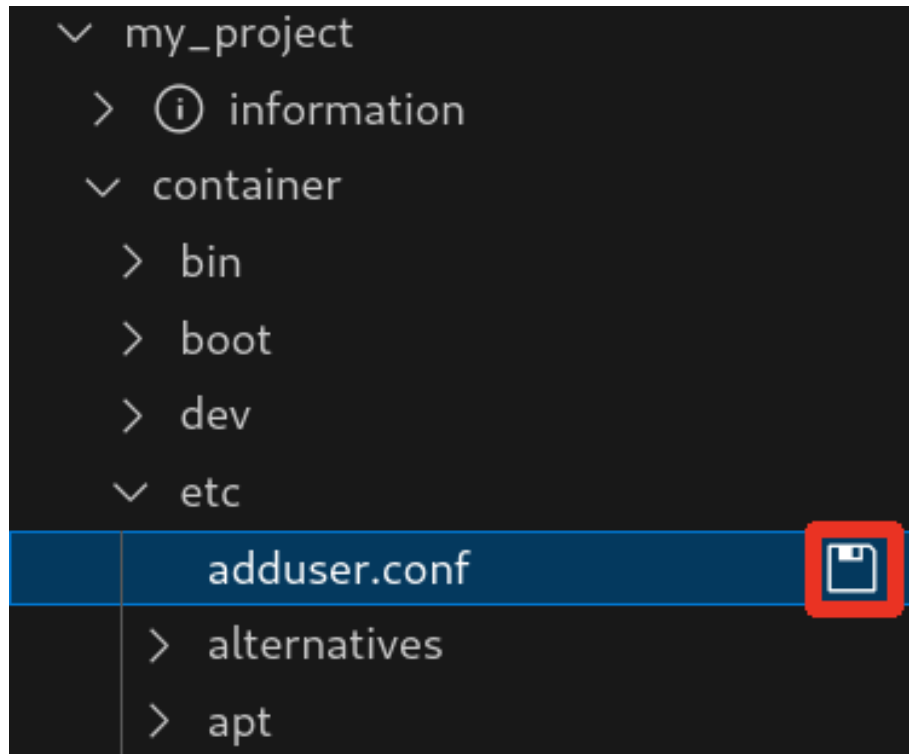


図 3.178 コンテナ内のファイルを container/resources 下に保存するボタン

ファイルが container/resources 下に保存されると、「図 3.179. 編集前のファイルを示すマーク」に示すように、ファイル名の右側に "U" のマークが表示されます。

"U" のマークはプロジェクトディレクトリにある container/resources 下のファイルとコンテナ内にあるファイルの内容が同一であることを示します。

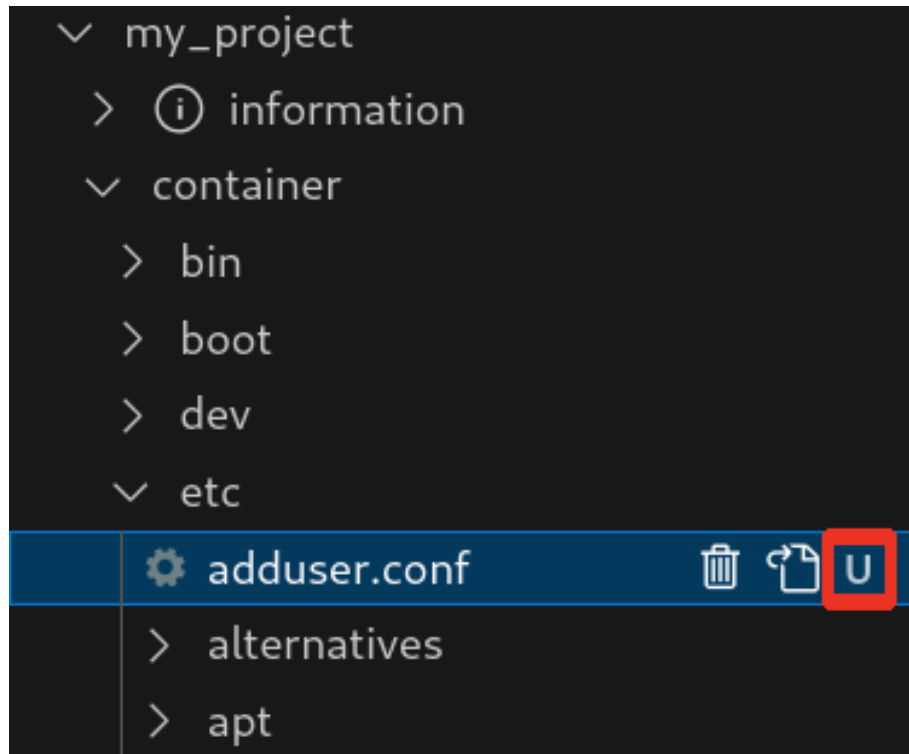


図 3.179 編集前のファイルを示すマーク

container/resources 下にあるファイルを編集して再表示すると、「図 3.180. 編集後のファイルを示すマーク」に示すように、ファイル名の右側に "M" のマークが表示されます。

"M" のマークはプロジェクトディレクトリにある container/resources 下のファイルとコンテナ内にあるファイルの内容に差があることを示します。

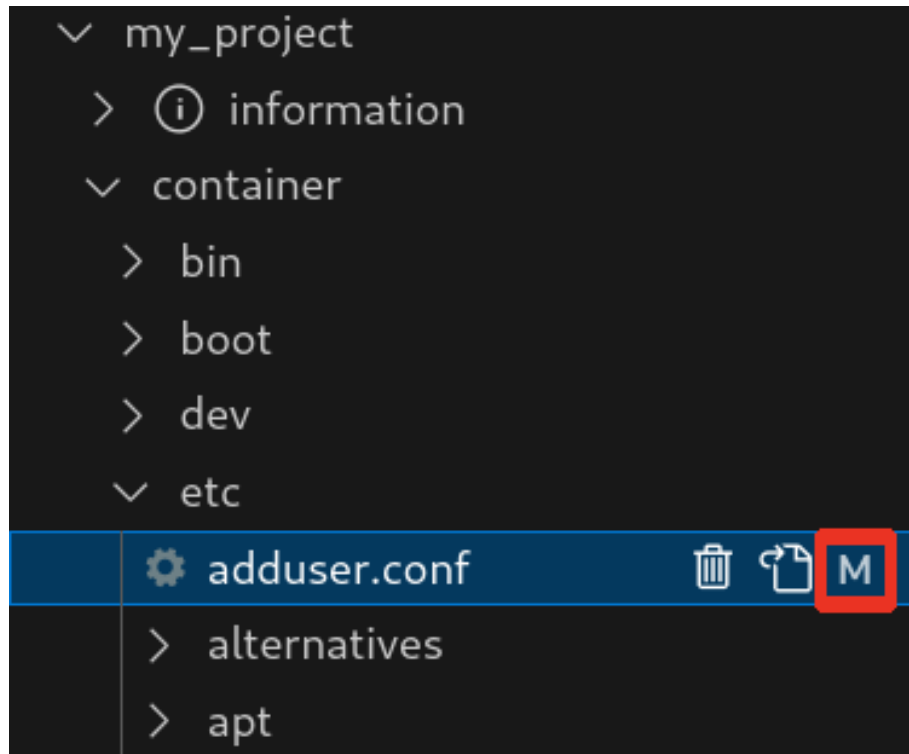


図 3.180 編集後のファイルを示すマーク

3.13.6.7. エラー表示

container/resources 下とコンテナ内にあるファイルまたはディレクトリを比較して、同名でかつファイルの種類が異なる場合、「図 3.181. コンテナ内にコピーされないことを示すマーク」に示すように、ファイル名の右側に "E" のマークが表示されます。

"E" のマークが表示された場合、そのファイルまたはディレクトリは [Generate development swu] または [Generate release swu] を実行してもコンテナにコピーされません。

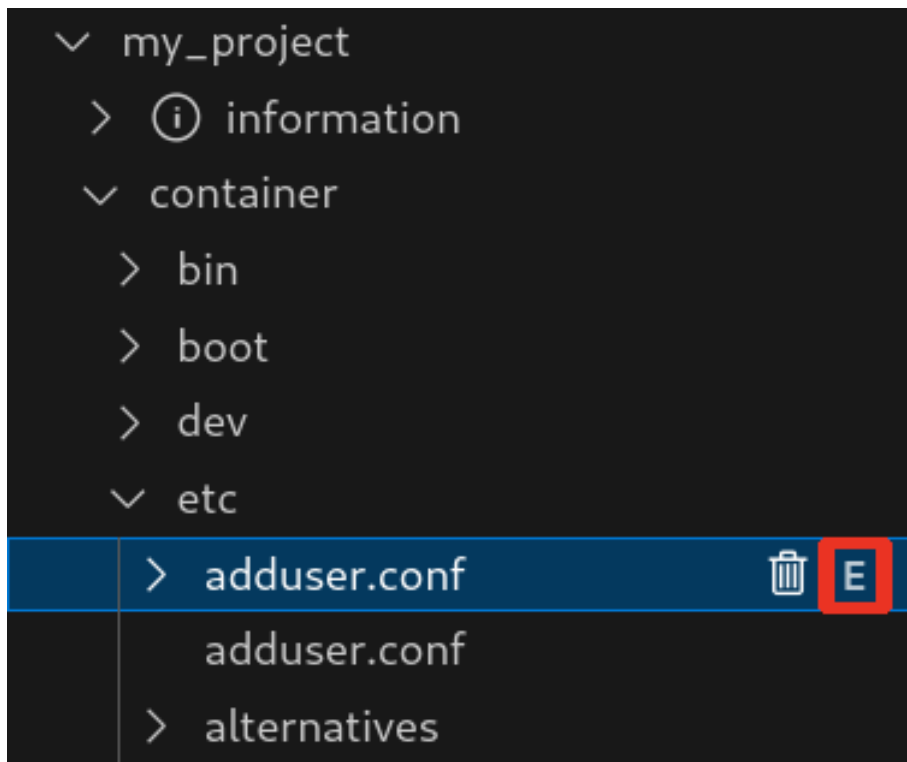


図 3.181 コンテナ内にコピーされないことを示すマーク

3.13.7. Armadillo 上でのセットアップ

3.13.7.1. 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

「3.13.3.4. 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 `development.swu` を「3.3.3.6. SWU イメージのインストール」 を参照して Armadillo へ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。

3.13.7.2. ssh 接続に使用する IP アドレスの設定

VS Code 上で ABOSDE(Armadillo Base OS Development Environment) から、ABOS Web が動作している Armadillo の一覧を確認し、指定した Armadillo の IP アドレスを ssh 接続に使用することができます。ただし、ATDE のネットワークを NAT に設定している場合は Armadillo がリストに表示されません。

「図 3.182. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする」 の赤枠で囲われているボタンをクリックすることで、ローカルネットワーク上で ABOS Web が実行されている Armadillo をスキャンすることができます。

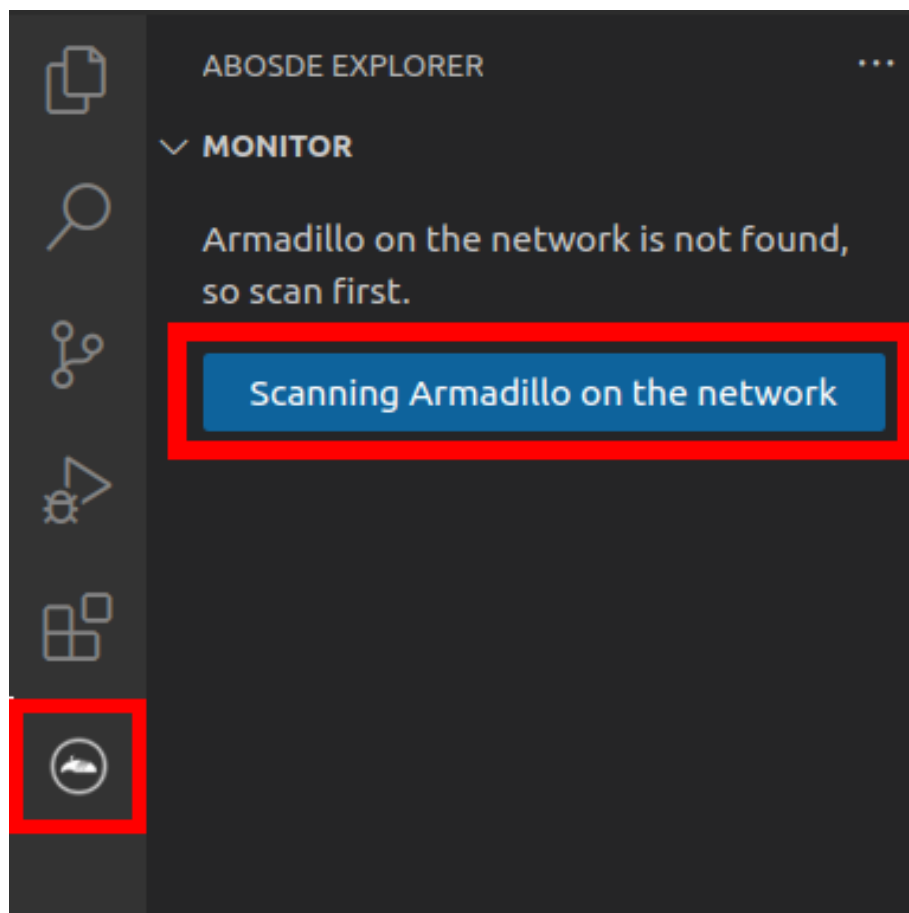


図 3.182 ABOSDE で ローカルネットワーク上の Armadillo をスキャンする

「図 3.183. ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する」の赤枠で囲われているマークをクリックすることで、指定した Armadillo の IP アドレスを ssh 接続に使用する IP アドレスに設定することができます。

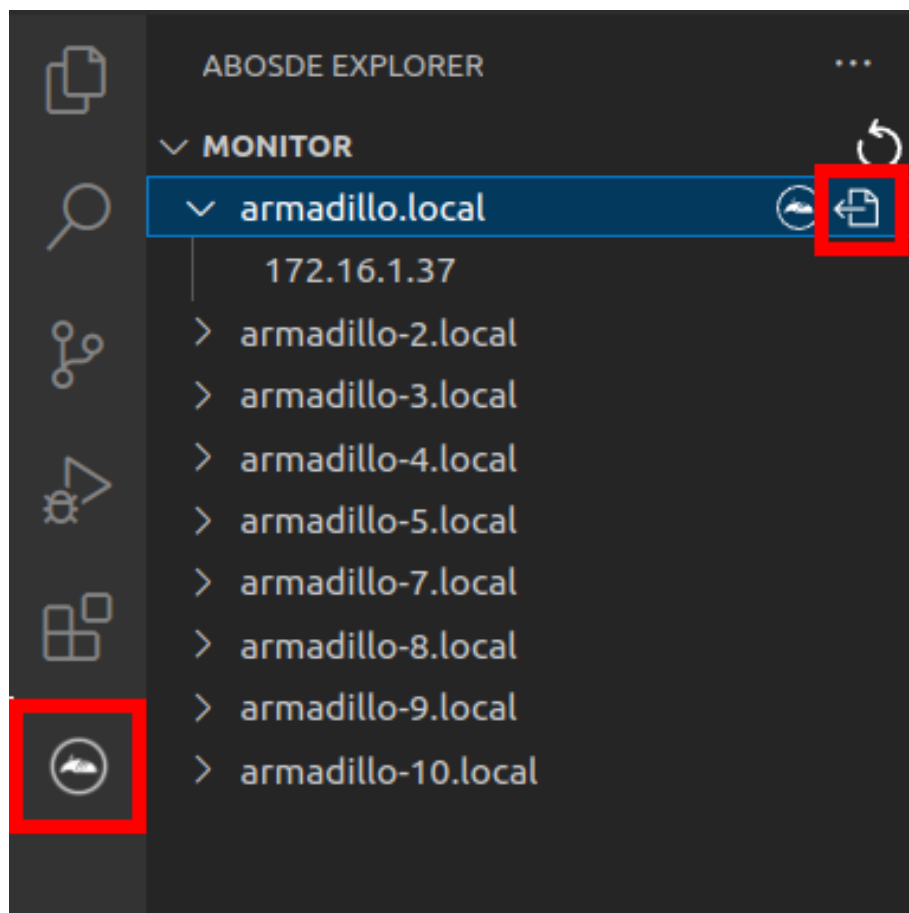


図 3.183 ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する

「図 3.184. ABOSDE に表示されている Armadillo を更新する」の赤枠で囲われているマークをクリックすることで、ABOSDE に表示されている Armadillo を更新することができます。

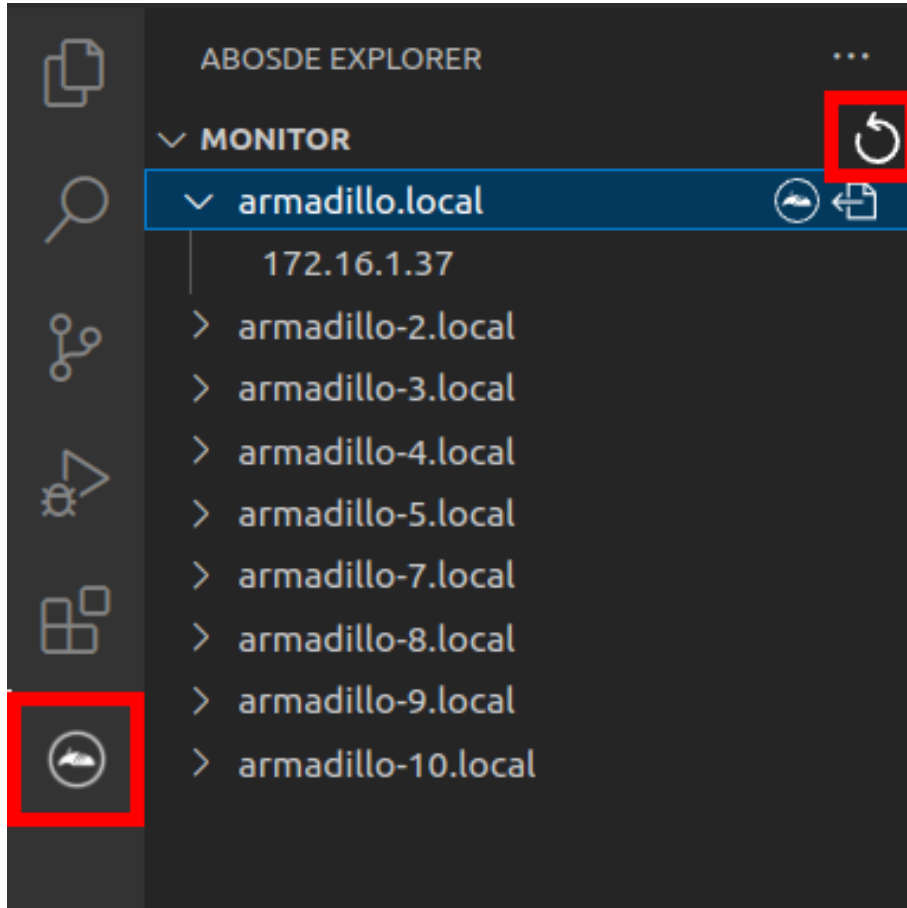


図 3.184 ABOSDE に表示されている Armadillo を更新する

ATDE のネットワークを NAT に設定している場合や、ABOS Web を起動していない場合等、ABOSDE のリストに Armadillo が表示されない場合は、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 x.x.x.x ❶
  User root
  IdentityFile ${HOME}/.ssh/id_ed25519_vscode
  UserKnownHostsFile config/ssh_known_hosts
  StrictHostKeyChecking accept-new
```

図 3.185 ssh_config を編集する

- ❶ 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。

3.13.7.3. アプリケーションの実行

VS Code 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。

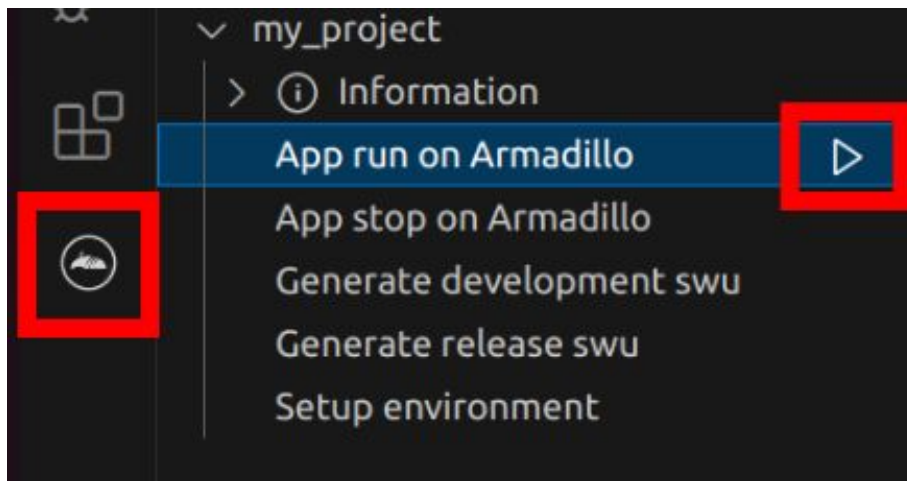


図 3.186 Armadillo 上でアプリケーションを実行する

VS Code 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 3.187 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 VS Code の左ペインの [my_project] から [App stop on Armadillo] を実行してください。

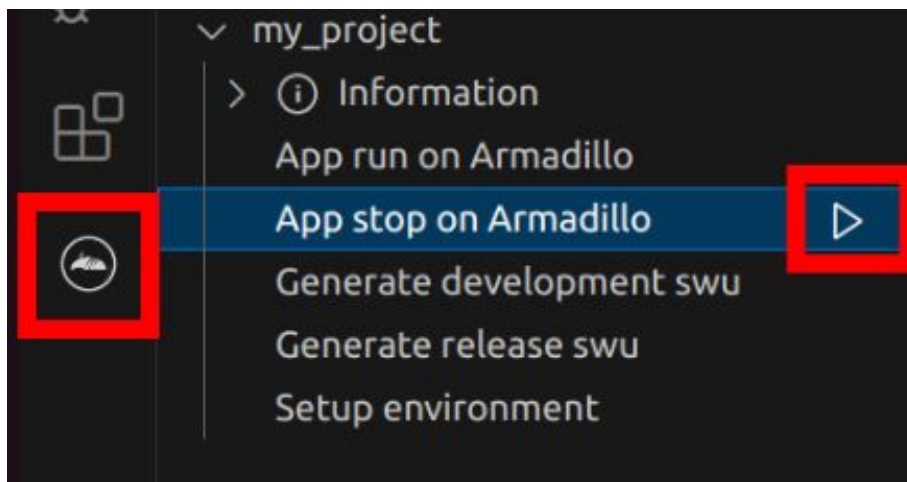


図 3.188 アプリケーションを終了する

3.13.8. SBOM 生成に関する設定

SWU イメージ作成時に、同時に SBOM を生成することができます。詳細は「3.15. SBOM 生成に関わる設定を行う」を参照してください。

3.13.9. 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。

VS Code 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「5.4.1. 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。

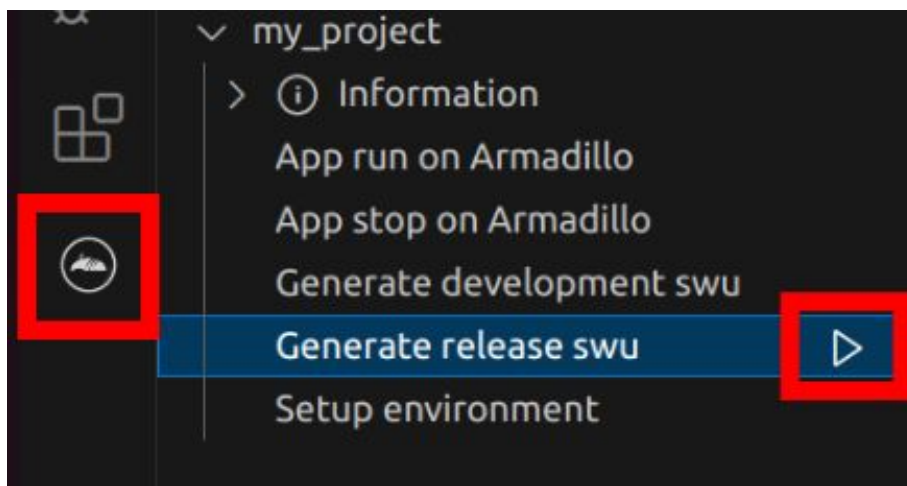


図 3.189 リリース版をビルドする



リリース版の SWU イメージには、開発用の機能は含まれていません。このため、リリース版の SWU イメージをインストールした Armadillo では、[App run on Armadillo] を使用したリモート実行は使用できません。

3.13.10. 製品への書き込み

作成した SWU イメージは my_project 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「3.3.3.6. SWU 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。

3.13.11. Armadillo 上のコンテナイメージの削除

「6.2.3. コンテナとコンテナに関連するデータを削除する」を参照してください。

3.14. C 言語によるアプリケーションの開発

ここでは C 言語によるアプリケーション開発の方法を紹介します。

C 言語によるアプリケーション開発は下記に当てはまるユーザーを対象としています。

- ・ 既存の C 言語によって開発されたアプリケーションを Armadillo で動作させたい
- ・ C 言語でないと実現できないアプリケーションを開発したい

上記に当てはまらず、開発するアプリケーションがシェルスクリプトまたは Python で実現可能であるならば、「3.13. CUI アプリケーションの開発」を参照してください。

3.14.1. C 言語によるアプリケーション開発の流れ

Armadillo 向けに C 言語による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。

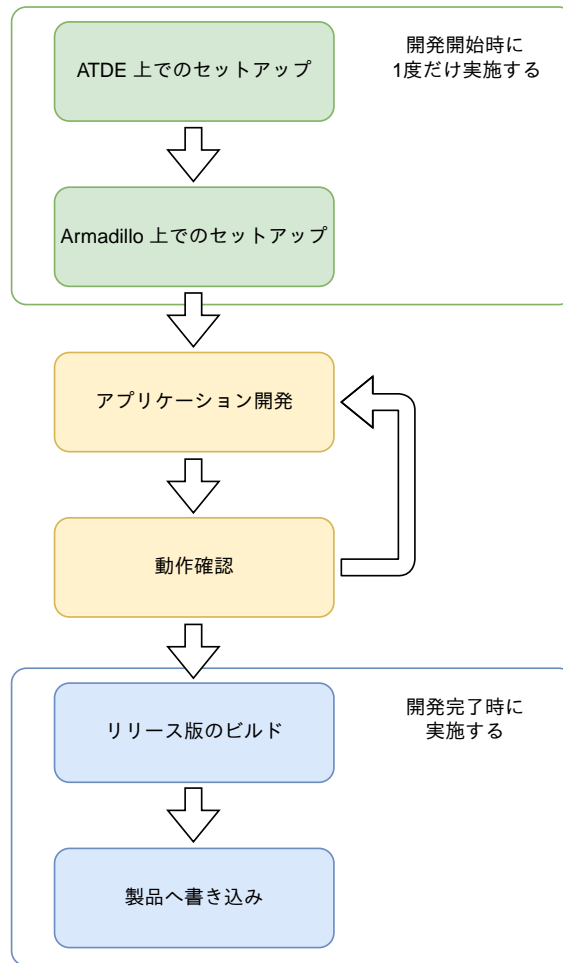


図 3.190 C 言語によるアプリケーション開発の流れ

3.14.2. 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「3.1. 開発の準備」を参照して ATDE 及び、VS Code のセットアップを完了してください。

3.14.2.1. プロジェクトの作成

VS Code の左ペインの [A600] から [C 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。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。この操作により、選択した保存先に、入力したプロジェクト名と同名のディレクトリが作成されます。

また、ここでは次のように設定しています。

- ・ 保存先 : ホームディレクトリ
- ・ プロジェクト名 : my_project

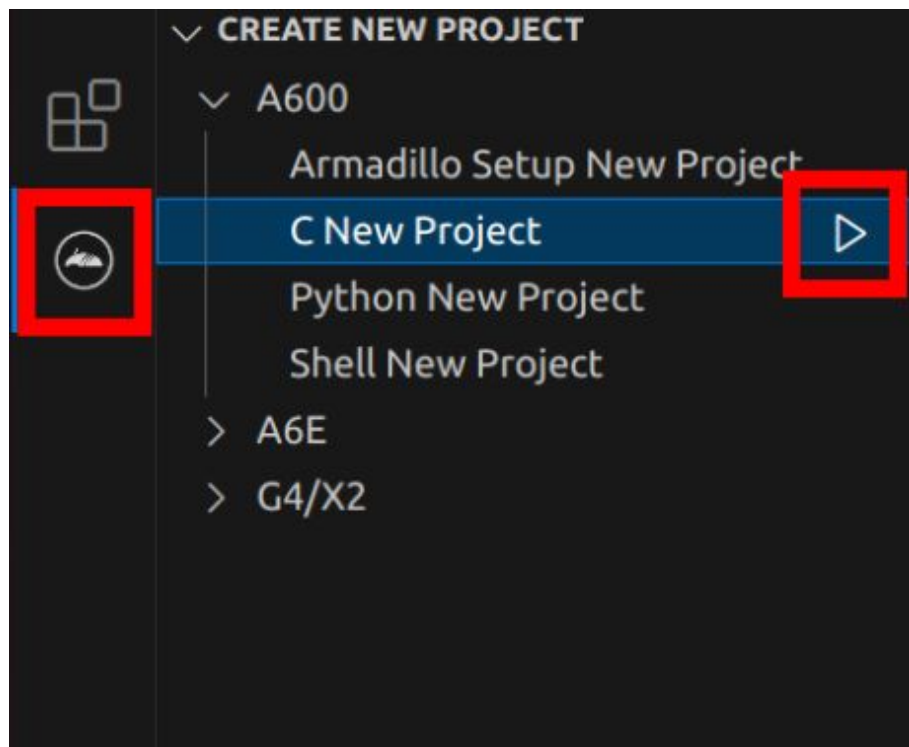


図 3.191 プロジェクトを作成する

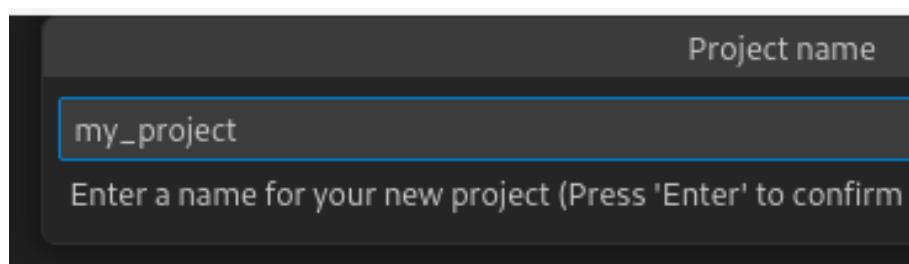


図 3.192 プロジェクト名を入力する

3.14.3. アプリケーション開発

3.14.3.1. VS Code 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 VS Code を起動します。

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

図 3.193 VS Code で my_project を起動する

3.14.3.2. ディレクトリ構成

プロジェクトには下記のディレクトリがあります。

- ・ **app** : 各ディレクトリの説明は以下の通りです。
 - ・ **src** : アプリケーションのソースファイル（拡張子が .c）と Makefile を配置してください。
 - ・ **build** : ここに配置した実行ファイルが Armadillo 上で実行されます。
 - ・ **lib** : 共有ライブラリの検索パスとしてこのディレクトリを指定しているので、ここに共有ライブラリ（拡張子が .so）を配置することができます。
- ・ **config** : 設定に関わるファイルが含まれるディレクトリです。
 - ・ **app.conf** : コンテナのコンフィグです。記載内容については「6.2.4. コンテナ起動設定ファイルを作成する」を参照してください。
 - ・ **app.desc** : SWU イメージを生成するための .desc ファイルです。記載内容については「6.4. mkswu の .desc ファイルを編集する」を参照してください。
 - ・ **ssh_config** : Armadillo への ssh 接続に使用します。「3.14.7.2. ssh 接続に使用する IP アドレスの設定」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が含まれるディレクトリです。
 - ・ **packages.txt** : このファイル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** : 直接編集することも可能です。

デフォルトのコンテナコンフィグ（app.conf）では C 言語の場合は build/main を実行しますので、リネームが必要な場合にコンテナのコンフィグも修正してください。

このサンプルアプリケーションは、CPU と SOC の温度を /vol_data/log/temp.txt に出力し、LED4 を点滅させます。

3.14.3.3. 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 VS Code を起動してください。

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

図 3.194 初期設定を行う

VS Code の左ペインの [my_project] から [Setup environment] を実行します。

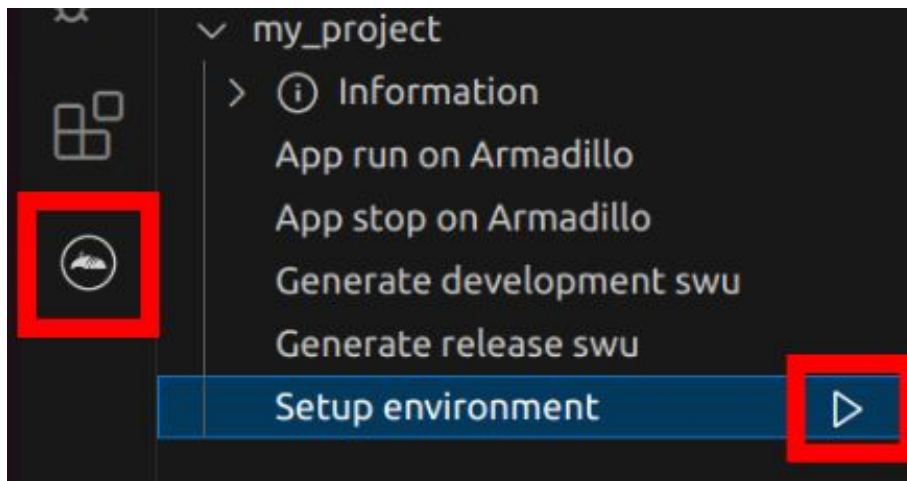


図 3.195 VS Code で初期設定を行う

選択すると、VS Code の下部に以下のようなターミナルが表示されます。

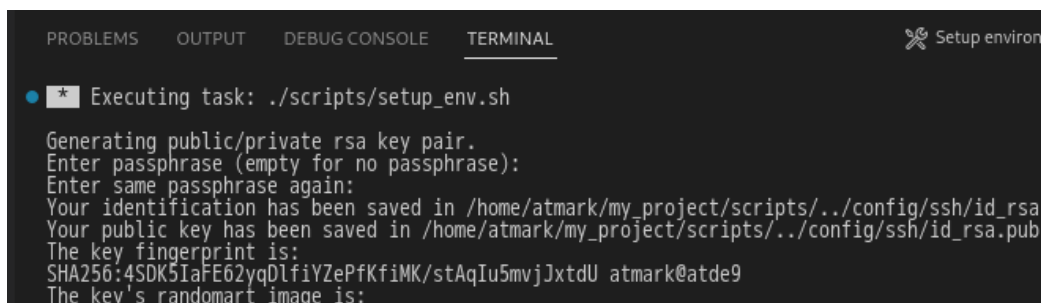


図 3.196 VS Code のターミナル

このターミナル上で以下のように入力してください。

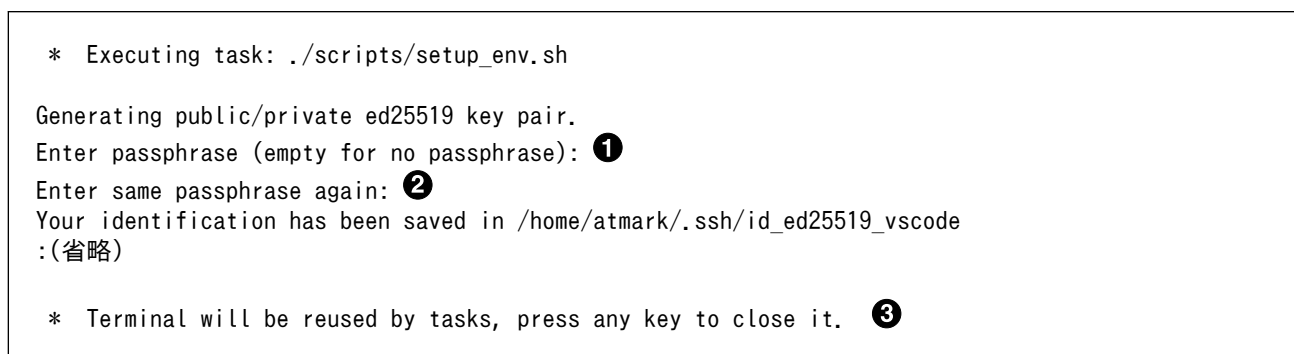


図 3.197 SSH 用の鍵を生成する

- 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。

3.14.3.4. packages.txt の書き方

ABOSDE ではコンテナイメージにパッケージをインストールするために container ディレクトリにある `packages.txt` を使用します。`packages.txt` に記載されているパッケージは "apt install" コマンドによってコンテナイメージにインストールされます。

C 言語による開発の場合、`packages.txt` に `[build]` というラベルを記載することで、ビルド時のみに使用するパッケージを指定することが出来ます。

「図 3.198. C 言語による開発における `packages.txt` の書き方」に C 言語による開発の場合における `packages.txt` の書き方の例を示します。ここでは、パッケージ名を `package_A`、`package_B`、`package_C` としています。

```
package_A
package_B

[build] ❶
package_C
```

図 3.198 C 言語による開発における `packages.txt` の書き方

❶ このラベル以降のパッケージはビルド時のみに使用されます。

上記の例の場合、Armadillo 上で実行される環境では `package_A`、`package_B` のみがインストールされ、`package_C` はインストールされません。

"`[build] package_C`" のように `[build]` の後に改行せずに、一行でパッケージ名を書くことは出来ませんのでご注意ください。

3.14.3.5. ABOSDE での開発における制約

Makefile は `app/src` 直下に配置してください。`app/src` 直下の Makefile を用いて `make` コマンドが実行されます。ABOSDE では `make` コマンドのみに対応しています。

`app/build` と `app/lib` 内のファイルが Armadillo に転送されますので、実行ファイルは `app/build`、共有ライブラリ（拡張子が `.so` ファイル）は `app/lib` に配置してください。

3.14.3.6. 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「5.4.1. 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。

コンテナイメージの作成、実行ファイルや共有ライブラリの作成および SWU イメージの作成も VS Code で行います。VS Code の左ペインの `[my_project]` から `[Generate development swu]` を実行します。

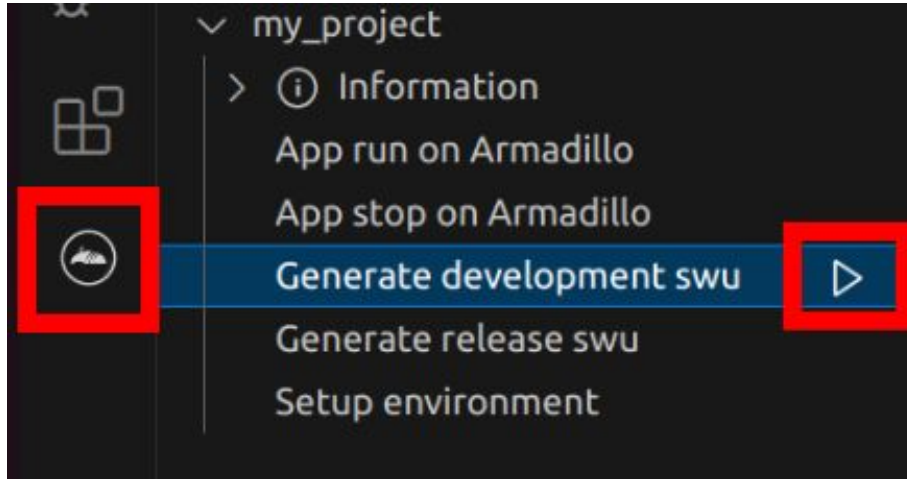


図 3.199 VS Code でコンテナイメージの作成を行う

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。VS Code 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。

```
コンテナイメージを ./swu/my_project.tar に保存しました。
./swu/app.desc のバージョンを 1 から 2 に変更しました。
./development.swu を作成しました。
次は Armadillo に ./development.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 3.200 コンテナイメージの作成完了

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

3.14.4. コンテナのディストリビューション

使用するコンテナのディストリビューションは以下のとおりです。

```
ディストリビューション ・ debian:bullseye-slim
```

3.14.5. コンテナ内のファイル一覧表示

「図 3.201. コンテナ内のファイル一覧を表示するタブ」の赤枠で囲われているタブをクリックすることで、development.swu または「3.14.9. リリース版のビルド」で作成される release.swu に含まれるコンテナ内のファイルおよびディレクトリを表示します。

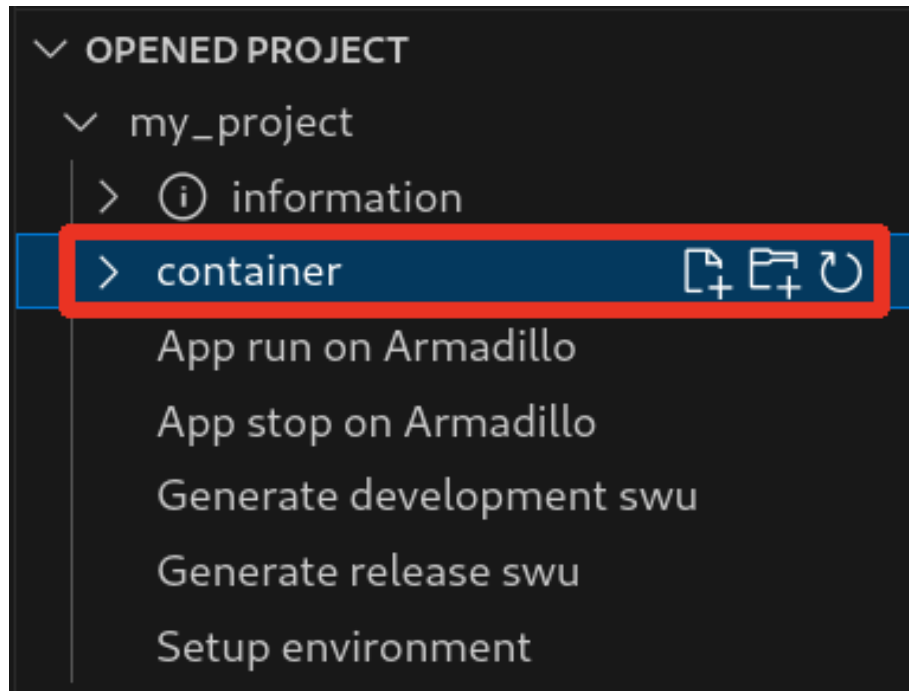


図 3.201 コンテナ内のファイル一覧を表示するタブ

クリック後の表示例を「図 3.202. コンテナ内のファイル一覧の例」に示します。

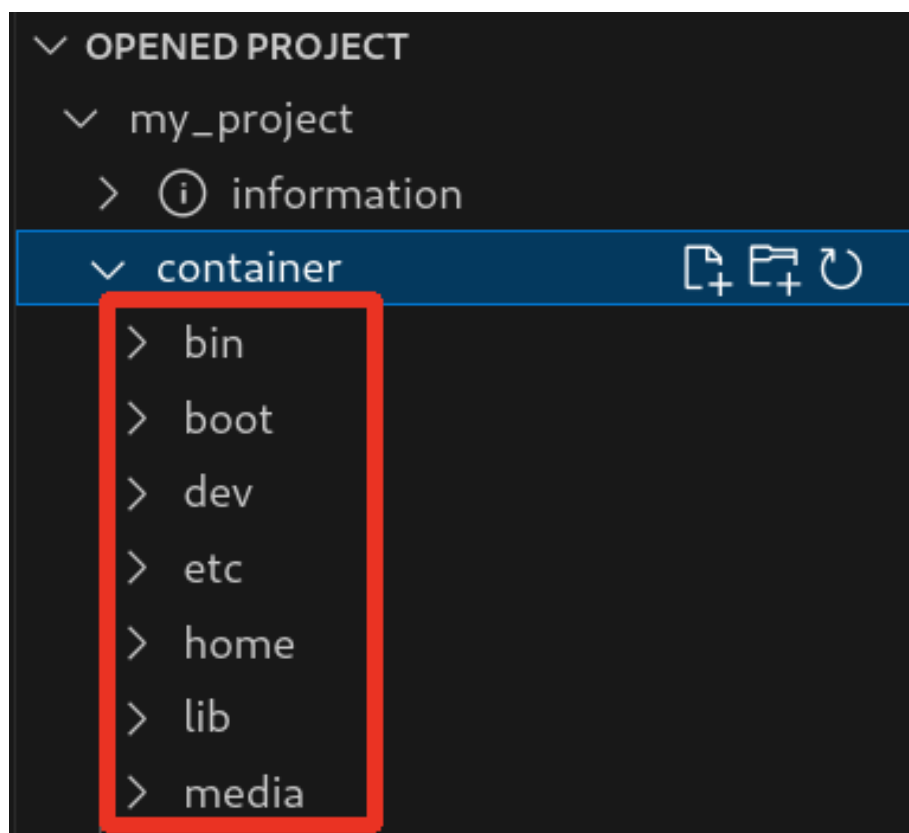


図 3.202 コンテナ内のファイル一覧の例

コンテナ内のファイル一覧は [Generate development swu] または [Generate release swu] を実行することで ATDE 上に作成されるコンテナイメージから取得しています。

そのため、[Generate development swu] または [Generate release swu] を実行していない場合はコンテナ内のファイル一覧は表示されません。その場合は [Generate development swu] または [Generate release swu] を先に実行してください。



この機能を使用するにあたり、ATDE 上でプロジェクトのコンテナイメージからコンテナを作成します。

コンテナ名は「プロジェクト名-abosde」を使用します。例えば、プロジェクト名が my_project の場合、コンテナ名は「my_project-abosde」になります。

ユーザー自身で同名のコンテナを既に作成していた場合、そのコンテナはこの機能を使用時に削除されます。



コンテナ内のファイル一覧には、ファイルおよびディレクトリのみを表示しています。シンボリックリンク、特殊デバイスファイルなどは表示していません。

3.14.5.1. resources ディレクトリについて

「図 3.203. resources ディレクトリ」に示すように ATDE 上のプロジェクトディレクトリには container/resources ディレクトリがあります。

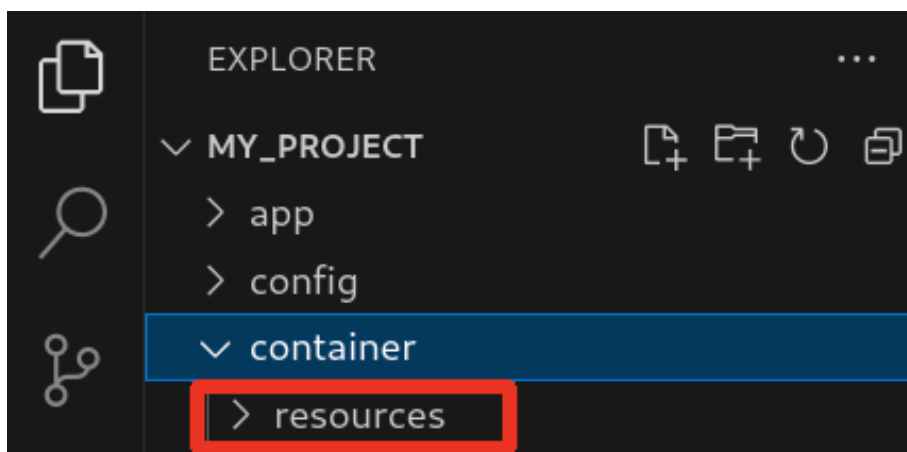


図 3.203 resources ディレクトリ

container/resources ディレクトリ下に、コンテナ内と同じパスでファイルまたはディレクトリを配置することで、それらは [Generate development swu] または [Generate release swu] を実行時にコンテナ内にコピーされます。

例えば、コンテナ内にある /etc/adduser.conf を上書きする場合は、編集した adduser.conf ファイルをプロジェクトディレクトリにある container/resources/etc/adduser.conf に配置してください。

プロジェクトディレクトリにある **container/resources** 下のファイルおよびディレクトリを操作する方法は以下の 2 通りがあります。

- ・ エクスプローラーを使用する
- ・ ABOSDE のコンテナ内のファイル一覧表示機能を使用する

ABOSDE のコンテナ内のファイル一覧表示機能を使用することで、視覚的にファイル構成や、差分があるファイルを把握しながら操作可能です。以降に詳細を説明します。

3.14.5.2. コンテナ内のファイル一覧の再表示

「図 3.201. コンテナ内のファイル一覧を表示するタブ」の赤枠で囲われているボタンをクリックすることで、コンテナ内のファイル一覧を再表示します。

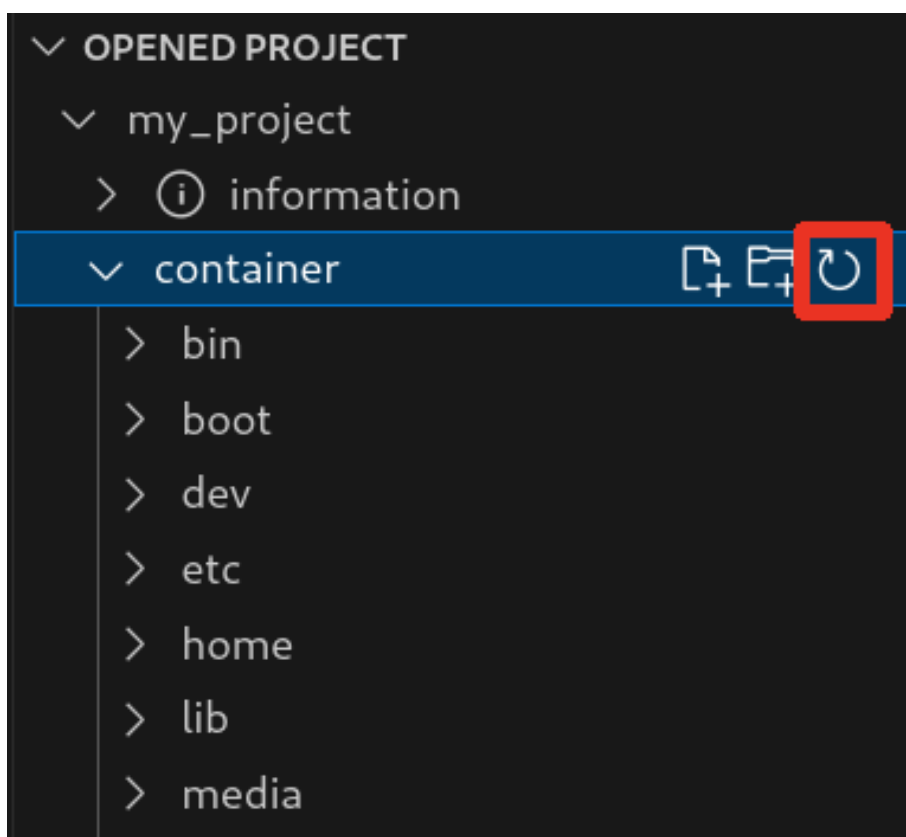


図 3.204 コンテナ内のファイル一覧を再表示するボタン

3.14.5.3. container/resources 下にファイルおよびフォルダーを作成

「図 3.205. container/resources 下にファイルを追加するボタン」の赤枠で囲われている表記のボタンをクリックすることで、プロジェクトディレクトリにある **container/resources** 下にファイルを追加することが可能です。

