Armadillo-610
製品マニュアル

A6100-U00Z
A6100-D00Z

Version 4.25.0
2025/09/24
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-610 製品マニュアル

株式会社アットマークテクノ

製作著作 © 2023-2025 Atmark Techno, Inc.

Version 4.25.0
2025/09/24

目次

1. はじめに 24
1.1. 本書について 24
1.1.1. 本書で扱うこと 24
1.1.2. 本書で扱わないこと 24
1.1.3. 本書で必要となる知識と想定する読者 25
1.1.4. 本書の構成 25
1.1.5. フォント 26
1.1.6. コマンド入力例 26
1.1.7. アイコン 26
1.1.8. ユーザー限定コンテンツ 27
1.1.9. 本書および関連ファイルのバージョンについて 27
1.2. 注意事項 27
1.2.1. 安全に関する注意事項 27
1.2.2. 取扱い上の注意事項 28
1.2.3. 製品の保管について 29
1.2.4. ソフトウェア使用に関しての注意事項 30
1.2.5. 本製品を廃棄する場合について 31
1.2.6. 電波障害について 31
1.2.7. 保証について 31
1.2.8. 輸出について 32
1.2.9. 商標について 32
1.3. 謝辞 32
2. 製品概要 33
2.1. 製品の特長 33
2.1.1. Armadillo とは 33
2.1.2. Armadillo-610 とは 33
2.1.3. Armadillo Base OS とは 35
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 37
2.1.5. Armadillo Twin とは 37
2.2. 製品ラインアップ 39
2.2.1. Armadillo-610 開発セット 39
2.2.2. Armadillo-610 量産ボード 39
2.3. インターフェースレイアウト 40
2.3.1. Armadillo-610 インターフェースレイアウト 40
2.3.2. Armadillo-610 拡張ボード インターフェースレイアウト 41
2.4. ブロック図 42
2.5. 使用可能なストレージデバイス 43
2.6. ストレージデバイスのパーティション構成 44
2.7. ソフトウェアのライセンス 45
3. 開発編 46
3.1. 開発の準備 46
3.1.1. 準備するもの 46
3.1.2. 仮想環境のセットアップ 47
3.1.3. VS Code のセットアップ 53
3.1.4. Armadillo の初期化と ABOS のアップデート 55
3.1.5. Armadillo に初期設定をインストールする 60
3.1.6. Python アプリケーションで動作確認する 71
3.1.7. シリアルコンソールを使用する 79
3.1.8. ユーザー登録 88
3.2. アプリケーション開発の流れ 89

3.3. 開発前に知っておくべき Armadillo Base OS の機能・特徴 91
3.3.1. 一般的な Linux OS 搭載組み込み機器との違い 91
3.3.2. Armadillo Base OS 搭載機器のソフトウェア開発手法 92
3.3.3. アップデート機能について 92
3.3.4. ファイルの取り扱いについて 98
3.3.5. インストールディスクについて 100
3.4. ハードウェアの設計 101
3.4.1. 信頼性試験データについて 101
3.4.2. 放射ノイズ 101
3.4.3. ESD/雷サージ 101
3.4.4. 放熱 102
3.4.5. 拡張ボードの設計 102
3.4.6. 電気的仕様 118
3.4.7. 形状図 124
3.4.8. オプション品 125
3.5. Device Tree をカスタマイズする 125
3.5.1. Linux カーネルソースコードの取得 125
3.5.2. at-dtweb のインストール 125
3.5.3. at-dtweb の起動 126
3.5.4. Device Tree をカスタマイズ 127
3.5.5. DT overlay によるカスタマイズ 135
3.6. インターフェースの使用方法とデバイスの接続方法 136
3.6.1. SD カードを使用する 137
3.6.2. Ethernet を使用する 141
3.6.3. UART を使用する 142
3.6.4. USB デバイスを使用する 143
3.6.5. 音声出力を行う 147
3.6.6. GPIO を制御する 147
3.6.7. I2C デバイスを使用する 152
3.6.8. SPI デバイスを使用する 153
3.6.9. CAN デバイスを使用する 153
3.6.10. PWM を使用する 154
3.6.11. JTAG デバッガを使用する 155
3.6.12. LCD を使用する 156
3.6.13. ユーザースイッチを使用する 158
3.6.14. LED を使用する 159
3.6.15. RTC を使用する 160
3.6.16. BT デバイスを使用する 162
3.6.17. Wi-SUN デバイスを扱う 163
3.6.18. EnOcean デバイスを扱う 163
3.6.19. Thread デバイスを扱う 164
3.7. ソフトウェアの設計 164
3.7.1. 開発者が開発するもの、開発しなくていいもの 164
3.7.2. ユーザーアプリケーションの設計 165
3.7.3. ログの設計 166
3.7.4. ウォッチドッグタイマー 166
3.7.5. コンテナに Armadillo の情報を渡す方法 167
3.7.6. Armadillo Base OS のデフォルトで開放しているポート 168
3.8. ネットワーク設定 169
3.8.1. ABOS Web とは 169
3.8.2. ABOS Web へのアクセス 170
3.8.3. ABOS Web のパスワード登録 173
3.8.4. ABOS Web のパスワード変更 176

3.8.5. ABOS Web の設定操作 177
3.8.6. ログアウト 177
3.8.7. WWAN 設定 177
3.8.8. WLAN 設定 180
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） 184
3.8.10. DHCP サーバー設定 185
3.8.11. NAT 設定 186
3.8.12. VPN 設定 188
3.8.13. 状態一覧 190
3.9. USB デバイスの接続を許可する 190
3.10. ABOS Web をカスタマイズする 196
3.11. ABOS Web から Armadillo の電源を操作する 199
3.12. Network Time Protocol (NTP, ネットワーク・タイム・プロトコル) の設定 200
3.13. Armadillo Twin を体験する 201
3.14. ABOSDE によるアプリケーションの開発 201
3.14.1. ABOSDE の対応言語 202
3.14.2. 参照する開発手順の章の選択 202
3.15. CUI アプリケーションの開発 203
3.15.1. CUI アプリケーション開発の流れ 203
3.15.2. ATDE 上でのセットアップ 204
3.15.3. アプリケーション開発 205
3.15.4. Armadillo に転送するディレクトリ及びファイル 209
3.15.5. コンテナ内のファイル一覧表示 210
3.15.6. Armadillo 上でのセットアップ 222
3.15.7. SBOM 生成に関する設定 227
3.15.8. リリース版のビルド 227
3.15.9. 製品への書き込み 227
3.15.10. Armadillo 上のコンテナイメージの削除 227
3.16. C 言語によるアプリケーションの開発 227
3.16.1. C 言語によるアプリケーション開発の流れ 228
3.16.2. ATDE 上でのセットアップ 228
3.16.3. アプリケーション開発 229
3.16.4. コンテナ内のファイル一覧表示 235
3.16.5. Armadillo に転送するディレクトリ及びファイル 247
3.16.6. Armadillo 上でのセットアップ 247
3.16.7. SBOM 生成に関する設定 252
3.16.8. リリース版のビルド 252
3.16.9. 製品への書き込み 252
3.16.10. Armadillo 上のコンテナイメージの削除 252
3.17. SBOM 生成に関わる設定を行う 252
3.17.1. SBOM 生成に必要なファイルを確認する 253
3.18. 生成した SBOM をスキャンする 254
3.18.1. OSV-Scanner のインストール 254
3.18.2. OSV-Scanner でソフトウェアの脆弱性を検査する 255
3.19. システムのテストを行う 256
3.19.1. ランニングテスト 256
3.19.2. 異常系における挙動のテスト 257
3.20. ユーザー設定とユーザーデータを一括削除する 257
4. 量産編 259
4.1. 概略 259
4.1.1. Armadillo Twin を契約する 259
4.1.2. リードタイムと在庫 260
4.1.3. Armadillo 納品後の製造・量産作業 260

4.2. BTO サービスを使わない場合と使う場合の違い 261
4.2.1. BTO サービスを利用しない(標準ラインアップ品) 261
4.2.2. BTO サービスを利用する 261
4.3. 量産時のイメージ書き込み手法 261
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 262
4.4.1. /etc/swupdate_preserve_file への追記 262
4.4.2. Armadillo Base OS の更新 263
4.4.3. パスワードの確認と変更 263
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 264
4.4.5. VS Code を使用して生成する 264
4.4.6. インストールディスクの動作確認を行う 268
4.4.7. コマンドラインから生成する 268
4.4.8. インストールの実行 275
4.5. SWUpdate を用いてイメージ書き込みする 275
4.5.1. SWU イメージの準備 275
4.5.2. desc ファイルの記述 276
4.6. イメージ書き込み後の動作確認 276
4.7. 量産時の組み立て 276
4.7.1. 拡張ボードの組み付け 276
4.7.2. オプションの組み付け 277
5. 運用編 278
5.1. Armadillo Twin に Armadillo を登録する 278
5.1.1. Armadillo の設置前に登録する場合 278
5.1.2. Armadillo の設置後に登録する場合 278
5.2. Armadillo を設置する 278
5.2.1. 設置場所 278
5.2.2. ケーブルの取り回し 278
5.2.3. サージ対策 278
5.2.4. 個体識別情報の取得 278
5.2.5. 電源を切る 281
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 281
5.3.1. アプリケーションのアップデート手順 281
5.4. Armadillo のソフトウェアをアップデートする 282
5.4.1. SWU イメージの作成 282
5.4.2. mkswu の desc ファイルを作成する 282
5.4.3. desc ファイルから SWU イメージを生成する 284
5.4.4. イメージのインストール 284
5.5. Armadillo Twin から複数の Armadillo をアップデートする 284
5.6. Armadillo Twin を利用してソフトウェアの脆弱性チェックを行う 285
5.7. eMMC の寿命を確認する 285
5.7.1. eMMC について 285
5.7.2. eMMC 予備領域の確認方法 285
5.8. Armadillo の部品変更情報を知る 285
5.9. Armadillo を廃棄する 286
6. 応用編 287
6.1. persist_file について 287
6.2. コンテナ 291
6.2.1. Podman - コンテナ仮想化ソフトウェアとは 291
6.2.2. コンテナの基本的な操作 291
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 306
6.2.4. コンテナ起動設定ファイルを作成する 308
6.2.5. アットマークテクノが提供するイメージを使う 316
6.2.6. alpine のコンテナイメージをインストールする 318

6.2.7. コンテナのネットワークを扱う 319
6.2.8. コンテナ内にサーバを構築する 321
6.2.9. 画面表示を行う 324
6.2.10. パワーマネジメント機能を使う 326
6.2.11. コンテナからの poweroff 及び reboot 329
6.2.12. 異常検知 329
6.3. swupdate を使用してアップデートする 330
6.3.1. swupdate で可能なアップデート 330
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 331
6.3.3. Armadillo Base OS の一括アップデート 334
6.3.4. ブートローダーのアップデート 338
6.3.5. swupdate がエラーする場合の対処 338
6.4. mkswu の .desc ファイルを編集する 338
6.4.1. インストールバージョンを指定する 338
6.4.2. Armadillo へファイルを転送する 340
6.4.3. Armadillo 上で任意のコマンドを実行する 341
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 342
6.4.5. 動作中の環境でのコマンドの実行 342
6.4.6. Armadillo にコンテナイメージを転送する 343
6.4.7. Armadillo のブートローダーを更新する 343
6.4.8. SWU イメージの設定関連 343
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 343
6.4.10. SWUpdate 実行中/完了後の挙動を指定する 343
6.4.11. desc ファイル設定例 344
6.5. swupdate_preserve_files について 346
6.6. SWU イメージの内容の確認 347
6.7. SWUpdate と暗号化について 347
6.8. SWUpdate の署名鍵と証明書の更新 347
6.8.1. 署名鍵と証明書の追加 347
6.8.2. 署名鍵と証明書の削除 349
6.9. Web UI から Armadillo をセットアップする (ABOS Web) 349
6.9.1. ABOS Web ではできないこと 349
6.9.2. ABOS Web の設定機能一覧と設定手順 350
6.9.3. コンテナ管理 350
6.9.4. SWU インストール 351
6.9.5. 時刻設定 353
6.9.6. アプリケーション向けのインターフェース (Rest API) 355
6.9.7. カスタマイズ 375
6.9.8. ユーザー設定とユーザーデータの削除 376
6.9.9. ABOS Web を停止する 376
6.9.10. ABOS Web を起動する 376
6.9.11. ABOS Web のセキュリティ対策 377
6.10. ABOSDE から ABOS Web の機能を使用する 378
6.10.1. Armadillo の SWU バージョンを取得する 379
6.10.2. Armadillo のコンテナの情報を取得する 380
6.10.3. Armadillo のコンテナを起動・停止する 381
6.10.4. Armadillo のコンテナのログを取得する 383
6.10.5. Armadillo に SWU をインストールする 383
6.11. ssh 経由で Armadillo Base OS にアクセスする 384
6.12. コマンドラインからネットワーク設定をする 384
6.12.1. 接続可能なネットワーク 385
6.12.2. IP アドレスの確認方法 385

6.12.3. ネットワークの設定方法 385
6.12.4. nmcli の基本的な使い方 386
6.12.5. 有線 LAN 389
6.12.6. ファイアウォールの設定方法 390
6.13. ストレージの操作 392
6.13.1. ストレージ内にアクセスする 392
6.13.2. ストレージを安全に取り外す 393
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 393
6.14. ボタンやキーを扱う 394
6.14.1. SW1 の短押しと長押しの対応 394
6.14.2. USB キーボードの対応 395
6.14.3. Armadillo 起動時にのみボタンに反応する方法 396
6.15. 動作中の Armadillo の温度を測定する 397
6.15.1. 温度測定の重要性 397
6.15.2. atmark-thermal-profiler をインストールする 397
6.15.3. atmark-thermal-profiler を実行・停止する 397
6.15.4. atmark-thermal-profiler が出力するログファイルを確認する 398
6.15.5. 温度測定結果の分析 398
6.15.6. Armadillo Twin から Armadillo の温度を確認する 400
6.15.7. 温度センサーの仕様 400
6.16. Armadillo Base OS をアップデートする 400
6.17. ロールバック状態を確認する 400
6.18. Armadillo 起動時にコンテナの外でスクリプトを実行する 402
6.19. u-boot の環境変数の設定 402
6.20. SD ブートの活用 404
6.20.1. ブートディスクの作成 405
6.20.2. SD ブートの実行 406
6.21. Armadillo のソフトウェアをビルドする 407
6.21.1. ブートローダーをビルドする 407
6.21.2. Linux カーネルをビルドする 408
6.21.3. Alpine Linux ルートファイルシステムをビルドする 412
6.22. eMMC の GPP(General Purpose Partition) を利用する 415
6.22.1. squashfs イメージを作成する 415
6.22.2. squashfs イメージを書き込む 415
6.22.3. GPP への書き込みを制限する 416
6.22.4. 起動時に squashfs イメージをマウントされるようにする 416
6.23. 動作ログ 417
6.23.1. 動作ログについて 417
6.23.2. 動作ログを取り出す 417
6.23.3. ログファイルのフォーマット 418
6.23.4. ログ用パーティションについて 418
6.23.5. /var/log/ 配下のログに関して 418
6.23.6. ABOS Web でログを確認する 418
6.24. ATDE・Linux でインストールディスクを作成する 419
6.24.1. GUI でインストールディスクを作成する 419
6.24.2. CUI でインストールディスクを作成する 422
6.25. シリアル通信ソフトウェア(minicom) 422
6.25.1. シリアル通信ソフトウェア(minicom)のセットアップ 422
6.25.2. minicom の起動 425
6.25.3. minicom の終了 426
6.26. vi エディタを使用する 426
6.26.1. vi の起動 427
6.26.2. 文字の入力 427

6.26.3. カーソルの移動 428
6.26.4. 文字の削除 428
6.26.5. 保存と終了 428
6.27. セキュリティ 429
6.27.1. SWUpdate と暗号化について 429
6.27.2. SBOM の提供 429
6.27.3. 不正な USB デバイスの接続を拒否する 431
6.27.4. EdgeLock SE050 を利用したキーストレージ 437
6.27.5. デバッグ機能を閉じる 446
6.27.6. コンテナの最小化 451
6.28. eFuse を変更する 453
6.28.1. ブートモード 453
6.28.2. ブートデバイス 454
6.28.3. eFuse の書き換え 455
6.28.4. eFuse の設定によるブートデバイスの選択 456
6.29. オプション品 458
6.29.1. USB シリアル変換アダプタ 458
6.29.2. Armadillo-610 拡張ボード 459
6.29.3. LCD オプションセット(7 インチタッチパネル WVGA 液晶) 491
6.29.4. Armadillo-400 シリーズ LCD オプションセット 492

図目次

1.1. 製品化までのロードマップ 25
2.1. Armadillo-610 とは 34
2.2. 拡張ボードの例 34
2.3. Armadillo Base OS とは 35
2.4. コンテナによるアプリケーションの運用 36
2.5. ロールバックの仕組み 36
2.6. Armadillo Twin　とは 38
2.7. Armadillo-610 インターフェースレイアウト 40
2.8. Armadillo-610 拡張ボード インターフェースレイアウト 41
2.9. Armadillo-610 ブロック図 43
3.1. GNOME 端末の起動 49
3.2. GNOME 端末のウィンドウ 49
3.3. ソフトウェアをアップデートする 50
3.4. ATDE にデバイスを接続する 50
3.5. 共有フォルダー設定を開く 51
3.6. 共有フォルダー設定 52
3.7. 共有フォルダーの追加 52
3.8. 「ファイル」に表示される共有フォルダー 53
3.9. VS Code を起動する 53
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 54
3.11. zip ファイルを展開 55
3.12. Win32 Disk Imager Renewal 設定画面 56
3.13. Armadillo-610 開発セットの組み立て 57
3.14. Armadillo-610 を初期化する接続 59
3.15. ジャンパピンの位置 60
3.16. initial_setup.swu を作成する 61
3.17. initial_setup.swu 初回生成時の各種設定 61
3.18. ABOS にアクセスするための接続 63
3.19. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする 64
3.20. ABOSDE に表示されている Armadillo を更新する 65
3.21. ABOSDE を使用して ABOS Web を開く 66
3.22. パスワード登録画面 66
3.23. パスワード登録完了画面 67
3.24. ログイン画面 67
3.25. トップページ 68
3.26. SWU インストール 69
3.27. SWU インストールに成功した画面 69
3.28. プロジェクトを作成する 71
3.29. プロジェクト名を入力する 71
3.30. VS Code で初期設定を行う 72
3.31. VS Code のターミナル 72
3.32. SSH 用の鍵を生成する 72
3.33. VS Code でコンテナイメージの作成を行う 73
3.34. コンテナイメージの作成完了 74
3.35. ABOSDE で Armadillo に SWU をインストール 74
3.36. ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する 75
3.37. ssh_config を編集する 75
3.38. Armadillo 上でアプリケーションを実行する 76
3.39. 実行時に表示されるメッセージ 76
3.40. アプリケーションを終了する 77

3.41. Armadillo 上のコンテナイメージを削除する 78
3.42. スライドスイッチの設定 79
3.43. USB シリアル変換アダプタの挿抜角度 80
3.44. シリアルコンソールを使用する配線例 81
3.45. COM7 が競合している状態 82
3.46. BT に割当の COM を変更した状態 82
3.47. Armadillo を接続している COM ポートを指定 83
3.48. Armadillo を接続しているシリアルポートの設定 84
3.49. アプリケーション開発の流れ 90
3.50. persist_file コマンド実行例 99
3.51. chattr によって copy-on-write を無効化する例 100
3.52. Armadillo-610 の CON2 102
3.53. Armadillo-610 のマルチプレクス表 102
3.54. 電源回路例 104
3.55. 起動設定ジャンパー例 105
3.56. LAN(Ethernet)接続例 106
3.57. WLAN/BT 接続例 106
3.58. USB Host 接続例 107
3.59. シリアル(UART)接続例 107
3.60. SD 接続例 108
3.61. スイッチ、LED、リレー接続例 109
3.62. MQS 接続例 110
3.63. PWRON 回路例 110
3.64. ONOFF 回路例 110
3.65. 拡張ボード推奨レイアウト 111
3.66. Armadillo-610 の固定例 112
3.67. 電源回路の構成 120
3.68. 電源シーケンス 121
3.69. リセット回路の構成 122
3.70. システムリセットする場合の Low レベル保持時間 122
3.71. ONOFF 回路の構成 123
3.72. 基板形状および固定穴寸法 124
3.73. コネクタ中心寸法 124
3.74. Armadillo-610 のスタッキング高さ 124
3.75. at-dtweb の起動開始 126
3.76. ボード選択画面 126
3.77. Linux カーネルディレクトリ選択画面 127
3.78. at-dtweb 起動画面 127
3.79. UART1(RXD/TXD)のドラッグ 128
3.80. CON2 83/85 ピンへのドロップ 129
3.81. 信号名の確認 130
3.82. プロパティの設定 131
3.83. プロパティの保存 131
3.84. 全ての機能の削除 132
3.85. UART1(RXD/TXD)の削除 133
3.86. DTS/DTB の生成 134
3.87. dtbo/desc の生成完了 134
3.88. /boot/overlays.txt の変更例 135
3.89. Armadillo-610 のインターフェース 137
3.90. カバーのロックを解除する 139
3.91. カバーを開ける 139
3.92. microSD カードの挿抜 139
3.93. カードマークの確認 140

3.94. カバーを閉める 140
3.95. カバーをロックする 140
3.96. 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 143
3.97. 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 143
3.98. USB シリアルデバイスを扱うためのコンテナ作成例 144
3.99. setserial コマンドによる USB シリアルデバイス設定の確認例 144
3.100. USB カメラを扱うためのコンテナ作成例 145
3.101. USB メモリをホスト OS 側でマウントする例 145
3.102. 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 145
3.103. USB メモリに保存されているデータの確認例 146
3.104. USB メモリをマウントするためのコンテナ作成例 146
3.105. コンテナ内から USB メモリをマウントする例 146
3.106. 音声出力を行うためのコンテナ作成例 147
3.107. alsa-utils による音声出力を行う例 147
3.108. GPIO を扱うためのコンテナ作成例 148
3.109. コンテナ内からコマンドで GPIO を操作する例 149
3.110. gpiodetect コマンドの実行 149
3.111. gpioinfo コマンドの実行 149
3.112. I2C を扱うためのコンテナ作成例 152
3.113. i2cdetect コマンドによる確認例 152
3.114. SPI を扱うためのコンテナ作成例 153
3.115. spi-config コマンドによる確認例 153
3.116. CAN を扱うためのコンテナ作成例 154
3.117. CAN の設定例 154
3.118. PWM を扱うためのコンテナ作成例 155
3.119. PWM の動作設定例 155
3.120. LCD の接続方法 157
3.121. フレキシブルフラットケーブルの形状 157
3.122. 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 159
3.123. evtest コマンドによる確認例 159
3.124. LED を扱うためのコンテナ作成例 160
3.125. LED の点灯/消灯の実行例 160
3.126. RTC を扱うためのコンテナ作成例 161
3.127. hwclock コマンドによる RTC の時刻表示と設定例 161
3.128. BT を扱うコンテナの作成例 162
3.129. BT を起動する実行例 162
3.130. bluetoothctl コマンドによるスキャンとペアリングの例 163
3.131. Wi-SUN デバイスを扱うためのコンテナ作成例 163
3.132. EnOcean デバイスを扱うためのコンテナ作成例 164
3.133. Thread デバイスを扱うためのコンテナ作成例 164
3.134. 開発者が開発するもの、開発しなくていいもの 165
3.135. 現在の面の確認方法 166
3.136. add_args を用いてコンテナに情報を渡すための書き方 167
3.137. add_args を用いてコンテナに情報を渡す例 167
3.138. avahi-daemon を停止する 168
3.139. avahi-daemon を起動する 168
3.140. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする 171
3.141. ABOSDE を使用して ABOS Web を開く 172
3.142. ABOSDE に表示されている Armadillo を更新する 172
3.143. パスワード登録画面 173
3.144. パスワード登録完了画面 174
3.145. ログイン画面 175
3.146. トップページ 176

3.147. ログイン画面 177
3.148. WWAN 設定画面 179
3.149. WLAN クライアント設定画面 181
3.150. WLAN アクセスポイント設定画面 183
3.151. 現在の接続情報画面 184
3.152. LAN 接続設定で固定 IP アドレスに設定した画面 185
3.153. eth0 に対する DHCP サーバー設定 186
3.154. LTE を宛先インターフェースに指定した設定 187
3.155. 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 188
3.156. VPN 設定 189
3.157. USB 接続制御の設定画面 191
3.158. 接続済みの USB デバイス 192
3.159. 接続済み USB デバイスの詳細情報 192
3.160. USB デバイスを許可する 193
3.161. 許可済み USB デバイス 193
3.162. 接続済みの個体と同じメーカー・製品の全ての個体を許可する 194
3.163. 全ての個体を許可する場合の表示 194
3.164. 許可ルールを削除する 195
3.165. USB デバイスクラス 195
3.166. USB デバイスクラスの追加 196
3.167. ABOS Web のカスタマイズ設定 197
3.168. メニュー変更画面 (一部) 199
3.169. 「電源制御」の位置 200
3.170. 「電源制御」の画面 200
3.171. chronyd のコンフィグの変更例 201
3.172. 参照する開発手順の章を選択する流れ 202
3.173. CUI アプリケーション開発の流れ 203
3.174. プロジェクトを作成する 204
3.175. プロジェクト名を入力する 204
3.176. VS Code で my_project を起動する 205
3.177. 初期設定を行う 206
3.178. VS Code で初期設定を行う 206
3.179. VS Code のターミナル 206
3.180. SSH 用の鍵を生成する 206
3.181. ABOSDE でコンテナ OS を選択する 207
3.182. VS Code でコンテナイメージの作成を行う 208
3.183. コンテナイメージの作成完了 208
3.184. BLE パッケージをインストールする 209
3.185. コンテナ内のファイル一覧を表示するタブ 210
3.186. コンテナ内のファイル一覧の例 211
3.187. resources ディレクトリ 212
3.188. コンテナ内のファイル一覧を再表示するボタン 213
3.189. container/resources 下にファイルを追加するボタン 214
3.190. ファイル名を入力 214
3.191. 追加されたファイルの表示 215
3.192. container/resources 下にフォルダーを追加するボタン 216
3.193. container/resources 下にあるファイルを開くボタン 217
3.194. container/resources 下にあるファイルを削除するボタン 218
3.195. コンテナ内のファイルを container/resources 下に保存するボタン 219
3.196. 編集前のファイルを示すマーク 220
3.197. 編集後のファイルを示すマーク 221
3.198. コンテナ内にコピーされないことを示すマーク 222
3.199. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする 223

3.200. ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する 224
3.201. ABOSDE に表示されている Armadillo を更新する 225
3.202. ssh_config を編集する 225
3.203. Armadillo 上でアプリケーションを実行する 226
3.204. 実行時に表示されるメッセージ 226
3.205. アプリケーションを終了する 226
3.206. リリース版をビルドする 227
3.207. C 言語によるアプリケーション開発の流れ 228
3.208. プロジェクトを作成する 229
3.209. プロジェクト名を入力する 229
3.210. VS Code で my_project を起動する 230
3.211. 初期設定を行う 231
3.212. VS Code で初期設定を行う 231
3.213. VS Code のターミナル 231
3.214. SSH 用の鍵を生成する 231
3.215. ABOSDE でコンテナ OS を選択する 232
3.216. C 言語による開発における packages.txt の書き方 233
3.217. VS Code でコンテナイメージの作成を行う 234
3.218. コンテナイメージの作成完了 235
3.219. コンテナ内のファイル一覧を表示するタブ 235
3.220. コンテナ内のファイル一覧の例 236
3.221. resources ディレクトリ 237
3.222. コンテナ内のファイル一覧を再表示するボタン 238
3.223. container/resources 下にファイルを追加するボタン 239
3.224. ファイル名を入力 239
3.225. 追加されたファイルの表示 240
3.226. container/resources 下にフォルダーを追加するボタン 241
3.227. container/resources 下にあるファイルを開くボタン 242
3.228. container/resources 下にあるファイルを削除するボタン 243
3.229. コンテナ内のファイルを container/resources 下に保存するボタン 244
3.230. 編集前のファイルを示すマーク 245
3.231. 編集後のファイルを示すマーク 246
3.232. コンテナ内にコピーされないことを示すマーク 247
3.233. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする 248
3.234. ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する 249
3.235. ABOSDE に表示されている Armadillo を更新する 250
3.236. ssh_config を編集する 250
3.237. Armadillo 上でアプリケーションを実行する 251
3.238. 実行時に表示されるメッセージ 251
3.239. アプリケーションを終了する 251
3.240. リリース版をビルドする 252
3.241. mkswu バージョン確認コマンド 253
3.242. mkswu のインストール・アップデートコマンド 253
3.243. make_sbom.sh 実行確認コマンド 253
3.244. python3-make-sbom のインストールコマンド 253
3.245. OSV-Scanner の実行ファイルをダウンロード 254
3.246. OSV-Scanner をインストールする 255
3.247. OSV-Scanner がインストールされたことを確認する 255
3.248. OSV-Scanner を用いて SBOM をスキャンする 255
3.249. メモリの空き容量の確認方法 256
3.250. 削除されるユーザー設定とユーザーデータを確認 257
3.251. 実際にユーザー設定とユーザーデータを削除する 258
4.1. Armadillo 量産時の概略図 259

4.2. BTO サービスで対応する範囲 261
4.3. 任意のファイルパスを/etc/swupdate_preserve_files に追記する 263
4.4. Armadillo Base OS を最新にアップデートする 263
4.5. パスワードを変更する 263
4.6. make-installer.swu を作成する 265
4.7. 対象製品を選択する 265
4.8. make-installer.swu 生成時のログ 265
4.9. make-installer.swu インストール時のログ 266
4.10. JTAG と SD ブートを無効化する 269
4.11. JTAG と SD ブートの設定値を確認する 269
4.12. JTAG と SD ブートの設定値をリセットする 269
4.13. U-Boot のコマンドプロンプトを無効化する 270
4.14. U-Boot のコマンドプロンプトの設定値を確認する 270
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 270
4.16. ip_config.txt の内容 273
4.17. IP アドレスの確認 273
4.18. allocated_ips.csv の内容 274
4.19. インストールログを保存する 274
4.20. インストールログの中身 275
4.21. Armadillo に書き込みたいソフトウェアを ATDE に配置 275
4.22. desc ファイルの記述例 276
5.1. 個体番号の取得方法（device-info） 279
5.2. device-info のインストール方法 279
5.3. 個体番号の取得方法（get-board-info） 280
5.4. 個体番号の環境変数を conf ファイルに追記 280
5.5. コンテナ上で個体番号を確認する方法 280
5.6. MAC アドレスの確認方法 280
5.7. 出荷時の Ethernet MAC アドレスの確認方法 281
5.8. VS Code を起動 281
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 283
5.10. コンテナイメージアーカイブ作成例 283
5.11. sample_container_update.desc の内容 284
5.12. sample_container_update.desc の内容 284
5.13. eMMC の予備領域使用率を確認する 285
6.1. OverlayFS の構成 287
6.2. persist_file コマンドの説明 288
6.3. 書き込み時の OverlayFS の挙動 288
6.4. 書き込み後の persist_file コマンドの実行 289
6.5. persist_file のヘルプ 289
6.6. persist_file 保存・削除手順例 290
6.7. 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 290
6.8. persist_file 変更ファイルの一覧表示例 290
6.9. persist_file でのパッケージインストール手順例 291
6.10. コンテナを作成する実行例 292
6.11. イメージ一覧の表示実行例 293
6.12. podman images --help の実行例 293
6.13. コンテナー覧の表示実行例 293
6.14. podman ps --help の実行例 294
6.15. コンテナを起動する実行例 294
6.16. コンテナを起動する実行例(a オプション付与) 294
6.17. podman start --help 実行例 295
6.18. コンテナを停止する実行例 295
6.19. podman stop --help 実行例 295

6.20. my_container を保存する例 295
6.21. podman build の実行例 296
6.22. podman build でのアップデートの実行例 296
6.23. コンテナを削除する実行例 297
6.24. イメージを削除する実行例 298
6.25. podman rmi --help 実行例 298
6.26. Read-Only のイメージを削除する実行例 298
6.27. コンテナ内部のシェルを起動する実行例 299
6.28. コンテナ内部のシェルから抜ける実行例 299
6.29. podman exec --help 実行例 299
6.30. コンテナを作成する実行例 300
6.31. コンテナの IP アドレスを確認する実行例 300
6.32. ping コマンドによるコンテナ間の疎通確認実行例 300
6.33. pod を使うコンテナを自動起動するための設定例 301
6.34. network を使うコンテナを自動起動するための設定例 301
6.35. abos-ctrl podman-rw の実行例 303
6.36. abos-ctrl podman-storage のイメージコピー例 304
6.37. Armadillo 上のコンテナイメージを削除する 307
6.38. abos-ctrl container-clear 実行例 308
6.39. コンテナを自動起動するための設定例 308
6.40. ボリュームを shared でサブマウントを共有する例 310
6.41. /proc/devices の内容例 311
6.42. add_armadillo_env で設定した環境変数の確認方法 312
6.43. 上記の例でエラーを発生させた際の起動ログ 315
6.44. インストール用のプロジェクトを作成する 317
6.45. 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 317
6.46. Docker ファイルによるイメージのビルドの実行例 318
6.47. ビルド済みイメージを load する実行例 318
6.48. 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 319
6.49. コンテナの IP アドレス確認例 319
6.50. ip コマンドを用いたコンテナの IP アドレス確認例 320
6.51. ユーザ定義のネットワーク作成例 320
6.52. IP アドレス固定のコンテナ作成例 320
6.53. コンテナの IP アドレス確認例 321
6.54. コンテナに Apache をインストールする例 321
6.55. コンテナに lighttpd をインストールする例 322
6.56. コンテナに vsftpd をインストールする例 322
6.57. ユーザを追加する例 322
6.58. 設定ファイルの編集例 323
6.59. vsftpd の起動例 323
6.60. コンテナに samba をインストールする例 323
6.61. ユーザを追加する例 323
6.62. samba の起動例 324
6.63. コンテナに sqlite をインストールする例 324
6.64. sqlite の実行例 324
6.65. X Window System を扱うためのコンテナ起動例 324
6.66. コンテナ内で X Window System を起動する実行例 325
6.67. 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 325
6.68. フレームバッファに直接描画する実行例 326
6.69. タッチパネルを扱うためのコンテナ作成例 326
6.70. パワーマネジメント機能を使うためのコンテナ作成例 326
6.71. サスペンド状態にする実行例 327
6.72. poweroff コマンドで電源を切断し、180 秒後に i.mx6ull の RTC で起床する 328

6.73. サスペンド状態にする実行例、rtc で起こす 328
6.74. コンテナから shutdown を行う 329
6.75. ソフトフ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 329
6.76. 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 330
6.77. 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 330
6.78. 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 330
6.79. Armadillo Base OS を B 面にコピー 332
6.80. desc ファイルに記述した swudesc_* コマンドを実行 332
6.81. アップデート完了後の挙動 333
6.82. B 面への切り替え 334
6.83. Armadillo Base OS とファイルを B 面にコピー 335
6.84. desc ファイルに記述した swudesc_* コマンドを実行 336
6.85. アップデート完了後の挙動 337
6.86. B 面への切り替え（component=base_os） 337
6.87. mkswu --genkey で署名鍵と証明書を追加する 347
6.88. mkswu --genkey により mkswu.conf に追加された内容 348
6.89. 新しい証明書が Armadillo に追加されていることを確認する 348
6.90. 署名鍵と証明書を削除する設定 349
6.91. 証明書がインストールされていることを確認する 349
6.92. コンテナ管理 351
6.93. SWU インストール 352
6.94. 最新バージョンに更新 352
6.95. SWU 管理対象ソフトウェアコンポーネントの一覧表示 353
6.96. 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 354
6.97. 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 354
6.98. ネットワークタイムサーバーの設定項目 354
6.99. タイムゾーンの設定項目 355
6.100. 設定管理の Rest API トークン一覧表示 356
6.101. ユーザ名とパスワード認証の例 370
6.102. 証明書認証の例 370
6.103. ABOS Web を停止する 376
6.104. ABOS Web を起動する 376
6.105. コンテナからのアクセスのみを許可する設定 377
6.106. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする 379
6.107. ABOSDE の ABOS Web パスワード入力画面 379
6.108. ABOSDE で Armadillo の SWU バージョンを取得 380
6.109. ABOSDE で Armadillo のコンテナ情報を取得 381
6.110. ABOSDE で Armadillo のコンテナを起動 382
6.111. ABOSDE で Armadillo のコンテナを停止 382
6.112. ABOSDE で Armadillo のコンテナのログを取得 383
6.113. ABOSDE で Armadillo に SWU をインストール 384
6.114. IP アドレスの確認 385
6.115. IP アドレス(eth0)の確認 385
6.116. nmcli のコマンド書式 386
6.117. コネクションの一覧 386
6.118. コネクションの有効化 386
6.119. コネクションの無効化 386
6.120. コネクションの作成 387
6.121. コネクションファイルの永続化 387
6.122. コネクションの削除 387
6.123. コネクションファイル削除時の永続化 388
6.124. 固定 IP アドレス設定 388
6.125. DNS サーバーの指定 388

6.126. DHCP の設定 388
6.127. コネクションの修正の反映 389
6.128. デバイスの一覧 389
6.129. デバイスの接続 389
6.130. デバイスの切断 389
6.131. 有線 LAN の PING 確認 390
6.132. 特定のポートに対する IP アドレスのフィルタリング 390
6.133. 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 391
6.134. mount コマンド書式 392
6.135. ストレージのマウント 392
6.136. ストレージのアンマウント 393
6.137. fdisk コマンドによるパーティション変更 393
6.138. EXT4 ファイルシステムの構築 394
6.139. buttond で SW1 を扱う 394
6.140. buttond で USB キーボードのイベントを確認する 395
6.141. buttond で USB キーボードを扱う 396
6.142. buttond で SW1 を Armadillo 起動時のみ受け付ける設定例 396
6.143. atmark-thermal-profiler をインストールする 397
6.144. atmark-thermal-profiler を実行する 397
6.145. atmark-thermal-profiler を停止する 398
6.146. ログファイルの内容例 398
6.147. 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) 399
6.148. Armadillo-IoT ゲートウェイ G4 で取得した温度のグラフ 399
6.149. abos-ctrl status の例 400
6.150. /var/at-log/atlog の内容の例 401
6.151. local サービスの実行例 402
6.152. uboot_env.d のコンフィグファイルの例 403
6.153. デフォルトコンフィギュレーションの適用 407
6.154. Linux カーネルを SWU でインストールする方法 410
6.155. Linux カーネルを build_rootfs でインストールする方法 411
6.156. squashfs イメージの作成 415
6.157. mmc-utils のインストール 416
6.158. eMMC の GPP に Temporary Write Protection をかける 416
6.159. 動作ログのフォーマット 418
6.160. zip ファイルを展開 420
6.161. 展開したフォルダ内にある img ファイルをダブルクリック 420
6.162. ディスクイメージをリストア 420
6.163. microSD カードを指定 421
6.164. 確認のウィンドウ 421
6.165. パスワードの要求 421
6.166. minicom の設定の起動 423
6.167. minicom の設定 423
6.168. minicom のシリアルポートの設定 423
6.169. 例. シリアルポート接続時のログ 424
6.170. minicom のシリアルポートのパラメータの設定 424
6.171. minicom シリアルポートの設定値 425
6.172. minicom 起動方法 426
6.173. minicom 終了確認 426
6.174. vi の起動 427
6.175. 入力モードに移行するコマンドの説明 427
6.176. 文字を削除するコマンドの説明 428
6.177. desc ファイルの追加例 431
6.178. USB 接続制御機能を管理するコマンド 432

6.179. USB 接続制御機能の状態を確認する 433
6.180. USB 接続制御機能を有効化する 433
6.181. USB 接続制御機能を無効化する 433
6.182. 接続されている USB デバイスをリストする 433
6.183. 指定した USB デバイスを許可する 434
6.184. パラメータで指定した USB デバイスを許可する 434
6.185. Armadillo に接続している USB デバイスのパラメータを調べる 434
6.186. 指定した USB デバイスを拒否する 435
6.187. 指定した USB デバイスクラスを許可する 435
6.188. 指定可能な USB デバイスクラスを確認する 435
6.189. 定義済みの USB デバイス許可ルールを表示する 436
6.190. 定義済みの USB デバイス許可ルールを削除する 437
6.191. Plug & Trust Middleware の周辺のソフトウェアスタック 438
6.192. EdgeLock SE050 内に保存されているユーザーが保存した鍵をすべて削除する 445
6.193. 削除したユーザーが保存した鍵を読み出そうとしてみる 445
6.194. インストールディスクの JTAG と SD ブートを無効化する 447
6.195. JTAG と SD ブートの設定値を確認する 448
6.196. JTAG と SD ブートの設定値をリセットする 448
6.197. インストールディスクを作成する 448
6.198. Debian コンテナの Dockerfile の例 452
6.199. USB シリアル変換アダプタの配線 458
6.200. Armadillo-610 拡張ボードのブロック図 461
6.201. Armadillo-610 拡張ボードの電源回路の構成 462
6.202. Armadillo-610 拡張ボードのインターフェースの概要 463
6.203. Armadillo-610 拡張ボード CON1 464
6.204. Armadillo-610 拡張ボード CON2 465
6.205. USB シリアル変換アダプタの挿抜角度 466
6.206. Armadillo-610 拡張ボード CON3 466
6.207. Armadillo-610 拡張ボード CON4 467
6.208. Armadillo-610 拡張ボード CON5 470
6.209. Armadillo-610 拡張ボード CON6 471
6.210. Armadillo-610 拡張ボード CON7、CON8、CON9、CON10 472
6.211. Armadillo-610 拡張ボード CON11 473
6.212. AC アダプタの極性マーク 474
6.213. Armadillo-610 拡張ボード CON12 475
6.214. CON13A の配置 475
6.215. CON13B の配置 476
6.216. CON13C の配置 477
6.217. スピーカーのリード線 477
6.218. CON13D の配置 477
6.219. Armadillo-610 拡張ボード CON14 478
6.220. Armadillo-610 拡張ボード CON15、CON16 478
6.221. Armadillo-610 拡張ボード CON17 480
6.222. バックアップ電源供給回路 480
6.223. Armadillo-610 拡張ボード CON18 481
6.224. Armadillo-610 拡張ボード CON19 482
6.225. Armadillo-610 拡張ボード CON20 483
6.226. Armadillo-610 拡張ボード CON21 484
6.227. Armadillo-610 拡張ボード CON22 485
6.228. Armadillo-610 拡張ボード CON23 486
6.229. Armadillo-610 拡張ボード CON24 486
6.230. Armadillo-610 拡張ボード JP1 487
6.231. Armadillo-610 拡張ボード SW1 488

6.232. Armadillo-610 拡張ボード SW2 488
6.233. Armadillo-610 拡張ボード SW3 489
6.234. Armadillo-610 拡張ボード LED1、LED2 489
6.235. Armadillo-610 拡張ボード LED3 490
6.236. Armadillo-610 拡張ボード 基板形状および固定穴寸法 490
6.237. Armadillo-610 拡張ボード コネクタ中心寸法および穴寸法 491
6.238. LCD 拡張ボードのブロック図 494
6.239. LCD 拡張ボードのインターフェース 495
6.240. タッチパネル LCD と LCD 拡張ボードの接続 502
6.241. 両面テープの貼付位置 503
6.242. Armadillo-610 と LCD 拡張ボードの接続 504
6.243. フレキシブルフラットケーブルの形状 504
6.244. LCD 拡張ボードの基板形状および固定穴寸法 505
6.245. LCD 拡張ボードの両面テープと固定部品配置可能位置 505
6.246. LCD 拡張ボードのコネクタ位置寸法 506

表目次

1.1. 使用しているフォント 26
1.2. 表示プロンプトと実行環境の関係 26
1.3. コマンド入力例での省略表記 26
1.4. 推奨温湿度環境について 29
2.1. Armadillo-610 ラインアップ 39
2.2. Armadillo-610 開発セットのセット内容 39
2.3. 仕様 39
2.4. Armadillo-610 インターフェース内容 41
2.5. Armadillo-610 拡張ボード インターフェース内容 41
2.6. ストレージデバイス 43
2.7. eMMC の GPP の用途 44
2.8. eMMC メモリマップ 44
2.9. eMMC ブートパーティション構成 44
2.10. eMMC GPP 構成 44
3.1. ユーザー名とパスワード 48
3.2. 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) 99
3.3. BJP1 の状態と起動デバイス 104
3.4. CON2 信号配列 113
3.5. 絶対最大定格 118
3.6. 推奨動作条件 118
3.7. 入出力インターフェース(電源)の電気的仕様 118
3.8. 入出力インターフェースの電気的仕様(OVDD = +3.3V_IO、VDD_SNVS_IN) 119
3.9. ONOFF ピンから電源オン、オフ切り替えする際の Low 保持時間 123
3.10. Armadillo-610 インターフェース一覧 137
3.11. CON1 信号配列 138
3.12. Armadillo-610 拡張ボード: CON13B ピン番号と GPIO 番号の対応 150
3.13. Armadillo-610: CON2 ピン番号と GPIO 番号の対応 150
3.14. I2C デバイス 152
3.15. CON10 信号配列 156
3.16. キーコード 158
3.17. LED5 159
3.18. LED クラスディレクトリと LED の対応 160
3.19. Armadillo Base OS のデフォルトで開放しているポート 168
3.20. デフォルトで許可されている USB デバイス 190
3.21. 用意する favicon 画像 198
3.22. 制御できる機能 200
3.23. ABOSDE の対応言語 202
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 262
4.2. インストール中に実行される関数 272
5.1. EXT_CSD_PRE_EOL_INFO の値の意味 285
6.1. add_hotplugs オプションに指定できる主要な文字列 311
6.2. add_armadillo_env で追加される環境変数 312
6.3. 対応するパワーマネジメント状態 327
6.4. swudesc_* コマンドの種類 333
6.5. アップデート完了後の挙動の種類 334
6.6. swudesc_* コマンドの種類 336
6.7. アップデート完了後の挙動の種類 337
6.8. ネットワークとネットワークデバイス 385
6.9. 固定 IP アドレス設定例 388
6.10. thermal_profile.csv の各列の説明 398

6.11. rollback-status の出力と意味 401
6.12. rollback-status 追加情報の出力と意味 401
6.13. u-boot の主要な環境変数 403
6.14. microSD カードのパーティション構成 405
6.15. build-rootfs のファイル説明 413
6.16. /var/log/ 配下のログ 418
6.17. シリアル通信設定 423
6.18. シリアルコンソールとして使用する取り外し可能デバイス 425
6.19. 入力モードに移行するコマンド 427
6.20. カーソルの移動コマンド 428
6.21. 文字の削除コマンド 428
6.22. 保存・終了コマンド 428
6.23. desc ファイルの設定項目 431
6.24. デバイスリストの各列の意味 433
6.25. 2 列目が device のときの許可ルールリストの各列の意味 437
6.26. Plug & Trust Middle のパッケージ 438
6.27. Plug & Trust Middleware のサポート 438
6.28. Debian と Alpine の比較 452
6.29. Python プロジェクトにおける Debian と Alpine コンテナのサイズ比較 452
6.30. Python プロジェクトにおける `--no-install-recommends` の有無による Debian コンテナのサイズ比較 453
6.31. GPIO override と eFuse 454
6.32. ブートデバイスと eFuse 455
6.33. オンボード eMMC のスペック 456
6.34. Armadillo-610 関連のオプション品 458
6.35. 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 459
6.36. Armadillo-610 拡張ボードの仕様 459
6.37. Armadillo-610 拡張ボードのインターフェース一覧 463
6.38. CON1 信号配列 465
6.39. CON2 信号配列 465
6.40. CON3 信号配列 466
6.41. CON4 信号配列 467
6.42. CON5 信号配列 470
6.43. CON6 信号配列 471
6.44. CON7 信号配列 472
6.45. CON8 信号配列 472
6.46. CON9 信号配列 472
6.47. CON10 信号配列 472
6.48. CON11 信号配列 473
6.49. CON12 信号配列 475
6.50. CON13A 信号配列 475
6.51. CON13B 信号配列 476
6.52. CON13C 信号配列 477
6.53. CON13D 信号配列 477
6.54. CON14 信号配列 478
6.55. CON16 信号配列 478
6.56. CON15、CON16 対応電池の例とバックアップ時間 478
6.57. CON17 信号配列 480
6.58. CON17 対応電池の例とバックアップ時間 480
6.59. CON18 信号配列 481
6.60. CON19 信号配列 482
6.61. CON20 信号配列 483
6.62. CON21 信号配列 484

6.63. CON22 信号配列 485
6.64. CON23 信号配列 486
6.65. CON24 信号配列 487
6.66. JP1 信号配列 487
6.67. ジャンパの設定と起動デバイス 487
6.68. SW1 信号配列 488
6.69. SW2 信号配列 488
6.70. SW3 信号配列 489
6.71. LAN LED の動作 489
6.72. LED3 490
6.73. 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 492
6.74. LCD の仕様 492
6.75. Armadillo-400 シリーズ LCD オプションセットについて 493
6.76. LCD 拡張ボードの仕様 493
6.77. LCD の仕様 493
6.78. LCD 拡張ボード インターフェース一覧 495
6.79. CON1 信号配列 496
6.80. CON2 信号配列 497
6.81. CON3 信号配列 498
6.82. CON4 信号配列 498
6.83. CON5 信号配列 498
6.84. CON6 信号配列 499
6.85. CON7 信号配列 500
6.86. CON8 信号配列 500
6.87. CON9、CON10 信号配列 500
6.88. SW1、SW2、SW3、SW4、SW5、SW6 の機能 501
6.89. LED1 の機能 501
6.90. LED2、LED3 の機能 501

1. はじめに

このたびは Armadillo-610 をご利用いただき、ありがとうございます。

Armadillo-610 は Armadillo-640 の性能と品質をそのままユーザーオリジナルの拡張ボードとの接続を可能にした、50mm×50mm の小型コンピューターモジュールです。

Ethernet PHY や microSD カードコネクタは既にボード上に搭載しているため、簡単な外付け部品で自由に拡張ボードを設計していただけます。

有線 LAN 通信時でも約 1W(モジュール部分のみ)で動作し、スリープ時には 0.1W 以下の低消費電力で設計しているためバッテリー駆動を想定した製品開発にも最適です。

Armadillo-610 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ので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-610 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-610 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-610 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-610 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-610 は組込みプラットフォームとして実績のある Armadillo をベースとしているため、標準で有効になっている機能以外にも様々な機能を実現することができます。

ソフトウェアエンジニア	端末からのコマンドの実行方法など、基本的な Linux の扱い方を知っているエンジニアを対象読者として想定しています。プログラミング言語として C/C++を扱えることは必ずしも必要ではありませんが、基礎的な知識がある方が理解しやすい部分もあります。
ハードウェアエンジニア	電子工学の基礎知識を有したエンジニアを対象読者として想定しています。回路図や部品表を読み、理解できる必要があります。

1.1.4. 本書の構成

本書には、Armadillo-610 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。

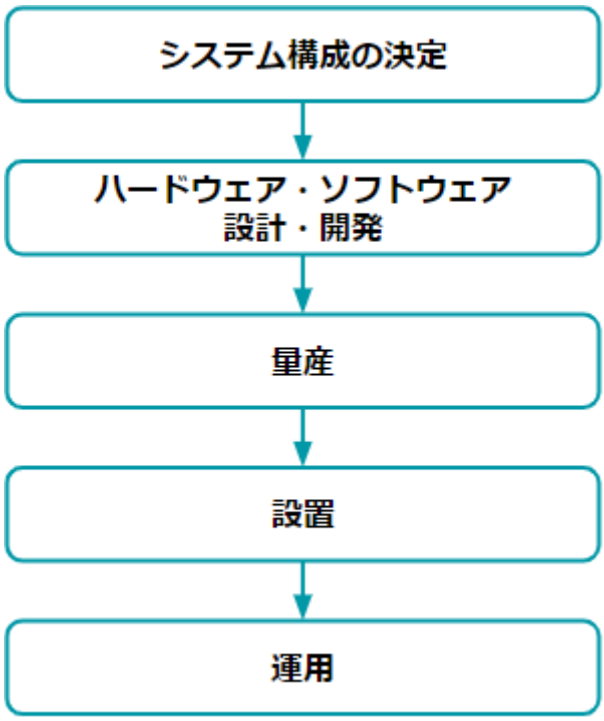


図 1.1 製品化までのロードマップ

- **「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-610 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いわけています。

表 1.1 使用しているフォント

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

1.1.6. 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-610 ドキュメントダウンロード

https://armadillo.atmark-techno.com/armadillo-610/resources/documents

Armadillo サイト - Armadillo-610 ソフトウェアダウンロード

https://armadillo.atmark-techno.com/armadillo-610/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	基板間コネクタ、microSD スロット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(SD [3]、LAN、USB)以外へのコネクタ着脱は、絶対に行わないでください。
静電気	本製品には CMOS デバイスを使用しており、静電気により破壊されるおそれがあります。本製品を開封するときは、低湿度状態にならないよう注意し、静電防止用マットの使用、導電靴や人体アースなどによる作業者の帯電防止対策、備品の放電対策、静電気対策を施された環境下で行ってください。また、本製品を保管する際は、静電気を帯びやすいビニール袋やプラスチック容器などは避け、導電袋や導電性の容器・ラックなどに収納してください。
ラッチアップ	電源および入出力からの過大なノイズやサージ、電源電圧の急激な変動等により、使用している CMOS デバイスがラッチアップを起こす可能性があります。いったんラッチアップ状態となると、電源を切断しないかぎりこの状態が維持されるため、デバイスの破損につながることがあります。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、電池の性能が充分に発揮できない場合や、電池を漏液させたり、製品を破損させるおそれがあります。

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。

[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際にはマスキングを行い、周囲の部品に半田くず、半田ボール等付着しないよう十分にご注意ください。
[3]Armadillo-610 の microSD スロットは活線挿抜非対応です

[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しての注意事項

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、「3.1.7.4. 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に実施します。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「5.9. 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 B 情報技術装置です。この装置は、住宅環境で使用することを目的としていますが、この装置がラジオやテレビジョン受信機に近接して使用されると、受信障害を引き起こすことがあります。取扱説明書に従って正しい取り扱いをして下さい。VCCI-B



Armadillo-610 と Armadillo-610 拡張ボードの組み合わせ(型番: A6100-D00Z)において、VCCI の技術基準に適合することを確認しています。新たに設計開発した拡張ボードを組み合わせる場合は、再び妨害波を測定し、VCCI 協会に届出をする必要があります。

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。



1.3. 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りたっています。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 Oracle VirtualBox 用の仮想イメージファイルです。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-610 とは

- 50mm×50mm の小型モジュール型組み込みプラットフォーム

  Armadillo-610 は、Arm Cortex-A7 コアの SoC「i.MX6ULL」(NXP セミコンダクターズ製)を搭載した、小型 CPU モジュール型の組み込みプラットフォームです。ユーザー自身が新たに拡張ボードを開発して機器に組み込むことを想定しており、設計が煩雑になりがちな CPU 周りは Armadillo-610 をそのまま使い、開発セット用拡張ボードの回路図(購入者向けに無償で提供)を参考にして設計開発が比較的簡単なインターフェース部分だけを拡張設計することにより、設計時間を節約しつつさまざまな形状の基板を実現することができます。

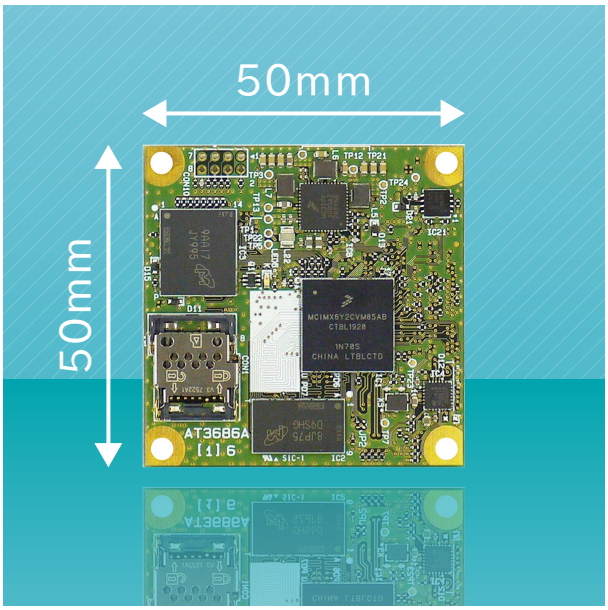


図 2.1 Armadillo-610 とは

- Armadillo-640 と同等の機能を搭載

  Armadillo-610 は、シングルボード型の組み込みプラットフォーム「Armadillo-640」の姉妹製品で、Armadillo-640 と同等のインターフェースを拡張することができます。

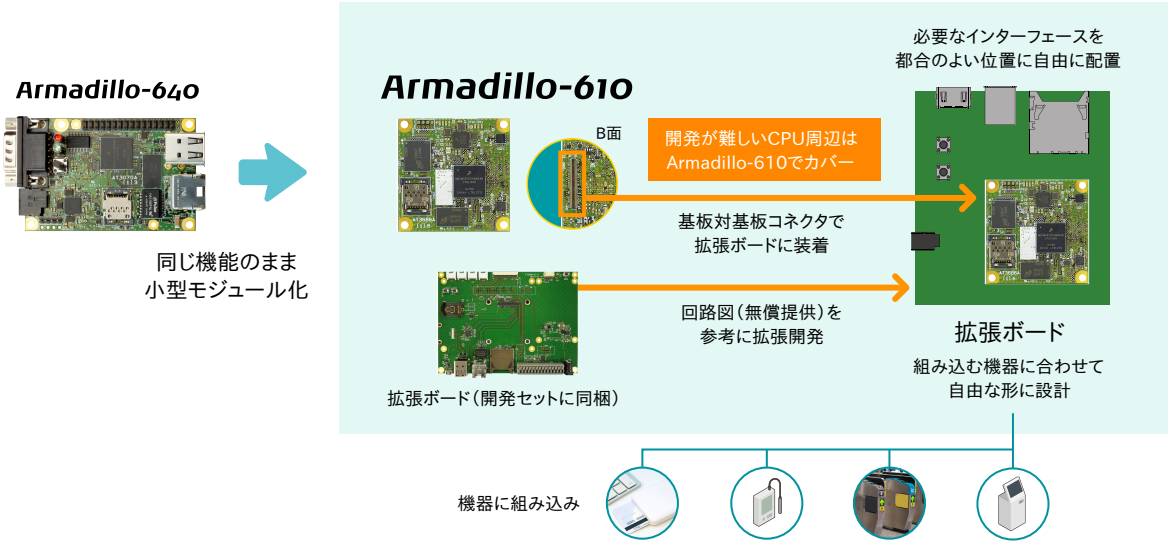


図 2.2 拡張ボードの例

- 耐環境性能に配慮・省電力、産業用途向け

  Armadillo-610 は動作温度範囲-20℃～＋ 70℃までをカバーする産業用途向けの仕様です。また、わずか数 W 程度で動作するので、低消費電力が必須条件となるバッテリ駆動の機器にも適しています。

- 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。

- Device Tree Blob（DTB）作成ツール「at-dtweb」

  使いたいインターフェースをブラウザ上の GUI で選択指定するだけで Device Tree Blob（DTB）を作成できるツールです。Linux カーネルのソースコードを手動で変更することなく GUI で選択したインターフェースを利用できます。