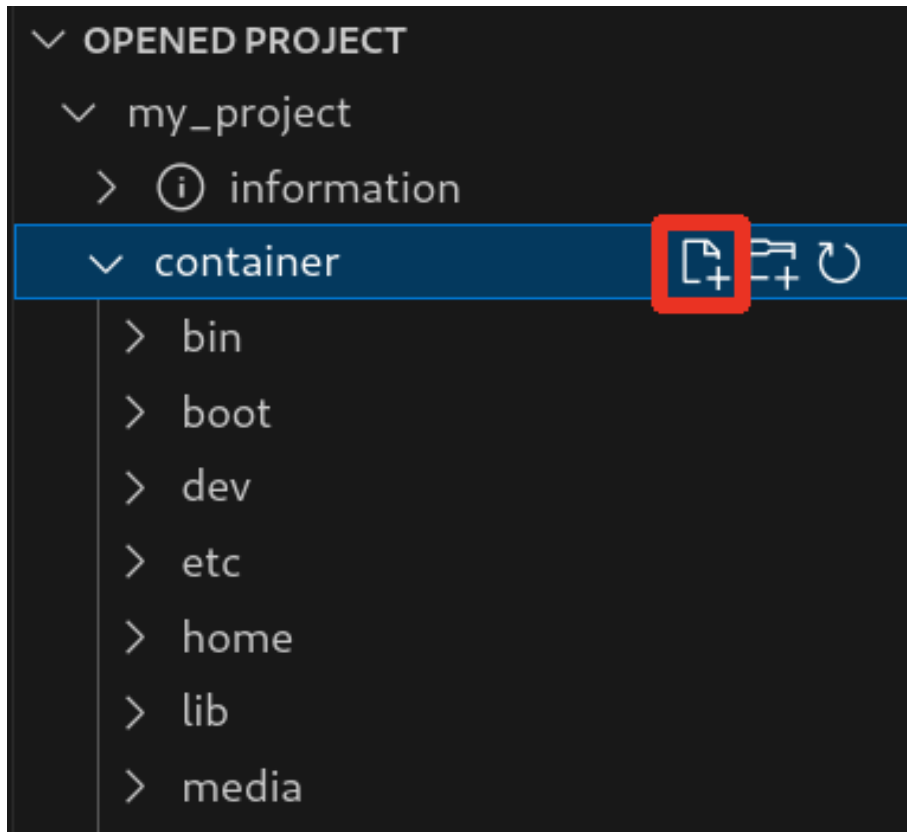


図 3.205 container/resources 下にファイルを追加するボタン

「図 3.206. ファイル名を入力」に示すように、コマンドパレットが表示されますのでファイル名を入力してください。

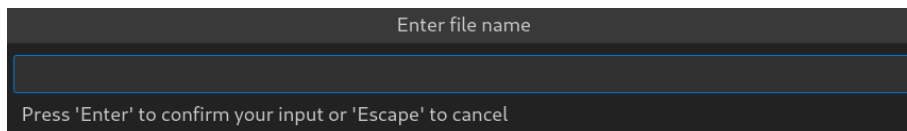


図 3.206 ファイル名を入力

例として、「add_file」というファイル名を入力したとします。「図 3.207. 追加されたファイルの表示」に示すように、追加したファイルには「A」というマークが表示されます。

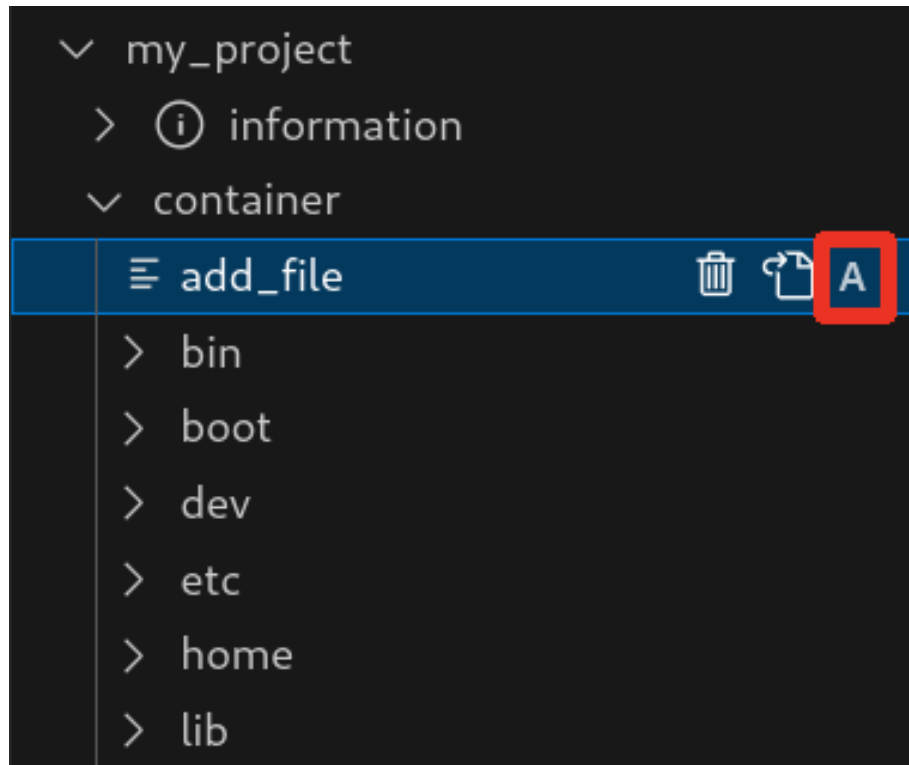


図 3.207 追加されたファイルの表示

また、「図 3.208. container/resources 下にフォルダーを追加するボタン」の赤枠で囲われている表記のボタンをクリックすることで、ファイルの追加と同様の操作でディレクトリを追加することが可能です。

追加したディレクトリも同様に "A" というマークが表示されます。

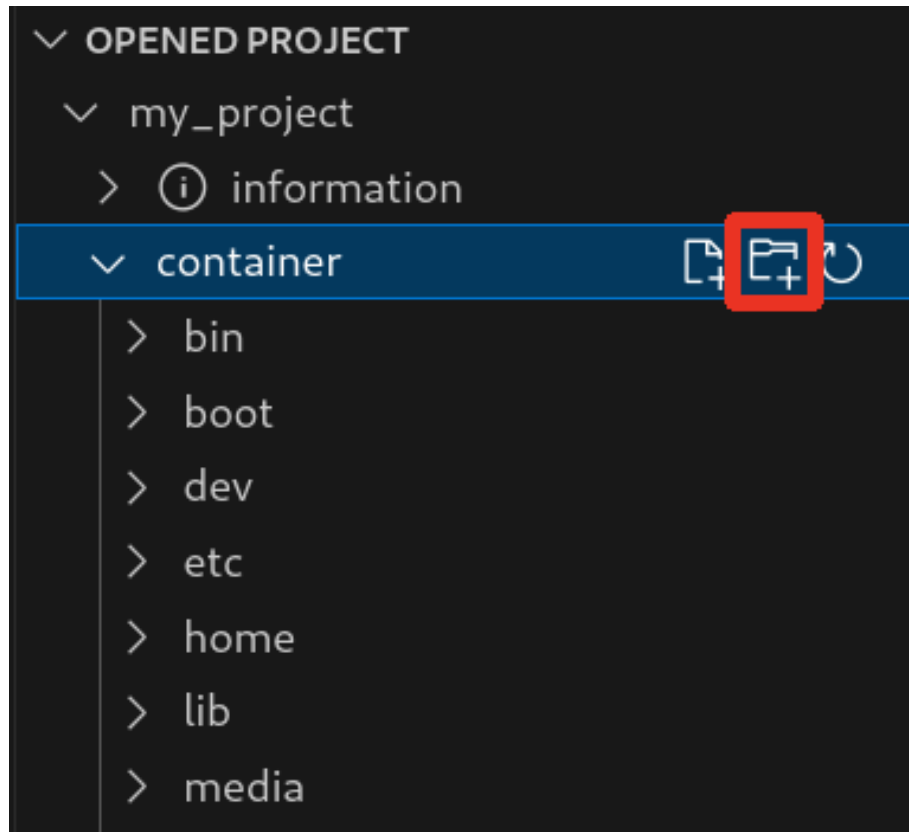


図 3.208 container/resources 下にフォルダーを追加するボタン

3.14.5.4. container/resources 下にあるファイルを開く

「図 3.209. container/resources 下にあるファイルを開くボタン」の赤枠で囲われている表記のボタンをクリックすることで、プロジェクトディレクトリにある container/resources 下のファイルをエディタに表示することができます。

この例では、プロジェクトディレクトリにある container/resources 下の add_file をエディタに表示します。

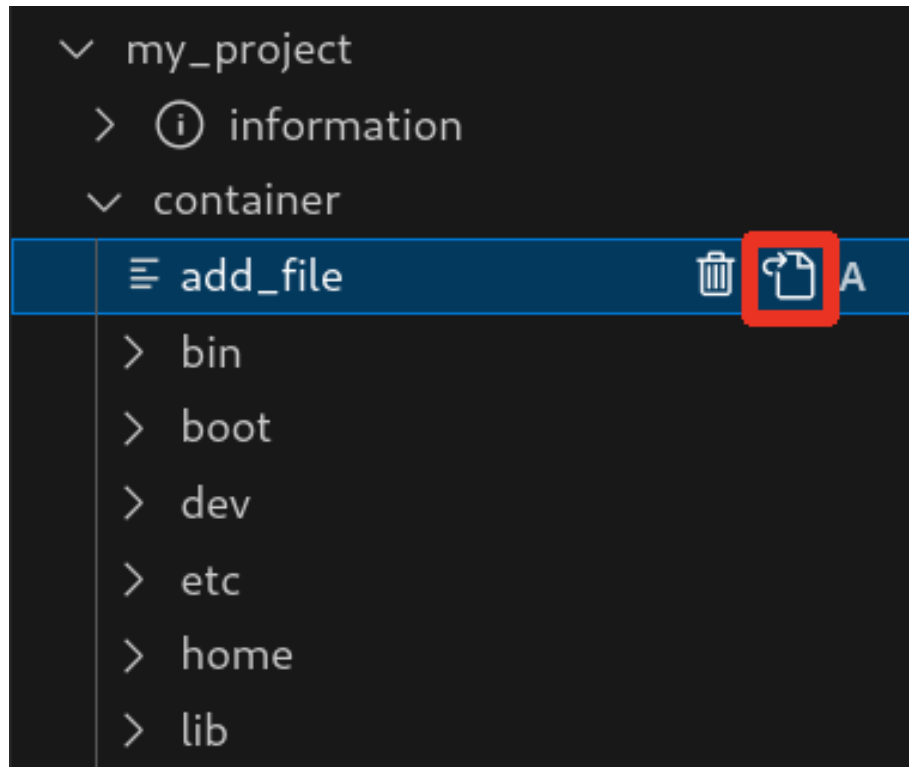


図 3.209 container/resources 下にあるファイルを開くボタン

3.14.5.5. container/resources 下にあるファイルおよびフォルダーの削除

「図 3.209. container/resources 下にあるファイルを開くボタン」の赤枠で囲われている表記のボタンをクリックすることで、container/resources 下にあるファイルを削除することができます。

この例では、プロジェクトディレクトリにある container/resources 下の add_file を削除します。

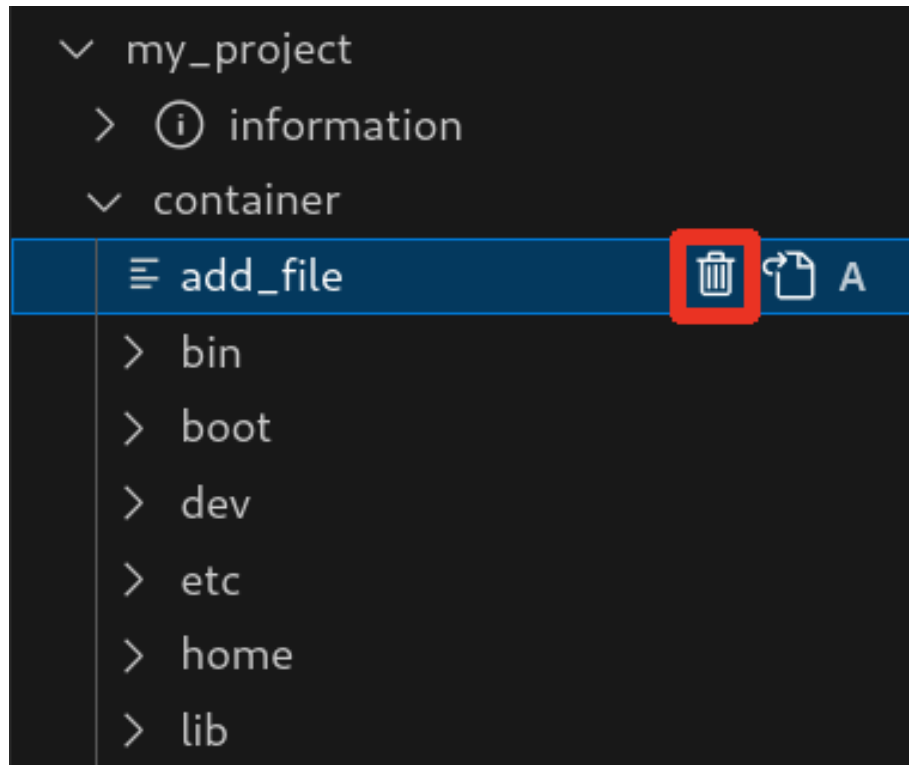


図 3.210 container/resources 下にあるファイルを削除するボタン

ディレクトリも同様に「図 3.209. container/resources 下にあるファイルを開くボタン」の赤枠で囲われている表記のボタンをクリックすることで削除することができます。

3.14.5.6. コンテナ内のファイルを container/resources 下に保存

「図 3.211. コンテナ内のファイルを container/resources 下に保存するボタン」の赤枠で囲われている表記のボタンをクリックすることで、コンテナ内にあるファイルをプロジェクトディレクトリにある container/resources 下に保存します。

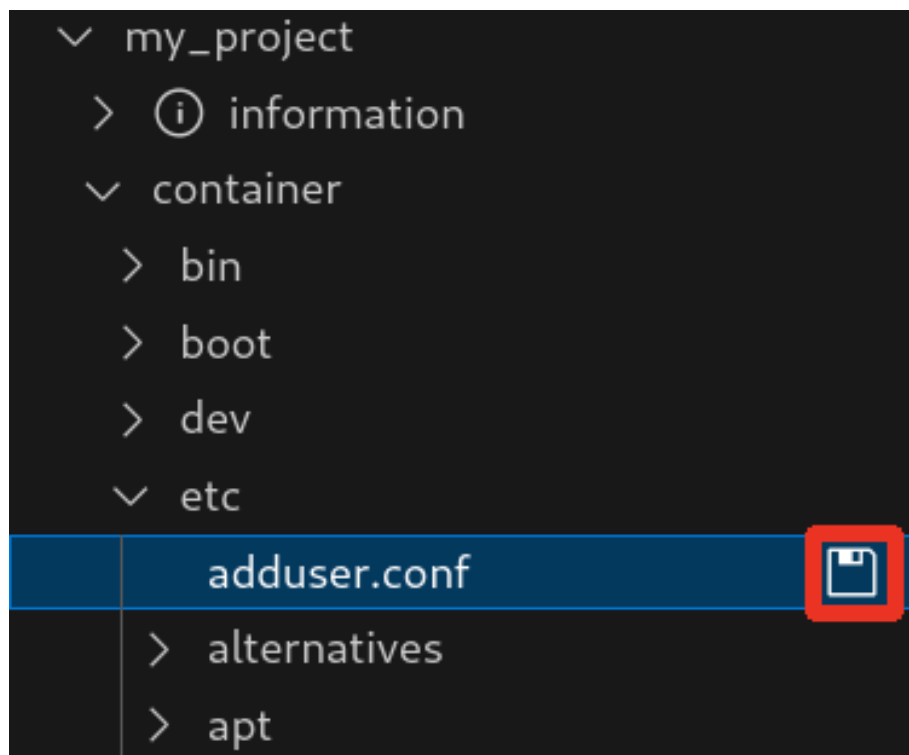


図 3.211 コンテナ内のファイルを container/resources 下に保存するボタン

ファイルが container/resources 下に保存されると、「図 3.212. 編集前のファイルを示すマーク」に示すように、ファイル名の右側に "U" のマークが表示されます。

"U" のマークはプロジェクトディレクトリにある container/resources 下のファイルとコンテナ内にあるファイルの内容が同一であることを示します。

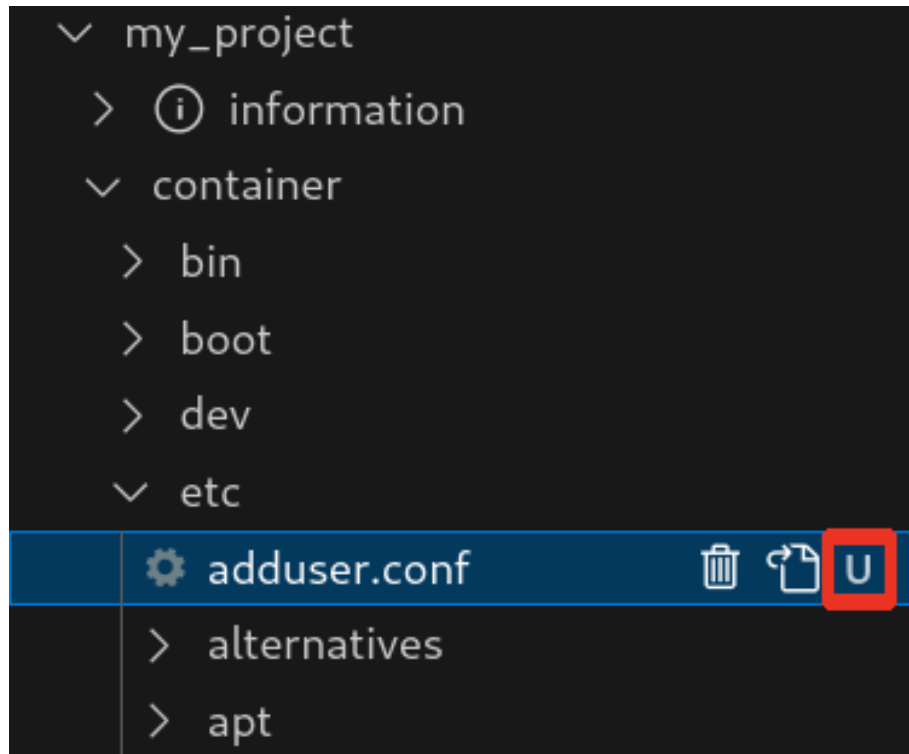


図 3.212 編集前のファイルを示すマーク

container/resources 下にあるファイルを編集して再表示すると、「図 3.213. 編集後のファイルを示すマーク」に示すように、ファイル名の右側に "M" のマークが表示されます。

"M" のマークはプロジェクトディレクトリにある container/resources 下のファイルとコンテナ内にあるファイルの内容に差があることを示します。

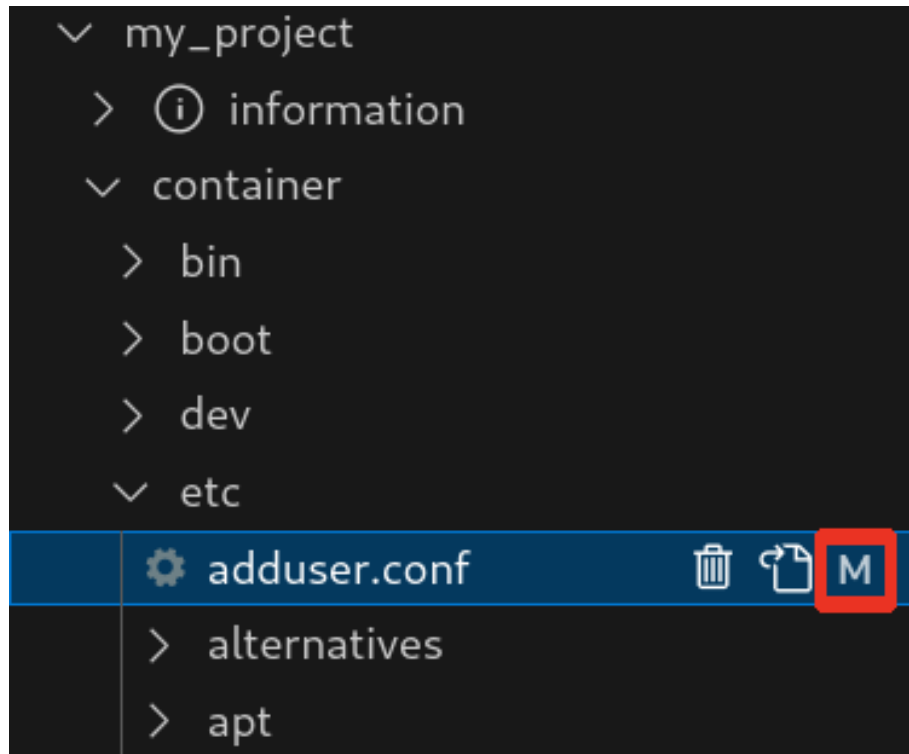


図 3.213 編集後のファイルを示すマーク

3.14.5.7. エラー表示

container/resources 下とコンテナ内にあるファイルまたはディレクトリを比較して、同名でかつファイルの種類が異なる場合、「図 3.214. コンテナ内にコピーされないことを示すマーク」に示すように、ファイル名の右側に "E" のマークが表示されます。

"E" のマークが表示された場合、そのファイルまたはディレクトリは [Generate development swu] または [Generate release swu] を実行してもコンテナにコピーされません。

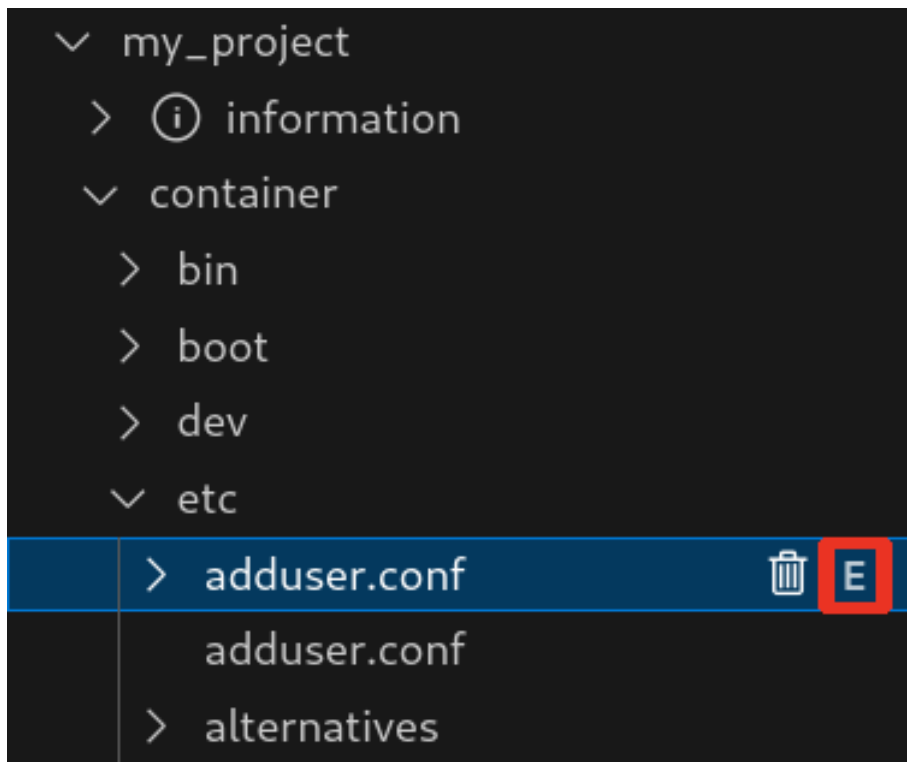


図 3.214 コンテナ内にコピーされないことを示すマーク

3.14.6. Armadillo に転送するディレクトリ及びファイル

コンテナイメージ以外に、以下に示すディレクトリやファイルを Armadillo に転送します。ここでは、プロジェクト名は `my_project` としています。

Armadillo に転送するディレクトリ及びファイル

- ・ `my_project/app/build`
- ・ `my_project/app/lib`

3.14.7. Armadillo 上でのセットアップ

3.14.7.1. 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

「3.14.3.6. 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 `development.swu` を「3.3.3.6. SWU イメージのインストール」 を参照して Armadillo へ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。

3.14.7.2. ssh 接続に使用する IP アドレスの設定

VS Code 上で ABOSDE(Armadillo Base OS Development Environment) から、ABOS Web が動作している Armadillo の一覧を確認し、指定した Armadillo の IP アドレスを ssh 接続に使用することができます。ただし、ATDE のネットワークを NAT に設定している場合は Armadillo がリストに表示されません。

「図 3.215. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする」 の赤枠で囲われているボタンをクリックすることで、ローカルネットワーク上で ABOS Web が実行されている Armadillo をスキャンすることができます。

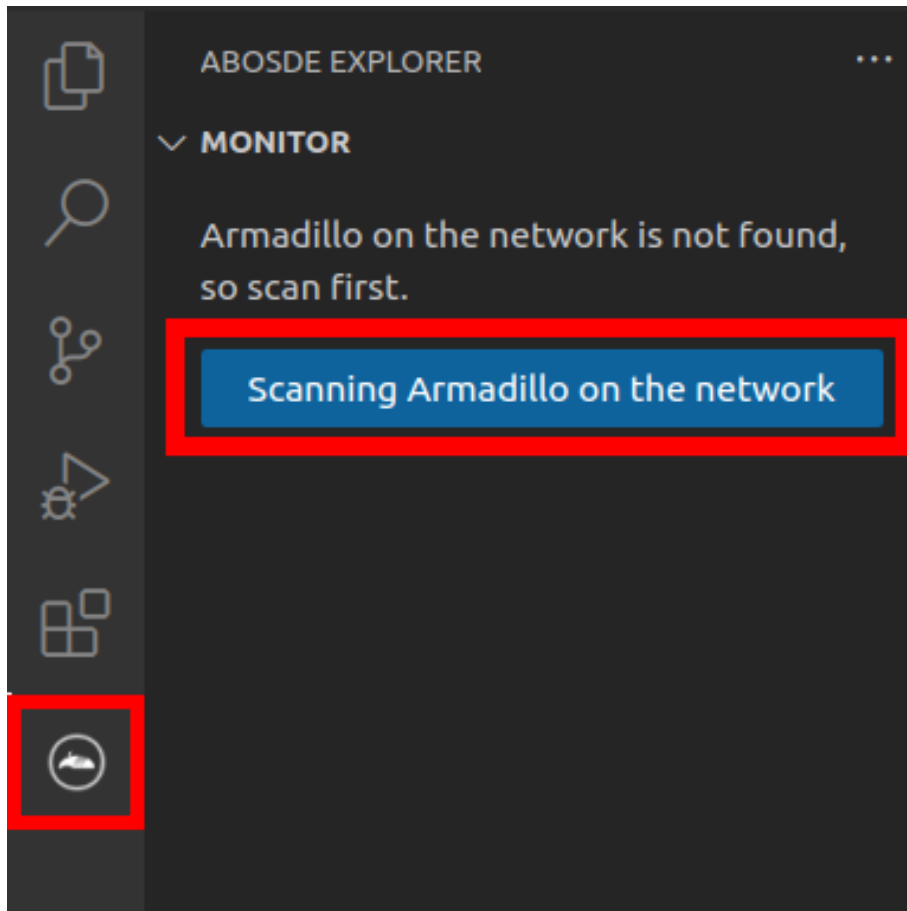


図 3.215 ABOSDE で ローカルネットワーク上の Armadillo をスキャンする

「図 3.216. ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する」の赤枠で囲われているマークをクリックすることで、指定した Armadillo の IP アドレスを ssh 接続に使用する IP アドレスに設定することができます。

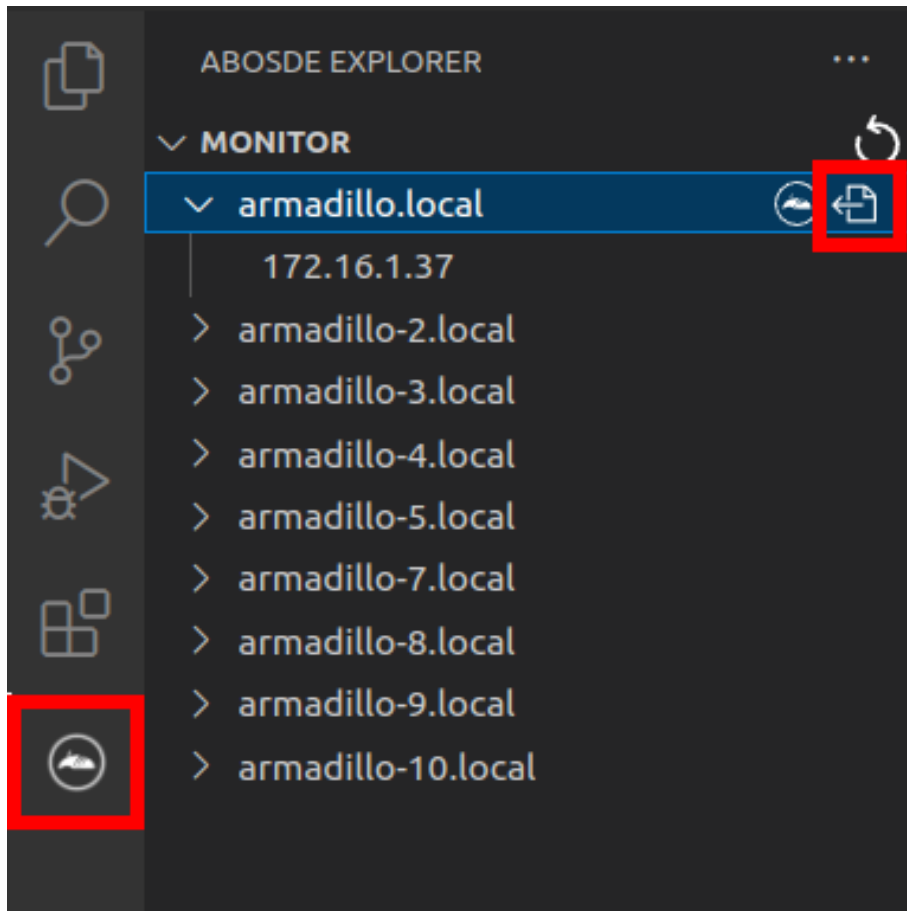


図 3.216 ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する

「図 3.217. ABOSDE に表示されている Armadillo を更新する」の赤枠で囲われているマークをクリックすることで、ABOSDE に表示されている Armadillo を更新することができます。

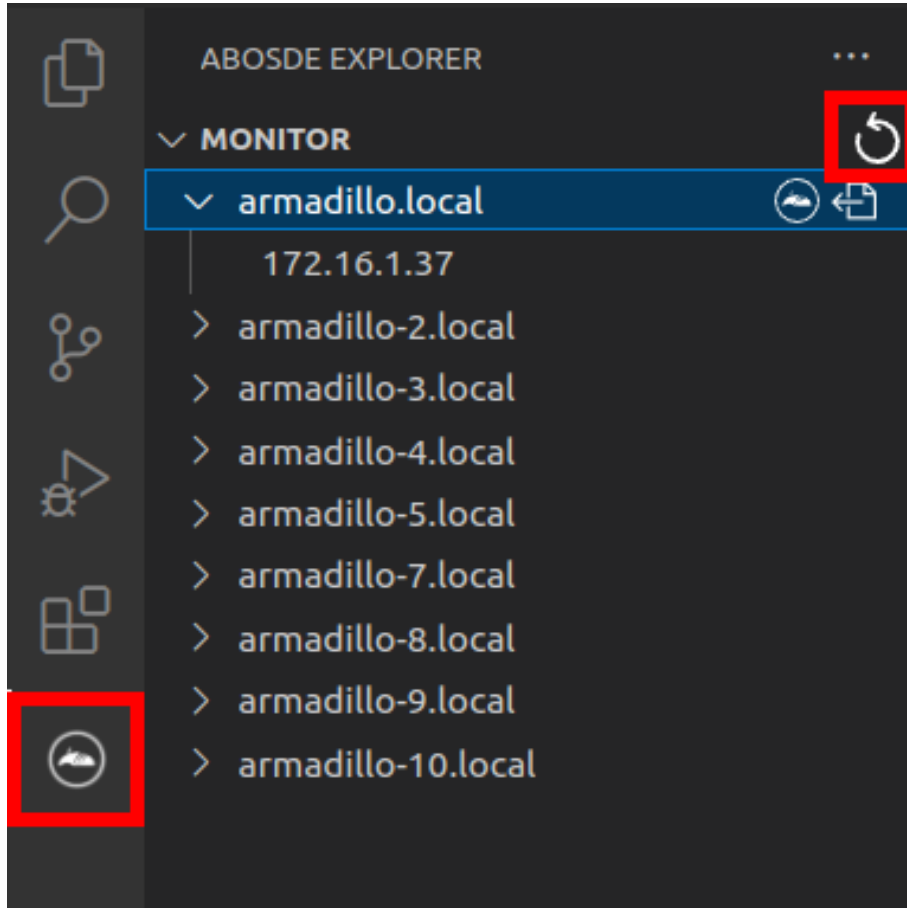


図 3.217 ABOSDE に表示されている Armadillo を更新する

ATDE のネットワークを NAT に設定している場合や、ABOS Web を起動していない場合等、ABOSDE のリストに Armadillo が表示されない場合は、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 x.x.x.x ❶
  User root
  IdentityFile ${HOME}/.ssh/id_ed25519_vscode
  UserKnownHostsFile config/ssh_known_hosts
  StrictHostKeyChecking accept-new
```

図 3.218 ssh_config を編集する

- ❶ 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。

3.14.7.3. アプリケーションの実行

VS Code の左ペインの [my_project] から [App run on Armadillo] を実行すると、実行ファイルや共有ライブラリを作成した後、アプリケーションが Armadillo へ転送されて起動します。

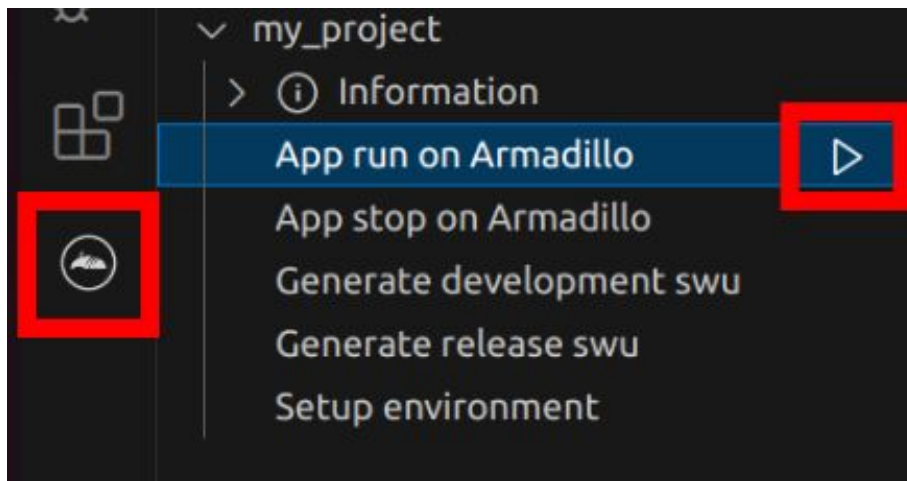


図 3.219 Armadillo 上でアプリケーションを実行する

VS Code 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 3.220 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 VS Code の左ペインの [my_project] から [App stop on Armadillo] を実行してください。

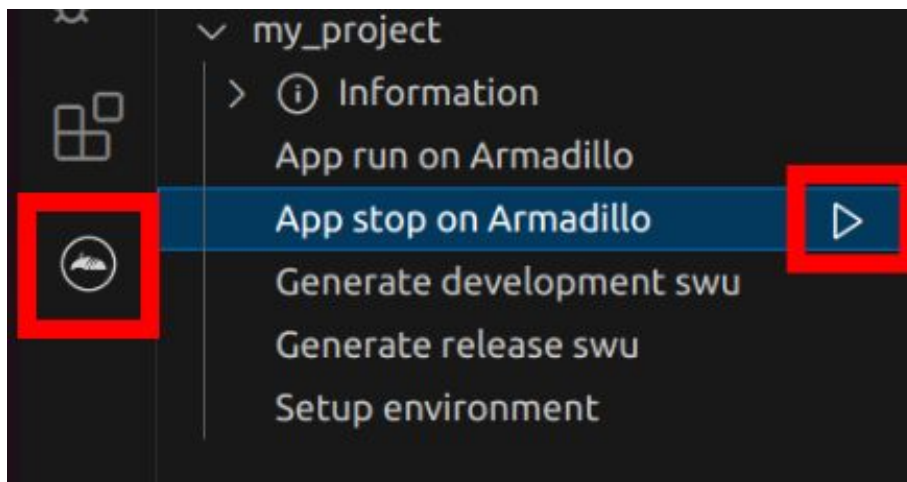


図 3.221 アプリケーションを終了する

3.14.8. SBOM 生成に関する設定

SWU イメージ作成時に、同時に SBOM を生成することができます。詳細は「3.15. SBOM 生成に関わる設定を行う」を参照してください。

3.14.9. 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。

VS Code 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「5.4.1. 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。

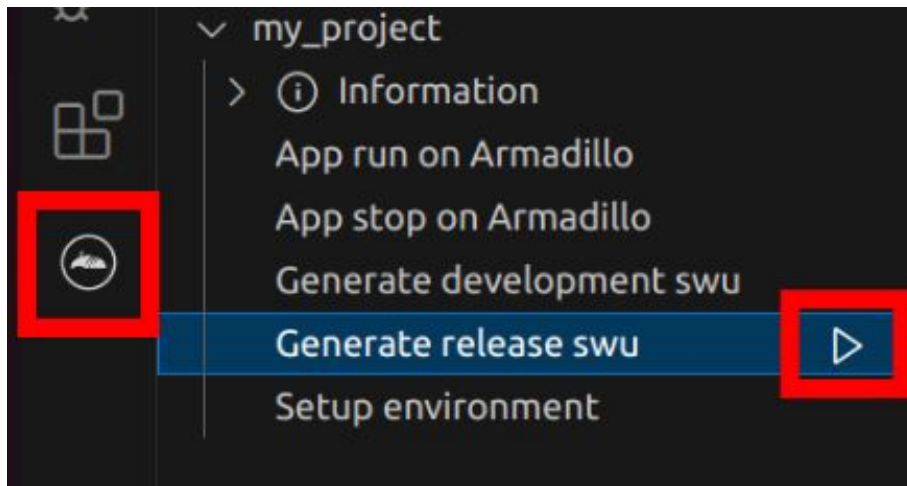


図 3.222 リリース版をビルドする



リリース版の SWU イメージには、開発用の機能は含まれていません。このため、リリース版の SWU イメージをインストールした Armadillo では、[App run on Armadillo] を使用したリモート実行は使用できません。

3.14.10. 製品への書き込み

作成した SWU イメージは my_project 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「3.3.3.6. SWU イメージのインストール」を参照して Armadillo に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。

3.14.11. Armadillo 上のコンテナイメージの削除

「6.2.3. コンテナとコンテナに関連するデータを削除する」を参照してください。

3.15. SBOM 生成に関わる設定を行う

ABOSDE では SWU イメージの生成と同時に SBOM が生成されます。生成される SBOM 名は SWU イメージ.spdx.json になります。json 形式で ISO/IEC5962 で国際標準となっている SPDX2.2 のフォーマットに準拠しています。SBOM についての詳細は「6.22. SBOM の提供」をご参照ください。



SBOM の生成には mkswu (6.4 以上) と、python3-make-sbom パッケージが必要です。python3-make-sbom パッケージがインストールされていない場合、SBOM は生成されません。「図 3.223. mkswu バージョン確認コマンド」を実行するとインストール済のバージョンが確認できます。

```
[ATDE ~]$ mkswu --version
mkswu バージョン 6.4
```

図 3.223 mkswu バージョン確認コマンド

表示されない場合は mkswu がインストールされていないので、「図 3.224. mkswu のインストール・アップデートコマンド」を実行してインストールしてください。mkswu をアップデートする場合もこちらを実行して下さい。

```
[ATDE ~]$ sudo apt update && sudo apt install mkswu
```

図 3.224 mkswu のインストール・アップデートコマンド

python3-make-sbom パッケージがインストールされている場合、make_sbom.sh が実行可能です。「図 3.225. make_sbom.sh 実行確認コマンド」を実行して、ヘルプが表示されるかご確認ください。

```
[ATDE ~]$ make_sbom.sh -h
```

図 3.225 make_sbom.sh 実行確認コマンド

表示されない場合は python3-make-sbom がインストールされていないので、「図 3.226. python3-make-sbom のインストールコマンド」を実行してインストールしてください。

```
[ATDE ~]$ sudo apt update && sudo apt install python3-make-sbom
```

図 3.226 python3-make-sbom のインストールコマンド

3.15.1. SBOM 生成に必要なファイルを確認する

SBOM の生成には以下の二つのファイルが必要です。

- ・ コンフィグファイル
- ・ desc ファイル

SBOM の生成にはライセンス情報を示したコンフィグファイルを使用します。コンフィグファイルは config/sbom_config.yaml.templ になります。SWU イメージ作成時にこのコンフィグファイルからバー

ジョン番号をアップデートした `swu/sbom_config.yaml` が生成されます。リリース時にはコンフィグファイルの内容を確認し、正しい内容に変更してください。各項目の詳細な説明については `SPDX specification v2.2.2` (<https://spdx.github.io/spdx-spec/v2.2.2/>) をご覧ください。

SBOM に含めるコンテナイメージ等の情報については `desc` ファイルに記載されています。各項目の説明については「6.22.4.2. desc ファイルを編集する」をご覧ください。

3.16. システムのテストを行う

Armadillo 上で動作するシステムの開発が完了したら、製造・量産に入る前に開発したシステムのテストを行ってください。

テストケースは開発したシステムに依ると思いますが、Armadillo で開発したシステムであれば基本的にテストすべき項目について紹介します。

3.16.1. ランニングテスト

長期間のランニングテストは実施すべきです。

ランニングテストで発見できる現象としては、以下のようなものが挙げられます。

- ・ 長期間稼働することでソフトウェアの動作が停止してしまう

開発段階でシステムを短い時間でしか稼働させていなかった場合、長期間ランニングした際になんらかの不具合で停止してしまう可能性が考えられます。

開発が完了したら必ず、長時間のランニングテストでシステムが異常停止しないことを確認するようにしてください。

コンテナの稼働情報は `podman stats` コマンドで確認することができます。

- ・ メモリリークが発生する

アプリケーションのなんらかの不具合によってメモリリークが起こる場合があります。

また、運用時の Armadillo は基本的に `overlayfs` で動作しています。そのため、外部ストレージやボリュームマウントに保存している場合などの例外を除いて、動作中に保存したデータは `tmpfs` (メモリ)上に保存されます。よくあるケースとして、動作中のログなどのファイルの保存先を誤り、`tmpfs` 上に延々と保存し続けてしまうことで、メモリが足りなくなってしまうことがあります。

長時間のランニングテストで、システムがメモリを食いつぶさないかを確認してください。

メモリの空き容量は「図 3.227. メモリの空き容量の確認方法」に示すように `free` コマンドで確認できます。

```
[armadillo ~]# free -h
              total        used         free       shared  buff/cache   available
Mem:           1.9G          327.9M          1.5G           8.8M           97.4M           1.5G
Swap:        1024.0M              0          1024.0M
```

図 3.227 メモリの空き容量の確認方法

3.16.2. 異常系における挙動のテスト

開発したシステムが、想定した条件下で正しく動作することは開発時点で確認できていると思います。しかし、そのような正常系のテストだけでなく、正しく動作しない環境下でどのような挙動をするのかも含めてテストすべきです。

よくあるケースとしては、動作中に電源やネットワークが切断されてしまった場合です。

電源の切断時には、Armadillo に接続しているハードウェアに問題はないか、電源が復旧した際に問題なくシステムが復帰するかなどをよくテストすると良いです。

ネットワークの切断時には、再接続を試みるなどの処理が正しく実装されているか、Armadillo とサーバ側でデータなどの整合性が取れるかなどをよくテストすると良いです。

この他にもシステムによっては多くの異常系テストケースが考えられるはずですので、様々な可能性を考慮しテストを実施してから製造・量産ステップに進んでください。

3.17. ユーザー設定とユーザーデータを一括削除する

ユーザー設定とユーザーデータを一括削除することができます。ユーザー設定の削除では ABOS Web から設定できる以下の項目を削除します。

- ・ ネットワーク設定
 - ・ LAN、WLAN、WWAN の設定を全て削除します。WLAN はクライアント設定とアクセスポイント設定の両方を削除します。
- ・ DHCP 設定
- ・ NAT 設定
- ・ VPN 設定
- ・ NTP 設定

ABOS Web から設定できるものであっても以下は削除されません。

- ・ Rest API トークン
- ・ UI カスタマイズの内容

ユーザーデータの削除では以下のデータを削除します。

- ・ /var/app/volumes ディレクトリ下のファイルを全て
- ・ /var/log ディレクトリ下のファイルを全て

ユーザー設定とユーザーデータを削除するには Armadillo 上で `abos-ctrl reset-default` コマンドを使用します。

```
[armadillo ~]# abos-ctrl reset-default ❶
Run with dry-run mode.
rm -f /etc/NetworkManager/system-connections/*
persist_file -r /etc/NetworkManager/system-connections
persist_file -r /etc/dnsmasq.d
```

```
rc-service dnsmasq restart
/etc/init.d/iptables save
sed -i -e '/NETAVARK/d' /etc/iptables/rules-save
persist_file /etc/iptables/rules-save
podman stop -a
find /var/app/volumes /var/log -mindepth 1 -delete
If you want to actually run the above commands,
add the -f/--force option.
```

図 3.228 削除されるユーザー設定とユーザーデータを確認

- ① 何もオプションを付けない場合、DRY-RUN モードとなり実際に削除は行われません。実際に削除を行う時に実行されるコマンドが表示されるのみです。

表示されたコマンドを確認し実際に削除されてもよい場合は、以下のように `-f` オプションを付けて実行してください。

```
[Armadillo ~]# abos-ctrl reset-default -f
rm -f /etc/NetworkManager/system-connections/*
persist_file -r /etc/NetworkManager/system-connections
persist_file -r /etc/dnsmasq.d
rc-service dnsmasq restart
/etc/init.d/iptables save
sed -i -e '/NETAVARK/d' /etc/iptables/rules-save
persist_file /etc/iptables/rules-save
podman stop -a
find /var/app/volumes /var/log -mindepth 1 -delete
Starting clone to /dev/mmcbk2p1
Cloning rootfs
Updating appfs snapshots
Reusing up-to-date bootloader
Rollback clone successful
WARNING: Rebooting!
```

図 3.229 実際にユーザー設定とユーザーデータを削除する

コマンド実行後は自動的に Armadillo が再起動します。

ABOS Web または Rest API から実行することもできます。ABOS Web から実行する場合は「6.9.8. ユーザー設定とユーザーデータの削除」を参照してください。Rest API から実行する場合は「6.9.6.16. Rest API : ユーザー設定とユーザーデータの管理」を参照してください。



再起動後、再び設定が必要な場合は ABOS Web や REST API を使用して行ってください。特に Armadillo Twin を利用している場合は、必ずネットワークの再設定を行ってください。

4. 量産編

本章では Armadillo を組み込んだ最終製品をお客様が製造・量産するうえで、必要となる情報や作業について記載します。

- ・「4.1. 概略」では、量産の進め方に関する概略を記載します。
- ・「4.2. BTO サービスを使わない場合と使う場合の違い」では、BTO(Build To Order) サービスに関する説明をします。
- ・「4.3. 量産時のイメージ書き込み手法」では、開発を完了したソフトウェアの量産用イメージ作成・書き込み方法を説明します。
- ・「4.4. インストールディスクを用いてイメージ書き込みする」は、インストールディスクを使用する方法を説明します。
- ・「4.5. SWUpdate を用いてイメージ書き込みする」は、SWUpdate を使用する方法を説明します。

4.1. 概略

量産の進め方の概略図を「図 4.1. Armadillo 量産時の概略図」に示します。お客様の製品仕様や製造工程の要件によってはこの例とは違った工程順となる場合や、工程の追加・削除がある可能性があります。

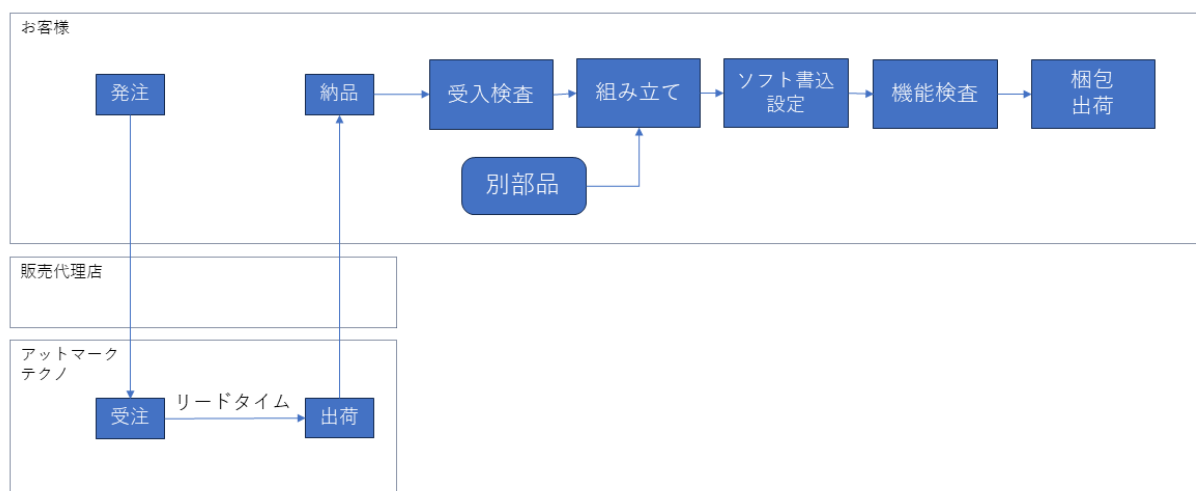


図 4.1 Armadillo 量産時の概略図

4.1.1. Armadillo Twin を契約する

Armadillo Twin を使用したデバイス運用管理を行う場合は、量産モデルの発注とは別に Armadillo Twin の契約が必要となります。Armadillo Twin の契約の詳細については、弊社営業、ご利用の販売代理店にお問い合わせください。

4.1.2. リードタイムと在庫

量産モデルを発注後、お客様に納品されるまでにリードタイムが発生します。開発セットや少量の量産モデル購入の場合、アットマークテクノや代理店在庫によって、短期間で納品できることもあります。しかし、まとまった数量の量産モデルの場合、納品までにお時間をいただくことがあります。新規に製品を量産・出荷する場合はリードタイムを考慮したスケジューリングをお願いします。また、リピート製造をする場合でも、欠品を起こさないよう適切な在庫の確保をお願いいたします。

リードタイムは状況・タイミングによって異なりますので、都度、弊社営業、ご利用の販売代理店にお問い合わせください。

4.1.3. Armadillo 納品後の製造・量産作業

お客様が Armadillo を納品後に次に示すようなキitting作業、組み立て、検査を実施し出荷を行います。

- ・ ソフトウェア書き込み
 - ・ Armadillo Base OS やアプリケーションコンテナイメージの書き込み
 - ・ 設定ファイルの書き込み
- ・ 別部品の組み立て
 - ・ SD カード/ SIM カード/ RTC バックアップ電池等の接続
 - ・ 拡張基板接続やセンサー・外部機器の接続
 - ・ お客様専用筐体への組み込み
- ・ 検査
 - ・ Armadillo の受け入れ検査
 - ・ 組み立て後の通電電検・機能検査
 - ・ 目視検査
- ・ 梱包作業
- ・ 出荷作業

有償の BTO サービスを利用することで、これらの作業の一部をアットマークテクノへ委託・実施済みの状態で Armadillo を納品することも可能です。費用はいただきますがお客様による工程立ち上げ、場所の確保、作業者の教育、品質管理等のトータルコストを考えると委託した方が安く済むケースが多いです。

また、BTO サービスではお受けできないようなキitting、検査、作業については、実施可能な業者をご紹介します等、個別の対応をすることで解決できる場合もございます。詳しくは弊社担当の営業、またはご利用の販売代理店にご相談ください。

4.2. BTO サービスを使わない場合と使う場合の違い



図 4.2 BTO サービスで対応する範囲

4.2.1. BTO サービスを利用しない(標準ラインアップ品)

有償の量産サービスを利用しない場合、標準ラインアップ仕様での納品となります。大きく分けて試作開発用途で使う「開発セット」と量産向けの「量産モデル」の2種類があります。量産用途では「量産モデル」をご利用ください。

「量産モデル」には AC アダプタ等のオプション品が付属されておりませんので、内容物を確認の上、発注をお願いいたします。ラインアップ一覧については「2.2. 製品ラインアップ」をご確認ください。

4.2.1.1. 標準ラインアップ品に書き込まれているソフトウェア

標準ラインアップ品に書き込まれるソフトウェアイメージ(Armadillo Base OS)は、アットマークテクノで公開している標準イメージとなります。また、ソフトウェアバージョンは指定することができず、ランニングチェンジで随時最新版を適用していきます。このため、納品後の Armadillo 個体では、開発段階で評価した Armadillo Base OS と異なるバージョンが書き込まれている可能性があります。

また、アプリケーションコンテナについては何も書き込まれていない状態となります。

納品後、お客様の量産工程でソフトウェアの書き込み作業が必要となります。詳しくは「4.3. 量産時のイメージ書き込み手法」をご確認ください。

4.2.2. BTO サービスを利用する

BTO サービスは、セミオーダー式メニューから選択して Armadillo の量産品を一括手配いただける有償サービスです。標準ラインアップ品の仕様をベースとして、搭載するモジュールの種類やケース、ACアダプタの有無、お客様支給品の SD カードや SIM カードの接続、お客様ご指定のソフトウェアイメージ書き込みなど、メニュー内から指定可能なキッティング項目を選択・指定することが可能です。

販売代理店またはアットマークテクノの窓口からお申し込みいただけます。

製品ごとに、対応できる作業とできない作業がございます。また、販売直後の製品の場合など BTO サービスに未対応である場合もあります。詳しくは Armadillo サイトの BTO サービス [<https://armadillo.atmark-techno.com/services/customize/bto>] をご確認ください。

4.3. 量産時のイメージ書き込み手法

量産時に必要な手順は最終製品によって異なりますが、開発したソフトウェアを Armadillo に書き込む手順は必ず実施することになります。Armadillo Base OS 搭載製品において、量産時に任意のソフトウェアを書き込む際には、以下の2つの手法のどちらかを用いると実現できます。

- ・ インストールディスクを用いてソフトウェアを書き込む

- ・ SWUpdate を用いてソフトウェアを書き込む

ただし、SWUpdate は運用中の Armadillo のアップデート機能であり、量産時のイメージ書き込みは本来の用途でないため、基本的にはイメージ書き込みに特化しているインストールディスクを用いた方法を選択してください。

それぞれの手法の特徴を「表 4.1. インストールディスクと SWUpdate によるソフトウェア書き込みの比較」にまとめます。ソフトウェア書き込み工程を決定する際の参考にしてください。

表 4.1 インストールディスクと SWUpdate によるソフトウェア書き込みの比較

手段	メリット	デメリット
インストールディスク	<ul style="list-style-type: none"> ・ インストールの前後処理を行なうシェルスクリプトのテンプレートが用意されている ・ インストールの前後処理は、microSD カード内にシェルスクリプトを配置するだけなので製造担当者にも編集しやすい 	<ul style="list-style-type: none"> ・ インストール実行にはジャンパピンをショートにするために Armadillo のケースを開ける必要がある ・ 動いているシステムをそのままインストールディスクにするため、出荷時の標準イメージから手動で同じ環境を構築する手順が残らない
SWUpdate	<ul style="list-style-type: none"> ・ ジャンパピンをショートにせずに実行できるため、Armadillo のケースを開ける必要がない ・ 必ず必要となる初回アップデートを別途実行する必要がない 	<ul style="list-style-type: none"> ・ SWU イメージの作成には、mkswu を使用できる環境と desc ファイルの記述方法を知る必要があるため、開発担当者以外に SWU イメージを更新させるハードルが少し高い ・ ログの取得など、インストール前後の処理が必要な場合は自分で記述する必要がある

量産時のイメージ書き込みにインストールディスクを使用する場合は、「4.4. インストールディスクを用いてイメージ書き込みする」に進んでください。

量産時のイメージ書き込みに SWUpdate を使用する場合は、「4.5. SWUpdate を用いてイメージ書き込みする」に進んでください。

4.4. インストールディスクを用いてイメージ書き込みする

「3.3.5. インストールディスクについて」でも紹介したとおり、Armadillo Base OS 搭載製品では、開発が完了した Armadillo のクローン用インストールディスクを作成することができます。

以下では、クローン用インストールディスクを作成する手順を準備段階から紹介します。

4.4.1. /etc/swupdate_preserve_file への追記

Armadillo Base OS のバージョンを最新版にしておくことを推奨しています。最新版でない場合は、バージョンが古いゆえに以下の作業を実施出来ない場合もありますので、ここで Armadillo Base OS のバージョンをアップデートしてください。

ここでは SWUpdate を使用して Armadillo Base OS のアップデートを行ないますが、このアップデートを行なうと、/etc/swupdate_preserve_files に記載の無いファイルは消えてしまいます。Armadillo Base OS のルートファイルシステム上に消えてほしくないファイルを開発中に配置していた場合は、「図 4.3. 任意のファイルパスを/etc/swupdate_preserve_files に追記する」に示すコマンドを実行することで /etc/swupdate_preserve_files にそのファイルが追記され、アップデート後も保持し続けるようになります。

一部のファイルやディレクトリは初めから /etc/swupdate_preserve_files に記載されている他、podman commit したコンテナイメージについてもアップデート後に引き継がれるので、本ドキュメントのサンプルアプリケーションの場合は実行する必要はありません。

```
[armadillo /]# persist_file -p <ファイルのパス>
```

図 4.3 任意のファイルパスを/etc/swupdate_preserve_files に追記する

4.4.2. Armadillo Base OS の更新

Armadillo-640 Armadillo Base OS [https://armadillo.atmark-techno.com/resources/software/armadillo-640/baseos]から「Armadillo-640 用 SWU イメージファイル」の URL をコピーして、「図 4.4. Armadillo Base OS をアップデートする」に示すコマンドを実行することで Armadillo Base OS を最新版にアップデートできます。

```
[armadillo /]# swupdate -d '-u https://armadillo.atmark-techno.com/files/downloads/armadillo-640/image/baseos-600-[VERSION].swu'
```



図 4.4 Armadillo Base OS をアップデートする



Armadillo Base OS 3.19.1-at.4 以降では「abos-ctrl update」コマンドで /etc/swupdate.watch に記載されている URL の SWU イメージでアップデートすることができます。デフォルトでは最新の Armadillo Base OS へアップデートします。

正常に実行された場合は自動的に再起動します。

4.4.3. パスワードの確認と変更

「3.1.5.1. initial_setup.swu の作成」で SWUpdate の初回アップデートを行った際に、各ユーザーのパスワード設定をしました。開発中はログインしやすいような単純なパスワードにしていることがよくあるので、製品に適用しないようにこのタイミングで強固なパスワードに変更しておきましょう。

```
[armadillo /]# passwd ❶  
Changing password for root  
New password: ❷  
Retype password: ❸  
passwd: password for root changed by root  
  
[armadillo /]# passwd atmark ❹  
Changing password for atmark  
New password: ❺  
Retype password: ❻  
passwd: password for atmark changed by root  
  
[armadillo /]# persist_file /etc/shadow ❼
```

図 4.5 パスワードを変更する

- ❶ root ユーザのパスワードを変更します。

- ② 新しい root ユーザ用パスワードを入力します。
- ③ 再度新しい root ユーザ用パスワードを入力します。
- ④ atmark ユーザのパスワードを変更します。
- ⑤ 新しい atmark ユーザ用パスワードを入力します。
- ⑥ 再度新しい atmark ユーザ用パスワードを入力します。
- ⑦ パスワードの変更を永続化させます。

4.4.4. 開発したシステムをインストールディスクにする

Armadillo Base OS では、現在起動しているルートファイルシステム及びブートローダーをそのままインストールディスクイメージとして生成することができます。インストールディスクイメージの生成方法は二種類あります。それぞれの特徴をまとめます。

- ・ VS Code を使用して生成

ATDE と VS Code を使用して、開発したシステムのインストールディスクイメージを USB メモリ上に生成します。USB メモリは vfat もしくは ext4 形式でフォーマットし、空き容量が 10GB 以上のものを使用してください。VS Code に開発用エクステンションである ABOSDE をインストールする必要があります。

- ・ コマンドラインから生成

abos-ctrl make-installer コマンドを実行すると microSD カードにインストールディスクイメージを生成することができます。コマンド実行前に、Armadillo がインターネットに接続されており、かつ 10GB 以上の空き容量がある microSD カードが挿入されていることを確認してください。microSD カード内のデータはインストールディスク作成時に上書きされて消えてしまうので、必要なデータは予めバックアップを取っておいてください。microSD カード上にインストールディスクイメージを生成した場合、インストール時に任意のシェルスクリプトを実行することが可能です。この機能が必要な場合はコマンドラインからの生成を推奨します。

コマンドラインから生成する場合は、Armadillo の JTAG と SD ブート、U-Boot のコマンドプロンプトを無効化するインストールディスクイメージを生成することができます。

4.4.5. VS Code を使用して生成する

ATDE と VS Code を使用して、開発したシステムのインストールディスクイメージを生成します。「3.1.3. VS Code のセットアップ」を参考に、ATDE に VS Code 開発用エクステンションをインストールしてください。VS Code を使用してインストールディスクを生成する場合は以下の手順になります。

- ・ VS Code を使用したインストールディスク作成用 SWU の生成
- ・ Armadillo に USB メモリを挿入
- ・ インストールディスク作成用 SWU を ABOS Web からインストール
- ・ USB メモリ上にインストールディスクイメージを生成



この機能を使用するには、以下に示すバージョンのソフトウェアが必要です。

- ・ ABOSDE 1.6.0 以上
- ・ mkswu 5.3 以上
- ・ abos-base 2.3 以上

4.4.5.1. VS Code を使用したインストールディスク作成用 SWU の生成

VS Code の左ペインの [COMMON PROJECT COMMAND] から [Generate Installer On USB Swu] を実行します。

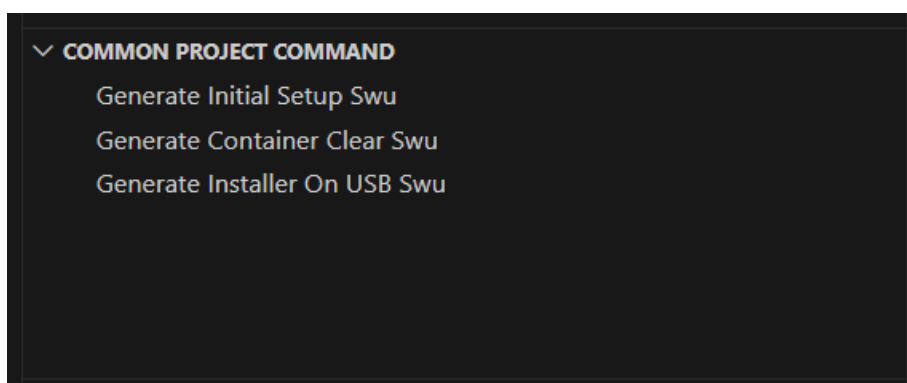


図 4.6 make-installer.swu を作成する

次に、対象製品を選択します。

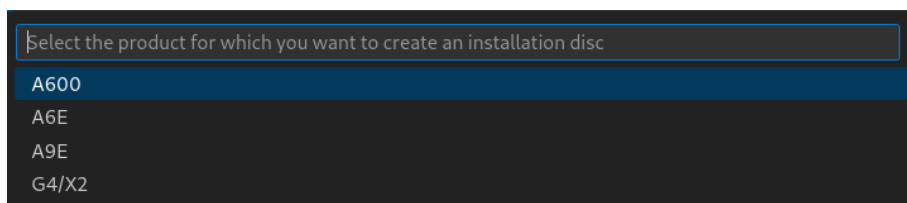


図 4.7 対象製品を選択する

無事に生成された場合、コンソールに以下のログが出力されます。

```
/home/atmark/.vscode/extensions/atmark-techno.armadillo-base-os-development-environment-1.6.0/  
shell/desc/make_installer_usb.desc のバージョンを 1 から 2 に変更しました。  
Enter pass phrase for /home/atmark/mkswu/swupdate.key: ❶  
/home/atmark/mkswu/make_installer_usb.swu を作成しました。  
To create Armadillo installer on USB memory install /home/atmark/mkswu/make_installer_usb.swu in  
Armadillo  
* Terminal will be reused by tasks, press any key to close it.
```

図 4.8 make-installer.swu 生成時のログ

- ❶ パスワードの入力を求められますので、初期化用 swu を生成したときと同じパスワードを入力します

/home/atmark/mkswu ディレクトリ内に make-installer.swu が作成されます。

4.4.5.2. Armadillo に USB メモリを挿入

Armadillo に電源を投入し、インストールディスクを保存するための USB メモリを挿入してください。



USB メモリは vfat もしくは ext4 形式でフォーマットし、空き容量が 10GB 以上のものを使用してください。Armadillo-640 への USB メモリのマウントは不要です。

インストールディスクイメージは `installer.img` という名前で保存します。すでに同名のファイルが存在する場合は上書きされます。

4.4.5.3. インストールディスク作成用 SWU を ABOS Web からインストール

ABOS Web を使用して、生成した `make-installer.swu` をインストールします。「6.9.4. SWU インストール」を参考に `make-installer.swu` を Armadillo へインストールしてください。実行時は ABOS Web 上に「[図 4.9. make-installer.swu インストール時のログ](#)」ようなログが表示されます。

```
make_installer_usb.swu をインストールします。  
SWU アップロード完了
```

```
SWUpdate v2023.05_git20231025-r0
```

```
Licensed under GPLv2. See source distribution for detailed copyright notices.
```

```
[INFO ] : SWUPDATE running : [main] : Running on iot-a6e Revision at1
```

```
[INFO ] : SWUPDATE started : Software Update started !
```

```
[INFO ] : SWUPDATE running : [install_single_image] : Installing pre_script
```

```
[INFO ] : SWUPDATE running : [read_lines_notify] : No base os update: copying current os over
```

```
[INFO ] : SWUPDATE running : [read_lines_notify] : Waiting for btrfs to flush deleted subvolumes
```

```
[INFO ] : SWUPDATE running : [install_single_image] : Installing Copying installer to USB device
```

```
[INFO ] : SWUPDATE running : [install_single_image] : Installing swdesc_command_nochroot 'podman  
kill -a'
```

```
[INFO ] : SWUPDATE running : [install_single_image] : Installing swdesc_command_nochroot --stdout-  
info 'abos-ctrl make-installer --noprompt --output /target/mnt/installer.img'
```

```
[INFO ] : SWUPDATE running : [read_lines_notify] : Using installer image on image file.
```

```
[INFO ] : SWUPDATE running : [read_lines_notify] : Would you like to create a windows partition?
```

```
[INFO ] : SWUPDATE running : [read_lines_notify] : That partition would only be used for  
customization script at the end of
```

```
[INFO ] : SWUPDATE running : [read_lines_notify] : install, leave at 0 to skip creating it.
```

```
[INFO ] : SWUPDATE running : [read_lines_notify] : Custom partition size (MB, [0] or 16 - 364): 0
```

```
[INFO ] : SWUPDATE running : [read_lines_notify] : Checking and growing installer main partition
[INFO ] : SWUPDATE running : [read_lines_notify] : Resize device id 1 (/dev/loop0p1) from 513.00MiB
to max
[INFO ] : SWUPDATE running : [read_lines_notify] : Copying boot image
[INFO ] : SWUPDATE running : [read_lines_notify] : Copying rootfs
[INFO ] : SWUPDATE running : [read_lines_notify] : Copying appfs
[INFO ] : SWUPDATE running : [read_lines_notify] : At subvol app/snapshots/volumes
[INFO ] : SWUPDATE running : [read_lines_notify] : At subvol app/snapshots/boot_volumes
[INFO ] : SWUPDATE running : [read_lines_notify] : At subvol app/snapshots/boot_containers_storage
[INFO ] : SWUPDATE running : [read_lines_notify] : Trying to shrink the installer partition...
[INFO ] : SWUPDATE running : [read_lines_notify] : Shrinking the installer partition...
[INFO ] : SWUPDATE running : [read_lines_notify] : Cleaning up and syncing changes to disk...
[INFO ] : SWUPDATE running : [read_lines_notify] : Installer updated successfully!
[INFO ] : SWUPDATE running : [read_lines_notify] : -rwxr-xr-x 1 root root 687.0M Jan 23 15:12 /
target/mnt/installer.img
[INFO ] : SWUPDATE running : [install_single_image] : Installing post_script
[INFO ] : SWUPDATE running : [read_lines_notify] : Removing unused containers
[INFO ] : SWUPDATE running : [read_lines_notify] : Command 'command podman rm -a -f' output:
[INFO ] : SWUPDATE running : [read_lines_notify] :
9f4f64ec1926d17e75de4060dac4a448e66ca3d9535c408f632e4e2de4bafa4f
[INFO ] : SWUPDATE running : Installation in progress
[INFO ] : SWUPDATE successful ! SWUPDATE successful !
[INFO ] : No SWUPDATE running : Waiting for requests...

swupdate exited

インストールが成功しました。
```

図 4.9 make-installer.swu インストール時のログ

完了後、USB メモリを抜いてください。もし、エラーが出た場合は Armadillo-640 の電源を再投入してやり直してください。

4.4.5.4. USB メモリ上にインストールディスクイメージを生成

無事に生成が完了した場合、USB メモリ上に installer.img が保存されています。この installer.img を microSD カードに書き込むことでインストールディスクを作成することができます。動作確認につい

では「4.4.6. インストールディスクの動作確認を行う」をご参照ください。これで、VS Code を使用してインストールディスクを生成する方法については終了です。

4.4.6. インストールディスクの動作確認を行う

作成したインストールディスクの動作確認を実施してください。開発に使用した Armadillo 以外の個体が必要になります。また、インストール先の Armadillo の eMMC 内のデータは上書きされて消えるため、必要なデータは予めバックアップを取っておいてください。

「4.4.4. 開発したシステムをインストールディスクにする」の手順で使用した USB メモリの中に installer.img が存在しますので、ATDE 上でこのイメージをもとに microSD カードにインストールディスクを作成してください。ATDE 上に installer.img をコピーした場合、コマンドは以下のようになります。/dev/sd[X] の [X] は microSD を示す文字を指定してください。

```
[ATDE ~] sudo dd if=installer.img of=/dev/sd[X] bs=1M oflag=direct status=progress
```

上記コマンドで作成した microSD のインストールディスクを、インストール先の Armadillo に挿入してください。その後、SW2 (起動デバイス設定スイッチ)を ON にしてから電源を入れます。Armadillo がインストールディスクから起動し、自動的にインストールスクリプトが動作します。

しばらくすると「reboot: Power down」と表示されるので、Armadillo の電源を切ります。その後 Armadillo から microSD カードを抜き、SW2 (起動デバイス設定スイッチ)を OFF にします。再度電源を投入することで、インストールは完了です。

実際にクローンした Armadillo が想定した通りの動作をすることを確認してください。

4.4.7. コマンドラインから生成する

4.4.7.1. JTAG と SD ブートを無効化する

コマンドラインから生成する場合は、Armadillo の JTAG と SD ブートを無効にするインストールディスクイメージを生成することができます。これらが無効にするには abos-ctrl installer-setting コマンドを実行します。



生成したインストールディスクを使用して初期化した Armadillo の JTAG と SD ブートを無効にする設定であり、開発用の Armadillo の JTAG と SD ブートが無効になることはありません。

```
[armadillo /]# abos-ctrl installer-setting
Would you like to disable JTAG in the installer ? [y/N] ❶
y
JTAG disabled setting for production Armadillo has been configured.
Would you like to disable SD boot in the installer ? [y/N] ❷
y
SD boot disabled setting for production Armadillo has been configured.
```

図 4.10 JTAG と SD ブートを無効化する

- ❶ JTAG を無効化する場合は y を入力します。無効化しない場合は何も入力せず Enter キーを押してください。
- ❷ SD ブートを無効化する場合は y を入力します。無効化しない場合は何も入力せず Enter キーを押してください。

現在の設定値を確認するには `abos-ctrl check-secure` コマンドを実行します。disabled の場合は無効化する設定になっています。

```
[armadillo /]# abos-ctrl check-secure
- JTAG access disabled for mass production.
- SD boot access disabled for mass production.
```

図 4.11 JTAG と SD ブートの設定値を確認する

設定をリセットするには `--reset` オプションを付けて `abos-ctrl installer-setting` コマンドを実行します。

```
[armadillo /]# abos-ctrl installer-setting --reset
cleaned up all settings.
```

図 4.12 JTAG と SD ブートの設定値をリセットする



JTAG および SD ブートを無効化したインストールディスクで初期化した Armadillo は、再びこれらを有効に戻すことはできなくなります。そのため不具合発生時にこれらのインターフェースを使用した不具合解析ができなくなる点に留意してください。

4.4.7.2. U-Boot のコマンドプロンプトを無効化する

コマンドラインから生成する場合は、Armadillo の U-Boot のコマンドプロンプトを無効にするインストールディスクイメージを生成することができます。U-Boot のコマンドプロンプトを無効にするには `abos-ctrl installer-setting` コマンドを実行します。このコマンドにより、SD ブート時の U-Boot のコマンドプロンプトも同時に無効化されます。



生成したインストールディスクを使用して初期化した Armadillo の U-Boot のコマンドプロンプトを無効にする設定であり、開発用の Armadillo の U-Boot のコマンドプロンプトが無効になることはありません。

```
[armadillo /]# abos-ctrl installer-setting
:(省略)
Would you like to disable boot prompt in the installer ? [y/N] ❶
y
Boot prompt disabled setting for production Armadillo has been configured.
```

図 4.13 U-Boot のコマンドプロンプトを無効化する

- ❶ U-Boot のコマンドプロンプトを無効化する場合は `y` を入力します。無効化しない場合は何も入力せず `Enter` キーを押してください。

現在の設定値を確認するには `abos-ctrl check-secure` コマンドを実行します。`disabled` の場合は無効化する設定になっています。

```
[armadillo /]# abos-ctrl check-secure
:(省略)
- boot prompt disabled for mass production.
```

図 4.14 U-Boot のコマンドプロンプトの設定値を確認する

設定をリセットするには「図 4.12. JTAG と SD ブートの設定値をリセットする」と同様に、`--reset` オプションを付けて `abos-ctrl installer-setting` コマンドを実行してください。

4.4.7.3. インストールディスクイメージを生成する

`abos-ctrl make-installer` コマンドを実行して、microSD カードにインストールディスクイメージを生成します。

```
[armadillo /]# abos-ctrl make-installer
Checking if /dev/mmcblk1 can be used safely...
It looks like your SD card does not contain an installer image
Download baseos-600-installer-latest.zip image from armadillo.atmark-techno.com (~170M) ? [y/N] ❶
WARNING: it will overwrite your SD card!!
y
Downloading and extracting image to SD card...
Finished writing baseos-600-installer-[VERSION].img, verifying written content...

Would you like to create a windows partition?
That partition would only be used for customization script at the end of
install, leave at 0 to skip creating it.
Custom partition size (MB, [0] or 16 - 29014): 500 ❷
Checking and growing installer main partition
Trying to install mkfs.exfat (exfatprogs) in memory from internet
fetch https://download.atmark-techno.com/alpine/v3.19/atmark/armv7/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.19/main/armv7/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.19/community/armv7/APKINDEX.tar.gz
(1/1) Installing exfatprogs (1.2.2-r0)
Executing busybox-1.36.1-r15.trigger
OK: 148 MiB in 197 packages
exfatprogs version : 1.2.2
Creating exFAT filesystem(/dev/mmcblk1p2, cluster size=131072)

Writing volume boot record: done
Writing backup volume boot record: done
Fat table creation: done
Allocation bitmap creation: done
Uppercase table creation: done
Writing root directory entry: done
Synchronizing...

exFAT format complete!
Resize device id 1 (/dev/mmcblk1p1) from 520.00MiB to max
Installer will disable JTAG access
```

```

Installer will disable SD boot after installation
Installer will disable uboot prompt
Environment OK, copy 1
Copying boot image
Copying rootfs
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!
    
```

図 4.15 開発完了後のシステムをインストールディスクイメージにする

- ① y を入力し Enter を押下します。
- ② インストールディスク内にインストールログを保存したい場合など、自由に使用できる第 2 パーティションを指定したサイズで作成します。サイズを入力し Enter を押下します。詳細は「4.4.7.4. インストール時に任意のシェルスクリプトを実行する」を参照してください。

「Installer updated successfully!」と表示されれば、正常に microSD カードにインストールディスクイメージを書き込むことができます。Armadillo から microSD カードを抜去してください。



セキュリティの観点から、インストールディスクによるインストール実行時に以下のファイルを再生成しております：

- ・ /etc/machine-id (ランダムな個体識別番号)
- ・ /etc/abos_web/tls (ABOS-Web の https 鍵)
- ・ /etc/ssh/ssh_host_*key* (ssh サーバーの鍵。なければ生成しません) ABOS 3.19.1-at.3 以降

他のファイルを個体毎に変更したい場合は「4.4.7.4. インストール時に任意のシェルスクリプトを実行する」で対応してください。

4.4.7.4. インストール時に任意のシェルスクリプトを実行する

作成したインストールディスクの所定の場所に、`installer_overrides.sh` というファイル名でシェルスクリプトを配置することで、インストール処理の前後で任意の処理を行なうことができます。

`installer_overrides.sh` に記載された「表 4.2. インストール中に実行される関数」に示す 3 つの名前の関数のみが、それぞれ特定のタイミングで実行されます。

表 4.2 インストール中に実行される関数

関数名	備考
preinstall	インストール中、eMMC のパーティションが分割される前に実行されます。
postinstall	<code>send_log</code> 関数を除く全てのインストール処理の後に実行されます。
send_log	全てのインストール処理が完了した後に実行されます。指定した場所にインストールログを保存できます。

`installer_overrides.sh` を書くためのサンプルとして、インストールディスクイメージの第 1 パーティション及び、「4.4.4. 開発したシステムをインストールディスクにする」で作成したのであれば第 2

パーティション直下に `installer_overrides.sh.sample` を用意してあります。このサンプルをコピーして編集するなどして、行ないたい処理を記述してください。

作成した `installer_overrides.sh` は、インストールディスクの第 1 パーティション(ラベル名は "rootfs_0")か、「4.4.4. 開発したシステムをインストールディスクにする」で作成したのであれば第 2 パーティション(ラベル名は "INST_DATA")の直下に配置することで実行されます。両方に配置した場合は、第 2 パーティションに配置した記述が適用されます。



インストールディスクの第 1 パーティションは btrfs、第 2 パーティションは exfat でフォーマットされているため、第 2 パーティションのみ Windows PC でもマウントして読み書きすることができます。

製造担当者が `installer_overrides.sh` を記述する場合に、仮に Windows PC しか作業環境がない場合でも、第 2 パーティションを作成しておくことで作業を行なうことができるというメリットもあります。

これを利用することで、複数台の Armadillo に対してそれぞれに異なる固定 IP アドレスを設定したり、各種クラウドへの接続鍵などを個体ごとに配置したりしたいなど、個体ごとに異なる設定を行なうなど柔軟な製造を行なうことも可能です。以下ではこの機能を利用して、個体ごとに異なる固定 IP アドレスを設定する方法と、インストール実行時のログを保存する方法を紹介します。

これらを必要としない場合は「4.4.8. インストールの実行」に進んでください。

4.4.7.5. 個体ごとに異なる固定 IP アドレスを設定する

インストール時に任意のシェルスクリプトを実行できる機能を利用して、複数の Armadillo に対して異なる固定 IP アドレスを割り当てる例を紹介します。

INST_DATA 内の `installer_overrides.sh.sample` と `ip_config.txt.sample` は個体ごとに異なる IP アドレスを割り振る処理を行なうサンプルファイルです。それぞれ `installer_overrides.sh` と `ip_config.txt` にリネームすることで、`ip_config.txt` に記載されている条件の通りに個体ごとに異なる固定 IP アドレスを設定することができます。全てをここでは説明しませんので、詳細はそれぞれのファイル内の記述も参照してください。

今回はそれぞれのファイルの内容は変更せず使用します。サンプルそのままですが、`ip_config.txt` の内容を「図 4.16. `ip_config.txt` の内容」に示します。

```
# mandatory first IP to allocate, inclusive
START_IP=10.3.4.2 ❶

# mandatory last IP to allocate, inclusive
END_IP=10.3.4.249 ❷

# netmask to use for the IP, default to 24
#NETMASK=24 ❸

# Gateway to configure
# not set if absent
GATEWAY=10.3.4.1 ❹

# DNS servers to configure if present, semi-colon separated list
```

```
# not set if absent
DNS="1.1.1.1;8.8.8.8" ⑤

# interface to configure, default to eth0
#IFACE=eth0 ⑥
```

図 4.16 ip_config.txt の内容

- ① このインストールディスクで割り振る IP アドレスの範囲の始まりを指定します。
- ② このインストールディスクで割り振る IP アドレスの範囲の終わりを指定します。
- ③ ネットマスクを指定します。指定しない場合は 24 になります。デフォルトでコメントアウトされています。
- ④ ゲートウェイアドレスを指定します。
- ⑤ DNS アドレスを指定します。セミコロンで区切ることでセカンダリアドレスも指定できます。
- ⑥ IP アドレスの設定を行なうインターフェースを指定します。指定しない場合は eth0 になります。デフォルトでコメントアウトされています。



インストール作業の並列化の為に、複数枚のインストールディスクで固定 IP アドレスを割り振る場合は、それぞれのインストールディスクが割り振る IP アドレスの範囲が被らないように ip_config.txt を設定してください。

これらのファイルを配置したインストールディスクでインストールを実行した Armadillo が、正しく設定できていることを確認します。

```
[armadillo /]# ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
    link/ether 00:11:22:33:44:55 brd ff:ff:ff:ff:ff:ff
    inet 10.3.4.2/24 brd 10.3.4.255 scope global noprefixroute eth0
        valid_lft forever preferred_lft forever
    inet6 ffff::ffff:ffff:ffff:ffff/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

図 4.17 IP アドレスの確認

また、サンプルスクリプトをそのまま使用すると、インストールディスクの第 2 パーティションに allocated_ips.csv というファイルが生成されます。このファイルには、このインストールディスクを使用して IP アドレスの設定を行なった個体のシリアル番号、MAC アドレス、設定した IP アドレスが追記されていきます。

```
SN, MAC, IP
00C700010009, 00:11:22:33:44:55, 10.3.4.2
```

図 4.18 allocated_ips.csv の内容



2 台目以降の Armadillo にこのインストールディスクで IP アドレスの設定を行なう際に、allocated_ips.csv を参照して次に割り振る IP アドレスを決めますので、誤って削除しないように注意してください。

4.4.7.6. インストール実行時のログを保存する

installer_overrides.sh 内の send_log 関数は、インストール処理の最後に実行されます。インストールしたルートファイルシステムやファームウェアのチェックサムなどの情報が記録されたログファイルのパスが LOG_FILE に入るため、この関数内でインストールディスクの第 2 パーティションに保存したり、外部のログサーバにアップロードしたりすることが可能です。

「図 4.19. インストールログを保存する」は、インストールディスクの第 2 パーティションにインストールログを保存する場合の send_log 実装例です。

```
send_log() {  
    : "This function is called after aggregating logs for archival"  
    local LOG_FILE="$1"  
  
    if [ -n "$USER_MOUNT" ]; then  
        mount /dev/mmcblk1p2 "$USER_MOUNT" ❶  
        cp $LOG_FILE $USER_MOUNT/${SN}_install.log ❷  
        umount "$USER_MOUNT" ❸  
    fi  
}
```

図 4.19 インストールログを保存する

- ❶ send_log 関数中では、SD カードの第 2 パーティション(/dev/mmcblk1p2)はマウントされていないのでマウントします。
- ❷ ログファイルを <シリアル番号>_install.log というファイル名で第 2 パーティションにコピーします。
- ❸ 第 2 パーティションをアンマウントします。

これらの変更を行なったインストールディスクでインストールを実行した後に、インストールディスクを PC などに接続して正しくログを保存できていることを確認してください。保存したログファイルの中身の例を「図 4.20. インストールログの中身」に示します。

```
RESULT:OK  
abos-ctrl make-rootfs on Tue Jun 21 17:57:07 JST 2022 4194304 6b8250df711de66b  
abos-ctrl make-rootfs on Tue Jun 21 17:57:24 JST 2022 314572800 58a9b6664158943e  
firm 8e9d83d1ba4db65d  
appfs 5108 1fa2cbaff09c2dbf
```

図 4.20 インストールログの中身

4.4.8. インストールの実行

前章までの手順で作成したインストールディスクを、開発に使用した Armadillo 以外の Armadillo に対して適用します。

クローン先の Armadillo の eMMC 内のデータは上書きされて消えるため、必要なデータは予めバックアップを取っておいてください。

「3.3.5.2. インストールディスクを使用する」を参照して、クローン先の Armadillo にインストールディスクを適用してください。

「4.4.7.1. JTAG と SD ブートを無効化する」で JTAG と SD ブートを無効化した場合は、インストールを行った Armadillo の JTAG と SD ブートが無効化されています。

ここまで完了したら、「4.6. イメージ書き込み後の動作確認」に進んでください。

4.5. SWUpdate を用いてイメージ書き込みする

4.5.1. SWU イメージの準備

ここでは、sample-container という名称のコンテナの開発を終了したとします。コンテナアーカイブの作成方法は「6.2.2.7. コンテナの自動作成やアップデート」を参照ください。

1. sample-container-v1.0.0.tar (動かしたいアプリケーションを含むコンテナイメージアーカイブ)
2. sample-container.conf (コンテナ自動実行用設定ファイル)

これらのファイルを /home/atmark/mkswu/sample-container ディレクトリを作成して配置した例を記載します。

```
[ATDE ~/mkswu/sample-container]$ ls
sample-container-v1.0.0.tar  sample-container.conf
```

図 4.21 Armadillo に書き込みたいソフトウェアを ATDE に配置

4.5.2. desc ファイルの記述

SWUpdate 実行時に、「4.5.1. SWU イメージの準備」で挙げたファイルを Armadillo に書き込むような SWU イメージを生成します。

SWU イメージを作成するためには、SWUpdate 時に実行する処理等を示した desc ファイルを記述し、その desc ファイルを入力として mkswu コマンドを実行することで、SWU イメージが出来上がります。

```
[ATDE ~/mkswu/sample-container]$ cat sample-container.desc
swdesc_option component=sample-container
swdesc_option version=1
swdesc_option POST_ACTION=poweroff ❶

swdesc_embed_container "sample-container-v1.0.0.tar" ❷
swdesc_files --extra-os --dest /etc/atmark/containers/ "sample-container.conf" ❸
```

図 4.22 desc ファイルの記述例

- ❶ SWUpdate 完了後に電源を切るように設定します。

- ② コンテナイメージファイルを SWU イメージに組み込み、Armadillo に転送します。
- ③ コンテナ起動設定ファイルを Armadillo に転送します。

ここまで完了したら、「4.6. イメージ書き込み後の動作確認」に進んでください。desc ファイルの詳細な書式については、「6.4. mkswu の .desc ファイルを編集する」を参照してください。また、作成された SWU イメージの内容を確認したい場合は、「6.6. SWU イメージの内容の確認」を参照してください。

4.6. イメージ書き込み後の動作確認

「4.4. インストールディスクを用いてイメージ書き込みする」で作成したインストールディスクを使用してインストール、または「4.5. SWUpdate を用いてイメージ書き込みする」にて SWUpdate によってイメージ書き込みを行った後には、イメージが書き込まれた Armadillo が正しく動作するか、実際に動かして確認してみます。

再度電源を投入して、期待したアプリケーションが動作することを確認してください。

ここまで完了したならば、量産時のイメージ書き込みは完了です。

4.7. 量産時の組み立て

Armadillo-640 を組み立てる際に必要な情報を紹介します。

4.7.1. オプションの組み付け

Armadillo-640 に組み付けることができるオプション品の仕様と組み付け方法については、「6.28. オプション品」を参照してください。

5. 運用編

5.1. Armadillo Twin に Armadillo を登録する

5.1.1. Armadillo の設置前に登録する場合

Armadillo を Armadillo Twin に登録する場合、ケース裏や基板本体に貼付されているシール上の QR コードを使用します。登録方法についての詳細は Armadillo Twin ユーザーマニュアル「Armadillo Twin にデバイスを登録する」 [<https://manual.armadillo-twin.com/register-device/>] をご確認ください。

5.1.2. Armadillo の設置後に登録する場合

Armadillo 設置後の登録については、弊社営業までお問い合わせください。

5.2. Armadillo を設置する

Armadillo を組み込んだ製品を設置する際の注意点や参考情報を紹介します。

5.2.1. 設置場所

開発時と同様に、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。

5.2.2. ケーブルの取り回し

一般的に以下の点を注意して設置してください。また、「3.4. ハードウェアの設計」に記載していることにも従ってください。

- ・ 設置時にケーブルを強く引っ張らないでください。
- ・ ケーブルはゆるやかに曲げてください。
- ・ ケーブルを結線する場合、きつくせず緩く束ねてください。

5.2.3. サージ対策

サージ対策については、「3.4.3. ESD/雷サージ」を参照してください。

5.2.4. Armadillo の状態を表すインジケータ

LED にて状態を表示しています。

有線 LAN の状態は「表 3.15. LAN LED の動作」を参照ください。

5.2.5. 個体識別情報の取得

設置時に Armadillo を個体ごとに識別したい場合、以下の情報を個体識別情報として利用できます。

- ・ 個体番号
- ・ MAC アドレス



Armadillo の設置前に個体識別情報を記録しておき、設置後の Armadillo を識別できるようにしておくことを推奨します。

これらの情報を取得する方法は以下のとおりです。状況に合わせて手段を選択してください。

- ・ 本体シールから取得する
- ・ コマンドによって取得する

5.2.5.1. 本体シールから取得

Armadillo の各種個体番号、MAC アドレスなどの個体識別情報は、ケース裏や基板本体に貼付されているシールに記載されています。製品モデル毎に記載されている内容やシールの位置が異なるので、詳細は各種納入仕様書を参照してください。

5.2.5.2. コマンドによる取得

シールだけでなくコマンドを実行することによっても個体識別情報を取得することができます。以下に個体番号と MAC アドレスを取得する方法を説明します。

個体番号を取得する場合、「図 5.1. 個体番号の取得方法 (device-info)」に示すコマンドを実行してください。device-info はバージョン v3.18.4-at.7 以降の ABOS に標準で組み込まれています。

```
[armadillo ~]# device-info -s  
00C900010001 ❶
```

図 5.1 個体番号の取得方法 (device-info)

❶ 使用している Armadillo の個体番号が表示されます。

device-info がインストールされていない場合は「図 5.2. device-info のインストール方法」に示すコマンドを実行することでインストールできます。

```
[armadillo ~]# persist_file -a update  
[armadillo ~]# persist_file -a add device-info
```

図 5.2 device-info のインストール方法

上記の方法で device-info をインストールできない場合は最新のバージョンの ABOS にアップデートすることを強く推奨します。非推奨ですが、ABOS をアップデートせずに個体番号を取得したい場合は

「図 5.3. 個体番号の取得方法 (get-board-info)」に示すように get-board-info を実行することでも取得できます。

```
[armadillo ~]# persist_file -a add get-board-info
[armadillo ~]# get-board-info -s
00C900010001 ❶
```

図 5.3 個体番号の取得方法 (get-board-info)

- ❶ 使用している Armadillo の個体番号が表示されます。



コンテナ上で個体番号を表示する場合は、個体番号を環境変数として設定することで可能となります。「図 5.4. 個体番号の環境変数を conf ファイルに追記」に示す内容を /etc/atmark/containers の下の conf ファイルに記入します。

```
add_args --env=SERIALNUM=$(device-info -s) ❶
```

図 5.4 個体番号の環境変数を conf ファイルに追記

- ❶ コンテナ起動毎に環境変数 SERIALNUM に値がセットされます。

「図 5.5. コンテナ上で個体番号を確認する方法」に示すコマンドを実行することでコンテナ上で個体番号を確認することができます。

```
[container ~]# echo $SERIALNUM
00C900010001
```

図 5.5 コンテナ上で個体番号を確認する方法

次に MAC アドレスを取得する方法を説明します。「図 5.6. MAC アドレスの確認方法」に示すコマンドを実行することで、各インターフェースの MAC アドレスを取得できます。

```
[armadillo ~]# ip addr
: (省略)
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
    link/ether 00:11:0c:12:34:56 brd ff:ff:ff:ff:ff:ff ❶
: (省略)
```

図 5.6 MAC アドレスの確認方法

- ❶ link/ether に続くアドレスが MAC アドレスです。

また、出荷時にアットマークテクノが書き込んだ Ethernet MAC アドレスは「図 5.7. 出荷時の Ethernet MAC アドレスの確認方法」に示すコマンドを実行することで取得することができます。

```
[armadillo ~]# device-info -m  
eth0: 00:11:0C:12:34:56 ❶
```

図 5.7 出荷時の Ethernet MAC アドレスの確認方法

- ❶ 出荷時にアットマークテクノが書き込んだ Ethernet MAC アドレスが表示されます。

ただし、「図 5.7. 出荷時の Ethernet MAC アドレスの確認方法」で示すコマンドでは、お客様自身で設定した Ethernet MAC アドレスを取得することはできないのでご注意ください。お客様自身で設定した Ethernet MAC アドレスを取得したい場合は「図 5.6. MAC アドレスの確認方法」に示すコマンドを実行してください。

5.2.6. 電源を切る

Armadillo の電源を切る場合は、`poweroff` コマンドを実行してから電源を切るのが理想的です。しかし、設置後はコマンドを実行できる環境にない場合が多いです。この場合、条件が整えば `poweroff` コマンドを実行せずに電源を切断しても安全に終了できる場合があります。

詳細は、「3.1.7.4. Armadillo の終了方法」を参照してください。

5.3. ABOSDE で開発したアプリケーションをアップデートする

ABOSDE で開発したアプリケーションのアップデートは、開発時と同様に ABOSDE を用いて行うことが出来ます。

「3.12. ABOSDE によるアプリケーションの開発」で示したように、開発時にはリリース版のアプリケーションを Armadillo にインストールするために、VS Code の左ペインの [Generate release swu] を実行して `release.swu` を作成しました。

アップデート時にも、アップデートに必要なアプリケーションの編集をした後に [Generate release swu] を実行して、アップデート版のアプリケーションを含む `release.swu` を作成します。

具体的な ABOSDE を用いたアプリケーションのアップデートの流れは「5.3.1. アプリケーションのアップデート手順」に示します。

5.3.1. アプリケーションのアップデート手順

ここでは、プロジェクト名を `my_project` としています。

5.3.1.1. アップデートするアプリケーションのプロジェクトを VS Code で開く

「図 5.8. VS Code を起動」で示すように、アップデートするアプリケーションのプロジェクトを指定して VS Code を起動してください。

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

図 5.8 VS Code を起動

5.3.1.2. アップデート前のバージョンのプロジェクトを管理する

ABOSDE では、プロジェクトのバージョン管理は行っていません。必要な場合はユーザー自身でアップデート前のプロジェクトを管理してください。



アップデート前のプロジェクトの release.swu のバージョンを知りたい場合は「6.6. SWU イメージの内容の確認」を参照してください。

5.3.1.3. アプリケーションのソースコードを編集しテストする

既存のアプリケーションのソースコードを編集した後、「3.12. ABOSDE によるアプリケーションの開発」を参考に、アプリケーションが Armadillo 上で問題なく動作するかテストを行ってください。

5.3.1.4. アップデート用の swu を作成する

VS Code の左ペインの [Generate release swu] を実行してください。my_project ディレクトリ下に release.swu というファイル名で SWU ファイルが作成されます。

5.3.1.5. 運用中の Armadillo のアプリケーションをアップデートする

アプリケーションをアップデートするために、作成した release.swu を運用中の Armadillo にインストールしてください。SWU イメージファイルをインストールする方法は「3.3.3.6. SWU イメージのインストール」を参照してください。

5.4. Armadillo のソフトウェアをアップデートする

設置後の Armadillo のソフトウェアアップデートは SWUpdate を使用することで実現できます。

ここでは、ソフトウェアのアップデートとして以下のような処理を行うことを例として説明します。

- ・すでに Armadillo に sample_container_image というコンテナイメージがインストールされている
- ・sample_container_image のバージョンを 1.0.0 から 1.0.1 にアップデートする
- ・sample_container_image からコンテナを自動起動するための設定ファイル (sample_container.conf) もアップデートする

5.4.1. SWU イメージの作成

アップデートのために SWU イメージを作成します。SWU イメージの作成には、mkswu というツールを使います。「3.1. 開発の準備」で作成した環境で作業してください。

5.4.1.1. SBOM の生成

SWU イメージ作成時に、同時に SBOM を生成することができます。詳細は「6.22. SBOM の提供」を参照してください。

5.4.2. mkswu の desc ファイルを作成する

SWU イメージを生成するには、desc ファイルを作成する必要があります。

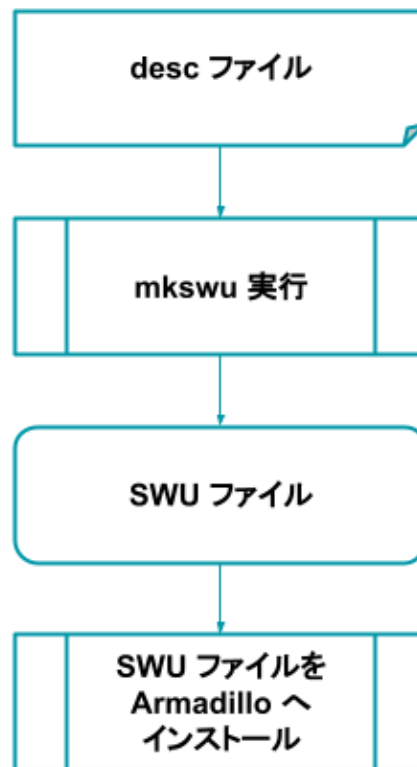


図 5.9 desc ファイルから Armadillo へ SWU イメージをインストールする流れ

desc ファイルとは、SWU イメージを Armadillo にインストールする際に行われる命令を記述したものです。/usr/share/mkswu/examples/ ディレクトリ以下にサンプルを用意していますので、やりたいことに合わせて編集してお使いください。なお、desc ファイルの詳細な書式については、「6.4. mkswu の .desc ファイルを編集する」を参照してください。

まず、以下のようなディレクトリ構成で、sample_container.conf を作成しておきます。設定ファイルの内容については割愛します。

```
[ATDE ~/mkswu]$ tree container_start
container_start
├── etc
│   └── atmark
│       └── containers
│           └── sample_container.conf
```

このような階層構造にしているのは、インストール先の Armadillo 上で sample_container.conf を /etc/atmark/containers/ の下に配置したいためです。

次に、アップデート先のコンテナイメージファイルである sample_container_image.tar を用意します。コンテナイメージを tar ファイルとして出力する方法を「図 5.10. コンテナイメージアーカイブ作成例」に示します。

```
[armadillo ~]# podman save sample_container:[VERSION] -o sample_container_image.tar
```