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。

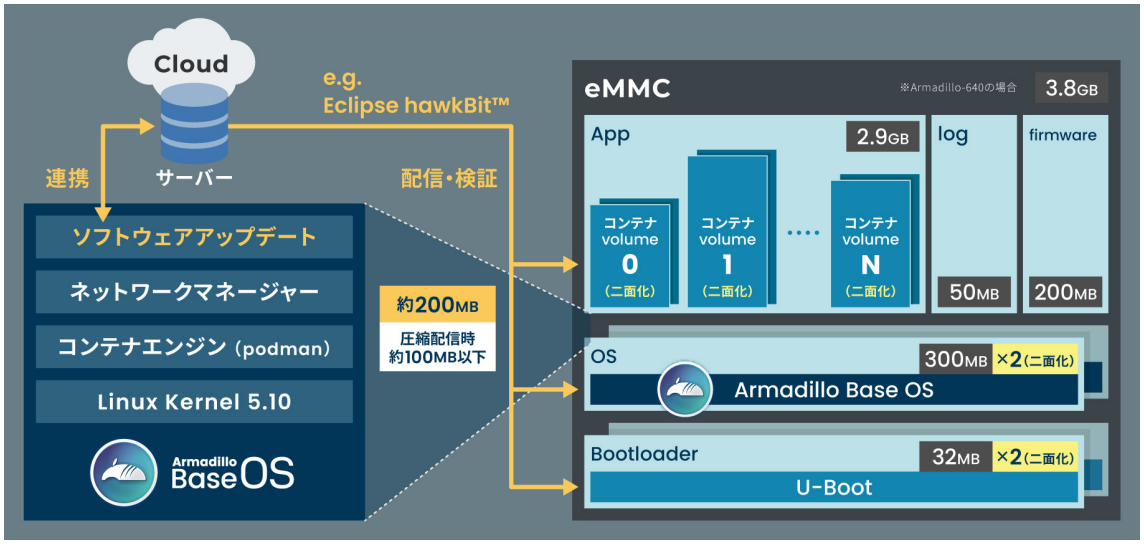


図 2.3 Armadillo Base OS とは

- 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。

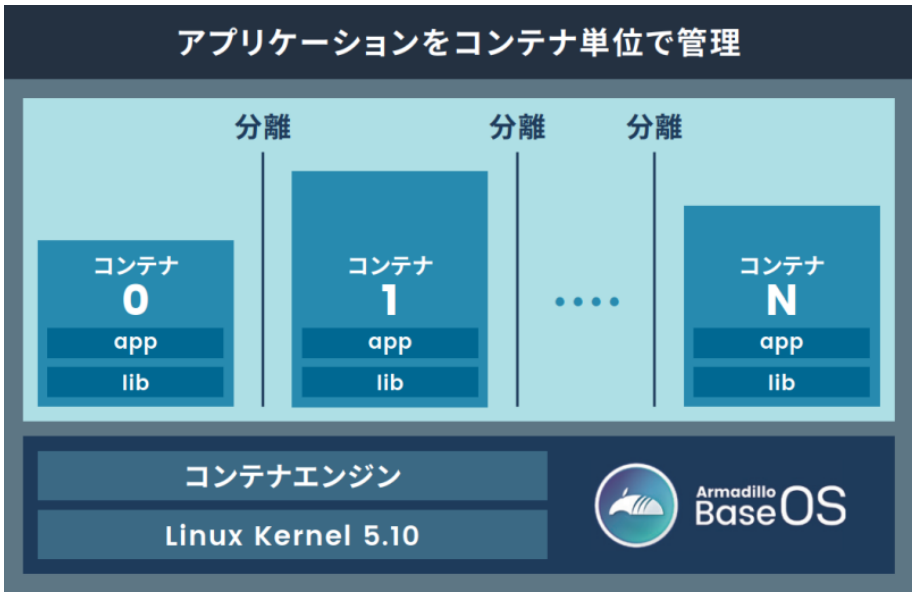


図 2.4 コンテナによるアプリケーションの運用

- 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。

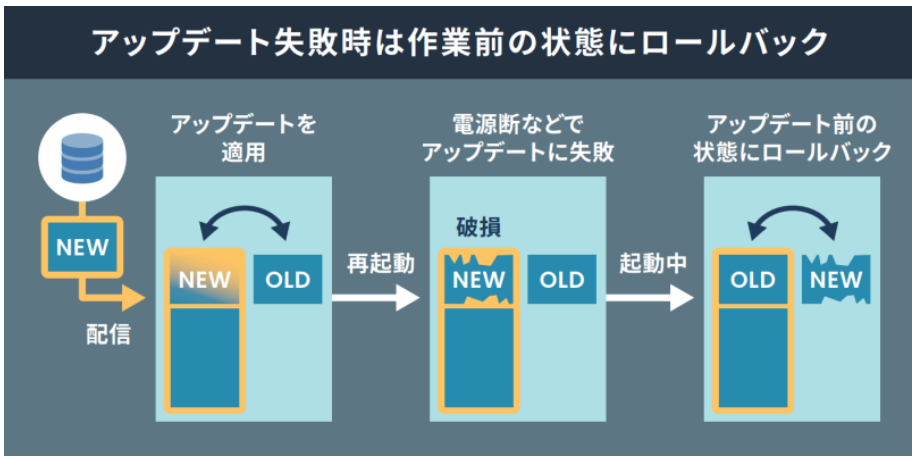


図 2.5 ロールバックの仕組み

- 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[1][2]」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しての注意事項」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クラウドサービスです。 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ます。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。

[1]Arm コア向け TEE(Trusted Execution Environment)実装の一つです (https://optee.readthedocs.io/en/latest/)
[2]ソースコードは imx-boot の imx-optee-os ディレクトリに入っています



図 2.6 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
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-610 は シリアルナンバー：00B700430001 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-610 の製品ラインアップは次のとおりです。

表 2.1 Armadillo-610 ラインアップ

名称	型番
Armadillo-610 量産ボード	A6100-U00Z
Armadillo-610 開発セット	A6100-D00Z

2.2.1. Armadillo-610 開発セット

Armadillo-610 開発セット(型番: A6100-D00Z)は、Armadillo-610 を使った開発がすぐに開始できるように、USB シリアル変換アダプタや AC アダプタなど、開発に必要なものを一式にしています。また、拡張ボードの回路図は製品購入者向けに無償で公開しています。

表 2.2 Armadillo-610 開発セットのセット内容

Armadillo-610
Armadillo-610 拡張ボード
USB(A オス-miniB)ケーブル
USB シリアル変換アダプタ
スピーカー
AC アダプタ(12V/2.0A)
ジャンパソケット
ねじ
スペーサ

2.2.2. Armadillo-610 量産ボード

Armadillo-610 量産ボード(型番: A6100-U00Z)は、量産向けのラインアップです。

表 2.3 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)
SD	microSD スロット x 1 [a]
拡張インターフェース	LANx1、USBx2、UARTx8、SDx1 [a]、LCDx1、I2Sx2、S/PDIFx1、MQSx1、I2Cx2、SPIx4、CANx2、A/Dx7、PWMx8、GPIOx66 等 [b]
カレンダー時計	SoC 内蔵リアルタイムクロック [c]
LED	黄色(面実装) x 1
セキュアエレメント	NXP Semiconductors SE050 [d]
電源電圧	DC 3.6～4.5V
消費電力(参考値)	約 0.8W(待機時)、約 1W(LAN 通信時) [e]
使用温度範囲	-20～+70℃(結露なきこと)
外形サイズ	50×50mm

[a]microSD スロットと拡張インターフェース側の SD は排他利用となります。

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

[c]電池は付属していません。

[d]シリアルナンバー：00B700430001 以降の Armadillo-610 に搭載

[e]Armadillo-610 拡張ボードとの組み合わせで LAN、シリアルコネクタにケーブルを接続した状態での消費電力です。外部接続機器の消費分は含みません。

2.3. インターフェースレイアウト

Armadillo-610 および開発セット付属の拡張ボードインターフェースレイアウトは次のとおりです。各インターフェースの配置場所等を確認してください。



コネクタの挿抜寿命を記載していますが、製品出荷時における目安であり、実際に挿抜可能な回数を保証するものではありません。また、無理な挿抜は接触不良や破損の原因となりますので、取り扱いには十分ご注意ください。

2.3.1. Armadillo-610 インターフェースレイアウト

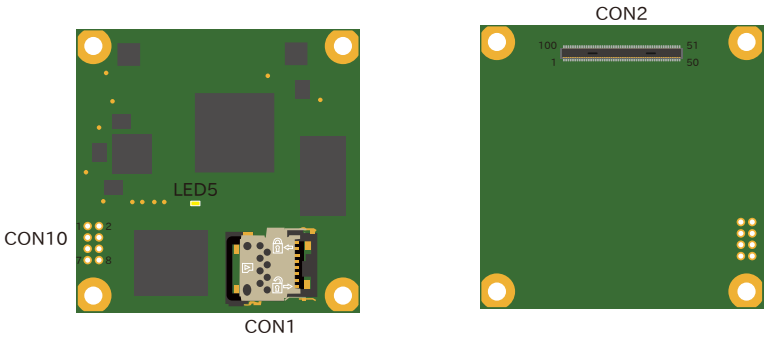


図 2.7 Armadillo-610 インターフェースレイアウト

表 2.4 Armadillo-610 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	拡張インターフェース	基板間コネクタ 100 ピン(0.4mm ピッチ)	挿抜寿命: 20 回
CON10	JTAG インターフェース	ピンヘッダ 8 ピン(2mm ピッチ)	コネクタ非搭載
LED5	ユーザー LED	LED(黄色,面実装)	

2.3.2. Armadillo-610 拡張ボード インターフェースレイアウト

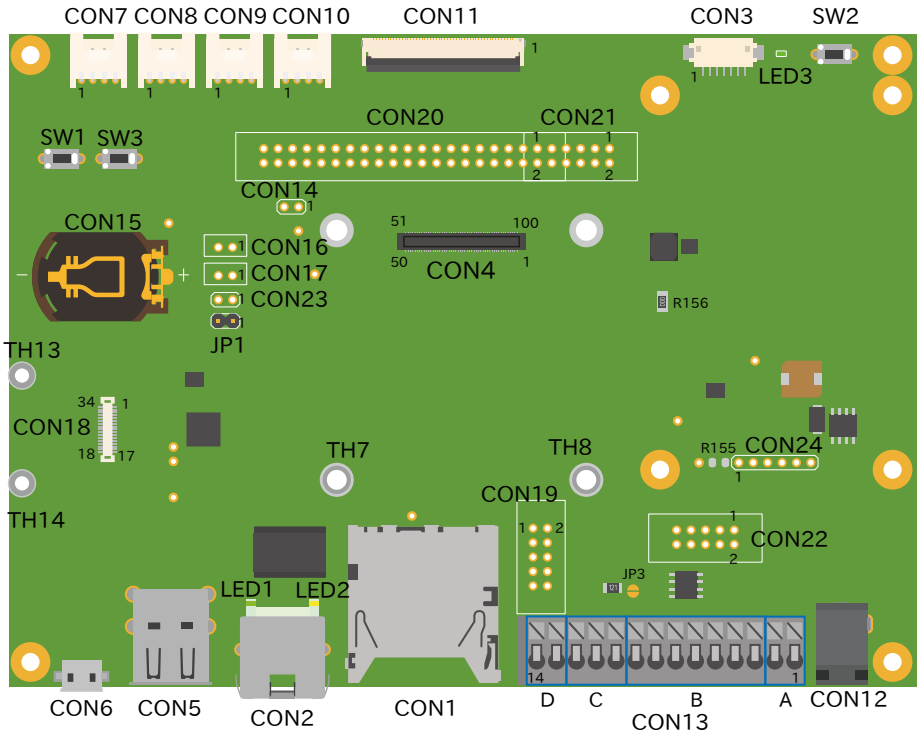


図 2.8 Armadillo-610 拡張ボード インターフェースレイアウト

表 2.5 Armadillo-610 拡張ボード 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	SD スロット	
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	ピンヘッダ 7 ピン(1.25mm ピッチ)	挿抜寿命: 40 回
CON4	Armadillo-610 インターフェース	基板間コネクタ 100 ピン(0.4mm ピッチ)	挿抜寿命: 20 回
CON5	USB ホストインターフェース	Type-A コネクタ(2 ポートスタック)	
CON6	USB OTG インターフェース	Micro-AB コネクタ	
CON7	Grove インターフェース	ピンヘッダ 4 ピン(2.54mm ピッチ)	
CON8		ピンヘッダ 4 ピン(2.54mm ピッチ)	
CON9		ピンヘッダ 4 ピン(2.54mm ピッチ)	
CON10		ピンヘッダ 4 ピン(2.54mm ピッチ)	
CON11	LCD 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
CON12	電源入力インターフェース	DC ジャック	対応プラグ: 内径 2.1 外形 5.5mm
CON13 A	電源出力インターフェース	端子台 2 ピン	

部品番号	インターフェース名	形状	備考
CON13 B	DIDO インターフェース	端子台 7 ピン	
CON13 C	RS-485 インターフェース	端子台 3 ピン	
CON13 D	オーディオインターフェース	端子台 2 ピン	
CON14	電源出力インターフェース	ピンヘッダ 2 ピン(2.54mm ピッチ)	コネクタ非搭載
CON15	RTC バックアップインターフェース	電池ボックス	対応電池: CR2032
CON16		ピンヘッダ 2 ピン(2.54mm ピッチ)	コネクタ非搭載
CON17	内蔵 RTC バックアップインターフェース	ピンヘッダ 2 ピン(2.54mm ピッチ)	コネクタ非搭載
CON18	WLAN インターフェース	基板間コネクタ 34 ピン(0.5mm ピッチ)	
CON19	拡張インターフェース	ピンヘッダ 10 ピン(2.54mm ピッチ)	コネクタ非搭載
CON20		ピンヘッダ 40 ピン(2.54mm ピッチ)	コネクタ非搭載
CON21		ピンヘッダ 10 ピン(2.54mm ピッチ)	コネクタ非搭載
CON22		ピンヘッダ 10 ピン(2.54mm ピッチ)	コネクタ非搭載
CON23	リセットインターフェース	ピンヘッダ 2 ピン(2.54mm ピッチ)	コネクタ非搭載
CON24	電源入力インターフェース	ピンヘッダ 6 ピン(2.54mm ピッチ)	コネクタ非搭載
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)	
SW1	ユーザースイッチ	タクトスイッチ	
SW2	リセットスイッチ	タクトスイッチ	
SW3	ONOFF スイッチ	タクトスイッチ	
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	LED(緑色,面実装)	
TH7	Armadillo-610 用スタッド	スペーサー M3 (L=3mm)	
TH8		スペーサー M3 (L=3mm)	
TH13	WLAN 用スタッド	スペーサー M2 (L=1.5mm)	
TH14		スペーサー M2 (L=1.5mm)	

2.4. ブロック図

Armadillo-610 のブロック図は次のとおりです。

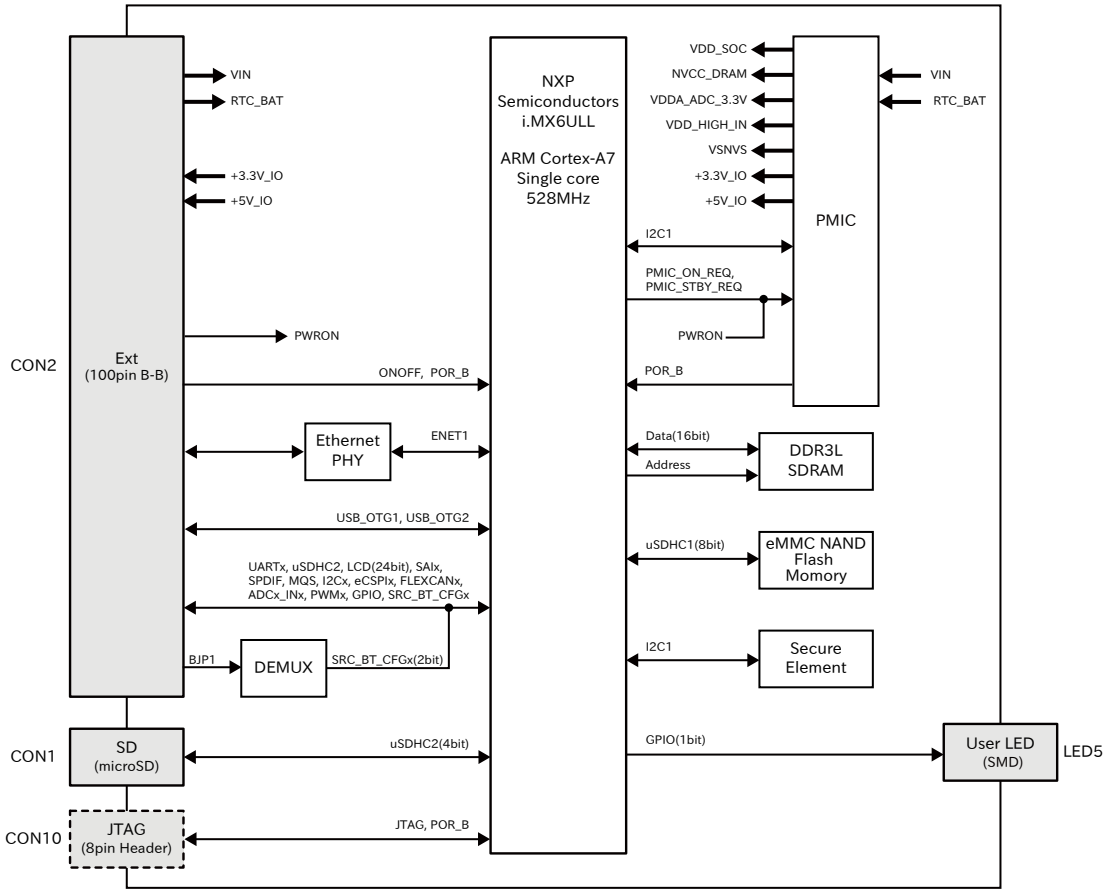


図 2.9 Armadillo-610 ブロック図

2.5. 使用可能なストレージデバイス

Armadillo-610 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 2.6 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
SD/SDHC/SDXC カード	/dev/mmcblk1	/dev/mmcblk1p1	SD インターフェース (Armadillo-610: CON1) [a] SD インターフェース (Armadillo-610 拡張ボード: CON1) [a]
USB メモリ	/dev/sd* [b]	/dev/sd*1	USB ホストインターフェース (Armadillo-610 拡張ボード: CON5)

[a]Armadillo-610: CON1 と Armadillo-610 拡張ボード: CON1 は排他利用となります。
[b]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-610 では、8 MiB の GPP を 4 つ作成しています。各領域の用途を「表 2.7. eMMC の GPP の用途」に示します。

表 2.7 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「6.23.4. ログ用パーティションについて」を参照ください。

2.6. ストレージデバイスのパーティション構成

Armadillo-610 の eMMC のパーティション構成を「表 2.8. eMMC メモリマップ」に示します。

表 2.8 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-610 の eMMC のブートパーティションの構成を「表 2.9. eMMC ブートパーティション構成」に示します。

表 2.9 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-610 の eMMC の GPP(General Purpose Partition)の構成を「表 2.10. eMMC GPP 構成」に示します。

表 2.10 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「6.23.4. ログ用パーティションについて」を参照ください。

2.7. 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-610 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きます。

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-610 開発セット一式	詳しくは「表 2.2. Armadillo-610 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。
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 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 VirtualBox には以下の特徴があります。

- 基本パッケージが GPLv3(GNU General Public License Version 3) で提供されている
- 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-610 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-610 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

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 [1]。

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。

[1]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。

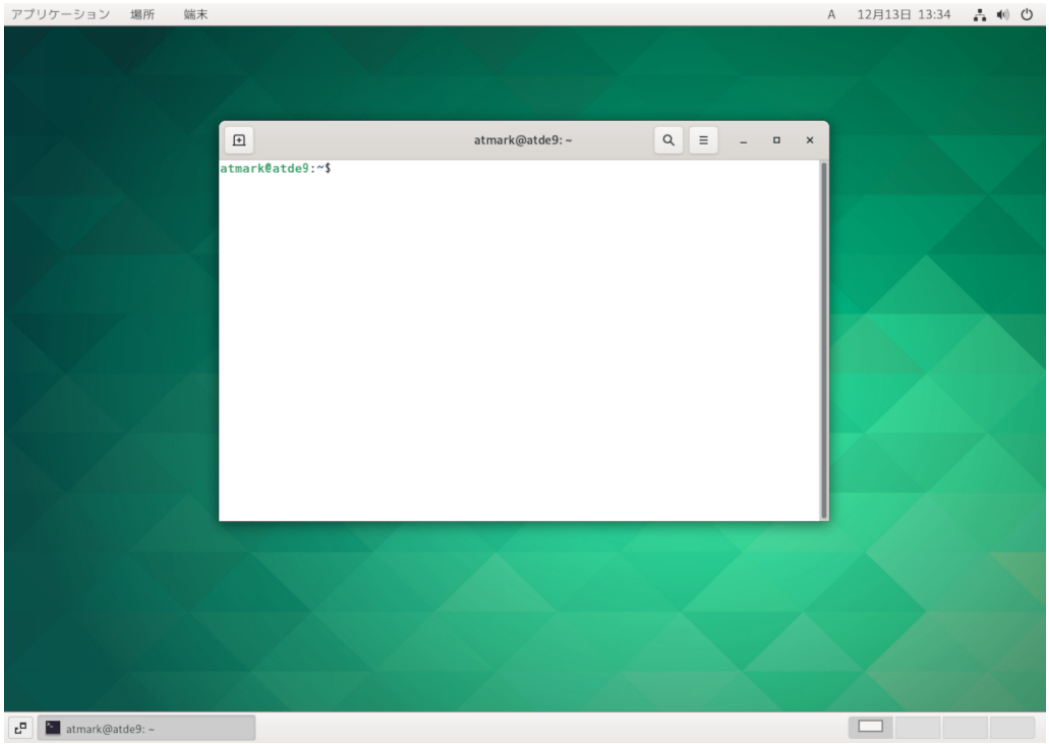


図 3.2 GNOME 端末のウィンドウ

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。

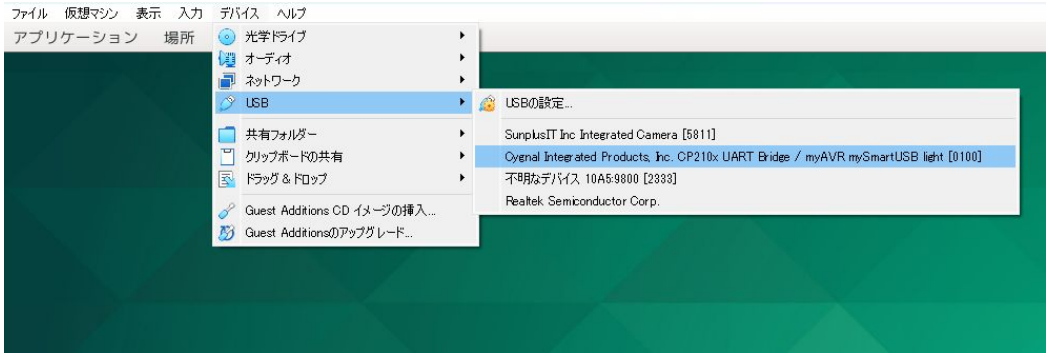


図 3.4 ATDE にデバイスを接続する

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[デバイス]-[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。

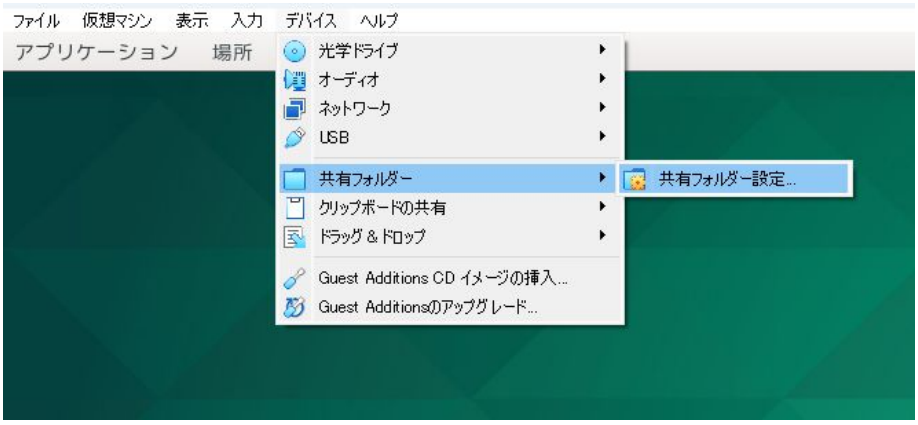


図 3.5 共有フォルダー設定を開く

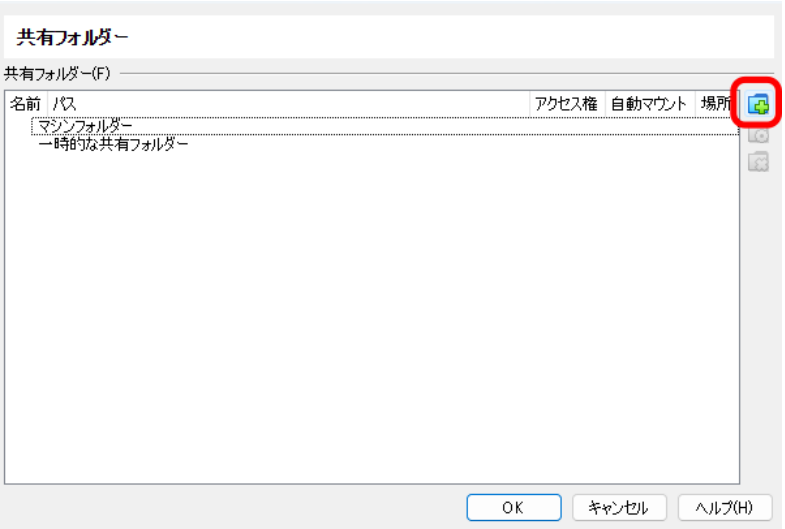


図 3.6 共有フォルダー設定

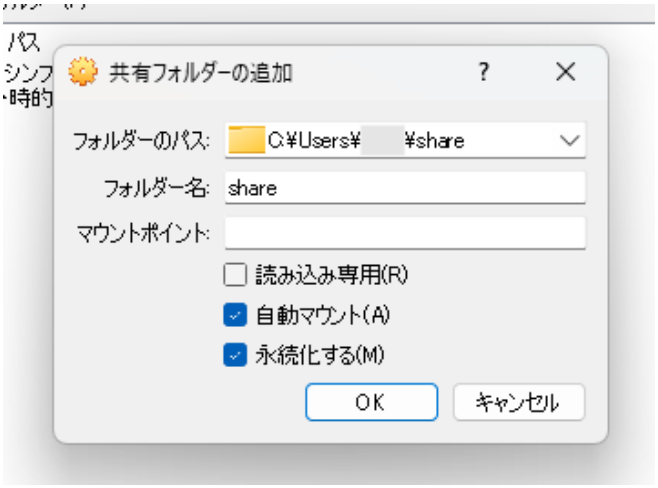


図 3.7 共有フォルダーの追加



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-610 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くることがあります。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。

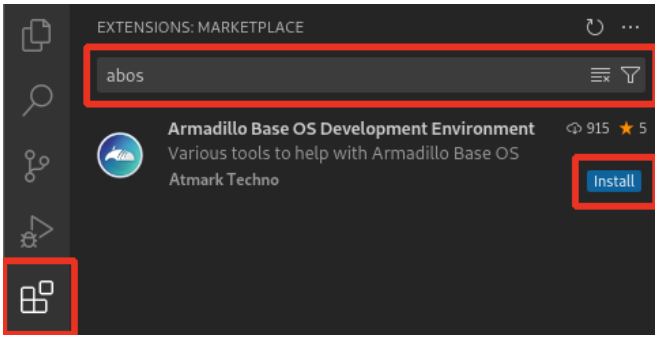


図 3.10 VS Code に開発用エクステンションをインストールする

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

初期化インストールディスクイメージを microSD カードに書き込み、初期化インストールディスクを作成します。ここでは、Windows で作成する方法を紹介します。(ATDE や Linux で実施する方法については「6.24. ATDE・Linux でインストールディスクを作成する」を参照してください)

1. 1 GB 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ダウンロードします。Armadillo-610 インストールディスクイメージ [https://armadillo.atmark-techno.com/resources/software/armadillo-610/abos-disc-image] から「Armadillo Base OS インストールディスクイメージ」をダウンロードしてください。

3. ダウンロードした zip ファイルを展開します。図のとおり、zip ファイルを選択して右クリックし、「すべて展開…」をしてください。

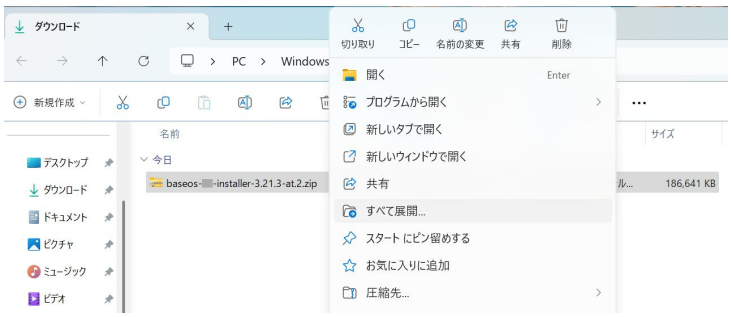


図 3.11 zip ファイルを展開

4. Win32 Disk Imager Renewal（ https://github.com/dnobori/DN-Win32DiskImagerRenewal/releases/）をダウンロードして起動します。

5. PC に microSD カードを接続します。

6. [Image File] に先ほど展開したフォルダ内の img ファイルを指定し、[Device] からは microSD カードに対応するものを選びます。



[Device]の選択には細心の注意を払ってください。他に接続されているデバイスに意図せず書き込んでしまった場合、そのデバイスのデータが上書きされてしまいます。選んだ[Device]が目的の microSD カードなのかどうか、しっかりと確認してください。microSD カードやカードリーダーの挿抜による[Device]の表示の変化で確かめることもできます。

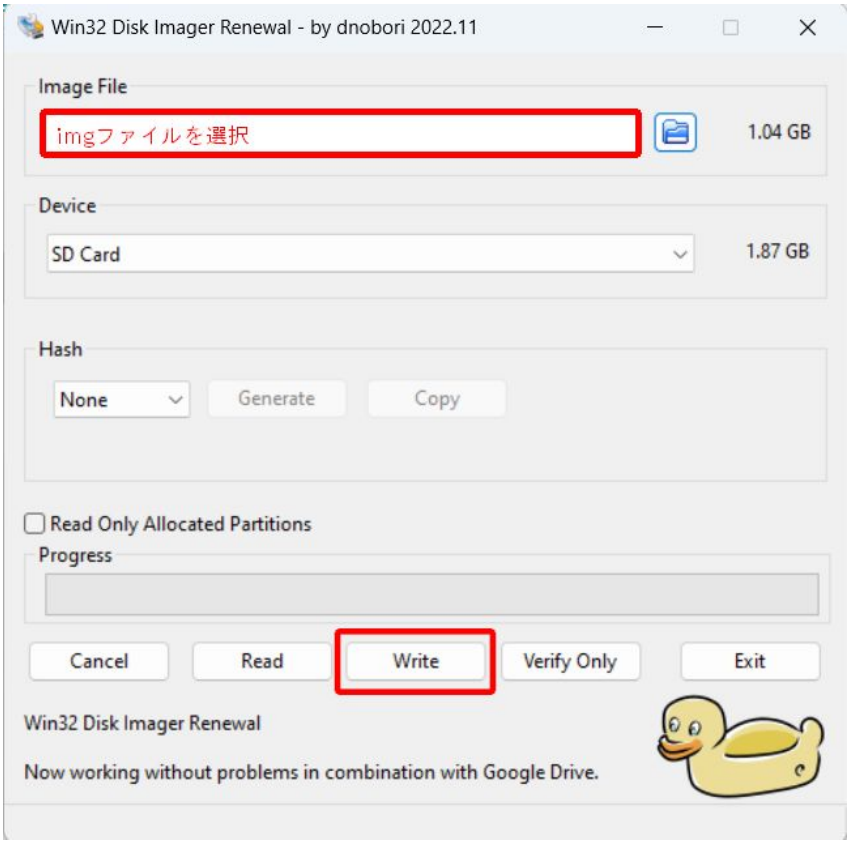


図 3.12 Win32 Disk Imager Renewal 設定画面

7. [Write] ボタンを押すと警告が出ますので、問題なければ[はい]を選択して書き込みます。
8. 書き込み終了ダイアログが表示されたらインストールディスクの作成は完了です。microSD カードを取り外してください。

3.1.4.2. Armadillo-610 開発セットの組み立て

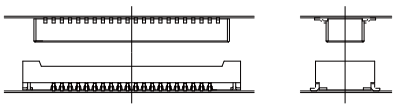
インストールディスクを使用する準備として、以下の手順を参照して Armadillo-610 拡張ボードに Armadillo-610 を組み付けてください。

Armadillo-610 拡張ボードの 4 隅のねじ穴は、スペーサー固定用です。Armadillo-610 開発セットに同梱されている M3 のねじとスペーサーを取り付けてください。

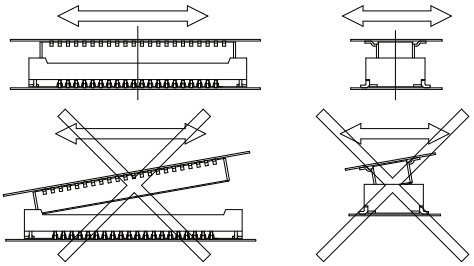


コネクタ嵌合時の注意

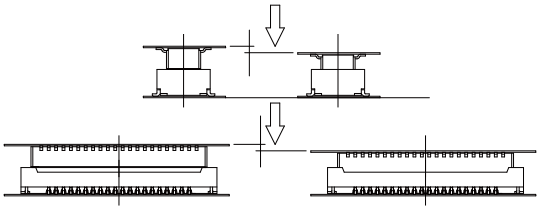
コネクタの中心を合わせて嵌合してください。



位置合わせをする際は、無理な力を加えることなく誘い込み口を探してください。無理な力を加えると、モールドの破損、削れが発生し、接触抵抗の不具合等に繋る場合があります。



コネクタが誘い込まれると、コネクタ間の距離が近くなり、平行になって前後左右に動かなくなります。この状態からまっすぐに嵌合してください。



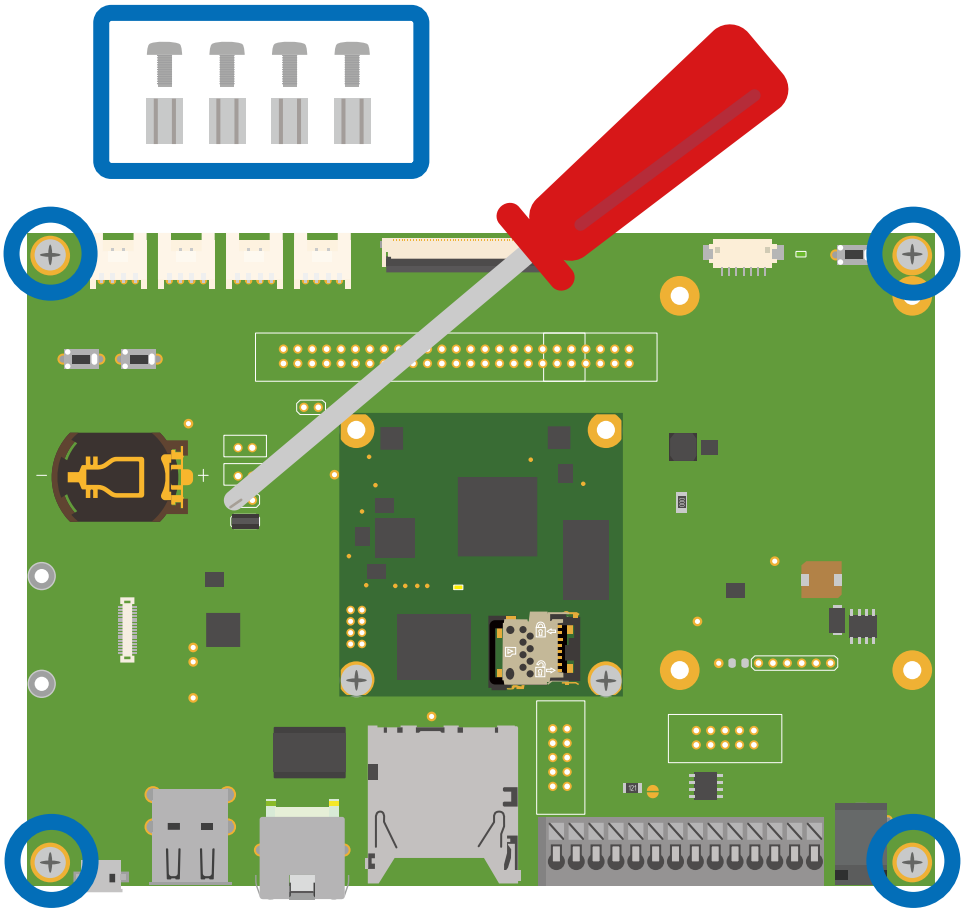


図 3.13 Armadillo-610 開発セットの組み立て

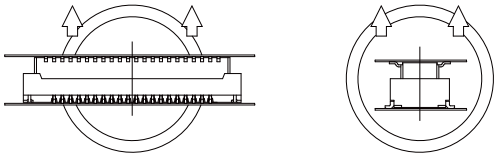


Armadillo-610 開発セットでは、Armadillo-610 は Armadillo-610 拡張ボードに搭載した状態で出荷されます。M2 のねじが同梱されていますが、こちらは Armadillo-WLAN(AWL13)固定用としてご使用ください。

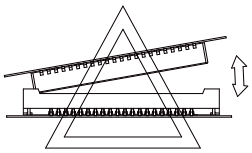


コネクタ抜去時の注意

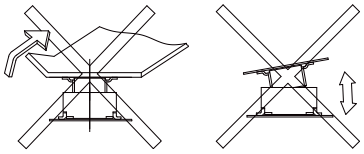
コネクタは平行に抜去してください。



平行に抜去することが困難な場合、コネクタ幅の狭い方向から斜めに抜去してください。



コネクタが損傷する可能性が高いため、コネクタのコーナー方向や幅の広い方向から斜めに抜去しないでください。





AC アダプタから電源を供給する際、DC プラグを Armadillo-610 拡張ボードの DC ジャックに接続してから、AC プラグをコンセントに接続してください。AC プラグをコンセントに接続してから Armadillo-610 拡張ボードの DC ジャックに接続すると、突入電流により故障する可能性があります。

3.1.4.3. インストールディスクを使用する

作成したインストールディスクを使用して、インストールを実施します。

以下の手順に沿って、 「図 3.14. Armadillo-610 を初期化する接続」 のとおりに接続します。

1. 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。
2. JP1 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)

3. AC アダプタを接続して電源を投入すると、数秒経過後に LED5 が点灯・点滅します。
4. 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(LED5 が消灯)。
5. LED5 が消灯したら、AC アダプタを取り外し、続いて JP1 のジャンパーソケットと microSD カードを外してください。

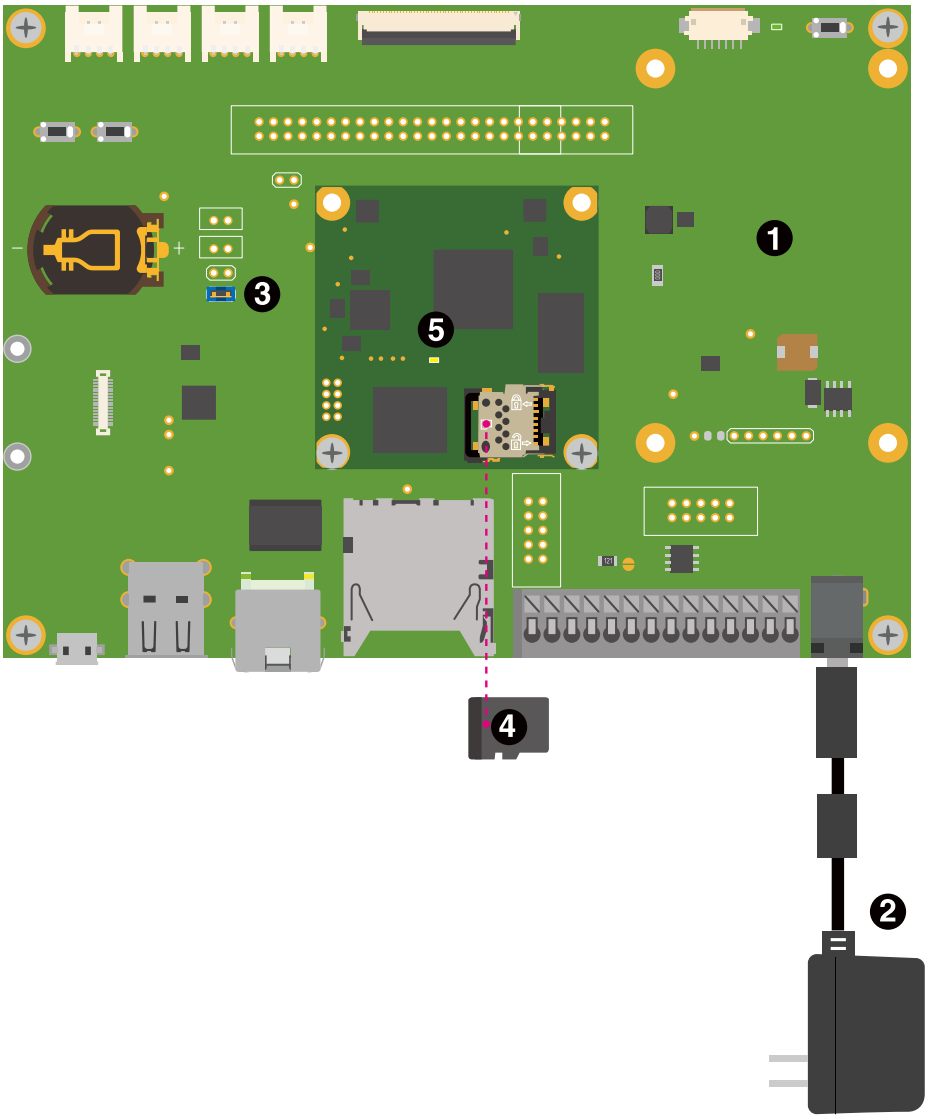


図 3.14 Armadillo-610 を初期化する接続

❶ Armadillo-610

❷ AC アダプタ(12V/2.0A)

❸ ジャンパーソケット

❹ microSD カード

❺ LED5



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。

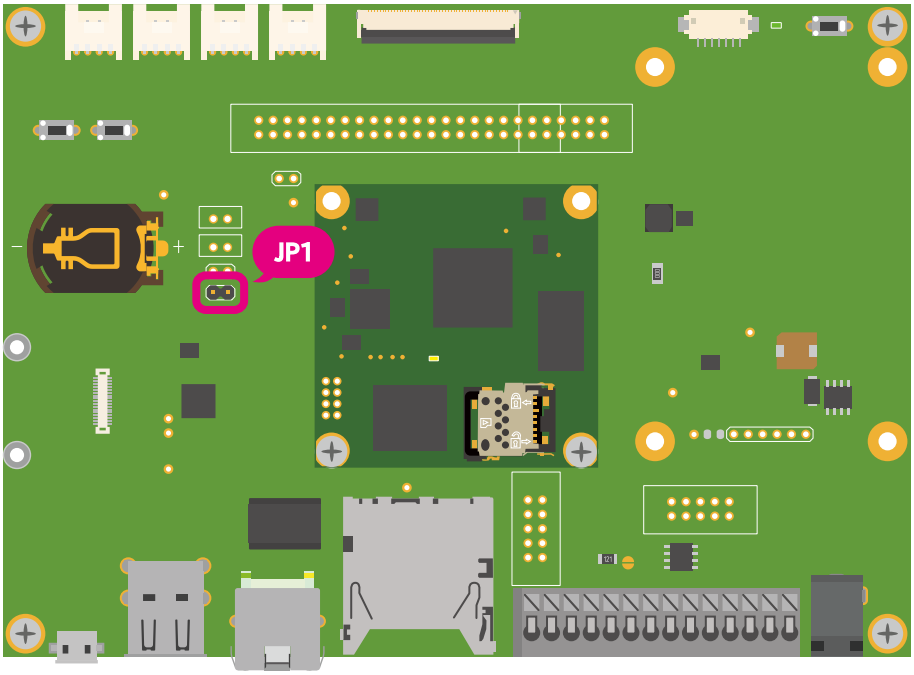


図 3.15 ジャンパピンの位置

ジャンパの設定は 「図 3.15. 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-610 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「6.29.2.28. JP1(起動デバイス設定ジャンパ)」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 3.16. 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。