図 5.10 コンテナイメージアーカイブ作成例

次に、sample_container_update.desc という名前で desc ファイルを作成します。「図 5.11. sample_container_update.desc の内容」に、今回の例で使用する sample_container_update.desc ファイルの内容を示します。sample_container_image.tar と、コンテナ起動設定ファイルを Armadillo にインストールする処理が記述されています。

```
[ATDE ~/mkswu]$ cat sample_container_update.desc
swdesc_option version=1.0.1

swdesc_usb_container "sample_container_image.tar" ❶
swdesc_files --extra-os "container_start" ❷
```

図 5.11 sample_container_update.desc の内容

- ❶ sample_container_image.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。
- ❷ container_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。

5.4.3. desc ファイルから SWU イメージを生成する

mkswu コマンドを実行することで、desc ファイルから SWU イメージを生成できます。

```
[ATDE ~/mkswu]$ mkswu -o sample_container_update.swu sample_container_update.desc ❶
[ATDE ~/mkswu]$ ls sample_container_update.swu ❷
sample_container_update.swu
```

図 5.12 sample_container_update.desc の内容

- ❶ mkswu コマンドで desc ファイルから SWU イメージを生成
- ❷ sample_container_update.swu が生成されていることを確認

作成された SWU イメージの内容を確認したい場合は、「6.6. SWU イメージの内容の確認」を参照してください。

5.4.4. イメージのインストール

インストールの手順については、「3.3.3.6. SWU イメージのインストール」を参照してください。

5.5. Armadillo Twin から複数の Armadillo をアップデートする

Armadillo Twin を使用することで、自身でサーバー構築を行うことなくネットワーク経由で SWU イメージを配信し、デバイスのソフトウェアを更新することができます。Armadillo Twin を使用したソフトウェアアップデートを行うためには、Armadillo Twin へのデバイスの登録が完了している必要があります。Armadillo Twin へのデバイスの登録方法については、「5.1. Armadillo Twin に Armadillo を登録する」をご確認ください。また、Armadillo Twin を使用したソフトウェアアップデートの実施方法については、Armadillo Twin ユーザーマニュアル「デバイスのソフトウェアをアップデートする」[<https://manual.armadillo-twin.com/update-software/>] をご確認ください。

5.6. eMMC の寿命を確認する

5.6.1. eMMC について

eMMC とは embedded Multi Media Card の頭文字を取った略称で NAND 型のフラッシュメモリを利用した内蔵ストレージです。当社で使用しているものは長期間運用を前提としている為、使用する容量を半分以下にして SLC モードで使用しています。(例えば 32GB 製品を 10GB で使用、残り 22GB は予備領域とする)。

eMMC は耐性に問題が発生した個所を内部コントローラがマスクし、予備領域を割り当てて調整しています。絶対ではありませんが、この予備領域がなくなると書き込みが出来なくなる可能性があります。

5.6.2. eMMC 予備領域の確認方法

Armadillo Base OS には emmc-utils というパッケージがインストールされています。

「図 5.13. eMMC の予備領域使用率を確認する」に示すコマンドを実行し、EXT_CSD_PRE_EOL_INFO の内容を確認することで eMMC の予備領域の使用率がわかります。EXT_CSD_PRE_EOL_INFO の値と意味の対応を「表 5.1. EXT_CSD_PRE_EOL_INFO の値の意味」に示します。

```
[armadillo ~]# mmc extcsd read /dev/mmcblk0 | grep EXT_CSD_PRE_EOL_INFO  
eMMC Pre EOL information [EXT_CSD_PRE_EOL_INFO]: 0x01
```

図 5.13 eMMC の予備領域使用率を確認する

表 5.1 EXT_CSD_PRE_EOL_INFO の値の意味

値	意味
0x01	定常状態(問題無し)
0x02	予備領域 80% 以上使用
0x03	予備領域 90% 以上使用

また、Armadillo Twin からも eMMC の予備領域使用率を確認することができます。詳細は Armadillo Twin ユーザーマニュアル「デバイス監視アラートを管理する」 [<https://manual.armadillo-twin.com/management-device-monitoring-alert/>]をご確認ください。

5.7. Armadillo の部品変更情報を知る

Armadillo に搭載されている部品が変更された場合や、製品が EOL となった場合には以下のページから確認できます。

Armadillo サイト - 変更通知(PCN)/EOL 通知

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

また、Armadillo サイトにユーザー登録していただくと、お知らせをメールで受信することが可能です。変更通知についても、メールで受け取ることが可能ですので、ユーザー登録をお願いいたします。

ユーザー登録については「3.1.8. ユーザー登録」を参照してください。

5.8. Armadillo を廃棄する

運用を終了し Armadillo を廃棄する際、セキュリティの観点から以下のようなことを実施する必要があります。

- ・ 設置場所に Armadillo を放置せず回収する
- ・ Armadillo をネットワークから遮断する
 - ・ SIM カードが挿入されているのであれば抜き、プロバイダーとの契約を終了する
 - ・ 無線 LAN の設定を削除する
 - ・ 接続しているクラウドのデバイス証明書を削除・無効にすることでクラウドに接続できなくする
- ・ 「3.1.4. Armadillo の初期化と ABOS のアップデート」の手順にしたがって初期化を行う
- ・ インストールディスクは、blkdiscard コマンドを用いて eMMC の GPP を含む全てのパーティション内のデータを消去しています
- ・ 物理的に起動できなくする
- ・ Armadillo Twin をご利用の場合は、Armadillo Twin 上で当該デバイスの廃棄手続きを行うか、Armadillo Twin を解約する際に廃棄する旨を申告する
 - ・ 実際の手続きなど詳細は Armadillo Twin お問い合わせフォーム [<https://apps.armadillo-twin.com/ja/inquiry>]からお問い合わせください
- ・ Armadillo を物理的に破壊する
 - ・ SD ブートを無効化していてインストールディスクが適用できない場合や、情報吸い出しのリスクをより下げたい場合は、eMMC やセキュアエレメント等の記憶媒体を物理的に破壊してください
 - ・ 詳細な破壊箇所や破壊方法については、弊社までお問い合わせください

6. 応用編

本章では、ここまでの内容で紹介しきれなかった、より細かな Armadillo の設定方法や、開発に役立つヒントなどを紹介します。

各トピックを羅列していますので、目次の節タイトルからやりたいことを探して辞書的にご使用ください。

6.1. persist_file について

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。SWUpdate に関しては「3.3.3. アップデート機能について」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「6.5. 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 6.1. 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 overlayfs lower directories
so might need a reboot to take effect
```

図 6.1 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 6.2 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 6.3 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 6.4 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。
 - ❷ 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 6.5 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。

6.2. コンテナ

Armadillo Base OS において、ユーザーアプリケーションは基本的にコンテナ内で実行されます。「3. 開発編」で紹介した開発手順では、基本的に SWUpdate を使用してコンテナを生成・実行していました。

以下では、より自由度の高いコンテナの操作のためにコマンドラインからの操作方法について紹介します。

6.2.1. 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 방법은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。

6.2.2. コンテナの基本的な操作

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。

6.2.2.1. 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「6.2.4. コンテナ起動設定ファイルを作成する」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 6.6 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「6.2.4. コンテナ起動設定ファイルを作成する」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。

6.2.2.2. 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 6.7 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 6.8 podman images --help の実行例

6.2.2.3. 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
my_container
```



図 6.9 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 6.10 podman ps --help の実行例

6.2.2.4. 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(vethe172e161) entered blocking state
[ 3119.101231] cni-podman0: port 1(vethe172e161) entered disabled state
[ 3119.107745] device vethe172e161 entered promiscuous mode
[ 3119.113185] cni-podman0: port 1(vethe172e161) entered blocking state
[ 3119.119546] cni-podman0: port 1(vethe172e161) entered forwarding state
my_container
[ 3119.620731] cni-podman0: port 1(vethe172e161) entered disabled state
[ 3119.627696] device vethe172e161 left promiscuous mode
[ 3119.632762] cni-podman0: port 1(vethe172e161) entered disabled state
```

図 6.11 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 6.12 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 6.13 `podman start --help` 実行例

6.2.2.5. 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 6.14 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 6.15 `podman stop --help` 実行例

6.2.2.6. 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 6.16 `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. 「3.3.4.1. 電源を切っても保持されるディレクトリ(ユーザーデータディレクトリ)」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` か `/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。

6.2.2.7. 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/arm64v8/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/arm64v8/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 6.17 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
9e9d9366072751007b2e70544d76c46b95a7a5a02df658ef0fa3f7dccccf8850a

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

図 6.18 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「6.4. mkswu の .desc ファイルを編集する」の swdesc_embed_container と swdesc_usb_container で使えます。

6.2.2.8. 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 6.19 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。

1. podman rm --help 実行例

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

6.2.2.9. 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: 02480aeb44d7f1a44b8791af7edf7d6e1b18707397a1dfb3ff4f21c5ce4a44f
[armadillo ~]# podman images
REPOSITORY          TAG          IMAGE ID        CREATED        SIZE
```

図 6.20 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 6.21 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「6.2.2.16. イメージを eMMC に保存する」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
"02480aeb44d7f1a44b8791af7edf7d6e1b18707397a1dfb3ff4f21c5ce4a44f"
[armadillo ~]# abos-ctrl podman-rw rmi docker.io/alpine
Untagged: docker.io/library/alpine:latest
Deleted:
02480aeb44d7f1a44b8791af7edf7d6e1b18707397a1dfb3ff4f21c5ce4a44f
[armadillo ~]# podman images
REPOSITORY          TAG          IMAGE ID        CREATED        SIZE
```

図 6.22 Read-Only のイメージを削除する実行例

6.2.2.10. 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 6.23 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 6.24 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 6.25 `podman exec --help` 実行例

6.2.2.11. 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 6.26 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 6.27 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に対し `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 6.28 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。

6.2.2.12. pod でコンテナのネットワークネームスペースを共有する

`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 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 6.29 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。

.conf ファイルで使用できる各種パラメータについては、「6.2.4. コンテナ起動設定ファイルを作成する」を参照してください。

6.2.2.13. network の作成

podman_start で podman の network も作成できます。

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 198.51.100.0/24

[armadillo ~]# cat /etc/atmark/containers/nginx.conf
set_image docker.io/library/nginx:alpine
add_ports 80:80
set_ip 198.51.100.10
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 6.30 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。

.conf ファイルで使用できる各種パラメータについては、「6.2.4. コンテナ起動設定ファイルを作成する」を参照してください。

6.2.2.14. 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。



コンテナの設定によって podman の socket へのパスが自動設定されない場合もあります。podman --remote でエラーが発生した場合に `CONTAINER_HOST=unix:/path/to/podman.sock` で socket へのパスを設定してください。

Armadillo のホスト側の `udev rules` からコンテナを起動する場合は `podman_start` 等を直接実行すると `udev` の子プロセス管理によってコンテナが停止されますので、その場合はサービスを有効にし、`podman_start --create <container>` コマンドまたは `set_autostart create` の設定でコンテナを生成した上 `podman --remote start <container>` で起動してください。

6.2.2.15. リモートリポジトリにコンテナを送信する

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「5.4. 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。
```

6.2.2.16. イメージを eMMC に保存する

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「6.2.2.17. イメージを SWUpdate で転送する」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「6.2.2.6. 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 6.31 `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 6.32 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 コピーされたイメージを確認します。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、「6.2.4. コンテナ起動設定ファイルを作成する」を参考にして、`/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。

6.2.2.17. イメージを SWUpdate で転送する

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 arm64 docker.io/nginx:alpine
[ATDE ~/mkswu]$ podman run --rm docker.io/nginx:alpine uname -m
aarch64
[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 arm64 docker.io/nginx:alpine
[ATDE ~/mkswu]$ podman run --rm docker.io/nginx:alpine uname -m
aarch64
[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。
```

6.2.2.18. 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。

6.2.3. コンテナとコンテナに関連するデータを削除する



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。

6.2.3.1. VS Code から実行する

VS Code 上で ABOSDE(Armadillo Base OS Development Environment) から、Armadillo のコンテナイメージを全て削除する SWU イメージを作成することができます。

VS Code の左ペインの [COMMON PROJECT COMMAND] から [Generate Container Clear Swu] を実行すると、SWU イメージが作成されます。SWU イメージは `~/mkswu/container_clear.swu` に保存されます。

この SWU イメージを 「3.3.3.6. SWU イメージのインストール」 を参照して Armadillo ヘインストールしてください。

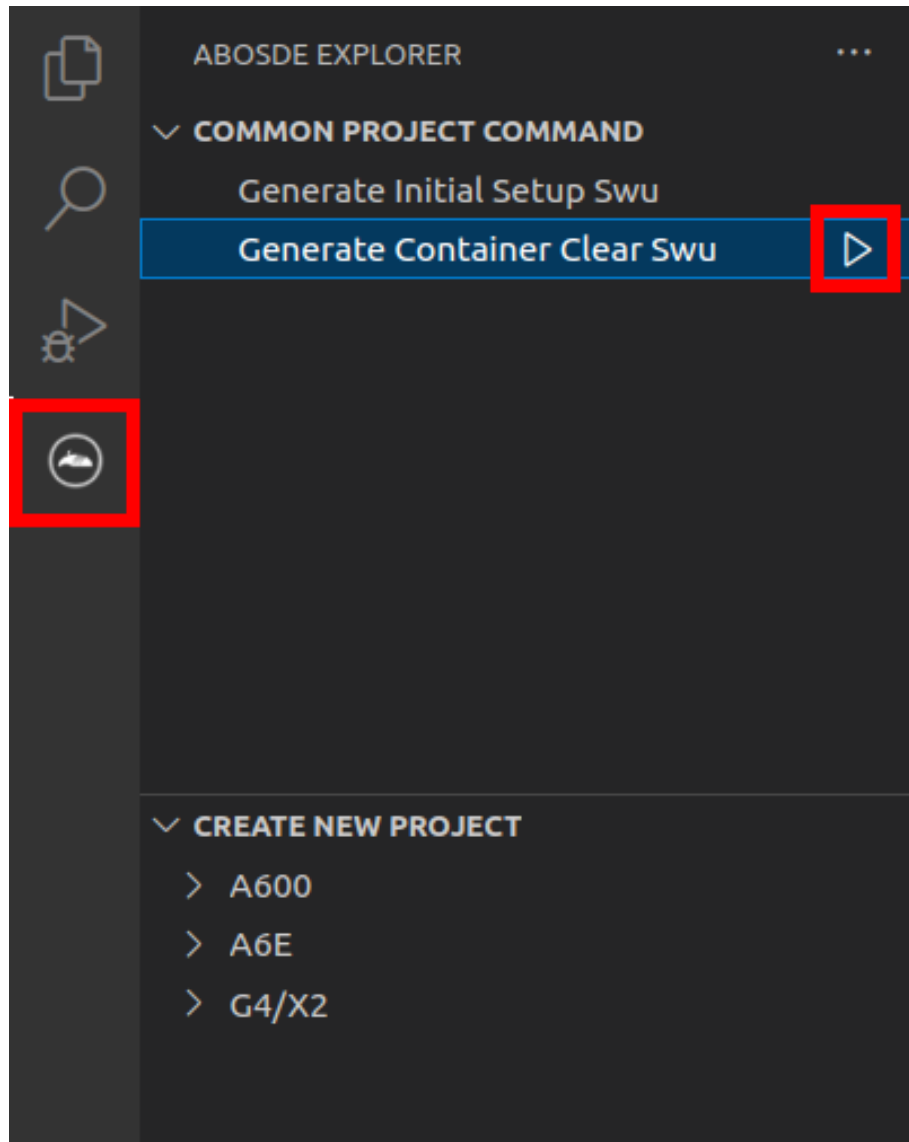


図 6.33 Armadillo 上のコンテナイメージを削除する

6.2.3.2. コマンドラインから実行する

`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。

`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 6.34 abos-ctrl container-clear 実行例

6.2.4. コンテナ起動設定ファイルを作成する

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 6.35 コンテナを自動起動するための設定例

.conf ファイルに設定可能なパラメーターの説明を以下に記載します。podman_start --long-help コマンドでも詳細を確認できます。

6.2.4.1. 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

6.2.4.2. 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。

6.2.4.3. 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/galcore /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。

6.2.4.4. 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ができます。

ホストパスは以下のどれ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側のパスを分割する意味があるため、ファイル名に「:」を使用することはできません。ホスト側のパスにのみ「:」が含まれてる場合は「add_volumes "[ホストパス]" "[コンテナパス]" "[オプション]" 」と指定することで設定できます。



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_devices /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 6.36 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。

6.2.4.5. 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 6.1. `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 6.1 `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 6.1. `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 6.37. `/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 `pty` などを指定できることがわかります。

```
[armadillo ~]# cat /proc/devices
Character devices:
 1 mem
 2 pty
 3 tty
 4 /dev/vc/0
 4 tty
 4 ttyS
 5 /dev/tty
 5 /dev/console
 5 /dev/ptmx
 7 vcs
10 misc
13 input
29 fb
81 video4linux
: (省略)
```

図 6.37 `/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 video4linux sd`

6.2.4.6. 個体識別情報の環境変数の追加

`add_armadillo_env`

アットマークテクノが設定した個体識別情報をコンテナの環境変数として追加することができます。

例: `add_armadillo_env`

`add_armadillo_env` を設定することで追加されるコンテナの環境変数について、「表 6.2. `add_armadillo_env` で追加される環境変数」に示します。

表 6.2 `add_armadillo_env` で追加される環境変数

環境変数	環境変数の説明	表示例
<code>AT_ABOS_VERSION</code>	ABOS のバージョン	3.18.4-at.5
<code>AT_LAN_MAC1</code>	アットマークテクノが設定した LAN1 (eth0) の MAC アドレス	00:11:0C:12:34:56
<code>AT_PRODUCT_NAME</code>	製品名	Armadillo-640
<code>AT_SERIAL_NUMBER</code>	個体番号	00C900010001

「表 6.2. `add_armadillo_env` で追加される環境変数」に示した環境変数をコンテナ上で確認する場合、「図 6.38. `add_armadillo_env` で設定した環境変数の確認方法」に示すコマンドを実行してください。ここでは、個体番号の環境変数を例に示します。

```
[container ~]# echo $AT_SERIAL_NUMBER
00C900010001
```

図 6.38 `add_armadillo_env` で設定した環境変数の確認方法

お客様が独自の環境変数をコンテナに追加する場合は「図 5.4. 個体番号の環境変数を `conf` ファイルに追記」を参考に `conf` ファイルを編集してください。

6.2.4.7. pod の選択

`set_pod` [ポッド名]

「6.2.2.12. pod でコンテナのネットワークネームスペースを共有する」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`

6.2.4.8. ネットワークの選択

`set_network` [ネットワーク名]

この設定に「6.2.2.13. 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` しかないネームスペースに入ります。

`host` の場合は OS のネームスペースをそのまま使います。

例: `set_network mynetwork`

6.2.4.9. 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。

6.2.4.10. 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。

6.2.4.11. 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`

6.2.4.12. 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`

6.2.4.13. 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`

6.2.4.14. podman logs 用のログサイズ設定

set_log_max_size <サイズ>

podman logs でログを表示するために /run にログファイルを保存しています。そのログのサイズが設定したサイズを越えるとクリアされます。デフォルトは「1MB」です。

6.2.4.15. podman のフックの仕組み

add_hook --stage <ステージ> [--] コマンド [コマンド引数]

コンテナが起動されるなど、動作ステージの変化をフックとしてコマンドを実行します。複数のステージで実行したい場合は --stage オプションを複数設定してください。

指定可能なステージは precreate, prestart, createRuntime, createContainer, startContainer, poststart, と poststop です。ステージの意味や使用方法の詳細は podman のドキュメンテーションを参照してください。



Armadillo Base OS 3.19.1-at.4 現在では set_restart によるコンテナの再起動でも 1 度目の停止時のみ poststop フックが実行されます。2 度目以降の停止では実行されませんのでご注意ください。

6.2.4.16. ヘルスチェック機能の設定

set_healthcheck [引数] [--] コマンド [コマンド引数]

定期的にコマンドを実行して、コンテナの正常性を確認します。指定可能な引数は以下のとおりです：

- ・ **--retries <リトライ数>**: エラーを検知するまでのリトライ回数。(デフォルト: 3)
- ・ **--action <none|restart|kill|stop|reboot|rollback>**: 指定したリトライ回数分連続でチェックが失敗したときのアクション (デフォルト: restart) :
 - ・ none: set_healthcheck_fail_command に指定した処理を実行する以外何もしません。
 - ・ restart: コンテナを再起動します。set_restart オプションと異なり、コンテナを起動しなおし初期状態で再起動します。
 - ・ kill/stop: コンテナを停止します。
 - ・ reboot: Armadillo を再起動します。
 - ・ rollback: ロールバック可能な場合はロールバックして Armadillo を再起動します。ロールバック不可能な場合はそのまま Armadillo を再起動します。
- ・ **--interval <時間>**: チェックする時間間隔です。(デフォルト: 1 min)
- ・ **--start-period <時間>**: 最初のチェックを実行する前の待ち時間です。(デフォルト: interval 設定の値)
- ・ **--timeout <秒数>**: 設定された時間以内にヘルスチェックが終了しなかった場合は失敗となります。(デフォルト: 無し)

また、いくつかのタイミングでコマンドを実行させることができます：

- ・ **set_healthcheck_start_command** コマンド [コマンド引数]: コンテナ起動後にヘルスチェックが初めて成功した際に実行されるコマンドです。
- ・ **set_healthcheck_fail_command** コマンド [コマンド引数]: ヘルスチェックが retries 回失敗した後に実行されるコマンドです。このコマンドは set_healthcheck の --action 設定の前に実行されますので、コマンドだけを実行したい場合は --action none で無効化してください。
- ・ **set_healthcheck_recovery_command** コマンド [コマンド引数]: ヘルスチェックが retries 回失敗した後に再び成功した際に実行されるコマンドです。コンテナを起動する際に成功せずに失敗した場合は、その 1 回目の成功の際に set_healthcheck_start_command で設定されたコマンドのみが実行されます。

例: set_healthcheck -- curl -s --fail http://localhost:8080/status

例: set_healthcheck_start_command abos-ctrl rollback-clone

```
armadillo:~# grep podman_atmark /var/log/messages
Jun 20 11:33:21 armadillo user.notice podman_atmark: my_container healthcheck is now healthy (was starting)
Jun 20 11:33:21 armadillo user.notice podman_atmark: my_container first healthy check: running abos-ctrl rollback-clone
Jun 20 11:40:21 armadillo user.notice podman_atmark: my_container healthcheck failed (from healthy, 1 / 3)
Jun 20 11:41:21 armadillo user.notice podman_atmark: my_container healthcheck failed (from healthy, 2 / 3)
Jun 20 11:42:21 armadillo user.notice podman_atmark: my_container healthcheck failed (from healthy, 3 / 3)
Jun 20 11:42:21 armadillo user.notice podman_atmark: my_container is unhealthy, restarting container
Jun 20 11:43:21 armadillo user.notice podman_atmark: my_container healthcheck is now healthy (was failed)
```

図 6.39 上記の例でエラーを発生させた際の起動ログ

6.2.4.17. 自動起動の無効化

set_autostart no または **set_autostart create**

Armadillo の起動時にコンテナを自動起動しないように設定できます。

create を指定した場合はコンテナは生成されており、podman start <name> で起動させることができます。

no を指定した場合は 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。

6.2.4.18. 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"`

6.2.4.19. コンテナ起動前にコマンドを実行する

`add_pre_command` [コマンド]

コンテナを起動する直前に設定したコマンドを実行します。

Armadillo Base OS の環境で実行されてますので、ハードウェアの設定等に適切です。

また、複数のコマンドを実行する場合は順番に実行されます。設定したコマンドが1つでも失敗した場合は、コンテナは起動されません。

例: `add_pre_command gpioset gpiochipX Y=1`

6.2.4.20. `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`

6.2.5. 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。ここでは以下の3つの手順について説明します。

- ・ ABOSDE からインストールする方法
- ・ Docker ファイルからイメージをビルドする方法
- ・ すでにビルド済みのイメージを使う方法

6.2.5.1. ABOSDE からインストールする

1. インストール用のプロジェクトを作成する

VS Code の左ペインの [A600] から [Atmark Container 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。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。この操作により、選択した保存先に、入力したプロジェクト名と同名のディレクトリが作成されます。

また、ここでは次のように設定しています。

- ・ **保存先** : ホームディレクトリ
- ・ **プロジェクト名** : `my_project`

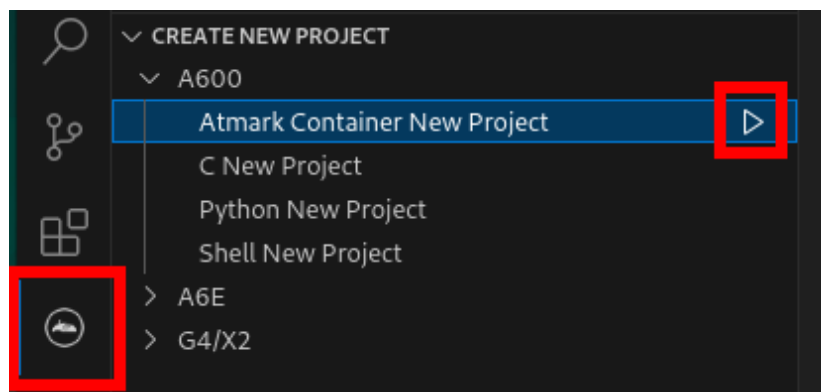


図 6.40 インストール用のプロジェクトを作成する

2. SWU イメージを作成する

VS Code の左ペインの [my_project] から [Generate at-debian-image container setup swu] を実行してください。

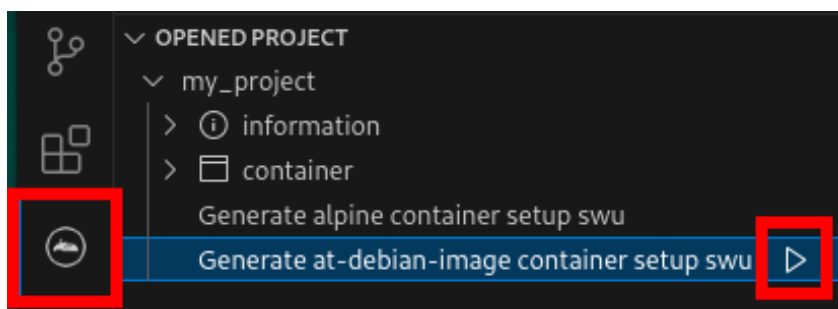


図 6.41 at-debian-image のコンテナイメージをインストールする SWU ファイルを作成する

作成した SWU ファイルは container_setup/at-debian-image/at-debian-image-armv7.swu に保存されています。この SWU イメージを 「3.3.3.6. SWU イメージのインストール」 を参照して Armadillo ヘインストールしてください。

3. SBOM 生成に関わる設定を行う

ABOSDE から作成した場合は SBOM が同時に生成されます。詳細は 「3.15. SBOM 生成に関わる設定を行う」 をご確認ください。SBOM の生成には以下の二つのファイルが必要です。

- ・ コンフィグファイル
- ・ desc ファイル

SBOM の生成にはライセンス情報を示したコンフィグファイルを使用します。コンフィグファイルは container_setup/at-debian-image-armv7.sbom_config.yaml.tpl になります。SWU イメージ作成時にこのコンフィグファイルからバージョン番号をアップデートした container_setup/at-debian-image-armv7.sbom_config.yaml が生成されます。

リリース時にはコンフィグファイルの内容を確認し、正しい内容に変更してください。各項目の詳細な説明については SPDX specification v2.2.2 (<https://spdx.github.io/spdx-spec/v2.2.2/>) をご覧ください。SBOM に含めるコンテナイメージ等の情報については desc ファ

イルに記載されています。各項目の説明については「6.22.4.2. desc ファイルを編集する」をご覧ください。

6.2.5.2. 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-[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-[VERSION].tar.gz
[armadillo ~]# cd at-debian-image-dockerfile-[VERSION]
[armadillo ~]# abos-ctrl podman-storage --disk
[armadillo ~]# podman build -t at-debian-image:latest .
:
: (省略)
:
[armadillo ~]# podman images
REPOSITORY          TAG          IMAGE ID          CREATED           SIZE
localhost/at-debian-image latest       c8e8d2d55456     About a minute ago 233 MB
docker.io/library/debian bullseye     723b4a01cd2a     18 hours ago      123 MB
```

図 6.42 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 がビルドされたことが確認できます。library/debian イメージはベースとなっている Debian イメージです。

6.2.5.3. 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-[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 ~]# podman load -i at-debian-image-[VERSION].tar
:
: (省略)
:
[armadillo ~]# podman images
REPOSITORY          TAG          IMAGE ID          CREATED           SIZE
localhost/at-debian-image [VERSION]    93a4ec873ac5     17 hours ago     233 MB
localhost/at-debian-image latest       93a4ec873ac5     17 hours ago     233 MB
```

図 6.43 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 がビルドされたことが確認できます。

6.2.6. alpine のコンテナイメージをインストールする

alpine のコンテナイメージは、ABOSDE を用いてインストールすることが可能です。「6.2.5.1. ABOSDE からインストールする」を参照して、インストール用のプロジェクトを作成しておいてください。

VS Code の左ペインの [my_project] から [Generate alpine container setup swu] を実行してください。

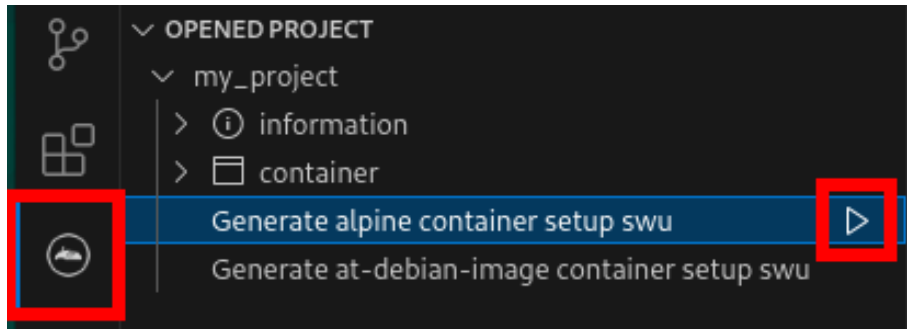


図 6.44 alpine のコンテナイメージをインストールする SWU ファイルを作成する

作成した SWU ファイルは `container_setup/alpine/alpine.swu` に保存されています。この SWU イメージを「3.3.3.6. SWU イメージのインストール」を参照して Armadillo へインストールしてください。

6.2.6.1. SBOM 生成に関わる設定を行う

ABOSDE から作成した場合は SBOM が同時に生成されます。詳細は「3.15. SBOM 生成に関わる設定を行う」をご確認ください。SBOM の生成には以下の二つのファイルが必要です。

- ・ コンフィグファイル
- ・ desc ファイル

SBOM の生成にはライセンス情報を示したコンフィグファイルを使用します。コンフィグファイルは `container_setup/alpine.sbom_config.yaml.tpl` になります。SWU イメージ作成時にこのコンフィグファイルからバージョン番号をアップデートした `container_setup/alpine.sbom_config.yaml` が生成されます。

リリース時にはコンフィグファイルの内容を確認し、正しい内容に変更してください。各項目の詳細な説明については SPDX specification v2.2.2 (<https://spdx.github.io/spdx-spec/v2.2.2/>) をご覧ください。SBOM に含めるコンテナイメージ等の情報については desc ファイルに記載されています。各項目の説明については「6.22.4.2. desc ファイルを編集する」をご覧ください。

6.2.7. コンテナの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。

6.2.7.1. 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 6.45 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
3: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether xx:xx:xx:xx:xx:xx brd ff:ff:ff:ff:ff:ff
    inet 10.88.0.17/16 brd 10.88.255.255 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::40e5:98ff:feec:4b17/64 scope link
        valid_lft forever preferred_lft forever
```

図 6.46 ip コマンドを用いたコンテナの IP アドレス確認例

6.2.7.2. 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 198.51.100.0/24 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 198.51.100.0/24
[armadillo ~]# podman_start my_network
Creating network 'my_network'
my_network
```

図 6.47 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 198.51.100.10 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 198.51.100.10
[armadillo ~]# podman_start network_example
```

```
Starting 'network_example'
3ea8c9031bf833228908bd73d8929b1d543b189b436c218e0634e0d39409e100
```

図 6.48 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、198.51.100.10 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
198.51.100.10
```

図 6.49 コンテナの IP アドレス確認例

6.2.8. コンテナ内にサーバを構築する

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。

6.2.8.1. 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 6.50 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 6.51 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。

6.2.8.2. 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 6.52 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 6.53 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 6.54 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 6.55 vsftpd の起動例

6.2.8.3. 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 6.56 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 6.57 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 ~]# smb
```

図 6.58 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。

6.2.8.4. 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'
114c5f1dbb7e81293dcb8f8e0c600b861626375b14cfe4023761acaa84fdcad1
[armadillo ~]# podman exec -it sqlite_example sh
[container ~]# apk upgrade && apk add sqlite
```

図 6.59 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 6.60 sqlite の実行例

6.2.9. 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。

6.2.9.1. 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「6.2.5. 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
set_command sleep infinity
add_devices /dev/tty7 ❶
add_devices /dev/fb0 ❷
```

```
add_devices /dev/input ③
add_volumes /run/udev:/run/udev:ro ④
add_args --cap-add=SYS_ADMIN ⑤
[armadillo ~]# podman_start x_example
Starting 'x_example'
26847e21bd519f99466af32fdf0d809e2216d3e8ddf05c185e5428fe46e6a09b
```

図 6.61 X Window System を扱うためのコンテナ起動例

- ① 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。
- ② 画面描画先となるフレームバッファを設定します。
- ③ 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。
- ④ 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。
- ⑤ 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 25297ceb226c 5.10.52-1-at #2-Alpine SMP PREEMPT Thu Nov 18 09:10:13
UTC 2021 aarch64
Kernel command line: console=ttyMXC1,115200 root=/dev/mmcblk2p1 rootwait ro
Build Date: 13 April 2021 04:07:31PM
xorg-server 2:1.20.11-1 (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: Sun Nov 21 23:51:18 2021
(==) Using system config directory "/usr/share/X11/xorg.conf.d"
```

図 6.62 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。

6.2.9.2. 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 6.63 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 6.64 フレームバッファに直接描画する実行例

6.2.9.3. 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 6.65 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。

6.2.10. 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。

6.2.10.1. 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 6.66 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 6.3 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	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 6.67 サスペンド状態にする実行例

6.2.10.2. 起床要因を有効化する

サスペンド状態からの起床要因として、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/ttymux0/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		<pre>[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</pre>
RTC(i.MX6ULL)	起床要因	アラーム割り込み	
	有効化		<pre>[container ~]# echo enabled > /sys/bus/platform/devices/20cc000.snvs*:snvs-rtc-lp/power/wakeup</pre>
	実行例		<pre>[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' 8fbef3edda3b7fcea5b1f8cbf960cf469b7e82c4d1ecd35477076e81fc24e39f [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. ...</pre>

図 6.68 サスペンド状態にする実行例、rtc で起こす

RTC(NR3225SA)	起床要因	アラーム割り込み	
	有効化		<pre>[armadillo ~]# echo enabled > /sys/bus/i2c/devices/1-0032/power/wakeup</pre>
	実行例		<pre>[armadillo ~]# echo +180 > /sys/class/rtc/rtc0/wakealarm [armadillo ~]# poweroff</pre>

図 6.69 180 秒後に RTC(NR3225SA)で起床する

RTC(i.MX6ULL)と RTC(NR3225SA)はサスペンド状態だけではなく、poweroff コマンドを使用した電源の切断状態から復帰が可能です。RTC_BAT ピンからバックアップ電源が供給されている状態で、

ONOFF ピンまたは `poweroff` コマンドを使用して電源を切った場合、5V 電源を入れなおしても再起動しません。この状態から再起動する方法は上記の RTC(i.MX6ULL)、RTC(NR3225SA)または以下を参照してください。

- ・ ONOFF ピンの制御による電源の ON

詳細は、「3.4.7.5. 外部からの電源制御」参照してください。

6.2.10.3. パワーマネジメントの仕様

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  
ル
```

6.2.11. コンテナからの `poweroff` 及び `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'  
c8e3b9b418fc72395db9f3c22b1eb69eb41eaaaf790d3b7151047ef066cc4c8ff  
[armadillo ~]# podman exec -ti shutdown_example sh  
[container ~]# kill -USR2 1 (poweroff)  
[container ~]# kill -TERM 1 (reboot)
```

図 6.70 コンテナから `shutdown` を行う

6.2.12. 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。

6.2.12.1. 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 6.71 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 6.72 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（V 以外の）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（V 以外の）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 6.73 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 6.74 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

6.3. swupdate を使用してアップデートする

6.3.1. swupdate で可能なアップデート

swupdate を実行する目的としては以下が考えられます。

- a. コンテナをアップデートしたい
開発したコンテナのアップデートが可能です。
- b. ユーザーデータディレクトリや Armadillo Base OS のファイルを差分アップデートしたい
ユーザーデータをアップデートする場合は、以下のディレクトリを更新します。

- ・ /var/app/rollback/volumes

ユーザーディレクトリについては「3.3.4.1. 電源を切っても保持されるディレクトリ(ユーザーデータディレクトリ)」を参照してください。



SWUpdate による /var/app/volumes の更新は推奨しません。

/var/app/volumes が二面化されていないため、書込みの途中で問題が発生した場合、不完全な書込みになる恐れがあります。

また、アップデート中にアプリケーションがそのデータにアクセスすると、書込み中のデータにアクセスすることになります。

/var/app/volumes のデータに対して更新が必要な場合、SWUpdate では /var/app/rollback/volumes に更新するデータを書き込んでください。その後、次回起動時にアプリケーション側から /var/app/rollback/volumes に書き込んだデータを /var/app/volumes に移動するようにしてください。

c. Armadillo Base OS を一括アップデートしたい

アットマークテクノがリリースする Armadillo Base OS の機能追加、更新、セキュリティパッチの追加が可能です。

d. ブートローダーをアップデートしたい

アットマークテクノがリリースするブートローダーのアップデートが可能です。

「2.1.3. Armadillo Base OS とは」で示すように、Armadillo Base OS は OS・ブートローダー・コンテナ部分の安全性を担保するため二面化しています。

それにより、万が一アップデートに失敗した場合でも起動中のシステムに影響ありません。

以降、それぞれの目的ごとに swupdate によるアップデートの流れを示します。

・ a, b の場合

「6.3.2. コンテナのアップデート、ユーザーデータディレクトリや Armadillo Base OS の差分アップデート」を参照してください。

・ c の場合

「6.3.3. Armadillo Base OS の一括アップデート」を参照してください。

・ d の場合

「6.3.4. ブートローダーのアップデート」を参照してください。

6.3.2. コンテナのアップデート、ユーザーデータディレクトリや Armadillo Base OS の差分アップデート

以下にアップデートの流れを示します。

ここでは、boot して起動中の面を A 面、アップデート先の面を B 面とします。

1. Armadillo Base OS を B 面へコピー

Armadillo Base OS を B 面にコピーする流れを「図 6.75. Armadillo Base OS を B 面にコピー」に示します。

A 面と B 面の Armadillo Base OS が同期しているか確認します。

同期していない場合、A 面の Armadillo Base OS を B 面にコピーします。

同期している場合はコピーしません。

swdesc_option version で指定するバージョンの書き方については「6.4.1. インストールバージョンを指定する」を参照してください。

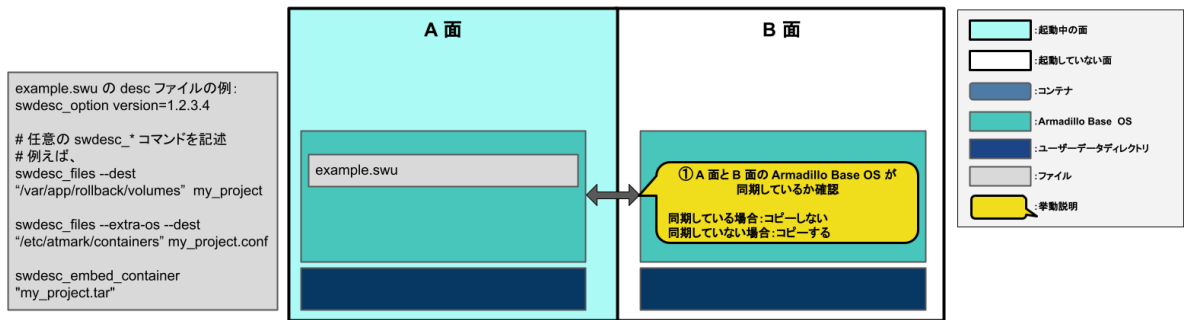


図 6.75 Armadillo Base OS を B 面にコピー

2. コマンドを順番に実行

「図 6.76. desc ファイルに記述した swudesc_* コマンドを実行」に示すように、desc ファイルに記述した順番に従って swudesc_* コマンドを実行します。

「6.4.1. インストールバージョンを指定する」に示すように、swdesc_* コマンドによって Armadillo Base OS に対して書き込みをする場合は --extra-os オプションをつけてください。

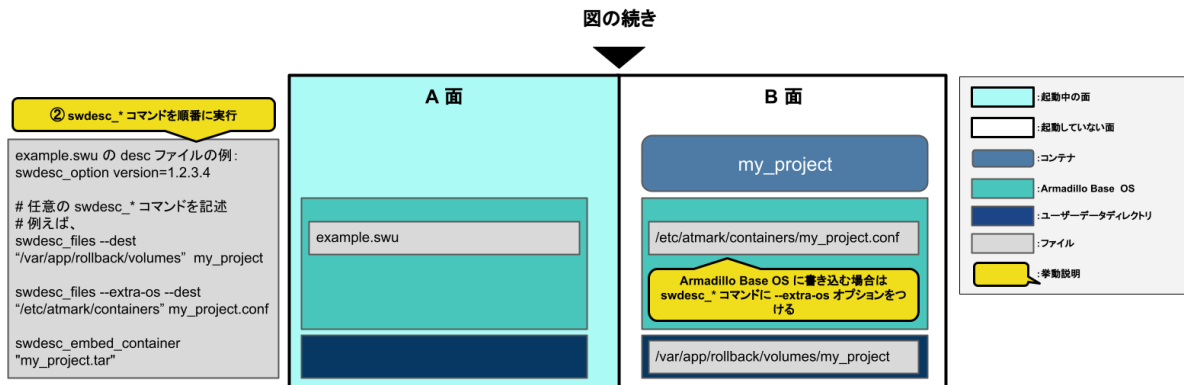


図 6.76 desc ファイルに記述した swudesc_* コマンドを実行

swudesc_* コマンドの種類を「表 6.4. swudesc_* コマンドの種類」に示します。

表 6.4 swudesc_* コマンドの種類

おおまかな機能	コマンド名	説明
ファイル転送 参照先: 「6.4.2. Armadillo ヘファイル を転送する」	swdesc_files	指定したファイルをアップデート先 の環境にコピー
	swdesc_tar	指定した tar アーカイブをアップデー ト先の環境に展開してコピー
コマンド実行 参照先: 「6.4.3. Armadillo 上で任意 のコマンドを実行する」	swdesc_command	指定したコマンドをアップデート先 の環境で実行
	swdesc_script	指定したスクリプトをアップデー ト先の環境で実行
ファイル転送 + コマンド実行 参照先: 「6.4.4. Armadillo にファイ ルを転送し、そのファイルをコマン ド内で使用する」	swdesc_exec	指定したファイルをアップデート先 の環境にコピーした後、そのファ イル名を"\$1"としてコマンドを実行
起動中の面に対してコマンド実行 (非 推奨) 参照先: 「6.4.5. 動作中の環境でのコ マンドの実行」	swdesc_command_nochroot	指定したコマンドを起動中の環境で 実行
	swdesc_script_nochroot	指定したスクリプトを起動中の環境 で実行
起動中の面に対してファイル転送 + コマンド実行 (非推奨) 参照先: 「6.4.5. 動作中の環境でのコ マンドの実行」	swdesc_exec_nochroot	指定したファイルを起動中の環境に コピーした後、そのファイル名を "\$1"としてコマンドを実行
コンテナイメージの転送 参照先: 「6.4.6. Armadillo にコンテ ナイメージを転送する」	swdesc_embed_container	SWU ファイルに含まれるコンテナ イメージの tar アーカイブをアップ デート先の環境に展開
	swdesc_pull_container	アップデート先の環境でコンテナイ メージをダウンロード
	swdesc_usb_container	SWU ファイルに含めない形で用意 したコンテナイメージの tar アーカ イブをアップデート先の環境に展開

3. アップデート完了後の挙動

デフォルトではアップデート後に再起動 (POST_ACTION=reboot) が行われます。

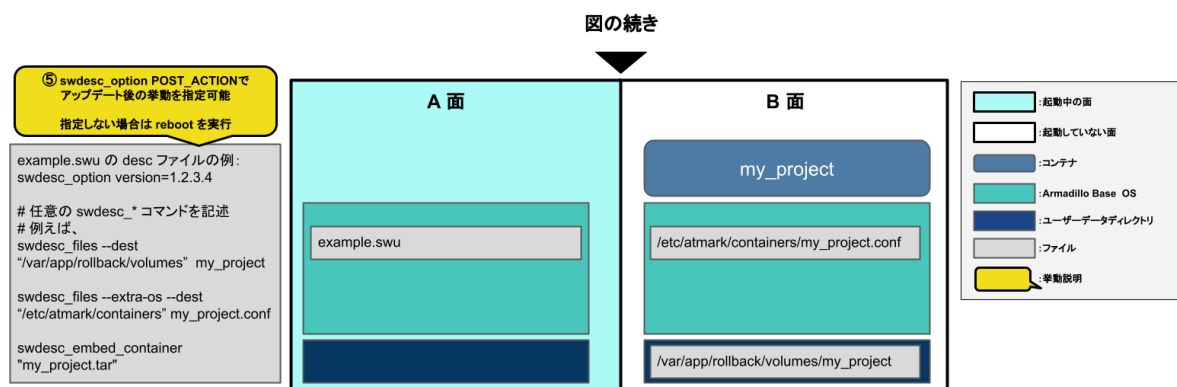


図 6.77 アップデート完了後の挙動

アップデート後の挙動を変更するには desc ファイルに swdesc_option POST_ACTION を追加してください。

swdesc_option POST_ACTION に指定できる挙動の種類を「表 6.5. アップデート完了後の挙動の種類」に示します。

表 6.5 アップデート完了後の挙動の種類

オプション名	説明
container	アップデート後にコンテナのみを再起動 (ただし、アップデート時に --extra_os オプションを指定したコマンドが実行された場合は reboot になる)
poweroff	アップデート後にシャットダウン
reboot	アップデートの後に再起動
wait	アップデート後に再起動は行われず、次回起動時に B 面に切り替わる

swdesc_option POST_ACTION の詳細は「6.4.10. SWUpdate 実行中/完了後の挙動を指定する」を参照してください。

4. B 面への切り替え

「図 6.78. B 面への切り替え」に示すように、正常にアップデートが行われると、次回起動時に B 面に切り替わります。

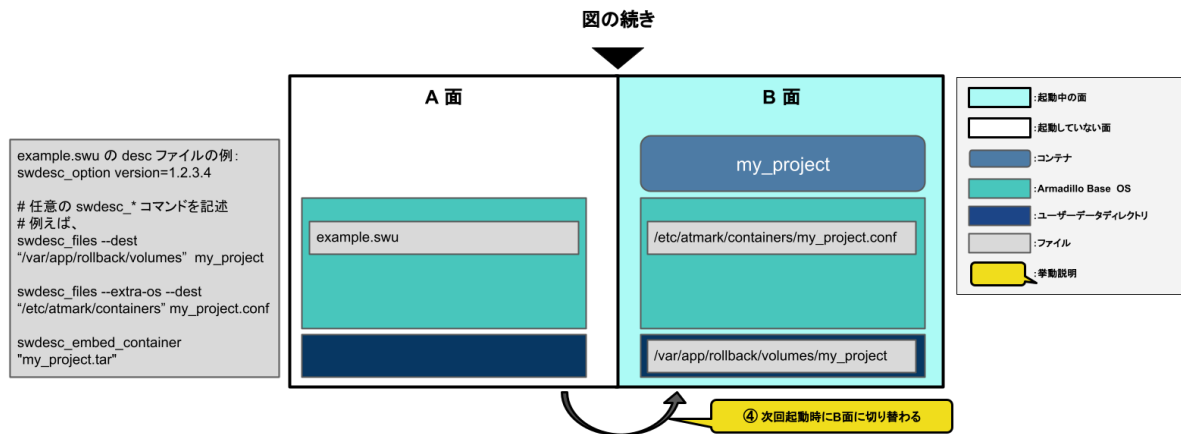


図 6.78 B 面への切り替え

5. desc ファイルの書き方の例

下記に SWUpdate を用いたアップデートの例を示します。

- ・ コンテナイメージとコンテナ自動設定ファイルのアップデート：「6.2.2.17. イメージを SWUpdate で転送する」
- ・ sshd の設定：「6.4.11.1. 例: sshd を有効にする」

6.3.3. Armadillo Base OS の一括アップデート

アップデートの流れを示します。

ここでは、boot して起動中の面を A 面、アップデート先の面を B 面とします。

1. Armadillo Base OS とアップデート後に保持するファイルを B 面へコピー

Armadillo Base OS とアップデート後に保持するファイルを B 面にコピーする流れを「図 6.79. Armadillo Base OS とファイルを B 面にコピー」に示します。

「6.4.1. インストールバージョンを指定する」に示すように、Armadillo Base OS の tar アーカイブを展開する swdesc_tar コマンドに --base-os オプションをつけてください。

swdesc_option version で指定するバージョンの書き方については「6.4.1. インストールバージョンを指定する」を参照してください。

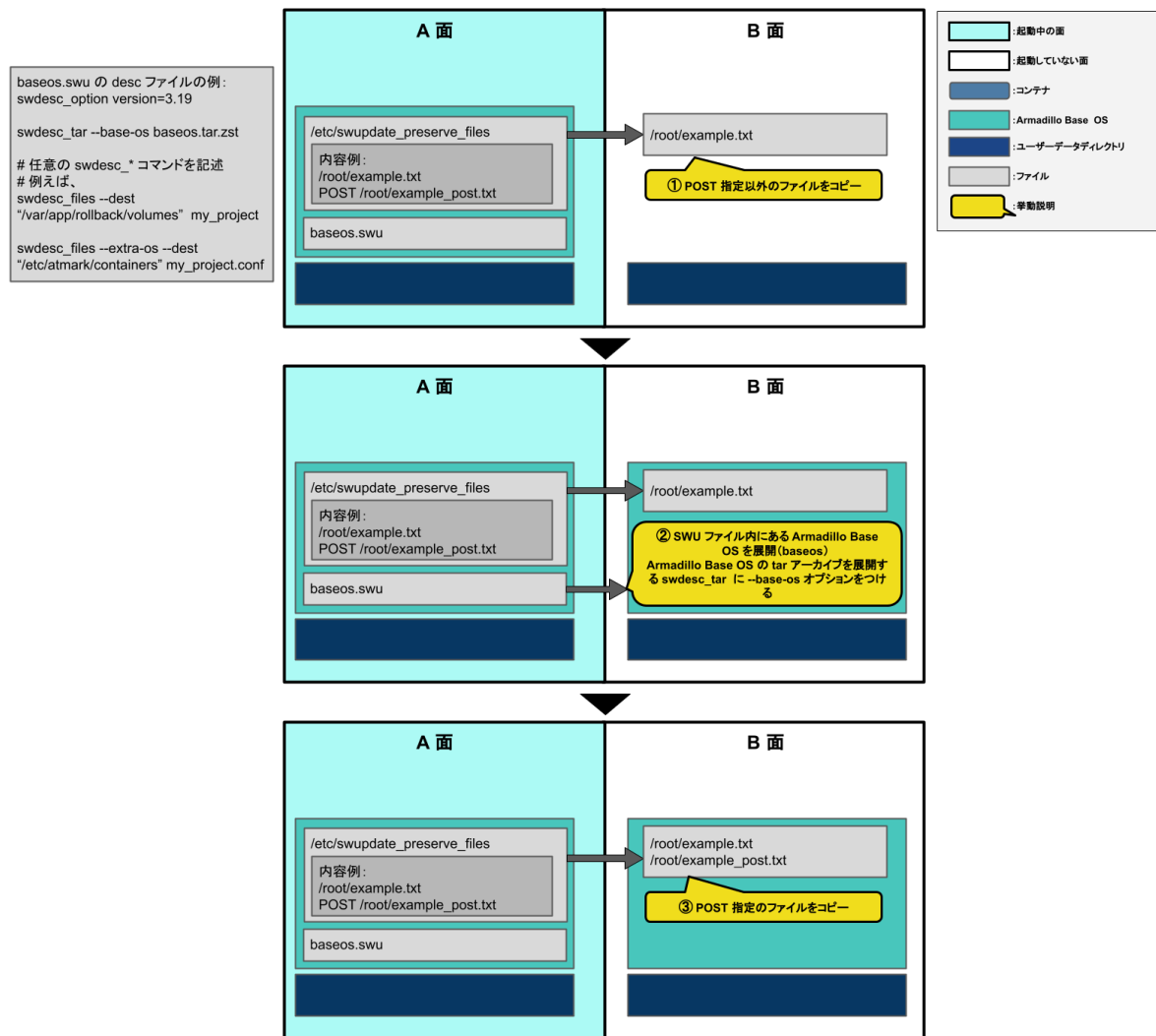


図 6.79 Armadillo Base OS とファイルを B 面にコピー

1. `/etc/swupdate_preserve` に記載された POST 指定以外のファイルを B 面にコピーします。
2. SWU ファイル内にある Armadillo Base OS を B 面に展開します。
3. `/etc/swupdate_preserve` に記載された POST 指定のファイルを B 面にコピーします。

`/etc/swupdate_preserve` への追記方法については「6.5. `swupdate_preserve_files` について」と「4.4.1. `/etc/swupdate_preserve_file` への追記」を参照してください。

2. コマンドを順番に実行

「図 6.80. desc ファイルに記述した `swudesc_*` コマンドを実行」に示すように、desc ファイルに記述した順番に従って `swudesc_*` コマンドを実行します。

「6.4.1. インストールバージョンを指定する」に示すように、`swdesc_*` コマンドによって Armadillo Base OS に対して書き込みをする場合は `--extra-os` オプションをつけてください。

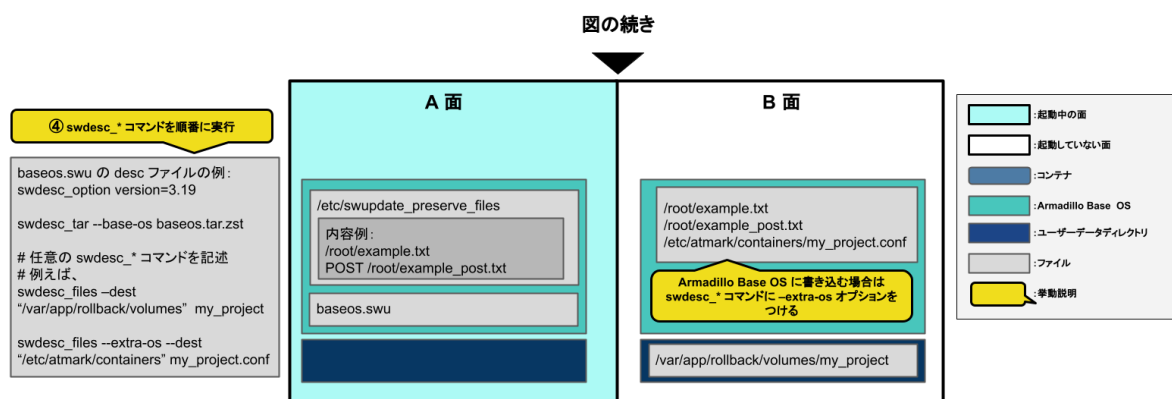


図 6.80 desc ファイルに記述した swudesc_* コマンドを実行

swudesc_* コマンドの種類を「表 6.6. swudesc_* コマンドの種類」に示します。

表 6.6 swudesc_* コマンドの種類

おおまかな機能	コマンド名	説明
ファイル転送 参照先：「6.4.2. Armadillo へファイル を転送する」	swdesc_files	指定したファイルをアップデート先 の環境にコピー
	swdesc_tar	指定した tar アーカイブをアップデー ト先の環境に展開してコピー
コマンド実行 参照先：「6.4.3. Armadillo 上で任意 のコマンドを実行する」	swdesc_command	指定したコマンドをアップデート先 の環境で実行
	swdesc_script	指定したスクリプトをアップデート 先の環境で実行
ファイル転送 + コマンド実行 参照先：「6.4.4. Armadillo にファイ ルを転送し、そのファイルをコマン ド内で使用する」	swdesc_exec	指定したファイルをアップデート先 の環境にコピーした後、そのファイ ル名を"\$1"としてコマンドを実行
起動中の面に対してコマンド実行（非 推奨） 参照先：「6.4.5. 動作中の環境でのコ マンドの実行」	swdesc_command_nochroot	指定したコマンドを起動中の環境で 実行
	swdesc_script_nochroot	指定したスクリプトを起動中の環境 で実行
起動中の面に対してファイル転送 + コマンド実行（非推奨） 参照先：「6.4.5. 動作中の環境でのコ マンドの実行」	swdesc_exec_nochroot	指定したファイルを起動中の環境に コピーした後、そのファイル名を "\$1"としてコマンドを実行
コンテナイメージの転送 参照先：「6.4.6. Armadillo にコンテ ナイメージを転送する」	swdesc_embed_container	SWU ファイルに含まれるコンテナ イメージの tar アーカイブをアップ デート先の環境に展開
	swdesc_pull_container	アップデート先の環境でコンテナイ メージをダウンロード
	swdesc_usb_container	SWU ファイルに含めない形で用意 したコンテナイメージの tar アーカ イブをアップデート先の環境に展開
ブートローダーの更新 参照先：「6.4.7. Armadillo のブート ローダーを更新する」	swdesc_boot	SWU ファイルに含まれるブートロー ダーをアップデート先の環境に展開

3. アップデート完了後の挙動

デフォルトではアップデート後に再起動（POST_ACTION=reboot）が行われます。

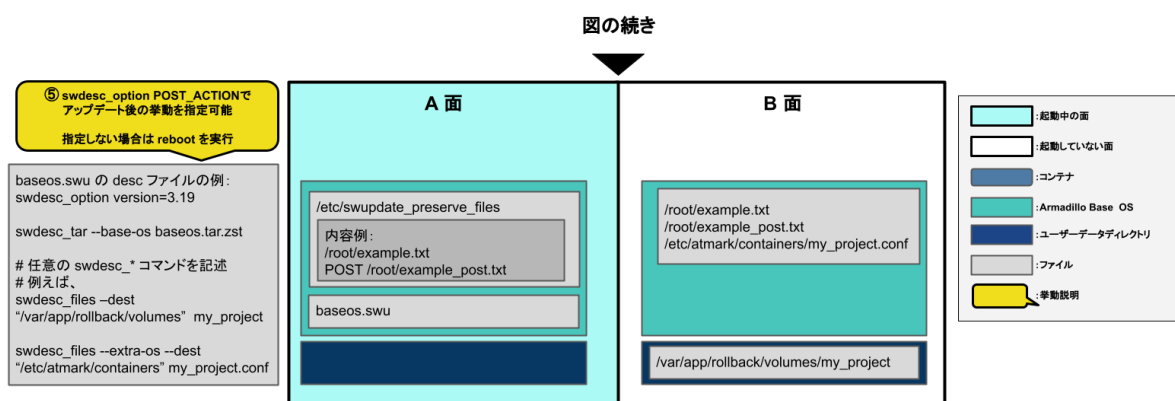


図 6.81 アップデート完了後の挙動

アップデート後の挙動を変更するには desc ファイルに swdesc_option POST_ACTION を追加してください。

swdesc_option POST_ACTION に指定できる挙動の種類を「表 6.7. アップデート完了後の挙動の種類」に示します。

表 6.7 アップデート完了後の挙動の種類

オプション名	説明
poweroff	アップデート後にシャットダウン
reboot	アップデートの後に再起動
wait	アップデート後に再起動は行われず、次回起動時に B 面に切り替わる

swdesc_option POST_ACTION の詳細は「6.4.10. SWUpdate 実行中/完了後の挙動を指定する」を参照してください。

4. B 面への切り替え

「図 6.82. B 面への切り替え (component=base_os)」に示すように、正常にアップデートが行われると、次回起動時に B 面に切り替わります。

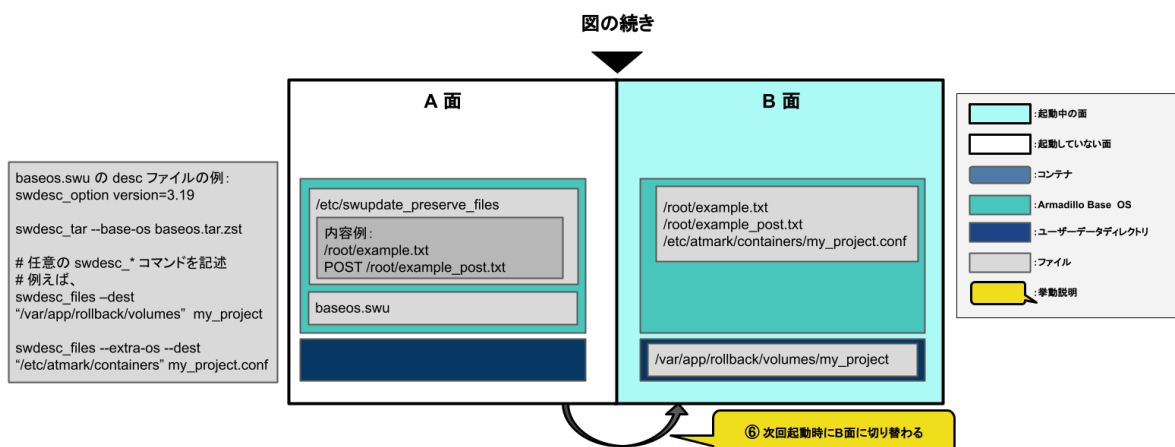


図 6.82 B 面への切り替え (component=base_os)

5. desc ファイルの書き方の例

下記に SWUpdate を用いたアップデートの例を示します。

- ・ Armadillo Base OS のアップデート：「6.4.11.2. 例: Armadillo Base OS アップデート」
- ・ Alpine Linux ルートファイルシステムのアップデート：「6.21.3. Alpine Linux ルートファイルシステムをビルドする」

6.3.4. ブートローダーのアップデート

swdesc_* コマンドには、swdesc_boot を指定してください。

swdesc_boot については「6.4.7. Armadillo のブートローダーを更新する」を参照してください。

ブートローダーのアップデートの流れは以下の通りです。

- ・ desc ファイルに swdesc_boot がある場合
SWU ファイルに含まれるブートローダーを B 面に書き込む
- ・ desc ファイルに swdesc_boot がない場合
A 面のブートローダーを B 面にコピーする

下記に SWUpdate を用いたアップデートの例を示します。

- ・ ブートローダーのアップデート：「6.21.1. ブートローダーをビルドする」

6.3.5. 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、「3.3.3.2. 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。

6.4. mkswu の .desc ファイルを編集する

mkswu で SWU イメージを生成するためには、desc ファイルを正しく作成する必要があります。以下では、desc ファイルの記法について紹介します。

6.4.1. インストールバージョンを指定する

```
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ません。

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。

`swdesc_tar` コマンドで rootfs (Armadillo Base OS) の tar アーカイブを展開する時に、`--base-os` オプションをつけることで component に `base_os` を指定したときと同じ動作となります。

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。

- ・ バージョンの指定方法

`swdesc_option version` で指定可能なバージョンのフォーマットは以下の 2 種類があります。

- ・ `x[y[z[-t]]]`

x, y, z にはそれぞれ 0 ~ 2147483647 の整数を適用してください。t には任意のアルファベットまたは 0 ~ 147483647 の整数を適用してください。

成功例は以下です：

- ・ 1
- ・ 1.2.3
- ・ 1.2.3-4
- ・ 1.2.3-abc.4
- ・ 1.2.3-a.b.c.4

失敗例は以下です：

- ・ 2147483648

x には 0 ~ 2147483647 の整数を適用してください。

- ・ 1.2.3-a.2147483648

t には 0 ~ 2147483647 の整数を適用してください。

- ・ 1.2.3-abc123

t には 数字とアルファベットを混在しないでください。1.2.3-abc.123 ならば可能です。

- ・ a.2.3

x にはアルファベットではなく 0 ~ 2147483647 の整数を適用してください。

- ・ 1.1.1.1-a

x[y[z[-t]]]の形式で書いてください。

- ・ x.y.z.t

x, y, z, t にはそれぞれ 0 ~ 65535 の整数を適用してください。

成功例は以下です：

- ・ 1.2.3.4
- ・ 65535.65535.65535.65535
- ・ 65535.2.3.4

失敗例は以下です：

- ・ 65536.2.3.4

x には 0 ~ 65535 の整数を適用してください。

- ・ 1.2.3.a

t にはアルファベットではなく 0 ~ 65535 の整数を適用してください。

- ・ 1.2.3.4.5

x.y.z.t の形式で書いてください。

6.4.2. Armadillo へファイルを転送する

- ・ `swdesc_tar` と `swdesc_files` でファイルを転送します。

```
swdesc_tar [--dest <dest>] [--preserve-attributes] <tar_file>
swdesc_files [--dest <dest>] [--basedir <basedir>] [--preserve-attributes] ¥
<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。

`--preserve-attributes` を指定しない場合はファイルのオーナー、モード、タイムスタンプ等が保存されませんので、必要な場合は設定してください。バージョンが `base_os` の場合は自動的に設定されます。

`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。

6.4.3. Armadillo 上で任意のコマンドを実行する

- ・ `swdesc_command` や `swdesc_script` でコマンドを実行します。

```
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実行します。

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` 以外に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。

6.4.4. Armadillo にファイルを転送し、そのファイルをコマンド内で使用する

- ・ `swdesc_exec` でファイルを配り、コマンド内でそのファイルを使用します。

```
swdesc_exec <file> <command>
```

`swdesc_command` と同じくコマンドを実行しますが、`<file>` を先に転送してコマンド内で転送したファイル名を"\$1"として使えます。

6.4.5. 動作中の環境でのコマンドの実行



本節で紹介する `swdesc_command_nochroot`、`swdesc_script_nochroot`、`swdesc_exec_nochroot` は基本的に使用することはありません。

`swdesc_command`、`swdesc_script`、`swdesc_exec` をご使用ください。

- ・ `swdesc_command_nochroot`、`swdesc_script_nochroot`、`swdesc_exec_nochroot` はアップデート先の環境ではなく動作中の環境でコマンドを実行します。

使い方は「6.4.2. Armadillo へファイルを転送する」と「6.4.4. Armadillo にファイルを転送し、そのファイルをコマンド内で使用する」に示した `nochroot` なしのバージョンと同じです。

アップデート先の環境は `/target` にマウントされるので、`nochroot` のコマンドを用いてアップデート先の環境に対してアクセスすることは可能です。

しかし、その方法によるアップデート先の環境に対するコマンドの実行は `nochroot` なしのコマンドでは実現できない特殊な場合にのみ行ってください。



`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。

6.4.6. Armadillo にコンテナイメージを転送する

- `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「6.2.2.15. リモートリポジトリにコンテナを送信する」、「6.2.2.17. イメージを SWUpdate で転送する」を参考にしてください。

6.4.7. Armadillo のブートローダーを更新する

- `swdesc_boot` でブートローダーを更新します。

```
swdesc_boot <boot image>
```

このコマンドだけはバージョンは自動的に設定されます。

6.4.8. SWU イメージの設定関連

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

6.4.9. Armadillo 上のコンテナイメージと自動起動用 conf ファイルを削除する

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、誤った使い方をした場合 `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`: インストールされているコンテナと `/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。

6.4.10. SWUpdate 実行中/完了後の挙動を指定する

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。

6.4.11. desc ファイル設定例

6.4.11.1. 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。

6.4.11.2. 例: Armadillo Base OS アップデート

ここでは、「6.21. 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 imx-boot script
swdesc_uboot imx-boot_armadillo_x2 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-x2-[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 imx-boot 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。

- ③ 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。
- ④ 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。

6.4.11.3. 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 ①
[armadillo ~]# persist_file /etc/swupdate_preserve_files ②
```

- ① swupdate_preserve_files に /boot と /lib/modules を保存するように追加します。
- ② 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" -- ¥ ①
"POST /boot" ¥
"POST /lib/modules"
```

- ① スクリプトの内容確認する場合は /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"
```

6.5. 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。「6.4.11.3. 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`

6.6. 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
```

6.7. 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。

6.8. SWUpdate の署名鍵と証明書の更新

mkswu で SWU イメージを生成する際に SWU イメージ内の sw-description という命令ファイルを ATDE にある署名鍵と証明書を用いて署名します。swupdate を実行する際には、署名に使用した証明書が /etc/swupdate.pem に含まれているかを確認します。

その署名を確認できなかった場合、SWU イメージをインストールできないので、Armadillo にある証明書を管理しなければなりません。

また、暗号鍵管理のガイドラインとしては定期的に鍵を交換することが強く推奨されています。mkswu --init を実行した際に 1 つだけ署名鍵と証明書を生成しましたが、ここでは他の署名鍵および証明書の生成と Armadillo 側の管理の方法について説明します。

6.8.1. 署名鍵と証明書の追加

署名鍵と証明書の生成は以下の様に、mkswu --genkey で行います。

```
[ATDE ~]$ mkswu --genkey
/home/atmark/mkswu/swupdate.key はすでに存在します。新しい鍵を作成しますか? [Y/n] ❶
証明書の共通名(一般名)を入力してください: [COMMON_NAME] ❷
署名鍵 /home/atmark/mkswu/swupdate-2.key と証明書 swupdate-2.pem を作成します。
Generating an EC private key
writing new private key to '/home/atmark/mkswu/swupdate-2.key.tmp'
Enter PEM pass phrase: ❸
Verifying - Enter PEM pass phrase:
-----
/home/atmark/mkswu/swupdate-2.pem をコンフィグファイルに追加します。
/home/atmark/mkswu/swupdate-2.pem が次のアップデートをインストールするときに転送されます。
インストールされてから現在の鍵を /home/atmark/mkswu/mkswu.conf から外してください。
```

図 6.83 mkswu --genkey で署名鍵と証明書を追加する

- ❶ Enter キーを押下します。
- ❷ [COMMON_NAME] には会社や製品が分かる任意の名称を入力してください。任意ではありますが、一つ目の鍵と違う名前にすることを推奨します。
- ❸ 署名鍵を保護するパスワードを 2 回入力します。

このコマンドを実行すると以下の文字列が ~/mkswu/mkswu.conf に追加されます。

```
# extra swupdate certificate. Remove the old one and use new
# PRIVKEY after having installed an update with this first
PUBKEY="$PUBKEY, $CONFIG_DIR/swupdate-2.pem" ❶
# remove "NEW_" to use
NEW_PRIVKEY="$CONFIG_DIR/swupdate-2.key" ❷
# This controls if we should update certificates on device, and can be
# removed once all devices have been updated to only allow new certificate
UPDATE_CERTS=yes
```

図 6.84 mkswu --genkey により mkswu.conf に追加された内容

- ❶ PUBKEY の値に生成した証明書のパスが追加されます。UPDATE_CERTS=yes を設定して生成した SWU イメージは、PUBKEY の値にリストされている証明書を全て Armadillo にインストールします。PUBKEY にリストされていない証明書、またはアットマークテクノ側で管理している証明書以外は全て削除されます。
- ❷ 新しい署名鍵のパスです。この段階では未使用になります。

この状態で任意 (POST_ACTION=container 以外) の SWU イメージを生成し、インストールすると Armadillo の /etc/swupdate.pem に PUBKEY にリストされている両方の証明書がインストールされます。Armadillo にインストールされている鍵は、abos-ctrl certificates list で確認できます：

```
[armadillo ~]# abos-ctrl certificates list
- atmark-2
- atmark-3
- swupdate-2.pem: [mkswu --genkey で入力した COMMON NAME] ❶
- swupdate.pem: [mkswu --init で入力した COMMON NAME] ❷
```

図 6.85 新しい証明書が Armadillo に追加されていることを確認する

- ❶ 追加した証明書のコモンネーム
- ❷ mkswu --init 時に生成した証明証のコモンネーム。当時の mkswu のバージョンによっては swupdate.pem が記載されていない可能性があります。

この状態で新しい鍵を使えるようになります。

6.8.2. 署名鍵と証明書の削除

上記のように Armadillo に複数の署名鍵を使用できる状態になった場合は証明に使う署名鍵と証明書を切り替えることができます。

mkswu.conf の PUBKEY の値から一つ目の値を削除し、PRIVKEY に新しい値を設定し、必要であれば UPDATE_CERTS=yes を記述します。

```
[ATDE ~] tail -n 3 ~/mkswu/mkswu.conf
PUBKEY="$CONFIG_DIR/swupdate-2.pem"
PRIVKEY="$CONFIG_DIR/swupdate-2.key"
UPDATE_CERTS=yes
```

図 6.86 署名鍵と証明書を削除する設定

署名鍵の追加と同じく、この状態で SWU イメージを生成し Armadillo にインストールすると前の証明書が削除されます。

こちらも abos-ctrl certificates list コマンドで確認可能です。

```
[armadillo ~]# abos-ctrl certificates list
- atmark-2
```

```
- atmark-3
- swupdate-2.pem: [mkswu --genkey で入力した COMMON NAME]
```

図 6.87 証明書がインストールされていることを確認する

6.9. Web UI から Armadillo をセットアップする (ABOS Web)

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。

詳細は、「3.8.1. ABOS Web とは」を参照してください。

6.9.1. 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にしています。そのため、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、「6.9.2. ABOS Web の設定機能一覧と設定手順」を参照してください。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。

6.9.2. ABOS Web の設定機能一覧と設定手順

現在、ネットワークに関して ABOS Web で設定できるのは以下のものです。

- ・ 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

これらについては、「3.8. ネットワーク設定」で紹介していますので、そちらを参照してください。

ネットワーク以外にも ABOS Web は以下の機能を持っています。

- ・ コンテナ管理
- ・ SWU インストール
- ・ 時刻設定
- ・ アプリケーション向けのインターフェース (Rest API)
- ・ カスタマイズ

本章では、これらのネットワーク以外の設定項目について紹介します。

6.9.3. コンテナ管理

ABOS Web から 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。

ABOS Web のトップページから、"コンテナ管理"をクリックすると、「図 6.88. コンテナ管理」の画面に遷移します。



図 6.88 コンテナ管理

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。



「3.8.12. 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。

6.9.4. SWU インストール

ABOS Web から 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ことができます。

SWU イメージについては、「3.3.3.2. SWU イメージとは」を参照してください。

ABOS Web のトップページから、"SWU インストール"をクリックすると、「図 6.89. SWU インストール」の画面に遷移します。



図 6.89 SWU インストール

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。

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 6.90 SWU 管理対象ソフトウェアコンポーネントの一覧表示

6.9.5. 時刻設定

ABOS Web から時刻に関する設定を行うことができます。

ABOS Web のトップページから "時刻設定" をクリックすると、以下の内容が表示されます。

「図 6.91. ネットワークタイムサーバーと同期されている場合の状況確認画面」では Armadillo の現在時刻と、同期中のサーバーとの時間差を確認することができます。

現在の状況

現在時刻: 2024-03-19 12:53:27 +09:00

NTP (ネットワークサーバー)と同期中

サーバー IP	133.243.238.163
サーバー時刻より	0.000011308 秒遅い

図 6.91 ネットワークタイムサーバーと同期されている場合の状況確認画面

時刻が同期されていない状態では「図 6.92. ネットワークタイムサーバーと同期されていない場合の状況確認画面」の様に「PC と同期する」ボタンを押すことで、Armadillo の時刻を PC と同期することができます。



図 6.92 ネットワークタイムサーバーと同期されていない場合の状況確認画面

「図 6.93. ネットワークタイムサーバーの設定項目」では NTP (ネットワークからの時刻同期) サーバーと Armadillo 起動時に同期するサーバーを設定することができます。



図 6.93 ネットワークタイムサーバーの設定項目

最後に、「図 6.94. タイムゾーンの設定項目」では Armadillo Base OS で使用するタイムゾーンの変更ができます。コンテナには影響ありませんのでご注意ください。



図 6.94 タイムゾーンの設定項目

6.9.6. アプリケーション向けのインターフェース (Rest API)

コンテナやスクリプトから ABOS Web の一部の機能を使用できます。

6.9.6.1. Rest API へのアクセス権の管理

Rest API は ABOS Web のパスワードと Rest API 用のトークンで認証されます。

また、接続可能なネットワークにも制限をかけております。初期状態では、同一サブネットからのアクセスのみ許容しています。同一サブネット外の IP アドレスからアクセスしたい場合は設定が必要です。設定方法は「3.8.2. ABOS Web へのアクセス」を参照してください。

各リクエストは以下のどちらかの Authorization ヘッダーで認証されます：

- ・ Basic (パスワード認証) : curl の `-u <password>` 等で認証可能です。<password> の文字列は ABOS Web で設定したパスワードです。
- ・ Bearer (トークン認証) : curl の `-H "Authorization: Bearer <token>"` 等で認証可能です。<token> は `/api/tokens` であらかじめ生成した文字列です。

また、トークンには権限も設定できます。Admin で生成されたトークンはすべてのインターフェースにアクセスできますが、一部のインターフェースしか使用しない場合はそのインターフェースに必要な権限だけを持つトークンを生成してください。

トークンの管理は ABOS Web の「設定管理」ページで行えます：

Rest API トークン一覧

Token ID	権限
<input checked="" type="radio"/> 35ac39a8-1eeb-4bb2-84d2-cb542cdbc873	Admin
<input type="radio"/> 5c426ce5-8fcb-4e54-9ff6-80aba50935ee	Reboot, NetworkView

トークンを追加権限を編集トークンを削除

図 6.95 設定管理の Rest API トークン一覧表示



ABOS Web のバージョン 1.2.3 以降では、Token ID の横にあるクリップボードアイコンをクリックするとクリップボードにコピーすることができます。

6.9.6.2. Rest API 使用例の前提条件

各 Rest API の使用例を説明します。使用例では以下を前提としています。:

- ・ ABOS Web に `https://armadillo.local:58080` でアクセスします。
- ・ 「AUTH」環境変数に ABOS Web で生成したトークンを設定します。例: `AUTH="Authorization: Bearer 35ac39a8-1eeb-4bb2-84d2-cb542cdbc873"`
- ・ curl コマンドを省略するため、以下のように alias を使用します:

```
[ATDE ~]$ alias curl_rest='curl -k -H "$AUTH" -w "%nhttp code: %{http_code}%n" '
```



コンテナから ABOS Web には「`https://host.containers.internal:58080`」でアクセスできます。



この章で説明する例では、curl のオプションに `-k` を指定して証明書を無視するようにしています。もし、証明書を使用したい場合は以下のように設定してください。

```
[ATDE ~]$ openssl s_client -showcerts -connect armadillo.local:58080 </
dev/null 2>/dev/null | openssl x509 -outform PEM > abosweb.pem
[ATDE ~]$ CERT="$PWD/abosweb.pem"
[ATDE ~]$ alias curl_rest='curl -H "$AUTH" --cacert "$CERT" -w "%nhttp
code: %{http_code}%n" '
```



6.9.6.3. Rest API の入力と出力

インターフェースの一部にはパラメータを取るものがあります。パラメータがある場合は json (Content-Type を `application/json` に設定する) と form (デフォルトの `application/x-www-form-urlencoded` のパラメータ) のどちらでも使用可能です。

インターフェースの出力がある場合は json object で出力されます。今後のバージョンアップで json object のキーが増える可能性があるため、出力された値を処理する場合はその点に留意してください。

エラーの場合は json object の「error」キーに文字列のエラーが記載されています。http のステータスコードも 50x になります。

エラーの例:

```
[ATDE ~]$ curl_rest -X DELETE https://armadillo.local:58080/api/tokens/
3b2d830d-2f64-4e76-9e59-316da82eefc4
{"error": "No such token"}
http code: 500
```



6.9.6.4. Rest API : トークン管理

トークン管理のためのインターフェースは以下のとおりです :

- ・ トークン一覧

GET `"/api/tokens"`

必要権限: Admin

パラメータ: 無し

出力: トークンリスト

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/tokens
{"tokens":[{"token":"35ac39a8-1eeb-4bb2-84d2-cb542cdbc873","permissions":["Admin"]},
{"token":"5c426ce5-8fcb-4e54-9ff6-80aba50935ee","permissions":["Reboot","NetworkView"]}]
http code: 200
```

↵

- ・ トークン取得

GET `"/api/tokens/<token>"`

必要権限: Admin

パラメータ: 無し

出力: トークン情報

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/tokens/35ac39a8-1eeb-4bb2-84d2-
cb542cdbc873
{"token":"35ac39a8-1eeb-4bb2-84d2-cb542cdbc873","permissions":["Admin"]}
http code: 200
```

↵

- ・ トークン生成

POST `"/api/tokens"`

必要権限: Admin

パラメータ: 付与したい permissions 権限リスト(ない場合は「Admin」で生成されます)

出力: 生成されたトークン情報

```
[ATDE ~]$ curl_rest -H "Content-type: application/json" -d '{"permissions":["SwuInstall",
"ContainerView"]}' https://armadillo.local:58080/api/tokens
{"token":"3b2d830d-2f64-4e76-9e59-316da82eefc4","permissions":
["SwuInstall","ContainerView"]}
http code: 200
```

↵

↵

- ・ トークン編集 (存在しない場合は指定のトークンで生成されます)

POST `"/api/tokens/{token_id}"`

必要権限: Admin

パラメータ: 付与したい permissions 権限リスト(ない場合は編集しません)

出力: 編集か生成されたトークン情報

```
[ATDE ~]$ curl_rest -X POST -d permissions=Poweroff -d permissions=ContainerAdmin https://
armadillo.local:58080/api/tokens/3b2d830d-2f64-4e76-9e59-316da82eefc4
{"token":"3b2d830d-2f64-4e76-9e59-316da82eefc4","permissions":["Poweroff","ContainerAdmin"]}
http code: 200
```

↵

- ・ トークン削除

DELETE `"/api/tokens/{token_id}"`

必要権限: Admin
 パラメータ: 無し
 出力: 無し

```
[ATDE ~]$ curl_rest -X DELETE https://armadillo.local:58080/api/tokens/
3b2d830d-2f64-4e76-9e59-316da82eefc4
http code: 200
```

- ・ **abos-web パスワード変更**

POST `"/api/password"`

必要権限: Admin

パラメータ: `password` でハッシュ済みのパスワード文字列か `hashed=false` が設定されている場合は平文の文字列

出力: 無し

```
[ATDE ~]$ PWD_HASH=$(openssl passwd -6)
Password:
Verifying - Password:
[ATDE ~]$ echo $PWD_HASH
$6$LuXQduN7L3PwbMaZ$txrw8vLjQEVUreQnZhM0CYMQ5U5B9b58L0mpVRULDiVCh2046GKscq/
xsDPskjxg.x8ym0ri1/8NqFBu..IZE0
[ATDE ~]$ curl_rest --data-urlencode "password=$PWD_HASH" -X POST https://armadillo.local:
58080/api/password
http code: 200
```

6.9.6.5. Rest API : SWU

- ・ **インストール済み SWU のバージョン情報取得**

GET `"/api/swu/versions"`

必要権限: SwuView

パラメータ: 無し

出力: Swupdate の各バージョン情報

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/swu/versions
{"extra_os.custom":"54","extra_os.container":"1","custom":"54","extra_os.initial_setup":"4",
"boot":"2020.4-at19","base_os":"3.18.4-at.6","extra_os.sshd":"1"}
http code: 200
```

- ・ **アップデートステータス取得**

GET `"/api/swu/status"`

必要権限: SwuView

パラメータ: 無し

出力: `rollback_ok`: ロールバック状態 (`false` の場合は `rollback` されています)、`last_update_timestamp`: UTC の unix epoch (数字での日付)、`last_update_versions`: 最新のアップデートで更新されたバージョン情報 (コンポーネント → [更新前のバージョン, 更新後のバージョン])。更新前に存在しなかったコンポーネントの場合は `null` で記載されています)

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/swu/status
{"rollback_ok":true,"last_update_timestamp":1703208559,"last_update_versions":{"custom":
```

```
[null, "54"], "extra_os.custom": ["53", "54"]}]
http code: 200
```

・ SWU をファイルアップロードでインストール

POST `"/api/swu/install/upload"`

必要権限: Swulninstall

パラメータ: multipart/form-data で **swu** の転送

出力: swupdate プロセスの出力 (stdout または stderr)、またはアップデートプロセスの出力ステータス (exit_code または exit_signal)

```
[ATDE ~]$ curl_rest -F swu=@"$HOME/mkswu/file.swu" https://armadillo.local:58080/api/swu/
install/upload
{"stdout": "SWUpdate v2023.05_git20231025-r0\n"}
{"stdout": "\n"}
{"stdout": "Licensed under GPLv2. See source distribution for detailed copyright notices.\n"}
{"stdout": "\n"}
{"stdout": "[INFO ] : SWUPDATE running : [main] : Running on AGX4500 Revision at1\n"}
{"stdout": "[INFO ] : SWUPDATE started : Software Update started !\n"}
{"stdout": "[INFO ] : SWUPDATE running : [install_single_image] : Installing pre_script\n"}
{"stdout": "[INFO ] : SWUPDATE running : [read_lines_notify] : No base os update: copying
current os over\n"}
: (省略)
{"stdout": "[INFO ] : SWUPDATE running : [install_single_image] : Installing post_script\n"}
{"stdout": "[INFO ] : SWUPDATE running : [read_lines_notify] : Removing unused containers\n"}
{"stdout": "[INFO ] : SWUPDATE running : [read_lines_notify] : swupdate triggering reboot!\n"}
{"stderr": "Killed\n"}
{"exit_code": 0}

http code: 200
```

・ SWU を URL でインストール

POST `"/api/swu/install/url"`

必要権限: Swulninstall

パラメータ: url=<SWU をダウンロードできる URL>

出力: swupdate プロセスの出力 (stdout または stderr)、またはアップデートプロセスの出力ステータス (exit_code または exit_signal)

```
[ATDE ~]$ curl_rest -d url=https://url/to/file.swu https://armadillo.local:58080/api/swu/
install/url
{"stdout": "Downloading https://url/to/file.swu...\n"}
{"stdout": "SWUpdate v2023.05_git20231025-r0\n"}
{"stdout": "\n"}
{"stdout": "Licensed under GPLv2. See source distribution for detailed copyright notices.\n"}
{"stdout": "\n"}
{"stdout": "[INFO ] : SWUPDATE running : [main] : Running on AGX4500 Revision at1\n"}
{"stdout": "[INFO ] : SWUPDATE started : Software Update started !\n"}
{"stdout": "[INFO ] : SWUPDATE running : [install_single_image] : Installing pre_script\n"}
{"stdout": "[INFO ] : SWUPDATE running : [read_lines_notify] : No base os update: copying
current os over\n"}
: (省略)
{"stdout": "[INFO ] : SWUPDATE running : [install_single_image] : Installing post_script\n"}
{"stdout": "[INFO ] : SWUPDATE running : [read_lines_notify] : Removing unused containers\n"}
{"stdout": "[INFO ] : SWUPDATE running : [read_lines_notify] : swupdate triggering reboot!\n"}
{"stderr": "Killed\n"}

http code: 200
```



```

{"exit_code":0}

http code: 200

```

6.9.6.6. Rest API : コンテナ操作

- ・ **コンテナ一覧**

GET `"/api/containers"`

必要権限: ContainerView

パラメータ: 無し

出力: 各コンテナの id, name, state, command, image 情報

```

[ATDE ~]$ curl_rest https://armadillo.local:58080/api/containers
{"containers":
[{"id":"02616122dcea5bd75c551b29b2ef54f54e09f59c50ce3282684773bc6bfb86a8", "name":"python_app",
"state":"running", "command":["python3", "/vol_app/src/main.py"], "image":"localhost/python_arm64_app_image:latest"}]}

http code: 200

```



- ・ **コンテナログ取得**

GET `"/api/containers/{container}/logs"`

必要権限: ContainerView

パラメータ: **follow=true** (podman logs -f と同様の効果)

出力: podman logs プロセスの出力 (stdout または stderr)、またはアップデートプロセスの出力ステータス (exit_code または exit_signal)

```

[ATDE ~]$ curl_rest https://armadillo.local:58080/api/containers/python_app/logs
{"stdout":"Some message\n"}
{"exit_code":0}

http code: 200

```

follow=true を付与する例

```

[ATDE ~]$ curl_rest https://armadillo.local:58080/api/containers/python_app/logs?follow=true
{"stdout":"Some message\n"}
Ctrl-C で終了

```

- ・ **コンテナ起動**

POST `"/api/containers/{container}/start"`

必要権限: ContainerAdmin

パラメータ: 無し

出力: 無し

```

[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/containers/python_app/start

http code: 200

```

- ・ **コンテナ停止**

POST `"/api/containers/{container}/stop"`

必要権限: ContainerAdmin

パラメータ: 無し

出力: 無し

```
[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/containers/python_app/stop
http code: 200
```

6.9.6.7. Rest API : ネットワーク設定

・ ネットワーク設定一覧

GET `"/api/connections"`

必要権限: NetworkView

パラメータ: 無し

出力: ネットワーク設定一覧と各接続の uuid, name, state, ctype, 存在すれば device 情報

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/connections
{"connections":[{"name":"Wired connection
1","state":"activated","uuid":"18d241f1-946c-3325-974f-65cda3e6eea5","ctype":"802-3-
ethernet","device":"eth0"}, {"name":"lo","state":"activated","uuid":"529ec241-f122-4cb2-843f-
ec9787b2aee7","ctype":"loopback","device":"lo"},
{"name":"podman0","state":"activated","uuid":"be4583bc-3498-4df2-
a31c-773d781433aa","ctype":"bridge","device":"podman0"},
{"name":"veth0","state":"activated","uuid":"03446b77-b1ab-47d0-98fc-
f167c3f3778a","ctype":"802-3-ethernet","device":"veth0"}, {"name":"Wired connection
2","state":"","uuid":"181f44df-850e-36c1-a5a4-6e461c768acb","ctype":"802-3-ethernet"},
{"name":"Wired connection 3","state":"","uuid":"e4381368-6351-3985-
ba6e-2625c62b8d39","ctype":"802-3-ethernet"}]}

http code: 200
```

・ ネットワーク設定詳細取得

GET `"/api/connections/{connection}"`

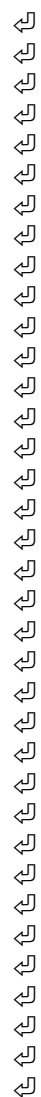
必要権限: NetworkView

パラメータ: 無し (URL の connection は UUID または接続名で使用可能)

出力: 接続の詳細情報 (Network Manager のプロパティ)

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/connections/Wired%20connection%201
{"name":"Wired connection
1","state":"activated","uuid":"18d241f1-946c-3325-974f-65cda3e6eea5","ctype":"802-3-
ethernet","device":"eth0","props":{"802-3-ethernet.accept-all-mac-addresses":"-1","802-3-
ethernet.auto-negotiate":"no","802-3-ethernet.cloned-mac-address":"","802-3-
ethernet.duplex":"","802-3-ethernet.generate-mac-address-mask":"","802-3-ethernet.mac-
address":"","802-3-ethernet.mac-address-blacklist":"","802-3-ethernet.mtu":"auto","802-3-
ethernet.port":"","802-3-ethernet.s390-nettype":"","802-3-ethernet.s390-options":"","802-3-
ethernet.s390-subchannels":"","802-3-ethernet.speed":"0","802-3-ethernet.wake-on-
lan":"default","802-3-ethernet.wake-on-lan-password":"","GENERAL.CON-PATH":"/org/
freedesktop/NetworkManager/Settings/1","GENERAL.DBUS-PATH":"/org/freedesktop/NetworkManager/
ActiveConnection/
6","GENERAL.DEFAULT":"yes","GENERAL.DEFAULT6":"no","GENERAL.DEVICES":"eth0","GENERAL.IP-
IFACE":"eth0","GENERAL.MASTER-PATH":"","GENERAL.NAME":"Wired connection 1","GENERAL.SPEC-
OBJECT":"","GENERAL.STATE":"activated","GENERAL.UUID":"18d241f1-946c-3325-974f-65cda3e6eea5"}
```

```
, "GENERAL.VPN": "no", "GENERAL.ZONE": "", "IP4.ADDRESS[1]": "198.51.100.123/16", "IP4.DNS[1]": "192.0.2.1", "IP4.DNS[2]": "192.0.2.2", "IP4.GATEWAY": "198.51.100.1", "IP4.ROUTE[1]": "dst = 198.51.100.0/16, nh = 0.0.0.0, mt = 100", "IP4.ROUTE[2]": "dst = 0.0.0.0/0, nh = 198.51.100.1, mt = 100", "IP6.ADDRESS[1]": "fe80::211:cff:fe00:b13/64", "IP6.GATEWAY": "", "IP6.ROUTE[1]": "dst = fe80::/64, nh = ::, mt = 1024", "connection.auth-retries": "-1", "connection.autoconnect": "yes", "connection.autoconnect-priority": "-999", "connection.autoconnect-retries": "-1", "connection.autoconnect-slaves": "-1", "connection.dns-over-tls": "-1", "connection.gateway-ping-timeout": "0", "connection.id": "Wired connection 1", "connection.interface-name": "eth0", "connection.lldp": "default", "connection.llmnr": "-1", "connection.master": "", "connection.mdns": "-1", "connection.metered": "unknown", "connection.mptcp-flags": "0x0", "connection.multi-connect": "0", "connection.permissions": "", "connection.read-only": "no", "connection.secondaries": "", "connection.slave-type": "", "connection.stable-id": "", "connection.timestamp": "1703208824", "connection.type": "802-3-ethernet", "connection.uuid": "18d241f1-946c-3325-974f-65cda3e6eea5", "connection.wait-activation-delay": "-1", "connection.wait-device-timeout": "-1", "connection.zone": "", "ipv4.addresses": "198.51.100.123/16", "ipv4.auto-route-ext-gw": "-1", "ipv4.dad-timeout": "-1", "ipv4.dhcp-client-id": "", "ipv4.dhcp-fqdn": "", "ipv4.dhcp-hostname": "", "ipv4.dhcp-hostname-flags": "0x0", "ipv4.dhcp-iaid": "", "ipv4.dhcp-reject-servers": "", "ipv4.dhcp-send-hostname": "yes", "ipv4.dhcp-timeout": "0", "ipv4.dhcp-vendor-class-identifier": "", "ipv4.dns": "192.0.2.1, 192.0.2.2", "ipv4.dns-options": "", "ipv4.dns-priority": "0", "ipv4.dns-search": "", "ipv4.gateway": "198.51.100.1", "ipv4.ignore-auto-dns": "no", "ipv4.ignore-auto-routes": "no", "ipv4.link-local": "0", "ipv4.may-fail": "yes", "ipv4.method": "manual", "ipv4.never-default": "no", "ipv4.replace-local-rule": "-1", "ipv4.required-timeout": "-1", "ipv4.route-metric": "-1", "ipv4.route-table": "0", "ipv4.routes": "", "ipv4.routing-rules": "", "ipv6.addr-gen-mode": "eui64", "ipv6.addresses": "", "ipv6.auto-route-ext-gw": "-1", "ipv6.dhcp-duid": "", "ipv6.dhcp-hostname": "", "ipv6.dhcp-hostname-flags": "0x0", "ipv6.dhcp-iaid": "", "ipv6.dhcp-send-hostname": "yes", "ipv6.dhcp-timeout": "0", "ipv6.dns": "", "ipv6.dns-options": "", "ipv6.dns-priority": "0", "ipv6.dns-search": "", "ipv6.gateway": "", "ipv6.ignore-auto-dns": "no", "ipv6.ignore-auto-routes": "no", "ipv6.ip6-privacy": "-1", "ipv6.may-fail": "yes", "ipv6.method": "auto", "ipv6.mtu": "auto", "ipv6.never-default": "no", "ipv6.rtimeout": "0", "ipv6.replace-local-rule": "-1", "ipv6.required-timeout": "-1", "ipv6.route-metric": "-1", "ipv6.route-table": "0", "ipv6.routes": "", "ipv6.routing-rules": "", "ipv6.token": "", "proxy.browser-only": "no", "proxy.method": "none", "proxy.pac-script": "", "proxy.pac-url": ""}}
http code: 200
```



・ ネットワーク設定の変更

PATCH "/api/connections/{connection}"

必要権限: NetworkAdmin

パラメータ: Network Manager で編集可能な値

出力: 無し

```
[ATDE ~]$ curl_rest -X PATCH -d ipv4.method=manual -d ipv4.addresses=198.51.100.123/16 https://armadillo.local:58080/api/connections/Wired%20connection%201
http code: 200
```



・ ネットワークの接続

POST "/api/connections/{connection}/up"

必要権限: NetworkAdmin

パラメータ: 無し

出力: 無し

```
[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/connections/Wired%20connection%201/up
```

```
http code: 200
```

・ ネットワークの切断

```
POST "/api/connections/{connection}/down"
```

必要権限: NetworkAdmin

パラメータ: 無し

出力: 無し

```
[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/connections/Wired%20connection%201/down
```

```
http code: 200
```

・ ネットワーク設定の削除

```
DELETE "/api/connections/{connection}"
```

必要権限: NetworkAdmin

パラメータ: 無し出力: 無し

```
[ATDE ~]$ curl_rest -X DELETE https://armadillo.local:58080/api/connections/178b8c95-fcad-4bb1-8040-5a02b9ad046f
```

```
http code: 200
```



通信に使用しているネットワークの設定を削除した場合は Armadillo へアクセスできなくなりますので、ご注意ください。

6.9.6.8. Rest API : WLAN

・ 無線ネットワークのリスト取得

```
GET "/api/wlan/scan"
```

必要権限: NetworkView

パラメータ: (任意)rescan=true/false, false を指定するとキャッシュされているスキャン結果を出力します。

出力: リスト

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/wlan/scan
[{"id":"my_ap","signal":74,"bssid":"04:42:1A:E4:78:0C","chan":44,"rate":"540 Mbit/s","security":"WPA2 WPA3"}, {"id":"other_ap","signal":65,"bssid":"AC:44:F2:56:22:38","chan":1,"rate":"130 Mbit/s","security":"WPA2"}]
```

```
http code: 200
```

・ *無線ネットワークの接続

```
POST "/api/wlan/connect"
```

必要権限: NetworkAdmin

パラメータ: ssid, passphrase, ifname, bssid, hidden. ssid 以外は任意です。

出力: 生成した接続の uuid

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/wlan/connect -d ssid=my_ap -d
passphrase=my_passphrase
{"uuid":"178b8c95-fcad-4bb1-8040-5a02b9ad046f"}
http code: 200
```



・無線ネットワーク アクセスポイントの設定

POST `"/api/wlan/ap"`

必要権限: NetworkAdmin

パラメータ: ssid, passphrase, bridge_addr, hw_mode/channel, interface.

interface は任意です。hw_mode:2.4GHz を使用する場合は "g"、5GHz を使用する場合は "a" を設定します。

channel: 2.4GHz の場合は 1 ~ 13、5GHz の場合は 36、40、44、48 を設定します。

hw_mode/channel を設定しない場合は自動的に選択されますが、両方を未設定にすることはできません。

出力: 無し

```
[ATDE ~]$ curl_rest -d ssid=my_ap -d passphrase=my_passphrase -d bridge_addr=198.51.100.1/24
-d channel=3 https://armadillo.local:58080/api/wlan/ap
http code: 200
```



アクセスポイントを設定するとクライアントの接続が無効になります。



クライアントの接続の削除は `DELETE "/api/connections/{connection}"` で行えます。

・無線ネットワーク アクセスポイントの削除

DELETE `"/api/wlan/ap"`

必要権限: NetworkAdmin

パラメータ: interface (任意)

出力: 無し

```
[ATDE ~]$ curl_rest -X DELETE https://armadillo.local:58080/api/wlan/ap
http code: 200
```

6.9.6.9. Rest API : WWAN の設定

・ WWAN の設定追加

POST `"/api/wwan"`

必要権限: NetworkAdmin

パラメータ: apn, user, password, auth_type (CHAP/PAP, デフォルト CHAP), mccmnc, ipv6 (bool, デフォルト true)

apn 以外は任意です。

出力: 追加された接続の uuid

```
[ATDE ~]$ curl_rest -d apn=provider.tld -d user=provider -d password=provider https://armadillo.local:58080/api/wwan  
{  
  "uuid": "ce603d3e-838b-4ac8-b7fd-6a3f1abe4003"  
}
```

```
http code: 200
```