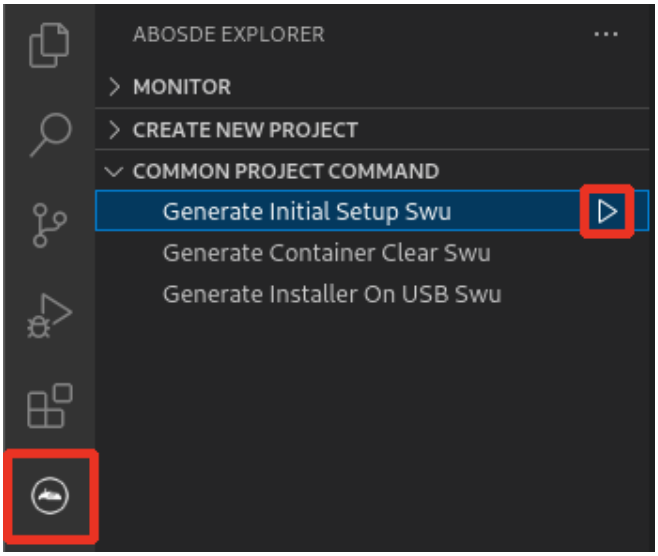


図 3.16 initial_setup.swu を作成する

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 3.17. 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
証明書のコモンネーム(一般名)を入力してください: [COMMON_NAME] ❶
証明書の鍵のパスワードを入力ください（4-1024 文字）❷
証明書の鍵のパスワード（確認）：
Generating an EC private key
writing new private key to '/home/atmark/mkswu/swupdate.key.tmp'
 -----
アップデートイメージを暗号化しますか？（N/y）❸
アットマークテクノが作成したイメージをインストール可能にしますか？（Y/n）❹
root パスワード: ❺
root 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）：❻
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) ❼
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）: ❽
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 ❾
```

図 3.17 initial_setup.swu 初回生成時の各種設定

❶ COMMON_NAME には証明鍵の「common name」として会社や製品が分かるような任意の名称を入力してください。

❷ 証明鍵を保護するパスフレーズを 2 回入力します。

❸ SWU イメージ自体を暗号化する場合に「y」を入力します。詳細は 「6.7. SWUpdate と暗号化について」 を参考にしてください。

❹ アットマークテクノのアップデートをインストールしない場合は「n」を入力します。

❺ root のパスワードを 2 回入力します。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

❻ atmark ユーザーのパスワードを 2 回入力します。何も入力しない場合はユーザーをロックします。使用できるパスワードの制限は root と同様です。

❼ 自動アップデートを無効のままで進みます。ここで「y」を入れると、定期的にアットマークテクノのサーバーからアップデートの有無を確認し、自動的にインストールします。

❽ 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。

❾ 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 3.18. ABOS にアクセスするための接続」 のとおりに Armadillo に配線を行い、電源を入れてください。

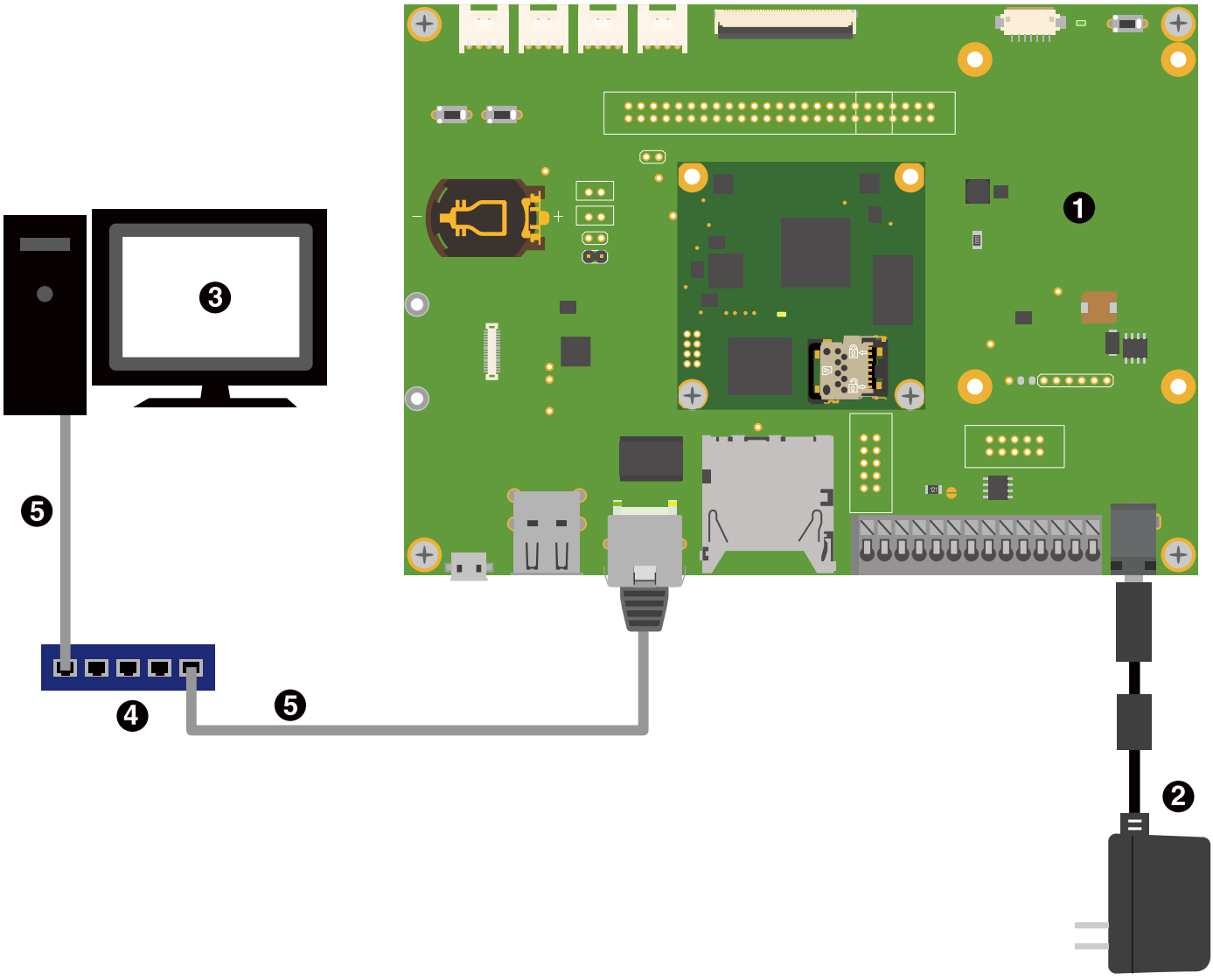


図 3.18 ABOS にアクセスするための接続

1. Armadillo-610
2. AC アダプタ(12V/2.0A)
3. 作業用 PC
4. LAN HUB

❺ Ethernet ケーブル

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 3.19. 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 3.20. 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 3.20. 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。

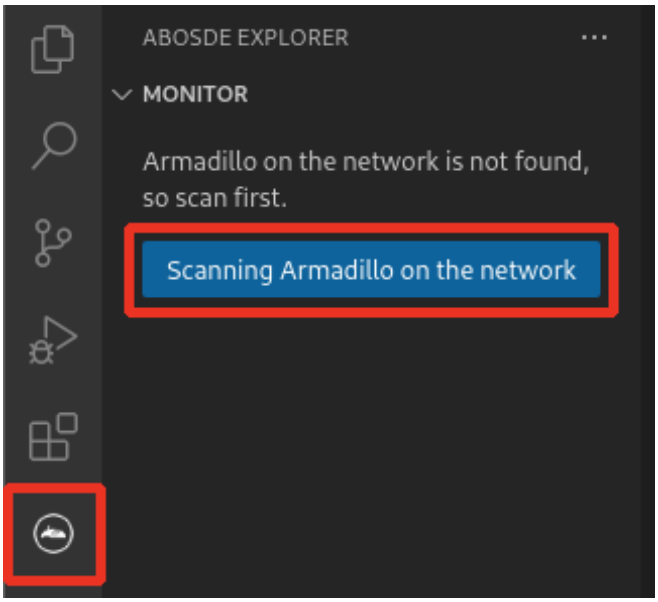


図 3.19 ABOSDE で ローカルネットワーク上の Armadillo をスキャンする

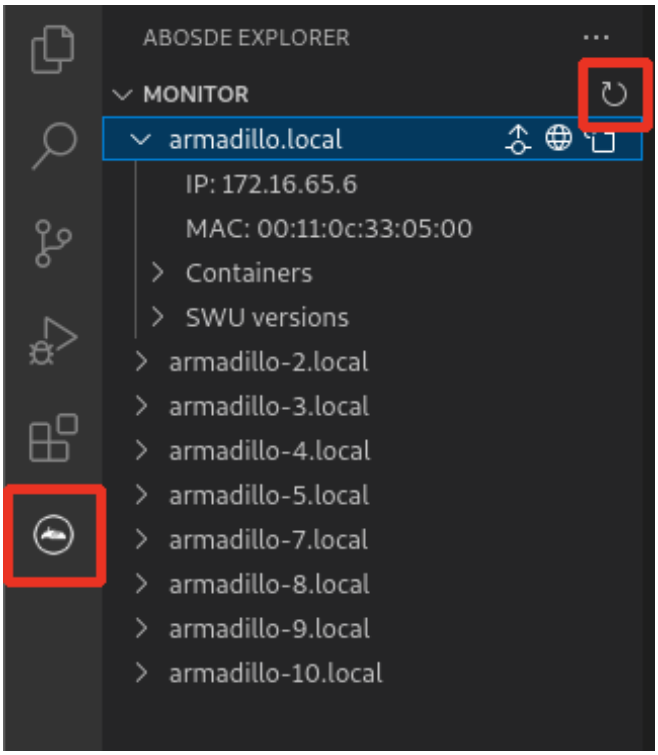


図 3.20 ABOSDE に表示されている Armadillo を更新する

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 3.21. 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。

また、 Armadillo を社内 LAN やインターネットに直接に接続できないなどの場合は、ATDE を社内 LAN（インターネット）に接続しながら Armadillo とリンクローカルアドレスで 1 対 1 の接続を行うこともできます。詳細はブログ 「Armadillo Base OS：Armadillo をインターネットに繋げない環境でのソフトウェア開発を行うために」 [https://armadillo.atmark-techno.com/blog/10899/25537/]をご参照ください。

続いて、「図 3.21. 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。

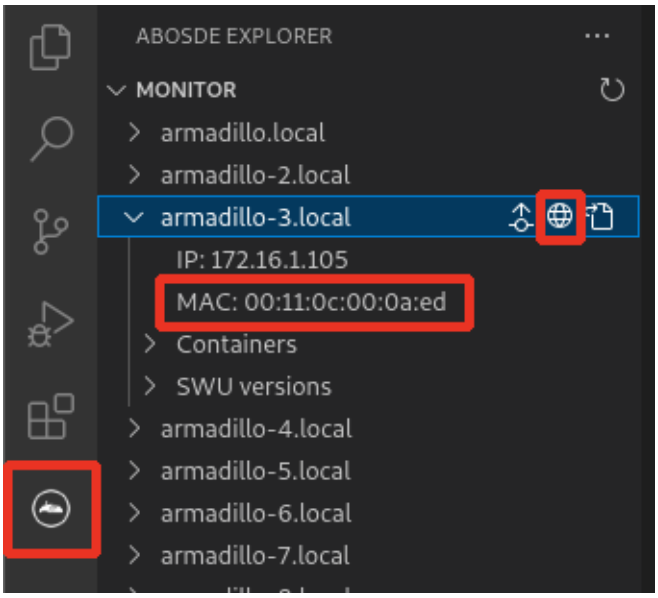


図 3.21 ABOSDE を使用して ABOS Web を開く

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 3.22. 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 3.22 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 3.23 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 3.24 ログイン画面

ログインに成功すると、ABOS Web の設定画面（ 「図 3.25. 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。

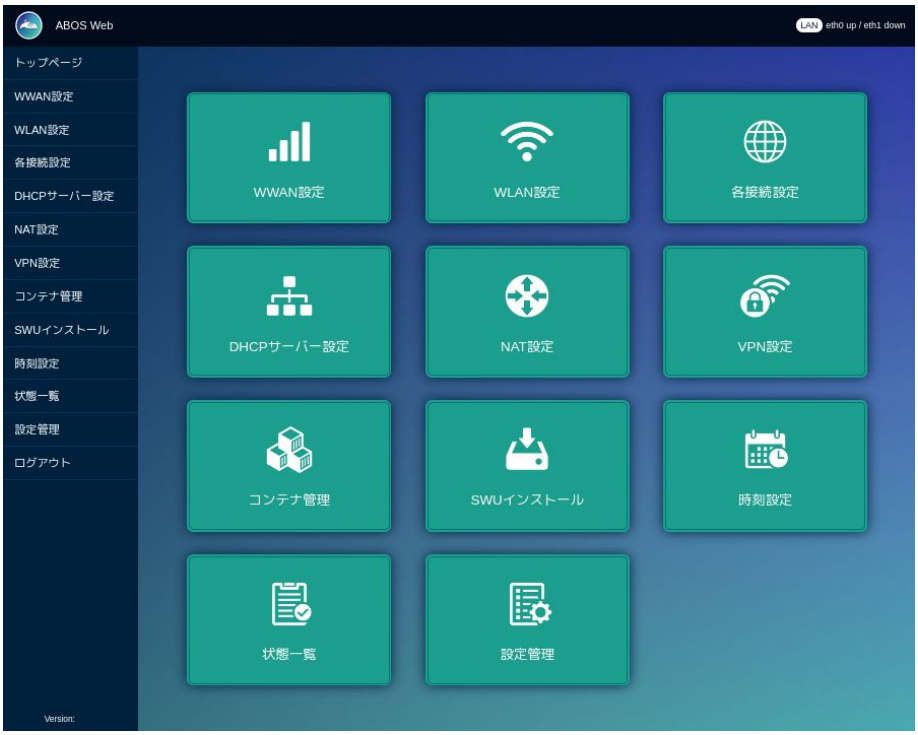


図 3.25 トップページ

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 3.26. SWU インストール」の画面に遷移します。



図 3.26 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 3.27. 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 3.27 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://192.0.2.80:5808

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`

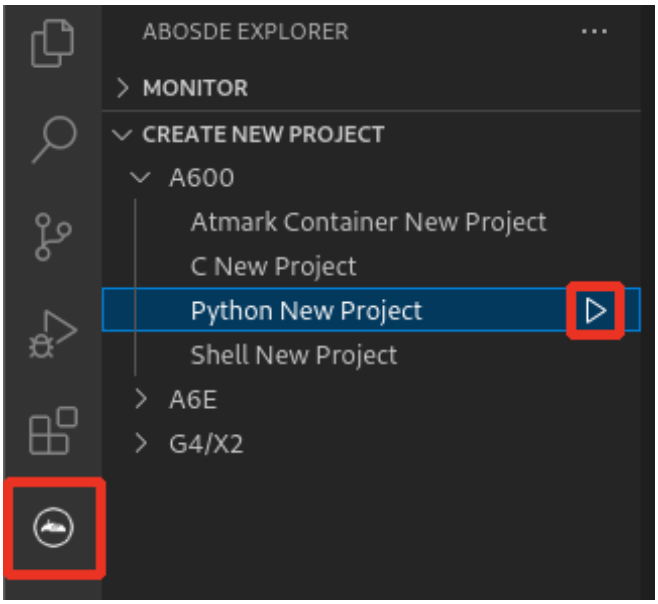


図 3.28 プロジェクトを作成する

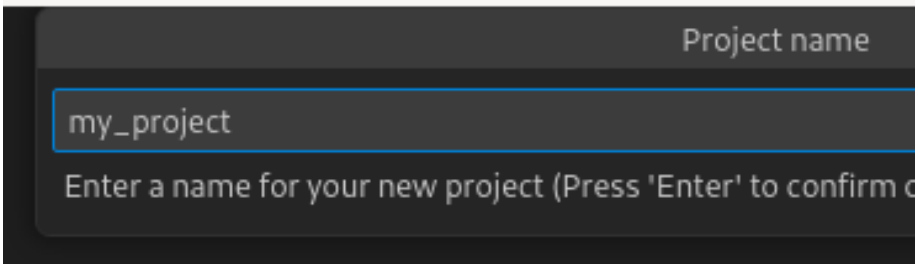


図 3.29 プロジェクト名を入力する

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。

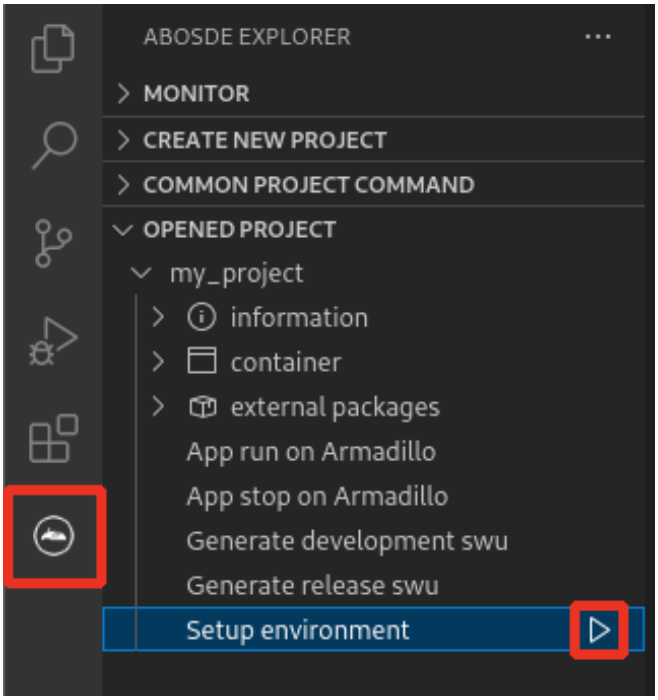


図 3.30 VS Code で初期設定を行う

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。

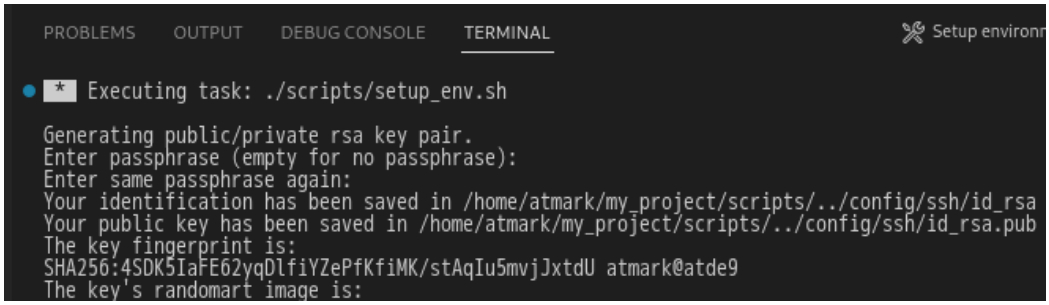


図 3.31 VS Code のターミナル

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 3.32 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。

3.1.6.3. 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。

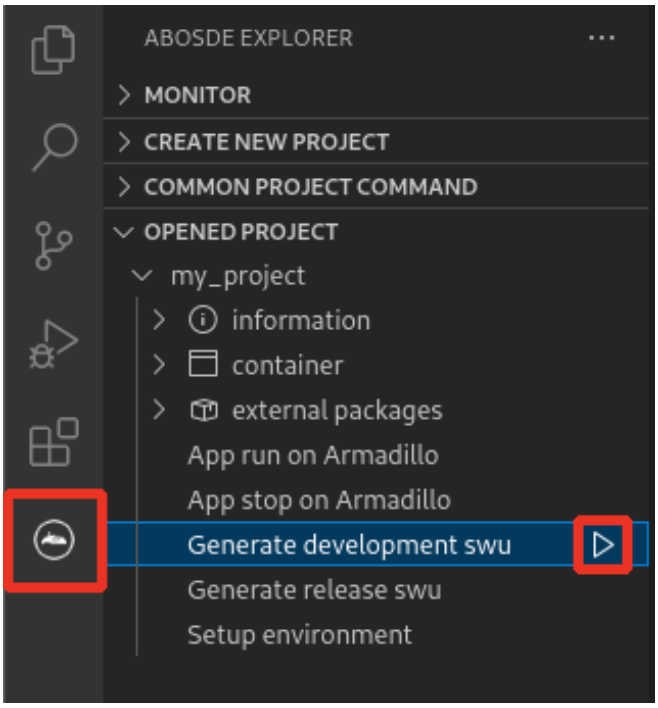


図 3.33 VS Code でコンテナイメージの作成を行う

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 3.34 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 3.35. 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。

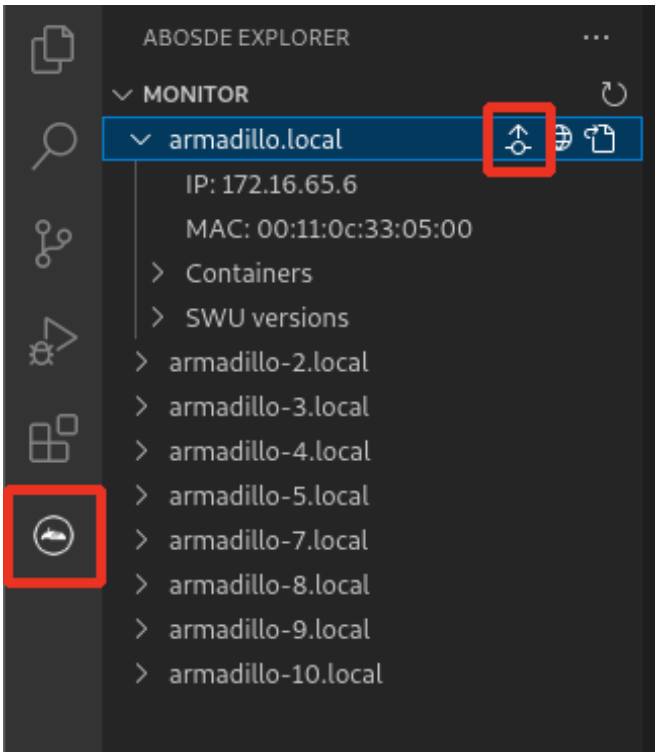


図 3.35 ABOSDE で Armadillo に SWU をインストール

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 3.19. 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 3.20. 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 3.36. 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。

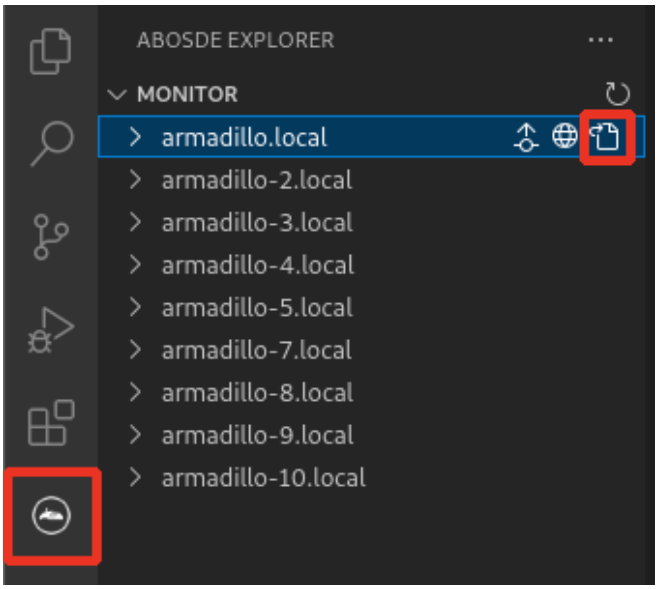


図 3.36 ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する

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 3.37 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。

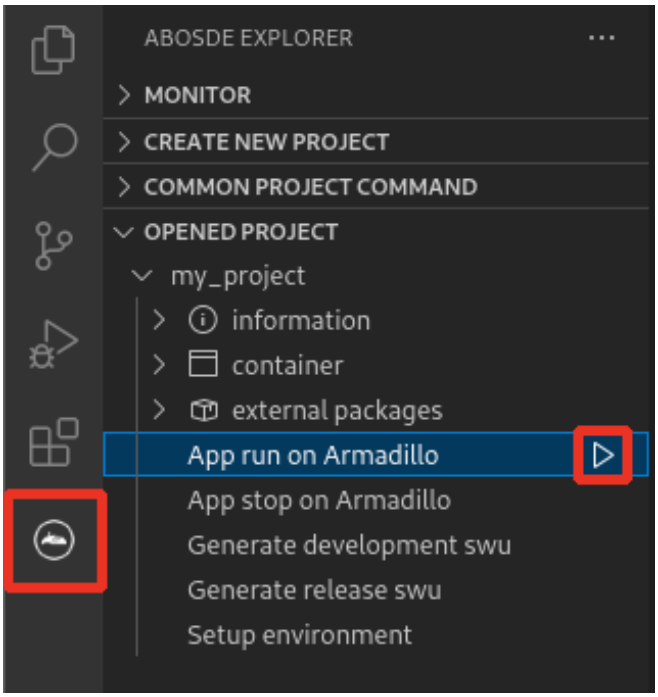


図 3.38 Armadillo 上でアプリケーションを実行する

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 3.39 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。

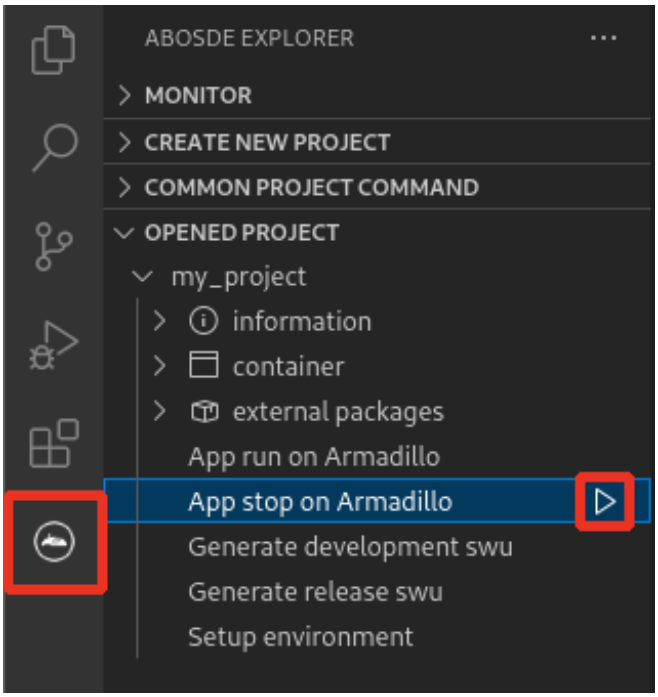


図 3.40 アプリケーションを終了する

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。

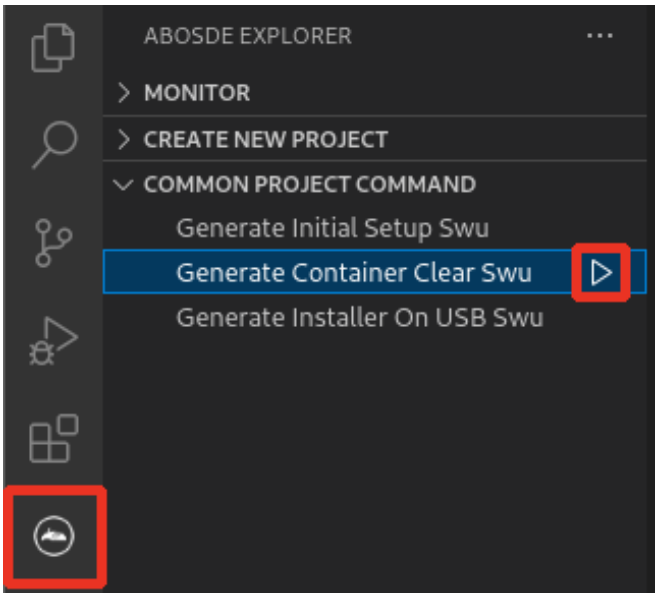


図 3.41 Armadillo 上のコンテナイメージを削除する

「図 3.35. 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(Tera Term)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。

3.1.7.1. Armadillo と開発用 PC を接続

Armadillo-610 のシリアルコンソールを使用するために、 「図 3.44. シリアルコンソールを使用する配線例」 のとおりに配線を行ってください。この配線図は Armadillo-610 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。

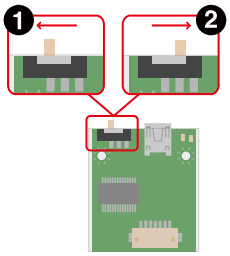


図 3.42 スライドスイッチの設定

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

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

USB シリアル変換アダプタが未接続の場合はオートブートモードとなり、Linux が起動します。また、ユーザースイッチ(Armadillo-610 拡張ボード: SW1)によっても起動モードが変わります。



シリアルインターフェース(Armadillo-610 拡張ボード: CON3)に USB シリアル変換アダプタを接続する際は、ケーブルの根本を軽く握り、指先でコネクタを押すようにして挿入してください。取り外しの際は、全ケーブルが均等に引きぬかれるようにケーブルをつかみ、引き抜いてください。また、両コネクタを水平にして挿入・抜去してください。30°以上傾けた状態での斜め挿入・抜去は、端子変形、ケース破損の原因となります。

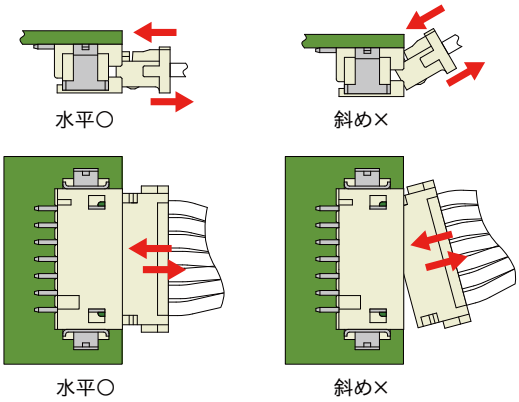


図 3.43 USB シリアル変換アダプタの挿抜角度

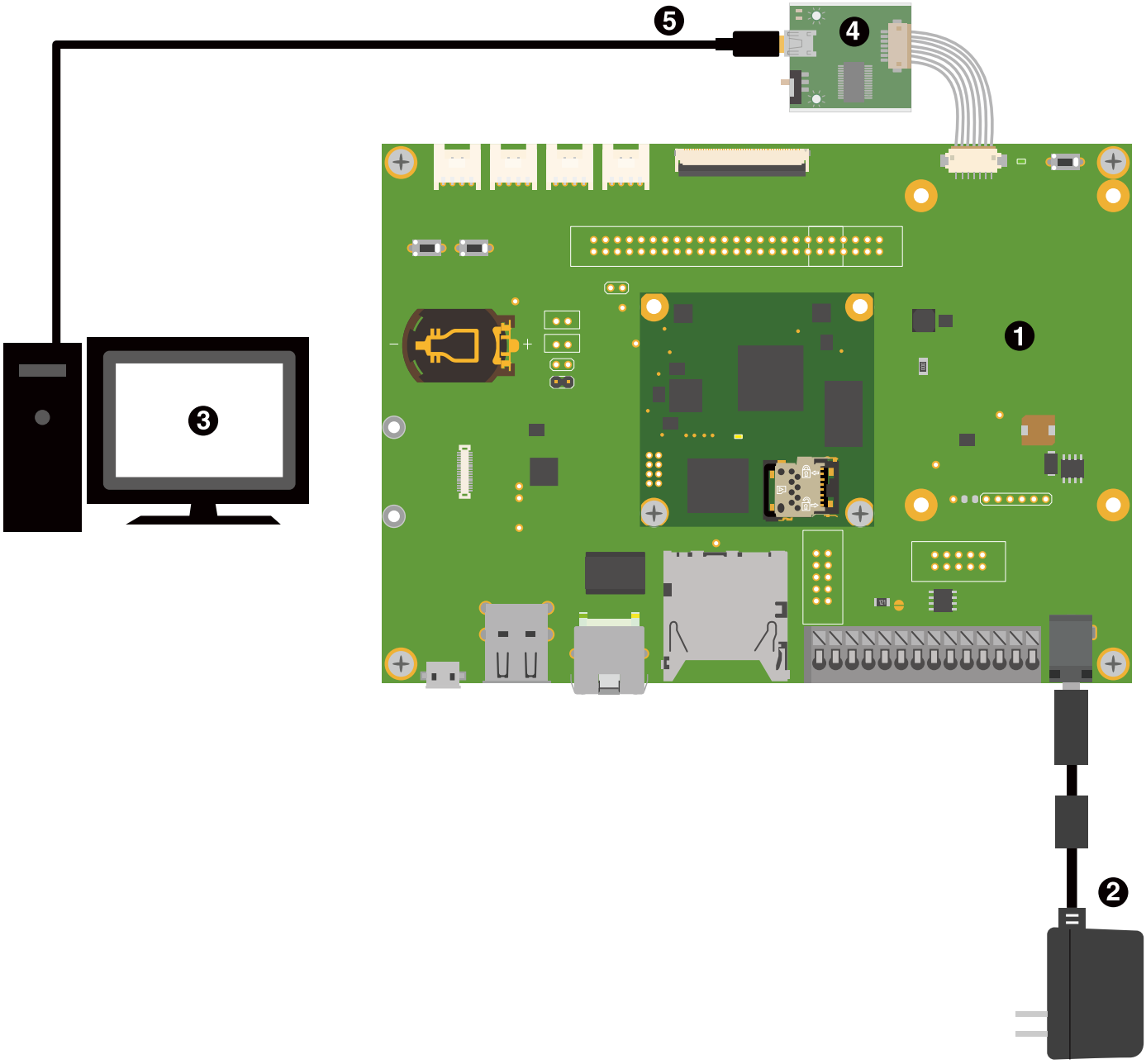


図 3.44 シリアルコンソールを使用する配線例

❶ Armadillo-610

❷ AC アダプタ(12V/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)



作業用 PC が Windows の場合、一部の BT 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。

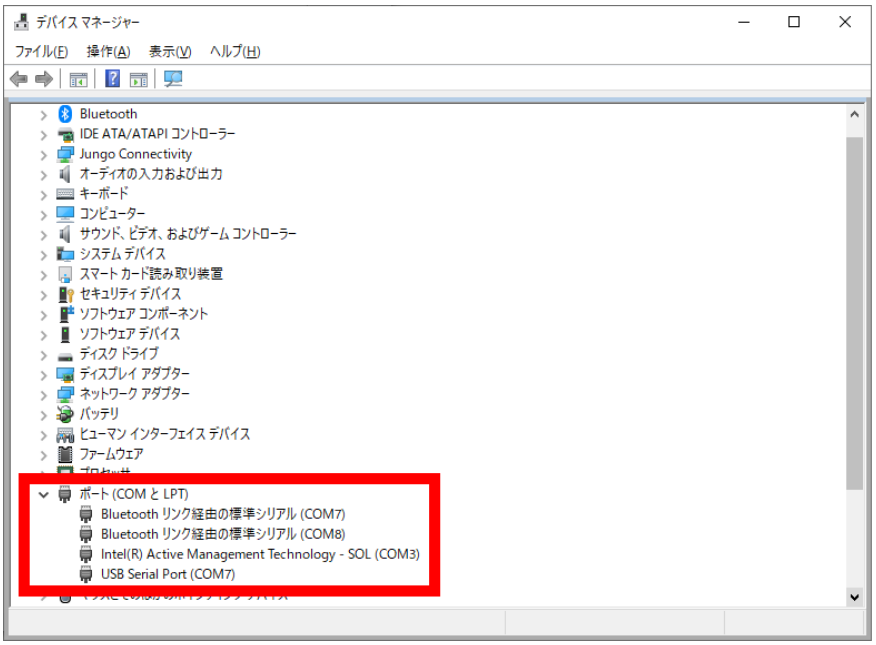


図 3.45 COM7 が競合している状態

この場合は、デバイスマネージャーから BT 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、BT デバイスを無効にしてください。

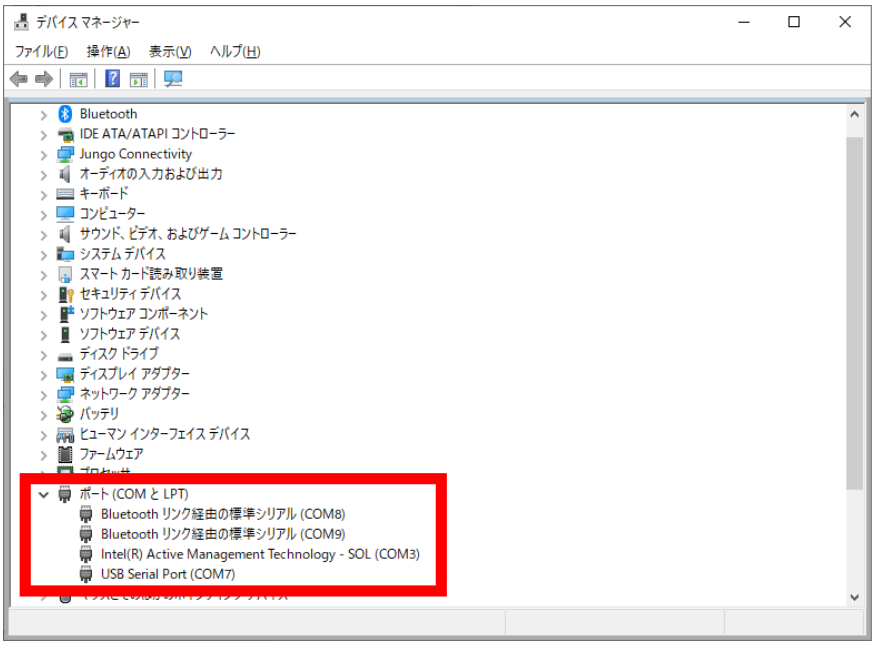


図 3.46 BT に割当の COM を変更した状態

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. Tera Term の起動

本項では Windows 環境で Tera Term 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。(Linux や ATDE の場合は 「6.25. シリアル通信ソフトウェア(minicom)」 を参照してください。)

1. **Tera Term**（https://github.com/TeraTermProject/teraterm/releases）をダウンロード・インストールして起動します。

2. 下図のように、[新しい接続]ダイアログに Armadillo を接続している COM ポートを指定して[OK]をクリックします。ダイアログが表示されていない場合は、[ファイル]-[新しい接続]をクリックして表示します。

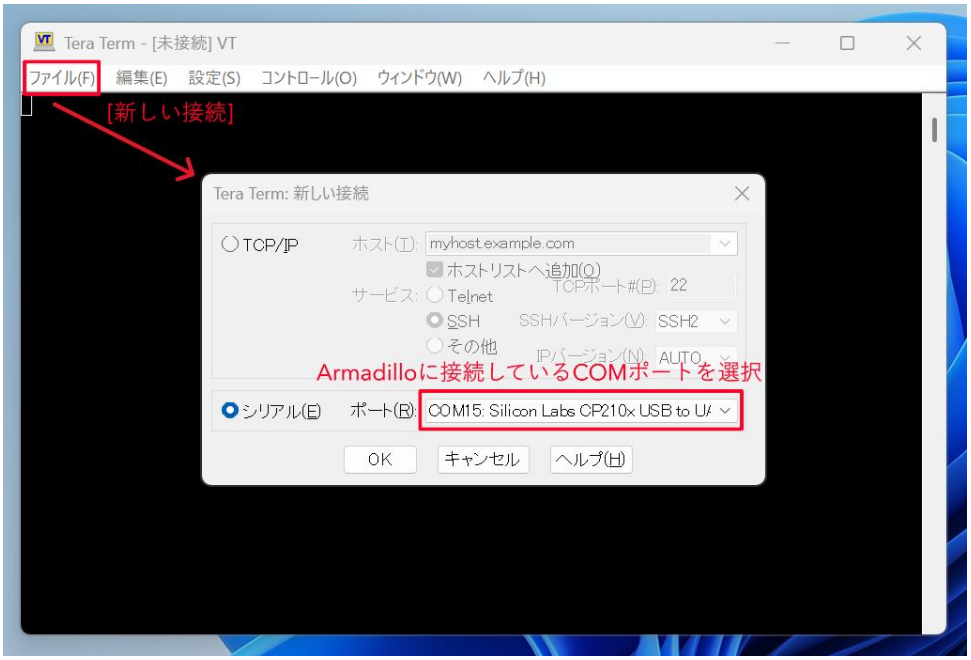


図 3.47 Armadillo を接続している COM ポートを指定

3. [設定]-[シリアルポート]をクリックして、Armadillo を接続しているシリアルポートの設定を下図のように行い、[OK]をクリックします。

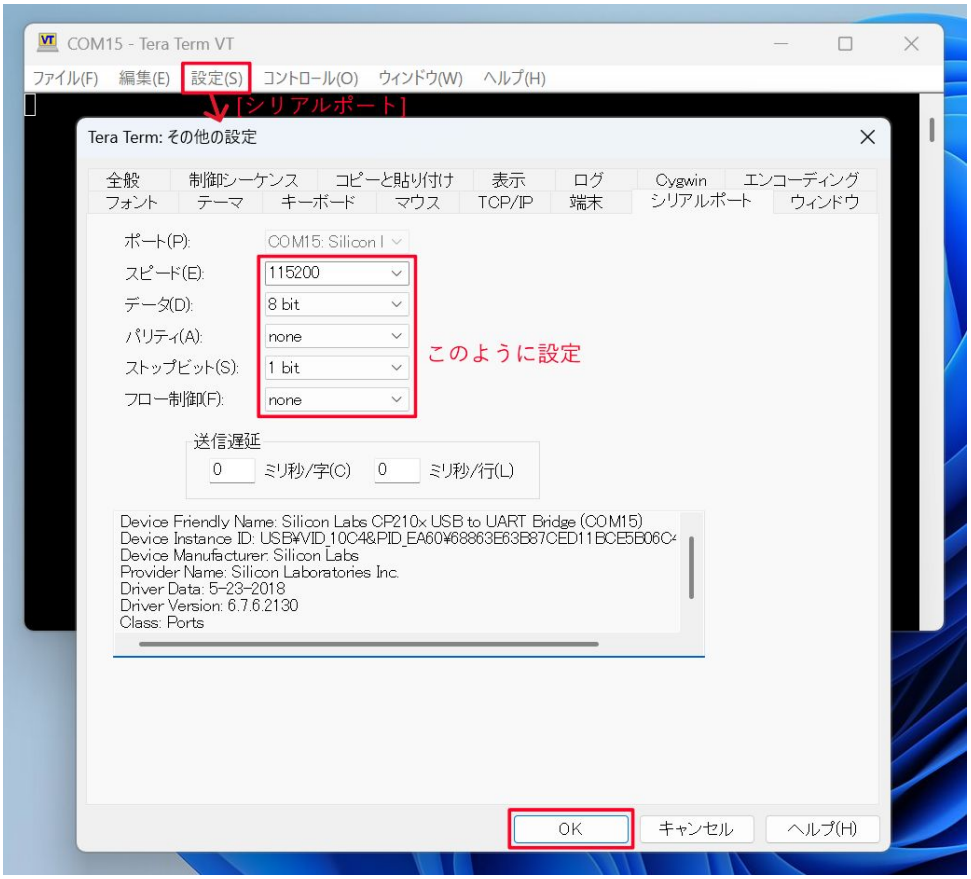


図 3.48 Armadillo を接続しているシリアルポートの設定

3.1.7.3. Armadillo の起動

電源を接続して Armadillo-610 を起動してください。シリアルコンソールに以下のような起動ログが出力されます。

```
U-Boot 2020.04-at16 (Jun 28 2023 - 16:16:51 +0900)

CPU:   i.MX6ULL rev1.1 at 396 MHz
Model: Atmark Techno Armadillo-600 Series

: (省略)

Starting kernel ...


   OpenRC 0.45.2 is starting up Linux 5.10.185-0-at (armv7l)

 * Mounting /proc ... [ ok ]
 * Mounting /run ... * /run/openrc: creating directory

: (省略)

Welcome to Alpine Linux 3.17
Kernel 5.10.185-0-at on an armv7l (/dev/ttymxc0)
```

```
armadillo login:
```

既に起動している場合は、Enter を 1 回押すことで armadillo login: が表示されます。

3.1.7.4. ログイン

起動が完了するとログインプロンプトが表示されます。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/ttymxc0)

   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に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。

3.1.7.5. Armadillo の終了方法

eMMC や USB メモリ等に書き込みを行っている時に電源を切断すると、データが破損する可能性が有ります。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!
zramswap                | * Deactivating zram swap device ... [ ok ]

:（省略）

The system is going down NOW!
Sent SIGTERM to all processes
Sent SIGKILL to all processes
Requesting system poweroff
[   52.336202] imx2-wdt 20bc000.watchdog: Device shutdown: Expect reboot!
[   52.343210] reboot: Power down
```



poweroff の場合、Armadillo-610 は、ONOFF ピンを GND とショートすることで電源をオフした場合と同じ状態になります。そのため、RTC_BAT ピンからバックアップ電源が供給されている限り、5V 電源を切ったのち 5V 電源を再入力しても Armadillo-610 が起動しません。詳しくは「3.4.6.6. 外部からの電源制御」を参照してください。



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



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。

3.1.8. 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-610 購入製品登録

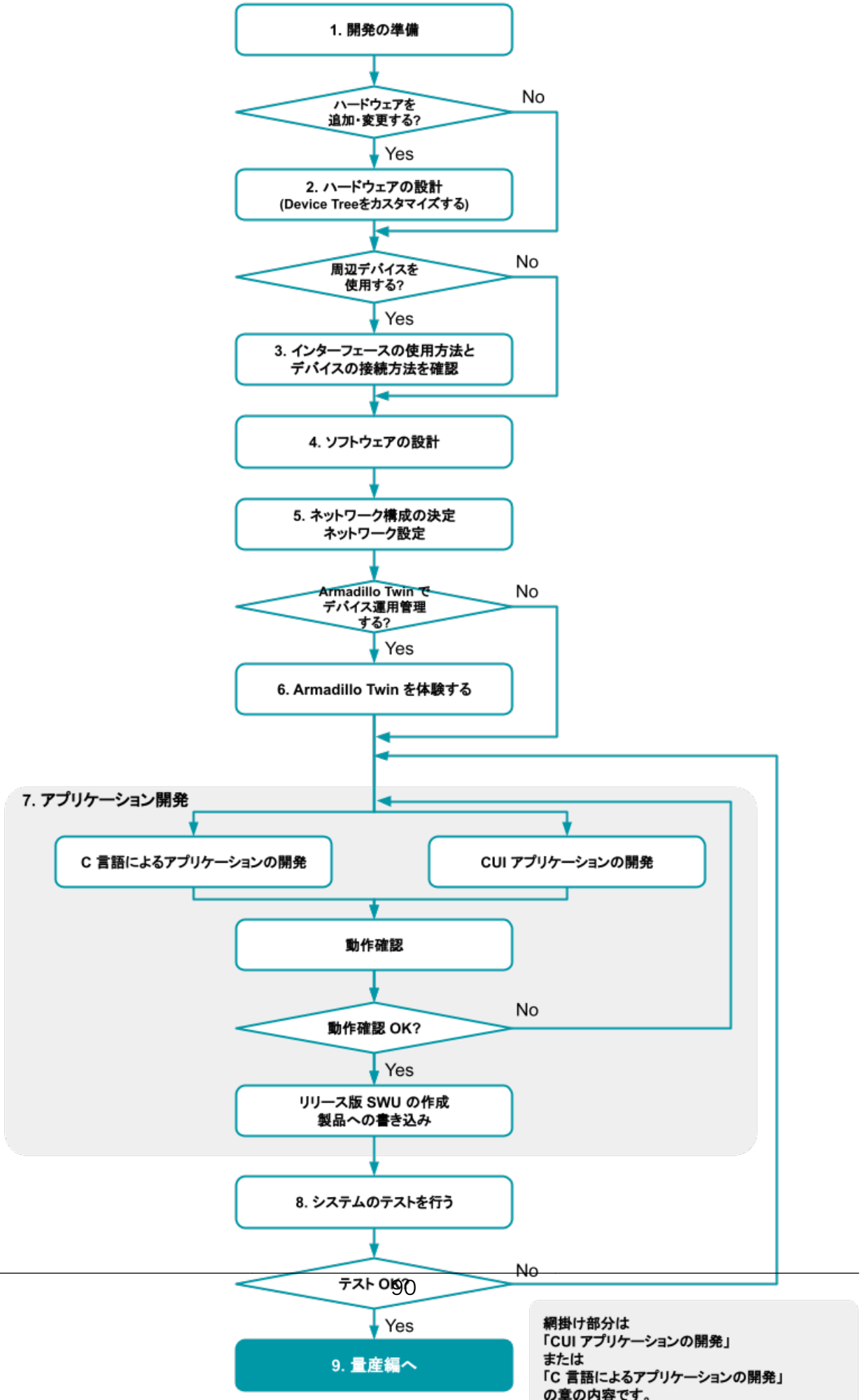
https://armadillo.atmark-techno.com/armadillo-610/register

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-610 でのアプリケーション開発の流れを「図 3.49. アプリケーション開発の流れ」に示します。

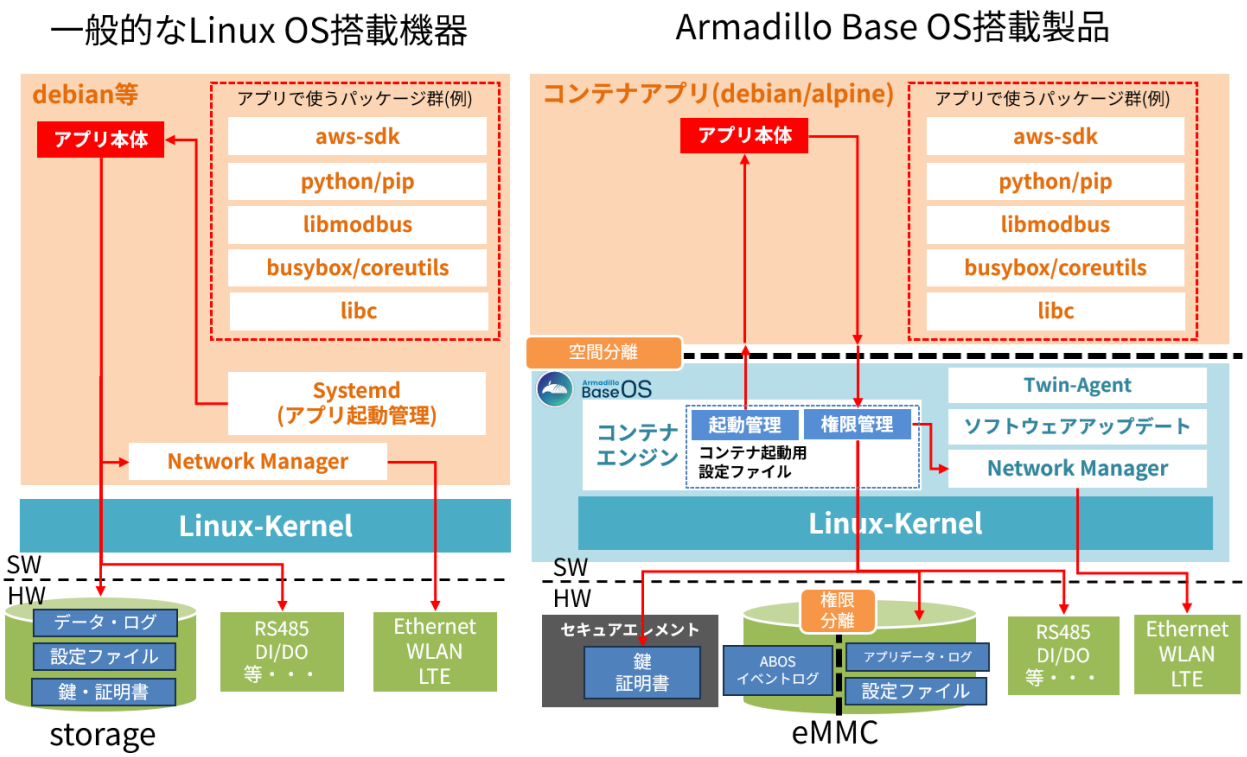
本章では、「図 3.49. アプリケーション開発の流れ」に示す開発時の流れに沿って手順を紹介していきます。


1. 開発の準備
ハードウェアを
追加・変更する?
No
Yes
2. ハードウェアの設計
(Device Treeをカスタマイズする)
周辺デバイスを
使用する?
No
Yes
3. インターフェースの使用方法と
デバイスの接続方法を確認
4. ソフトウェアの設計
5. ネットワーク構成の決定
ネットワーク設定
Armadillo Twin で
デバイス運用管理
する?
No
Yes
6. Armadillo Twin を体験する
7. アプリケーション開発
C 言語によるアプリケーションの開発
CUI アプリケーションの開発
動作確認
動作確認 OK?
No
Yes
リリース版 SWU の作成
製品への書き込み
8. システムのテストを行う
テスト OK?
No
Yes
9. 量産編へ
網掛け部分は
「CUI アプリケーションの開発」
または
「C 言語によるアプリケーションの開発」
の章の内容です。

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

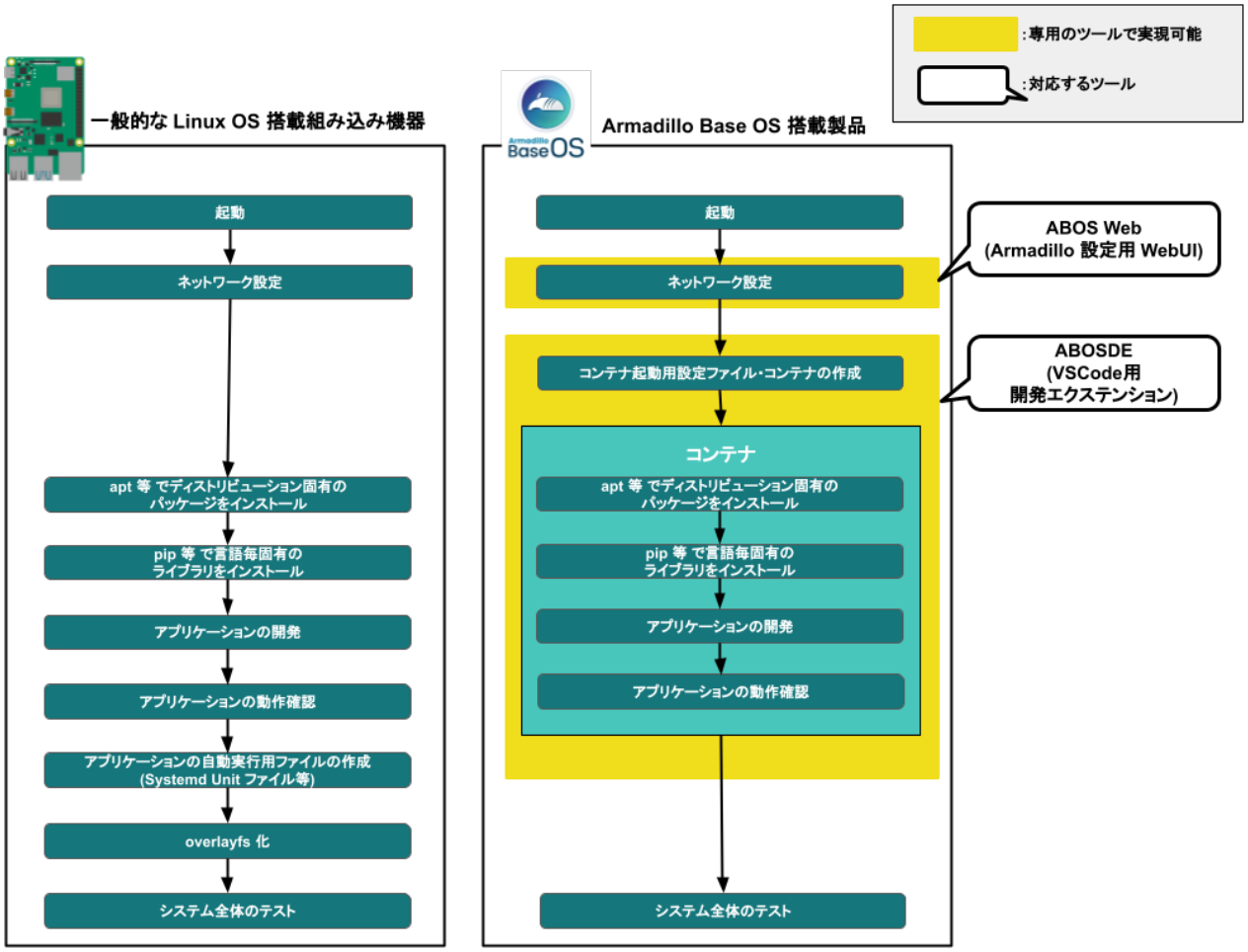


Linux OS 搭載組み込み機器ではアプリケーションの実行環境をユーザーランド上に直接用意し、Systemd などでアプリケーションを自動実行させるのが一般的です。Armadillo Base OS 搭載機器では、アプリケーションの実行環境をコンテナ内に用意して、コンテナ起動用設定ファイルを所定の場所に配置することでコンテナ(=アプリケーション)を自動実行させます。

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 overlayfs 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 overlayfs 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-610 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-610 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-610 開発用ツール [https://armadillo.atmark-techno.com/resources/software/armadillo-610/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の値を計算して、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分かります。

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もありません。

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/uImage, /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こと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
        eend $? "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-610 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-610 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-610 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-610 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。

❸ /media/USBDRIVE をアンマウントします。コマンド終了後に USB メモリを取り外してください。

エラーの場合、/var/log/message に保存されます。例えば、コンソールで証明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-610 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-610/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 3.50 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 3.2. 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ありません。

表 3.2 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 3.51 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-610 インストールディスクイメージ [https://armadillo.atmark-techno.com/resources/software/armadillo-610/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-610 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. インストールディスクの作成