・ WWAN の設定削除

DELETE `"/api/wwan"`

必要権限: NetworkAdmin

パラメータ: 無し

出力: 無し

```
[ATDE ~]$ curl_rest -X DELETE https://armadillo.local:58080/api/wwan
```

```
http code: 200
```



WWAN の設定確認または一時的な切断は connection の API で行ってください。

・ IMEI の取得

GET `"/api/wwan/imei"`

必要権限: NetworkView

パラメータ: 無し

出力: LTE モジュールの IMEI

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/wwan/imei
```

```
{  
  "imei": "XXXXXXXXXXXXXXX"  
}
```

```
http code: 200
```

・ 電話番号の取得

GET `"/api/wwan/phone_numbers"`

必要権限: NetworkView

パラメータ: 無し

出力: SIM カードに設定されている電話番号

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/wwan/phone_numbers
{"phone_numbers":["XXXXXXXXXX"]}
http code: 200
```

・ 電波品質の取得

GET `"/api/wwan/signal_quality"`

必要権限: NetworkView

パラメータ: 無し

出力: 電波品質(mmcli コマンドで出力される signal quality 相当の値)

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/wwan/signal_quality
{"signal_quality": "54"}
http code: 200
```

・ SMS 一覧の取得

GET `"/api/wwan/sms"`

必要権限: SmsView

パラメータ: 無し

出力: SMS メッセージ ID 一覧

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/wwan/sms
{"message_ids":["0", "1"]}
http code: 200
```

・ SMS の取得

GET `"/api/wwan/sms/{message_id}"`

必要権限: SmsView

パラメータ: 無し

出力: SMS の内容

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/wwan/sms/0
{"sms":{"content":{"data":"--", "number":"XXXXXXXXXX", "text":"message"}, "dbus_path":"/org/freedesktop/ModemManager1/SMS/0", "properties":{"pdu_type":"deliver", "state":"received", "storage":"me", "timestamp":"20YY-MM-DDThh:mm:ss+ZZ"}}}
http code: 200
```



message_id は SMS 一覧の取得で表示された値を使用します。

・ SMS の作成・送信

POST `"/api/wwan/sms"`

必要権限: SmsSend

パラメータ: phone_number: 送信先電話番号, message: 送信するメッセージ

出力: 無し

```
[ATDE ~]$ curl_rest -d phone_number=XXXXXXXXXX -d message="message" https://armadillo.local:58080/api/wwan/sms  
  
http code: 200
```



・ SMS の削除

```
DELETE "/api/wwan/sms/{message_id}"  
必要権限: SmsView  
パラメータ: 無し  
出力: 無し
```

```
[ATDE ~]$ curl_rest -X DELETE https://armadillo.local:58080/api/sms/0  
  
http code: 200
```



message_id は SMS 一覧の取得で表示された値を使用します。

・ SMS の全削除

```
DELETE "/api/wwan/sms"  
必要権限: SmsView  
パラメータ: 無し  
出力: 無し
```

```
[ATDE ~]$ curl_rest -X DELETE https://armadillo.local:58080/api/sms  
  
http code: 200
```

6.9.6.10. Rest API : DHCP の設定

・ DHCP の設定確認

```
GET "/api/dhcp"  
必要権限: NetworkView  
パラメータ: 無し  
出力: interface, ip_addr, start_addr, end_addr, lease_time のリスト
```

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/dhcp  
[{"interface": "br_ap", "ip_addr": "198.51.100.1/24", "start_addr": "198.51.100.10", "end_addr": "198.51.100.20", "lease_time": "3600"}]  
http code: 200
```



・ DHCP の設定

```
POST "/api/dhcp/{interface}"  
必要権限: NetworkAdmin  
パラメータ: start_addr, end_addr, lease_time  
lease_time を設定しなかった場合は 3600 (秒) とする
```


出力: 無し

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/dhcp/br_ap -d start_addr=198.51.100.10  
-d end_addr=198.51.100.20  
  
http code: 200
```

↩

・ DHCP の設定削除

DELETE `"/api/dhcp/{interface}"`

必要権限: NetworkAdmin

パラメータ: 無し

出力: 無し

```
[ATDE ~]$ curl_rest -X DELETE https://armadillo.local:58080/api/dhcp/br_ap  
  
http code: 200
```

6.9.6.11. Rest API : NAT の設定

・ NAT (masquerade) の設定確認

GET `"/api/nat"`

必要権限: NetworkView

パラメータ: 無し

出力: NAT されている **interface** のリスト

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/nat  
[{"interface": "eth0"}]  
http code: 200
```

・ NAT の設定

POST `"/api/nat/{interface}"`

必要権限: NetworkAdmin

パラメータ: 無し

出力: 無し

```
[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/nat/eth0  
  
http code: 200
```

・ NAT の削除

DELETE `"/api/nat/{interface}"`

必要権限: NetworkAdmin

パラメータ: 無し

出力: 無し

```
[ATDE ~]$ curl_rest -X DELETE https://armadillo.local:58080/api/nat/eth0  
  
http code: 200
```

- ・ **ポートフォワードの設定確認**

GET `"/api/port_forwarding"`

必要権限: NetworkView

パラメータ: 無し

出力: フォワードされてるポート

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/port_forwarding
[{"interface":"eth0","protocol":"tcp","dport":"22","destination":"127.0.0.1","destination_port":"2222"}]
http code: 200
```

↵

- ・ **ポートフォワードの設定**

POST `"/api/port_forwarding"`

必要権限: NetworkAdmin

パラメータ: interface, protocol (デフォルト tcp), dport, destination, destination_port

出力: 無し

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/port_forwarding -d interface=eth0 -d
dport=22 -d destination=127.0.0.1 -d destination_port=2222
http code: 200
```

↵

- ・ **ポートフォワードの削除**

DELETE `"/api/port_forwarding"`

必要権限: NetworkAdmin

パラメータ: interface, protocol (デフォルト tcp), dport, destination, destination_port

出力: 無し

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/port_forwarding -X DELETE -H "Content-
Type: application/json" -d
'{"interface":"eth0","protocol":"tcp","dport":"22","destination":"127.0.0.1","destination_port":"2222"}'
http code: 200
```

↵
↵
↵

6.9.6.12. Rest API : VPN の設定

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。

- ・ **VPN の設定**

POST `"/api/vpn/openvpn"`

必要権限: VpnAdmin

パラメータ: setting_name, conf, auth_type, username, password, cert, key, key_pass

setting_name: 設定名です。任意の文字列を設定してください。

conf: OpenVPN の設定ファイルです。

auth_type: 認証方式です。userpass(ユーザ名とパスワード) または cert(証明書) を設定してください。

username: auth_type が userpass の場合、ユーザ名を設定します。

password: auth_type が userpass の場合、パスワードを設定します。

cert: auth_type が cert の場合、証明書ファイルを設定します。OpenVPN の設定ファイルに含まれている場合は不要です。
 key: auth_type が cert の場合、鍵ファイルを設定します。OpenVPN の設定ファイルに含まれている場合は不要です。
 key_pass: 鍵がパスワードで保護されている場合、そのパスワードを設定します。
 出力: 無し

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/vpn/openvpn -F setting_name=myvpn -F
conf=@"$HOME/conf.ovpn" -F auth_type=userpass -F username=myname -F password=myspass
http code: 200
```



図 6.96 ユーザ名とパスワード認証の例

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/vpn/openvpn -F setting_name=myvpn -F
conf=@"$HOME/conf.ovpn" -F auth_type=cert -F cert=@"$HOME/cert.crt" -F key=@"$HOME/key.key" -F
key_pass=mykeypass
http code: 200
```



図 6.97 証明書認証の例



コンテナ内から VPN 設定の Rest API を使う場合、Armadillo 上に alpine_openvpn コンテナイメージが存在していないと、正しく設定されません。このコンテナイメージが存在していない場合、先に ABOS Web の Web UI やコンテナ外からの Rest API で設定を行ってください。一度 alpine_openvpn コンテナイメージがインストールされれば、コンテナ内からも Rest API で設定できます。alpine_openvpn コンテナイメージは VPN 設定を削除しても残り続けますが、VPN 設定を削除した後に swupdate でアップデートを行うと削除されますので、その場合は再度 alpine_openvpn コンテナイメージのインストールを行う必要があります。

・ VPN の接続

POST `"/api/vpn/up"`
 必要権限: VpnAdmin
 パラメータ: 無し
 出力: 無し

```
[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/vpn/up
http code: 200
```

・ VPN の切断

POST `"/api/vpn/down"`
 必要権限: VpnAdmin
 パラメータ: 無し
 出力: 無し

```
[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/vpn/down
http code: 200
```

- ・ VPN 設定の削除

DELETE "/api/vpn/openvpn"
 必要権限: VpnAdmin
 パラメータ: 無し
 出力: 無し

```
[ATDE ~]$ curl_rest -X DELETE https://armadillo.local:58080/api/vpn/openvpn
http code: 200
```

6.9.6.13. Rest API : 時刻の設定

- ・ 時刻の状況確認

GET "/api/time/ntp_info"
 必要権限: TimeView
 パラメータ: 無し

出力: time_now: epoch 形式の現在時刻、ntp_server_ip: 現在同期中のサーバーアドレス。同期されていない場合は「null」となります。ntp_server_offset: 現在同期中のサーバーとの時刻の遅れ (マイナスの場合は Armadillo がサーバーより早いです)

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/time/ntp_info
{"ntp_server_ip":"203.0.113.10","ntp_server_offset":"-0.000015824","time_now":1710139558}
http code: 200
```

- ・ NTP の設定確認

GET "/api/time/ntp_config"
 必要権限: TimeView
 パラメータ: 無し

出力: servers: 同期する対象、initstepslew: Armadillo 起動時に同期するかどうかの設定

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/time/ntp_config
{"servers":["pool pool.ntp.org iburst"],"initstepslew":"10 pool.ntp.org"}
http code: 200
```

- ・ NTP の設定

POST "/api/time/ntp_config"
 必要権限: TimeAdmin

パラメータ: servers: 同期する対象、initstepslew: Armadillo 起動時に同期するかどうかの設定。パラメータを送信しない場合は設定されません。値が空の場合は設定が削除されて、「default」の場合は Armadillo Base OS のデフォルトに戻ります。
 出力: 取得時と同じ

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/time/ntp_config -X POST -d "servers=server 203.0.113.10 iburst" -d "servers=server 203.0.113.11 iburst" -d "initstepslew="
{"servers":["server 203.0.113.10 iburst","server 203.0.113.11 iburst"],"initstepslew":null}
http code: 200
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/time/ntp_config -X POST -d "servers=default&initstepslew=default"
{"servers":["pool pool.ntp.org iburst"],"initstepslew":"10 pool.ntp.org"}
http code: 200
```

- ・ **タイムゾーンの確認**

GET `"/api/time/timezone"`

必要権限: TimeView

パラメータ: 無し

出力: timezone: 使用されているタイムゾーン

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/time/timezone
{"timezone":"Asia/Tokyo"}
http code: 200
```

- ・ **タイムゾーンの設定**

POST `"/api/time/timezone"`

必要権限: TimeAdmin

パラメータ: timezone: 設定するタイムゾーン

出力: 無し

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/time/timezone -X POST -d "timezone=Asia/
Tokyo"
http code: 200
```

[↩](#)

- ・ **時刻を強制的に設定する**

POST `"/api/time/set"`

必要権限: TimeAdmin

パラメータ: timestamp: epoch 形式の時刻

出力: 無し

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/time/set -X POST -d "timestamp=$(date +%s)"
http code: 200
```

6.9.6.14. Rest API : 電源制御

- ・ **再起動**

POST `"/api/reboot"`

必要権限: Reboot

パラメータ: 無し

出力: 無し

```
[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/reboot
http code: 200
```

- ・ **停止**

POST `"/api/poweroff"`

必要権限: Poweroff

パラメータ: 無し

出力: 無し

```
[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/poweroff
http code: 200
```

6.9.6.15. Rest API : ABOS Web 制御

- ・ リスタート

POST `"/api/abosweb/restart"`

必要権限: AbosWebRestart

パラメータ: 無し

出力: コネクションリセット。ABOS Web はリスタートする前に一度終了するためコネクションリセットが発生します。

```
[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/abosweb/restart
http code: 000
curl: (52) Empty reply from server
```

6.9.6.16. Rest API : ユーザー設定とユーザーデータの管理

- ・ ユーザー設定とユーザーデータの削除

POST `"/api/reset_default"`

必要権限: ResetDefault

パラメータ: 無し

出力: `abos-ctrl reset-default` の出力 (stdout または stderr)、および出力ステータス (`exit_code` または `exit_signal`)

```
[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/reset_default
{"stdout": "rm -f /etc/NetworkManager/system-connections/*\n"}
{"stdout": "persist_file -r /etc/NetworkManager/system-connections\n"}
{"stdout": "persist_file -r /etc/dnsmasq.d\n"}
{"stdout": "rc-service dnsmasq restart\n"}
{"stdout": "/etc/init.d/iptables save\n"}
{"stdout": "sed -i -e '/NETAVARK/d' /etc/iptables/rules-save\n"}
{"stdout": "persist_file /etc/iptables/rules-save\n"}
{"stdout": "podman stop -a\n"}
{"stdout": "find /var/app/volumes /var/log -mindepth 1 -delete\n"}
{"stdout": "Starting clone to /dev/mmcblk0p1\n"}
{"stdout": "Cloning rootfs\n"}
{"stdout": "Updating appfs snapshots\n"}
{"stdout": "Reusing up-to-date bootloader\n"}
{"stdout": "Rollback clone successful\n"}
{"stderr": "WARNING: Rebooting!\n"}
{"exit_code": 0}
http code: 200
```

6.9.6.17. Rest API : カスタムスクリプトの実行

ユーザが Armadillo に追加したスクリプトを Rest API を使用して実行することができます。実行したいスクリプトに実行権限を付与し、Armadillo の `/etc/atmark/abos_web/customize_rest` ディレクトリ下に置いてください。

実行に root 権限が必要なスクリプトの場合は、以下のように /etc/doas.d/abos_web_customize.conf にスクリプトを追加してください。

```
[armadillo ~]# cat /etc/doas.d/abos_web_customize.conf
permit nopass abos-web-admin as root cmd /etc/atmark/abos_web/customize_rest/root_command.sh
```

・ 任意のスクリプト実行

POST "/api/custom/{script}"

必要権限: Custom パラメータ: **args** でスクリプトの引数を順番に指定できます。

root を true に設定すると root 権限でスクリプトを実行します。

出力: /etc/atmark/abos_web/customize_rest/{script} {args} {args...} を実行して、そのスクリプトの出力を stdout/stderr で返します。スクリプトが終了した際の出力ステータスは exit_code または exit_signal (どちらも int) です。

```
[armadillo ~]# cat /etc/atmark/abos_web/customize_rest/print_args.sh
#!/bin/sh

printf "arg: %s\n" "$@"
[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/custom/print_args.sh ¥
-H 'Content-type: application/json' -d '{"args": ["param", "second arg"], "root":false}'
{"stdout": "arg: param\n"}
{"stdout": "arg: second arg\n"}
{"exit_code": 0}
```



標準の ABOS Web には最小限の権限しか与えていません。
root 権限でスクリプトを実行する場合、Armadillo の故障やセキュリティにも関わりますので、十分注意して追加してください。

6.9.7. カスタマイズ

ABOS Web をお客様の最終製品へ組み込む場合に、ロゴ画像や背景色、メニューの文言などをカスタマイズすることができます。詳細は「3.9. ABOS Web をカスタマイズする」を参照してください。

6.9.8. ユーザー設定とユーザーデータの削除

カスタマイズと Rest API トークン以外の設定内容と、ユーザーデータを一括削除することができます。

ユーザーデータの削除では以下のデータを削除します。

- ・ /var/app/volumes ディレクトリ下のファイルを全て
- ・ /var/log ディレクトリ下のファイルを全て

ABOS Web のトップページから、「設定管理」ページへ移動し「ユーザー設定とユーザーデータの削除」にある「削除」ボタンを押すと削除できます。削除後は Armadillo が再起動するので引き続き ABOS Web を使用する場合は、再起動が完了してからアクセスしてください。

6.9.9. ABOS Web を停止する

「図 6.98. ABOS Web を停止する」に ABOS Web のサービスを停止する方法を示します。

```
[armadillo ~]# rc-update | grep abos-web ❶
      abos-web |      default
[armadillo ~]# rc-service abos-web status ❷
* status: started
[armadillo ~]# rc-service abos-web stop ❸
abos-web          | * Stopping abos-web ... [ ok ]
[armadillo ~]# rc-update del abos-web ❹
* service abos-web deleted from runlevel default
[armadillo ~]# persist_file -d /etc/runlevels/default/abos-web ❺
```

図 6.98 ABOS Web を停止する

- ❶ OpenRC に ABOS Web のサービスが登録されていることを確認します。
- ❷ ABOS Web のサービスが起動していることを確認します。
- ❸ ABOS Web のサービスを停止します。
- ❹ サービスを管理している OpenRC から ABOS Web のサービスの登録を解除します。
- ❺ サービス設定ファイルの削除を永続化します。

ABOS Web を停止すると ABOS Web の Rest API も使用できなくなります。

6.9.10. ABOS Web を起動する

「図 6.99. ABOS Web を起動する」に ABOS Web のサービスを起動する方法を示します。

```
[armadillo ~]# rc-update | grep abos-web ❶
[armadillo ~]# rc-update add abos-web ❷
* service abos-web added to runlevel default
[armadillo ~]# rc-service abos-web start ❸
abos-web          | * Starting abos-web ... [ ok ]
[armadillo ~]# persist_file /etc/runlevels/default/abos-web ❹
```

図 6.99 ABOS Web を起動する

- ❶ OpenRC に ABOS Web のサービスが登録されていないことを確認します。何も出力されなければ登録されていません。
- ❷ サービスを管理している OpenRC に ABOS Web のサービスを登録します。
- ❸ ABOS Web のサービスを起動します。
- ❹ サービス設定ファイルを永続化します。

6.10. ABOSDE から ABOS Web の機能を使用する

ABOSDE は以下に示す ABOS Web の情報取得や動作を行うことができます。

- ・ Armadillo の SWU バージョンを取得する
- ・ Armadillo のコンテナの情報を取得する

- ・ Armadillo のコンテナを起動・停止する
- ・ Armadillo のコンテナのログを取得する
- ・ Armadillo に SWU をインストールする

ABOSDE は ABOS Web の Rest API を用いて通信を行っていますので、ABOS Web にパスワードでログインができる状態である必要があります。ABOS Web へのログインを行っていない場合は「3.8.1. ABOS Web とは」を参考にしてください。

ABOSDE から ABOS Web の機能を使用するには通信を行う対象の Armadillo を選択する必要があります。「図 6.100. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする」の赤枠で囲まれているボタンをクリックすることで、ローカルネットワーク上で ABOS Web が動作している Armadillo をスキャンすることができます。ただし、ATDE のネットワークを NAT に設定している場合は Armadillo がリストに表示されません。

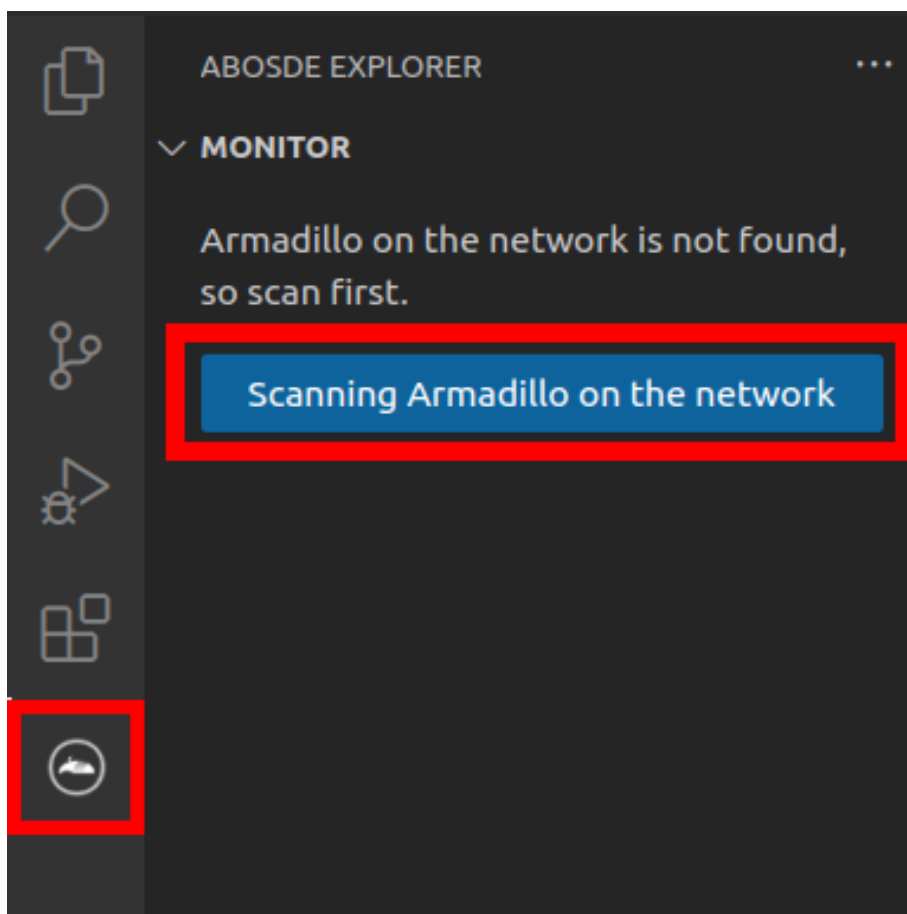


図 6.100 ABOSDE で ローカルネットワーク上の Armadillo をスキャンする

ABOSDE から ABOS Web に初めて通信を行う時、ABOS Web は通信に使用するためのトークンを発行します。そのため、ABOSDE では「図 6.101. ABOSDE の ABOS Web パスワード入力画面」のように ABOS Web のパスワードを求められますので、設定したパスワードを入力してください。

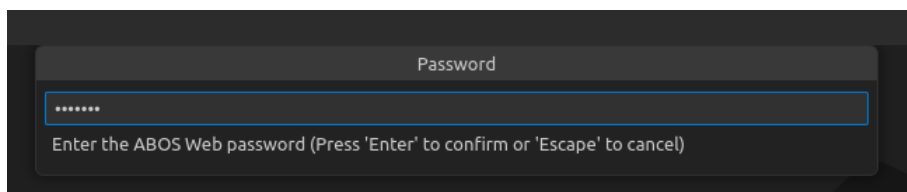


図 6.101 ABOSDE の ABOS Web パスワード入力画面

6.10.1. Armadillo の SWU バージョンを取得する

ローカルネットワーク上の Armadillo をスキャンした後に、「図 6.102. ABOSDE で Armadillo の SWU バージョンを取得」の赤枠で囲まれているボタンをクリックすることで、選択した Armadillo の SWU バージョンを取得することができます。

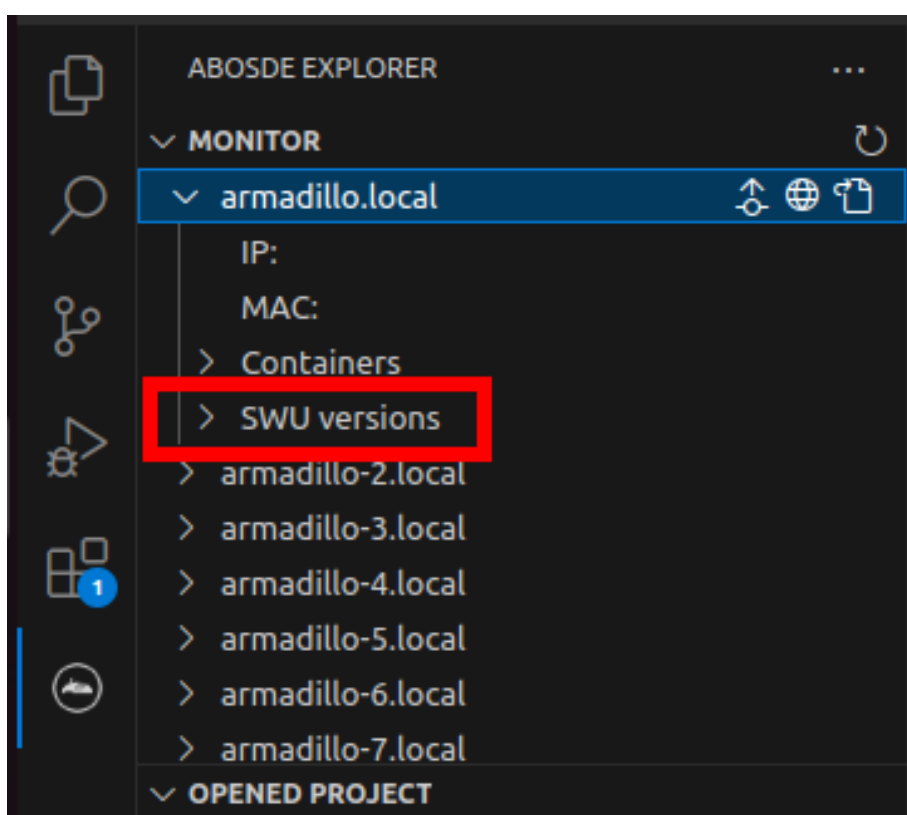


図 6.102 ABOSDE で Armadillo の SWU バージョンを取得

6.10.2. Armadillo のコンテナの情報を取得する

ローカルネットワーク上の Armadillo をスキャンした後に、「図 6.103. ABOSDE で Armadillo のコンテナ情報を取得」の赤枠で囲まれているボタンをクリックすることで、選択した Armadillo のコンテナの情報を取得できます。表示されるコンテナの情報は以下の通りとなります。

- ・ **state** : コンテナが起動中の場合は running、コンテナが停止中の場合は exited
- ・ **image** : コンテナのイメージ名
- ・ **command** : コンテナ起動時に実行しているコマンド

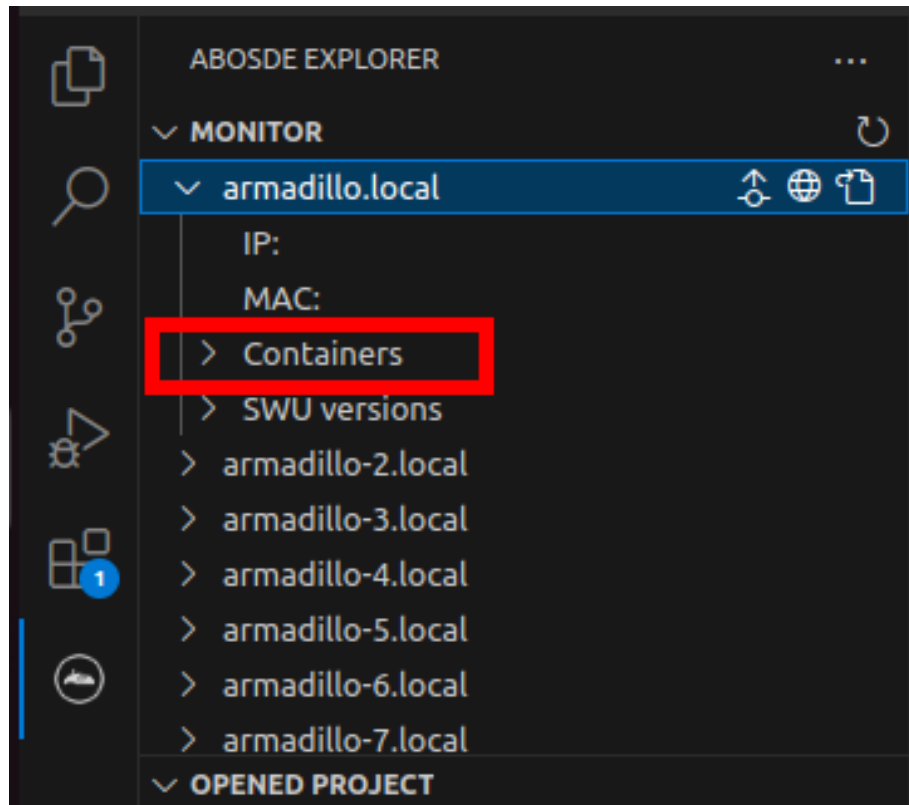


図 6.103 ABOSDE で Armadillo のコンテナ情報を取得

6.10.3. Armadillo のコンテナを起動・停止する

ローカルネットワーク上の Armadillo をスキャンした後に、「図 6.104. ABOSDE で Armadillo のコンテナを起動」の赤枠で囲まれているボタンをクリックすることで、選択したコンテナを起動することができます。コンテナを起動できた場合はコンテナの status が running に変化します。また、「図 6.105. ABOSDE で Armadillo のコンテナを停止」の赤枠で囲まれているボタンをクリックすることで、選択したコンテナを停止することができます。コンテナを停止できた場合はコンテナの status が exited に変化します。

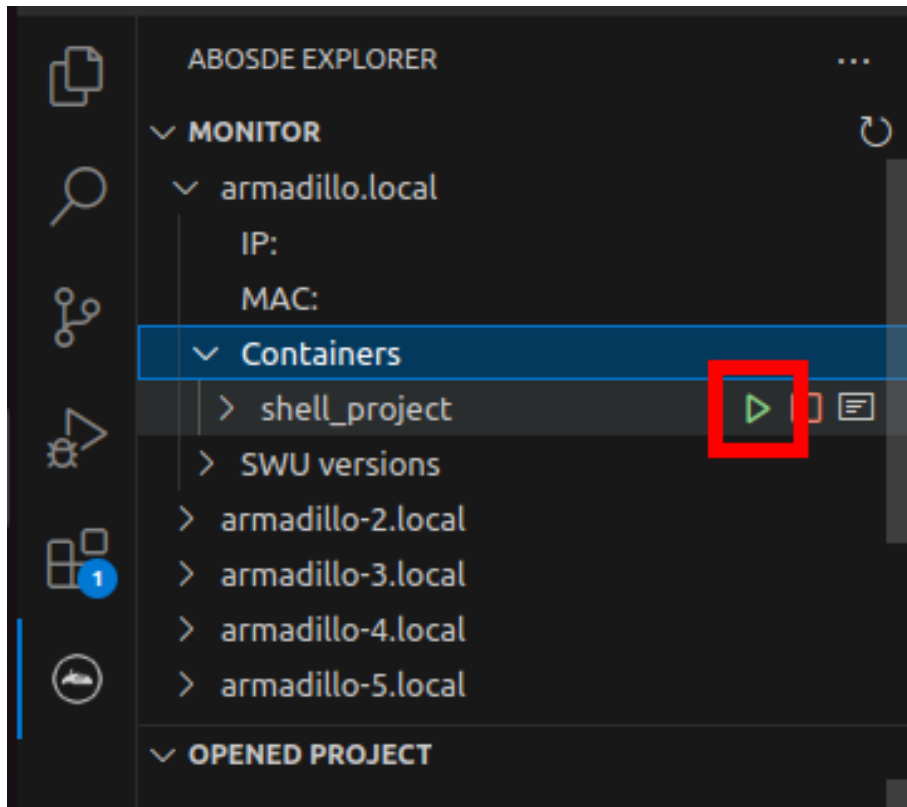


図 6.104 ABOSDE で Armadillo のコンテナを起動

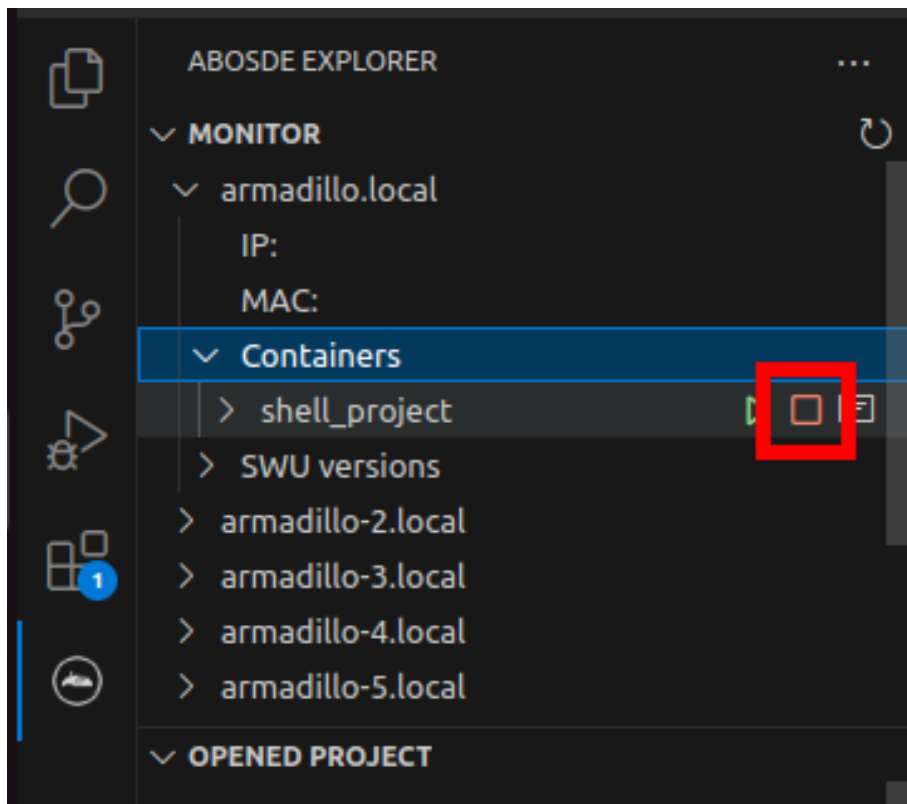


図 6.105 ABOSDE で Armadillo のコンテナを停止

6.10.4. Armadillo のコンテナのログを取得する

「図 6.106. ABOSDE で Armadillo のコンテナのログを取得」の赤枠で囲まれているボタンをクリックすることで、コンテナが出力したログを取得することができます。ログは VS Code のテキストエディタに開かれます。コンテナが何もログを出力していない場合は表示されません。

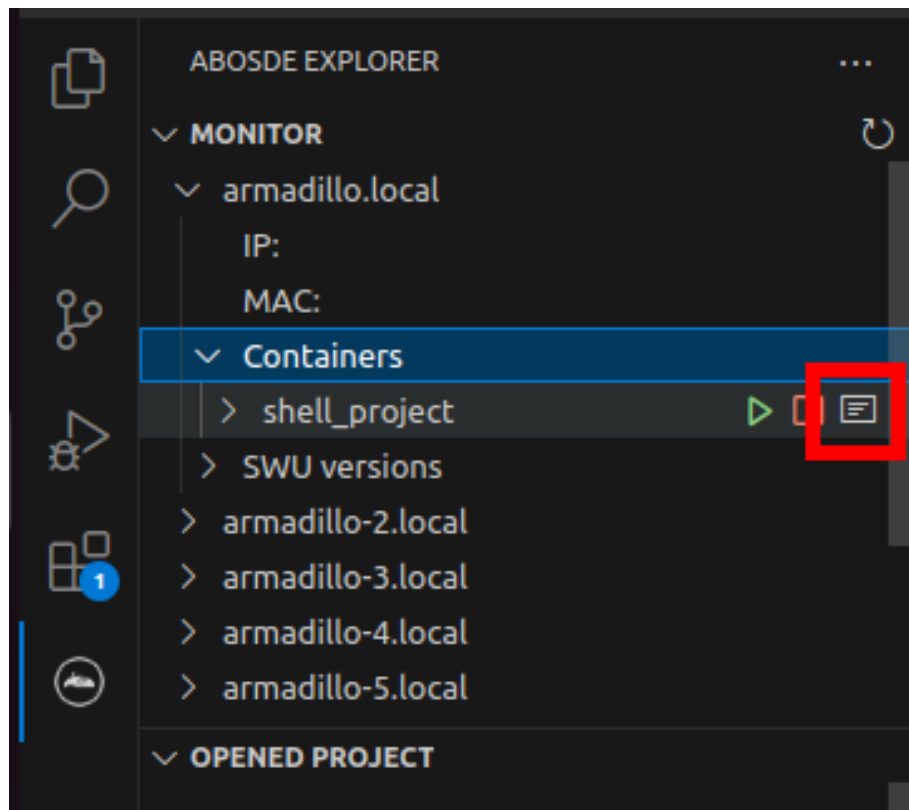


図 6.106 ABOSDE で Armadillo のコンテナのログを取得

6.10.5. Armadillo に SWU をインストールする

ローカルネットワーク上の Armadillo をスキャンした後に、「図 6.107. ABOSDE で Armadillo に SWU をインストール」の赤枠で囲まれているボタンをクリックすることで、選択した Armadillo に SWU をインストールすることができます。SWU インストールのログは VS Code 画面下部の OUTPUT に表示されます。

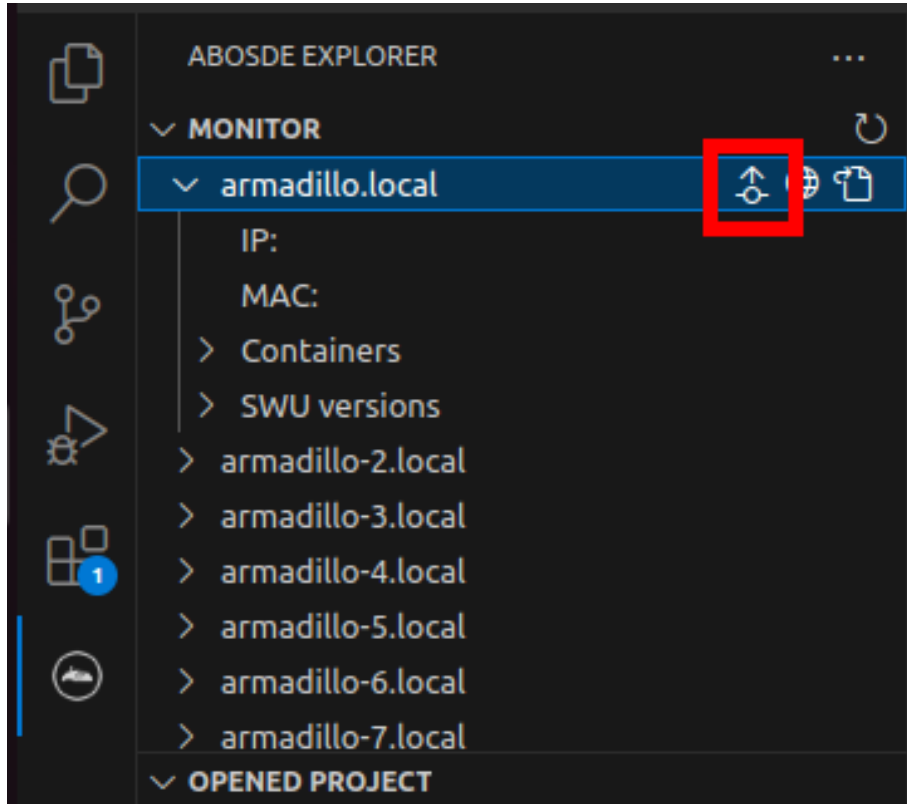


図 6.107 ABOSDE で Armadillo に SWU をインストール

6.11. ssh 経由で Armadillo Base OS にアクセスする

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 (mmcblk2p1): 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。

6.12. コマンドラインからネットワーク設定をする

基本的に、Armadillo-640 のネットワーク設定は、「3.8. ネットワーク設定」で紹介したとおり、ABOS Web で行います。しかし、ABOS Web で対応できない複雑なネットワーク設定を行いたい場合などは、コマンドラインからネットワークの設定を行うことも可能です。

ここでは、コマンドラインからネットワークを設定する方法について説明します。

6.12.1. 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 6.8 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

6.12.2. 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:00:0b:79 brd ff:ff:ff:ff:ff:ff
   inet 172.16.1.84/16 brd 172.16.255.255 scope global dynamic noprefixroute eth0
       valid_lft 28786sec preferred_lft 28786sec
   inet6 fe80::e9c0:7b3c:c0c9:3c4/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

図 6.108 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 172.16.1.84/16 brd 172.16.255.255 scope global dynamic noprefixroute eth0
       valid_lft 28656sec preferred_lft 28656sec
   inet6 fe80::e9c0:7b3c:c0c9:3c4/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

図 6.109 IP アドレス(eth0)の確認

6.12.3. 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。

6.12.3.1. 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 6.110. 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 6.110 nmcli のコマンド書式

6.12.4. 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。

6.12.4.1. 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。^[1]

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

図 6.111 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。

6.12.4.2. 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。

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

図 6.112 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 6.113 コネクションの無効化

6.12.4.3. 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。

^[1] 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 add con-name [ID] type [type] ifname [interface name]
```

図 6.114 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「6.1. 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 6.115 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「3.3.3. アップデート機能について」を参考にしてください。

6.12.4.4. 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 6.116 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 6.117 コネクションファイル削除時の永続化

6.12.4.5. 固定 IP アドレスに設定する

「表 6.9. 固定 IP アドレス設定例」の内容に設定する例を、「図 6.118. 固定 IP アドレス設定」に示します。

表 6.9 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 6.118 固定 IP アドレス設定

6.12.4.6. DNS サーバーを指定する

DNS サーバーを指定する例を、「図 6.119. 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 6.119 DNS サーバーの指定

6.12.4.7. DHCP に設定する

DHCP に設定する例を、「図 6.120. DHCP の設定」に示します。

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

図 6.120 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。

6.12.4.8. 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 6.121 コネクションの修正の反映

6.12.4.9. 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 6.122 デバイスの一覧

6.12.4.10. 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 6.123 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などで有効なコネクションがあるかを確認してください。

6.12.4.11. 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 6.124 デバイスの切断

6.12.5. 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 6.125 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。

6.12.6. ファイアウォールの設定方法

開放しているポートが存在すると攻撃者の標的になる可能性があります。開発したサーバーが使用するポートに対して、アクセスできる IP アドレスを制限することでセキュリティ上のリスクを低減することができます。

ここでは、iptables コマンドを使用した、パケットフィルタリングによるアクセス制限方法を紹介します。

「図 6.126. 特定のポートに対する IP アドレスのフィルタリング」の例では、Armadillo の特定のポートに対して、特定の IP アドレスからのアクセスのみを受け入れるようにします。この例では、<送信元 IP アドレス> は Armadillo にパケットを送信する IP アドレス、<ポート番号> はパケットを受け入れる Armadillo のポート番号、<プロトコル> は通信プロトコルのことを指します。また、<ポート番号> はパケットを受け入れる Armadillo のポート番号のことを指します。

```
[armadillo ~]# iptables -I INPUT -s <送信元 IP アドレス> -p <プロトコル> --dport <ポート番号> -j ACCEPT ❶
[armadillo ~]# iptables -A INPUT -p <プロトコル> --dport <ポート番号> -j REJECT ❷
[armadillo ~]# iptables -L ❸
Chain INPUT (policy ACCEPT)
target     prot opt source                destination
ACCEPT    <プロトコル> -- <送信元 IP アドレス> anywhere          <プロトコル> dpt:<ポート番号>
REJECT    <プロトコル> -- anywhere              anywhere          <プロトコル> dpt:<ポート番号>
> reject-with icmp-port-unreachable
... 省略

[armadillo ~]# /etc/init.d/iptables save ❹
[armadillo ~]# persist_file /etc/iptables/rules-save ❺
```

図 6.126 特定のポートに対する IP アドレスのフィルタリング

- ❶ <ポート番号> に <送信元 IP アドレス> から送られてきたパケットを受け入れるように設定します。
- ❷ <ポート番号> に <送信元 IP アドレス> 以外から送信されてきたパケットを拒否するように設定します。
- ❸ 想定通りに設定されているか確認します。
- ❹ 上記の設定を設定ファイル /etc/iptables/rules-save に保存します。
- ❺ 保存した設定ファイルを永続化します。

「図 6.126. 特定のポートに対する IP アドレスのフィルタリング」はあくまで一例ですが、このように iptables コマンドを用いることで開発したサーバーにアクセスできる IP アドレスを制限することができます。

上記の設定を削除する場合は「図 6.127. 特定のポートに対する IP アドレスのフィルタリングの設定を削除」に示すコマンドを実行してください。

```
[armadillo ~]# armadillo:~# iptables -L --line-number ❶
Chain INPUT (policy ACCEPT)
num target prot opt source destination
1 ACCEPT <プロトコル> -- <送信元 IP アドレス> anywhere <プロトコル> dpt:<ポート番号>
2 REJECT <プロトコル> -- anywhere anywhere <プロトコル> dpt:<ポ
<ポート番号> reject-with icmp-port-unreachable
... 省略

[armadillo ~]# iptables -D INPUT 2 ❷
[armadillo ~]# iptables -D INPUT 1 ❸
[armadillo ~]# /etc/init.d/iptables save ❹
[armadillo ~]# persist_file /etc/iptables/rules-save ❺
```

図 6.127 特定のポートに対する IP アドレスのフィルタリングの設定を削除

- ❶ 削除する設定の番号 (num) を確認します。ここでは 1 番と 2 番の設定を削除します。
- ❷ 2 番の設定を削除します。
- ❸ 1 番の設定を削除します。
- ❹ 上記の設定を設定ファイル /etc/iptables/rules-save に保存します。
- ❺ 保存した設定ファイルを永続化します。

6.13. ストレージの操作

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。

6.13.1. ストレージ内にアクセスする

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 6.128 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。

SD インターフェース (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 6.129 ストレージのマウント

6.13.2. ストレージを安全に取り外す

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 6.130 ストレージのアンマウント

6.13.3. 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 6.131. 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 6.131 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 6.132 EXT4 ファイルシステムの構築

6.14. ボタンやキーを扱う

buttd サービスを使用することで、ボタンやキー入力をトリガーとする処理を簡単に実装できます。

/etc/atmark/buttd.conf に BUTTD_ARGS を指定することで、動作を指定することができます：

- ・ `--short <key> --action "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。認識する最大時間は `--time <time_ms>` オプションで変更可能です。
- ・ `--long <key> --action "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。長押しと認識する最低時間は `--time <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。途中でキーを離した場合は、キーを離した時間に合った "command" を実行します。(例 : `buttd --short <key> --action "cmd1" --long <key> --time 2000 --action "cmd2" --long <key> --time 10000 --action "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。
- ・ `--exit-timeout <time_ms>` : 設定した時間の後に buttd を停止します。起動時のみに対応したい場合に使えます。
- ・ キーの設定の `--exit-after` オプション : キーのコマンドを実行した後に buttd を停止します。キーの対応を一回しか実行しないように使えます。

6.14.1. SW1 の短押しと長押しの対応


以下にデフォルトを維持したままで SW1 の短押しと長押しのそれぞれの場合にコマンドを実行させる例を示します。

```
[armadillo ~]# vi /etc/atmark/buttd.conf ❶
BUTTD_ARGS="$BUTTD_ARGS --short prog1 --action 'date >> /tmp/shortpress'"
BUTTD_ARGS="$BUTTD_ARGS --long prog1 --time 5000 --action 'date >> /tmp/longpress'"
[armadillo ~]# persist_file /etc/atmark/buttd.conf ❷
[armadillo ~]# rc-service buttd restart ❸
buttd          | * Stopping button watching daemon ...           [ ok ]
buttd          | * 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 6.133 buttd で SW1 を扱う

- ❶ buttd 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/shortpress 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。

- ③ 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。
- ④ 押された回数を確認します。



Debian 版と ABOS 3.18.4-at.4 以前では、SW1 の押下を ENTER キーのリリースとして割り当てていました。ABOS 3.18.4-at.4 以降の ABOS で同じ動作にしたい場合は `armadillo-600-button-enter.dtbo` を `/boot/overlays.txt` に追加してください。詳細は「3.5.5. DT overlay によるカスタマイズ」を参照ください。

6.14.2. USB キーボードの対応

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-*/ * ①
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 ②
[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
```

図 6.134 buttond で USB キーボードのイベントを確認する

- ① `buttond` を `-vvv` で冗長出力にして、すべてのデバイスを指定します。
 - ② 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 `buttond` が停止します。

```
[armadillo ~]# vi /etc/atmark/buttond.conf
BUTTOND_ARGS="$BUTTOND_ARGS -i /dev/input/by-id/usb-0566_3029-event-kbd"
BUTTOND_ARGS="$BUTTOND_ARGS --short LEFTCTRL --action 'podman_start
button_pressed_container'"
```

```
[armadillo ~]# persist_file /etc/atmark/buttond.conf
[armadillo ~]# rc-service buttond restart
```

図 6.135 buttond で USB キーボードを扱う

6.14.3. Armadillo 起動時にのみボタンに反応する方法

Armadillo 起動時にのみ、例として SW1 の長押しに反応する方法を紹介します。

/etc/local.d/boot_switch.start に稼働期間を指定した buttond を起動させる設定を記載します。

buttond が起動してから 10 秒以内に SW1 を一秒以上長押しすると myapp のコンテナの親プロセスに USR1 信号を送ります（アプリケーション側で信号を受信して、デバッグモードなどに切り替える想定です）。SW1 が Armadillo 起動前に押された場合は、buttond の起動一秒後に実行されます。

```
[armadillo ~]# vi /etc/local.d/boot_switch.start
#!/bin/sh

buttond /dev/input/by-path/platform-gpio-keys-event ¥ ❶
    --exit-timeout 10000 ¥ ❷
    --long PROG1 --time 1000 --exit-after ¥ ❸
        --action "podman exec myapp kill -USR1 1" & ❹
[armadillo ~]# chmod +x /etc/local.d/boot_switch.start
[armadillo ~]# persist_file /etc/local.d/boot_switch.start
```

図 6.136 buttond で SW1 を Armadillo 起動時のみ受け付ける設定例

- ❶ SW1 の入力を /dev/input/by-path/platform-gpio-keys-event ファイルの PROG1 として認識できます。
- ❷ buttond 起動後 10 秒経過すると終了します。
- ❸ SW1 を一度検知した後すぐに終了します。
- ❹ サービスとして動作させる必要がないため & を付けてバックグラウンド起動します。

6.15. 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。

6.15.1. 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かめる必要があります。

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。

6.15.2. 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 6.137 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。

6.15.3. atmark-thermal-profiler を実行・停止する

「図 6.138. 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 6.138 atmark-thermal-profiler を実行する

「図 6.139. 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 6.139 atmark-thermal-profiler を停止する

6.15.4. 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 6.140 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 6.10. thermal_profile.csv の各列の説明」に記載します。

表 6.10 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。
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。

6.15.5. 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。

6.15.5.1. 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 6.141 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。

6.15.5.2. 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 6.142. 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。