インストールディスクの作成方法は「3.1.4.1. 初期化インストールディスクの作成」を参照してください。

参照先では初期化インストールディスクの場合の手順を示していますが、「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についても同じ手順になります。その際のインストールディスクイメージ（.img）は、以下のコマンドを実行して作成してください。

```
[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 ~/uboot-[VERSION]/u-boot-dtb.imx ¥
        --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。

3.3.5.2. インストールディスクを使用する

インストールディスクを使用する方法については、「3.1.4.3. インストールディスクを使用する」を参照してください。

3.4. ハードウェアの設計

Armadillo-610 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-610 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

フレキシブルフラットケーブル(FFC)を使用して拡張を行った場合、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 FFC で接続する拡張ボードに搭載している場合、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
  - シールドは拡張ボードの GND に接続する
- 固定穴を GND と接続し、固定穴同士を太い導線や金属スペーサー等で接続する
- 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-610 を組み込んだ機器の ESD 耐性を向上させるために、以下の対策が効果的です。

- 金属筐体に組み込み、GND(固定穴等)を金属ねじ等で接続する
- 金属筐体を接地する

また、Armadillo-610 を組み込んだ機器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. 拡張ボードの設計

Armadillo-610 は拡張インターフェース(Armadillo-610: CON2)から拡張します。電源、リセット、複数の機能をもった i.MX6ULL の信号線等、Armadillo-610 を拡張するために必要な信号線はすべて、CON2 の 100 ピンコネクタに接続されています。

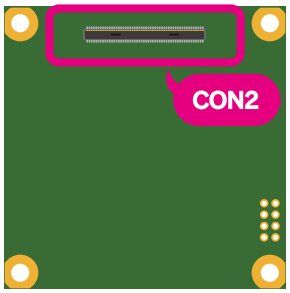


図 3.52 Armadillo-610 の CON2

Armadillo-610 では、「表 2.3. 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-610 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、CON2 の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。

Armadillo-610 マルチプレクス表

CON2 ピン番号	信号名	i.MX6ULL ピン名	i.MX6ULL 電圧グループ	i.MX6ULL リセット時期待の信号状態 機能	In/Out	Value	基板上のPull-Up/Pull-Down とコンデンサ	GPIO	USDHC2	
1	USB_OTG1_DP	USB_OTG1_DP	-	USB_OTG1_DP	-	-				
2	USB_OTG1_DN	USB_OTG1_DN	-	USB_OTG1_DN	-	-				
3	GND									
4	USB_OTG2_DN	USB_OTG2_DN	-	USB_OTG2_DN	-	-				
5	USB_OTG2_DP	USB_OTG2_DP	-	USB_OTG2_DP	-	-				
6	GND									
7	USB_OTG1_VBUS	USB_OTG1_VBUS	-	USB_OTG1_VBUS	-	-	1uF Bypass Capacitor			
8	USB_OTG2_VBUS	USB_OTG2_VBUS	-	USB_OTG2_VBUS	-	-	1uF Bypass Capacitor			
9	SPEEDLED	-	-	-	-	-				
10	LINK_ACTLED	-	-	-	-	-				
11	GPIO1_IO19	UART1_RTS_B	+3.3V_IO	GPIO1_IO19	Input	Keeper	10kΩ Pull-Down	GPIO1_IO19	USDHC2_CD_B	
12	GPIO4_IO17	CSI_MCLK	+3.3V_IO	GPIO4_IO17	Input	Keeper		GPIO4_IO17	USDHC2_CD_B	
13	GPIO5_IO00	SNVS_TAMPER0	VDD_SNVS_IN	GPIO5_IO00	Input	Keeper		GPIO5_IO00		
14	GPIO1_IO04	GPIO1_IO04	+3.3V_IO	GPIO1_IO04	Input	Keeper		GPIO1_IO04		
15	GPIO1_IO03	GPIO1_IO03	+3.3V_IO	GPIO1_IO03	Input	Keeper		GPIO1_IO03		
16	GPIO1_IO02	GPIO1_IO02	+3.3V_IO	GPIO1_IO02	Input	Keeper		GPIO1_IO02		
17	GPIO1_IO01	GPIO1_IO01	+3.3V_IO	GPIO1_IO01	Input	Keeper		GPIO1_IO01		
18	LCD_DATA00	LCD_DATA00	+3.3V_IO	GPIO3_IO05	Input	Keeper	10kΩ Pull-Down	GPIO3_IO05		LCD
19	LCD_DATA01	LCD_DATA01	+3.3V_IO	GPIO3_IO06	Input	Keeper	10kΩ Pull-Up	GPIO3_IO06		LCD
20	LCD_DATA02	LCD_DATA02	+3.3V_IO	GPIO3_IO07	Input	Keeper	10kΩ Pull-Down	GPIO3_IO07		LCD
21	LCD_DATA03	LCD_DATA03	+3.3V_IO	GPIO3_IO08	Input	Keeper	10kΩ Pull-Down	GPIO3_IO08		LCD
22	LCD_DATA04	LCD_DATA04	+3.3V_IO	GPIO3_IO09	Input	Keeper	10kΩ Pull-Down	GPIO3_IO09		LCD
23	LCD_DATA05	LCD_DATA05	+3.3V_IO	GPIO3_IO10	Input	Keeper	10kΩ Pull-Up or 10kΩ Pull-Down	GPIO3_IO10		LCD
24	LCD_DATA06	LCD_DATA06	+3.3V_IO	GPIO3_IO11	Input	Keeper	10kΩ Pull-Up	GPIO3_IO11		LCD
25	LCD_DATA07	LCD_DATA07	+3.3V_IO	GPIO3_IO12	Input	Keeper	10kΩ Pull-Down	GPIO3_IO12		LCD
26	LCD_DATA08	LCD_DATA08	+3.3V_IO	GPIO3_IO13	Input	Keeper	10kΩ Pull-Down	GPIO3_IO13		LCD
27	LCD_DATA09	LCD_DATA09	+3.3V_IO	GPIO3_IO14	Input	Keeper	10kΩ Pull-Down	GPIO3_IO14		LCD
28	LCD_DATA10	LCD_DATA10	+3.3V_IO	GPIO3_IO15	Input	Keeper	10kΩ Pull-Down	GPIO3_IO15		LCD
29	LCD_DATA11	LCD_DATA11	+3.3V_IO	GPIO3_IO16	Input	Keeper	10kΩ Pull-Up or 10kΩ Pull-Down	GPIO3_IO16		LCD
30	LCD_DATA12	LCD_DATA12	+3.3V_IO	GPIO3_IO17	Input	Keeper	10kΩ Pull-Down	GPIO3_IO17		LCD
31	LCD_DATA13	LCD_DATA13	+3.3V_IO	GPIO3_IO18	Input	Keeper	10kΩ Pull-Down	GPIO3_IO18	USDHC2_RESET_B	LCD
32	LCD_DATA14	LCD_DATA14	+3.3V_IO	GPIO3_IO19	Input	Keeper	10kΩ Pull-Down	GPIO3_IO19	USDHC2_DATA4	LCD

割り当て可能な機能

図 3.53 Armadillo-610 のマルチプレクス表

本書には各機能の概要しか記載していませんので、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』、

『i.MX 6ULL Applications Processors for Industrial Products』をご確認ください。Armadillo-610 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「3.4.5.15. CON2(拡張インターフェース)の概要」に i.MX6ULL 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-610 の拡張ボードを設計開発するためのリファレンス回路を公開しています。Armadillo-610 リファレンス回路では、以下の機能の拡張方法を確認することができます。また、一部の回路の動作は Armadillo-610 拡張ボードで確認することができます。

- 電源
- 起動デバイス設定
- LAN(Ethernet)、無線 LAN
- USB HUB、USB Host、USB OTG
- SD
- LCD [2]
- RS-485、UART(デバッグ用)
- MQS、SAI [2]
- 絶縁デジタル入出力
- リアルタイムクロック
- スイッチ、LED
- A/D、I2C



Armadillo-610 リファレンス回路図、部品表[3]は「アットマークテクノ Armadillo サイト」 [https://armadillo.atmark-techno.com/]からダウンロードしてください。



Armadillo-610 リファレンス回路図、部品表に記載している部品につきまして、既に販売終了していたり、終息部品になっている場合があります。部品採用を決定する前に、部品の現在のステートについて、十分にご確認ください。

3.4.5.1. 電源

Armadillo-610 の電源電圧は 3.6～4.5V です。拡張ボード側に USB Host を搭載するのであれば、USB デバイスに供給するための 5V が必要となりますので、拡張ボード側の主電源を 5V にして USB への供給電源とし、5V を 3.6～4.5V の電圧に降圧して、Armadillo-610 に供給するのがおすすめです。

[2]Armadillo-400 シリーズ LCD オプションセットの LCD 拡張ボードの回路図も合わせてご確認ください。
[3]Armadillo-610 リファレンス回路図、部品表は購入者向けの限定データです。

Armadillo-610 リファレンス回路で、5V を 3.7V に降圧して Armadillo-610 に供給する回路を確認することができます。抵抗値で出力電圧を変更できるタイプの DC/DC コンバータを採用しておりますので、別の電圧値が必要な場合は、抵抗値を変更することで対応可能です。

電源は拡張インターフェース(Armadillo-610: CON2)の VIN ピンから供給します。CON2 に搭載している 100 ピンのコネクタは、1 ピンあたり流せる電流値が最大 0.3A です。VIN ピンは全部で 4 本ありますので、CON2 から供給できる電流値は最大 1.2A となります。

CON2 から、5V 電源(+5V_IO)、3.3V 電源(+3.3V_IO)が拡張ボード用に出力されています。+5V_IO は最大 600mA、+3.3V_IO は最大 500mA まで供給可能ですが、Armadillo-610 への最大供給電流が 1.2A であるため、必ずしも最大まで出力することはできません。各最大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ってください。

全体の消費電力が少ないのであれば、Armadillo-610 へは 3.6～4.5V 電源を直接供給し、5V 電源、3.3V 電源を Armadillo-610 からの出力で賄うという構成も可能です。

Armadillo-610 の電源投入時、+5V_IO は Armadillo-610 の電源(VIN)とほぼ同時に立ち上がり、一定時間 VIN と同電位を維持した後、ソフトウェアから有効にされたタイミングで 5V まで立ち上がります。+5V_IO は、VIN を昇圧して 5V 電源を生成しているため、+5V_IO からの出力電圧を VIN 以下にすることはできません。+5V_IO の出力無効時は VIN と同電位の電圧が出力され、+5V_IO の出力有効時は +5V が出力されます。+5V_IO の出力を 0V にしたい場合は、電源制御のためのパワースイッチの搭載をおすすめします。

「3.4.6. 電気的仕様」で Armadillo-610 の電気的仕様について説明しておりますので、詳細についてはこちらでご確認ください。

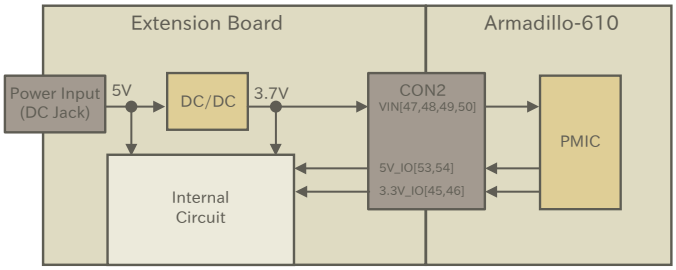


図 3.54 電源回路例

3.4.5.2. 起動デバイスの設定

Armadillo-610 は下記 2 つのデバイスから起動が可能です。

- オンボード eMMC
- microSD カード(Armadillo-610: CON1)

どちらのデバイスから起動するかは、eFUSE もしくは拡張インターフェース(Armadillo-610: CON2)の BJP1 ピンで設定します。eFUSE で設定する方法については、「6.28. eFuse を変更する」をご確認ください。BJP1 ピンの状態は、Armadillo-610 の電源(VIN)投入時に読み出され、起動デバイスが選択されます。

表 3.3 BJP1 の状態と起動デバイス

BJP1	起動デバイス
Low	eMMC

BJP1	起動デバイス
High	microSD カード

BJP1 ピンは Armadillo-610 上で 47kΩ でプルダウンされているため、eMMC から起動したい場合は BJP1 ピンをオープン、microSD カードから起動したい場合は、BJP1 ピンを+3.3V_IO にプルアップ(抵抗値は 1kΩ 程度)してください。

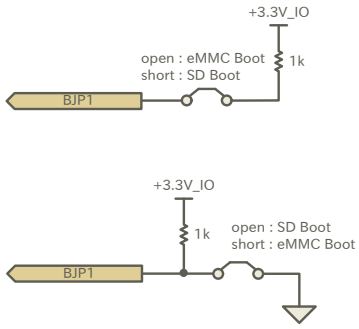


図 3.55 起動設定ジャンパー例

起動デバイスを microSD カードに設定した場合でも、microSD スロットに microSD カードが挿さっていなかった場合は、eMMC から起動します。 [4]

出荷時、i.MX6ULL の起動デバイスに関する eFUSE は未設定です。

eFUSE を設定した場合、BJP1 の設定は無視されます。

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-610 が正常に動作しなくなる可能性がありますので、書き込みを行う際には細心の注意を払うようお願いいたします。eFUSE の設定によって異常が起こった場合は、保証対象外となります。

3.4.5.3. LAN(Ethernet)

LAN を拡張する場合は、拡張ボード側にトランスと LAN コネクタ、LED を搭載してください。拡張インターフェース(Armadillo-610: CON2)には Ethernet PHY の送受信の信号線および LAN のスピード LED、アクティビティ LED 用の信号線が接続されています。

[4]eFUSE で eMMC からの起動を禁止した場合を除きます

回路の詳細は Armadillo-610 リファレンス回路で確認することが可能です。また、Armadillo-610 拡張ボードで動作確認することも可能です。

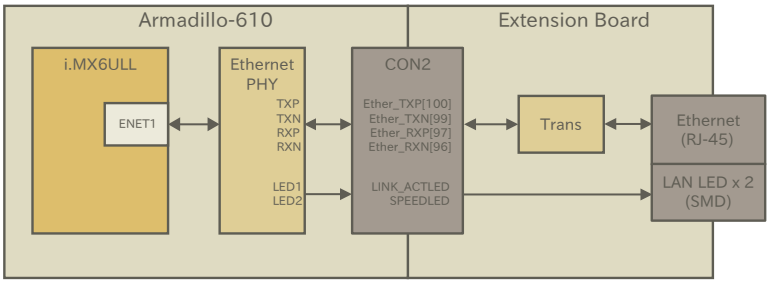


図 3.56 LAN(Ethernet)接続例

3.4.5.4. 無線 LAN/BT

無線 LAN/BT を拡張する場合、動作確認済みデバイスである、Ezurio 製の Sterling LWB5+がおすすめです。Sterling LWB5+の無線 LAN 回路は USB もしくは SDIO、BT 回路は USB もしくは UART で接続することができます。

無線 LAN、BT 共に USB 接続した場合の回路の詳細をリファレンス回路で確認することが可能です。また、姉妹製品である Armadillo-640 と Armadillo-600 シリーズ WLAN オプションモジュールで無線 LAN/BT の動作を確認することも可能です。

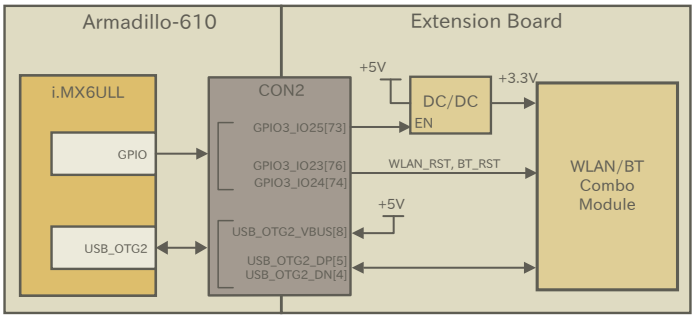


図 3.57 WLAN/BT 接続例

3.4.5.5. USB

USB は 2 ポート拡張可能で、Host は最大 2 ポート、OTG は最大 1 ポート拡張することが可能です。

USB Host を拡張する場合は、USB デバイスへ供給する電源制御のためのパワースイッチを搭載してください。パワースイッチのイネーブルピン制御のための GPIO は電源投入時、プルアップされていないものを選定してください。

USB_OTGx_VBUS ピンを使用しないデバイスを搭載する場合でも、デバイス検出のために USB_OTGx_VBUS ピンへ電圧を印加する必要があります。

USB_OTGx_VBUS ピンへの電源が ON/OFF 制御される場合、USB_OTGx_VBUS ピンへの電圧印加はダイオード経由にする必要があります。



USB_OTGx_VBUS ピンへは最低 4.4V の電圧を印加する必要がありますので、ダイオードの Vf 電圧にご注意ください。電圧が低すぎる場合、USB メモリの挿抜を検出できない等の不具合が発生します。

USB ポートが 3 ポート以上必要な場合は、USB ハブを接続してください。

Armadillo-610 リファレンス回路では、USB ハブ、Host、OTG の回路を確認することが可能です。リファレンス回路で採用している USB ハブは 3 ポート品ですが、ピンコンパチで 4 ポート品もラインアップされています。

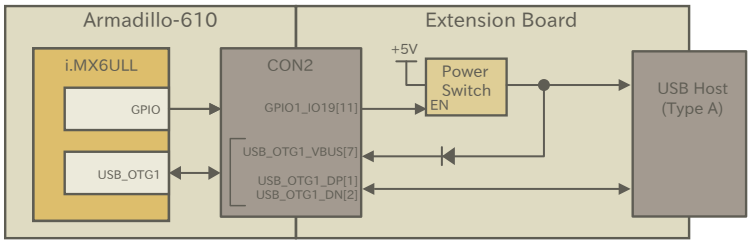


図 3.58 USB Host 接続例

3.4.5.6. シリアル(UART)

Armadillo-610 のシリアル(UART)の信号レベルは+3.3V_IO ですので、必要なインターフェースの規格に合わせて、レベル変換 IC 等を拡張ボード側に搭載してください。

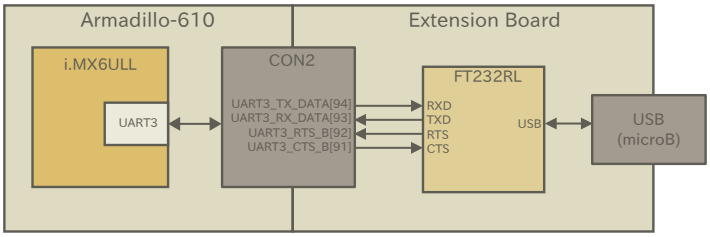


図 3.59 シリアル(UART)接続例



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。誤接続にご注意ください。



デバッグやメンテナンス用途であれば、拡張ボード上にレベル変換 IC を搭載せずに、外付けのレベル変換アダプタを使用するのもおすすめです。レベル変換アダプタは、弊社からもオプション品として購入することが可能です。

3.4.5.7. SD

SD ホストコントローラ(uSDHC2)を使用できる信号線が SD インターフェース(Armadillo-610: CON1)と拡張インターフェース(Armadillo-610: CON2)に接続されており、SD ホストコントローラはどちらか一方でしか使用することができません。

SD スロットを基板端に配置したい場合や SDIO 接続のデバイスを拡張したい場合などにご使用ください。

SD スロットを拡張する場合、SD カード検出、ライトプロテクト検出は GPIO で行うことが可能ですので、専用ピンを割り当てる必要はありません。

BJP1 ピンを High レベルにして Armadillo-610 の電源(VIN)を投入した場合、SD インターフェース(Armadillo-610: CON1)に接続された microSD カードがブートデバイスに設定されます。電源投入時、SD ホストコントローラ(uSDHC2)は SD インターフェース(Armadillo-610: CON1)に接続されており、microSD カードに書き込まれたイメージファイルから起動します。起動後、マルチプレクスの設定により、SD ホストコントローラ(uSDHC2)の接続先が変更になるため、拡張インターフェース(Armadillo-610: CON2)側のデバイスがブートデバイスになることはできません。

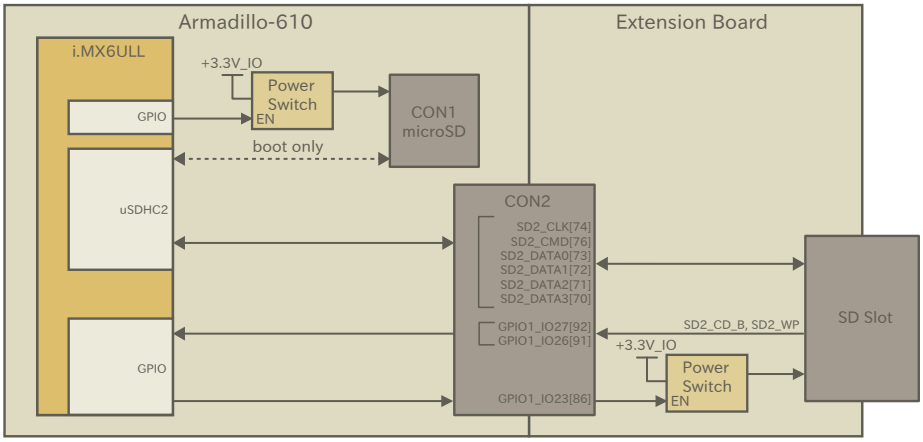


図 3.60 SD 接続例

SD2_DATA3 として使用可能な GPIO3_IO28 ピンはブートモード設定ピンを兼用しています。Armadillo-610 の電源(VIN)投入時から U-Boot が動作するまでは、Low レベルを保持する必要があり、47kΩ のプルダウン抵抗が接続されています。ただし、SD カード等が正常動作するには、SD2_DATA3 にプルアップ抵抗が必要となるため、Armadillo-610 の電源(VIN)投入時は立ち上がらない電源でプルアップ(抵抗値は 15kΩ 程度)し、起動後にプルアップ抵抗の接続された電源を立ち上げる等の対処が必要となります。

SD スロットを拡張する回路の詳細は、Armadillo-610 リファレンス回路で確認することが可能です。

3.4.5.8. スイッチ、LED、リレー

スイッチや LED、リレーを拡張する場合は、GPIO を割り当てます。GPIO に割り当て可能なピンは多数ありますので、プルアップ/プルダウン抵抗の有無と電圧レベルを確認して、使用するピンを決定してください。

拡張インターフェース(Armadillo-610: CON2)には、i.MX6ULL の信号線が直接接続されています。スイッチは人の手で操作するインターフェースですので、静電気等による内部回路の故障を防ぐため、電流制限抵抗を接続することをおすすめします。

LED、リレーは GPIO ピンで直接駆動せずにトランジスタ等を経由して駆動してください。

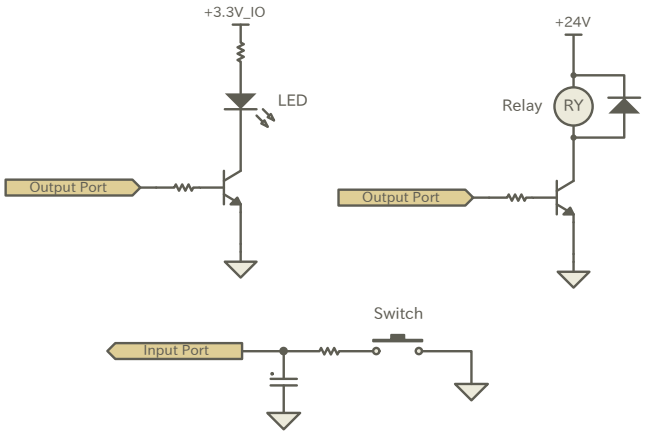


図 3.61 スイッチ、LED、リレー接続例

3.4.5.9. リアルタイムクロック

i.MX6ULL 内蔵のリアルタイムクロックのバックアップ用のピン(RTC_BAT)が拡張インターフェース(Armadillo-610: CON2)に接続されていますので、Armadillo-610 の電源(VIN)を切断しても時刻データを保持したい場合にバッテリー等を接続してください。

i.MX6ULL 内蔵の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速くなります。

バッテリーの消耗が製品の運用に支障をきたす場合は、消費電力が少ないリアルタイムクロックを拡張ボード側に搭載してください。

3.4.5.10. LCD

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ます。Armadillo-610 リファレンス回路では、以下のタッチパネル LCD を接続するための回路を確認することが可能です。

- 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-00)
- Armadillo-400 シリーズ LCD オプションセット(4.3 インチタッチパネル WQVGA 液晶)(型番: OP-A400-LCD43EXT-L01)

オプションセットの詳細につきましては「6.29.3. LCD オプションセット(7 インチタッチパネル WVGA 液晶)」、「6.29.4. Armadillo-400 シリーズ LCD オプションセット」をご確認ください。

3.4.5.11. オーディオ

I2S で最大 2 ポート、MQS で最大 1 ポート、S/PDIF で最大 1 ポートオーディオを拡張することが可能です。MQS は拡張ボード側にオーディオアンプを搭載するだけで良いので、高品質な音が必要でない場合にはおすすめです。SAI、S/PDIF を使用する場合は、オーディオコーデックとオーディオアンプ等を拡張ボード側に搭載してください。

MQS の回路は Armadillo-610 リファレンス回路図、I2S の回路は Armadillo-400 シリーズ LCD オプションセットの LCD 拡張ボード回路図で確認することが可能です。

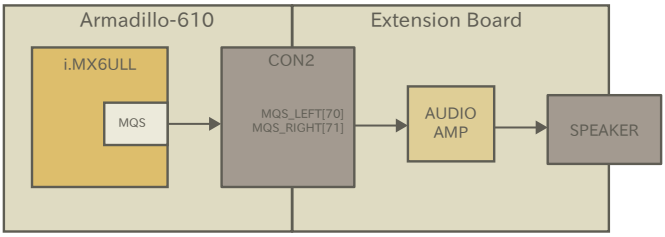


図 3.62 MQS 接続例

3.4.5.12. リセットスイッチ

拡張インターフェース(Armadillo-610: CON2)の EXT_RESET_B ピン、PWRON ピンによりリセットスイッチを拡張することが可能です。押していない時はオープン、押した時は GND とショートするスイッチを接続してください。

スイッチは人の手で操作するインターフェースですので、静電気等による内部回路の故障を防ぐため、電流制限抵抗を接続することをおすすめします。

EXT_RESET_B ピンではシステムリセット、PWRON ピンではパワーマネジメント IC からの電源の供給/切断の制御が可能です。それぞれの機能の詳細については、「3.4.6.5. リセット回路の構成」、「3.4.6.6. 外部からの電源制御」をご確認ください。

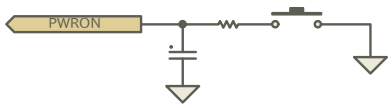


図 3.63 PWRON 回路例

3.4.5.13. ONOFF スイッチ

拡張インターフェース(Armadillo-610: CON2)の ONOFF ピンにより、長押しで電源の制御を行うスイッチを拡張することが可能です。押していない時はオープン、押した時は GND とショートするスイッチを接続してください。

スイッチは人の手で操作するインターフェースですので、静電気等による内部回路の故障を防ぐため、電流制限抵抗を接続することをおすすめします。

ONOFF ピンの機能の詳細については「3.4.6.6. 外部からの電源制御」をご確認ください。

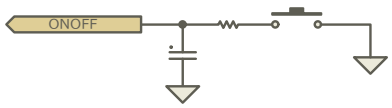


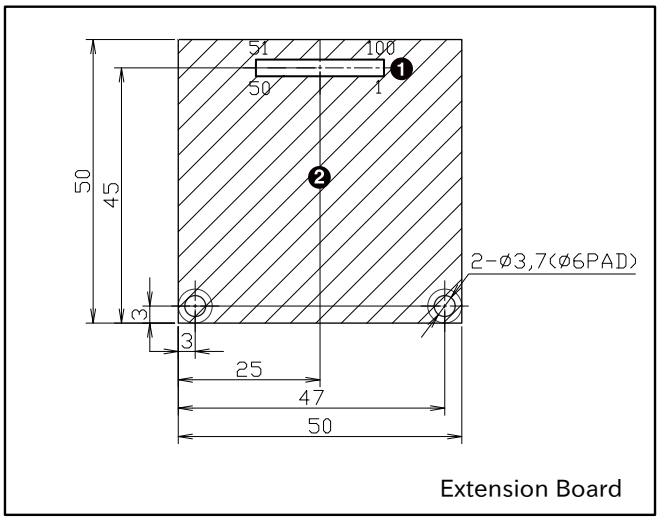
図 3.64 ONOFF 回路例

3.4.5.14. 基板形状

Armadillo-610 の拡張ボードを設計する際の、推奨レイアウトは「図 3.65. 拡張ボード推奨レイアウト」のとおりです。

Armadillo-610 との接続コネクタは、HIROSE ELECTRIC 製の DF40HC(3.0)-100DS-0.4V(51)を搭載してください。嵌合高さは 3mm となりますので、Armadillo-610 の下に部品を配置する場合、部品高さにご注意ください。

固定穴径とパッド寸法はマックエイト製のスルーホールタップ(TH-1.6-3.0-M3)を実装する場合の推奨となります。別の方法で固定する場合は適宜寸法を変更してください。



[Unit:mm]

図 3.65 拡張ボード推奨レイアウト

❶ DF40HC(3.0)-100DS-0.4V(51)/HIROSE ELECTRIC

❷ Armadillo-610 側の最大部品高さ: 2mm

Armadillo-610 の固定穴は GND に接続されています。GND 強化のため、拡張ボード側の固定穴も GND に接続し、金属製のスペーサーとねじで固定してください。スペーサーの長さはコネクタの嵌合高さと同じ 3mm としてください。

Armadillo-610 の固定例は「図 3.66. Armadillo-610 の固定例」のとおりです。

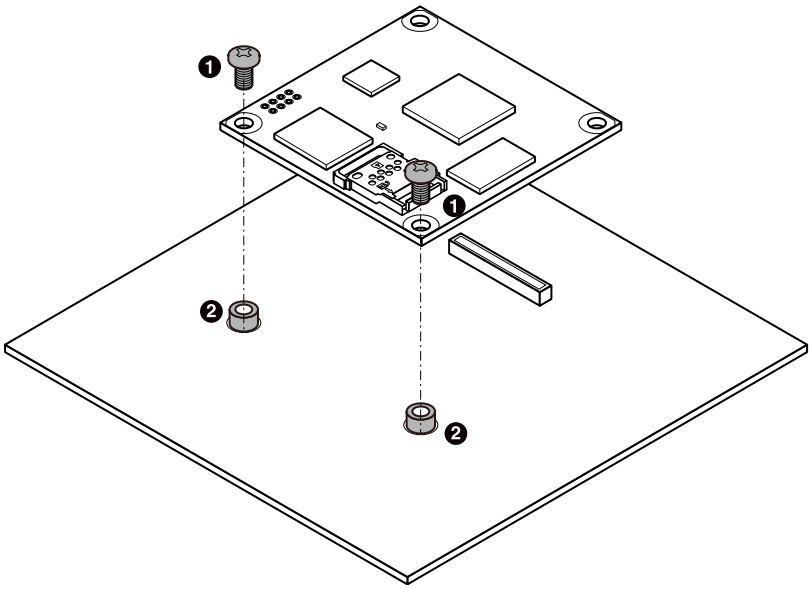


図 3.66 Armadillo-610 の固定例

❶ なべ小ねじ(M3、L=5mm)

❷ スルーホールタップ(TH-1.6-3.0-M3/Mac-Eight)



Armadillo-610 の固定穴は 4 箇所ありますが、コネクタから離れた位置の 2 箇所のみ、スペーサーとねじで固定して各種試験を行い、動作に異常がないことを確認しております。試験の詳細につきましては、「アットマークテクノ Armadillo サイト」 [https://armadillo.atmark-techno.com/] の「Armadillo-610 信頼性試験報告書」にてご確認ください。

3.4.5.15. CON2(拡張インターフェース)の概要

CON2 は Armadillo-610 拡張用のインターフェースです。電源、リセット、複数の機能(マルチプレクス)をもった i.MX6ULL の信号線、USB、Ethernet PHY の信号線等、Armadillo-610 を拡張するために必要な信号線がすべて接続されています。

Armadillo-610 の電源は VIN ピンから供給します。

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、長時間電源が切断されても i.MX6ULL の一部データ(時刻データ等)を保持させたい場合にご使用ください。



プルアップ/ダウン抵抗が接続されている拡張入出力ピンは、i.MX6ULL の内蔵 ROM によるブートモード設定ピンを兼用しており、ブートモード設定のため、プルアップ/ダウン抵抗で電源投入時に 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。

表 3.4 CON2 信号配列

ピン番号	ピン名	I/O	説明
1	USB_OTG1_DP	In/Out	USB_OTG1 のプラス側信号、i.MX6ULL の USB_OTG1_DP ピンに接続
2	USB_OTG1_DN	In/Out	USB_OTG1 のマイナス側信号、i.MX6ULL の USB_OTG1_DN ピンに接続
3	GND	Power	電源(GND)
4	USB_OTG2_DN	In/Out	USB_OTG2 のマイナス側信号、i.MX6ULL の USB_OTG2_DN ピンに接続
5	USB_OTG2_DP	In/Out	USB_OTG2 のプラス側信号、i.MX6ULL の USB_OTG2_DP ピンに接続
6	GND	Power	電源(GND)
7	USB_OTG1_VBUS	Power	電源(USB_OTG1_VBUS)、i.MX6ULL の USB_OTG1_VBUS ピンに接続、1uF のバイパスコンデンサが接続されています。
8	USB_OTG2_VBUS	Power	電源(USB_OTG2_VBUS)、i.MX6ULL の USB_OTG2_VBUS ピンに接続、1uF のバイパスコンデンサが接続されています。
9	SPEEDLED	Out	LAN スピード LED 用信号、Ethernet PHY の LED2 ピンに接続
10	LINK_ACTLED	Out	LAN リンクアクティビティ LED 用信号、Ethernet PHY の LED1 ピンに接続
11	GPIO1_IO19	In/Out	拡張入出力、i.MX6ULL の UART1_RTS_B ピンに接続、基板上で 10kΩ プルダウンされています。
12	GPIO4_IO17	In/Out	拡張入出力、i.MX6ULL の CSI_MCLK ピンに接続
13	GPIO5_IO00	In/Out	拡張入出力、i.MX6ULL の SNVS_TAMPER0 ピンに接続、オープンドレインでの使用推奨 [a]
14	GPIO1_IO04	In/Out	拡張入出力、i.MX6ULL の GPIO1_IO04 ピンに接続
15	GPIO1_IO03	In/Out	拡張入出力、i.MX6ULL の GPIO1_IO03 ピンに接続
16	GPIO1_IO02	In/Out	拡張入出力、i.MX6ULL の GPIO1_IO02 ピンに接続
17	GPIO1_IO01	In/Out	拡張入出力、i.MX6ULL の GPIO1_IO01 ピンに接続
18	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。
19	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(+3.3V_IO)されています。
20	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。
21	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。
22	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。
23	LCD_DATA05	In/Out	拡張入出力、i.MX6ULL の LCD_DATA05 ピンに接続、BJP1 が Low レベル時 10kΩ プルアップ(+3.3V_IO)、High レベル時 10kΩ プルダウンされます。
24	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(+3.3V_IO)されています。
25	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。
26	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。
27	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。
28	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。
29	LCD_DATA11	In/Out	拡張入出力、i.MX6ULL の LCD_DATA11 ピンに接続、BJP1 が Low レベル時 10kΩ プルダウン、High レベル時 10kΩ プルアップ(+3.3V_IO)されます。
30	LCD_DATA12	In/Out	拡張入出力、i.MX6ULL の LCD_DATA12 ピンに接続、基板上で 10kΩ プルダウンされています。
31	LCD_DATA13	In/Out	拡張入出力、i.MX6ULL の LCD_DATA13 ピンに接続、基板上で 10kΩ プルダウンされています。
32	LCD_DATA14	In/Out	拡張入出力、i.MX6ULL の LCD_DATA14 ピンに接続、基板上で 10kΩ プルダウンされています。
33	LCD_DATA15	In/Out	拡張入出力、i.MX6ULL の LCD_DATA15 ピンに接続、基板上で 10kΩ プルダウンされています。

ピン番号	ピン名	I/O	説明
34	LCD_DATA16	In/Out	拡張入出力、i.MX6ULL の LCD_DATA16 ピンに接続、基板上で 10kΩ プルダウンされています。
35	LCD_DATA17	In/Out	拡張入出力、i.MX6ULL の LCD_DATA17 ピンに接続、基板上で 10kΩ プルダウンされています。
36	GND	Power	電源(GND)
37	LCD_CLK	In/Out	拡張入出力、i.MX6ULL の LCD_CLK ピンに接続
38	LCD_HSYNC	In/Out	拡張入出力、i.MX6ULL の LCD_HSYNC ピンに接続
39	LCD_VSYNC	In/Out	拡張入出力、i.MX6ULL の LCD_VSYNC ピンに接続
40	LCD_ENABLE	In/Out	拡張入出力、i.MX6ULL の LCD_ENABLE ピンに接続
41	PWM5_OUT	In/Out	拡張入出力、i.MX6ULL の NAND_DQS ピンに接続
42	BJP1	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、基板上で 47kΩ プルダウンされています。 (Low: LCD_DATA05 ピンは 10kΩ プルアップ(+3.3V_IO)、LCD_DATA11 ピンは 10kΩ プルダウンされます。High: LCD_DATA05 ピンは 10kΩ プルダウン、LCD_DATA11 ピンは 10kΩ プルアップ(+3.3V_IO)されます。)
43	JTAG_MOD	In	SJC モード設定ピン、i.MX6ULL の JTAG_MOD ピンに接続、基板上で 11kΩ プルダウンされています。 [b]
44	EXT_RESET_B	In	システムリセット、i.MX6ULL の POR_B ピンに接続、オープンドレイン入力
45	+3.3V_IO	Power	電源(+3.3V_IO)
46	+3.3V_IO	Power	電源(+3.3V_IO)
47	VIN	Power	電源(VIN)
48	VIN	Power	電源(VIN)
49	VIN	Power	電源(VIN)
50	VIN	Power	電源(VIN)
51	GND	Power	電源(GND)
52	GND	Power	電源(GND)
53	+5V_IO	Power	電源(+5V_IO)
54	+5V_IO	Power	電源(+5V_IO)
55	GPIO4_IO19	In/Out	拡張入出力、i.MX6ULL の CSI_VSYNC ピンに接続
56	GPIO4_IO20	In/Out	拡張入出力、i.MX6ULL の CSI_HSYNC ピンに接続
57	GPIO4_IO25	In/Out	拡張入出力、i.MX6ULL の CSI_DATA04 ピンに接続
58	GPIO4_IO26	In/Out	拡張入出力、i.MX6ULL の CSI_DATA05 ピンに接続
59	GPIO4_IO27	In/Out	拡張入出力、i.MX6ULL の CSI_DATA06 ピンに接続
60	GPIO4_IO28	In/Out	拡張入出力、i.MX6ULL の CSI_DATA07 ピンに接続
61	GPIO4_IO23	In/Out	拡張入出力、i.MX6ULL の CSI_DATA02 ピンに接続
62	GPIO4_IO22	In/Out	拡張入出力、i.MX6ULL の CSI_DATA01 ピンに接続
63	GPIO4_IO24	In/Out	拡張入出力、i.MX6ULL の CSI_DATA03 ピンに接続
64	GPIO4_IO21	In/Out	拡張入出力、i.MX6ULL の CSI_DATA00 ピンに接続
65	GPIO4_IO18	In/Out	拡張入出力、i.MX6ULL の CSI_PIXCLK ピンに接続
66	GPIO4_IO09	In/Out	拡張入出力、i.MX6ULL の NAND_DATA07 ピンに接続
67	GPIO4_IO08	In/Out	拡張入出力、i.MX6ULL の NAND_DATA06 ピンに接続
68	GPIO4_IO07	In/Out	拡張入出力、i.MX6ULL の NAND_DATA05 ピンに接続
69	GPIO4_IO06	In/Out	拡張入出力、i.MX6ULL の NAND_DATA04 ピンに接続
70	GPIO3_IO28	In/Out	拡張入出力、i.MX6ULL の LCD_DATA23 ピンに接続、基板上で 47kΩ プルダウンされています。
71	GPIO3_IO27	In/Out	拡張入出力、i.MX6ULL の LCD_DATA22 ピンに接続
72	GPIO3_IO26	In/Out	拡張入出力、i.MX6ULL の LCD_DATA21 ピンに接続
73	GPIO3_IO25	In/Out	拡張入出力、i.MX6ULL の LCD_DATA20 ピンに接続
74	GPIO3_IO24	In/Out	拡張入出力、i.MX6ULL の LCD_DATA19 ピンに接続
75	GND	Power	電源(GND)
76	GPIO3_IO23	In/Out	拡張入出力、i.MX6ULL の LCD_DATA18 ピンに接続

ピン番号	ピン名	I/O	説明
77	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_IN)されています。
78	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_IN)されています。
79	RTC_BAT	Power	電源(RTC_BAT)、パワーマネジメント IC の LICELL ピンに接続
80	GPIO1_IO08	In/Out	拡張入出力、i.MX6ULL の GPIO1_IO08 ピンに接続
81	GPIO1_IO05	In/Out	拡張入出力、i.MX6ULL の GPIO1_IO05 ピンに接続
82	GPIO1_IO30	In/Out	拡張入出力、i.MX6ULL の UART5_TX_DATA ピンに接続
83	GPIO1_IO16	In/Out	拡張入出力、i.MX6ULL の UART1_TX_DATA ピンに接続
84	GPIO1_IO31	In/Out	拡張入出力、i.MX6ULL の UART5_RX_DATA ピンに接続
85	GPIO1_IO17	In/Out	拡張入出力、i.MX6ULL の UART1_RX_DATA ピンに接続
86	GPIO1_IO23	In/Out	拡張入出力、i.MX6ULL の UART2_RTS_B ピンに接続
87	GPIO1_IO22	In/Out	拡張入出力、i.MX6ULL の UART2_CTS_B ピンに接続
88	GPIO1_IO21	In/Out	拡張入出力、i.MX6ULL の UART2_RX_DATA ピンに接続
89	GPIO1_IO20	In/Out	拡張入出力、i.MX6ULL の UART2_TX_DATA ピンに接続
90	GPIO1_IO00	In/Out	拡張入出力、i.MX6ULL の GPIO1_IO00 ピンに接続
91	GPIO1_IO26	In/Out	拡張入出力、i.MX6ULL の UART3_CTS_B ピンに接続
92	GPIO1_IO27	In/Out	拡張入出力、i.MX6ULL の UART3_RTS_B ピンに接続
93	GPIO1_IO25	In/Out	拡張入出力、i.MX6ULL の UART3_RX_DATA ピンに接続
94	GPIO1_IO24	In/Out	拡張入出力、i.MX6ULL の UART3_TX_DATA ピンに接続
95	GND	Power	電源(GND)
96	Ether_RXN	In/Out	Ethernet 送信/受信データ(+) CH1、Ethernet PHY(LAN8720AI)の RXN ピンに接続
97	Ether_RXP	In/Out	Ethernet 送信/受信データ(-) CH1、Ethernet PHY(LAN8720AI)の RXP ピンに接続
98	GND	Power	電源(GND)
99	Ether_TXN	In/Out	Ethernet 送信/受信データ(-) CH2、Ethernet PHY(LAN8720AI)の TXN ピンに接続
100	Ether_TXP	In/Out	Ethernet 送信/受信データ(+) CH2、Ethernet PHY(LAN8720AI)の TXP ピンに接続

[a]GPIO5_IO00 のみ VDD_SNVS_IN 系の拡張入出力ピンとなります。電圧レベルにご注意ください。
[b]Armadillo-610 拡張ボードでは GPIO で使用していますが、JTAG モード設定ピンですので、GPIO での使用は推奨しません。動作を理解した上でご使用ください。

以降では、CON2 から拡張可能な機能の概要について説明します。



拡張入出力となっている信号線のほとんどが、複数の機能をもっています。拡張できる機能の詳細につきましては、「Armadillo-610 マルチプレクス表」 [https://armadillo.atmark-techno.com/resources/documents/armadillo-610/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。

3.4.5.16. LAN(Ethernet)

LAN を 1 ポート拡張することが可能です。

信号線は Ethernet PHY(LAN8720AI-CP/Microchip Technology) を経由して i.MX6ULL の Ethernet コントローラ(ENET1)に接続されています。

- 通信速度: 10BASE-T/100BASE-TX(AUTO-MDIX 対応)

3.4.5.17. USB

USB を 2 ポート拡張可能で、Host は最大 2 ポート、OTG は最大 1 ポート拡張することが可能です。信号線は i.MX6ULL の USB コントローラ(USB_OTG1、USB_OTG2)に接続されています。

- USB 2.0
  - High Speed(480Mbps)
  - Full Speed(12Mbps)
  - Low Speed(1.5Mbps)

3.4.5.18. UART

シリアル(UART)を最大 8 ポート拡張することが可能です。信号線は i.MX6ULL の UART(UART1、UART2、UART3、UART4、UART5、UART6、UART7、UART8)に接続されています。

- 最大データ転送レート: 4Mbps
- 信号レベル: +3.3V_IO

3.4.5.19. SD/SDIO/MMC

CON1(SD インターフェース)と排他で SD/SDIO/MMC を 1 ポート拡張することが可能です。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。

- 最大クロック周波数: 49.5MHz
- 信号レベル: +3.3V_IO

3.4.5.20. LCD

LCD を最大 1 ポート拡張することが可能です。信号線は i.MX6ULL の LCD インターフェース(eLCDIF)に接続されています。

- 最大解像度: WXGA(1366 x 768/24bpp)
- 信号レベル: +3.3V_IO

3.4.5.21. I2S(SAI)

I2S を最大 2 ポート拡張することが可能です。信号線は i.MX6ULL の同期式オーディオインターフェース(SAI1、SAI3)に接続されています。

- 信号レベル: +3.3V_IO

3.4.5.22. MQS

MQS を最大 1 ポート拡張することが可能です。信号線は i.MX6ULL の Medium Quality Sound(MQS)に接続されています。

・信号レベル: +3.3V_IO

3.4.5.23. S/PDIF

S/PDIF を最大 1 ポート拡張することが可能です。信号線は i.MX6ULL の Sony/Philips デジタルインターフェース(SPDIF)に接続されています。

・信号レベル: +3.3V_IO

3.4.5.24. I2C

I2C を最大 2 ポート拡張することが可能です。信号線は i.MX6ULL の I2C コントローラ(I2C2、I2C4)に接続されています。

・最大データ転送レート: 400kbps

・信号レベル: +3.3V_IO

3.4.5.25. SPI

SPI を最大 4 ポート拡張することが可能です。信号線は i.MX6ULL の ESPI(ECSPI1、ECSPI2、ECSPI3、ECSPI4)に接続されています。

・信号レベル : +3.3V_IO

3.4.5.26. CAN

CAN を最大 2 ポート拡張することが可能です。信号線は i.MX6ULL の FLEXCAN(FLEXCAN1、FLEXCAN2)に接続されています。

・プロトコルバージョン 2.0B アクティブ対応

・信号レベル: +3.3V_IO

3.4.5.27. A/D

A/D を最大 8 ポート拡張することが可能です。信号線は i.MX6ULL の AD コンバーター(ADC1、ADC2)に接続されています。

・分解能: 最大 12 ビット

・サンプリングレート: 最大 1MS/s

・測定電圧範囲: DC 0～3.3V

3.4.5.28. PWM

PWM を最大 8 ポート拡張することが可能です。

・最大周波数: 66MHz

・信号レベル: +3.3V_IO

3.4.5.29. GPIO

GPIO を最大 66 ポート拡張することが可能です。

・信号レベル：+3.3V_IO

3.4.5.30. リアルタイムクロック

i.MX6ULL 内蔵のリアルタイムクロックを使用可能です。バックアップ用のピン(RTC_BAT)が接続されていますので、Armadillo-610 の電源(VIN)が切断されても時刻データを保持したい場合にご使用ください。

・平均月差: 約 70 秒@25 [mailto:約 70 秒@25]℃(参考値)

・バックアップ時間: 約 4 か月(CR2032 使用時の参考値)

3.4.6. 電気的仕様

3.4.6.1. 絶対最大定格

表 3.5 絶対最大定格

項目	記号	Min.	Max.	単位	備考
電源電圧	VIN	-0.3	4.8	V	—
入出力電圧(USB 信号以外)	VI,VO	-0.5	OVDD +0.3	V	OVDD=+3.3V_IO, VDD_SNVS_IN [a]
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
入力電圧(USB_VBUS)	VI_VBUS	—	5.5	V	USB_OTG1_VBUS, USB_OTG2_VBUS
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	℃	結露なきこと

[a]Armadillo-610: CON2 の 13 ピンのみ OVDD=VDD_SNVS_IN となります。VDD_SNVS_IN はダイオードを介して VSNVS と VDD_HIGH_IN の電源が供給されています。



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。

3.4.6.2. 推奨動作条件

表 3.6 推奨動作条件

項目	記号	Min.	Typ.	Max.	単位	備考
電源電圧	VIN	3.6	—	4.5	V	—
入力電圧 (USB_VBUS)	VI_VBUS	4.4	—	5.5	V	USB_OTG1_VBUS, USB_OTG2_VBUS
RTC バックアップ電源電圧	RTC_BAT	2.75	—	3.3	V	Topr=+25℃
使用温度範囲	Ta	-20	25	70	V	結露なきこと

3.4.6.3. 入出力インターフェースの電気的仕様

表 3.7 入出力インターフェース(電源)の電気的仕様

項目	記号	Min.	Typ.	Max.	単位	備考
5V 電源電圧	+5V_IO	4.8	5	5.15	V	—

項目	記号	Min.	Typ.	Max.	単位	備考
3.3V 電源電圧	+3.3V_IO	3.102	3.3	3.498	V	—
	VDD_HIGH_IN	3.201	3.3	3.399	V	—
セキュア用電源電圧	VSNVS	2.85	3.0	3.21	V	3.2V < VIN < 4.5V, OFF mode
		2.85	3.0	3.15	V	3.2V < VIN < 4.5V, On mode
		RTC_BAT-0.1	—	RTC_BAT	V	2.84V < RTC_BAT < 3.3V, Coin cell mode

表 3.8 入出力インターフェースの電気的仕様(OVDD = +3.3V_IO、VDD_SNVS_IN)

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。

3.4.6.4. 電源回路の構成

Armadillo-610 の電源回路の構成は「図 3.67. 電源回路の構成」のとおりです。

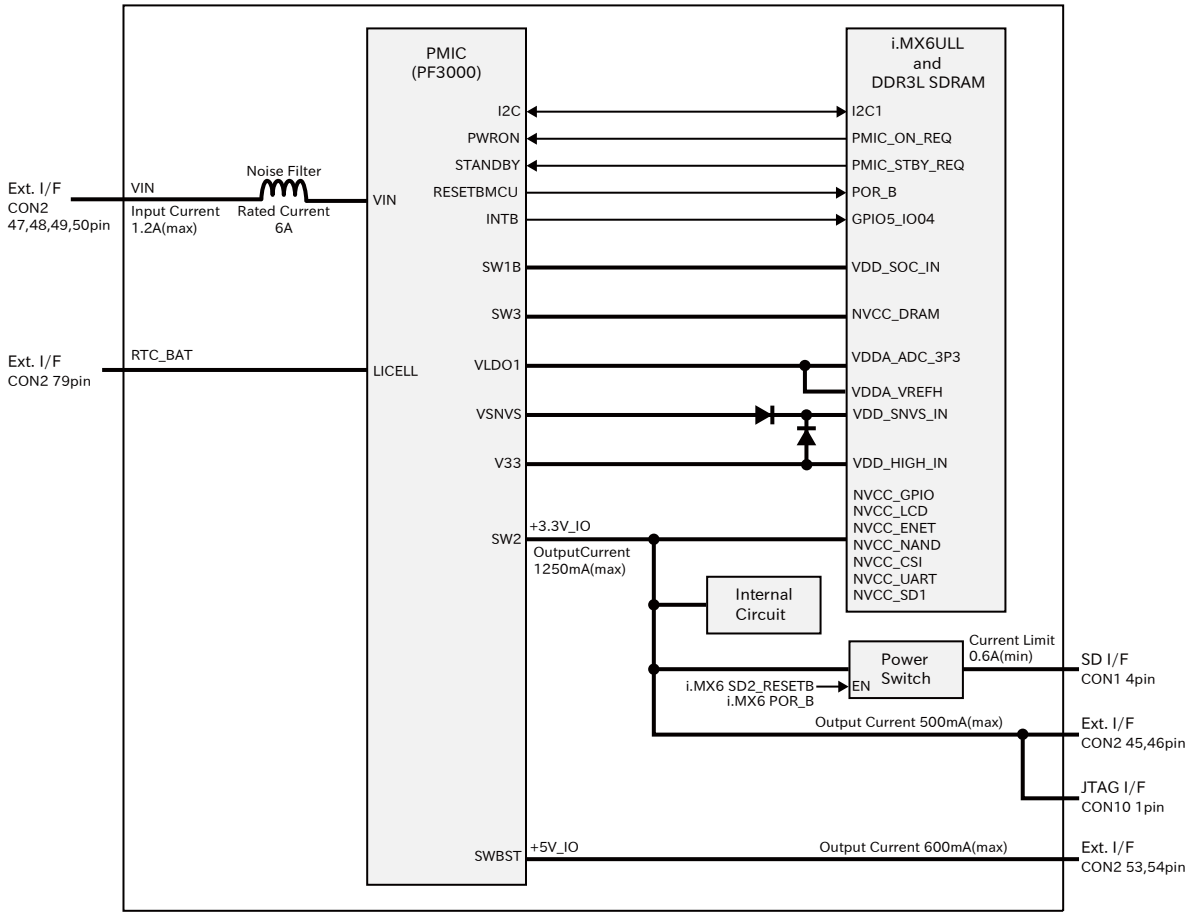


図 3.67 電源回路の構成

拡張インターフェース(Armadillo-610: CON2)からの入力電圧(VIN)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。

電源シーケンスは次のとおりです。

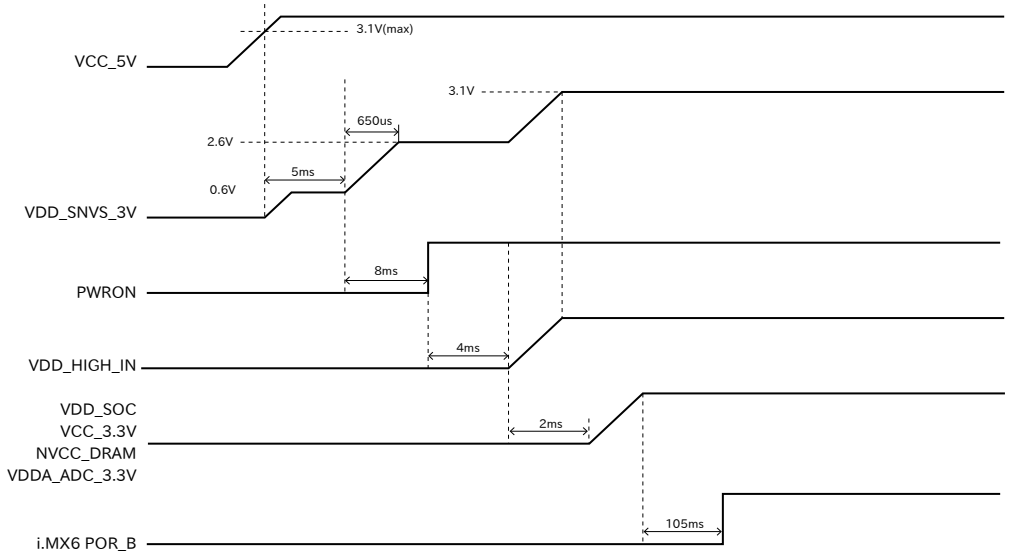


図 3.68 電源シーケンス [5]



USB_OTG1_VBUS, USB_OTG2_VBUS は電源シーケンスに関わらず、いつ電源を投入しても問題ありません。

3.4.6.5. リセット回路の構成

リセット回路の構成は「図 3.69. リセット回路の構成」のとおりです。

[5]T1: 任意のタイミング

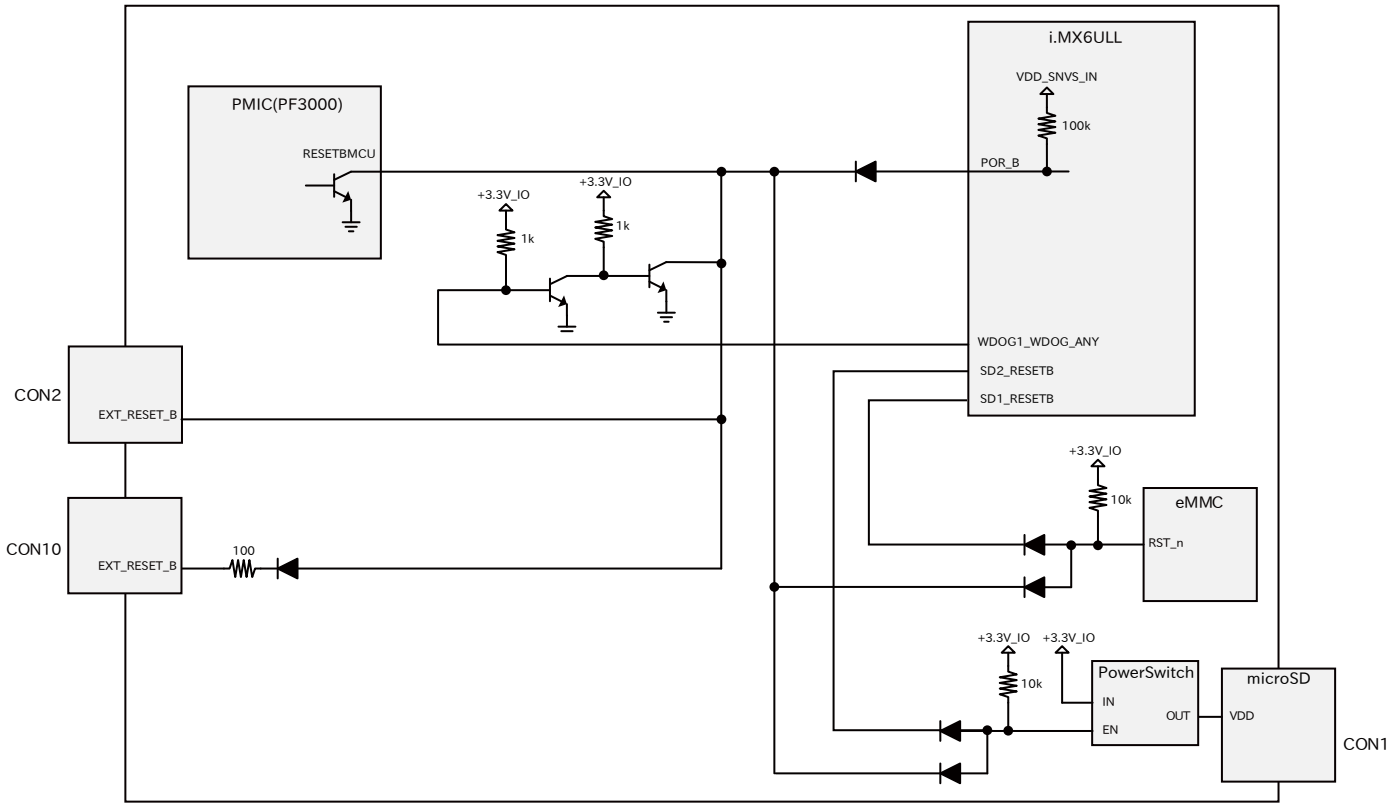


図 3.69 リセット回路の構成

拡張インターフェース(Armadillo-610: CON2)および JTAG インターフェース(Armadillo-610: CON10)の EXT_RESET_B ピンは i.MX6ULL の POR_B ピンに接続されています。EXT_RESET_B ピンから Low レベル出力することで、システムリセットすることができます。確実にシステムリセットするためには、20 ミリ秒以上 Low レベルを保持する必要があります。



図 3.70 システムリセットする場合の Low レベル保持時間

EXT_RESET_B ピンからシステムリセットする場合は、オープンドレイン出力等で GND とショートする回路を接続してください。

リセット信号が必要なデバイスを拡張する場合、拡張インターフェース(Armadillo-610: CON2)の EXT_RESET_B ピンを利用してリセットすることも可能です。

3.4.6.6. 外部からの電源制御

拡張インターフェース(Armadillo-610: CON2)の ONOFF ピンおよび PWRON ピンより、パワーマネジメント IC から i.MX6ULL への電源供給を制御することが可能です。

- ONOFF ピンからの電源制御

拡張インターフェース(Armadillo-610: CON2)の ONOFF ピンは i.MX6ULL の ONOFF ピンに接続されています。ONOFF ピンから一定時間以上 Low レベル出力することで、i.MX6ULL の保持している電源のオン状態、オフ状態が切り替わります。

電源がオフ状態に切り替わった場合、i.MX6ULL からパワーマネジメント IC の PWRON ピンに Low レベルが出力され、パワーマネジメント IC からの電源が切断されます。

電源オン状態からオフ状態に切り替える場合は 5 秒以上、電源オフ状態からオン状態に切り替える場合は 500 ミリ秒以上、Low レベルを保持する必要があります。

連続して電源オンとオフを切り替える場合は、確実に動作させるために 5 秒以上の間隔を空けてください。

ONOFF ピンから電源制御する場合は、オープンドレイン出力等で GND とショートする回路を接続してください。

表 3.9 ONOFF ピンから電源オン、オフ切り替えする際の Low 保持時間

状態	Low 保持時間
電源オンからオフ	5 秒以上
電源オフからオン	500 ミリ秒以上

電源オンまたはオフの状態は、拡張インターフェース(Armadillo-610: CON2)の VIN ピンと RTC_BAT ピンのどちらか一方でも電源が供給されている限り、保持されます。

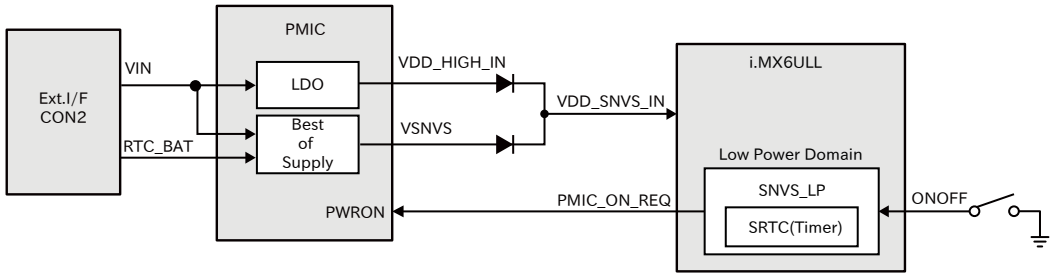


図 3.71 ONOFF 回路の構成 [6]



電源オフの状態にして拡張インターフェース(Armadillo-610: CON2)の VIN ピンからの電源を切断した場合、電荷が抜けるまでは電源オフであることが保持されます。電源オフを保持した状態で電源を投入したくない場合は、5 秒以上間隔を空けて電源を投入してください。

- PWRON ピンからの電源制御

  拡張インターフェース(Armadillo-610: CON2)の PWRON ピンはパワーマネジメント IC の PWRON ピンに接続されています。PWRON ピンから Low レベル出力することで、パワーマネジメント IC からの電源が切断されます。

  PWRON ピンから電源制御する場合は、オープンドレイン出力等で GND とショートする回路を接続してください。

[6]SNVS_LP：低消費電力ドメインです。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。

3.4.7. 形状図

3.4.7.1. 基板形状図

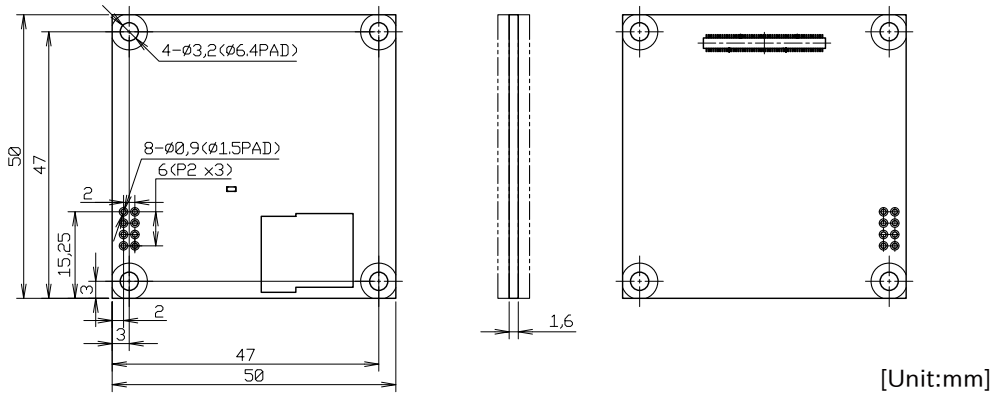


図 3.72 基板形状および固定穴寸法

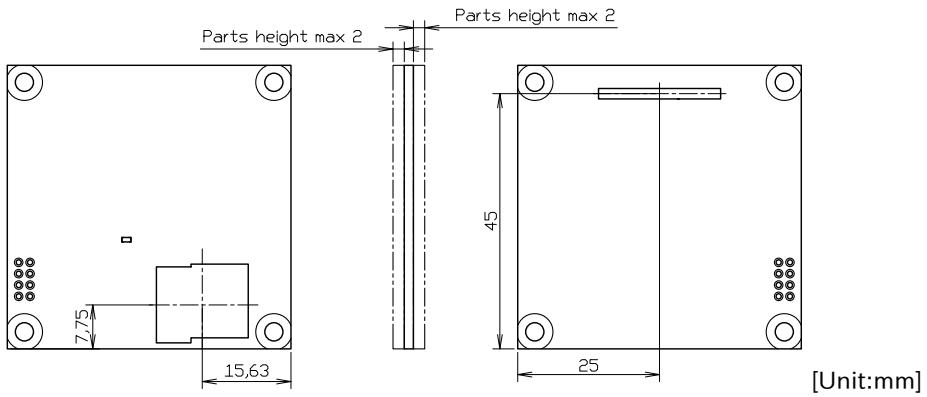


図 3.73 コネクタ中心寸法

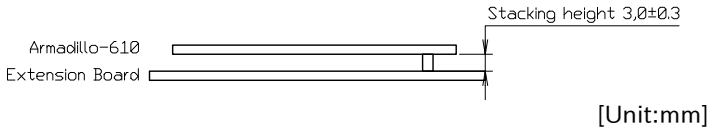


図 3.74 Armadillo-610 のスタッキング高さ



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 形式の基板形状図を、購入者向けの限定データとして「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。

3.4.8. オプション品

Armadillo-610 のオプション品については、「6.29. 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(Armadillo-610: CON2)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-610 Linux カーネル [https://armadillo.atmark-techno.com/resources/software/armadillo-610/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-at-a6-[VERSION]/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。

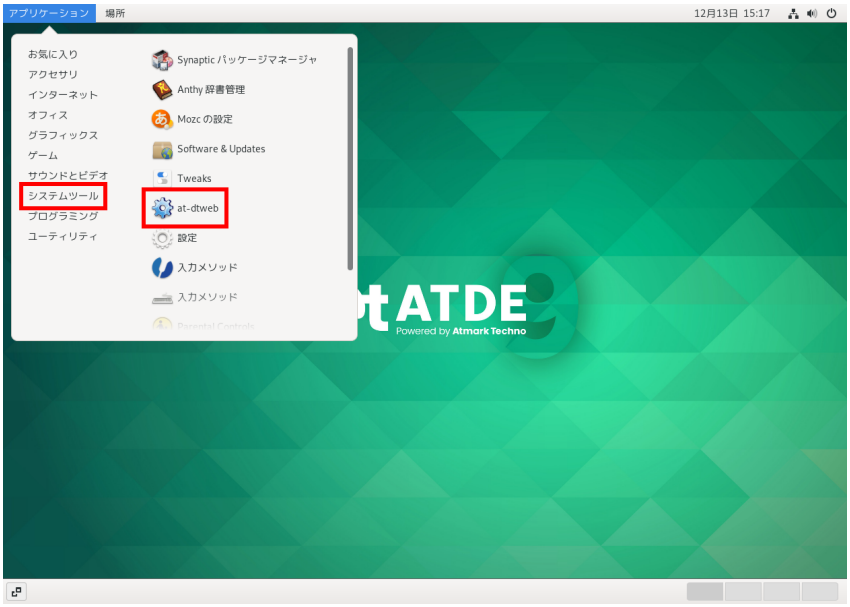


図 3.75 at-dtweb の起動開始

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-610 を選択して、「OK」をクリックします。

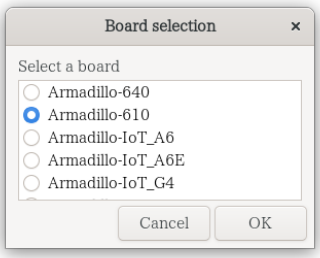


図 3.76 ボード選択画面

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。

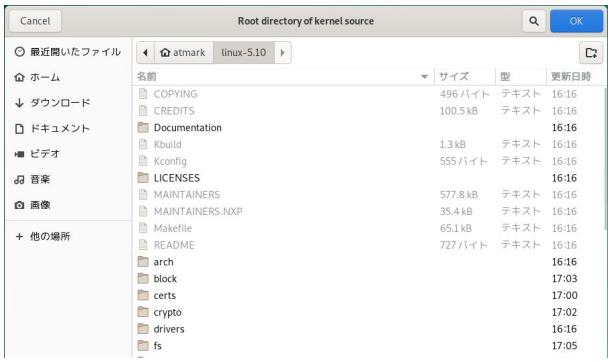


図 3.77 Linux カーネルディレクトリ選択画面

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。

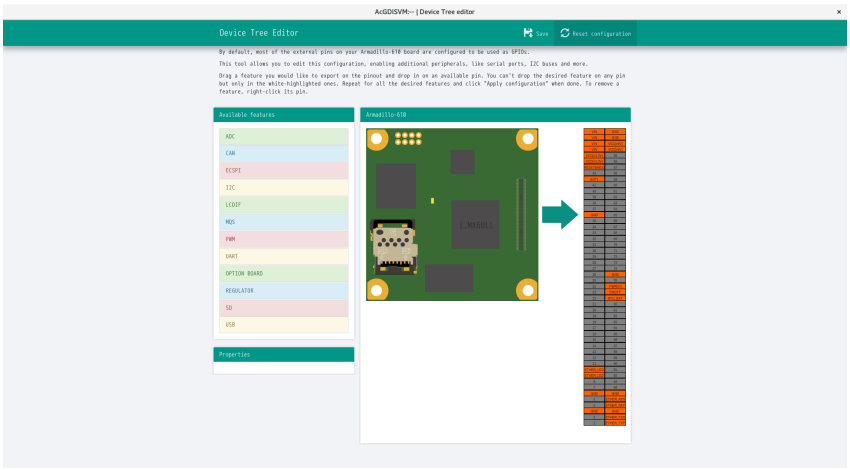


図 3.78 at-dtweb 起動画面

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-610」の白色に変化したピンにドロップします。例として CON2 83/85 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。GPIO の機能が自動的に割り当てられるのを防ぐには、「Unset GPIO」を対象のピンに割り当ててください。

at-dtweb で作成した dtbo と 独自にカスタマイズして作成した dtbo（*-customize.dtbo など）を Armadillo に両方適用する場合、at-dtweb

で自動的に GPIO に設定された pinctrl と、カスタマイズにより意図して設定した pinctrl が競合する可能性があります。

このような競合を防ぐためには、独自にカスタマイズしたデバイスツリーで使用したピンに対して、at-dtweb で「Unset GPIO」を割り当ててください。

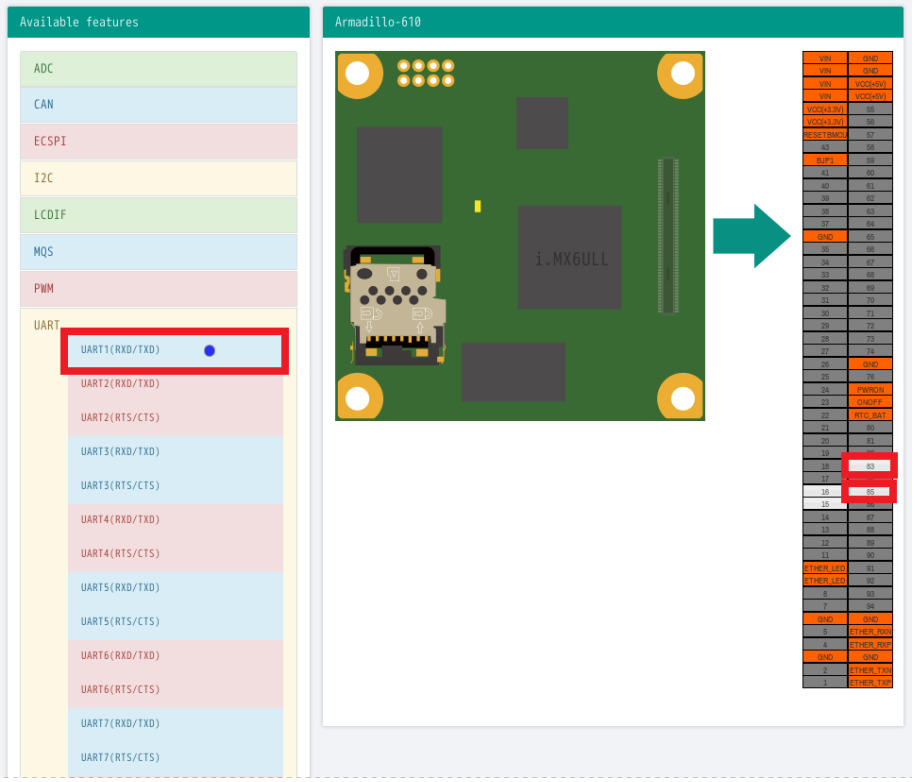


図 3.79 UART1(RXD/TXD)のドラッグ

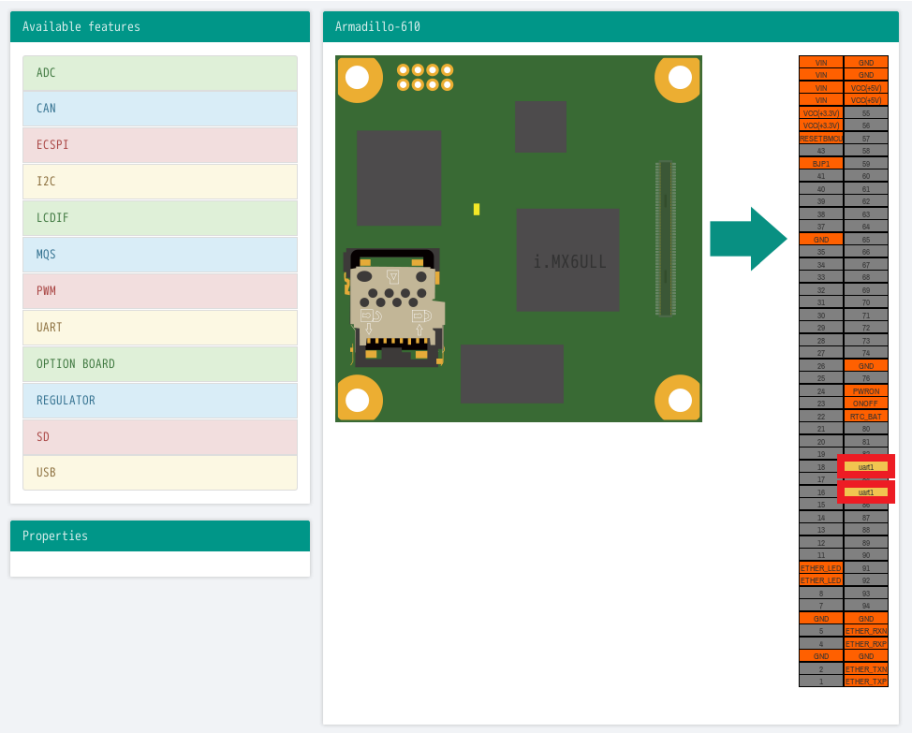


図 3.80 CON2 83/85 ピンへのドロップ

3.5.4.2. 信号名の確認

画面右側の「Armadillo-610」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。

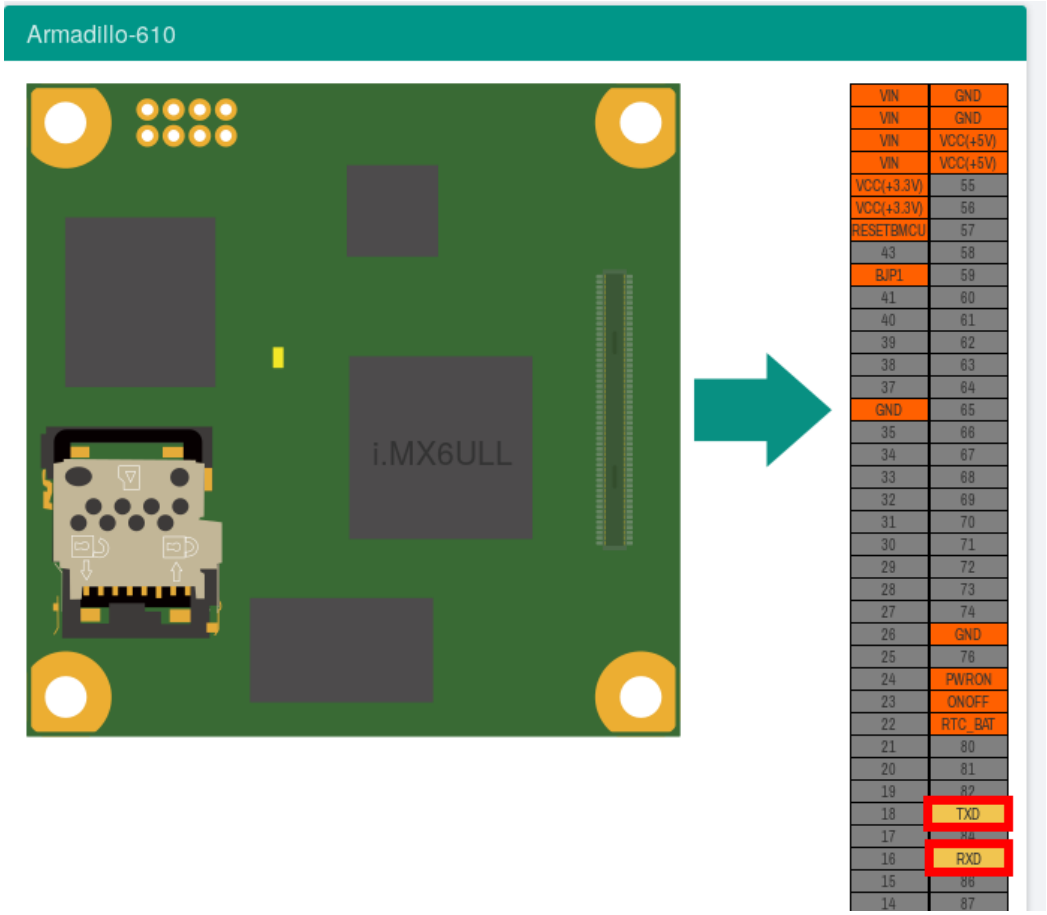


図 3.81 信号名の確認



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-610」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 CON2 88/89 ピンの I2C4(SCL/SDA)の clock_frequency プロパティを設定します。

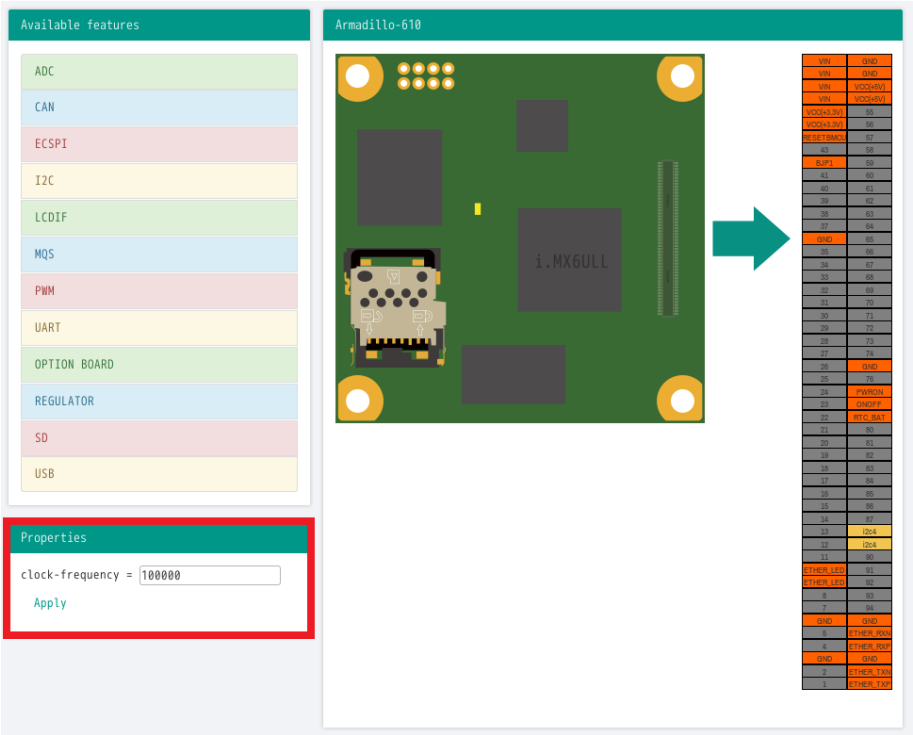


図 3.82 プロパティの設定

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。

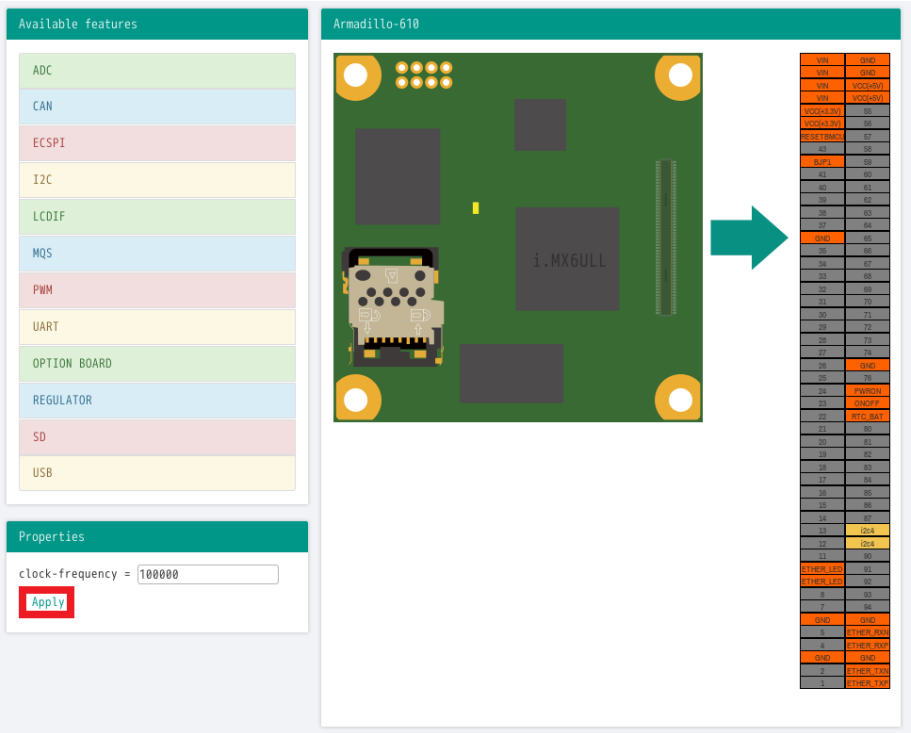


図 3.83 プロパティの保存

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-610」のピンを右クリックして「Remove」をクリックします。

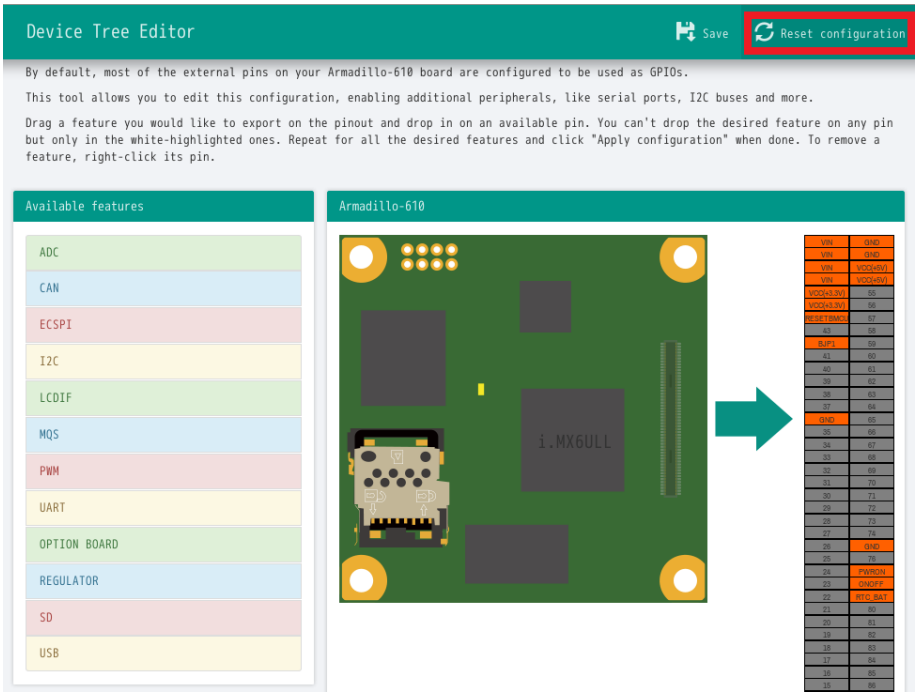


図 3.84 全ての機能の削除

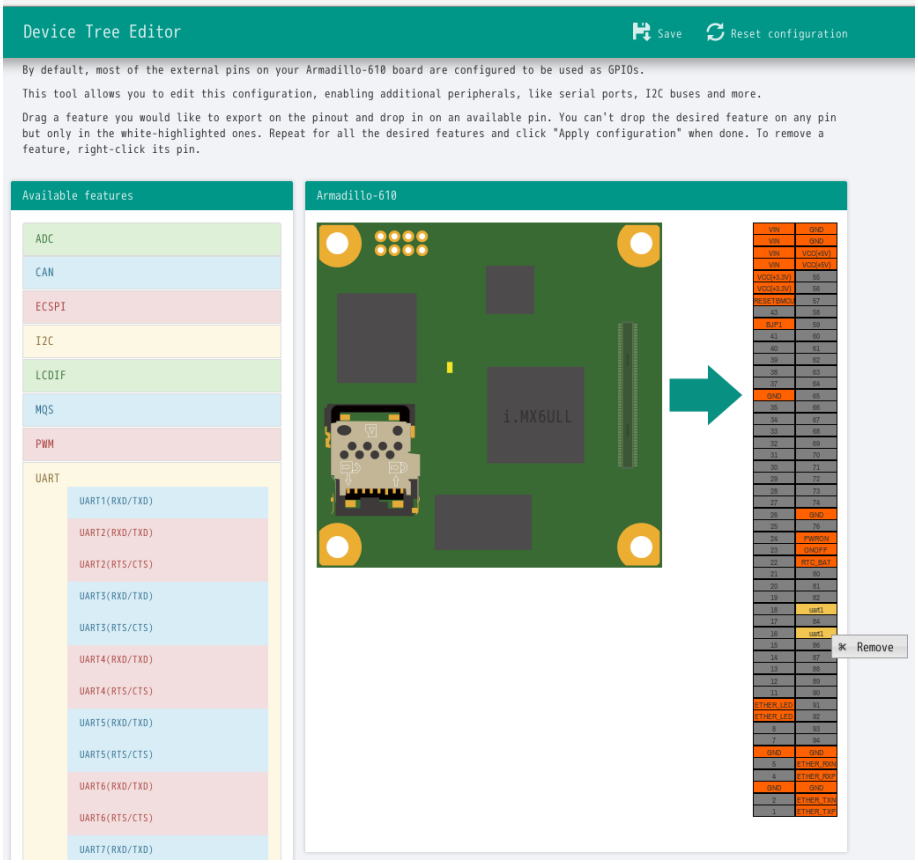


図 3.85 UART1(RXD/TXD)の削除

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。

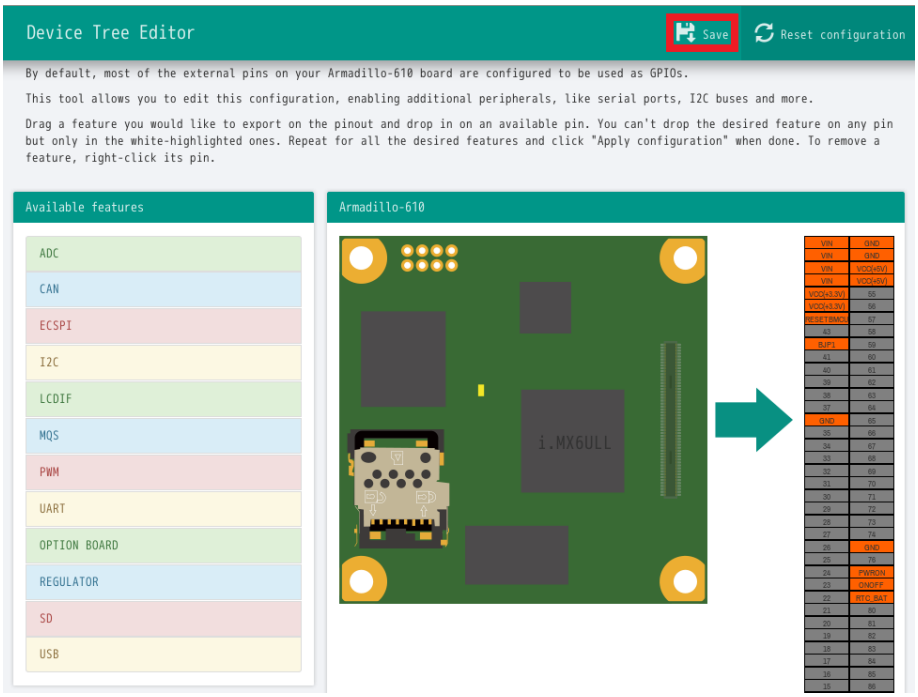


図 3.86 DTS/DTB の生成

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。

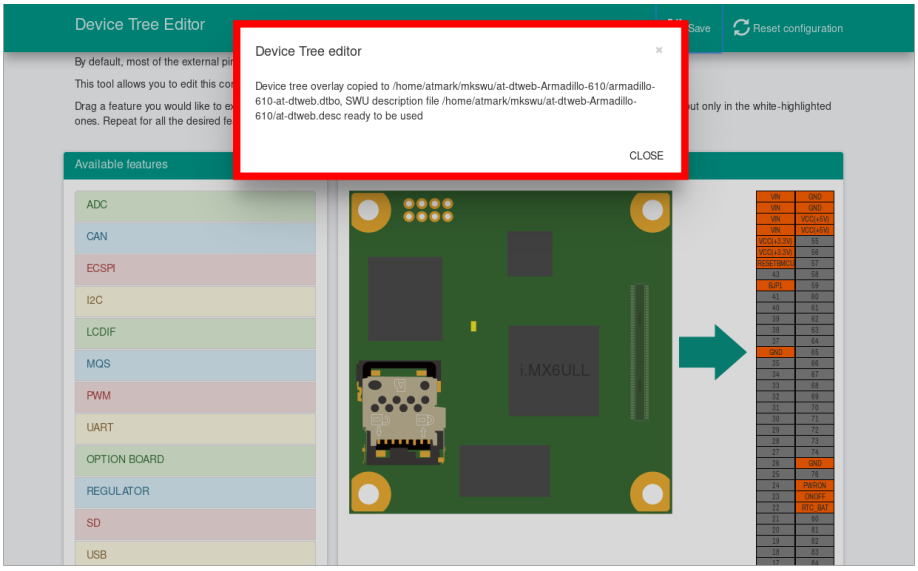


図 3.87 dtbo/desc の生成完了

ビルドが完了するとホームディレクトリ下の mkswu/at-dtweb-Armadillo-610 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-610 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-610
armadillo-610-at-dtweb.dtbo  update_overlays.sh
at-dtweb.desc                update_preserve_files.sh
[ATDE ~]$ cd ~/mkswu/at-dtweb-Armadillo-610
```

```
[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-610-onboard-usdhc2.dtbo armadillo-610-extboard-eva.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-610-onboard-usdhc2.dtbo ❹
Applying fdt overlay: armadillo-610-extboard-eva.dtbo
: (省略)
```

図 3.88 /boot/overlays.txt の変更例

❶ /boot/overlays.txt ファイルに「armadillo-610-extboard-eva.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-610-onboard-usdhc2.dtbo: SD インターフェース(Armadillo-610: CON1)を利用する場合にご使用ください。

- **armadillo-610-extboard-eva.dtbo**: Armadillo-610 拡張ボードを利用する場合にご使用ください。SD インターフェース(Armadillo-610 拡張ボード: CON1)、Grove インターフェース(Armadillo-610 拡張ボード: CON7、CON8、CON9、CON10)、LCD インターフェース(Armadillo-610 拡張ボード: CON11)を利用する場合は、後述の dtbo と合わせてご使用ください。
- **armadillo-610-extboard-eva-usdhc2.dtbo**: SD インターフェース(Armadillo-610 拡張ボード: CON1)を利用する場合にご使用ください。
- **armadillo-610-extboard-eva-grove.dtbo**: Grove インターフェース(Armadillo-610 拡張ボード: CON7、CON8、CON9、CON10)を利用する場合にご使用ください。
- **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-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。



armadillo-610-onboard-usdhc2.dtbo と armadillo-610-extboard-eva-usdhc2.dtbo を同時に使用することはできません。



armadillo-610-extboard-eva-grove.dtbo と armadillo-640-lcd70ext-l00.dtbo を同時に使用することはできません。



at-dtweb で作成した armadillo-610-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と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 3.89. Armadillo-610 のインターフェース」に Armadillo-610 の各インターフェースの位置を、「表 3.10. Armadillo-610 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。

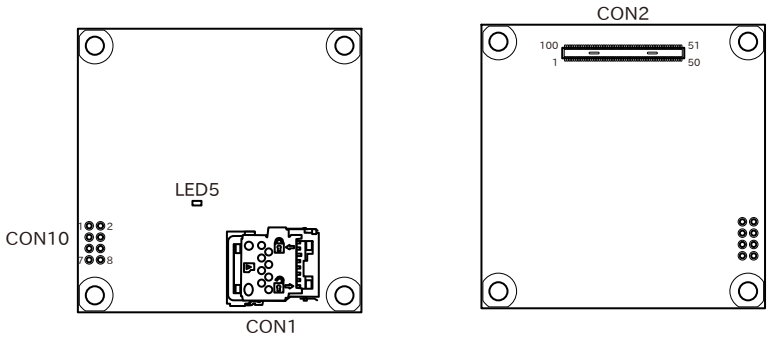


図 3.89 Armadillo-610 のインターフェース

表 3.10 Armadillo-610 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	拡張インターフェース	DF40C-100DP-0.4V(51)	HIROSE ELECTRIC
CON10	JTAG インターフェース	A3B-08PA-2DSA(51)	HIROSE ELECTRIC
LED5	ユーザー LED(黄)	SML-310YTT86	ROHM

[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。



「表 3.10. Armadillo-610 インターフェース一覧 」には搭載可能な代表型番を記載しており、実際に搭載されている部品型番と違う場合があります。お手元の製品に搭載されている部品型番や部品の実装、未実装の情報については、「アットマークテクノ Armadillo サイト」 [https://armadillo.atmark-techno.com/]からダウンロードできる納入仕様書および変更履歴表にてご確認ください。

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-610 の SD ホストは、i.MX6ULL の uSDHC (Ultra Secured Digital Host Controller) を利用しています。

「3.4.5.19. SD/SDIO/MMC」も合わせて参照してください。

Armadillo-610 開発セットでは、SD インターフェース(Armadillo-610: CON1) と SD インターフェース(Armadillo-610 拡張ボード: CON1)が uSDHC2 を共用しています。そのため、どちらか一方しか利用することができません。Armadillo-610 開発セットの標準状態では、SD インターフェース(Armadillo-610: CON1)が有効になっています。

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。

microSD 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 CON2(拡張インターフェース)でも利用可能ですが、排他利用となります。

表 3.11 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(+3.3V_IO)
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。

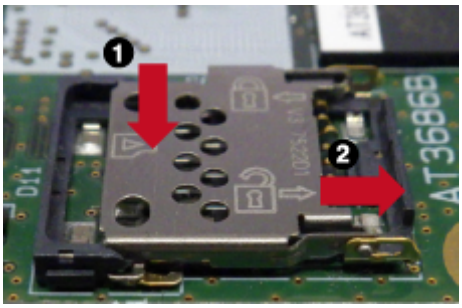


図 3.90 カバーのロックを解除する

2. カバーを開けます。

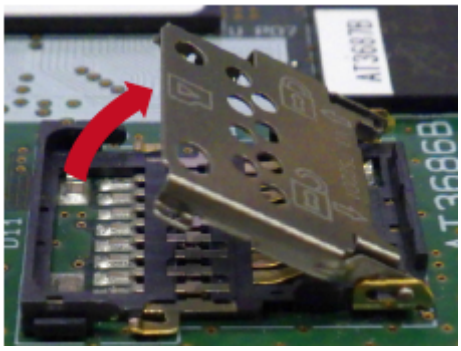


図 3.91 カバーを開ける



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 3.92 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 3.93 カードマークの確認

4. カバーを閉めます。

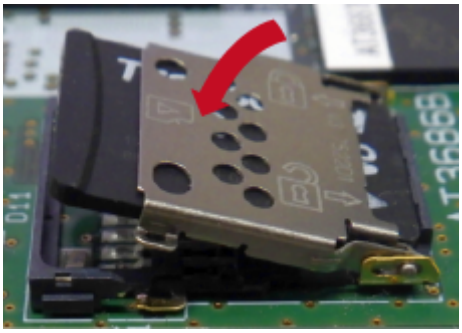


図 3.94 カバーを閉める

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。

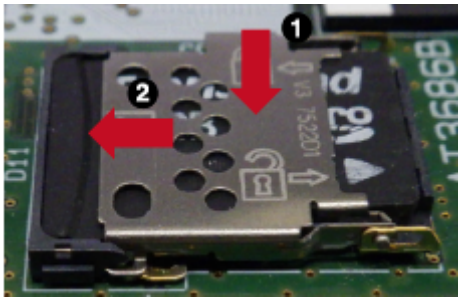


図 3.95 カバーをロックする



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-610 の Ethernet (LAN) は、i.MX6ULL の ENET(10/100-Mbps Ethernet MAC)を利用しています。

「3.4.5.16. LAN(Ethernet)」も合わせて参照してください。

Armadillo-610 開発セットでは、LAN インターフェース(Armadillo-610 拡張ボード: CON2)が ENET を利用しています。詳細は、「6.29.2.6. CON2(LAN インターフェース)」を参照してください。

機能 ・ 通信速度: 100Mbps (100BASE-TX), 10Mbps (10BASE-T)

- 通信モード: Full-Duplex (全二重), Half-Duplex (半二重)
- Auto Negotiation サポート
- キャリア検知サポート
- リンク検出サポート

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-610 のシリアルは、i.MX6ULL の UART (Universal Asynchronous Receiver/Transmitter) を利用しています。

「3.4.5.18. UART」も合わせて参照してください。

Armadillo-610 開発セットの標準状態では、シリアルインターフェース(Armadillo-610 拡張ボード: CON3)が UART1 をコンソールとして利用しています。また、Grove インターフェース(Armadillo-610 拡張ボード: CON7)には UART5 を利用しています。詳細は、「6.29.2.7. CON3(シリアルインターフェース)」及び、「6.29.2.11. CON7、CON8、CON9、CON10(Grove インターフェース)」を参照してください。

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

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/ttymxc0
UART3	/dev/ttymxc2
UART5	/dev/ttymxc4

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/ttymxcN を渡す必要があります。以下は、/dev/ttymxc2 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/ttymxc2
[armadillo ~]# podman_start serial_example
Starting 'serial_example'
3999f09d51253371cacffd68967c90fdd5250770888a82f59d7810b54fcc873e
```

図 3.96 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/ttymxc2
/dev/ttymxc2, Line 2, UART: undefined, Port: 0x0000, IRQ: 52
        Baud_base: 5000000, close_delay: 50, divisor: 0
        closing_wait: 3000
        Flags: spd_normal
```

図 3.97 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

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-610 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。

  「3.4.5.17. USB」も合わせて参照してください。

  Armadillo-610 開発セットでは、USB ハブが USB_OTG1 を利用しています。USB インターフェース(Armadillo-610 拡張ボード: CON5)には USB ハブが接続されています。詳細は、「6.29.2.9. CON5(USB ホストインターフェース)」を参照してください。

  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)

- USB ハブ

  Armadillo-610 拡張ボードには、Microchip 製 USB2513B が搭載されています。USB2513B には、WLAN インターフェース(Armadillo-610 拡張ボード: CON18)および USB インターフェース(Armadillo-610 拡張ボード: CON5)が接続されています。

  機能
  - USB specification rev 2.0 準拠

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



USB が認識されない場合、USB デバイスの接続を拒否する設定が行われている可能性があります。

設定を変更するには「3.9. USB デバイスの接続を許可する」をご確認ください。

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 3.98 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 3.99 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/ttyUSBN 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 3.100 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 3.101 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 3.102 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 3.103 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 3.104 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 3.105 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. 音声出力を行う

Armadillo-610 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 3.106 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 3.107 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。

3.6.6. GPIO を制御する

3.6.6.1. ハードウェア仕様

Armadillo-610 の GPIO は、i.MX6ULL の GPIO(General Purpose Input/Output)を利用しています。

「3.4.5.29. GPIO」も合わせて参照してください。

3.6.6.2. ソフトウェア仕様

デバイスファイル
- /dev/gpiochip0
- /dev/gpiochip1

- /dev/gpiochip2
- /dev/gpiochip3
- /dev/gpiochip4



sysfs GPIO クラス（/sys/class/gpio/）は使用しないでください。



Linux 5.10.233-r0 以降では拡張インターフェースの GPIO に「CON8_14」の様な名前でアクセス可能となりました。利用できるピンは gpioinfo コマンドでご確認ください。

3.6.6.3. 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、GPIO**N+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'
956a0fecc48d5ea1210069910f7bb48b9e90b2dadb12895064d9776dae0360b5
```

図 3.108 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「図 3.109. コンテナ内からコマンドで GPIO を操作する例」に示します。この例では DIDO インターフェース(Armadillo-610 拡張ボード: CON13B) の 4 ピン(GPIO1_IO20) を操作しています。



libgpiod バージョンの注意

gpioset, gpioget, または gpioinfo のコマンドは libgpiod のバージョンによって引数の仕様が変わります。

Armadillo Base OS では v3.20 までは libgpiod 1, v3.21 以降は libgpiod 2 がインストールされています。コンテナの場合は Alpine 3.21 および Debian trixie 以降では libgpiod 2 に更新されています。

使い方の違いは https://armadillo.atmark-techno.com/howto/libgpiod2-update をご参照ください。

本書では libgpiod 2 準拠で手順を紹介します。

```
[armadillo ~]# podman exec -it gpio_example sh
[container ~]# apk upgrade
[container ~]# apk add libgpiod
[container ~]# gpioget --numeric -c gpiochip0 20 ❶
1 ❷
[container ~]# gpioset -t0 -c gpiochip0 20=0 ❸
```

図 3.109 コンテナ内からコマンドで GPIO を操作する例

❶ GPIO1 番号 20 の値を取得します。

❷ 取得した値を表示します。

❸ GPIO1 番号 20 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 3.110 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 -c gpiochip0
gpiochip0 - 32 lines:
        line   0:       unnamed                 input
        line   1:       unnamed                 input
        line   2:       unnamed                 input
        line   3:       unnamed                 input
        line   4:       unnamed                 input
        line   5:       unnamed                 input
        line   6:       unnamed                 input
        line   7:       unnamed                 input
        line   8:       unnamed                 output consumer=green
        line   9:       unnamed                 output
        line  10:       unnamed                 input consumer=SW1
        line  11:       unnamed                 input
        line  12:       unnamed                 input
        line  13:       unnamed                 input
        line  14:       unnamed                 input
        line  15:       unnamed                 input
        line  16:       unnamed                 input
        line  17:       unnamed                 input
        line  18:       unnamed                 output consumer=yellow
        line  19:       unnamed                 output consumer=regulators:regulator-usbotg1vbu
```

```
        line  20:       unnamed                 input
        line  21:       unnamed                 input
        line  22:       unnamed                 input
        line  23:       unnamed                 input
        line  24:       unnamed                 input
        line  25:       unnamed                 output consumer=awlan-reset
        line  26:       unnamed                 input
        line  27:       unnamed                 input
        line  28:       unnamed                 input
        line  29:       unnamed                 input
        line  30:       unnamed                 input
        line  31:       unnamed                 input
```

図 3.111 gpioinfo コマンドの実行

デフォルトソフトウェアでは、DIDO インターフェース(Armadillo-610 拡張ボード: CON13B)の各ピンを GPIO として利用することができます。ピン番号と GPIO 番号の対応を次に示します。

表 3.12 Armadillo-610 拡張ボード: CON13B ピン番号と GPIO 番号の対応

ピン番号	GPIO 番号
4	GPIO1_20
5	GPIO1_21
6/7	GPIO1_30
8/9	GPIO1_31

at-dtweb を利用すると、「表 3.12. Armadillo-610 拡張ボード: CON13B ピン番号と GPIO 番号の対応」以外のピンも GPIO として利用することができるようになります。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。

拡張インターフェース(Armadillo-610: CON2)のピン番号と GPIO 番号の対応を次に示します。

表 3.13 Armadillo-610: CON2 ピン番号と GPIO 番号の対応

ピン番号	GPIO 番号
11	GPIO1_IO19
12	GPIO4_IO17
14	GPIO1_IO04
15	GPIO1_IO03
16	GPIO1_IO02
17	GPIO1_IO01
18	GPIO3_IO05
19	GPIO3_IO06
20	GPIO3_IO07
21	GPIO3_IO08
22	GPIO3_IO09
23	GPIO3_IO10
24	GPIO3_IO11
25	GPIO3_IO12
26	GPIO3_IO13
27	GPIO3_IO14
28	GPIO3_IO15
29	GPIO3_IO16
30	GPIO3_IO17
31	GPIO3_IO18
32	GPIO3_IO19

ピン番号	GPIO 番号
33	GPIO3_IO20
34	GPIO3_IO21
35	GPIO3_IO22
37	GPIO3_IO00
38	GPIO3_IO02
39	GPIO3_IO03
40	GPIO3_IO01
41	GPIO4_IO16
43	GPIO1_IO10
55	GPIO4_IO19
56	GPIO4_IO20
57	GPIO4_IO25
58	GPIO4_IO26
59	GPIO4_IO27
60	GPIO4_IO28
61	GPIO4_IO23
62	GPIO4_IO22
63	GPIO4_IO24
64	GPIO4_IO21
65	GPIO4_IO18
66	GPIO4_IO09
67	GPIO4_IO08
68	GPIO4_IO07
69	GPIO4_IO06
70	GPIO3_IO28
71	GPIO3_IO27
72	GPIO3_IO26
73	GPIO3_IO25
74	GPIO3_IO24
76	GPIO3_IO23
80	GPIO1_IO08
81	GPIO1_IO05
82	GPIO1_IO30
83	GPIO1_IO16
84	GPIO1_IO31
85	GPIO1_IO17
86	GPIO1_IO23
87	GPIO1_IO22
88	GPIO1_IO21
89	GPIO1_IO20
90	GPIO1_IO00
91	GPIO1_IO26
92	GPIO1_IO27
93	GPIO1_IO25
94	GPIO1_IO24

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

3.6.7. I2C デバイスを使用する

3.6.7.1. ハードウェア仕様

Armadillo-610 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。

「3.4.5.24. I2C」も合わせて参照してください。

Armadillo-610 開発セットで利用している I2C バスと、接続される I2C デバイスを次に示します。

表 3.14 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
1(I2C2)	0x32	NR3225SA

3.6.7.2. ソフトウェア仕様

Armadillo-610 開発セットの標準状態では、CONFIG_I2C_CHARDEV が有効となっているためユーザードライバで I2C デバイスを制御することができます。ユーザードライバを利用する場合は、Linux カーネルで I2C デバイスに対応するデバイスドライバを無効にする必要があります。

機能
- 最大転送レート: 384kbps

デバイスファイル
- /dev/i2c-0 (I2C1)
- /dev/i2c-1 (I2C2)

3.6.7.3. 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-1 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-1
[armadillo ~]# podman_start i2c_example
Starting 'i2c_example'
efa1eb129c1f036a709755f0d53b21a0f2a39307ecae32b24aac98c0b6567bf0
```

図 3.112 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 1
```

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

図 3.113 i2cdetect コマンドによる確認例

3.6.8. SPI デバイスを使用する

3.6.8.1. ハードウェア仕様

「3.4.5.25. SPI」を参照してください。

3.6.8.2. 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 3.114 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 3.115 spi-config コマンドによる確認例

3.6.9. CAN デバイスを使用する

3.6.9.1. ハードウェア仕様

「3.4.5.25. SPI」を参照してください。

3.6.9.2. 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 docker.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 3.116 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 3.117 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 125 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。

3.6.10. PWM を使用する

3.6.10.1. ハードウェア仕様

「3.4.5.28. PWM」を参照してください。

3.6.10.2. 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 3.118 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 3.119 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。

3.6.11. 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。



JTAG のセキュリティ状態は eFuse で変更することが可能です。



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 CON2(拡張インター

フェース)に接続されています。詳細は、「6.29.2.6. CON2(LAN インターフェース)」を参照してください。

表 3.15 CON10 信号配列

ピン番号	ピン名	I/O	説明
1	+3.3V_IO	Power	電源(+3.3V_IO)
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(+3.3V_IO)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(+3.3V_IO)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(+3.3V_IO)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(+3.3V_IO)されています。
6	JTAG_TDO	Out	テストデータ出力、i.MX6ULL の JTAG_TDO ピンに接続
7	EXT_RESET_B	In	システムリセット、i.MX6ULL の POR_B ピンに接続、オープンドレイン入力
8	GND	Power	電源(GND)

3.6.12. LCD を使用する

3.6.12.1. ハードウェア仕様

Armadillo-610 の LCD ホストは、i.MX6ULL の eLCDIF(Enhanced LCD Interface)を利用しています。

「3.4.5.20. LCD」も合わせて参照してください。

Armadillo-610 開発セットでは、LCD インターフェース(Armadillo-610 拡張ボード: 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。詳細は「6.29.2.12. CON11(LCD インターフェース)」を参照してください。

3.6.12.2. 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)

3.6.12.3. 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「6.29.3. 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 3.120. 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-610 拡張ボードの CON11 の 1 ピンが対応するように、FFC を接続します。

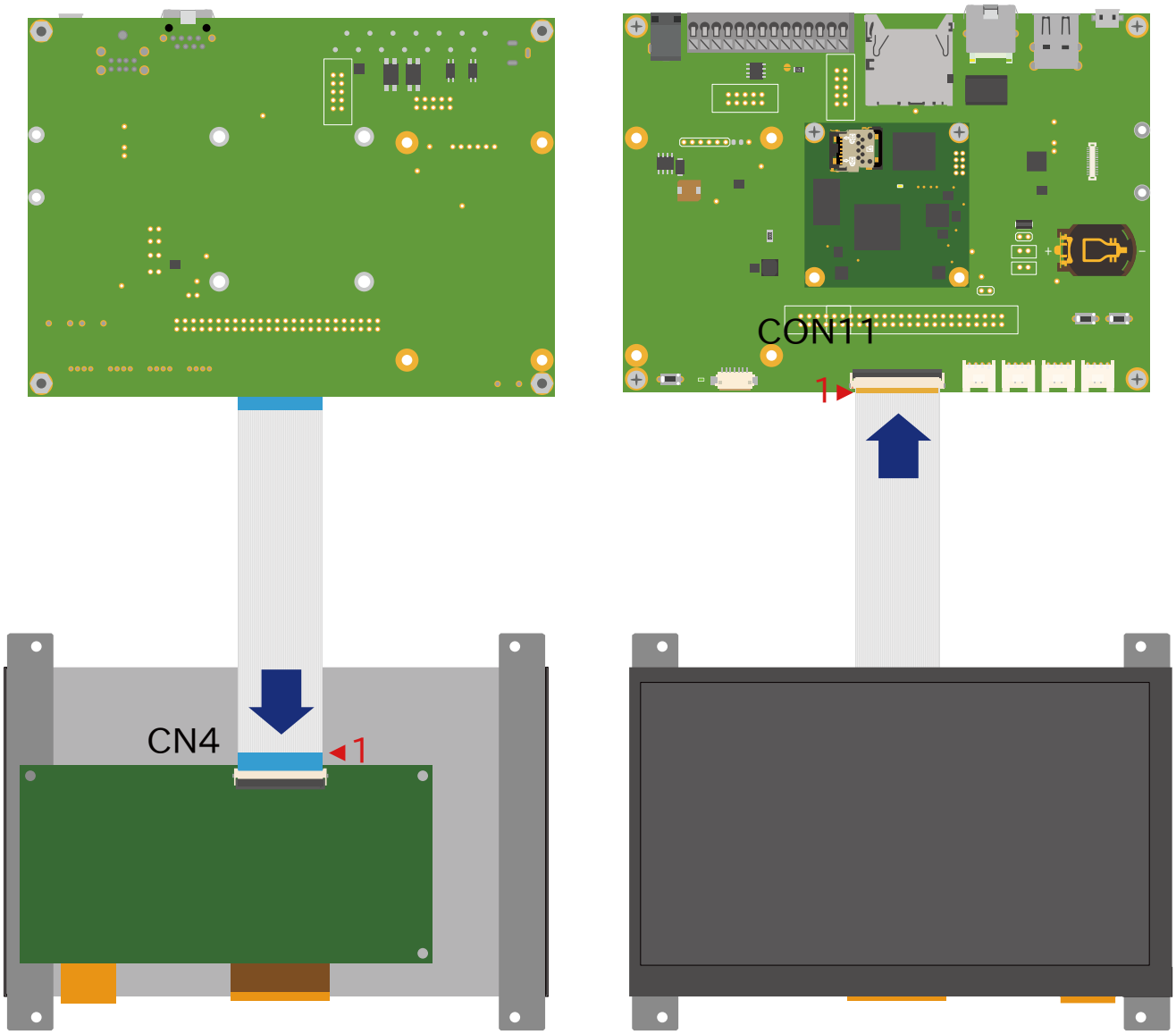


図 3.120 LCD の接続方法



図 3.121 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。

「3.5.5. DT overlay によるカスタマイズ」 を行っている場合のみ、DT overlay の設定を行います。そうでない場合は行う必要はありません。

```
[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。

3.6.13. ユーザースイッチを使用する

3.6.13.1. ハードウェア仕様

Armadillo-610 では、GPIO ピンを GPIO 接続用キーボードドライバ(gpio-keys)に設定することで、スイッチとして使用することが可能です。

Armadillo-610 拡張ボード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。詳細は「6.29.2.29. SW1(ユーザースイッチ)」を参照してください。

3.6.13.2. ソフトウェア仕様(Armadillo-610 拡張ボードの場合)

Armadillo-610 開発セットの標準状態を例に説明します。

Linux では、ユーザー空間でイベント(Press/Release)を検出することができます。Linux では、GPIO 接続用キーボードドライバ(gpio-keys)で制御することができます。

ユーザースイッチと信号には、次に示すキーコードが割り当てられています。

表 3.16 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 [7]

3.6.13.3. 使用方法

Armadillo-610 拡張ボード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。

[7]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

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

図 3.122 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 3.123 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。

3.6.14. LED を使用する

Armadillo-610 および Armadillo-610 拡張ボードに搭載されているソフトウェア制御可能な LED には、GPIO が接続されています。

3.6.14.1. ハードウェア仕様

Armadillo-610 に搭載されている LED5 は、ユーザー側で自由に利用できる LED です。

表 3.17 LED5

部品番号	名称(色)	説明
LED5	ユーザー LED(黄)	i.MX6ULL の UART1_CTS_B ピンに接続、(Low: 消灯、High: 点灯)

Armadillo-610 拡張ボードに搭載されているソフトウェア制御可能な LED については「6.29.2.33. LED3(ユーザー LED)」を参照してください。

3.6.14.2. 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/green (LED3) ・ /sys/class/leds/yellow (LED5)

3.6.14.3. 使用方法

Armadillo-610 および Armadillo-610 拡張ボードには 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 3.124 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 3.125 LED の点灯/消灯の実行例

LED クラスディレクトリと LED の対応を次に示します。

表 3.18 LED クラスディレクトリと LED の対応

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

3.6.15. RTC を使用する

3.6.15.1. ハードウェア仕様

Armadillo-610 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。

また、Armadillo-610 拡張ボードには、日本電波工業(NDK)製 NR3225SA が搭載されています。NR3225SA は、「3.6.7. I2C デバイスを使用する」に示す I2C2 (I2C ノード: 2-0032) に接続されています。



LCD インターフェース(Armadillo-610 拡張ボード: CON11)および拡張インターフェース(Armadillo-610 拡張ボード: CON20)には、NR3225SA に接続された I2C と共通の信号線が接続されています。そのため、同時に利用できない場合があります。

「3.4.5.30. リアルタイムクロック」も合わせて参照してください。

機能 ・ アラーム割り込みサポート

3.6.15.2. 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
- /dev/rtc1

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。

3.6.15.3. 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。

3.6.16. BT デバイスを使用する



標準出荷状態の Armadillo Base OS には BlueZ をはじめとした BT のプロトコルスタックがインストールされていません。動作確認時は以下の手順に従って、コンテナ内にプロトコルスタックをインストールして行ってください。

コンテナ内から BT 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 3.128. BT を扱うコンテナの作成例」に、alpine イメージから BT 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_args --cap-add=NET_ADMIN
[armadillo ~]# podman_start bt_example
Starting 'bt_example'
45fe1eb6b25529f0c84cd4b97ca1aef8451785fc9a87a67d54873c1ed45b70a4
```

図 3.128 BT 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、BT を起動します。

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

図 3.129 BT を起動する実行例

これにより、bluetoothctl で BT 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 3.130 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。

3.6.17. 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/ttymxcN 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/ttymxc1 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/ttymxc1
[armadillo ~]# podman_start wisun_example
Starting 'wisun_example'
ef9a5a2f7eee4236cb28c1fbf5090a6f0db9d6dfe7f3a34573867e0833dd3122
[armadillo ~]# podman exec -it wisun_example sh
[container ~]# ls /dev/ttymxc1
/dev/ttymxc1
```

図 3.131 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/ttymxc1 を使って Wi-SUN データの送受信ができるようになります。

3.6.18. 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/ttymxcN 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/ttymxc1 を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# vi /etc/atmark/containers/enocean_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/ttymxc1
[armadillo ~]# podman_start enocean_example
Starting 'enocean_example'
a808b491a100f9078d8c72a7b36966d9182614f3657fe054fb8d7f87b0d4b31c
[armadillo ~]# podman exec -it enocean_example sh
[container ~]# ls /dev/ttymxc1
/dev/ttymxc1
```

図 3.132 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/ttymxc1 を使って EnOcean データの送受信ができるようになります。

3.6.19. Thread デバイスを扱う

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

```
[armadillo ~]# vi /etc/atmark/containers/thread_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/ttyACM0
[armadillo ~]# podman_start thread_example
Starting 'thread_example'
a808b491a100f9078d8c72a7b36966d9182614f3657fe054fb8d7f87b0d4b31c
[armadillo ~]# podman exec -it thread_example sh
[container ~]# ls /dev/ttyACM0
/dev/ttyACM0
```

図 3.133 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。

3.7. ソフトウェアの設計

Armadillo-610 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 3.134. 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。

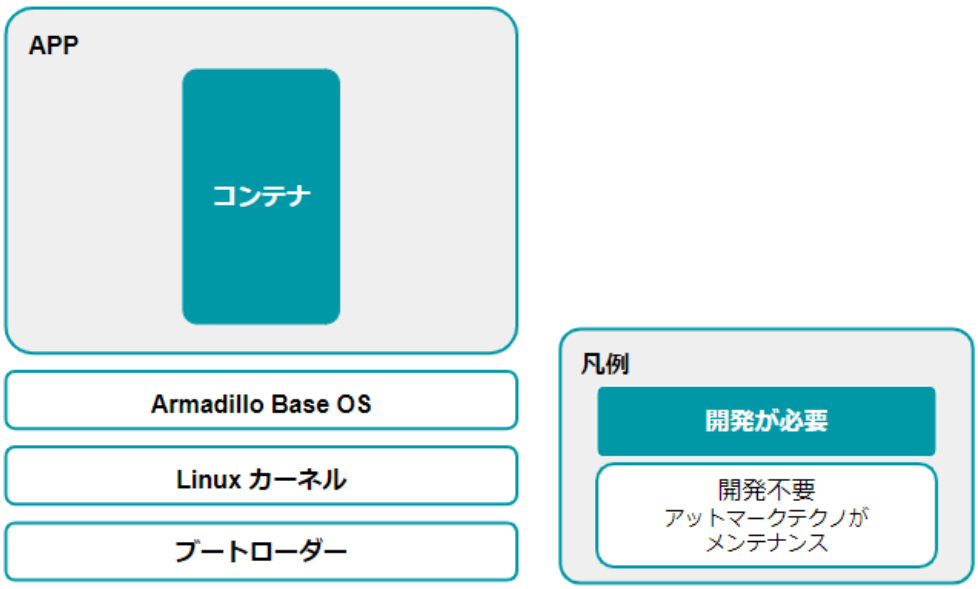


図 3.134 開発者が開発するもの、開発しなくていいもの

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-610 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 3.135. 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 3.135 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-610 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 3.136 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 3.137 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 3.19. 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 3.19 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 3.138. 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 3.138 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 3.139. 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 3.139 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。

❹ 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 で開くことが出来ます。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 3.140. 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。

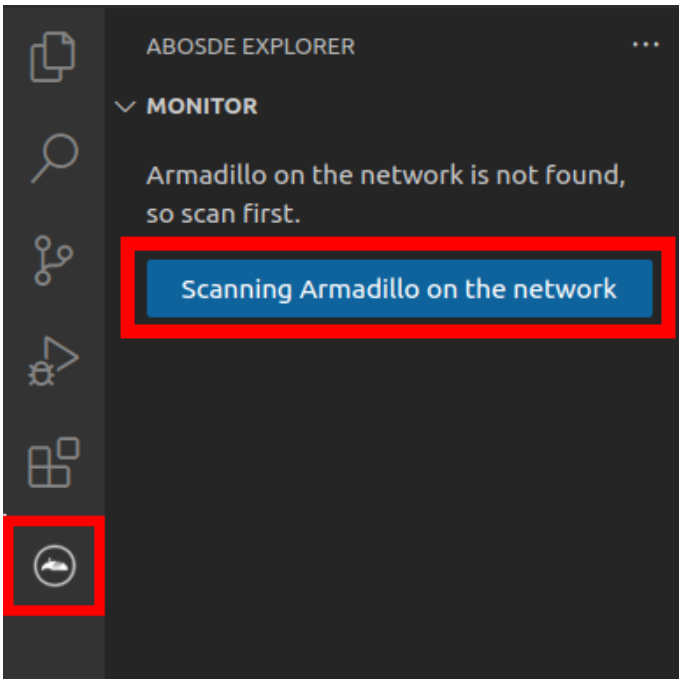


図 3.140 ABOSDE で ローカルネットワーク上の Armadillo をスキャンする

「図 3.141. 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。

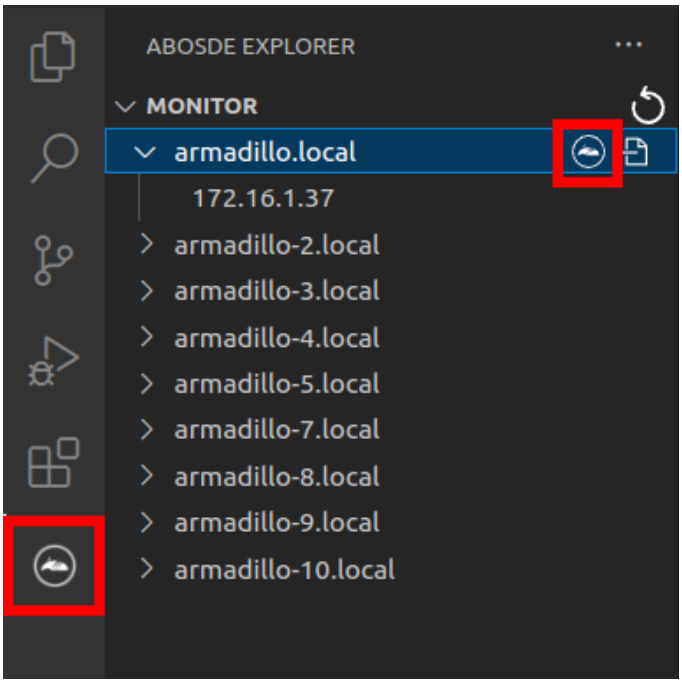


図 3.141 ABOSDE を使用して ABOS Web を開く

「図 3.142. 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。

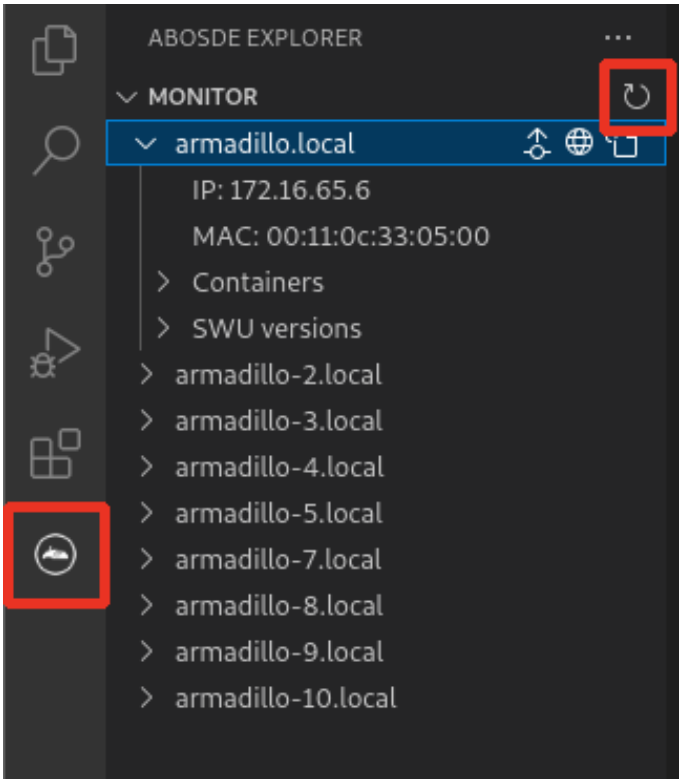


図 3.142 ABOSDE に表示されている Armadillo を更新する

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 3.143 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 3.144 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 3.145 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。

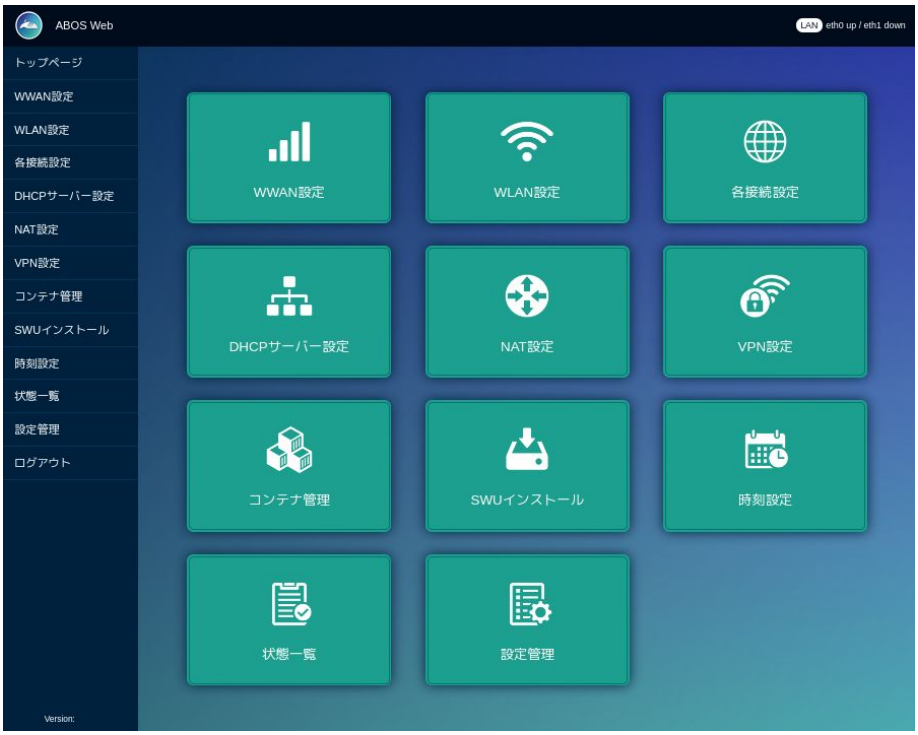


図 3.146 トップページ

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。

「パスワードリセット」画面では、ABOS Web のパスワードをリセットすることができます。リセットを行うと、次回、ABOS Web にログインする際にパスワードの再設定を求められます。

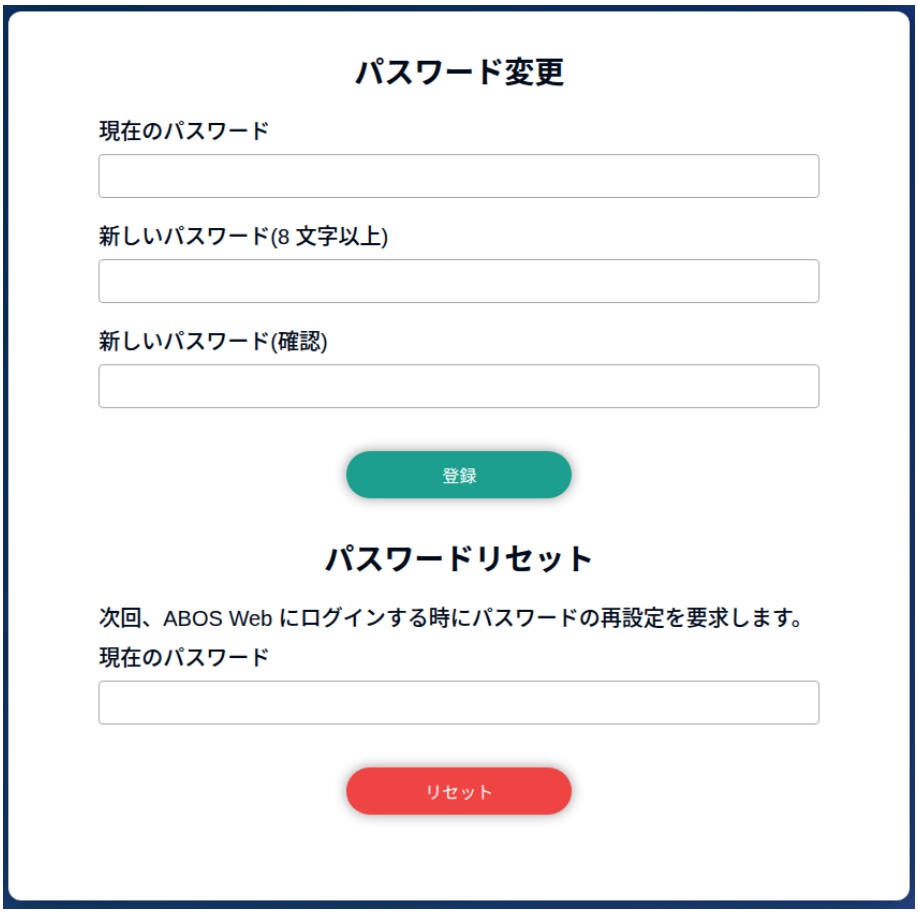


図 3.147 ログイン画面

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 に搭載されていなければ、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/MNM 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、割当たっている IP アドレスなどを "現在の WWAN 接続情報" に表示します。「図 3.148. WWAN 設定画面」に、WWAN 設定を行った状態を示します。



図 3.148 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、割当たっている 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。設定項目のうちに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 3.149. 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 3.149 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 3.150. WLAN アクセスポイント設定画面」に、WLAN アクセスポイント設定を行った状態を示します。



図 3.150 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 3.151 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://networkmanager.dev/docs/api/latest/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 3.152. 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。



図 3.152 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 3.153. 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。



図 3.153 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 3.154. 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。



図 3.154 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 3.155. LTE からの受信パケットに対するポートフォワーディング設定」 に、受信インターフェース ppp0 について、ポート 8080 番宛の tcp パケットをポートフォワーディングする設定を行った状態を示します。



図 3.155 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 3.156. VPN 設定」に、VPN 接続設定を行った状態を示します。



図 3.156 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. USB デバイスの接続を許可する

IoT 機器において、悪意のある第三者が容易に悪用できる USB コネクタなどのインターフェースを外部に露出したままでの運用は、セキュリティ的な脆弱性に繋がります。しかし、保守運用のために USB インターフェースを露出しておかなければならない場合や、機器として USB で接続する周辺デバイスがある場合など、全ての USB インターフェースを物理的に閉じておくことが難しいことも考えられます。

Armadillo Base OS は、 USB デバイスの許可リストを作成・管理し、許可リストにないデバイスが接続されても認識しないように設定できる USB 接続制御機能を持っています。この機能を用いることで、悪意のある第三者が不正な USB デバイスを接続しても、システムに影響を与えません。



ABOS Web による USB 接続制御機能は、ABOS バージョン 3.21.3-at.12 以降で対応しています。

この機能について、CUI で実行したい場合は「6.27.3. 不正な USB デバイスの接続を拒否する」をご参照ください。



工場出荷状態における USB 接続制御機能

製品のセキュリティ向上のため、 ABOS バージョン 3.22.1-at.5 以降では、USB 接続制御機能がデフォルトで有効化されています。デフォルトで接続が許可されている USB デバイスは次の通りです。

表 3.20 デフォルトで許可されている USB デバイス

デバイス名/デバイスクラス名	利用目的
MassStorage クラス	SWUpdate で使用するため
Hub クラス	Armadillo 内部で使用しているため

開発中に追加で上記以外の USB デバイスを接続する場合は、以下の手順にしたがってデバイスの接続を許可してください。

ABOS Web では、Armadillo の USB 接続制御の設定を行うことができます。

ここでは、USB デバイスの許可リストを作成・管理し、許可リストにないデバイスが接続されても認識しないように設定できます。この機能を用いることで、悪意のある第三者が不正な USB デバイスを接続しても、システムに影響を与えません。

「図 3.157. USB 接続制御の設定画面」に設定画面を示します。



図 3.157 USB 接続制御の設定画面

一番上の「機能の状態」に現在の USB 接続制御機能が有効であるか（enabled）無効であるか（disabled）表示します。

有効である場合、「接続済みの USB デバイス」、「許可済みの USB デバイス」および「許可済みの USB デバイスクラス」の欄が表示されます。

「図 3.158. 接続済みの USB デバイス」に示すように、「接続済みの USB デバイス」では、現在接続されている USB デバイスが一覧として表示されます。



図 3.158 接続済みの USB デバイス

"詳細表示" ボタンを押すと、「図 3.159. 接続済み USB デバイスの詳細情報」に示す画面が表示されて、その USB デバイスに関するより詳細な情報が表示されます。