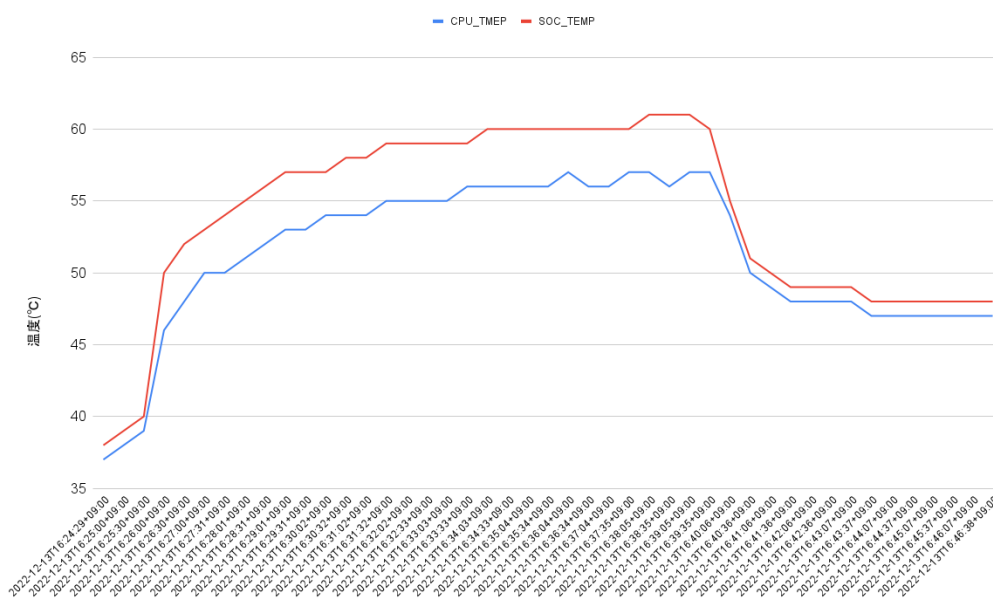


図 6.142 Armadillo-IoT ゲートウェイ G4 で取得した温度のグラフ

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、「6.15.5.1. 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。

6.15.5.3. 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。各列について詳しくは「表 6.10. 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の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。

6.15.6. Armadillo Twin から Armadillo の温度を確認する

atmark-thermal-profiler の他に、Armadillo Twin から温度や CPU 負荷率等の情報を確認することができます。詳細は Armadillo Twin ユーザーマニュアル「デバイス監視アラートを管理する」[<https://manual.armadillo-twin.com/management-device-monitoring-alert/>] をご確認ください。

6.15.7. 温度センサーの仕様

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

6.16. Armadillo Base OS をアップデートする

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。「6.5. 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。

6.17. ロールバック状態を確認する

Armadillo Base OS のルートファイルシステムが破損し起動できなくなった場合、自動的に以前のバージョンで再起動します。

abos-ctrl status コマンドでロールバックされてるかどうかを確認できます。

```
[armadillo ~]# abos-ctrl status
Currently booted on /dev/mmcblk2p1
Last update on Fri Jun 7 16:03:37 JST 2024, updated: ❶
  boot: 2020.4-at23-00001-g01508f65b8 -> 2020.4-at23
  base_os: 3.19.1-at.3.20240523.pc.gtr -> 3.19.1-at.4
rollback-status: OK: available, no auto-rollback ❷
```

図 6.143 abos-ctrl status の例


- ❶ 最新のアップデートの日付と内容が出力されています。
- ❷ 「表 6.11. rollback-status の出力と意味」「表 6.12. rollback-status 追加情報の出力と意味」に示す状態と追加情報が出力されています。

表 6.11 rollback-status の出力と意味

出力	説明
OK	ロールバックされていません。
rolled back	ロールバックされています。

表 6.12 rollback-status 追加情報の出力と意味

出力	説明
no fallback (fresh install)	初期化状態。
no fallback	何かの理由で B 面が起動できない状態になっています (アップデート失敗後等)。
auto-rollback enabled (post-update)	アップデート直後でもまだ再起動していない状態です。再起動して失敗した場合にロールバックが発生します。
auto-rollback enabled (cloned)	abos-ctrl rollback-clone コマンドを実行した後の状態です。同じくロールバック可能です。
available, no auto-rollback	アップデートの後に正常に起動できたので、自動ロールバックが無効になっていますが abos-ctrl rollback --allow-downgrade コマンドで手動ロールバック可能です。



Armadillo Base OS 3.19.1-at.4 以下のバージョンでは「アップデート直後」の概念がなかったため、ステータスは「no fallback」(B 面がない状態)、「optimal」(ロールバック可能)、と「rolled back」の 3 択だけでした。

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。以下の例では、Armadillo Base OS を更新した後に起動できないカーネルをインストールして、起動できなかったためにロールバックされました。

```
[armadillo ~]# cat /var/at-log/atlog
Jun  7 16:03:37 armadillo NOTICE swupdate: Installed update to /dev/mmcblk0p2: ¥
boot: 2020.4-at22 -> 2020.4-at23, base_os: 3.19.1-at.3 -> 3.19.1-at.4
Jun  7 16:11:39 armadillo NOTICE swupdate: Installed update to /dev/mmcblk0p1: ¥
extra_os.kernel: unset -> 5.10.218-1
Jun  7 16:12:18 armadillo WARNING uboot: reset by wdt
Jun  7 16:12:42 armadillo WARNING uboot: reset by wdt
Jun  7 16:13:06 armadillo WARNING uboot: reset by wdt
Jun  7 16:13:09 armadillo WARNING uboot: Counted 3 consecutive unfinished boots
Jun  7 16:13:09 armadillo WARNING uboot: Rolling back to mmcblk0p2
```

図 6.144 /var/at-log/atlog の内容の例

6.18. Armadillo 起動時にコンテナの外でスクリプトを実行する

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。「6.2.4. コンテナ起動設定ファイルを作成する」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 6.145 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。
- ❸ スクリプトを保存して、再起動します。
- ❹ 実行されたことを確認します。

6.19. 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 ❶
# 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 ❷
'/boot/uboot_env.d/no_prompt' -> '/mnt/boot/uboot_env.d/no_prompt'
[armadillo ~]# fw_setenv -s /boot/uboot_env.d/no_prompt ❸
Environment OK, copy 0
```



```
[armadillo ~]# fw_printenv | grep bootdelay ④
bootdelay=-2
```

図 6.146 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。



「6.21.1. 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 6.13 u-boot の主要な環境変数

環境変数	説明	デフォルト値
console	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 reset_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、「3.3.3.5. 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
upgrade_availability	1 以上の場合 bootcount を管理してロールバック可能になります。0 か空の場合はロールバックできません。値を abos-ctrl status で確認できます。	状況による

環境変数	説明	デフォルト値
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"> ・ -1: キー入力の有無に関らず保守モードに遷移します。 ・ -2: キー入力の有無に関らず保守モードに遷移しません。 ・ -3: キー入力の有無に関らず保守モードに遷移しません。ただし、SW1 が押下されている場合は保守モードに遷移します。 	0
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、"mmcpart"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/ulimage
fdt_file	DTB ファイルのパスです。"mmcdev"で指定されたデバイスの、"mmcpart"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、"mmcpart"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、「3.5.5. 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"> ・ 0: eMMC ・ 1: microSD/microSDHC/microSDXC カード "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。	0
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。"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。	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。"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。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。	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

6.19.1. u-boot の環境変数の変更を制限する

u-boot のソースに含まれる u-boot-[VERSION]/configs/armadillo-640_defconfig に CONFIG_ENV_WRITEABLE_LIST=y を追加すると、変更可能と明示したもの以外の環境変数を変更不可にすることができます。変更可能とする環境変数のリストは u-boot-[VERSION]/include/configs/armadillo-640.h ファイルの CFG_ENV_FLAGS_LIST_STATIC で設定します。

デフォルトのコンフィグでは、以下の環境変数に変更可能です：

- ・ upgrade_available と bootcount: ロールバック機能に必要な変数です。ロールバック機能を無効にする場合は必ず upgrade_available のデフォルト値も空にしてください。
- ・ ethaddr: ネットワークコマンド関連の変数です。デフォルトのブートコマンドはネットワークを使用しませんので動作に影響ありません。

u-boot のソースの取得方法、ビルド方法およびインストール方法については「6.21.1. ブートローダーをビルドする」を参照してください。ビルドしたものをインストールすると CFG_ENV_FLAGS_LIST_STATIC で設定した環境変数以外は変更できなくなります。

6.20. 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。

6.20.1. ブートディスクの作成

1. ブートディスクイメージをビルドします

「6.21.3. 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「6.21.1. 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「3.1.2.7. 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,mask=0022,dmask=0077,codepage=cp437,ioccharset
```



```
=utf8,shortname=mixed,showexec,utf8,flush,errors=remount-ro,uhelper=udisks)
[ATDE ~]$ sudo umount /dev/sdb1
```

図 6.147 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 6.14 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/mmcbk1: 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

6.20.2. SD ブートの実行

「6.20.1. 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.MX6GULL 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/ttyxc0)

armadillo login:
```

6.21. 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。

6.21.1. 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 (u-boot-[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/source/u-
boot-[VERSION].tar.gz
[ATDE ~]$ tar xf u-boot-[VERSION].tar.gz
[ATDE ~]$ cd u-boot-[VERSION]
```

2. デフォルトコンフィギュレーションの適用

「[図 6.148. 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 CROSS_COMPILE=arm-linux-gnueabihf-
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 6.148 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 ARCH=arm 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 「3.3.3.6. SWU イメージのインストール」を参照ください。

- ・ 「6.20.1. 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。

6.21.2. 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。「6.21.3. 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 CROSS_COMPILE=arm-linux-gnueabihf-
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 ~/linux-[VERSION]]$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
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。

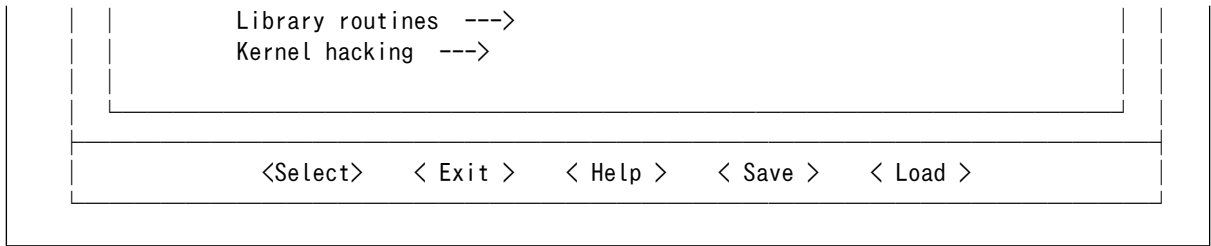
```
.config - Linux/arm 5.10.145 Kernel Configuration
```


```

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 --->

```



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 6.149 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 「3.3.3.6. SWU 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 6.150. 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 6.150 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。

6.21.3. 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 6.15 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、「6.20.1. ブートディスクの作成」や「3.3.5.1. 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、「3.3.5.1. 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]]$ ./build_rootfs.sh -b a600
use default(outdir=/home/atmark/git/build-rootfs)
use default(output=baseos-600-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-600-[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
```



「Alpine Linux ルートファイルシステムビルドツール」のバージョンが 3.18-at.7 以降を使用している場合は、ビルドが終わると SBOM も `[output].spdx.json` として出力されます。ライセンス情報等を記載するためのコンフィグファイルはデフォルトは `baseos_sbom.yaml` となっています。コンフィグファイルを変更する場合は `--sbom-config <config>` に引数を入れてください。SBOM が不要な場合は `--nosbom` を引数に追加してください。

SBOM のライセンス情報やコンフィグファイルの設定方法については「6.22.3. ビルドしたルートファイルシステムの SBOM を作成する」をご覧ください。

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-600-[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「3.3.3.6. SWU イメージのインストール」を参照ください。

- ・「6.20.1. 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-600-[VERSION].tar.zst` が自動的に利用されます。

6.22. SBOM の提供

アットマークテクノでは ABOS 及び ABOS 上で動作する標準ソフトウェアの SBOM を提供しています。また、開発したソフトウェアの SWU イメージを作成するタイミングで SBOM を生成することができます。SBOM 生成手順は「6.22.3. ビルドしたルートファイルシステムの SBOM を作成する」もしくは「6.22.4. SWU イメージと同時に SBOM を作成する」を参照ください。

6.22.1. SBOM について

SBOM (Software Bill of Materials: ソフトウェア部品表) は、ソフトウェアを構成するコンポーネントやソフトウェア間の依存関係、ライセンス情報を記したリストです。経済産業省は、ソフトウェアサプライチェーンが複雑化する中で、急激に脅威が増しているソフトウェアのセキュリティを確保するための管理手法の一つとして SBOM の導入を推進しています。SBOM の導入はソフトウェアのトレーサビリティを確保し、脆弱性残留リスクの低減、脆弱性対応期間の低減に繋がります。アットマークテクノが提供する SBOM は ISO/IEC5962 で国際標準となっている SPDX2.2 のフォーマットに準拠しています。

SPDX2.2 の詳細については以下のドキュメントをご参照ください。

The Software Package Data Exchange® (SPDX®) Specification Version 2.2.2 [<https://spdx.github.io/spdx-spec/v2.2.2/>]

アットマークテクノの提供する mkswu コマンドでは SWU を作成するタイミングで SBOM を生成することができます。

6.22.2. SBOM の利点

SBOM の利点はソフトウェアのサプライチェーン攻撃への対応です。ソフトウェアのセキュリティ対策は日々見直されており、トレーサビリティが明らかになることで、ソフトウェアに含まれる脆弱性に速やかに対処することが可能になります。SBOM はトレーサビリティを辿るのに優れており、加えて、脆弱性スキャンツールを用いることで、表面化していない脆弱性の発見に利用できます。脆弱性スキャンツールには例として、Google が提供する osv-scanner が挙げられます。脆弱性に関する詳細なリンクや、脆弱性の深刻度を示す CVSS(Common Vulnerability Scoring System) を出力します。アットマークテクノが提供する SBOM は osv-scanner のスキャンに対応しています。

osv-scanner の詳細については以下をご参照ください。

OSV-Scanner [<https://github.com/google/osv-scanner/>]

アットマークテクノが提供している ABOS は GPLv3 (GNU General Public License 第 3 版) のソフトウェアを含まない構成で提供しています。OSS (オープンソース・ソフトウェア) 利用者に広く普及している GPLv3 は、インストール用情報の開示義務、関連する特許ライセンスの許諾について定める条項が含まれ、組み込み機器に適用する際の妨げになる場合があります。SBOM にはパッケージのライセンス情報が含まれているため、GPLv3 ライセンスが含まれているかどうかの検出を可能にします。

6.22.3. ビルドしたルートファイルシステムの SBOM を作成する

「6.21.3. Alpine Linux ルートファイルシステムをビルドする」を実行すると、OS_update.swu と同じ場所に SBOM を作成します。SBOM を作成するには、作成する対象のファイルとライセンス情報等を記載するためのコンフィグファイルが必要となります。また、baseos-600-[VERSION].tar.zst から、アーカイブに含まれるパッケージ情報やファイル情報を SBOM に記載します。

ライセンス情報等を記載するためのコンフィグファイルの例は以下のコマンドで確認することができます。各項目に関する説明はコメントに記載しておりますので、必要に応じて値を変更してください。

各項目の詳細な説明については SPDX specification v2.2.2 (<https://spdx.github.io/spdx-spec/v2.2.2/>) をご覧ください。

```
[ATDE ~/build-rootfs-[VERSION]]$ cat submodules/make-sbom/config.yaml
```

作成したコンフィグファイルと、baseos-600-[VERSION].tar.zst から OS_update.swu の SBOM を作成します。

```
[ATDE ~/build-rootfs-[VERSION]]$ ./build_sbom.sh -i OS_update.swu -c <コンフィグファイル> -f
baseos-600-[VERSION].tar.zst
INFO:root:created OS_update.swu.spdx.json
```

↪

作成される SBOM は OS_update.swu.spdx.json になります。json 形式で ISO/IEC5962 で国際標準となっている SPDX2.2 のフォーマットに準拠しています。

アットマークテクノが提供しているソフトウェアの SBOM はソフトウェアダウンロード [<https://armadillo.atmark-techno.com/armadillo-640/resources/software/>]の各ソフトウェアダウンロードページからダウンロードすることができます。

6.22.4. SWU イメージと同時に SBOM を作成する

「5.4.1. SWU イメージの作成」の実行時に SBOM を作成する方法について説明します。SWU イメージは desc ファイルから作成されます。この desc ファイルに SBOM 作成に必要な情報についても記載します。

6.22.4.1. コンフィグファイルを作成する

SBOM を作成するには、作成する対象のファイルとライセンス情報等を記載するためのコンフィグファイルが必要となります。コンフィグファイルについて指定がない場合はデフォルトのコンフィグファイルで SBOM を作成します。デフォルトのコンフィグファイルは /usr/share/make-sbom/config/config.yaml にあります。このファイルは SBOM 作成ツールによって配置されます。コンフィグファイルを編集するために、例としてカレントディレクトリにコピーします。リリース時には正しいコンフィグファイルの内容を記載してください。

```
[ATDE ~]$ cp /usr/share/make-sbom/config/config.yaml .
[ATDE ~]$ vi config.yaml
```

ライセンス情報等を記載するためのコンフィグファイルの例は以下のコマンドで確認することができます。各項目に関する説明はコメントに記載しておりますので、必要に応じて値を変更してください。各項目の詳細な説明については SPDX specification v2.2.2 (<https://spdx.github.io/spdx-spec/v2.2.2/>) をご覧ください。

「6.22.4.2. desc ファイルを編集する」で desc ファイルに編集したコンフィグファイルのパスを指定します。

6.22.4.2. desc ファイルを編集する

SBOM 作成のために、desc ファイルに記載する項目を以下に示します。

表 6.16 desc ファイルの設定項目

項目	設定値	説明
swdesc_option BUILD_SBOM=<mode>	auto(デフォルト): SBOM 作成ツールがある場合作成する yes: SBOM を作成する。SBOM 作成ツールがない場合はエラーする no: SBOM を作成しない	SBOM を作成するかどうか。記載がない場合は auto が選択される
swdesc_option sbom_config_yaml=<path>	ファイルパス	コンフィグファイルのパスを指定する。記載がない場合はデフォルトのコンフィグファイルを使用する
swdesc_sbom_source_file <path>	ファイルパス	SBOM に含めるファイルを指定する。記載がない場合は SBOM に含まれない

以下に desc ファイルの記載例について示します。

```
swdesc_option component=make_sbom
swdesc_option version=1
swdesc_option BUILD_SBOM=yes❶
swdesc_option sbom_config_yaml=config.yaml❷

swdesc_sbom_source_file manifest.json❸
```

図 6.151 desc ファイルの追加例

- ❶ SBOM を作成するように設定します。例として必ず作成するように "yes" を指定します。
- ❷ コンフィグファイルのパスを設定します。例としてカレントディレクトリにある config.yaml を指定します。
- ❸ SBOM に含めたいファイルがある場合に指定します。例として manifest.json を指定します。

desc ファイルの作成が出来たら「5.4.1. SWU イメージの作成」を実行すると、SWU イメージと同じ場所に SBOM が作成されます。desc ファイルの内容によっては SBOM 作成に数分かかります。作成される SBOM のファイル名は <SWU イメージ名>.spdx.json になります。json 形式で ISO/IEC5962 で国際標準となっている SPDX2.2 のフォーマットに準拠しています。

6.23. 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。

6.23.1. 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 mk squashfs 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]$ mk squashfs sample squashfs.img
```

図 6.152 squashfs イメージの作成

6.23.2. squashfs イメージを書き込む

以降の作業は Armadillo 上で行います。

「6.23.1. 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 へ書き込みます。



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
```

6.23.3. 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 6.153 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 6.154 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。
- ❷ /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
```

6.23.4. 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 ❶  
[armadillo]# persist_file /opt/sample/  
[armadillo]# vi /etc/fstab  
:  
:(省略)  
:  
/dev/mmcblk0gp3 /opt/sample squashfs defaults,nofail 0 0 ❷  
[armadillo]# persist_file /etc/fstab
```

- ❶ squashfs イメージをマウントするディレクトリを作成します
- ❷ 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)
```

6.24. 動作ログ

6.24.1. 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。

Armadillo-640 で /var/log 配下に出力するログに関しては「6.24.5. /var/log/ 配下のログに関して」を参照ください。

6.24.2. 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。



/var/at-log/atlog はファイルサイズが 3MiB になるとローテートされ /var/at-log/atlog.1 に移動されます。

/var/at-log/atlog.1 が存在する状態で、更に /var/at-log/atlog のファイルサイズが 3MiB になった場合は、/var/at-log/atlog の内容が /var/at-log/atlog.1 に上書きされます。/var/at-log/atlog.2 は生成されません。

6.24.3. 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 6.155 動作ログのフォーマット

atlog には以下の内容が保存されています。

- ・ インストール状態のバージョン情報
- ・ swupdate によるアップデートの日付とバージョン変更
- ・ abos-ctrl / uboot の rollback 日付
- ・ uboot で wdt による再起動があった場合にその日付

6.24.4. 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。

6.24.5. /var/log/ 配下のログに関して

「表 6.17. /var/log/ 配下のログ」に Armadillo-640 で /var/log/ 配下に出力するログを示します。

最大ファイルサイズを超えると「表 6.17. /var/log/ 配下のログ」の「ファイル名」の 2 行目に記載されたファイル名にコピーします。

その状態から更に最大ファイルサイズを超えた場合、「表 6.17. /var/log/ 配下のログ」の「ファイル名」の 2 行目に記載されたファイル名に上書きします。

表 6.17 /var/log/ 配下のログ

ファイル名	説明	最大ファイルサイズ	最大ファイル数
/var/log/messages /var/log/messages.0	通常のログです。	4MiB	2

ファイル名	説明	最大ファイルサイズ	最大ファイル数
/var/log/armadillo-twin-agent/agent_log /var/log/armadillo-twin-agent/agent_log. 1	Armadillo Twin Agent の動作ログです。	1MiB	2

6.25. シリアル通信ソフトウェア(minicom)のセットアップ



ATDE9 v20240925 以降の ATDE では以下の設定を実施した状態のイメージを配布しています。これより前のバージョンの場合は、次の手順に沿って minicom のシリアル通信設定を実施してください。

minicom を使用して Armadillo とシリアルコンソール経由で通信を行うためには、「表 6.18. シリアル通信設定」のとおりあらかじめ設定しておく必要があります。ここでは、その設定手順について説明します。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 6.18 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

- 「図 6.156. 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 6.156 minicom の設定の起動

- 「図 6.157. 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 6.157 minicom の設定

- 「図 6.158. 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 6.158 minicom のシリアルポートの設定

4. Serial Device に使用するデバイスファイル名として /dev/ttyUSB0 を入力して Enter キーを押してください。



デバイスファイル名の確認方法

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。

その場合は以下の方法でデバイスファイル名を確認してください。

Linux で PC と Armadillo 側のシリアルポートを接続した場合、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 6.159 例. シリアルポート接続時のログ