図 3.159 接続済み USB デバイスの詳細情報

「可否」が allow である場合、その USB デバイスは接続が許可されています。

block の場合、「図 3.160. USB デバイスを許可する」に表示されている "許可" ボタンを押すことで allow に変更できます。



図 3.160 USB デバイスを許可する

"許可" ボタンを押すと、「図 3.161. 許可済み USB デバイス」に示すように「許可済みの USB デバイス」の欄に許可ルールが追加されます。

「許可済みの USB デバイス」に表示されている USB デバイスは永続的に接続が許可されます。



図 3.161 許可済み USB デバイス

「図 3.162. 接続済みの個体と同じメーカー・製品の全ての個体を許可する」に示すように、「同じメーカー・製品の全ての個体を許可」のチェックボックスを選択した状態で "許可" ボタンを押すと、選択した USB デバイスと同じメーカー・製品であれば、どの個体であっても接続が許可されます。



図 3.162 接続済みの個体と同じメーカー・製品の全ての個体を許可する

その場合、「図 3.163. 全ての個体を許可する場合の表示」に示すように、「許可済みの USB デバイス」の「全ての個体を許可」の欄に「✓」が表示されます。



図 3.163 全ての個体を許可する場合の表示

「図 3.163. 全ての個体を許可する場合の表示」に示すように、USB デバイスの許可ルールを選択した状態で "削除" ボタンを押すと、「図 3.164. 許可ルールを削除する」に示す画面に遷移します。



図 3.164 許可ルールを削除する

確認画面が表示されて、問題なければ改めて "削除" のボタンを押すことでその許可ルールを削除できます。

その USB デバイスに関する許可ルールが削除されると永続的にその USB デバイスの接続は拒否されます。「接続済みの USB デバイス」では、「可否」が block に変更されます。

「図 3.165. USB デバイスクラス」に表示されている「許可済みの USB デバイスクラス」では、許可する USB デバイスをデバイスクラス単位で設定することができます。



図 3.165 USB デバイスクラス

一覧に表示されている USB デバイスクラスは現在許可されているデバイスクラスです。

デバイスクラスを選択した後、"削除" ボタンを押すことでそのデバイスクラスの許可ルールを削除できます。削除の手順は「図 3.164. 許可ルールを削除する」と同様です。

"追加" ボタンを押すと、「図 3.166. USB デバイスクラスの追加」に示す画面が表示されます。追加可能な USB デバイスクラスを選択し、"追加" ボタンを押してください。



図 3.166 USB デバイスクラスの追加

3.10. 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
- 製品型番
- 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。製品型番のカスタマイズに関してはバージョン 1.10.0 以降で対応しています。



図 3.167 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。

- **製品型番**

  設定するとヘッダタイトルの下に表示されます。お客様の最終製品の型番を 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 3.21 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。



図 3.168 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。

3.11. ABOS Web から Armadillo の電源を操作する

開発中及び運用中に Armadillo の再起動または電源を切るには、ターミナルでコマンドを実行することでも可能ですが、ABOS Web からも実行可能です。ABOS Web から Armadillo の電源を操作するには「図 3.169. 「電源制御」の位置」に示すサイドメニューの項目を選択してください。

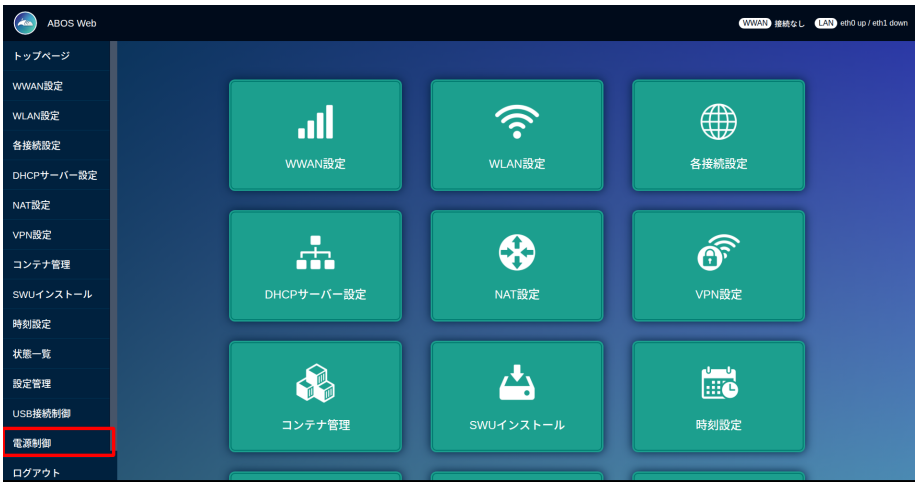


図 3.169 「電源制御」の位置

この項目を選択すると「図 3.170. 「電源制御」の画面」に示す画面が表示されます。



図 3.170 「電源制御」の画面

操作可能な機能は以下です。

表 3.22 制御できる機能

機能名	Armadillo 内で実行されるコマンド	説明
再起動	reboot	Armadillo を再起動します。
電源オフ	poweroff	Armadillo の電源を切ります。

これらの機能を ABOS Web で選択すると、確認画面が表示されます。よければ「OK」を選択してください。

電源オフを実行した場合、Armadillo の LED が消えるまで電源を抜かないでください。

3.12. 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なります。

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 3.171. 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 3.171 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。

3.13. 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 をお試しいただくことをおすすめします。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。

3.14. 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

3.14.1. ABOSDE の対応言語

「表 3.23. 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 3.23 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

3.14.2. 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 3.172. 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。

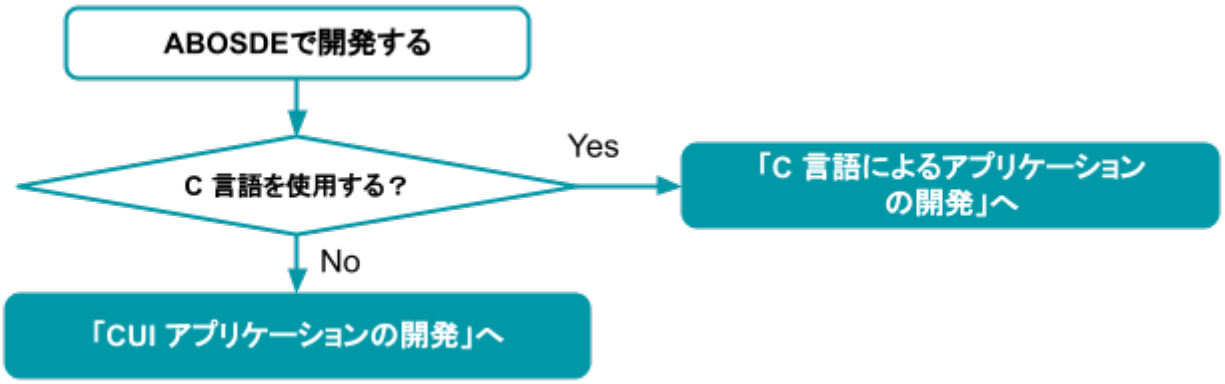


図 3.172 参照する開発手順の章を選択する流れ

CUI アプリケーション

- 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
  - 「3.15. CUI アプリケーションの開発」を参照

C 言語アプリケーション

- 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
  - 「3.16. C 言語によるアプリケーションの開発」を参照

3.15. 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。

3.15.1. 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。

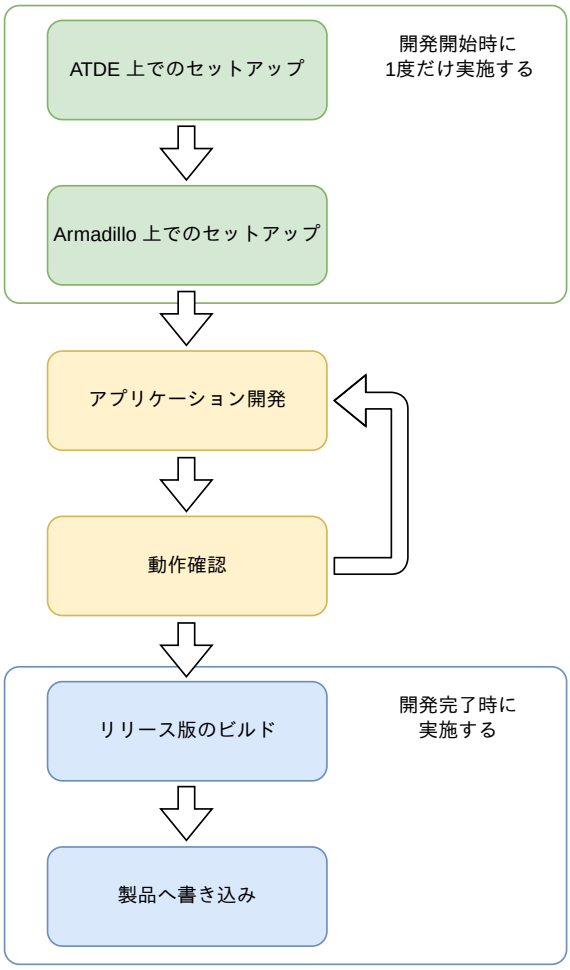


図 3.173 CUI アプリケーション開発の流れ

3.15.2. 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。

3.15.2.1. 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`

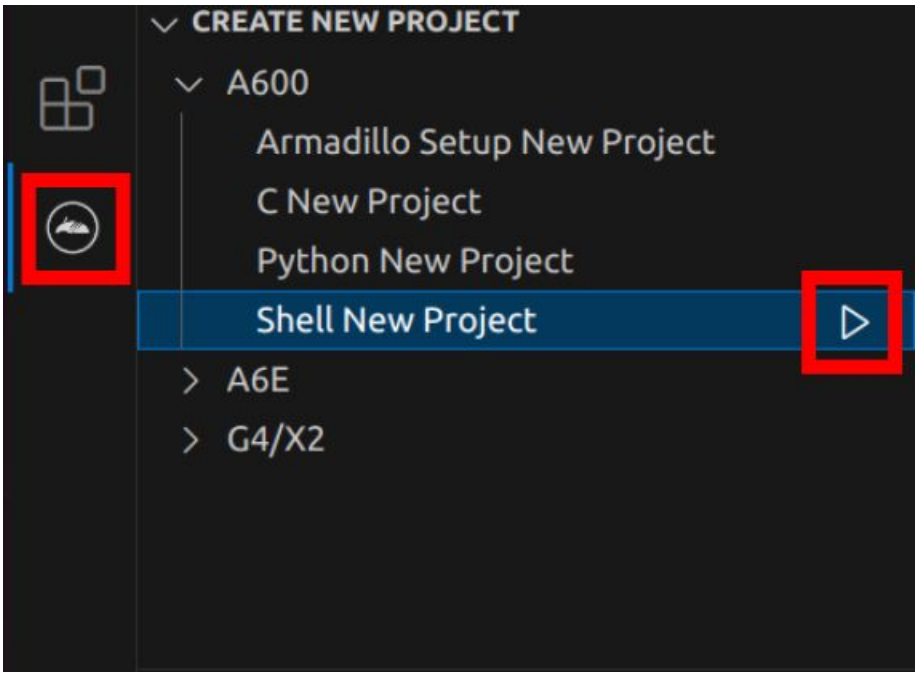


図 3.174 プロジェクトを作成する

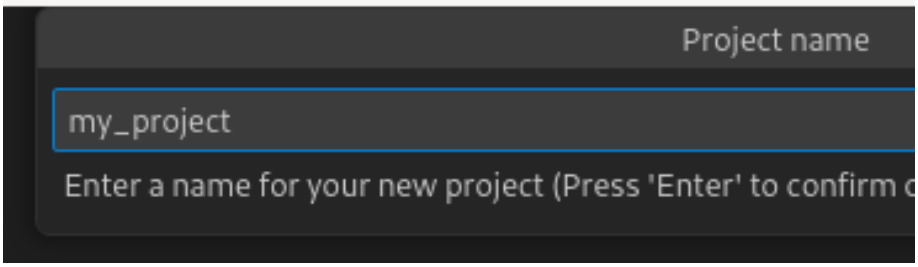


図 3.175 プロジェクト名を入力する

3.15.3. アプリケーション開発

3.15.3.1. 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 3.176 VS Code で my_project を起動する

3.15.3.2. 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。「3.15.6.2. 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。
  - **resources**: Armadillo の ルートディレクトリ（ / ）配下に同様のディレクトリ構造でコピーされます。
  - **resources_python**: resources と同様ですが、Python プロジェクト固有のファイルが配置されています。
    - **bin/python_launch**: Python プロジェクトの場合に、コンテナ起動時に実行するスクリプトです。

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、LED3 を点滅させます。

3.15.3.3. 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 3.177 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。

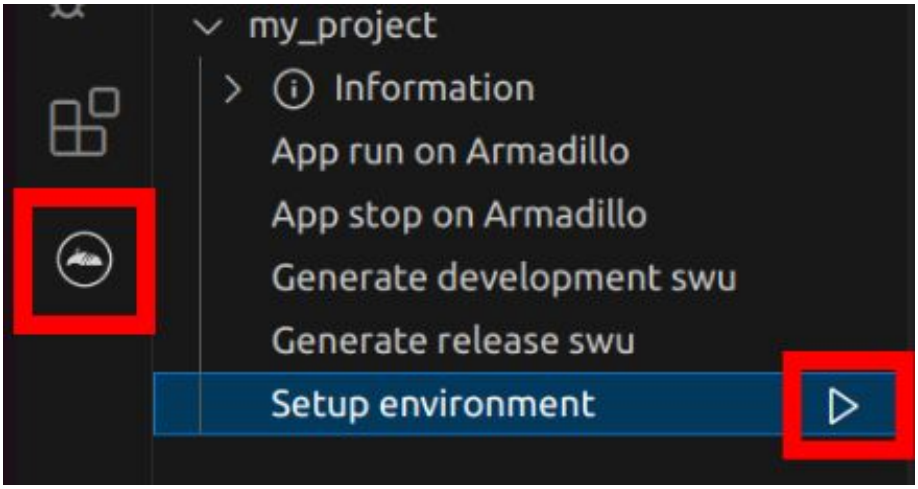


図 3.178 VS Code で初期設定を行う

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。

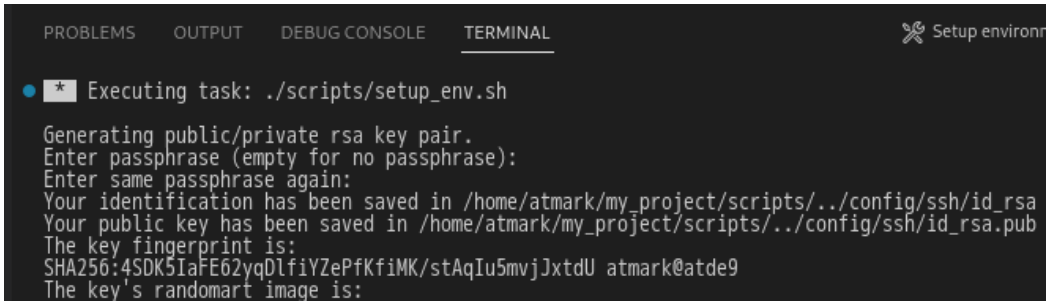


図 3.179 VS Code のターミナル

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 3.180 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。

3.15.3.4. ABOSDE におけるコンテナ OS の選択方法



コンテナ OS を Debian から Alpine に変更する ABOSDE の機能は ABOSDE のバージョン 1.9.3 以降で、かつ 2025 年 6 月 25 日以降に「3.15.2.1. 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。

ABOSDE では、コンテナ OS として Debian と Alpine を提供しています。デフォルトでは、Alpine よりも開発が容易である Debian が使用されます。

Alpine を使用する場合は、VS Code の左ペインの [OPENED PROJECT] の [Generate development swu] または [Generate release swu] を開き、[Use alpine container] を選択してください。

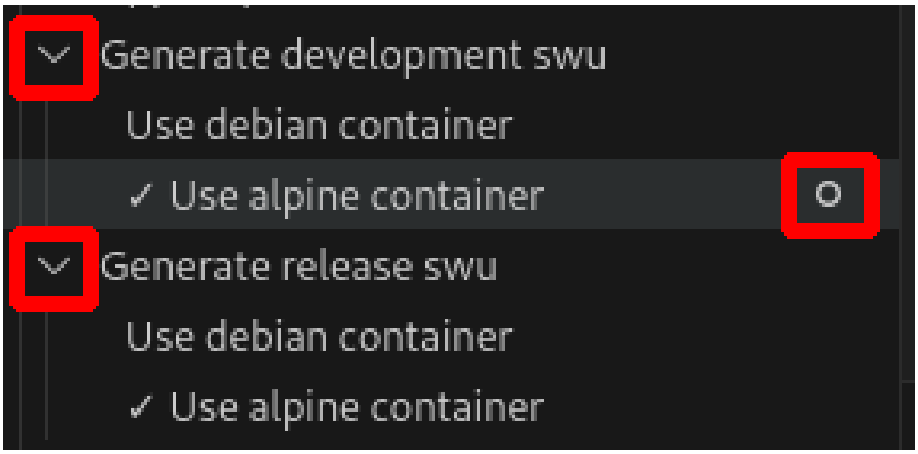


図 3.181 ABOSDE でコンテナ OS を選択する

SWU イメージを生成する時に以下のファイルが使用されます。使用するコンテナ OS にあわせて、対応するファイルを編集してください。

[Use debian container] を選択した場合：

- container/Dockerfile
- container/packages.txt

[Use alpine container] を選択した場合：

- container/Dockerfile_alpine
- container/packages_alpine.txt（シェルスクリプトプロジェクトの場合は packages.txt）

以下の説明では、Dockerfile および packages.txt を使用する場合を前提としていますが、Dockerfile_alpine および packages_alpine.txt を使用する場合も同様の手順で進めることができます。

3.15.3.5. Debian コンテナのディストリビューション

Debian コンテナを使用する場合、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

3.15.3.6. 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。

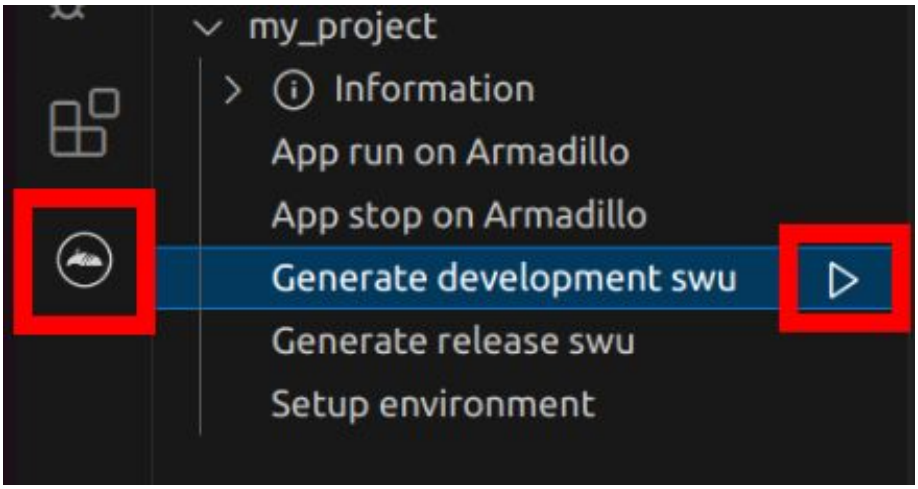


図 3.182 VS Code でコンテナイメージの作成を行う

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 3.183 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。

3.15.3.7. 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。

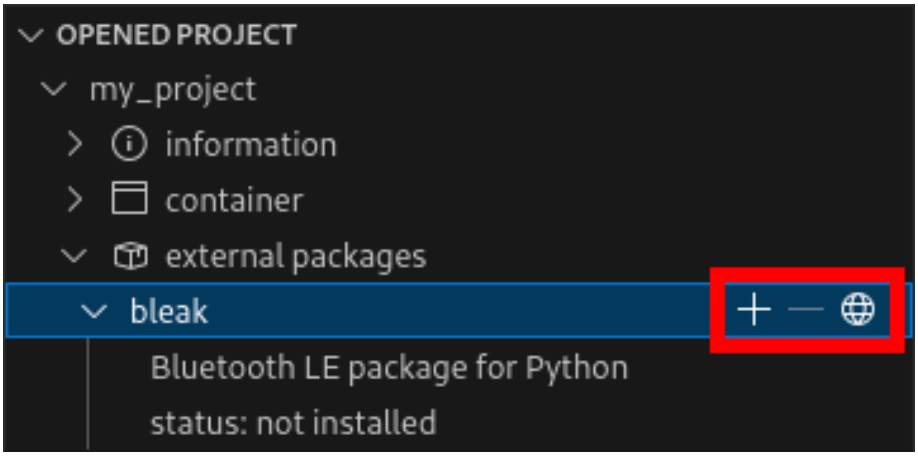


図 3.184 BLE パッケージをインストールする

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「3.15.2.1. 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。

3.15.4. 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

3.15.5. コンテナ内のファイル一覧表示

「図 3.185. 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「3.15.8. 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。

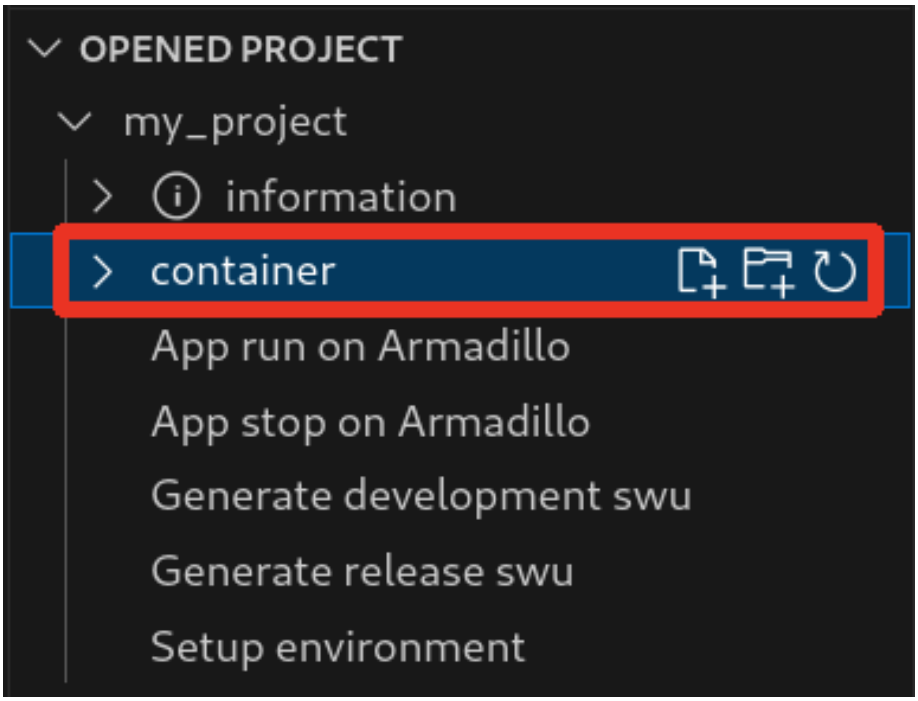


図 3.185 コンテナ内のファイル一覧を表示するタブ

クリック後の表示例を 「図 3.186. コンテナ内のファイル一覧の例」 に示します。

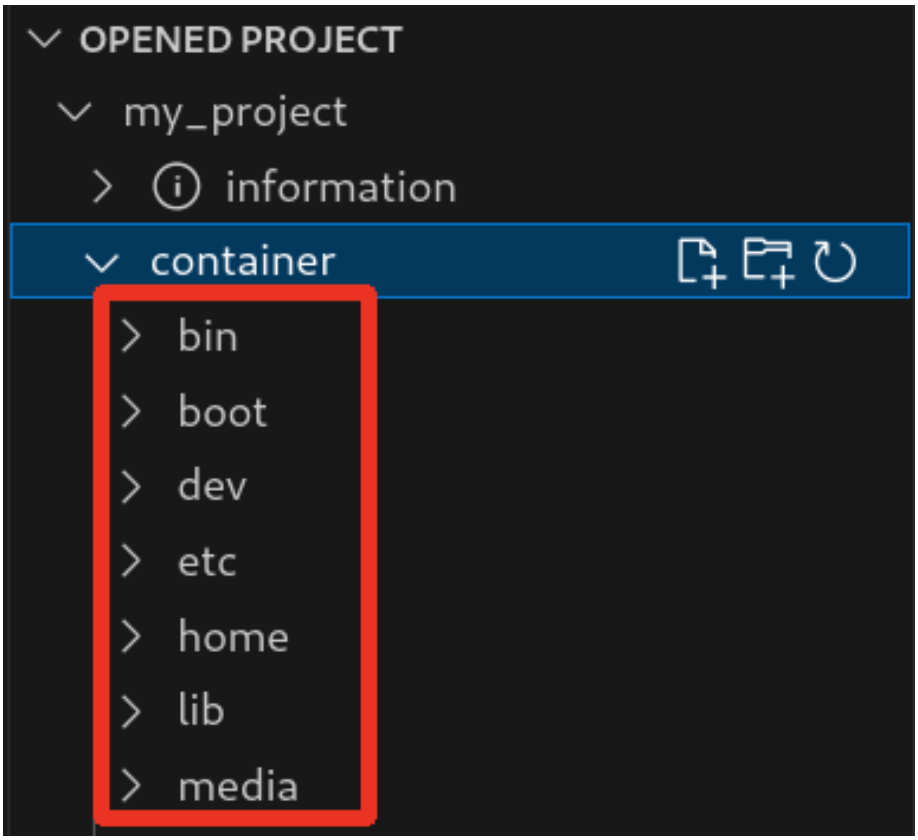


図 3.186 コンテナ内のファイル一覧の例

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。

3.15.5.1. resources ディレクトリについて

「図 3.187. 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。

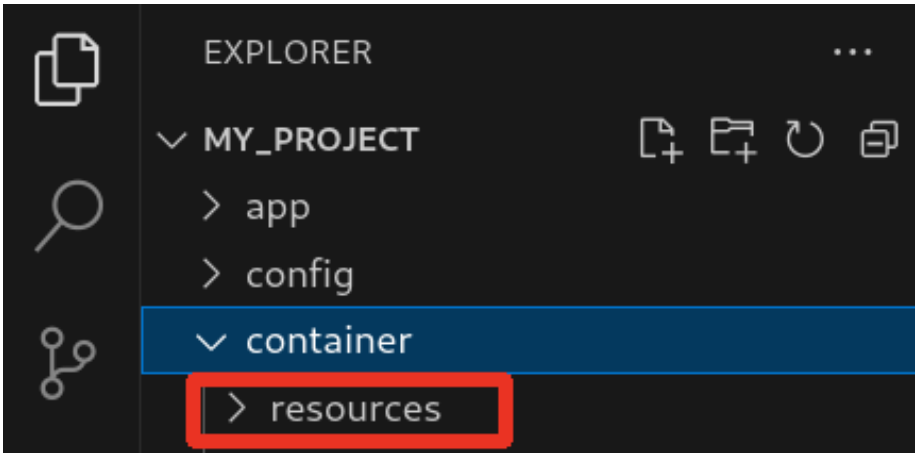


図 3.187 resources ディレクトリ

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。

3.15.5.2. コンテナ内のファイル一覧の再表示

「図 3.185. 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。

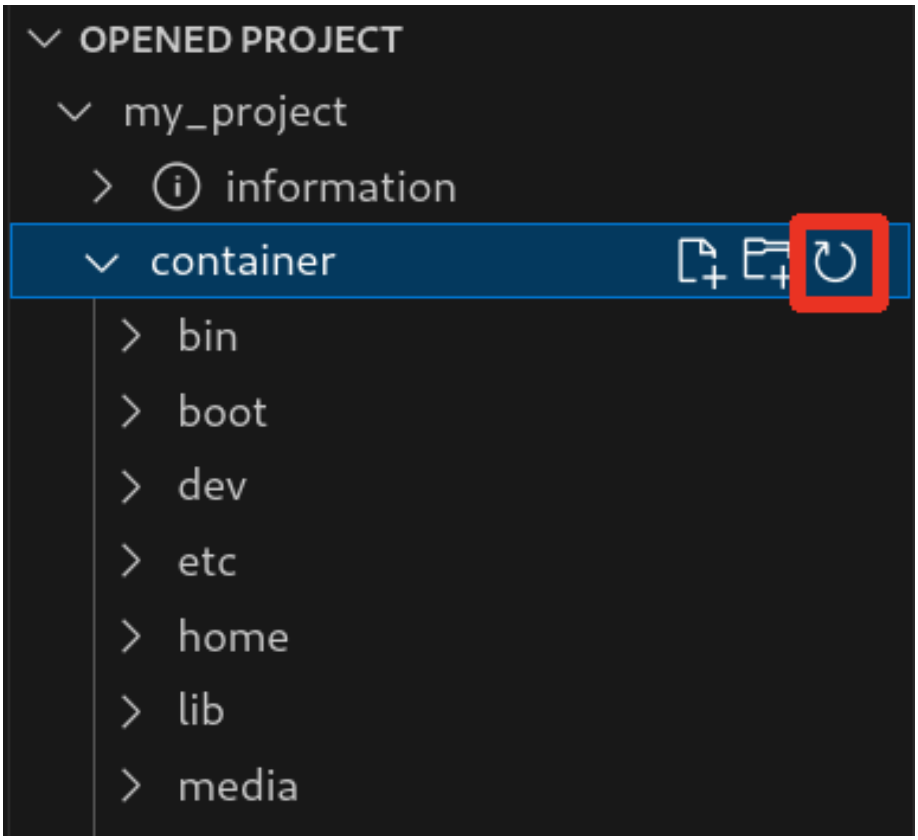


図 3.188 コンテナ内のファイル一覧を再表示するボタン

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

「図 3.189. 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。

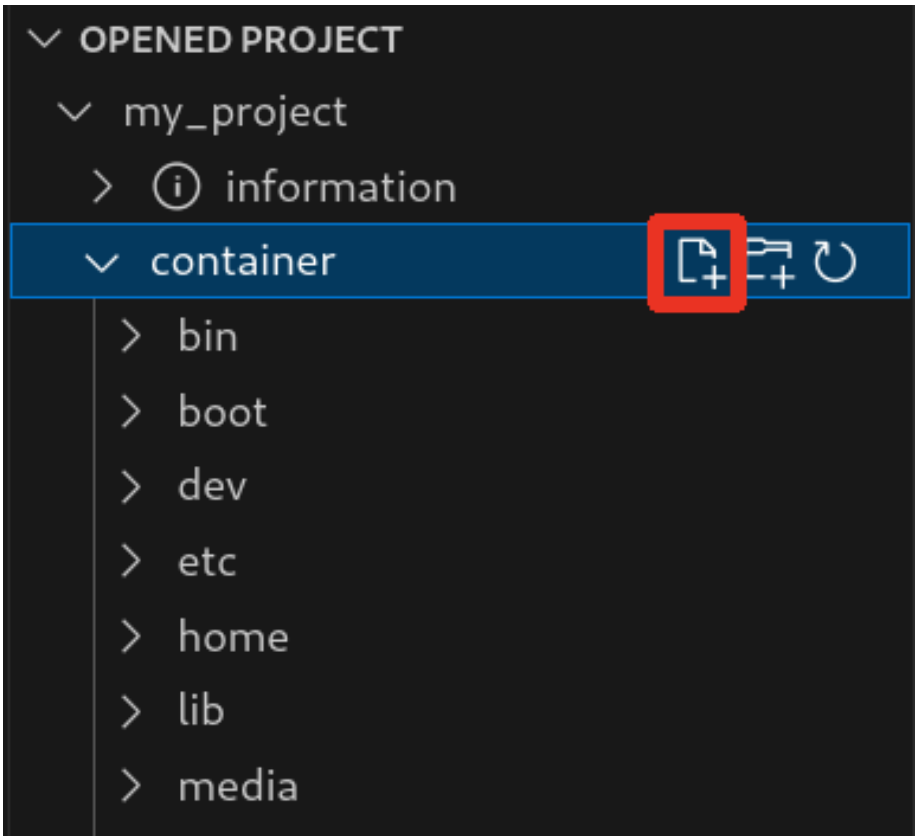


図 3.189 container/resources 下にファイルを追加するボタン

「図 3.190. 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。

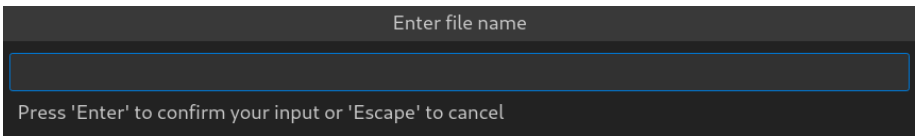


図 3.190 ファイル名を入力

例として、「add_file」というファイル名を入力したとします。

「図 3.191. 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。

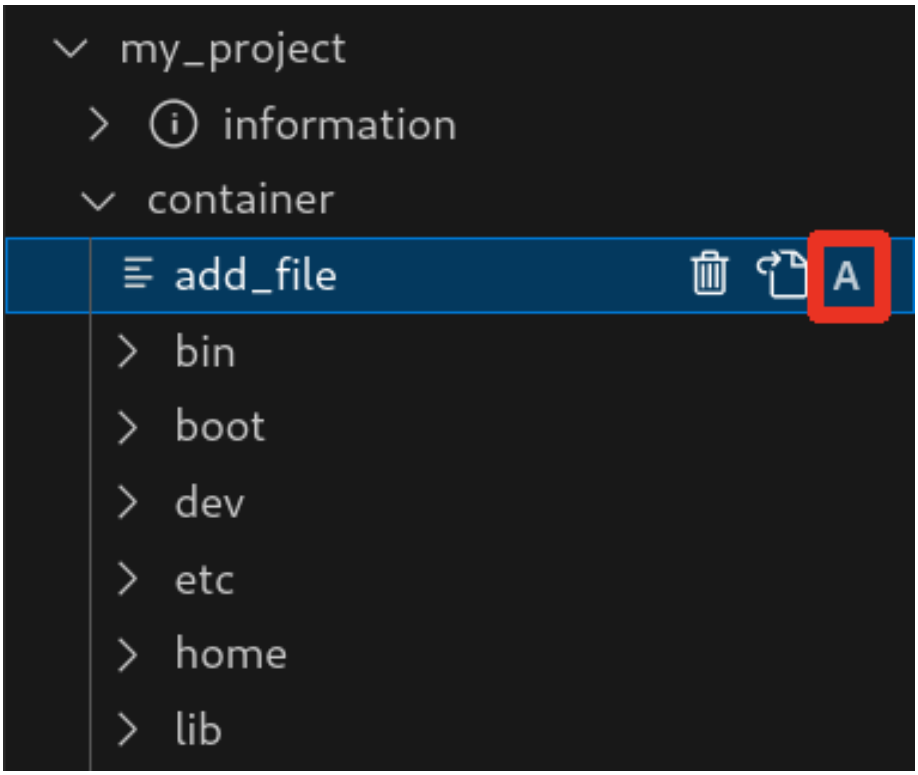


図 3.191 追加されたファイルの表示

また、「図 3.192. 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。

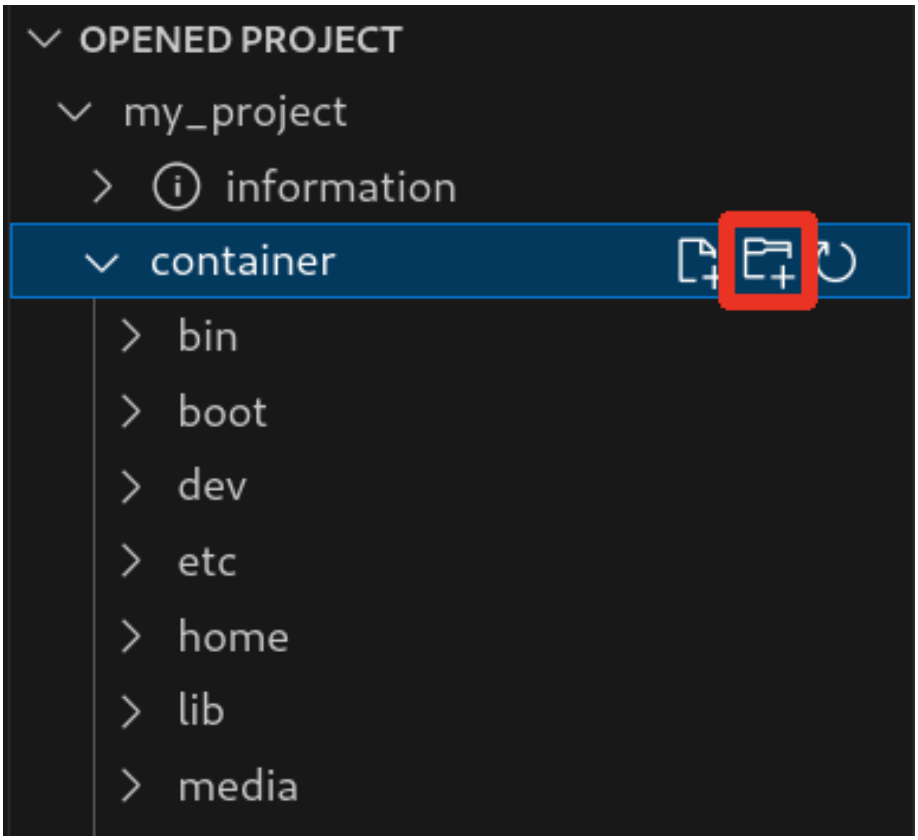


図 3.192 container/resources 下にフォルダーを追加するボタン

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

「図 3.193. 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。

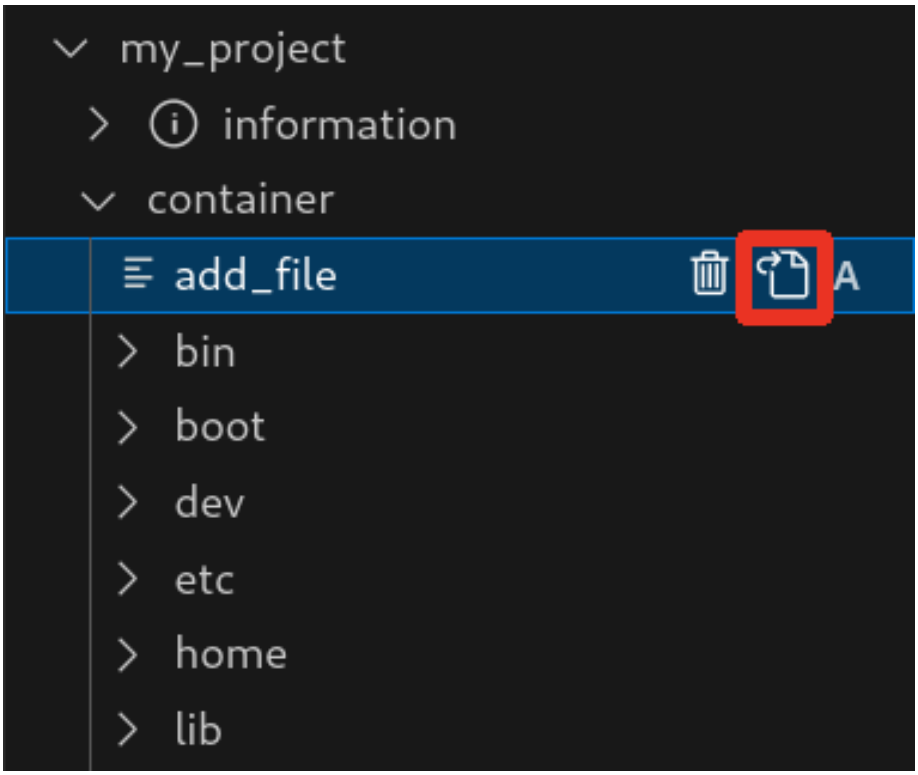


図 3.193 container/resources 下にあるファイルを開くボタン

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

「図 3.193. 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。

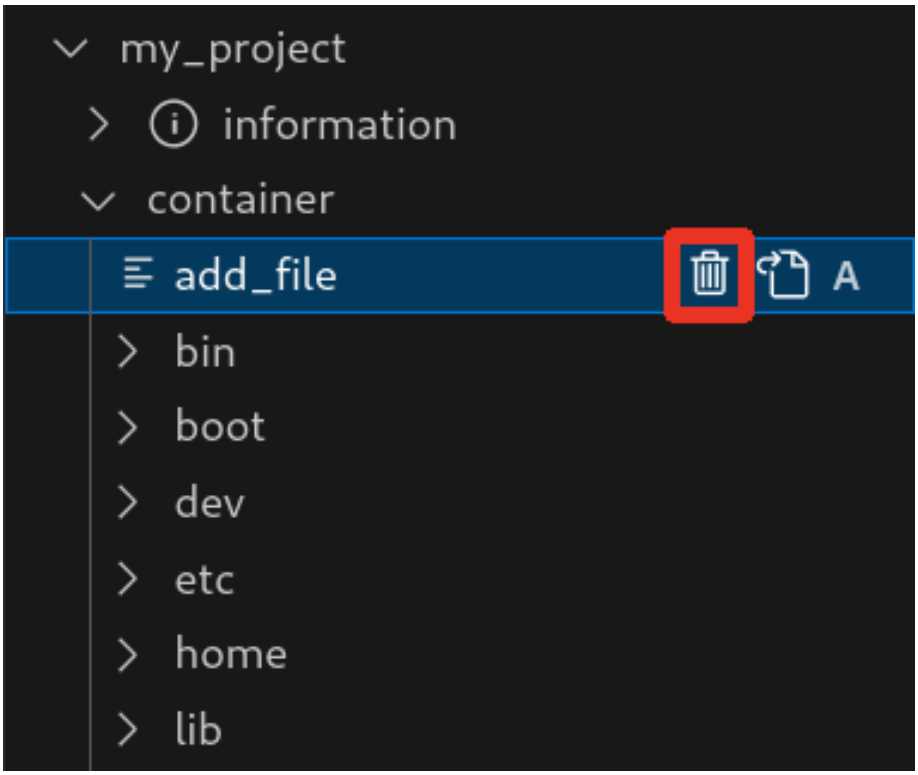


図 3.194 container/resources 下にあるファイルを削除するボタン

ディレクトリも同様に「図 3.193. 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。

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

「図 3.195. 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。

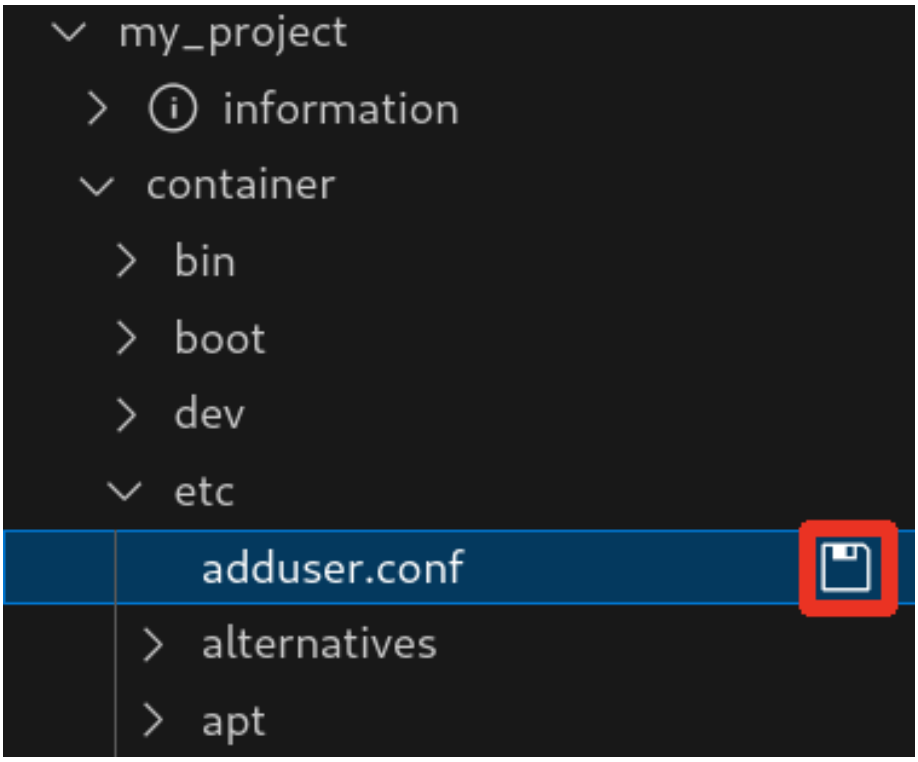


図 3.195 コンテナ内のファイルを container/resources 下に保存するボタン

ファイルが container/resources 下に保存されると、「図 3.196. 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。

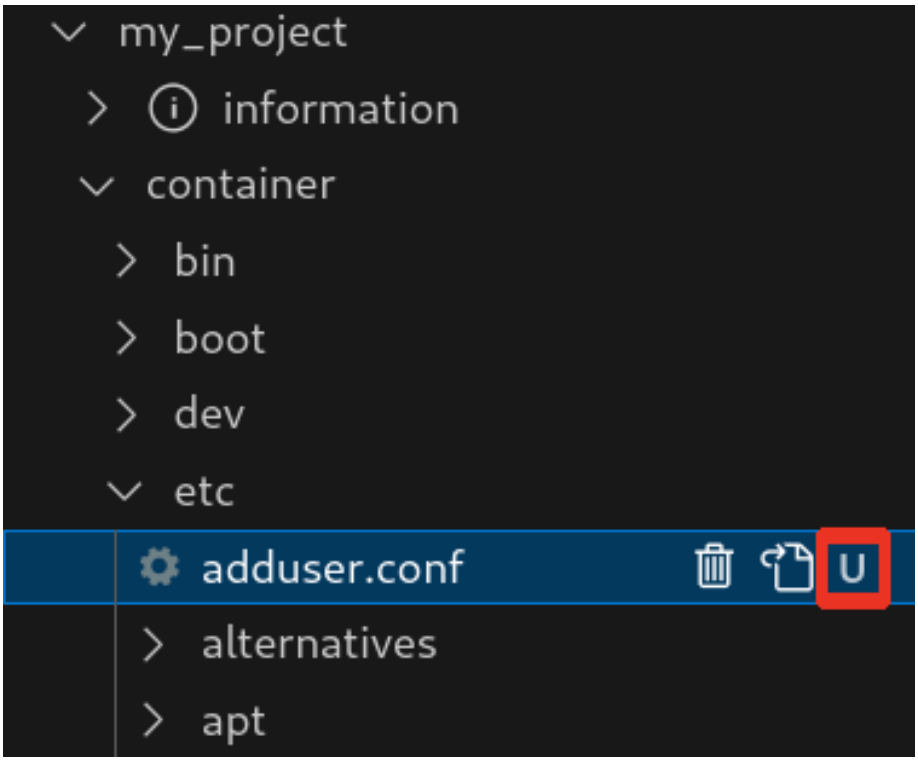


図 3.196 編集前のファイルを示すマーク

container/resources 下にあるファイルを編集して再表示すると、「図 3.197. 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。

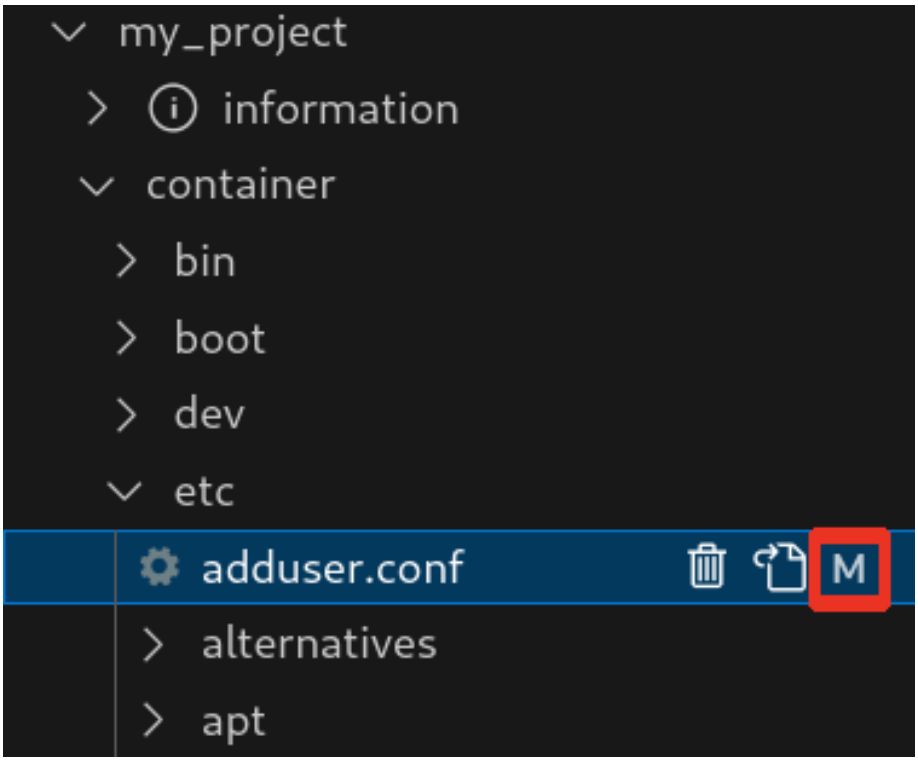


図 3.197 編集後のファイルを示すマーク

3.15.5.7. 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 3.198. 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。

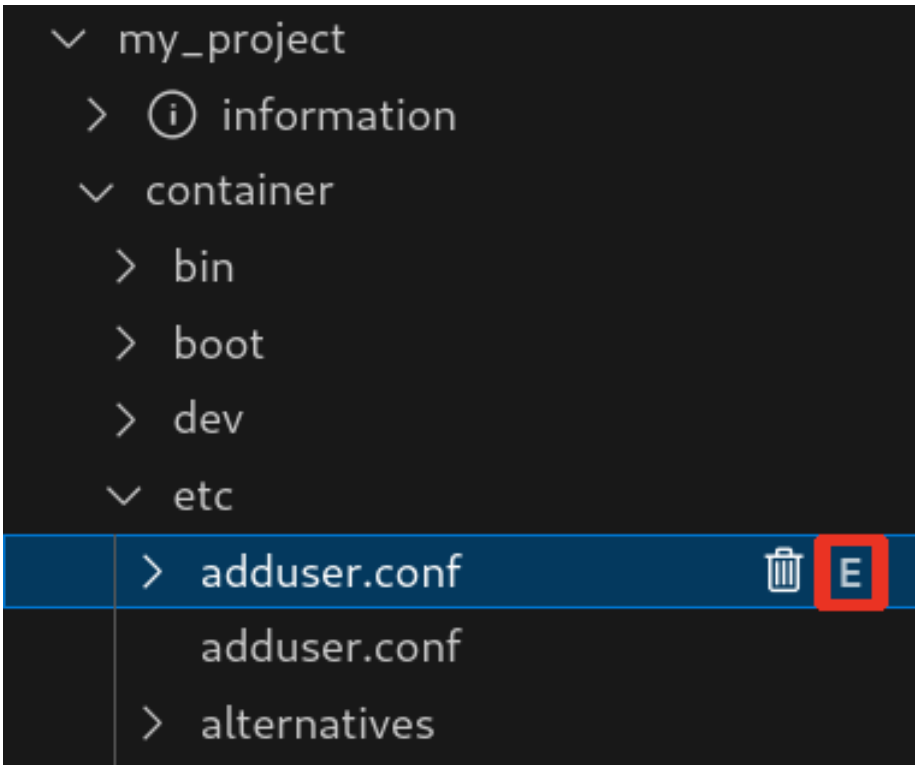


図 3.198 コンテナ内にコピーされないことを示すマーク

3.15.6. Armadillo 上でのセットアップ

3.15.6.1. 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

「3.15.3.6. 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。

3.15.6.2. 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 3.199. 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。

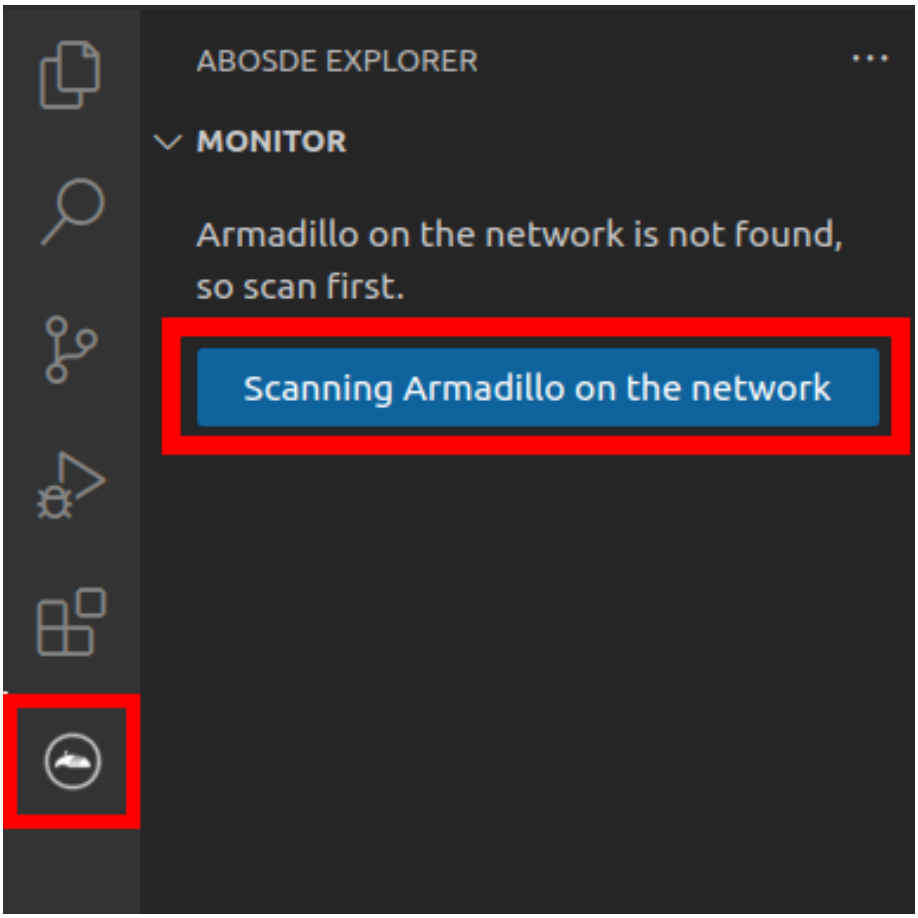


図 3.199 ABOSDE で ローカルネットワーク上の Armadillo をスキャンする

「図 3.200. 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。

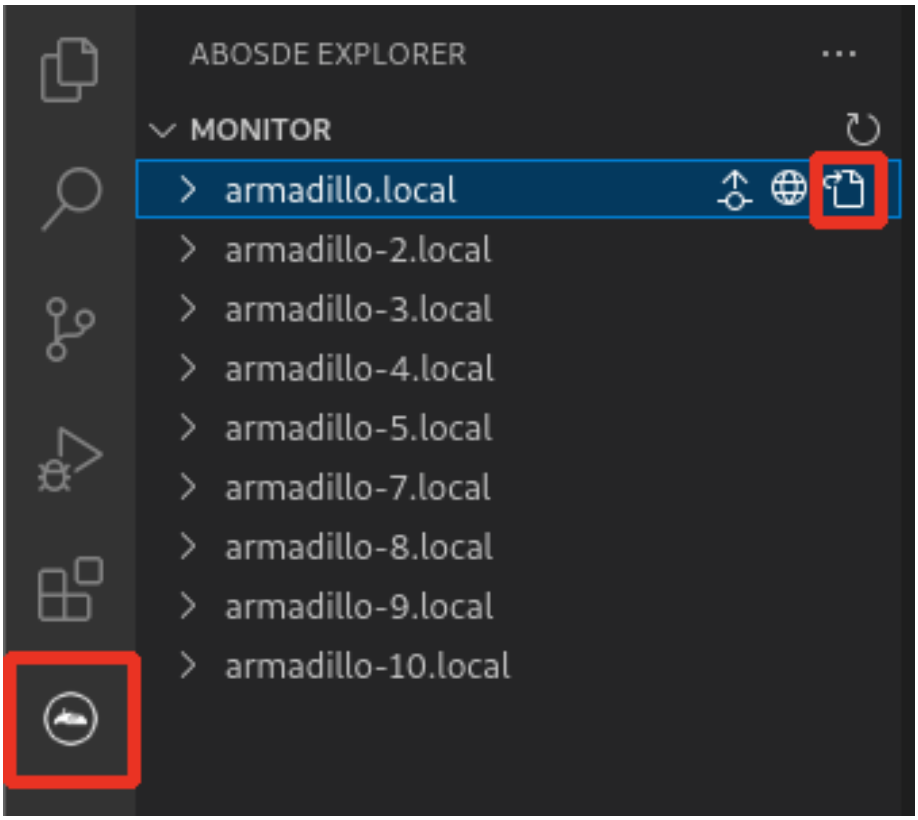


図 3.200 ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する

「図 3.201. 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。

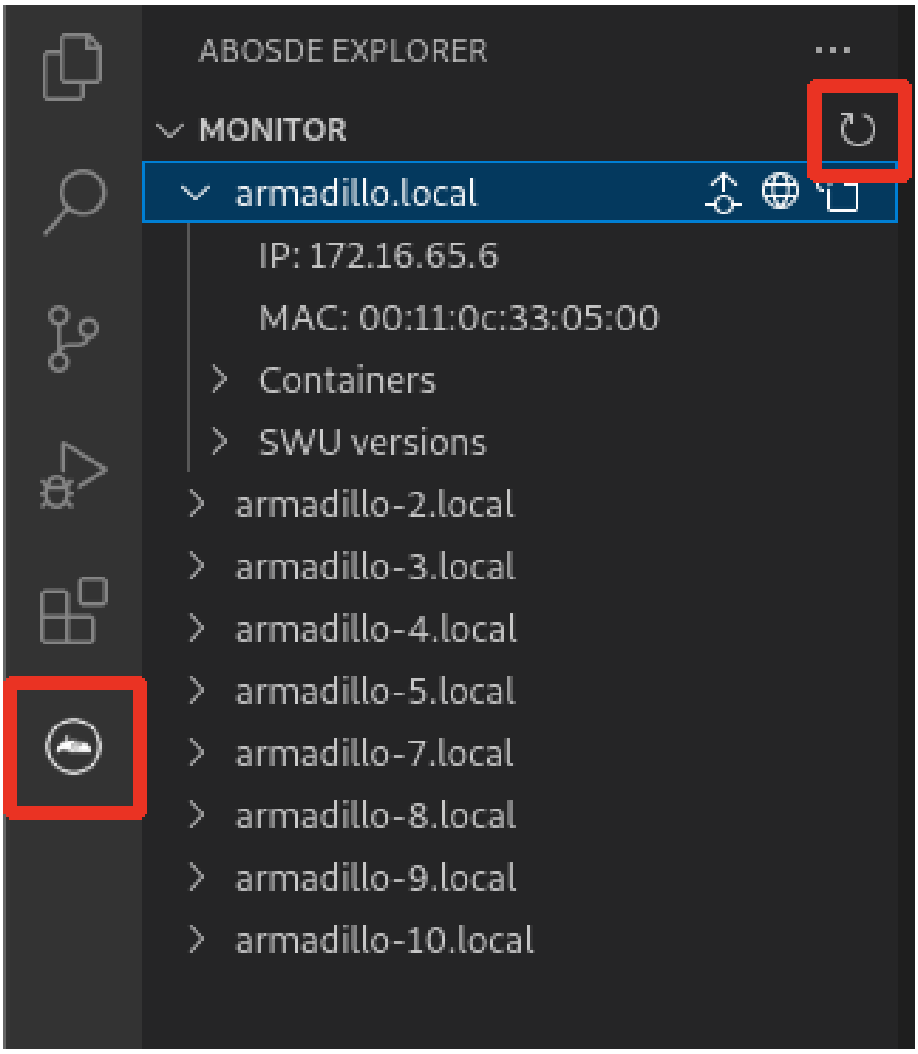


図 3.201 ABOSDE に表示されている Armadillo を更新する

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 3.202 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。

3.15.6.3. 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。

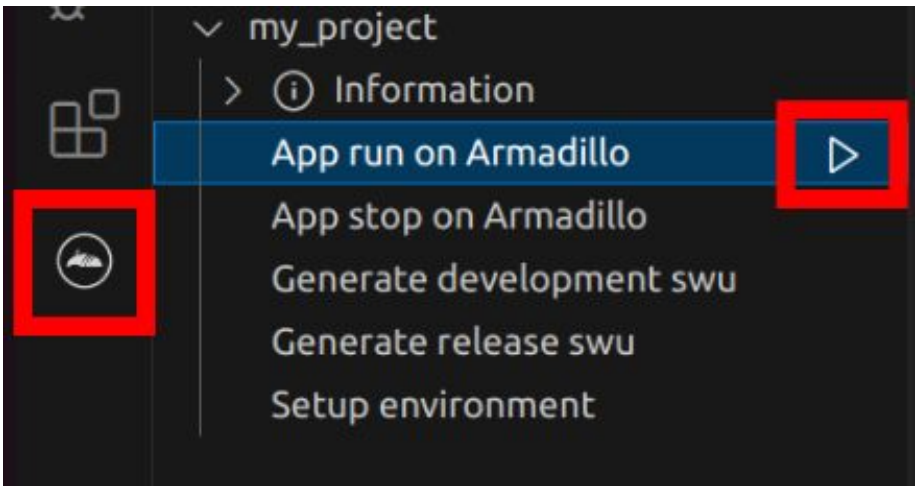


図 3.203 Armadillo 上でアプリケーションを実行する

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 3.204 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。

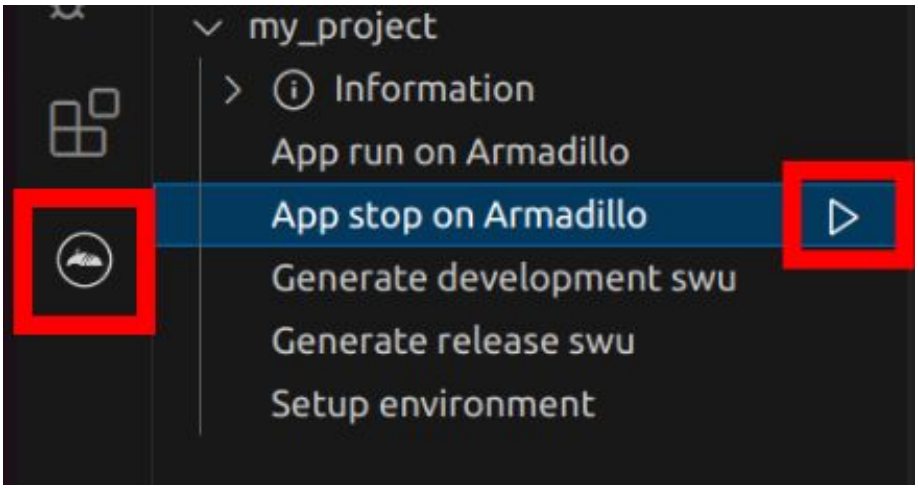


図 3.205 アプリケーションを終了する

3.15.7. 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 「3.17. SBOM 生成に関わる設定を行う」 を参照してください。

3.15.8. 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。

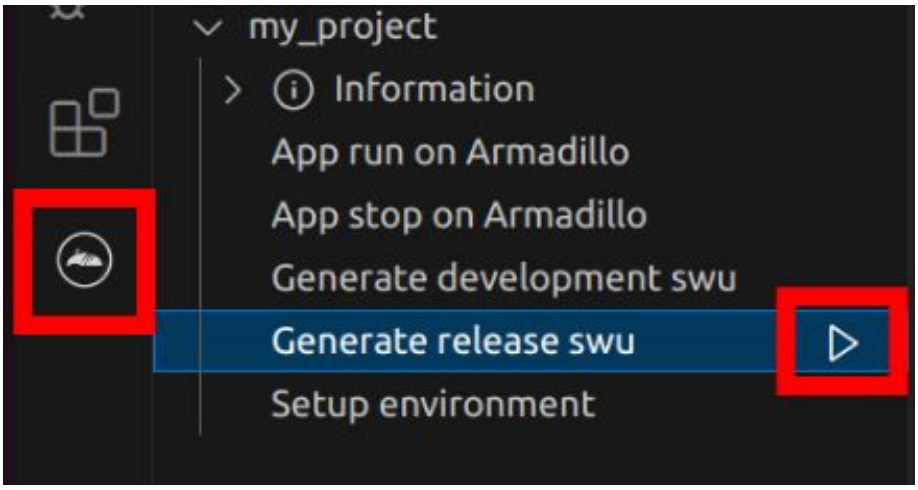


図 3.206 リリース版をビルドする



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。

3.15.9. 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。

3.15.10. 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。

3.16. 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、「3.15. CUI アプリケーションの開発」を参照してください。

3.16.1. 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。

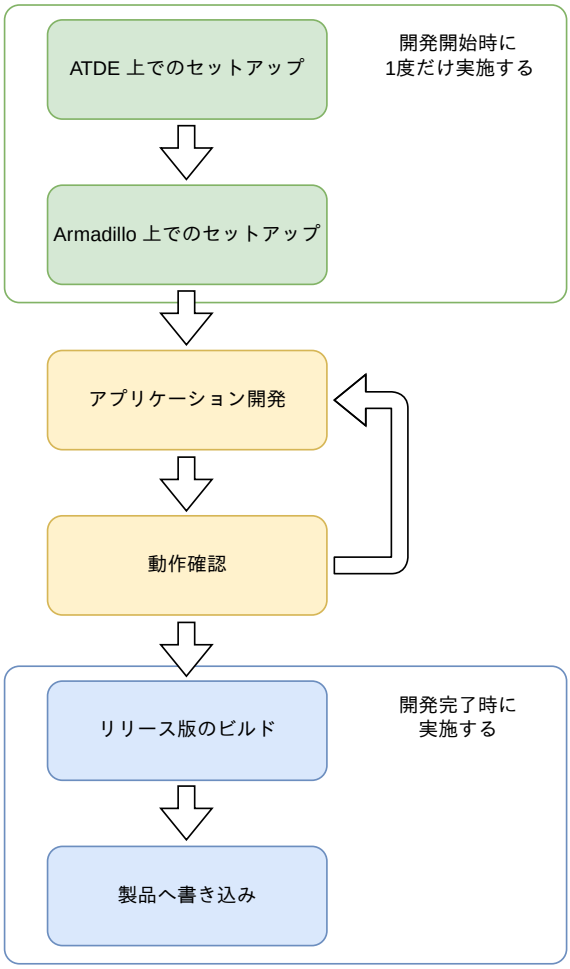


図 3.207 C 言語によるアプリケーション開発の流れ

3.16.2. 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。

3.16.2.1. 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`

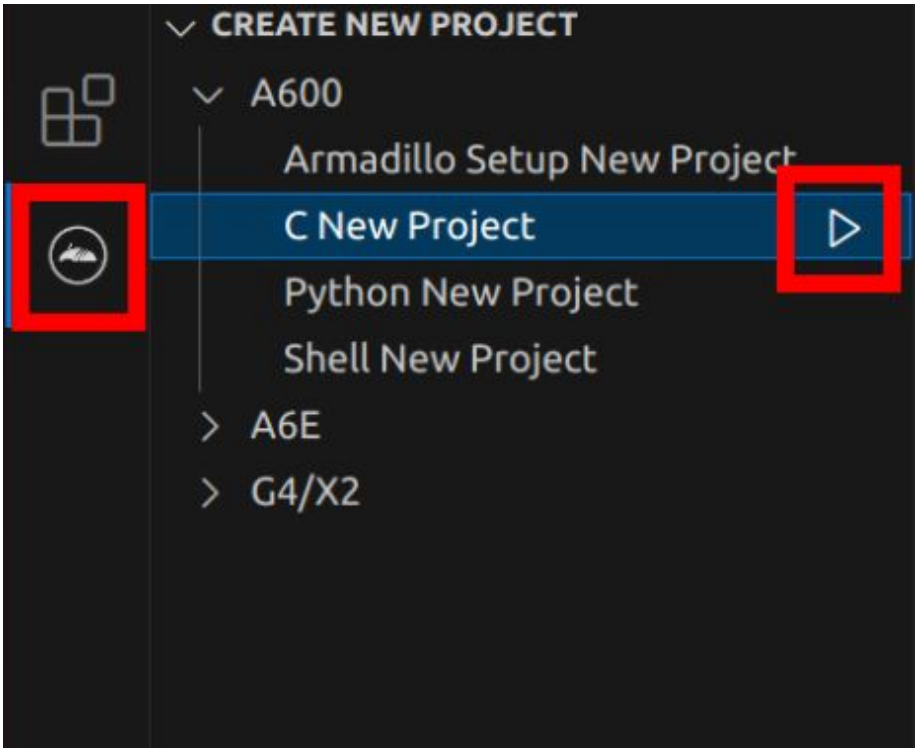


図 3.208 プロジェクトを作成する

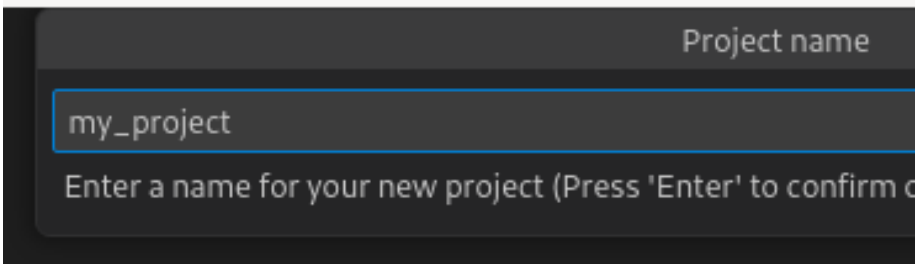


図 3.209 プロジェクト名を入力する

3.16.3. アプリケーション開発

3.16.3.1. 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 3.210 VS Code で my_project を起動する

3.16.3.2. 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。「3.16.6.2. 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。
  - **resources**: Armadillo の ルートディレクトリ（ / ）配下に同様のディレクトリ構造でコピーされます。
  - **resources_c**: resources と同様ですが、C 言語プロジェクト固有のファイルが配置されています。
    - **build.sh**: 実行ファイルをビルドするためのスクリプトです。必要に応じて内容を変更してください。
    - **bin/c_launch**: コンテナ起動時に実行するスクリプトです。

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、LED3 を点滅させます。

3.16.3.3. 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 3.211 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。

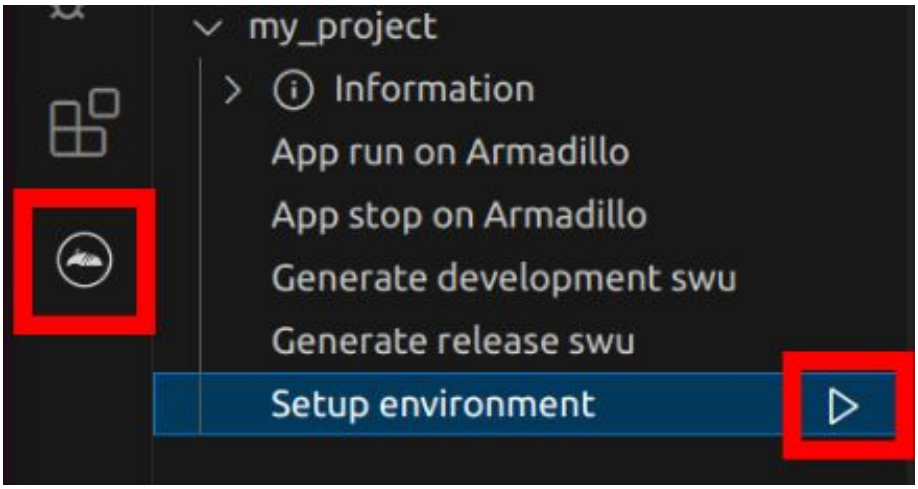


図 3.212 VS Code で初期設定を行う

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。

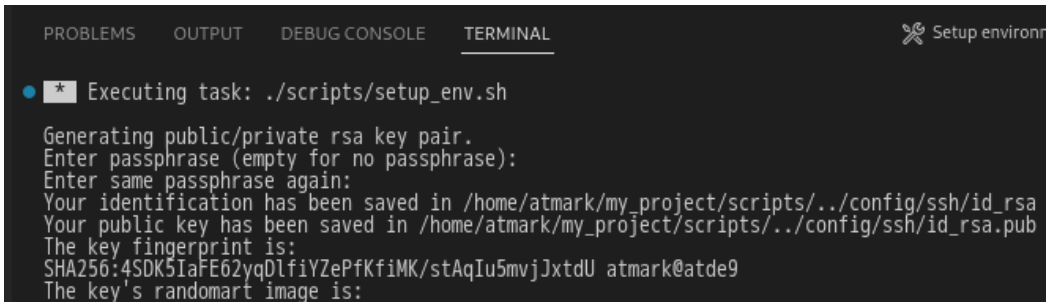


図 3.213 VS Code のターミナル

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 3.214 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。

3.16.3.4. ABOSDE におけるコンテナ OS の選択方法



コンテナ OS を Debian から Alpine に変更する ABOSDE の機能は ABOSDE のバージョン 1.9.3 以降で、かつ 2025 年 6 月 25 日以降に「3.16.2.1. 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。

ABOSDE では、コンテナ OS として Debian と Alpine を提供しています。デフォルトでは、Alpine よりも開発が容易である Debian が使用されます。

Alpine を使用する場合は、VS Code の左ペインの [OPENED PROJECT] の [Generate development swu] または [Generate release swu] を開き、[Use alpine container] を選択してください。

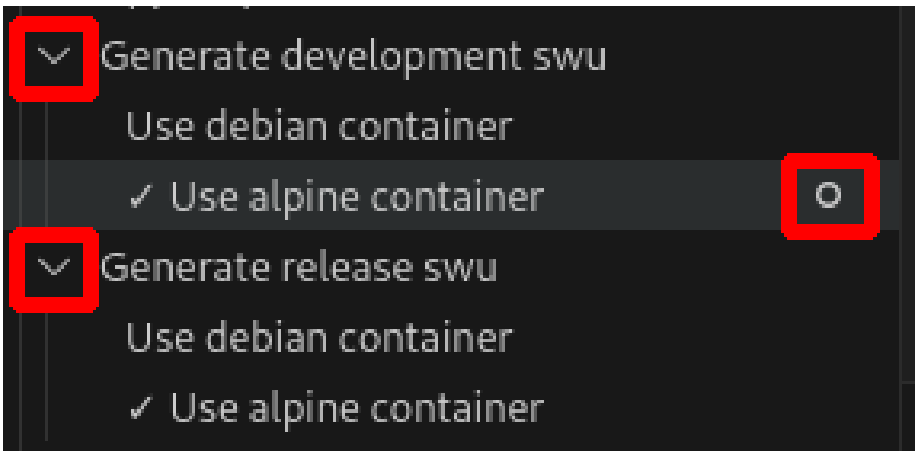


図 3.215 ABOSDE でコンテナ OS を選択する

SWU イメージを生成する時に以下のファイルが使用されます。使用するコンテナ OS にあわせて、対応するファイルを編集してください。

[Use debian container] を選択した場合：

- container/Dockerfile

・ container/packages.txt

[Use alpine container] を選択した場合：

・ container/Dockerfile_alpine

・ container/packages_alpine.txt

以下の説明では、Dockerfile および packages.txt を使用する場合を前提としていますが、Dockerfile_alpine および packages_alpine.txt を使用する場合も同様の手順で進めることができます。

3.16.3.5. Debian コンテナのディストリビューション

Debian コンテナを使用する場合、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

3.16.3.6. 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 3.216. 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 3.216 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。

3.16.3.7. 実行ファイルのビルド

以下のいずれかのボタンを実行すると、container/resources_c 直下にある build.sh をビルド用コンテナ内で実行して、実行ファイルが app/build に生成されます。

・ [Generate development swu]

・ [Generate release swu]

・ [App run on Armadillo]

なお [App run on Armadillo] は [Generate development swu] または [Generate release swu] を先に実行して、Armadillo に SWU イメージをインストールしておく必要があります。

build.sh のデフォルトの中身は以下のようになっています。

container/resources_c/build.sh の中身.

```
#!/bin/sh

make -C /vol_app/src all
```

デフォルトでは、ATDE 上のプロジェクトディレクトリ内の app/src はコンテナ内の /vol_app/src にコピーされます。

Makefile は app/src 直下に配置しているため、C オプションでパスを指定した後、make all を実行しています。

build.sh の内容を変更することで任意のビルドコマンドを使用することが可能です。

3.16.3.8. ABOSDE での開発における制約

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。

3.16.3.9. 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。

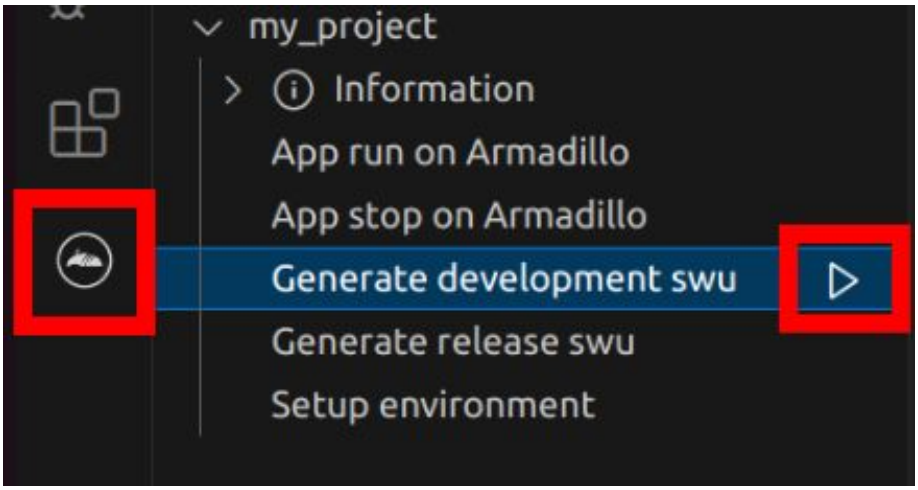


図 3.217 VS Code でコンテナイメージの作成を行う

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 3.218 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。

3.16.4. コンテナ内のファイル一覧表示

「図 3.219. 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「3.16.8. 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。

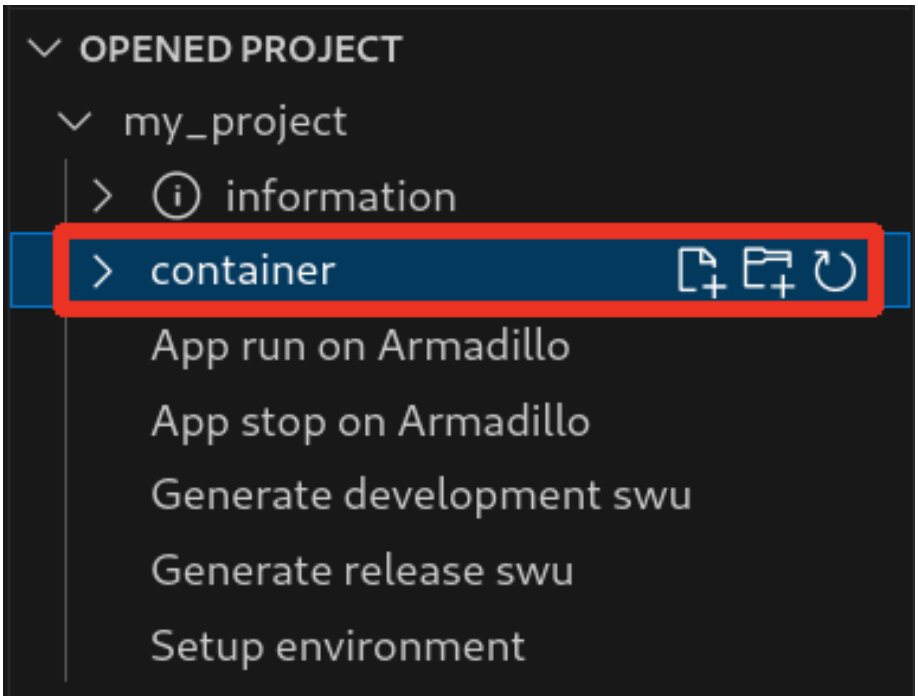


図 3.219 コンテナ内のファイル一覧を表示するタブ

クリック後の表示例を 「図 3.220. コンテナ内のファイル一覧の例」 に示します。

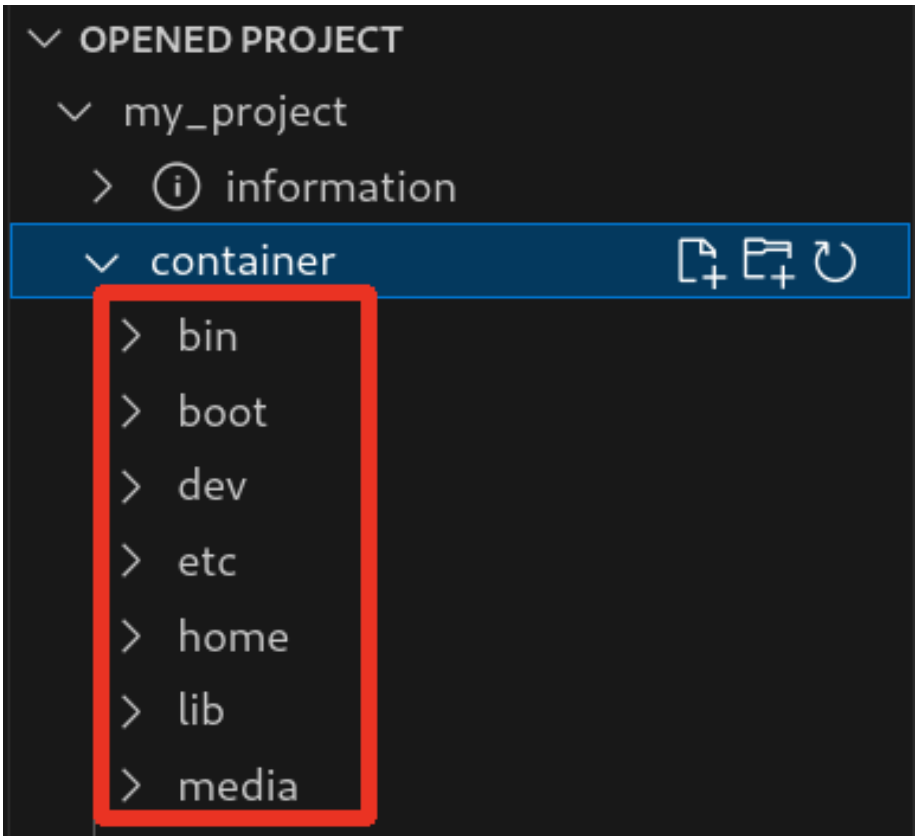


図 3.220 コンテナ内のファイル一覧の例

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。

3.16.4.1. resources ディレクトリについて

「図 3.221. 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。

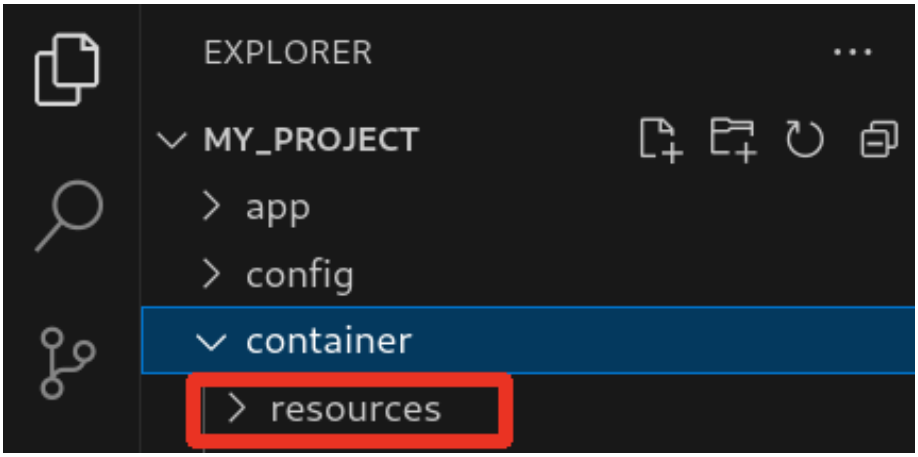


図 3.221 resources ディレクトリ

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。

3.16.4.2. コンテナ内のファイル一覧の再表示

「図 3.219. 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。

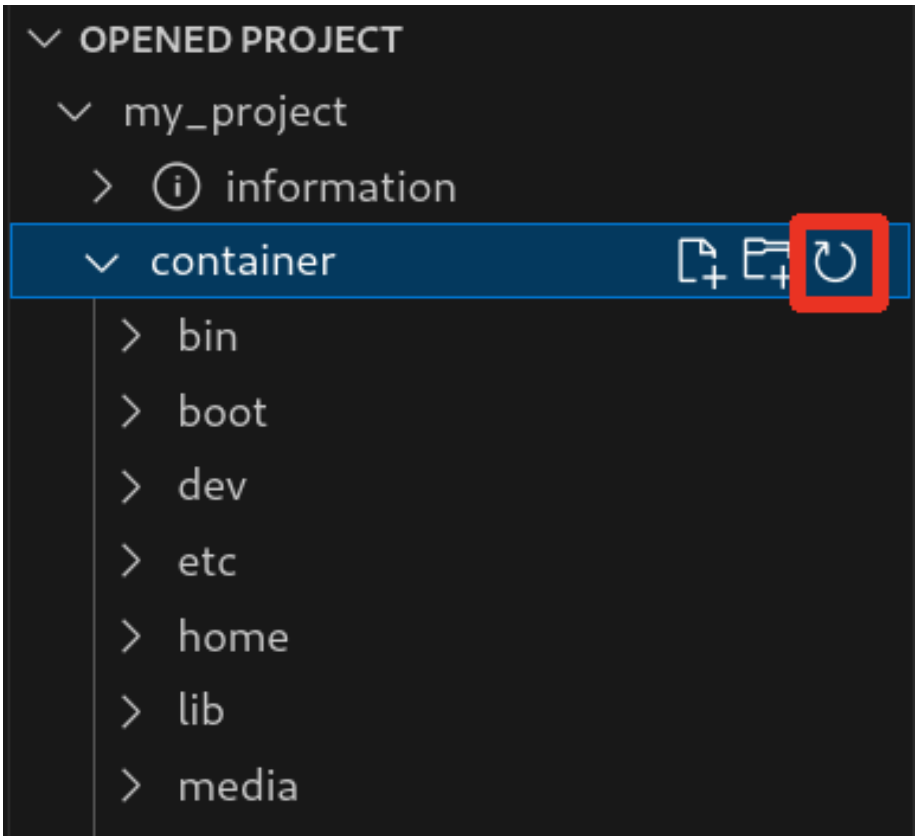


図 3.222 コンテナ内のファイル一覧を再表示するボタン

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

「図 3.223. 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。

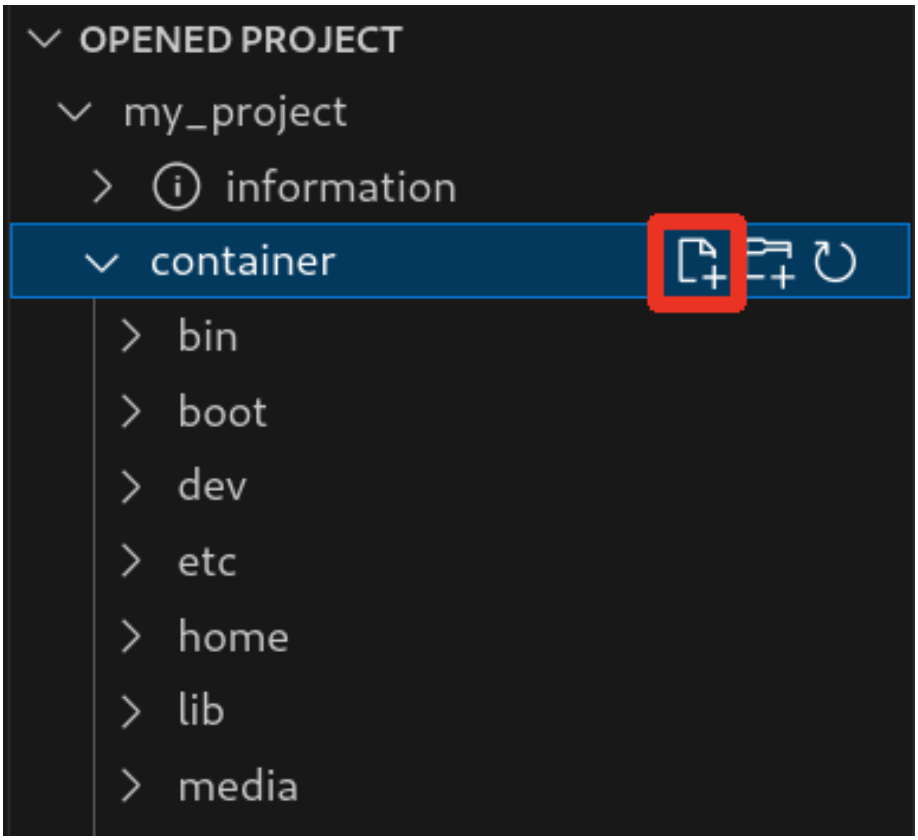


図 3.223 container/resources 下にファイルを追加するボタン

「図 3.224. 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。

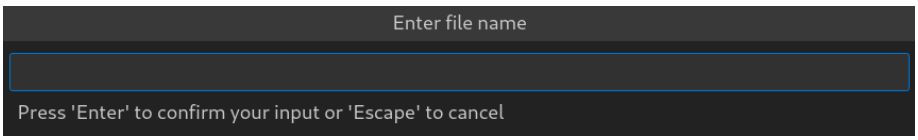


図 3.224 ファイル名を入力

例として、「add_file」というファイル名を入力したとします。「図 3.225. 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。

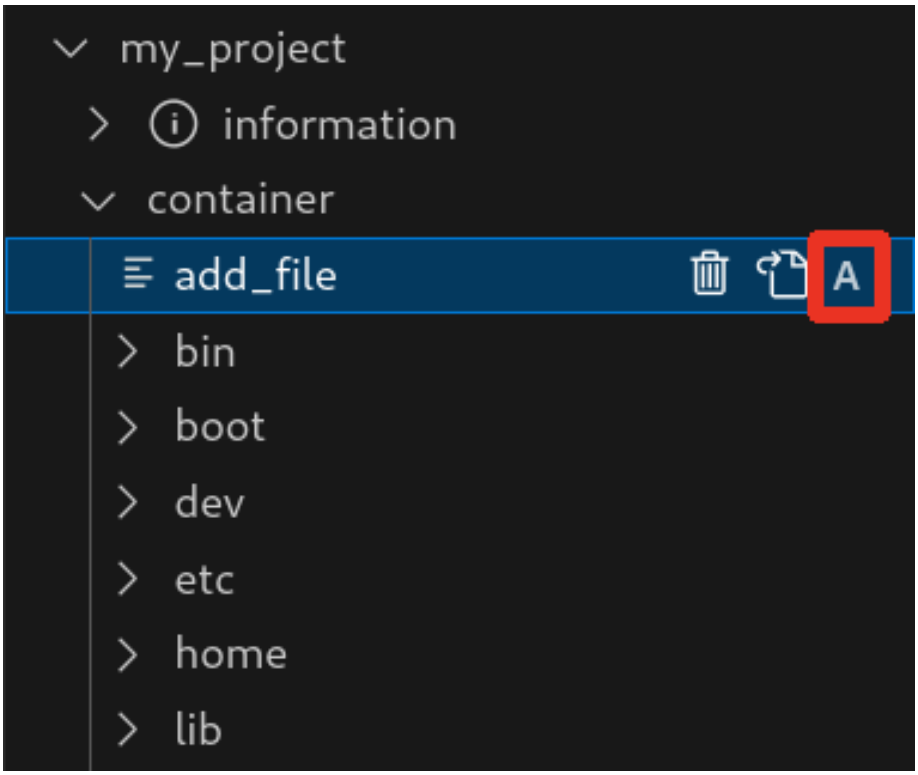


図 3.225 追加されたファイルの表示

また、「図 3.226. 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。

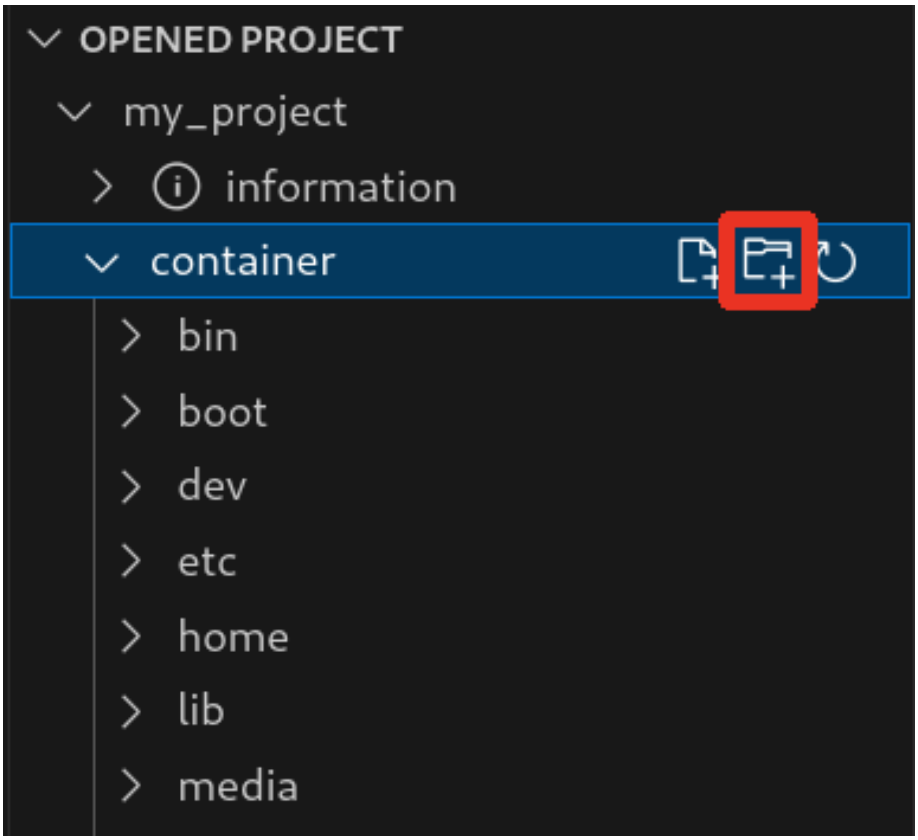


図 3.226 container/resources 下にフォルダーを追加するボタン

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

「図 3.227. 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。

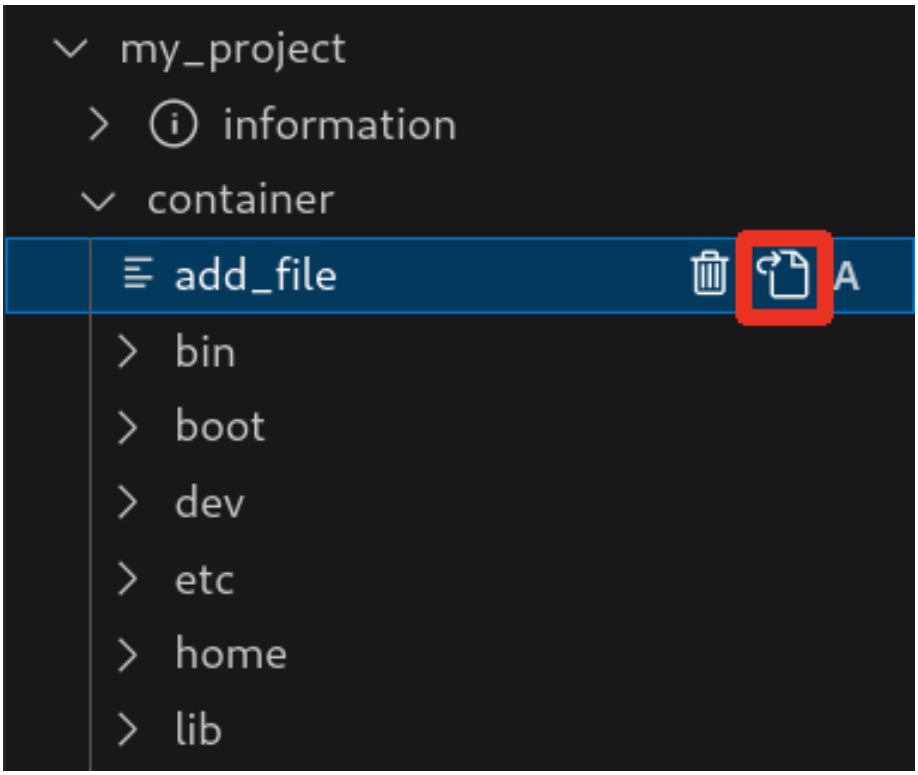


図 3.227 container/resources 下にあるファイルを開くボタン

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

「図 3.227. 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。

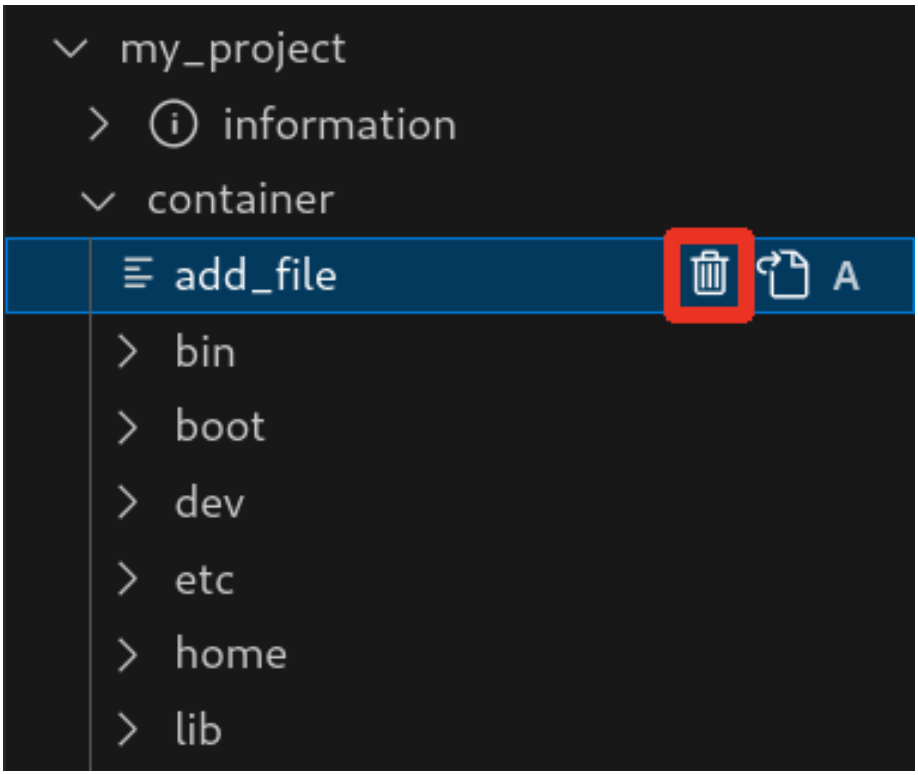


図 3.228 container/resources 下にあるファイルを削除するボタン

ディレクトリも同様に「図 3.227. 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。

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

「図 3.229. 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。

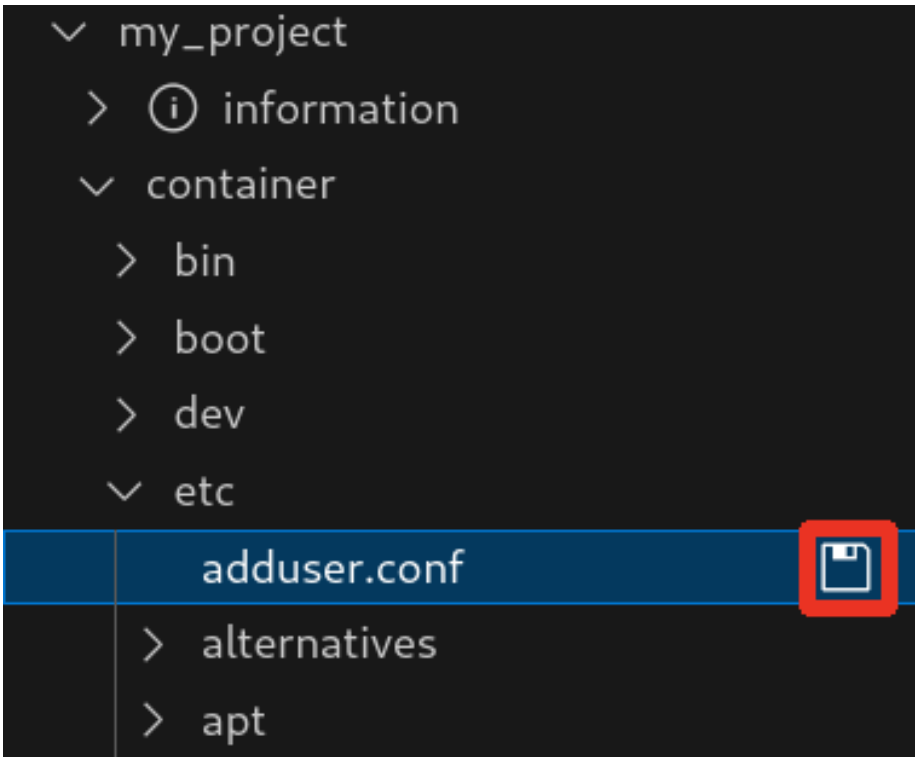


図 3.229 コンテナ内のファイルを container/resources 下に保存するボタン

ファイルが container/resources 下に保存されると、「図 3.230. 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。

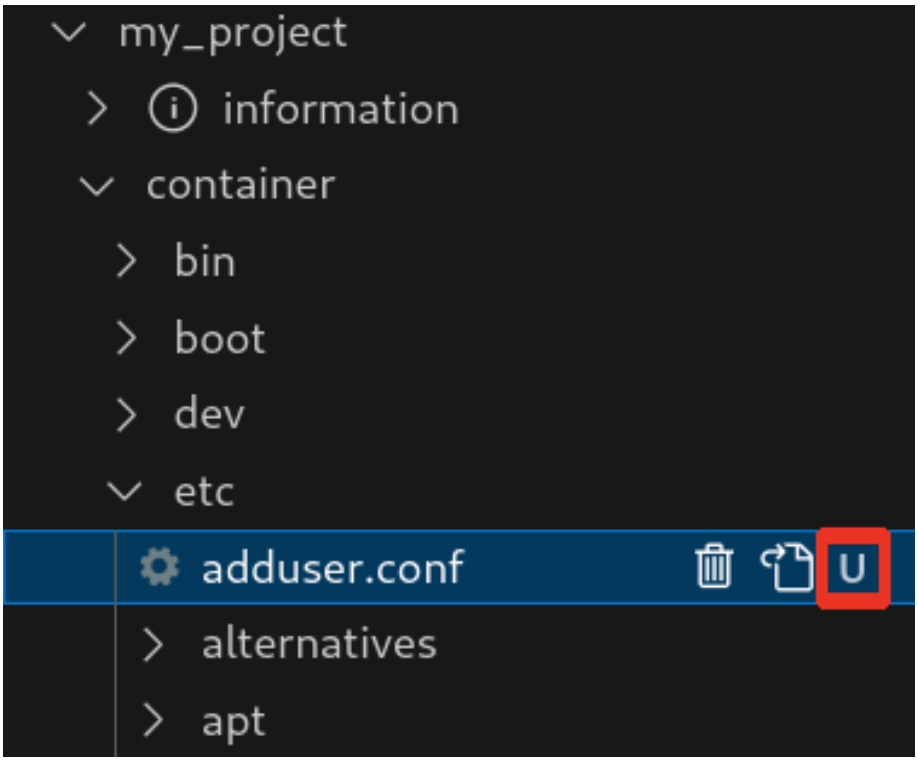


図 3.230 編集前のファイルを示すマーク

container/resources 下にあるファイルを編集して再表示すると、「図 3.231. 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。

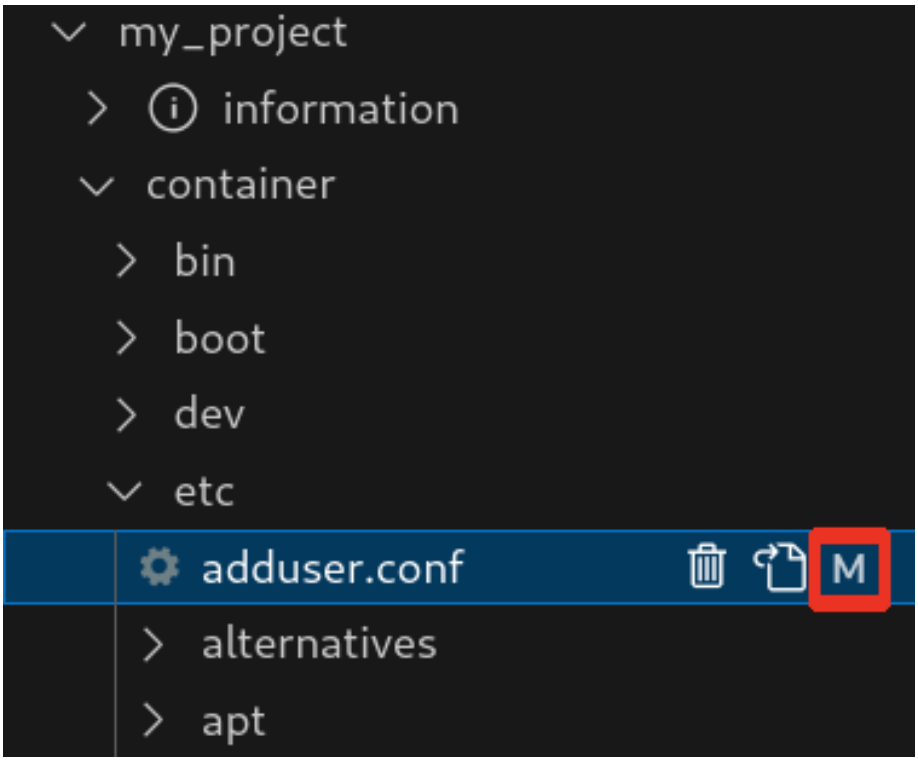


図 3.231 編集後のファイルを示すマーク

3.16.4.7. 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 3.232. 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。

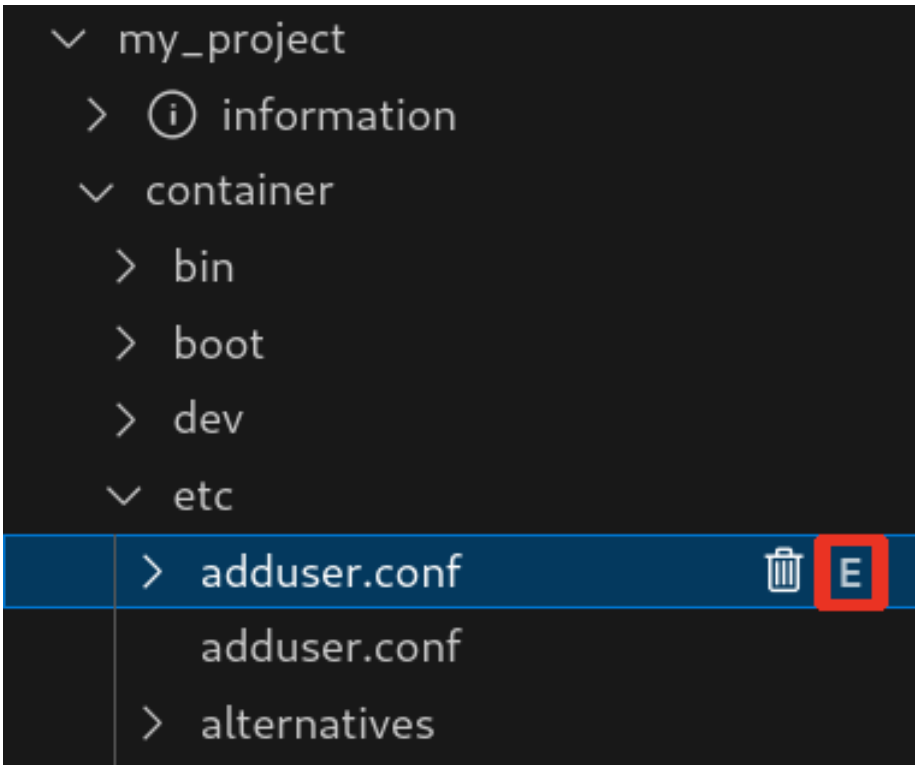


図 3.232 コンテナ内にコピーされないことを示すマーク

3.16.5. 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/build
- my_project/app/lib

3.16.6. Armadillo 上でのセットアップ

3.16.6.1. 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

「3.16.3.9. 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。

3.16.6.2. 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 3.233. 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。

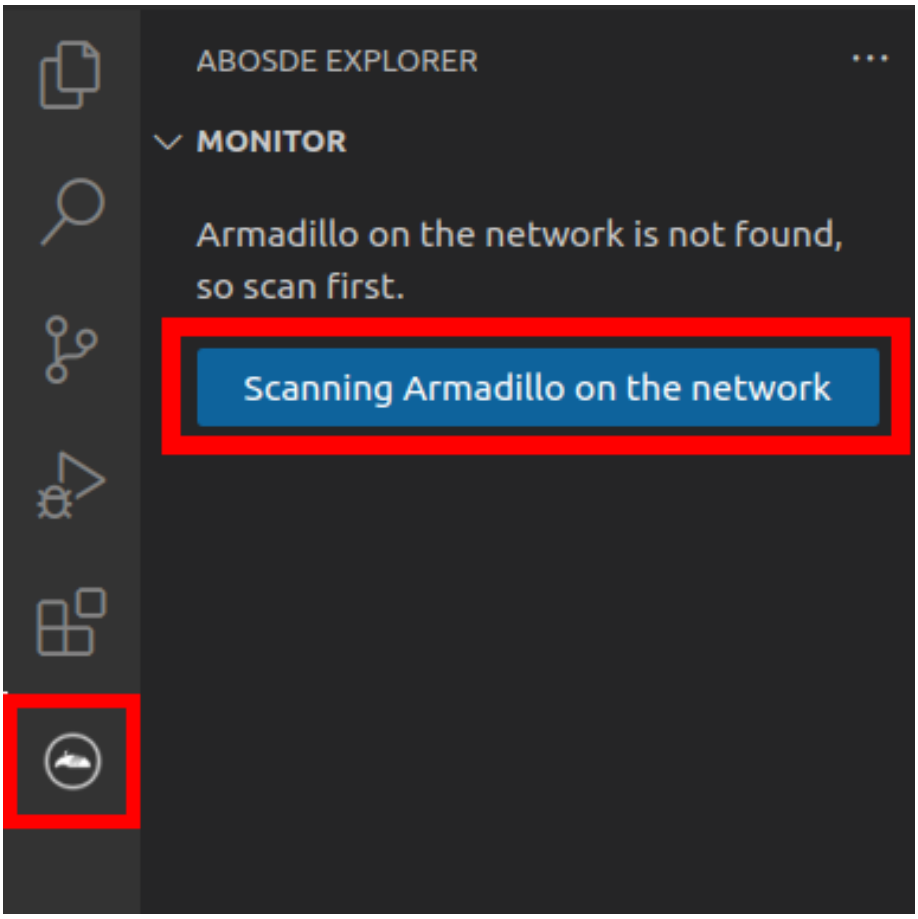


図 3.233 ABOSDE で ローカルネットワーク上の Armadillo をスキャンする

「図 3.234. 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。

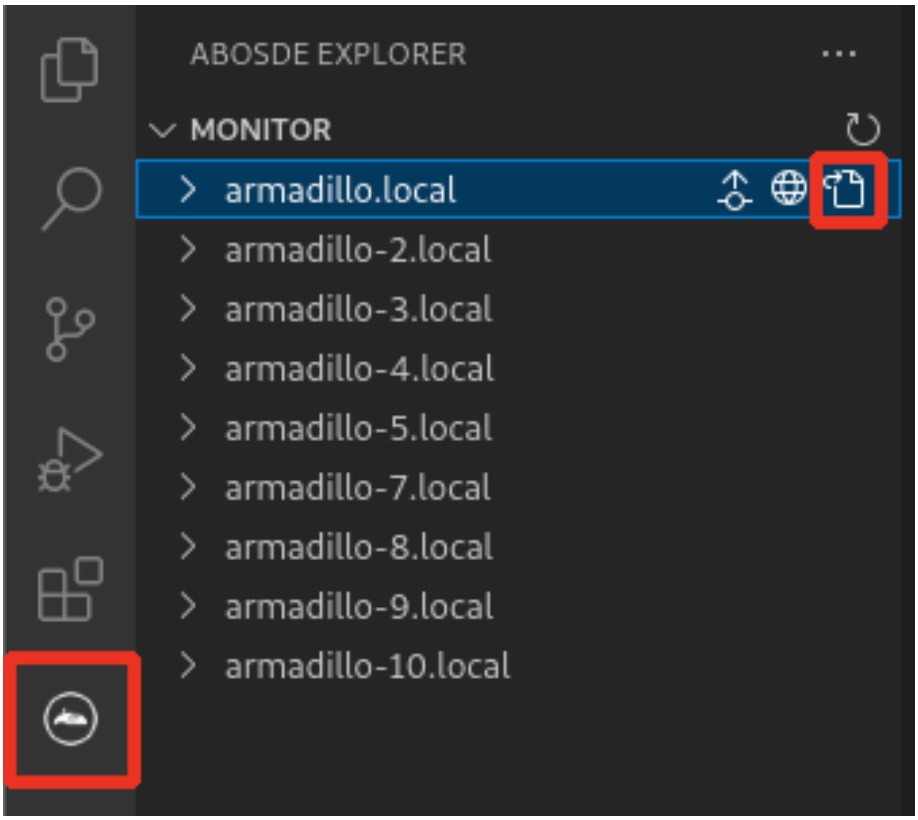


図 3.234 ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する

「図 3.235. 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。

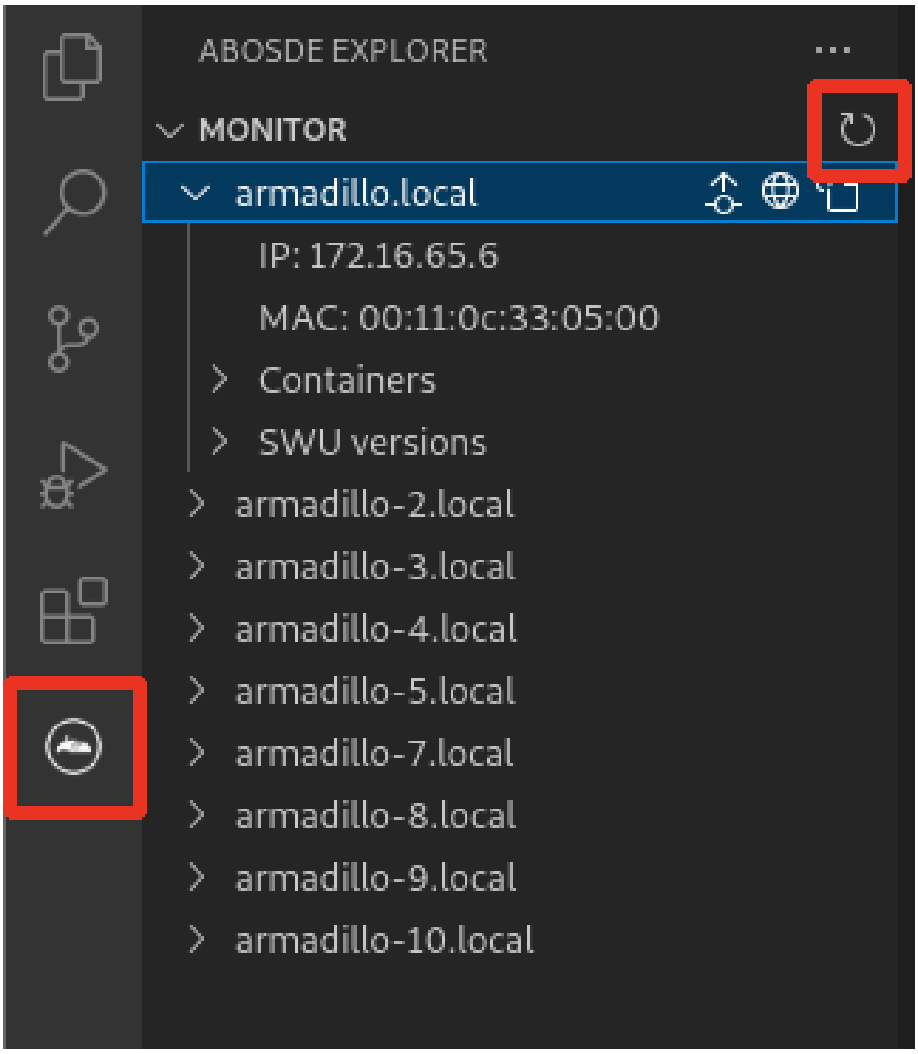


図 3.235 ABOSDE に表示されている Armadillo を更新する

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 3.236 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。

3.16.6.3. 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。

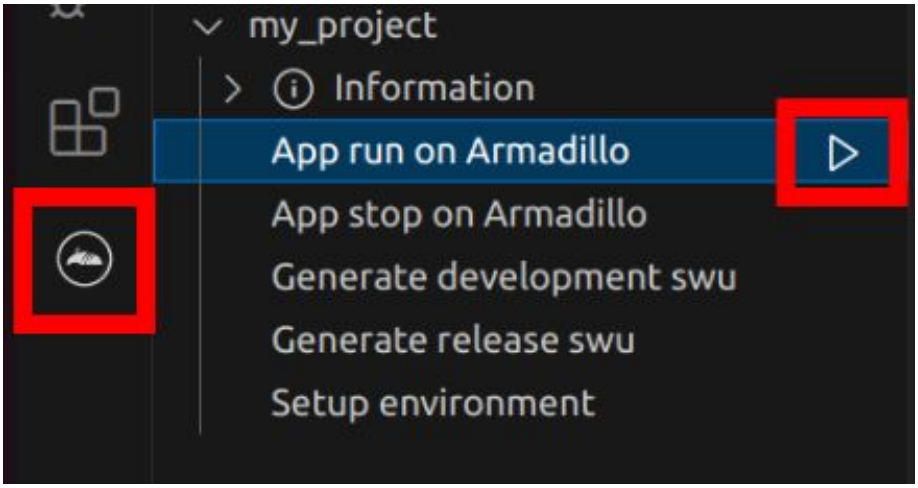


図 3.237 Armadillo 上でアプリケーションを実行する

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 3.238 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。