上記の例では Armadillo 側のシリアルポートが 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 6.160. minicom のシリアルポートのパラメータの設定」が表示されます。

```

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

図 6.160 minicom のシリアルポートのパラメータの設定

8. 「図 6.160. 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 6.161. minicom シリアルポートの設定値」を参照してください。

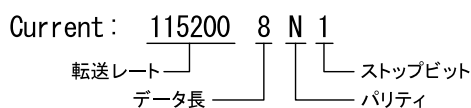



図 6.161 minicom シリアルポートの設定値

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 6.157. minicom の設定」に戻ってください。
13. 「図 6.157. 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。



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。

6.26. vi エディタを使用する

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。

6.26.1. 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 6.162 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。

6.26.2. 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 6.19. 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 6.19 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

「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 6.163. 入力モードに移行するコマンドの説明」に示します。

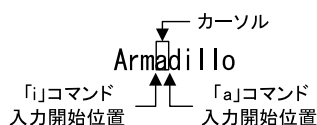




図 6.163 入力モードに移行するコマンドの説明

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。



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、「6.26.4. 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。

6.26.3. 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 6.20. カーソルの移動コマンド」に示すコマンドを入力することでもカーソルを移動することができます。

表 6.20 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

6.26.4. 文字の削除

文字を削除する場合は、コマンドモードで「表 6.21. 文字の削除コマンド」に示すコマンドを入力します。

表 6.21 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 6.164. 文字を削除するコマンドの説明」に示します。

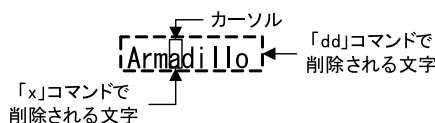


図 6.164 文字を削除するコマンドの説明

6.26.5. 保存と終了

ファイルの保存、終了を行うコマンドを「表 6.22. 保存・終了コマンド」に示します。

表 6.22 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。

6.27. eFuse を変更する

Armadillo-640 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-640 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-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。

MAC アドレスは Armadillo-640 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。

6.27.1. ブートモードとジャンパーピン

6.27.1.1. ブートモードと JP2

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。 `BOOT_MODE0` は GND になっているので必ず 0 になります。 `BOOT_MODE1` は JP2 に接続されています。 JP2 がショートされていると `BOOT_MODE1` は 1 になり **Internal Boot** モードになります。 開放されていると 0 となり、 **Boot From Fuses** というモードになります。

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-640 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。 つまり eFuse の設定がどうなっていると、 GPIO のピンでブートデバイスを決めることができます。

Boot From Fuses モードでは、単純に言えば GPIO による override が禁止され eFuse に書き込まれた状態でしかブートしません。 この機能を有効にすることで、 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…)

6.27.1.2. ブートデバイスと JP1

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、 JP1 はまさにこの機能を使っています。 JP1 は `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 6.23 GPIO override と eFuse

信号名	eFuse 名	eFuse アドレス	OCOTP 名	Bank	Word
<code>LCD1_DATA05</code>	<code>BOOT_CFG1[5]</code>	<code>0x450[5]</code>	<code>OCOTP_CFG4</code>	0	5
<code>LCD1_DATA11</code>	<code>BOOT_CFG2[3]</code>	<code>0x450[11]</code>	<code>OCOTP_CFG4</code>	0	5
N/A	<code>BT_FUSE_SEL</code>	<code>0x460[4]</code>	<code>OCOTP_CFG5</code>	0	6

Armadillo-640 では microSD 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、 B00T_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-640 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-640 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 6.24 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

6.27.2. eFuse の書き換え

Armadillo-640 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。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

6.27.3. Boot From Fuses モード

6.27.3.1. BT_FUSE_SEL

Boot From Fuses を有効にするには、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。BOOT_MODE が 00、つまり JP2 がオープンで、且つこのビットが 1 であれば Boot From Fuses モードになります。BOOT_MODE が 00 でもこのビットが 0 であれば Boot From Fuses モードにはならず、SD/MMC マニファクチャリングモードやシリアルダウンロードモードになってしまいます。SD/MMC マニファクチャリングモードについては「8.12 SD/MMC manufacture mode」に、シリアルダウンロードモードについては「8.9 Serial Downloader」に記載されています。



Armadillo-640 では BOOT_MODE が 00 で、且つ BT_FUSE_SEL が 0 の場合は SD/MMC マニファクチャリングモードで eMMC から起動します。Internal Boot モードで起動する場合は、JP2 をショートしてください。

6.27.3.2. 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 6.25 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	-

名前	Bit	eFuse	値	bit 列	可変
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	Disabled	0	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、0C0TP_CFG4 にマップされており Bank 0 Word 5 です。つまり 010000000 01100000 の 16 bit (0x4060) 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-640 ができあがりしました。



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-640 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。

6.27.3.3. eFuse のロック

書き込んだ eFuse の値を変更されてしまっは、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。

6.28. 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 6.26 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 コンポ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 コンポ対応	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	

6.28.1. USB シリアル変換アダプタ(Armadillo-640 用)

6.28.1.1. 概要

FT232RL を搭載した USB-シリアル変換アダプタです。シリアルの信号レベルは 3.3V CMOS です。

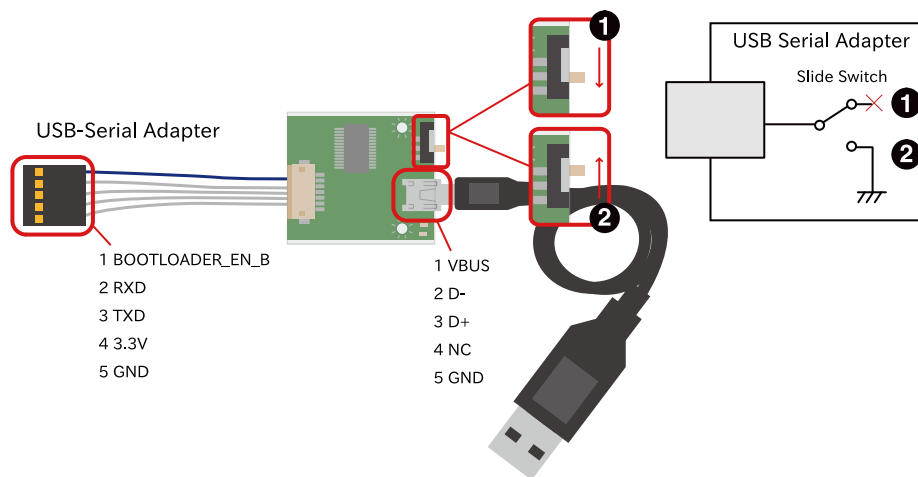


図 6.165 USB シリアル変換アダプタの配線

- ❶ オープン
- ❷ 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 6.27 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

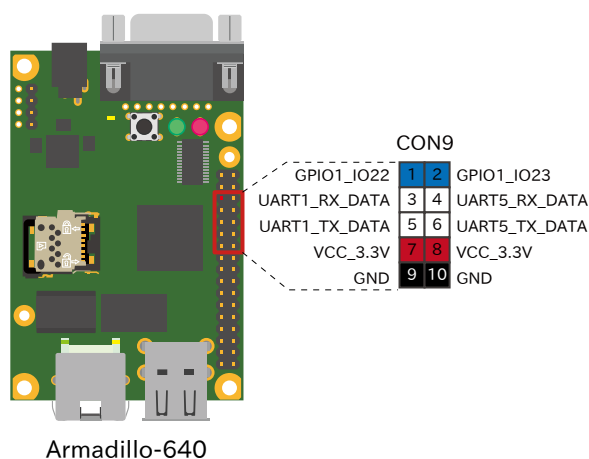



図 6.166 Armadillo-640 のシリアル信号線


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 6.28 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。

6.28.2. Armadillo-600 シリーズ オプションケース(樹脂製)

6.28.2.1. 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 6.29 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 6.30 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。

6.28.2.2. 組み立て

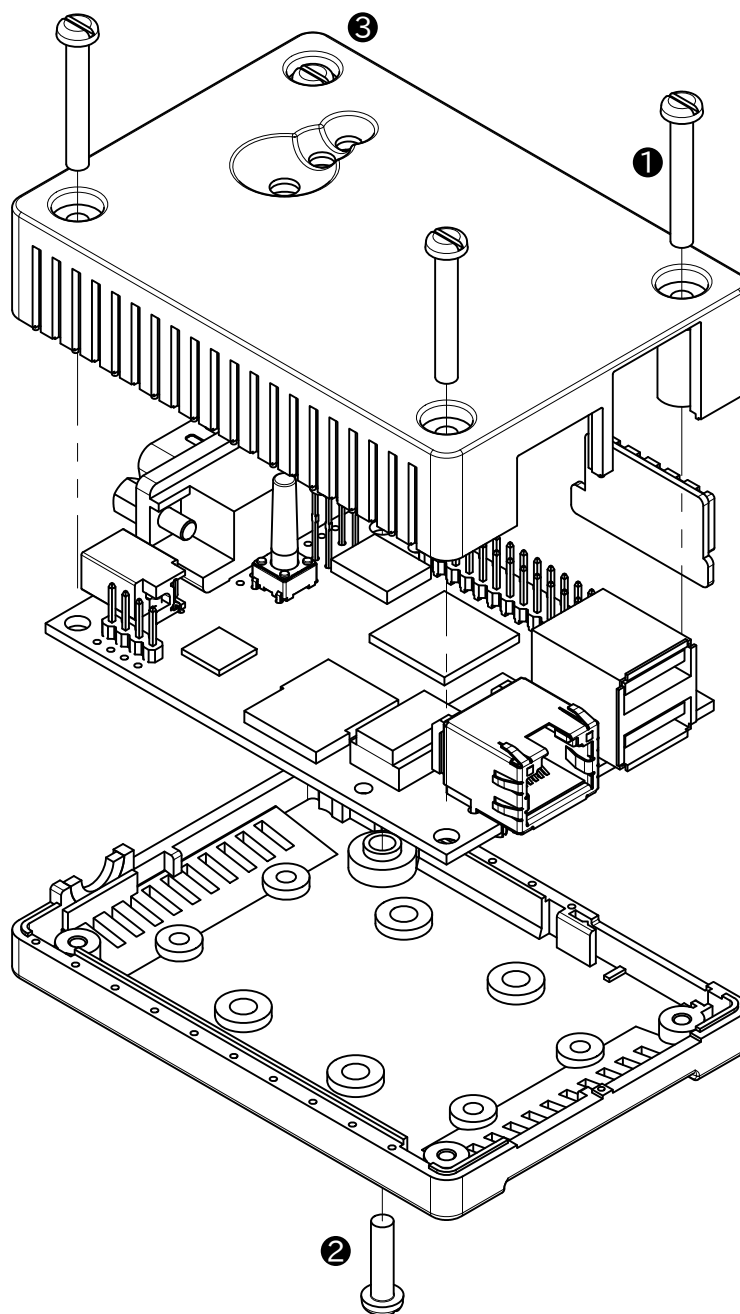




図 6.167 Armadillo-600 シリーズ オプションケース(樹脂製)の組み立て

- ① タッピングねじ(M2.6、L=20mm) x 3
- ② タッピングねじ(M3、L=12mm) x 1
- ③ 飾りねじ x 1

ねじ締めトルクは 30.0cN・m です。

 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。

6.28.2.3. 形状図

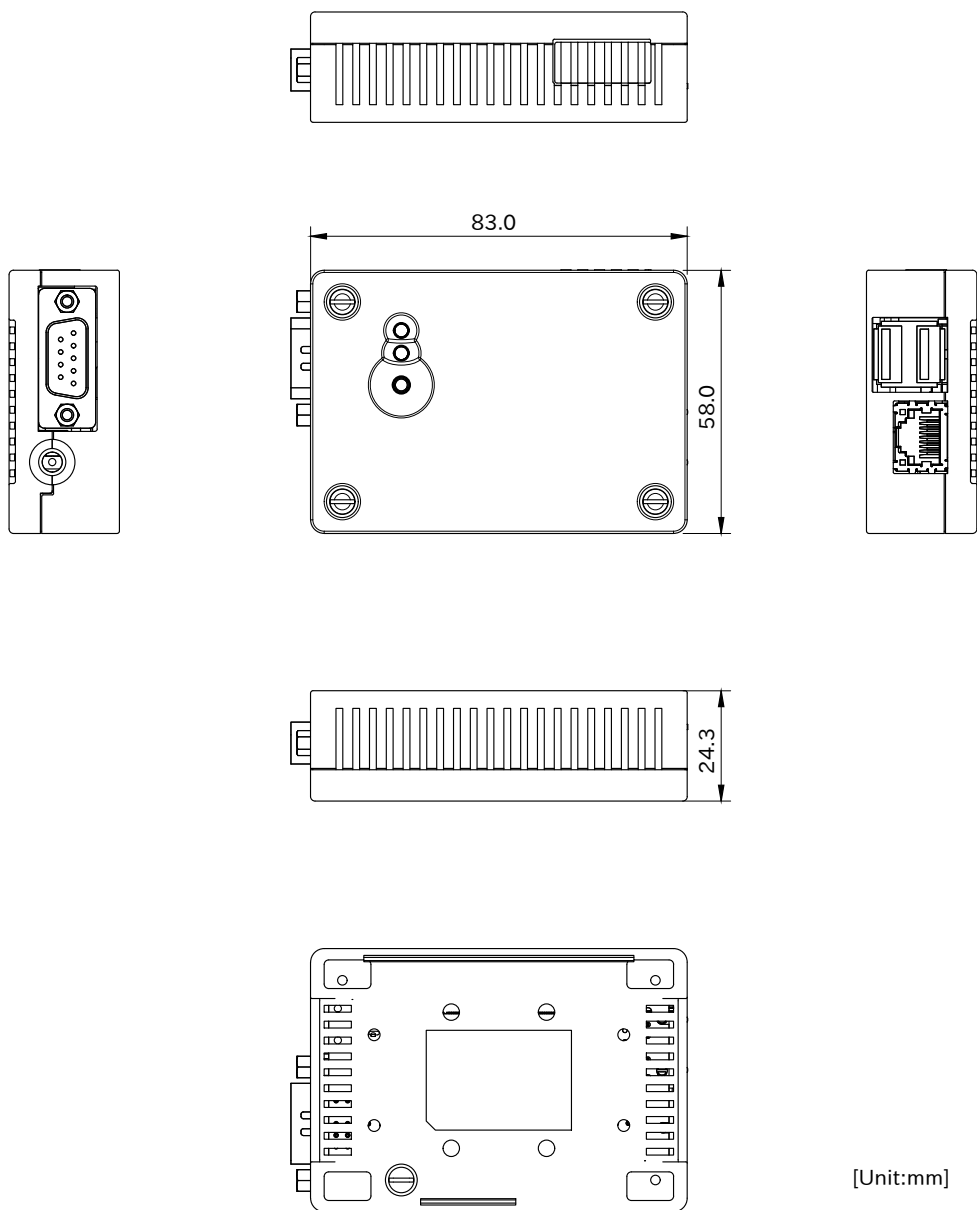



図 6.168 樹脂ケース形状図

6.28.3. Armadillo-600 シリーズ オプションケース(金属製)

6.28.3.1. 概要

Armadillo-640 用のアルミ製小型ケースです。Armadillo-640 の基板を取めた状態で、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。



金属ケースはオプション品として別売りしています。

表 6.31 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

6.28.3.2. 組み立て

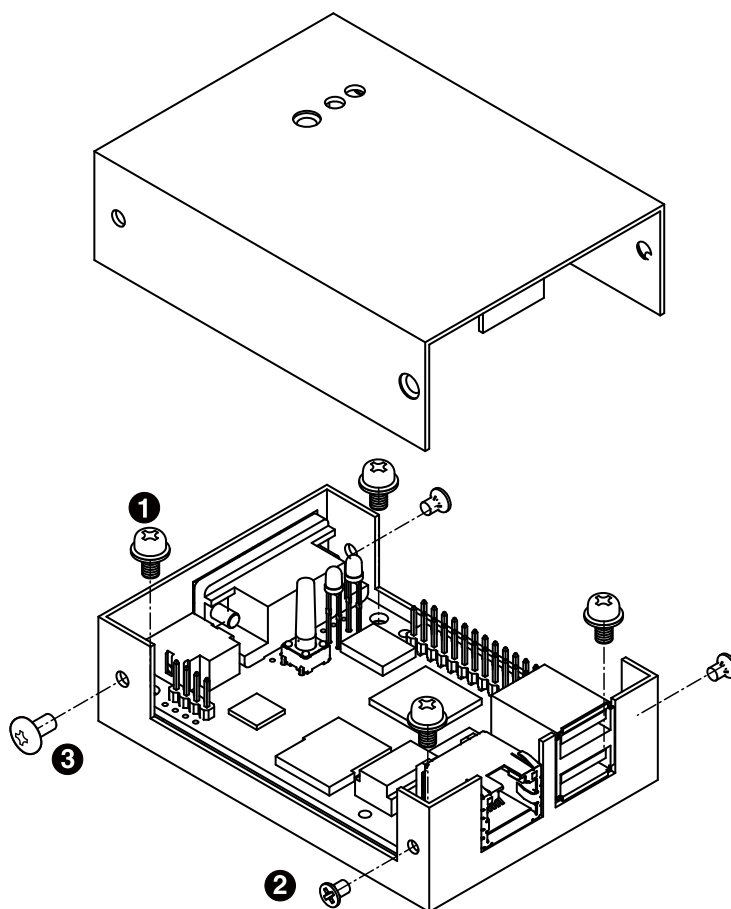


図 6.169 Armadillo-600 シリーズ オプションケース(金属製)の組み立て

- ① なべ小ネジ、スプリングワッシャ付き (M3、L=5mm) x 4

ねじ締めトルクは 31.5cN・m です。

- ② 皿ネジ (M2.6、L=4mm) x 3

ねじ締めトルクは 10.0cN・m です。

- ③ トラス小ネジ (M3、L=5mm) x 1

ねじ締めトルクは 10.0cN・m です。

6.28.3.3. 形状図

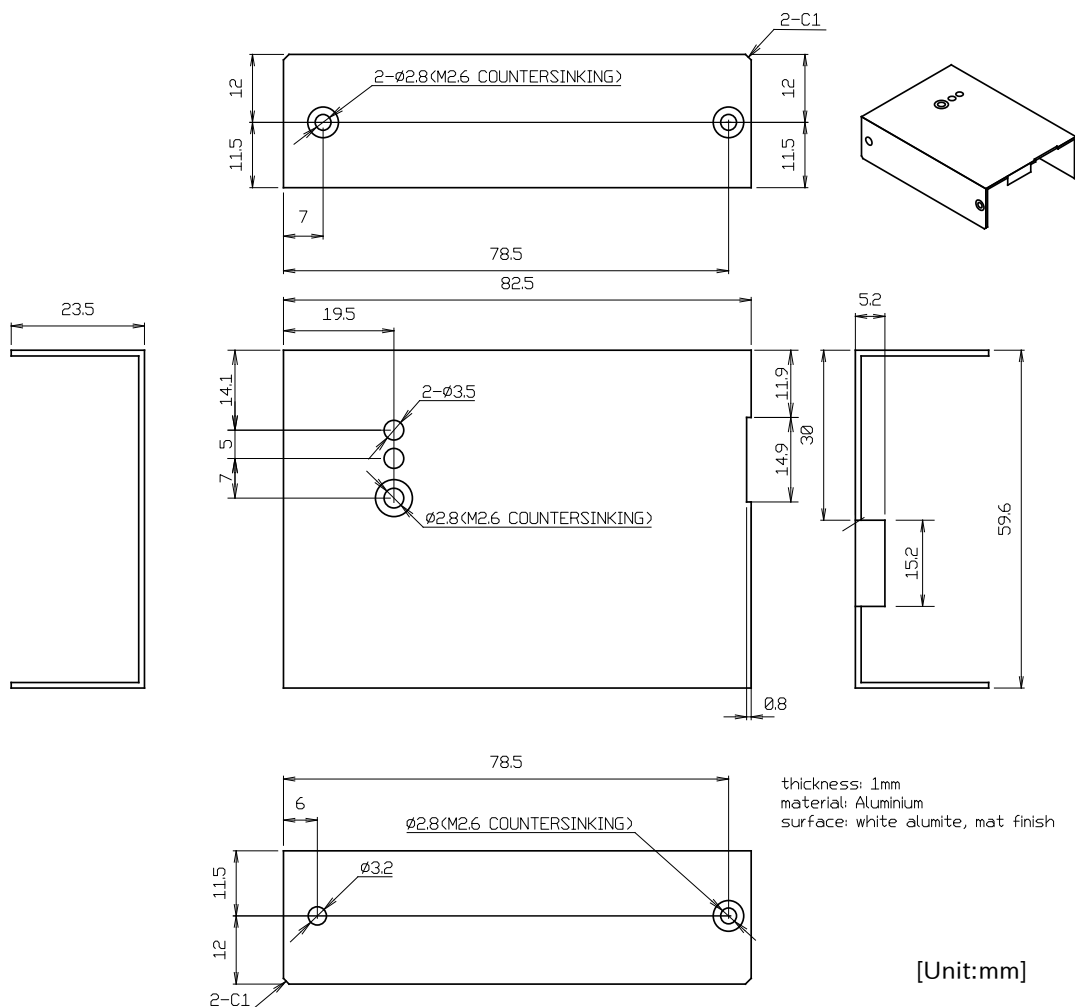


図 6.170 金属ケース(上板)寸法図

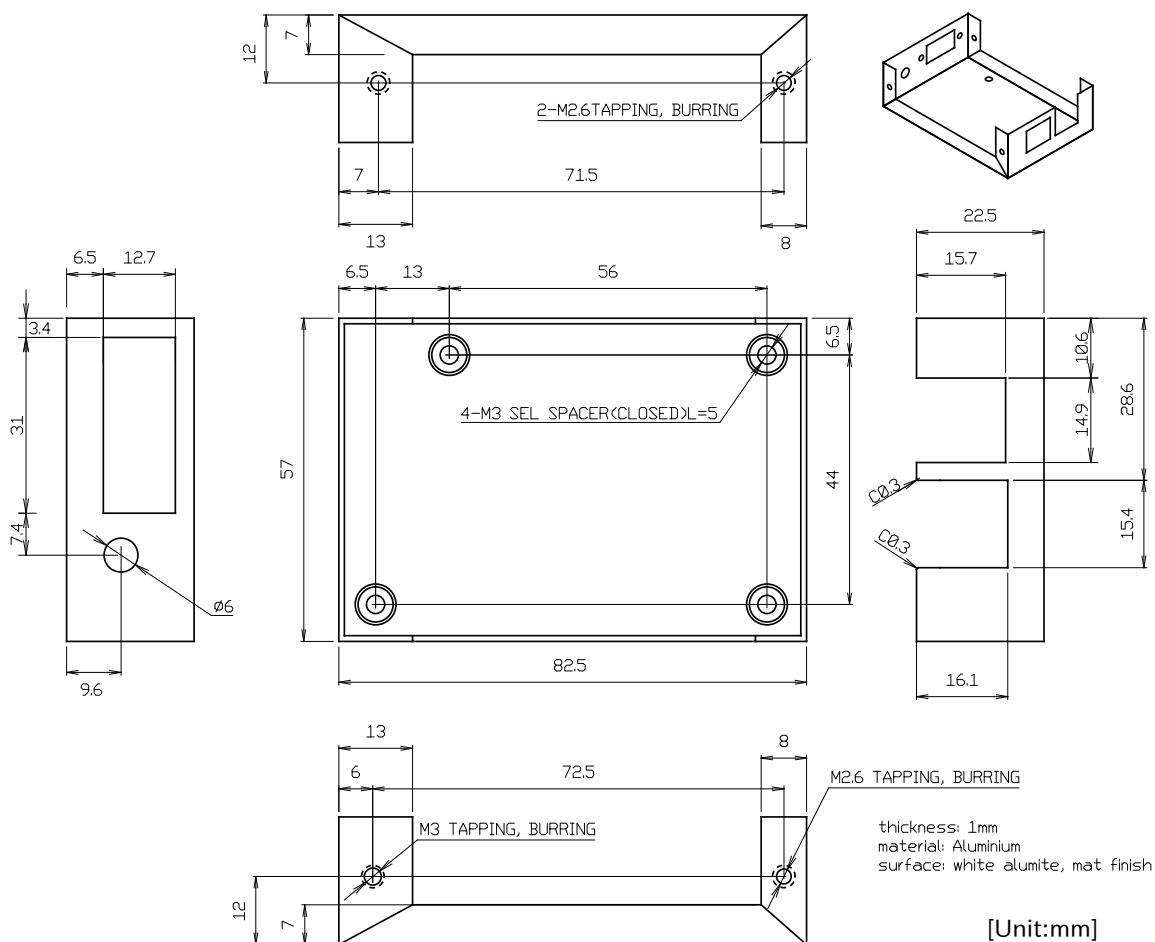


図 6.171 金属ケース(下板)寸法図

6.28.4. LCD オプションセット(7 インチタッチパネル WVGA 液晶)

6.28.4.1. 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、「3.6.15. LCD を使用する」を参照してください。

表 6.32 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 6.33 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 ² (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 パラレル(18bit/24bit) ^[a]
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

^[a]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。

6.28.4.2. 組み立て

「3.6.15. LCD を使用する」を参照してください。

6.28.5. Armadillo-600 シリーズ RTC オプションモジュール

6.28.5.1. 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 6.34 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	

6.28.5.2. 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 6.35 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)を組み合わせる場合、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。

6.28.5.3. 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。

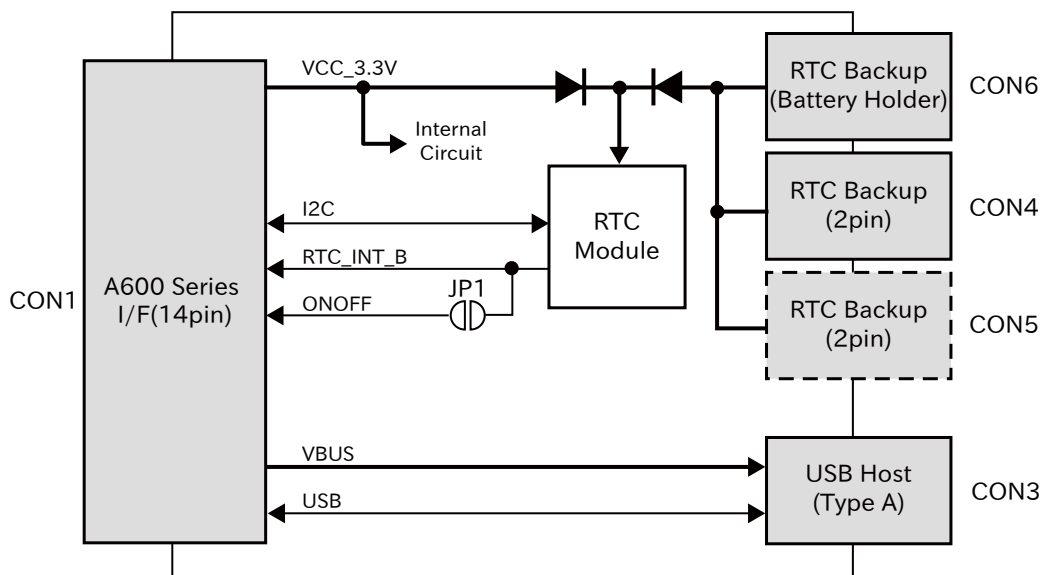


図 6.172 RTC オプションモジュールのブロック図

6.28.5.4. インターフェース一覧

RTC オプションモジュールのインターフェースについて説明します。

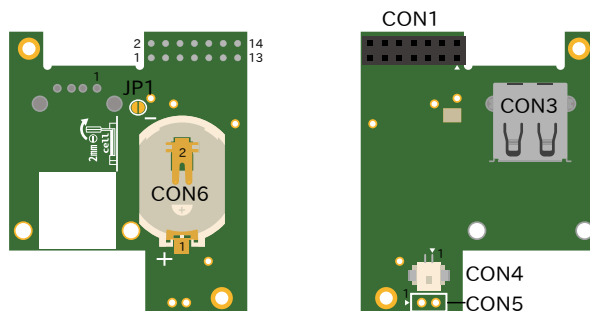


図 6.173 RTC オプションモジュールのインターフェース

表 6.36 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。

6.28.5.5. 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 6.37 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。

6.28.5.6. 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、「3.6.4. USB デバイスを使用する」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。

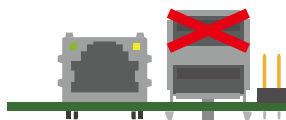


図 6.174 Armadillo-640 CON5 上段と RTC オプションモジュール CON3 は排他利用



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 6.38 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)

6.28.5.7. 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 6.39 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 ^[a]	-	-
CON6	CR2016	2.5 年

^[a]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 6.40 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、「6.28.5.10. 電池の取り付け、取り外し」をご確認ください。

6.28.5.8. 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。

6.28.5.9. 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 6.175. 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。

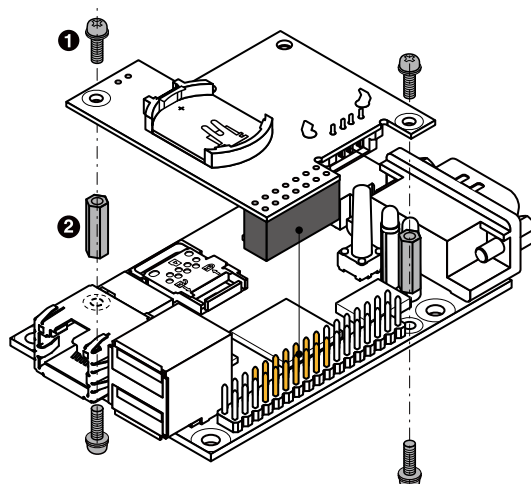


図 6.175 RTC オプションモジュールの組み立て

- 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

6.28.5.10. 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

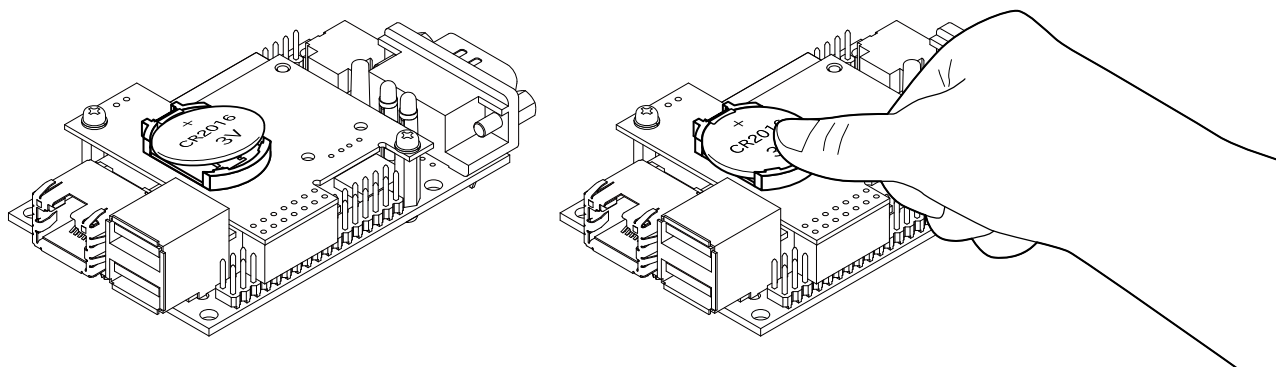


図 6.176 電池ホルダに電池を取り付ける

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

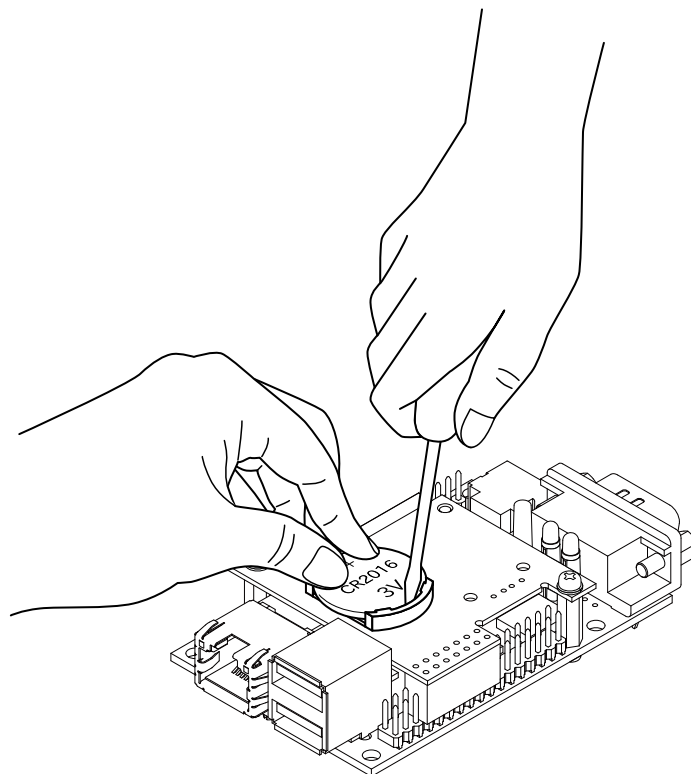


図 6.177 電池ホルダから電池を取り外す

6.28.5.11. 形状図

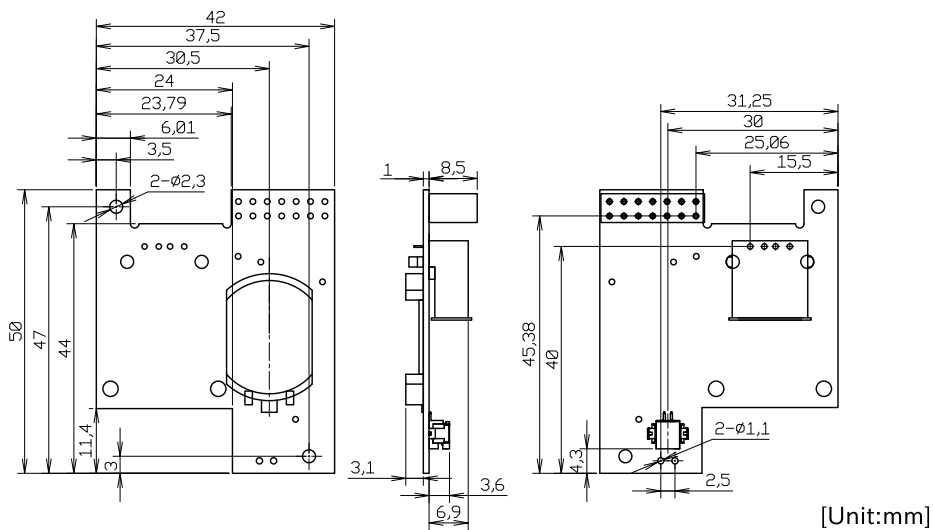


図 6.178 RTC オプションモジュール形状

6.28.6. Armadillo-600 シリーズ WLAN コンボ、BT/TH オプションモジュール

6.28.6.1. 概要

Ezurio 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 6.41 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「3.6.5. WLAN を使用する」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「3.6.6. BT デバイスを使用する」または、「3.6.7. Thread デバイスを扱う」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 6.42. 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 6.42 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 6.43 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、「3.6.4. USB デバイスを使用する」 および回路図をご確認ください。

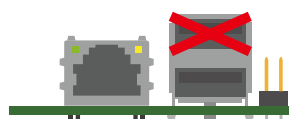


図 6.179 WLAN コンボ、BT/TH オプションモジュール接続時に Armadillo-640 CON5 上段は使用不可



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、「6.28.1. 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、「6.28.6.11. 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。

6.28.6.2. 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 6.44 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	Ezurio
	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) ^[a]
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) ^[b]	
外形サイズ	41 x 49.7 mm	

^[a]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。

^[b]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。



Ezurio 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。

6.28.6.3. 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。

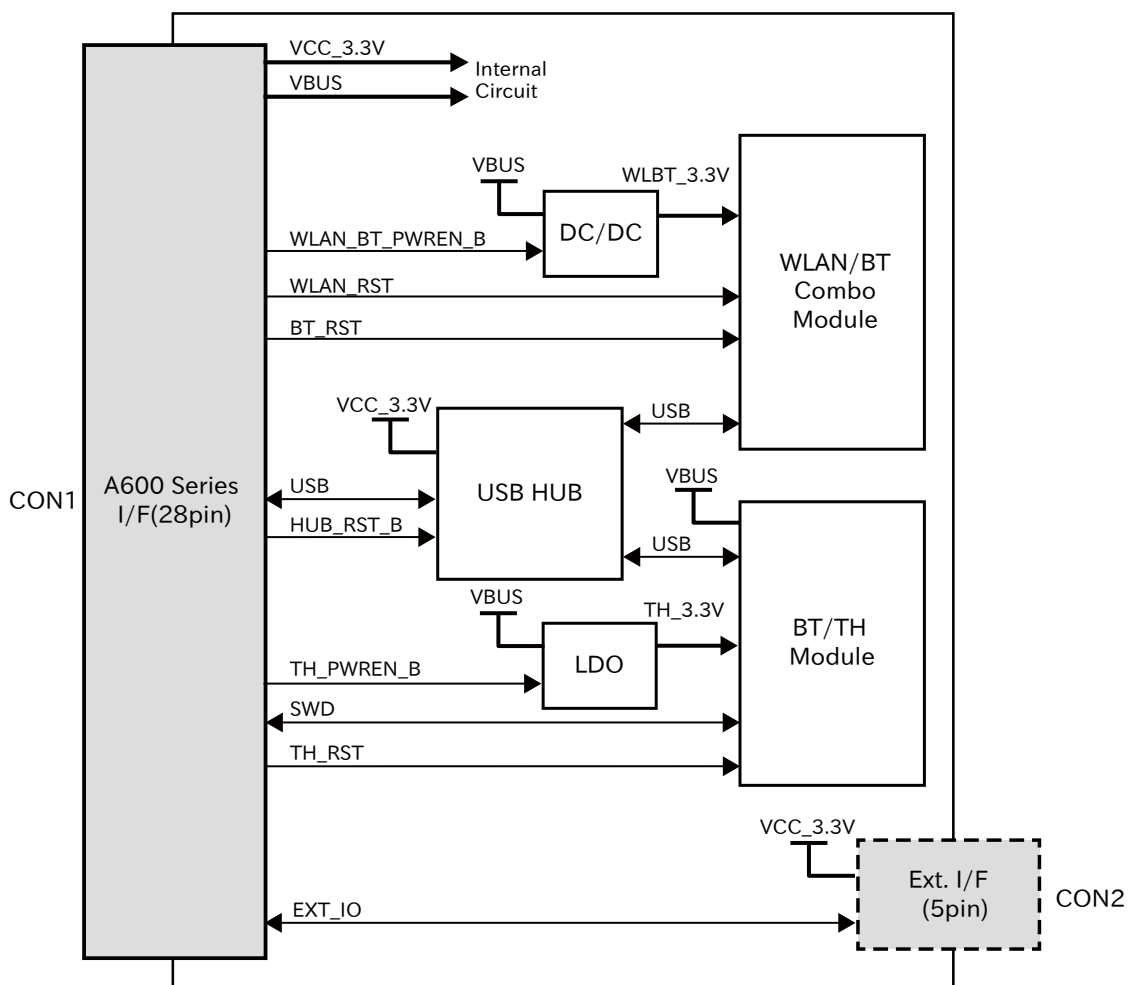


図 6.180 WLAN コンボ、BT/TH オプションモジュールのブロック図

6.28.6.4. インターフェース一覧

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について説明します。

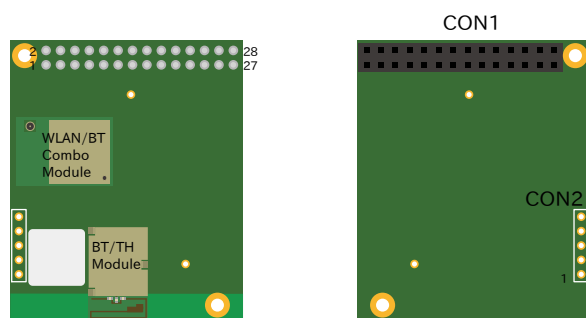


図 6.181 WLAN コンボ、BT/TH オプションモジュールのインターフェース

表 6.45 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。

6.28.6.5. 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 6.46 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。
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。

6.28.6.6. 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 6.47 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 6.48. 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 等のピンヘッダを実装すると、「6.28.1. 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 6.48 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.

6.28.6.7. 形状図

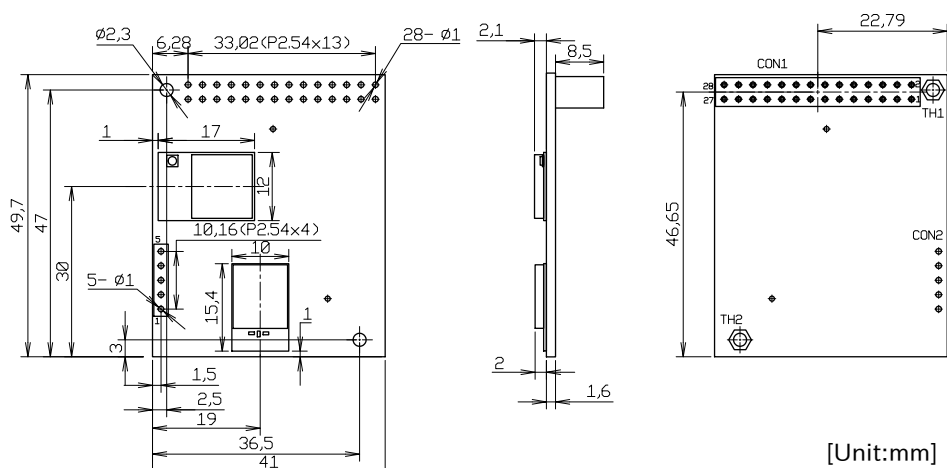


図 6.182 WLAN コンボ、BT/TH オプションモジュール形状図

6.28.6.8. 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。

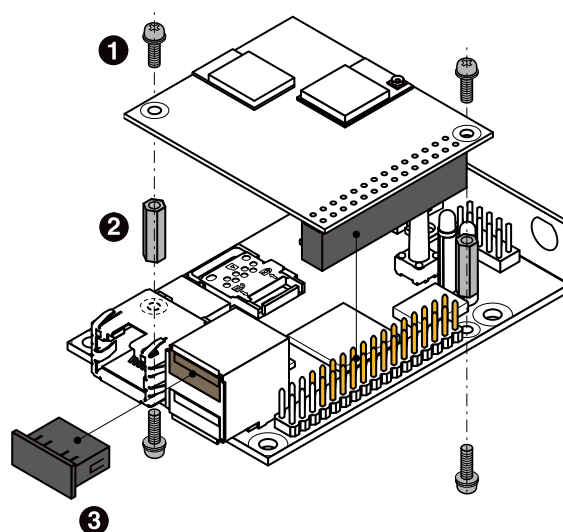


図 6.183 WLAN コンボ、BT/TH オプションモジュールの組み立て

- 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。

6.28.6.9. 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。

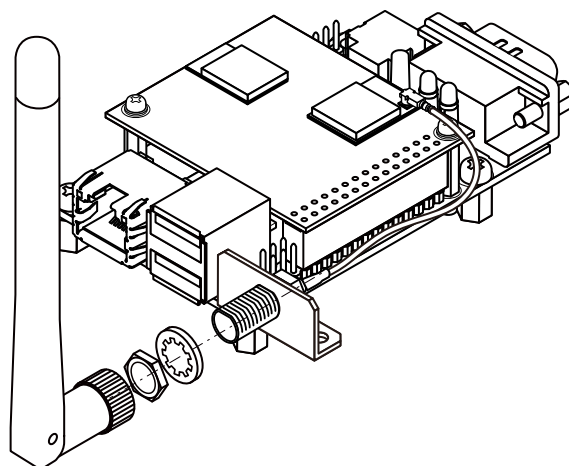


図 6.184 外付けアンテナの組み立て(OP-MNT-ANT-MET-00 使用)

CON3 に実装されている D-Sub9 ピンコネクタの替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。

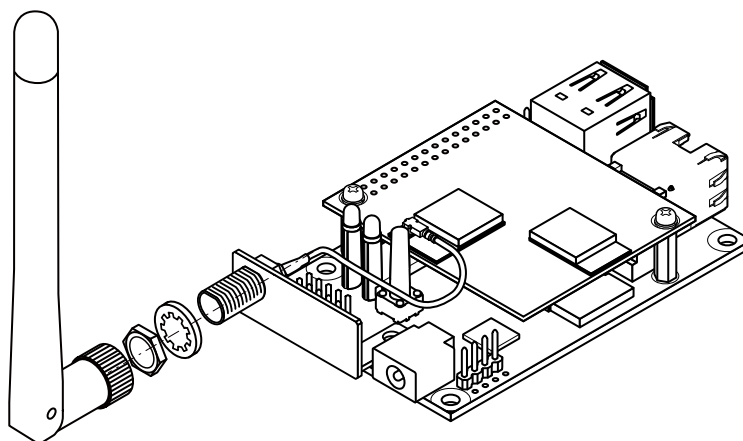


図 6.185 外付けアンテナの組み立て(オプションケース対応のアンテナ固定金具使用)



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 ピンコネクタの替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。

6.28.6.10. 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。

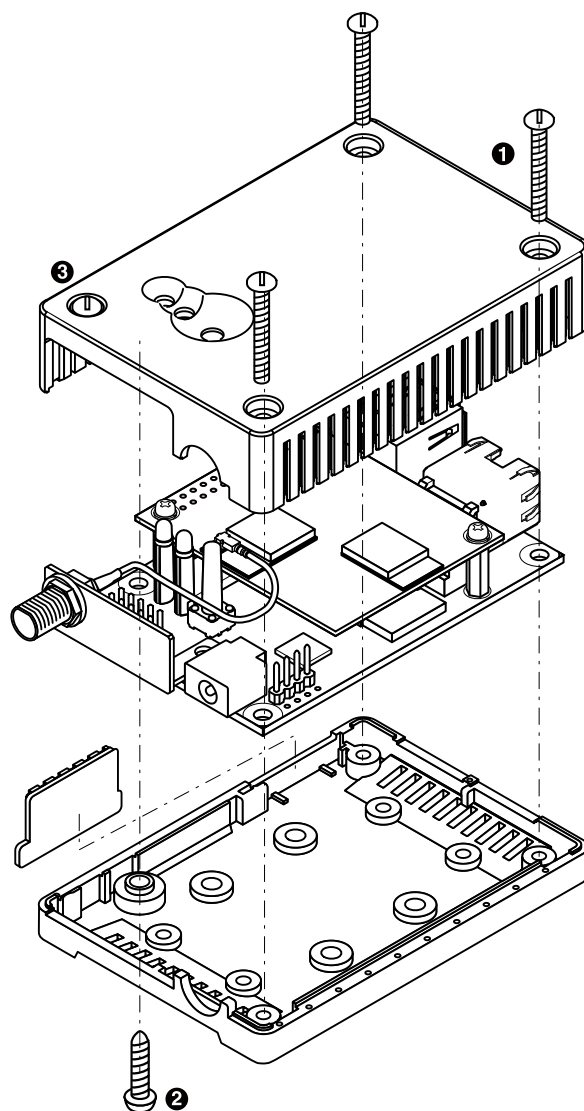


図 6.186 ケースの組み立て

- 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

ねじ締めトルクは 30.0cN・m です。

6.28.6.11. 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、「6.28.1. 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 ピンのピンヘッダを実装することで、「6.28.1. 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。

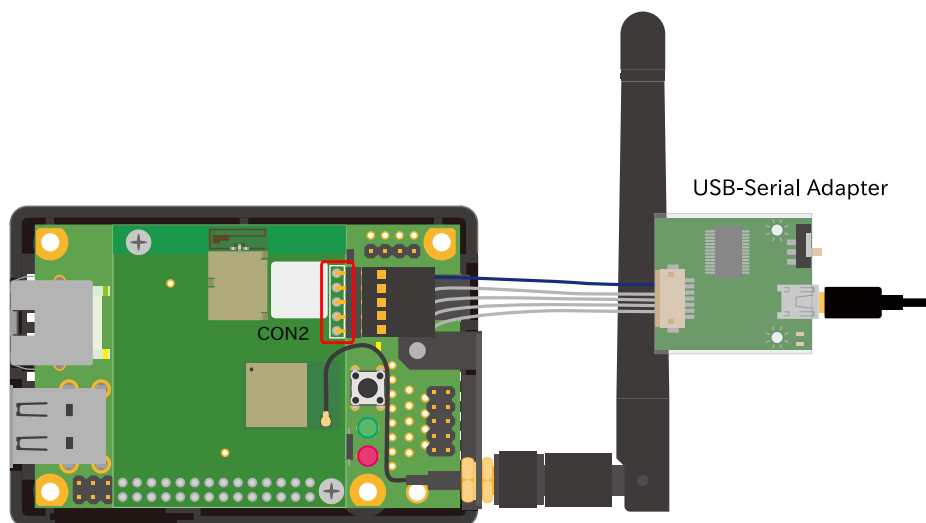


図 6.187 オプションモジュールの CON2 をシリアルコンソールとして使用する場合の接続例

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。

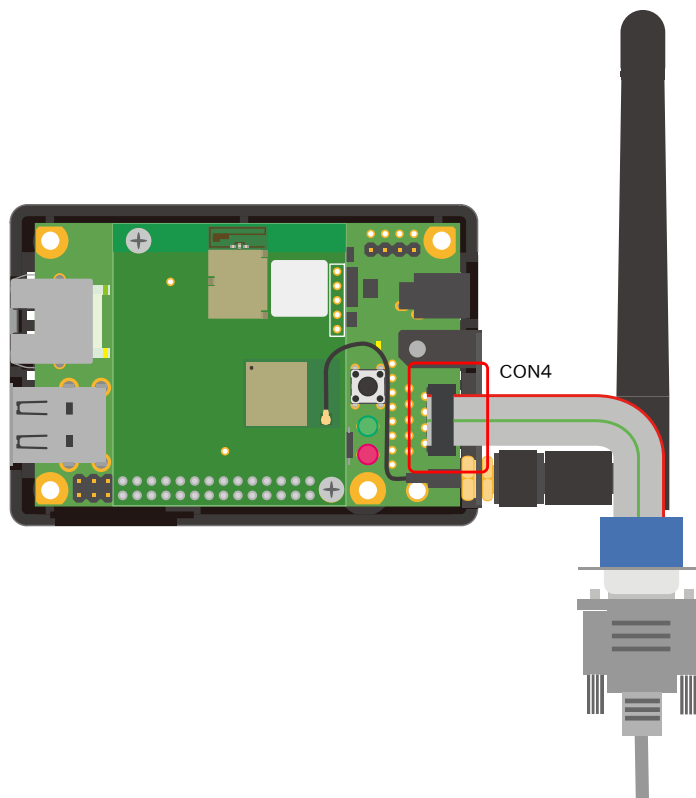


図 6.188 CON4 をシリアルコンソールとして使用する場合の接続例

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。

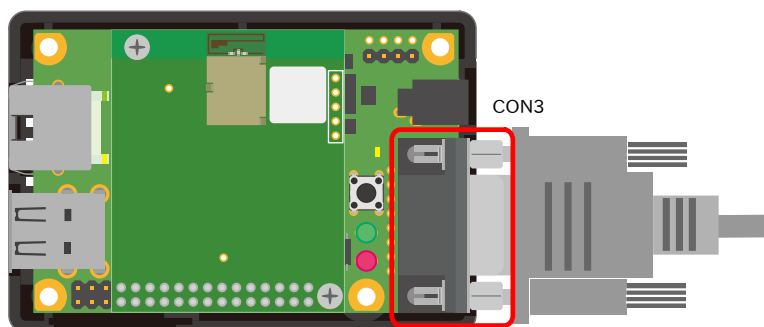


図 6.189 CON3 をシリアルコンソールとして使用する場合の接続例

6.28.7. 無線 LAN 用 外付けアンテナセット 08

6.28.7.1. 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 Ezurio 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 6.49 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。

6.28.7.2. 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。

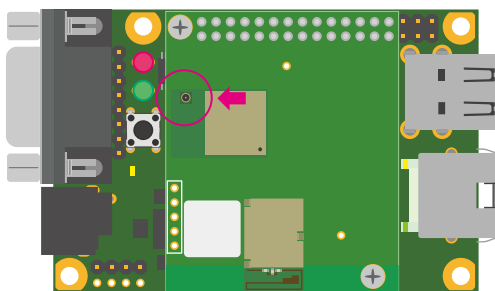


図 6.190 Sterling LWB5+上のコネクタの位置

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。

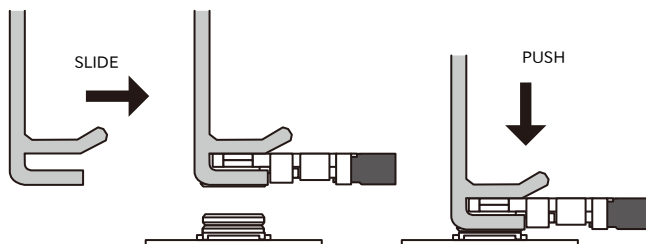


図 6.191 挿抜治具によるアンテナケーブルコネクタの嵌合



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。

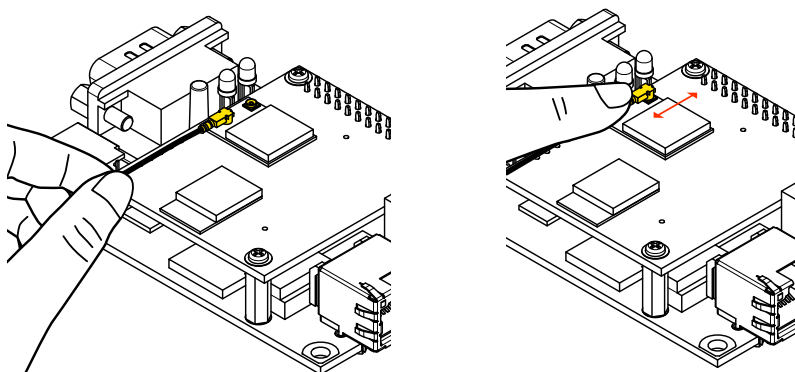


図 6.192 アンテナケーブルコネクタのセット

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。

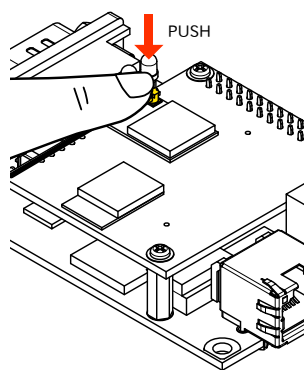


図 6.193 アンテナケーブルコネクタの嵌合

6.28.7.3. 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。

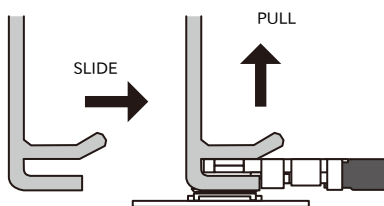


図 6.194 挿抜治具によるアンテナケーブルコネクタの抜去



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。

6.28.7.4. 形状図

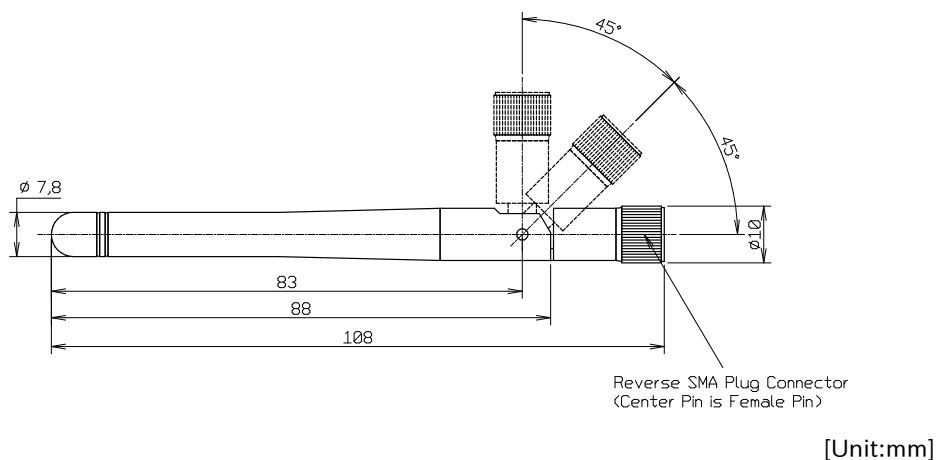


図 6.195 無線 LAN 用 外付けアンテナ 08 形状図

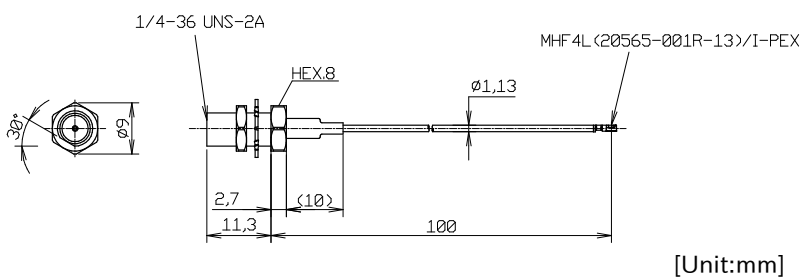


図 6.196 無線 LAN 用 外付けアンテナケーブル 08 形状図

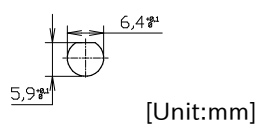


図 6.197 無線 LAN 用 外付けアンテナ 取り付け穴寸法

6.28.8. 外付けアンテナ固定金具 00

6.28.8.1. 概要

アンテナを固定する金具です。

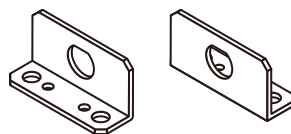


図 6.198 外付けアンテナ固定金具 00 の外観

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。

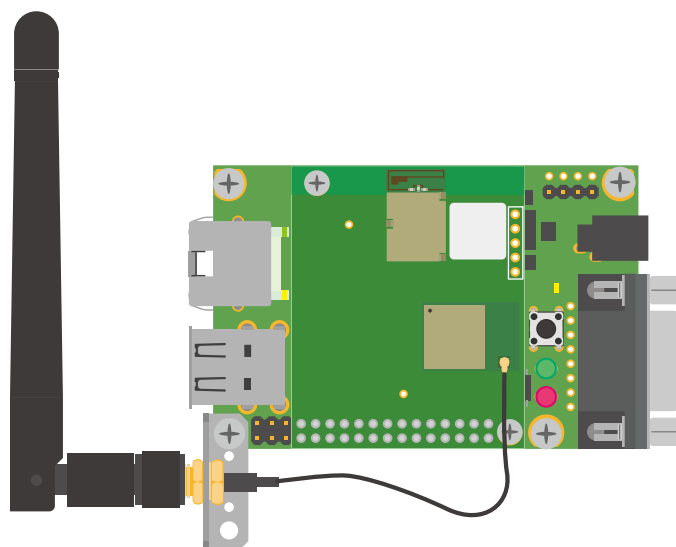


図 6.199 Armadillo-640 へのアンテナ固定金具の固定

表 6.50 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 ^[a]

^[a]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 6.51 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。

6.28.8.2. 組み立て

「図 6.200. 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。

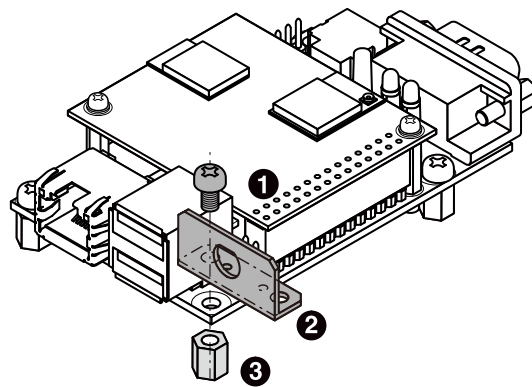


図 6.200 Armadillo-640 へのアンテナ固定金具の固定

- 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)

ねじ締めトルクは 17.5cN・m です。

アンテナは「図 6.201. 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。

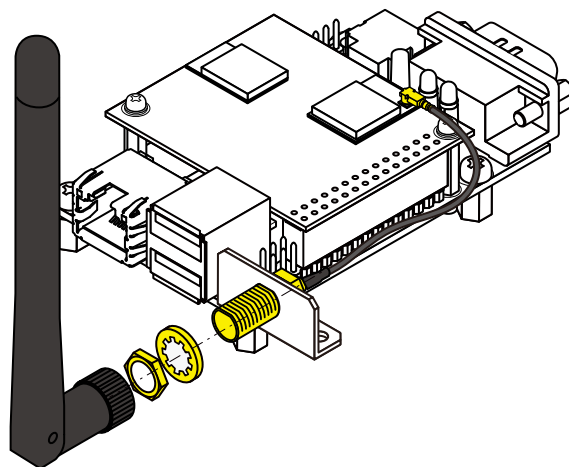


図 6.201 アンテナ固定金具へのアンテナの固定

6.28.8.3. 形状図

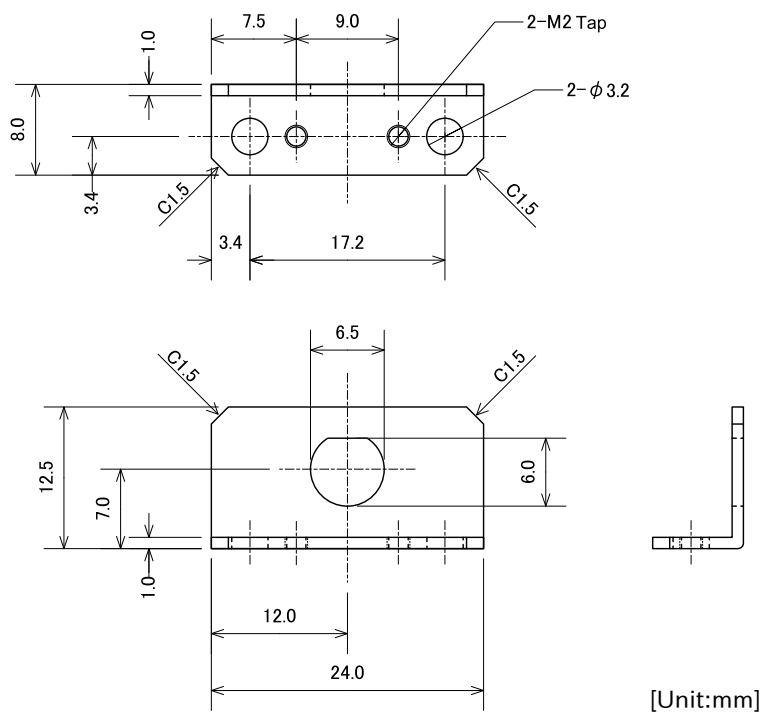


図 6.202 アンテナ固定金具 形状図

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"> ・ 初版発行
3.1.0	2023/07/26	<ul style="list-style-type: none"> ・ 「6.9.2. ABOS Web の設定機能一覧と設定手順」 に 「3.8.12. VPN 設定」 を追加 ・ 「6.9. Web UI から Armadillo をセットアップする (ABOS Web)」 に 「6.9.3. コンテナ管理」 を追加 ・ 「6.9. Web UI から Armadillo をセットアップする (ABOS Web)」 に 「6.9.4. SWU インストール」 を追加 ・ 「3.13. CUI アプリケーションの開発」 の内容を最新のソフトウェアのものに更新 ・ 「3.13. CUI アプリケーションの開発」 に ssh_known_hosts に関する注意を追加 ・ 「6.2.2.12. pod でコンテナのネットワークネームスペースを共有する」 の説明を set_infra_image を使わないものに変更 ・ 誤記修正
3.2.0	2023/08/29	<ul style="list-style-type: none"> ・ 文章全体の構成を変更 ・ 「3.6.10. I2C デバイスを使用する」 の最大転送レートを 384kbps に修正 ・ 「3.13.7.2. ssh 接続に使用する IP アドレスの設定」 に VSCode から IP アドレスを設定する方法を追加 ・ 「6.2.5. アットマークテクノが提供するイメージを使う」 に 「6.2.5.1. ABOSDE からインストールする」 を追加 ・ 「6.2.6. alpine のコンテナイメージをインストールする」 を追加 ・ 誤記修正
4.0.0	2023/09/05	<ul style="list-style-type: none"> ・ 前回の更新で章構成を大きく変更したため、メジャーバージョンを変更 ・ 「3.5.5.1. 提供している DT overlay」 にある WLAN コンボ/BT/TH オプションモジュール用の dtbo ファイルのファイル名を修正
4.1.0	2023/10/30	<ul style="list-style-type: none"> ・ 一部の章構成を変更 ・ 本文中にあるマルチプレクス表のダウンロードページのリンク先を修正 ・ 「3.1.5.1. initial_setup.swu の作成」 の説明文を修正 ・ 「3.5.5.1. 提供している DT overlay」 に armadillo-600-button-enter.dtbo を追加 ・ 「3.5.5.1. 提供している DT overlay」 に at-dtweb で作成した dtbo を使用する場合の注意事項を追加 ・ 「3.6.4. USB デバイスを使用する」 に記載しているコンテナの作成例を、add_hotplugs を使用した例に修正 ・ 「3.13. CUI アプリケーションの開発」 内にある 「3.13.3.2. ディレクトリ構成」 の説明文を修正 ・ 「6.2.3. コンテナとコンテナに関連するデータを削除する」 に VSCode から削除する方法を追加 ・ 「6.14. ボタンやキーを扱う」 に SW1 に関する説明を追加 ・ 「3.10. Network Time Protocol (NTP, ネットワーク・タイム・プロトコル) の設定」 に記載している chrony の設定ファイルに関する説明を修正 ・ 誤記および分かりにくい表記の修正
4.2.0	2023/11/28	<ul style="list-style-type: none"> ・ 「図 4.19. インストールログを保存する」 内の umount コマンド例から -R オプションを削除 ・ 「6.2.2.14. コンテナからのコンテナ管理」 の説明文を修正

		<ul style="list-style-type: none"> ・「6.2.4.17. 自動起動の無効化」の説明文を修正 ・「表 6.13. u-boot の主要な環境変数」に bootdelay=-3 の説明を追加 ・「6.22.3. ビルドしたルートファイルシステムの SBOM を作成する」を追加 ・誤記および分かりにくい表記の修正
4.3.0	2023/12/26	<ul style="list-style-type: none"> ・「表 2.2. 仕様」にセキュアエレメントの項目追加 ・「図 2.7. Armadillo-640 ブロック図」に Secure Element 追加 ・「図 2.6. インターフェースレイアウト」の DC ジャック部分変更 ・「図 3.49. Armadillo-640 の接続例」の DC ジャック部分変更 ・「図 3.40. CON9-USB シリアル変換アダプタ接続図」の DC ジャック部分変更 ・「図 3.12. ジャンパピンの位置」の DC ジャック部分変更 ・「図 3.70. Armadillo-640 のインターフェース」の DC ジャック部分変更 ・「表 3.11. Armadillo-640 インターフェース一覧」の CON12 の型番を HEC3600-016110 に変更 ・「図 3.52. 基板形状および固定穴寸法」の DC ジャック部分変更 ・「図 3.53. コネクタ中心寸法(A 面側)」の図名変更及び、DC ジャック部分変更 ・「図 3.54. コネクタ中心寸法(B 面側)」を追加 ・「図 3.55. コネクタ穴寸法」の DC ジャック部分変更 ・「図 6.166. Armadillo-640 のシリアル信号線」の DC ジャック部分変更 ・「図 3.115. LCD の接続方法」の DC ジャック部分変更 ・「3.3.5. インストールディスクについて」に SBOM の自動生成に関する説明を追記 ・「3.8.1. ABOS Web とは」に ABOS をバージョンアップした際の注意点を追記 ・「5.2.5. 個体識別情報の取得」内の個体識別情報取得方法を修正 ・「6.2.4.6. 個体識別情報の環境変数の追加」を追加 ・「6.9.6. アプリケーション向けのインターフェース (Rest API)」を追加 ・「6.21.3. Alpine Linux ルートファイルシステムをビルドする」に SBOM の自動生成に関する説明を追記 ・「図 6.69. 180 秒後に RTC(NR3225SA)で起床する」を追加 ・誤記および分かりにくい表記の修正
4.4.0	2024/01/29	<ul style="list-style-type: none"> ・「3.8.2. ABOS Web へのアクセス」にアクセス可能なネットワークに関する説明を追記 ・「3.13.3.2. ディレクトリ構成」に requirements.txt に関する説明を追記 ・「3.14. C 言語によるアプリケーションの開発」を追加 ・「6.9.6.1. Rest API へのアクセス権の管理」にアクセス可能なネットワークに関する説明を追記 ・誤記および分かりにくい表記の修正
4.5.0	2024/02/02	<ul style="list-style-type: none"> ・「4.4.4. 開発したシステムをインストールディスクにする」に「4.4.5. VS Code を使用して生成する」を追記
4.6.0	2024/02/27	<ul style="list-style-type: none"> ・Armadillo Twin に関する情報を追加 ・「3.3.3.6. SWU イメージのインストール」に ABOS Web を使用したインストールを追加 ・「3.12. ABOSDE によるアプリケーションの開発」を追加 ・「3.13. CUI アプリケーションの開発」に「3.13.4. コンテナのディストリビューション」を追加

		<ul style="list-style-type: none"> ・「3.14. C 言語によるアプリケーションの開発」に「3.14.4. コンテナのディストリビューション」を追加 ・「5.3. ABOSDE で開発したアプリケーションをアップデートする」を追加 ・「6.10. ABOSDE から ABOS Web の機能を使用する」を追加
4.7.0	2024/03/26	<ul style="list-style-type: none"> ・本文中に記載のある WLAN+BT コンボモジュールの製造メーカーを「Laird Connectivity」から「Ezurio」へ変更 ・「3.5. Device Tree をカスタマイズする」、「6.21. Armadillo のソフトウェアをビルドする」でデフォルトコンフィギュレーションを適用する場合でも CROSS_COMPILE 変数を設定するように修正 ・「3.8.3. ABOS Web のパスワード登録」にパスワード変更に関する補足説明を追加 ・「3.9. ABOS Web をカスタマイズする」を追加 ・「6.9.5. 時刻設定」を追加 ・「6.9.6.13. Rest API : 時刻の設定」を追加
4.8.0	2024/04/04	<ul style="list-style-type: none"> ・「3.13. CUI アプリケーションの開発」に「3.13.6. コンテナ内のファイル一覧表示」を追加 ・「3.14. C 言語によるアプリケーションの開発」に「3.14.5. コンテナ内のファイル一覧表示」を追加
4.9.0	2024/04/23	<ul style="list-style-type: none"> ・「hawkBit サーバーから複数の Armadillo をアップデートする」の章を削除し「3.3.3.1. SWUpdate とは」に hawkBit の利用に関する注意を追加 ・「3.3.5.1. 初期化インストールディスクの作成」にシリアルコンソールに関する説明を追加 ・「4.4.4. 開発したシステムをインストールディスクにする」にインストール実行時に再生成するファイルに関する説明を追加 ・「6.9.6.15. Rest API : ABOS Web 制御」を追加 ・誤記および分かりにくい表記の修正
4.10.0	2024/05/29	<ul style="list-style-type: none"> ・「3.7.5. コンテナに Armadillo の情報を渡す方法」を追加 ・「図 4.15. 開発完了後のシステムをインストールディスクイメージにする」の内容を最新に更新 ・「6.3.1. swupdate で可能なアップデート」を追加 ・「6.3.2. コンテナのアップデート、ユーザーデータディレクトリや Armadillo Base OS の差分アップデート」を追加 ・「6.3.3. Armadillo Base OS の一括アップデート」を追加 ・「6.3.4. ブートローダーのアップデート」を追加 ・「6.4.1. インストールバージョンを指定する」にバージョンの指定方法に関する説明を追加 ・「6.4.2. Armadillo へファイルを転送する」の swdesc_tar、swdesc_files コマンドに --preserve-attributes オプションを追加 ・「6.4.5. 動作中の環境でのコマンドの実行」に nochroot についての記述を追加 ・誤記修正
4.11.0	2024/06/26	<ul style="list-style-type: none"> ・「3.3.3.5. ロールバック(リカバリー)」の説明内容を追加 ・「3.3.3.5. ロールバック(リカバリー)」に --allow-downgrade オプションの説明追加 ・「3.6.3. UART を使用する」の最大ボーレートを 4Mbps に修正 ・「3.9. ABOS Web をカスタマイズする」に対応バージョンを追記 ・「3.10. Network Time Protocol (NTP, ネットワーク・タイム・プロトコル) の設定」の説明を改善

		<ul style="list-style-type: none"> ・「4.4.2. Armadillo Base OS の更新」 に abos-ctrl update コマンドでアップデートする手順を追記 ・「6.2.4. コンテナ起動設定ファイルを作成する」の podman_start コマンドの幾つかのオプションに関する説明を追加 ・「6.2.5.1. ABOSDE からインストールする」に「インストール用のプロジェクトを作成する」手順を追加 ・「6.17. ロールバック状態を確認する」の説明内容を追加 ・「表 6.12. rollback-status 追加情報の出力と意味」に --allow-downgrade オプションの説明追加 ・「表 6.13. u-boot の主要な環境変数」の upgrade_available 変数の説明を追加 ・「6.24.5. /var/log/ 配下のログに関して」追加 ・ABOSDE の説明画像を最新に更新 ・誤記修正
4.12.0	2024/07/24	<ul style="list-style-type: none"> ・「3.8.4. ABOS Web のパスワード変更」を追加 ・「3.13.3.5. Python アプリケーションに BLE パッケージをインストールする」を追加 ・「6.9.6.2. Rest API 使用例の前提条件」にコンテナから使用する際の URL に関する TIP を追加 ・「6.9.6.7. Rest API : ネットワーク設定」にネットワーク接続・切断を追加 ・「6.9.6.8. Rest API : WLAN」を追加 ・「6.9.6.9. Rest API : WWAN の設定」を追加 ・「6.9.6.10. Rest API : DHCP の設定」を追加 ・「6.9.6.11. Rest API : NAT の設定」を追加 ・「6.9.6.17. Rest API : カスタムスクリプトの実行」を追加 ・誤記修正
4.13.0	2024/08/28	<ul style="list-style-type: none"> ・「3.1. 開発の準備」の構成を開発の流れが体験できるように変更 ・「図 3.14. initial_setup.swu 初回生成時の各種設定」に initial_setup のパスワード制限を追記 ・「6.19.1. u-boot の環境変数の変更を制限する」を追加 ・「6.22. SBOM の提供」を追加 ・「図 6.167. Armadillo-600 シリーズ オプションケース(樹脂製)の組み立て」にねじ締めトルクを追記 ・「図 6.169. Armadillo-600 シリーズ オプションケース(金属製)の組み立て」にねじ締めトルクを追記 ・「図 6.186. ケースの組み立て」にねじ締めトルクを追記 ・「図 6.200. Armadillo-640 へのアンテナ固定金具の固定」にねじ締めトルクを追記 ・誤記・わかりにくい文章の修正
4.14.0	2024/09/25	<ul style="list-style-type: none"> ・「3.1.2. 仮想環境のセットアップ」を Oracle VM VirtulaBox を使用したものに変更 ・「3.3.3.3. 機密性、完全性、真正性の担保」を追加 ・「3.7.2.1. ユーザーデータの保存場所」を追加 ・「3.7.2.2. アプリケーション設定情報の保存場所」を追加 ・「3.7.6. Armadillo Base OS のデフォルトで開放しているポート」を追加 ・「3.13.8. SBOM 生成に関する設定」を追加 ・「3.14.8. SBOM 生成に関する設定」を追加 ・「3.15. SBOM 生成に関わる設定を行う」を追加 ・「3.17. ユーザー設定とユーザーデータを一括削除する」を追加 ・「5.4.1.1. SBOM の生成」を追加 ・「6.2.5.1. ABOSDE からインストールする」に SBOM 生成に関する説明を追加

		<ul style="list-style-type: none"> ・ 「6.2.6.1. SBOM 生成に関わる設定を行う」 を追加 ・ 「6.9.6.16. Rest API : ユーザー設定とユーザーデータの管理」 を追加 ・ 「6.9.8. ユーザー設定とユーザーデータの削除」 を追加 ・ 「6.9.9. ABOS Web を停止する」 を追加 ・ 「6.9.10. ABOS Web を起動する」 を追加 ・ 「6.12.6. ファイアウォールの設定方法」 を追加 ・ 「表 6.25. オンボード eMMC のスペック」 の SD Power Cycle Enable の値を修正 ・ 誤記修正
4.15.0	2024/10/30	<ul style="list-style-type: none"> ・ 「1.2.4. ソフトウェア使用に関する注意事項」 に「パスワードの設定について」を追加 ・ 「1.2.4. ソフトウェア使用に関する注意事項」 に「ソフトウェアのアップデートについて」を追加 ・ 「1.2.5. 本製品を廃棄する場合について」 を追加 ・ 「2.1.4. Armadillo Base OS のメンテナンスポリシーとアップデートの推奨」 にサポート期間に関する内容を追加 ・ 「4.4.7.1. JTAG と SD ブートを無効化する」 を追加 ・ 「図 4.15. 開発完了後のシステムをインストールディスクイメージにする」 の説明を修正 ・ 「5.8. Armadillo を廃棄する」 に詳細情報を追加 ・ 「6.9.6.12. Rest API : VPN の設定」 を追加 ・ 誤記修正
4.16.0	2024/11/27	<ul style="list-style-type: none"> ・ 「3.13.9. リリース版のビルド」 にリリース版ビルドでは開発用の機能が使用できないことを追記 ・ 「3.14.9. リリース版のビルド」 にリリース版ビルドでは開発用の機能が使用できないことを追記 ・ 「4.4.7.2. U-Boot のコマンドプロンプトを無効化する」 追加 ・ 「6.2.4.19. コンテナ起動前にコマンドを実行する」 追加 ・ 「6.8. SWUpdate の署名鍵と証明書の更新」 追加 ・ 「6.9.6.9. Rest API : WWAN の設定」 に IMEI・電話番号・電波品質の取得および SMS の作成送信・削除・一覧表示・内容表示を追加 ・ 誤記修正

Armadillo-640 製品マニュアル
Version 4.16.0
2024/11/27