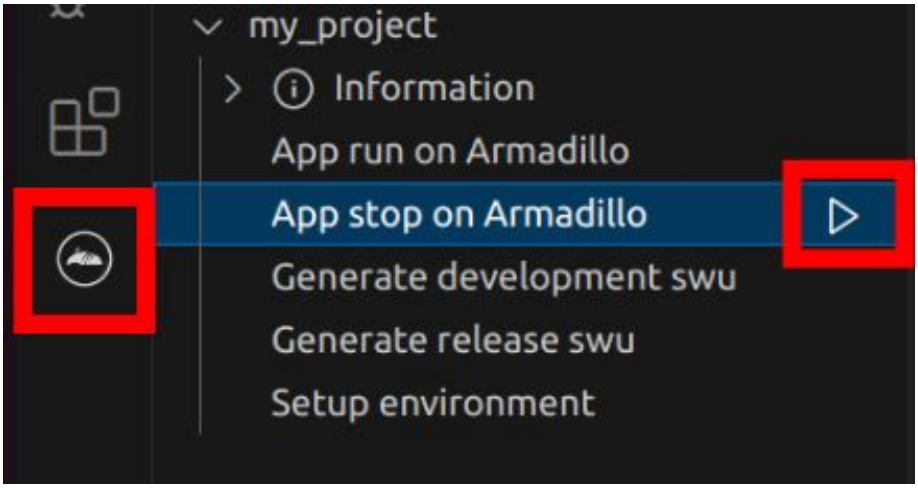


図 3.239 アプリケーションを終了する

3.16.7. 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 「3.17. SBOM 生成に関わる設定を行う」 を参照してください。

3.16.8. 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。

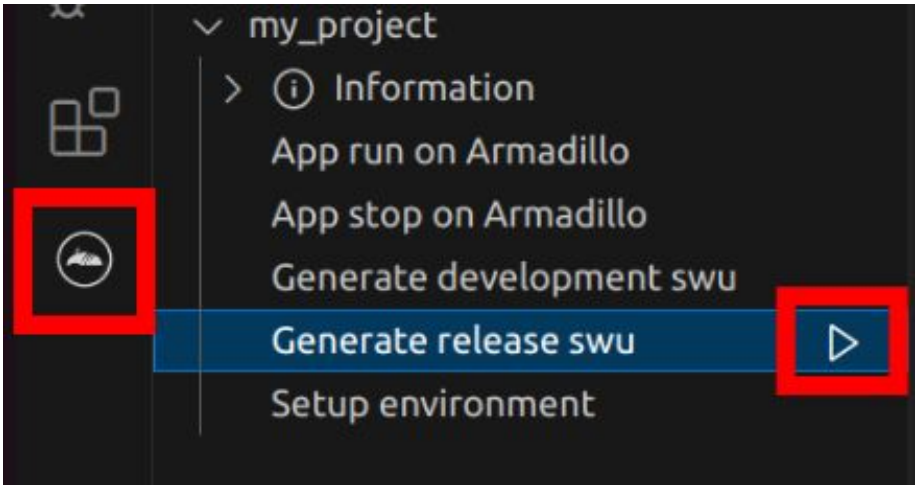


図 3.240 リリース版をビルドする



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。

3.16.9. 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。

3.16.10. 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。

3.17. 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 「6.27.2. 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 3.241. 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 3.241 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ませんので、「図 3.242. 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 3.242 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 3.243. 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 3.243 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ませんので、「図 3.244. 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 3.244 python3-make-sbom のインストールコマンド

3.17.1. 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.tmpl 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 「6.27.2.4. SWU イメージと同時に SBOM を作成する」 をご覧ください。

3.18. 生成した SBOM をスキャンする

SBOM の利点のひとつに、スキャンツールに入力することでソフトウェアに含まれる脆弱性を検出することができる点が挙げられます。ここでは、 Google が提供しているオープンソース SBOM スキャンツール OSV-Scanner[8] を用いて、開発したソフトウェアに既知の脆弱性が含まれているかを確認する方法を紹介します。



OSV-Scanner は脆弱性を確認するため、検査対象のソフトウェアの情報を api.osv.dev に送信しています。送信されるのはソフトウェアに関する以下の情報です。

- パッケージ名
- パッケージエコシステム(npm, crates.io など)
- バージョン

3.18.1. OSV-Scanner のインストール

以下の手順はすべて ATDE 上で行います。

OSV-Scanner は GitHub にてビルド済みの実行ファイルが配布されているのでそちらを使用します。OSV-Scanner のリリースページ [https://github.com/google/osv-scanner/releases]から、最新の実行ファイル(osv-scanner_linux_amd64)をクリックしてダウンロードしてください。

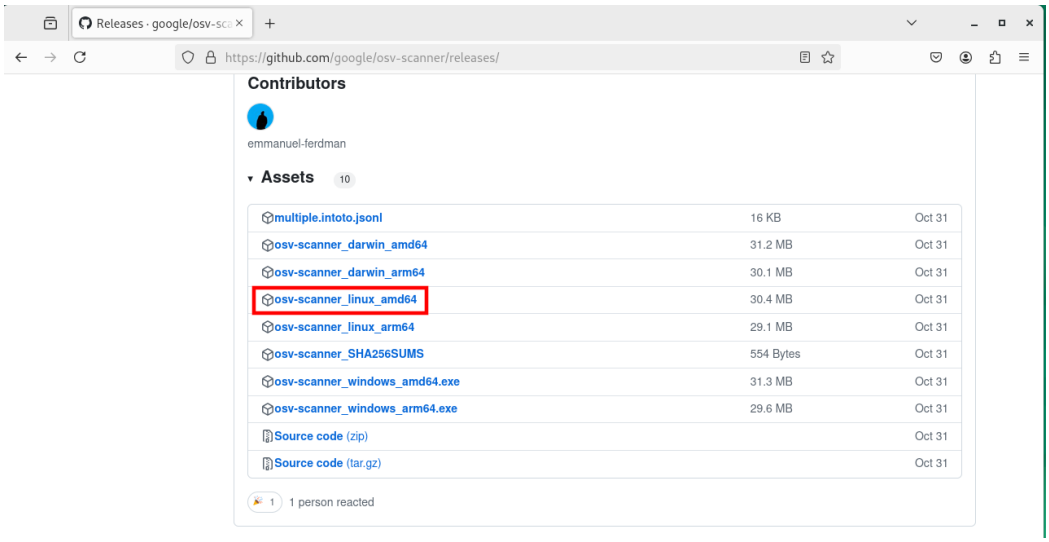


図 3.245 OSV-Scanner の実行ファイルをダウンロード

[8]OSV-Scanner: https://github.com/google/osv-scanner

次に、「図 3.246. OSV-Scanner をインストールする」のコマンドを実行することで、 OSV-Scanner がインストールされます。

```
[ATDE ~]$ sudo install ~/ダウンロード/osv-scanner_linux_arm64 /usr/local/bin/osv-scanner
```

図 3.246 OSV-Scanner をインストールする

osv-scanner --help コマンドを実行して、正しくインストールされていることを確認してください。

```
[ATDE ~]$ osv-scanner --help
NAME:
   osv-scanner - scans various mediums for dependencies and checks them against the OSV database

USAGE:
   osv-scanner [global options] command [command options]

VERSION:
   1.9.1

COMMANDS:
   scan     scans various mediums for dependencies and matches it against the OSV database
   fix      [EXPERIMENTAL] scans a manifest and/or lockfile for vulnerabilities and suggests changes for remediating them
   help, h  Shows a list of commands or help for one command

GLOBAL OPTIONS:
   --help, -h     show help
   --version, -v  print the version
```

図 3.247 OSV-Scanner がインストールされたことを確認する

これで OSV-Scanner のインストールが完了しました。

3.18.2. OSV-Scanner でソフトウェアの脆弱性を検査する

「図 3.248. OSV-Scanner を用いて SBOM をスキャンする」に示すコマンドを実行することで、ソフトウェアに含まれる既知の脆弱性を検出します。ここでは例として ABOSDE で開発したアプリケーションの SBOM をスキャンします。SBOM 生成については 「6.27.2.4. SWU イメージと同時に SBOM を作成する」 を参照してください。

```
[ATDE ~]$ osv-scanner scan --sbom ~/my_project/development.swu.spdx.json --format markdown ❶
Scanned /home/atmark/my_project/development.swu.spdx.json as SPDX SBOM and found 97 packages
5 unimportant vulnerabilities have been filtered out.
Filtered 5 vulnerabilities from output
| OSV URL | CVSS | Ecosystem | Package | Version | Source |
| --- | --- | --- | --- | --- | --- |
| https://osv.dev/CVE-2022-3715 | 7.8 | Debian | bash | 5.1-2+deb11u1 | development.swu.spdx.json |
| https://osv.dev/CVE-2016-2781 | 6.5 | Debian | coreutils | 8.32-4 | development.swu.spdx.json |
| https://osv.dev/CVE-2021-33560 | 7.5 | Debian | libgcrypt20 | 1.8.7-6 | development.swu.spdx.json
```

```
|
| https://osv.dev/CVE-2024-2236 |  | Debian | libgcrypt20 | 1.8.7-6 | development.swu.spdx.json |
```

図 3.248 OSV-Scanner を用いて SBOM をスキャンする

❶ 今回は見やすさのために format を markdown に設定しています

上記の例では、 development.swu に含まれるソフトウェアから既知の重要な脆弱性が 5 件検出されました。OSV URL 列の URL にアクセスすることで各脆弱性の詳細を確認することができます。

3.19. 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。

3.19.1. 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 3.249. 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 3.249 メモリの空き容量の確認方法

3.19.2. 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。

3.20. 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 3.250 削除されるユーザー設定とユーザーデータを確認

❶ 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/mmcblk2p1
Cloning rootfs
Updating appfs snapshots
Reusing up-to-date bootloader
Rollback clone successful
WARNING: Rebooting!
```

図 3.251 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。

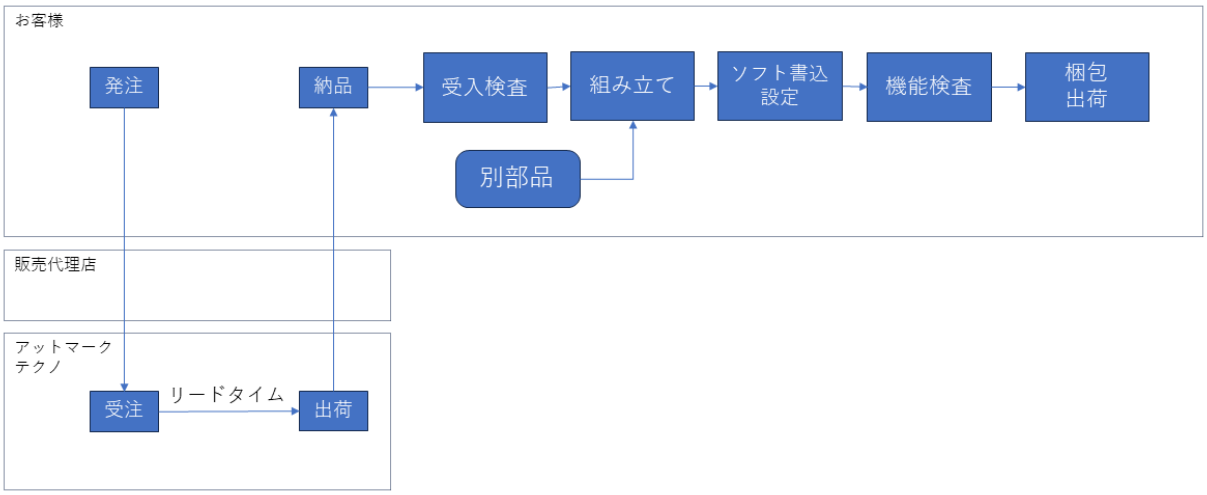


図 4.1 Armadillo 量産時の概略図

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ッティング作業、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ッティング、検査、作業については、実施可能な業者をご紹介する等、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	・ 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	・ インストール実行には起動デバイスを SD カードに設定する必要がある ・ 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	・ 起動デバイスを変更する必要がない ・ 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	・ 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

「 abos-ctrl update 」で Armadillo Base OS を更新できます。

/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を行いますので、デフォルトでは 最新の Armadillo Base OS へアップデートします。



Armadillo Base OS 3.19.1-at.3 以前のバージョンで abos-ctrl update を利用できないか、 /etc/swupdate.watch ファイルに記載されていない場合は任意の URL にある SWU もインストール可能です。

「図 4.4. Armadillo Base OS を最新にアップデートする」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-610/image/baseos-600-latest.swu'
```

図 4.4 Armadillo Base OS を最新にアップデートする

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。

❷ 新しい root ユーザ用パスワードを入力します。

❸ 再度新しい root ユーザ用パスワードを入力します。

❹ atmark ユーザのパスワードを変更します。

❺ 新しい atmark ユーザ用パスワードを入力します。

❻ 再度新しい atmark ユーザ用パスワードを入力します。

❼ 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。

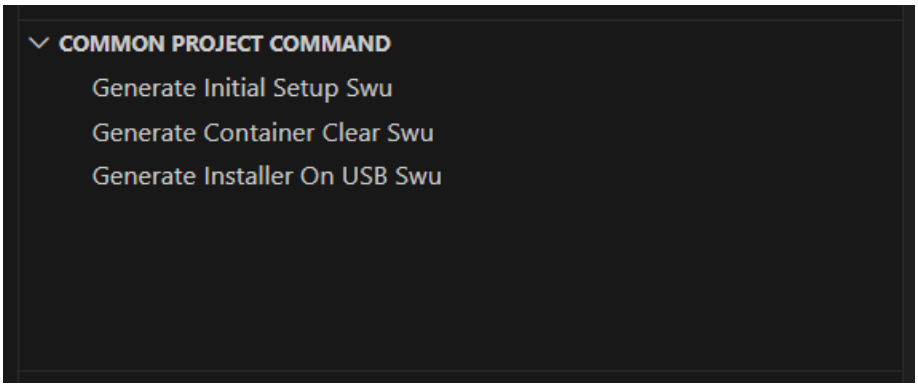


図 4.6 make-installer.swu を作成する

次に、対象製品を選択します。

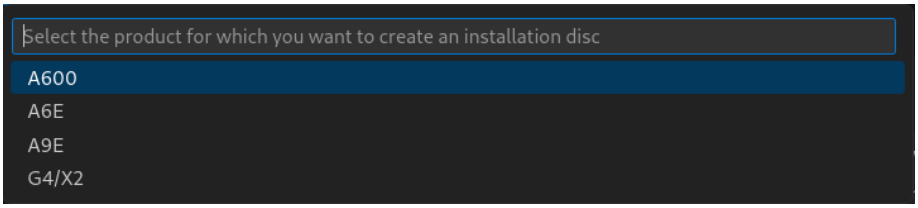


図 4.7 対象製品を選択する

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-610 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-610 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

ては「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。

Armadillo の出荷時は、JTAG ポートは有効なままで出荷されます。JTAG が有効なままですと攻撃者が悪意のあるコードを実行する、メモリをダンプして鍵などセキュアな情報を取り出すなどの行為が可能となります。

SD ブートは SD メディアを挿すだけで起動する便利な機能ですが、その反面、SD メディアの盗難や流出によってシステムへの侵入、SD ブートを利用したセキュアブート鍵に対する攻撃が考えられます。

JTAG と SD ブートの無効化はこれらのセキュリティリスクに対して有効です。

JTAG と SD ブートを無効にするには `abos-ctrl installer-setting` コマンドを実行します。



SD ブートはシステムの復旧の役割も担っています。SD ブートを無効化することによって、eMMC ブートでは起動できない状態に陥った場合、合わせて JTAG の無効化が設定されていると、二度と復旧することができないデバイスになる (廃棄するしかない) 可能性があることに留意してください。



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



この手順で作成したインストールディスクの使用は慎重に行ってください。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
Upcase 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

❶ y を入力し Enter を押下します。

❷ 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ています。Armadillo から microSD カードを抜去してください。



セキュリティーの観点から、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	send_log 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" ❺

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

図 4.16 ip_config.txt の内容

❶ 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。

❷ 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。

❸ 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。

❹ ゲートウェイアドレスを指定します。

❺ 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。

❻ 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。

❷ 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。

❸ 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-610 を組み立てる際に必要な情報を紹介します。

4.7.1. 拡張ボードの組み付け

Armadillo-610 の拡張インターフェース(CON2)に拡張ボードを接続する場合は、作成した拡張ボードにもよりますが、「3.1.4.2. Armadillo-610 開発セットの組み立て」を参考に組み付けを行ってください。

4.7.2. オプションの組み付け

Armadillo-610 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、「6.29. 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. 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

5.2.4.1. 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。

5.2.4.2. 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。

5.2.5. 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、「3.1.7.5. 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。

「3.14. 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、「3.14. 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 「6.27.2. 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。

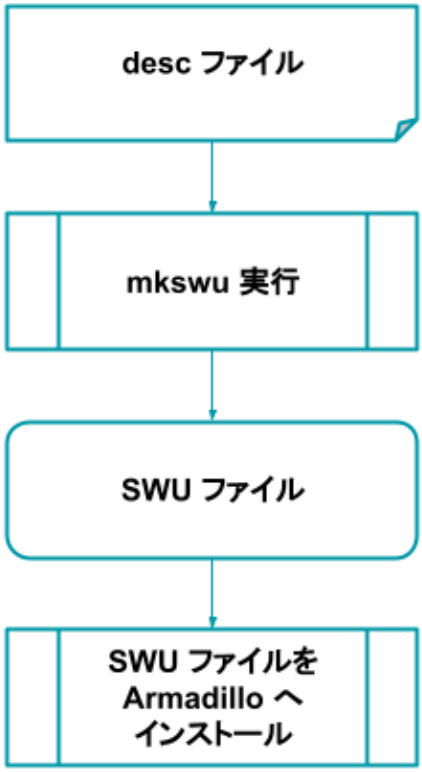


図 5.9 desc ファイルから Armadillo へ SWU イメージをインストールする流れ

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. Armadillo Twin を利用してソフトウェアの脆弱性チェックを行う

ソフトウェア開発時には発見されなかった脆弱性が運用時に発見されることがあるため、運用時でもソフトウェアの脆弱性を定期的に確認することが重要です。Armadillo Twin の脆弱性チェック機能では、登録した SBOM を定期的にスキャンし、新たに脆弱性が発見された際に管理画面に表示します。機能の詳細は Armadillo Twin ユーザーマニュアル 「SWU イメージを管理する」 [https://manual.armadillo-twin.com/management-swu-image/] をご確認ください。

5.7. eMMC の寿命を確認する

5.7.1. 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。

5.7.2. 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。

5.8. 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。

5.9. Armadillo を廃棄する

運用を終了し Armadillo を廃棄する際、セキュリティー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 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。

「図 6.1. OverlayFS の構成」のように、Armadillo Base OS では「下位層」である eMMC 上に存在するファイルシステムに対して「上位層」である RAM 上のファイルシステムを統合しています。

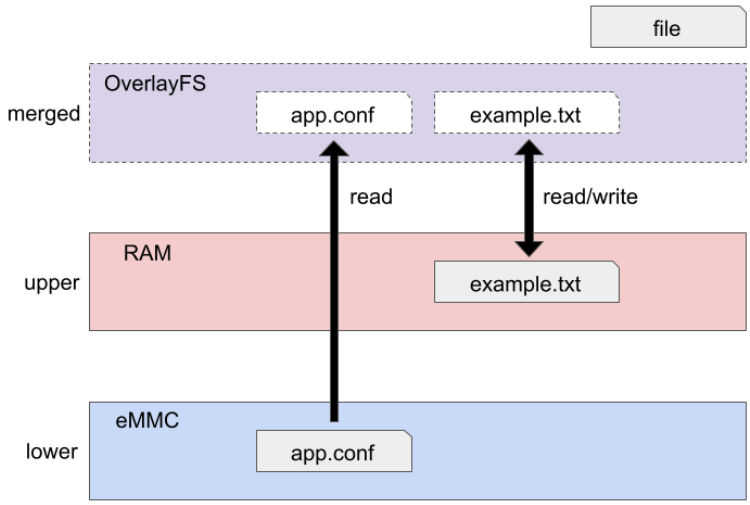


図 6.1 OverlayFS の構成

このシステムを OverlayFS と呼び、ユーザーランド上では OverlayFS がファイルシステムとして見えています。

Armadillo Base OS でファイルを作成すると RAM 上でファイル（またはディレクトリ）が作成されます。これにより、ファイル内容を変更した後 Armadillo の電源を切ると変更内容は保持されません。

`persist_file` コマンドは RAM 上のファイル（またはディレクトリ）を eMMC 上に反映するためのコマンドです。

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` コマンドを使用する必要があります。

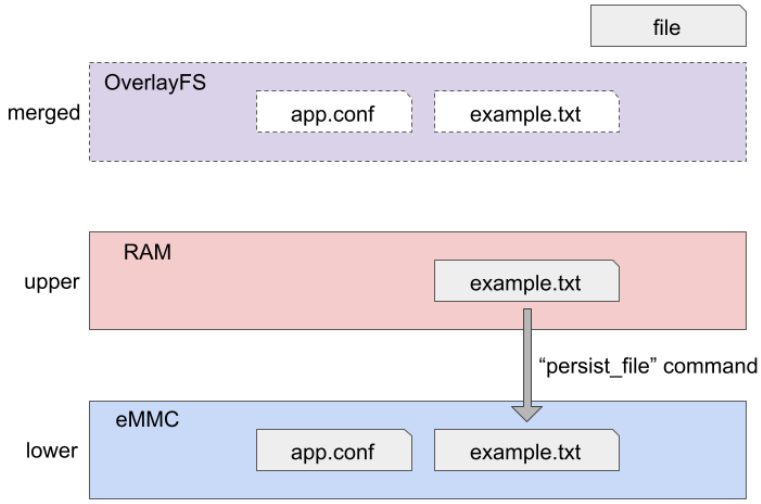


図 6.2 persist_file コマンドの説明

OverlayFS からファイルを書き込むと RAM 上のファイルに書き込みが行われます。eMMC 上にのみファイルが存在する場合、「図 6.3. 書き込み時の OverlayFS の挙動」に示すように、そのファイルは eMMC から読み込まれて編集した内容は RAM 上に書き込まれます。

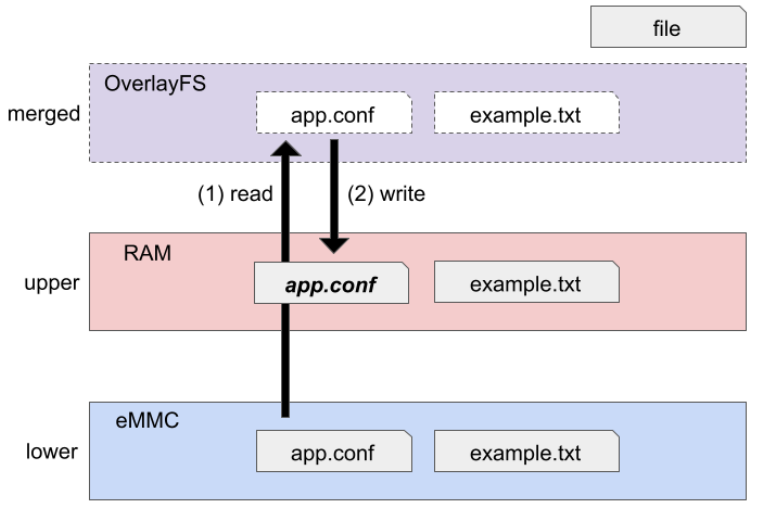


図 6.3 書き込み時の OverlayFS の挙動

編集内容を eMMC 上のファイルに反映させるためには、「図 6.4. 書き込み後の persist_file コマンドの実行」に示すように、persist_file コマンドを実行する必要があります。

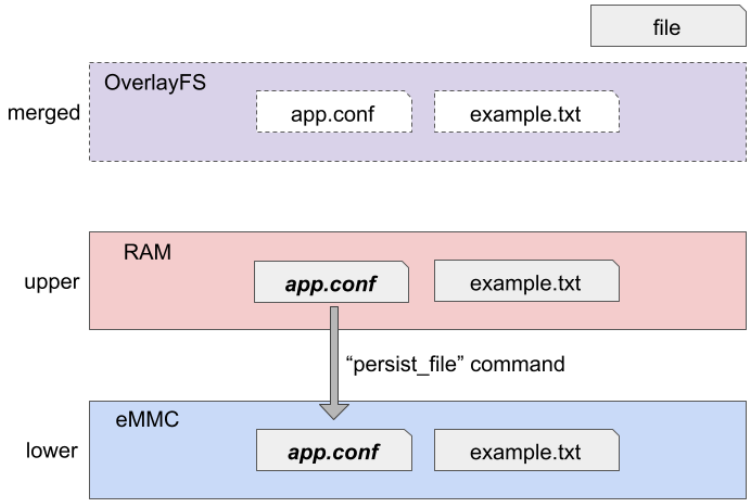


図 6.4 書き込み後の persist_file コマンドの実行

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。

以下、persist_file コマンドの使い方について説明します。

persist_file コマンドの概要を 「図 6.5. 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 6.5 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 6.6 persist_file 保存・削除手順例

❶ 追加・変更したファイルを rootfs へコピーします。

❷ -r を指定すると、ひとつ前の rm -f コマンドで削除したファイルが rootfs からも削除されますのでご注意ください。

❸ すでに rootfs に存在するファイルも一度削除してからコピーするため、このようなメッセージが表示されます。

2. ソフトウェアアップデート後も変更を維持する手順例

```
[armadillo ~]# vi /etc/conf.d/podman-atmark ❶
[armadillo ~]# persist_file -P /etc/conf.d/podman-atmark ❷
[armadillo ~]# tail -n 2 /etc/swupdate_preserve_files ❸
# persist_file 20211216
POST /etc/conf.d/podman-atmark
```

図 6.7 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 6.8 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 6.9 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-610 でも同一の環境がすぐに再現できます。ゲスト OS を必要としない仮想化であるため、アプリケーションの起動が素早いという特徴があります。

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

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-610 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-610 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 6.10 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 6.11 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 6.12 podman images --help の実行例

6.2.2.3. コンテナー一覧を表示する

作成済みコンテナー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                                                
```

```
d6de5881b5fb  docker.io/library/alpine:latest  ls /       12 minutes ago  Exited (0) 11 minutes ago          my_container
```

図 6.13 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 6.14 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 6.15 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 6.16 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 6.17 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 6.18 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 6.19 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なわれてしまいます。

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 6.20 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 6.21 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
9e9d9366072751007b2e70544d76c46b95a7a5a02df658ef0fa3f7dcccf8850a

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

図 6.22 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 6.23 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: 02480aeb44d78f1a44b8791af7edf7d6e1b18707397a1dfb3ff4f21c5ce4a44f
[armadillo ~]# podman images
REPOSITORY                TAG         IMAGE ID      CREATED      SIZE
```

図 6.24 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 6.25 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
"02480aeb44d78f1a44b8791af7edf7d6e1b18707397a1dfb3ff4f21c5ce4a44f"
[armadillo ~]# abos-ctrl podman-rw rmi docker.io/alpine
Untagged: docker.io/library/alpine:latest
Deleted:
02480aeb44d78f1a44b8791af7edf7d6e1b18707397a1dfb3ff4f21c5ce4a44f
[armadillo ~]# podman images
REPOSITORY                TAG         IMAGE ID      CREATED      SIZE
```

図 6.26 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 6.27 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 6.28 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 6.29 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 6.30 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 6.31 コンテナの IP アドレスを確認する実行例

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

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

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

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

図 6.32 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 の中のコンテナが IP の場合 localhost で、 unix socket の場合 abstract path で相互に接続することができます。

```
[armadillo ~]# cat /etc/atmark/containers/mypod.conf
set_type pod
add_ports 80:80

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

[armadillo ~]# podman ps
CONTAINER ID  IMAGE                             COMMAND               CREATED      STATUS                  ⏎
PORTS               NAMES
0cdb0597b610  localhost/podman-pause:4.3.1-1683096588               2 hours ago  Up 2 hours ago          ⏎
0.0.0.0:80->80/tcp  5ba7d996f673-infra
3292e5e714a2  docker.io/library/nginx:alpine  nginx -g daemon o...  2 hours ago  Up 2 hours ago          ⏎
0.0.0.0:80->80/tcp  nginx
```

図 6.33 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 6.34 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 6.35 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 6.36 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ので、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」の状態で作業を行い、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、充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 へインストールしてください。

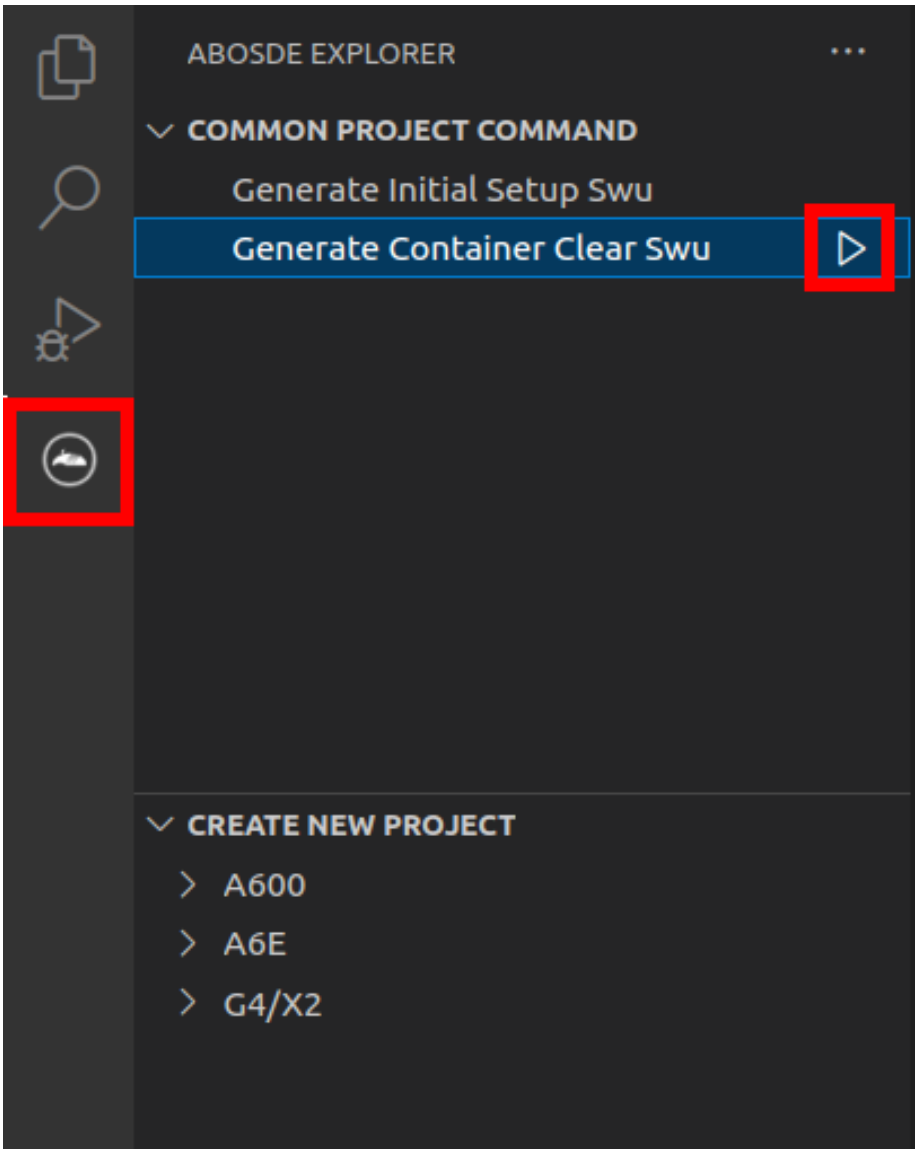


図 6.37 Armadillo 上のコンテナイメージを削除する

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 6.38 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 6.39 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

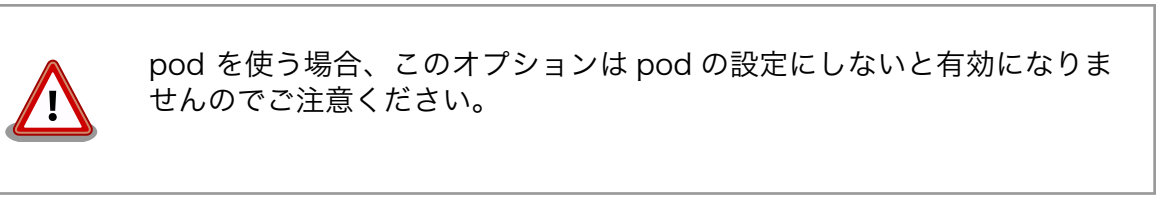
add_ports [ホストポート]:[コンテナポート]

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



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 6.40 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なっても認識される(ホットプラグ)デバイスを設定できます。

通常、コンテナ内からデバイスを扱うためには、あらかじめ Armadillo 本体に当該のデバイスを接続した状態で、コンテナを起動する必要がありますが、 add_hotplugs を使用することでホットプラグに対応できます。

例: add_hotplugs input

add_hotplugs に指定できる主要な文字列とデバイスファイルの対応について、「表 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
input	マウスやキーボードなどの入力デバイス	/dev/input/mouse0, /dev/input/event0 など
video4linux	USB カメラなどの video4linux デバイスファイル	/dev/video0 など
sd	USB メモリなどの SCSI ディスクデバイスファイル	/dev/sda1 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 6.41. /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 ttyp
  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 6.41 /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
AT_ABOS_VERSION	ABOS のバージョン	3.18.4-at.5
AT_LAN_MAC1	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
AT_PRODUCT_NAME	製品名	Armadillo-610
AT_SERIAL_NUMBER	個体番号	00C900010001
AT_SE_PARAM	セキュアエレメントのデバイスファイルのパス	/dev/i2c-0:0x48

「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 6.42. 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 6.42 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: ロールバック可能の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_healtcheck -- 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 6.43 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 --daemonize CONx_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

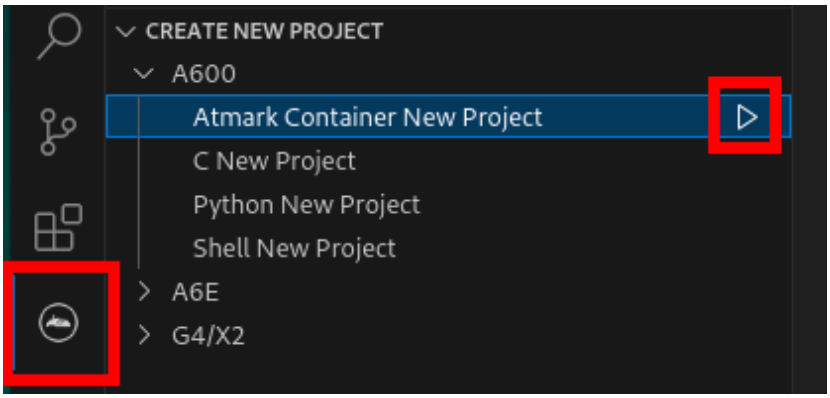


図 6.44 インストール用のプロジェクトを作成する

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。

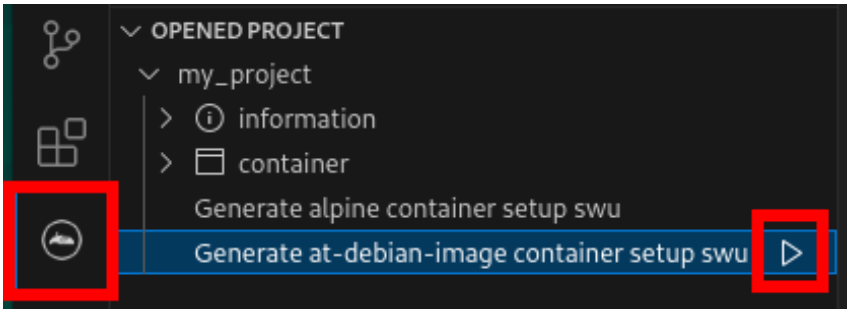


図 6.45 at-debian-image のコンテナイメージをインストールする SWU ファイルを作成する

   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 へ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 「3.17. 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.tmpl` 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 「6.27.2.4. SWU イメージと同時に SBOM を作成する」 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-610 コンテナ [https://armadillo.atmark-techno.com/resources/software/armadillo-610/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 6.46 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-610 コンテナ [https://armadillo.atmark-techno.com/resources/software/armadillo-610/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 6.47 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。

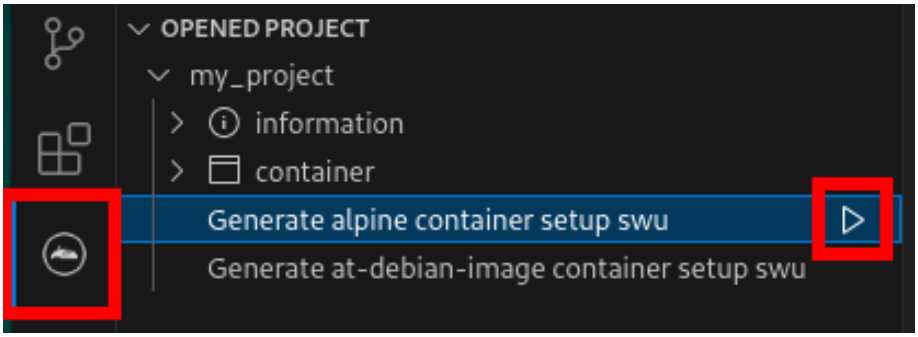


図 6.48 alpine のコンテナイメージをインストールする SWU ファイルを作成する

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 「3.17. 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.tmpl 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 「6.27.2.4. SWU イメージと同時に SBOM を作成する」 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 6.49 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 6.50 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 6.51 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 6.52 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 6.53 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 6.54 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 6.55 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 vsftp 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 6.56 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 6.57 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 6.58 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 vftpd 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 6.59 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 6.60 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 6.61 ユーザを追加する例

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

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

図 6.62 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'
114c5f1dbb7e81293dcb8fbe0c600b861626375b14cfe4023761acaa84fdcad1
[armadillo ~]# podman exec -it sqlite_example sh
[container ~]# apk upgrade && apk add sqlite
```

図 6.63 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 6.64 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-610 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 ❸
add_volumes /run/udev:/run/udev:ro ❹
add_args --cap-add=SYS_ADMIN ❺
[armadillo ~]# podman_start x_example
Starting 'x_example'
26847e21bd519f99466af32fdf0d809e2216d3e8ddf05c185e5428fe46e6a09b
```

図 6.65 X Window System を扱うためのコンテナ起動例

❶ X Window System に必要な tty を設定します。どこからも使われていない tty とします。

❷ 画面描画先となるフレームバッファを設定します。

❸ キーボードやマウスなどを使用可能にするためのデバイスを設定します。

❹ ホスト OS 側の /run/udev をコンテナ内からマウントするように設定します。

❺ X Window System の動作に必要な権限を設定します。

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

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

X.Org X Server 1.20.11
X Protocol Version 11, Revision 0
Build Operating System: linux Debian
Current Operating System: Linux 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 6.66 コンテナ内で X Window System を起動する実行例

Armadillo-610 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 6.67 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 6.68 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 6.69 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 6.70 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 6.71 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

例として、標準状態の Armadillo-610 開発セットにおいて、サスペンド状態から起床要因として利用可能なデバイスを以下に示します。

UART1 (Armadillo-610 拡張ボード: CON3)

起床要因: データ受信

有効化:

```
[armadillo ~]# echo enabled > /sys/class/tty/
ttymxc0/power/wakeup
```

USB OTG2 (Armadillo-610 拡張ボード: CON6)

起床要因: USB デバイスの挿抜

有効化:

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

RTC(i.MX6ULL)

起床要因　アラーム割り込み

有効化

```
デフォルトで有効化されています
```

実行例

```
[armadillo ~]# echo +180 > /sys/class/rtc/rtc0/
wakealarm
[armadillo ~]# poweroff
```

図 6.72 poweroff コマンドで電源を切断し、180 秒後に i.mx6ull の RTC で起床する

RTC(NR3225SA)

起床要因　アラーム割り込み

有効化

```
[armadillo ~]# echo enabled > /sys/bus/i2c/devices/
1-0032/power/wakeup
```

実行例

```
[armadillo ~]# vi /etc/atmark/containers/
rtc_pm_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_volumes /sys
add_devices /dev/rtc0
[armadillo ~]# podman_start rtc_pm_example
Starting 'rtc_pm_example'
8fbef3edda3b7fcea5b1f8cbf960cf469b7e82c4d1ecd35477
076e81fc24e39f
[armadillo ~]# podman exec -ti rtc_pm_example sh
[container ~]# apk add util-linux
[container ~]# rtcwake -m mem -s 5
:（省略）
[  572.720300] printk: Suspending console(s) (use
no_console_suspend to debug)
<ここで 5 秒を待つ>
[  573.010663] OOM killer enabled.
...
```

図 6.73 サスペンド状態にする実行例、rtc で起こす

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、「3.4.6.6. 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-610 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-610 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

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'
c8e3b9b418fc72395db9f3c22b1eb69eb41eaaf790d3b7151047ef066cc4c8ff
[armadillo ~]# podman exec -ti shutdown_example sh
[container ~]# kill -USR2 1  (poweroff)
[container ~]# kill -TERM 1  (reboot)
```

図 6.74 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 6.75 ソフトフ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 6.76 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 6.77 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 6.78 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 6.79. 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。

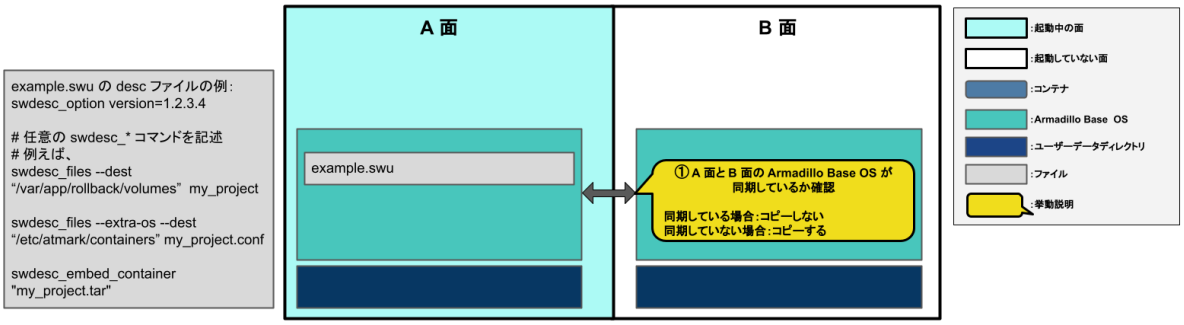


図 6.79 Armadillo Base OS を B 面にコピー

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。

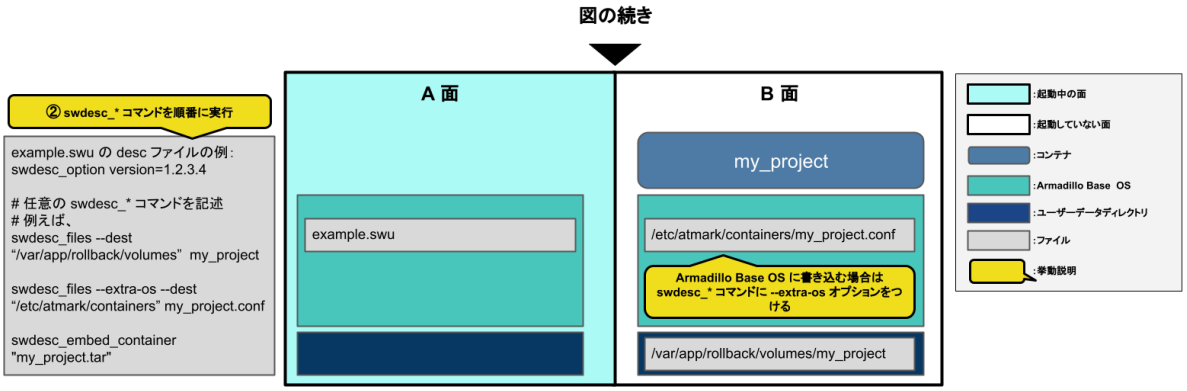


図 6.80 desc ファイルに記述した swudesc_* コマンドを実行

swudesc_* コマンドの種類を「表 6.4. 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 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

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。

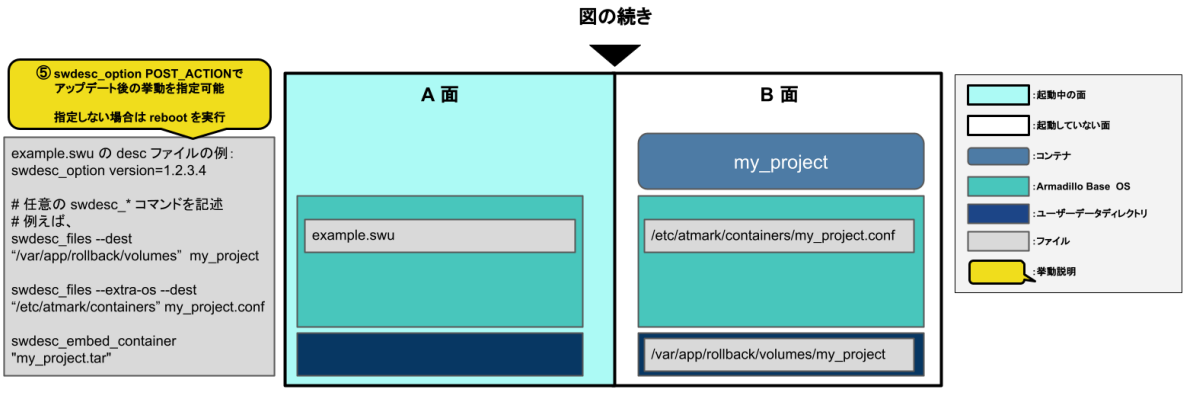


図 6.81 アップデート完了後の挙動

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 6.5. 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 6.82. 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。

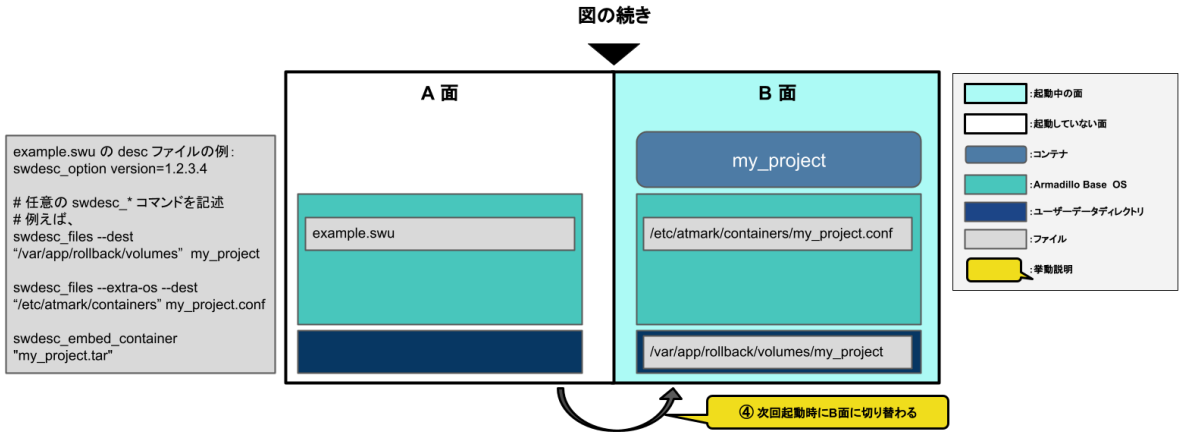


図 6.82 B 面への切り替え

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 6.83. 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。

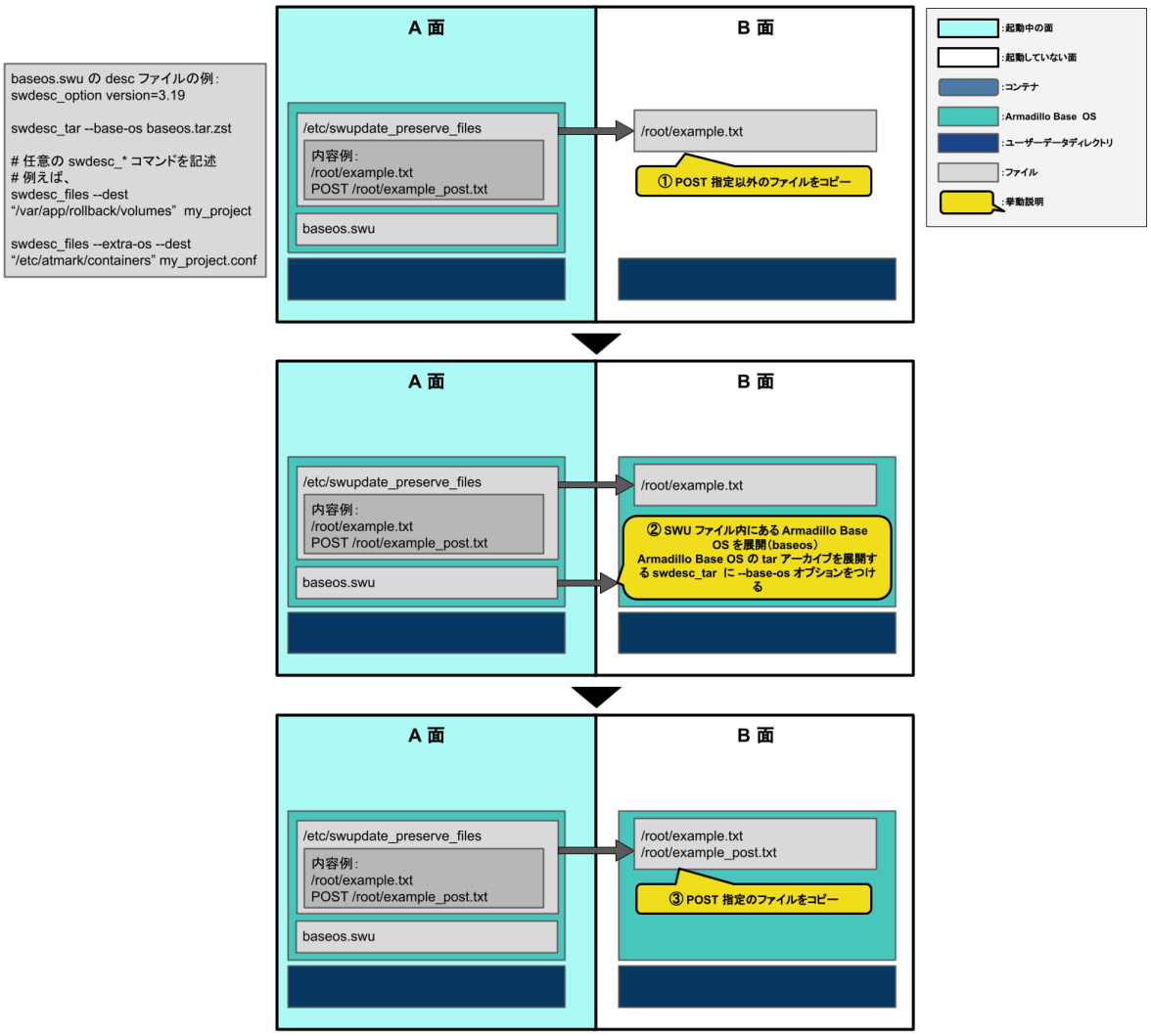


図 6.83 Armadillo Base OS とファイルを B 面にコピー

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 6.84. 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。

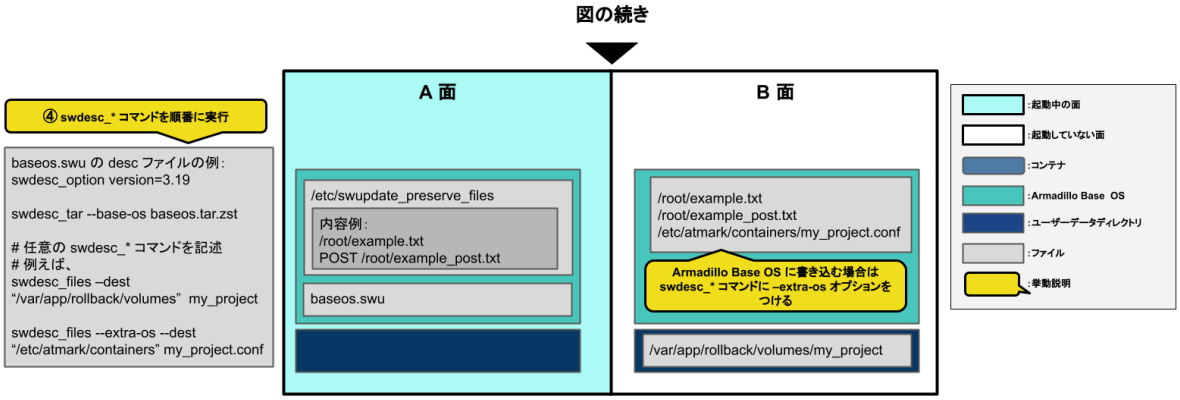


図 6.84 desc ファイルに記述した swudesc_* コマンドを実行

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。

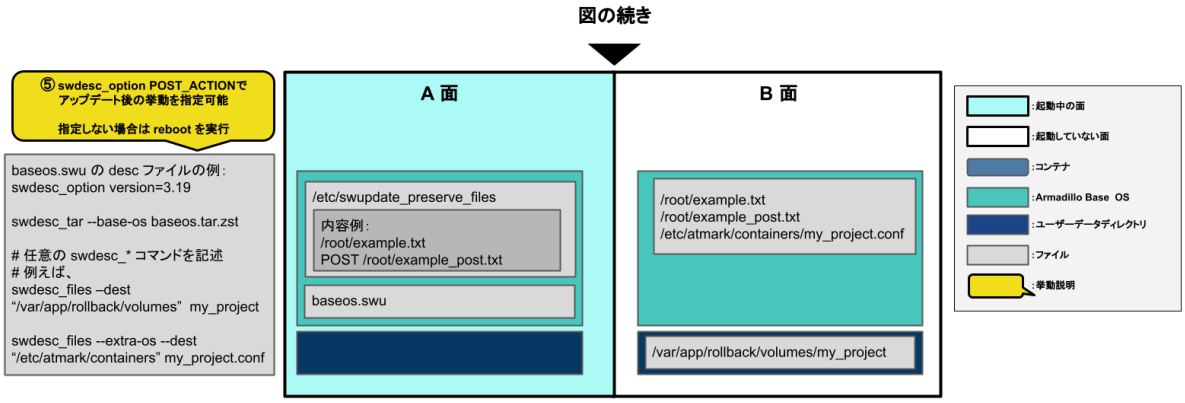


図 6.85 アップデート完了後の挙動

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 6.86. 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。

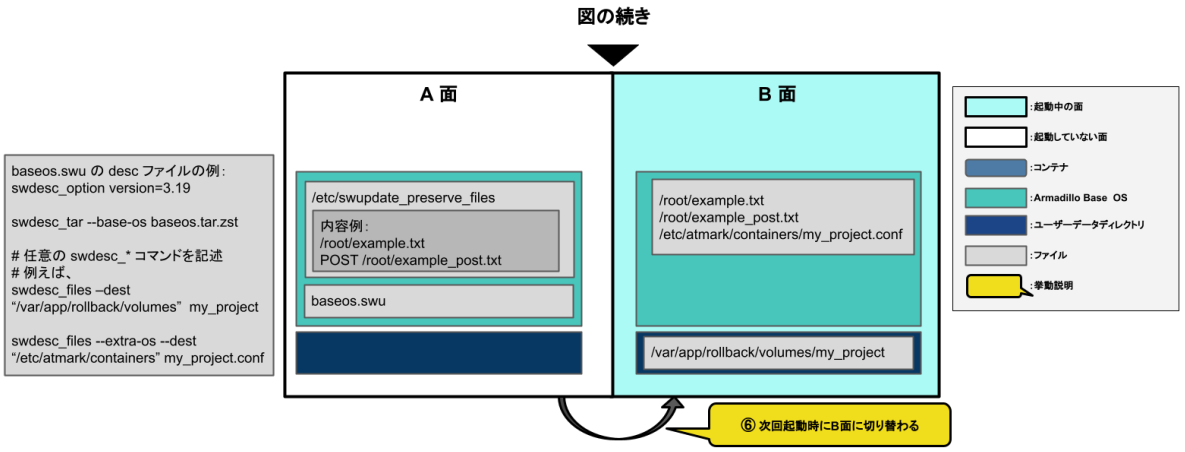


図 6.86 B 面への切り替え（component=base_os)

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ないです。

   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



アットマークテクノがリリースするファームウェアはバージョンのサフィックスとして"-at.[数字]"を含めています。オリジナルのサフィックスをつける場合は、"-at.[数字]"を使用しないことを強く推奨します。

  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。swdesc_files は、機能としては swdesc_tar と同じですが、tar アーカイブを作る手間を省けます。

--dest <dest> で展開先を選ぶことができます。component や指定するオプションによってデフォルトのコピー先が違います。

- --extra-os オプションを指定する場合
  - コピー先：/
  - 例： swdesc_files --extra-os "resources"
    - resources ディレクトリの中身を / に展開します。
- component が 'base_os` または extra_os.* の場合
  - コピー先：/
  - 例： swdesc_files --version extra_os.my_component 1 "resources"
    - resources ディレクトリの中身を / に展開します。
- それ以外の component の場合
  - コピー先：/var/app/rollback/volume/
  - 例：swdesc_files --version my_component 1 --dest "my_project" "resources"
    - resources ディレクトリの中身を /var/app/rollback/volume/my_project に展開します。
    - この場合は /var/app/volumes と /var/app/rollback/volumes 以外にコピーできませんので、--extra-os をご使用ください。

--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 では、--basedir <basedir> でアーカイブ内のパスをどこで切るかを決めます。(このオプションは swdesc_tar にはありません。)

- 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 以外は変更できないのでご注意ください。

コマンドの実行が失敗した場合、アップデートも失敗します。

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-610 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このまま /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。

❸ バージョンが上がるときにしかインストールされませんので、現在の/etc/sw-versions を確認して適切に設定してください。

❹ イメージを作成します。パスワードは証明鍵の時のパスワードです。

6.4.11.3. 例: swupdate_preserve_files で Linux カーネル以外の Armadillo-610 向けのイメージをインストールする方法

Armadillo-610 向けのアップデートイメージに Linux カーネルが含まれています。

swupdate_preserve_files を使って、以下のコマンドでインストール後に現在のカーネルをコピーして更新させないようにします。

```
[armadillo ~]# echo 'POST /boot' >> /etc/swupdate_preserve_files
[armadillo ~]# echo 'POST /lib/modules' >> /etc/swupdate_preserve_files ❶
[armadillo ~]# persist_file /etc/swupdate_preserve_files ❷
```

❶ swupdate_preserve_files に /boot と /lib/modules を保存するように追加します。

❷ 変更した設定ファイルを保存します



/usr/share/mkswu/examples/kernel_update*.desc のように update_preserve_files.sh のヘルパーで、パスを自動的に /etc/swupdate_preserve_files に追加することができます。

```
[ATDE ~/mkswu]$ cat example.desc
swdesc_script "$SCRIPT_DIR/examples/update_preserve_files.sh" -- ¥ ❶
        "POST /boot" ¥
        "POST /lib/modules"
```

❶ スクリプトの内容確認する場合は /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-610 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
証明書のコモンネーム(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 6.87 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フレーズを 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 6.88 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 6.89 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 6.90 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 6.91 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。そのための、Armadillo の動作設定を行う機能ですから、動作設定以外のこと、たとえば、Armadillo の動作状態を監視したりすることは、できません。さらに、Armadillo をインターネットから設定操作する、リモート操作もできません。セキュリティの観点から、ABOS Web は、同じ LAN 内からの接続しか受け付け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 6.92. コンテナ管理」の画面に遷移します。



図 6.92 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 6.93. SWU インストール」の画面に遷移します。



図 6.93 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ければ、"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。



図 6.94 最新バージョンに更新

この画面の "更新" を押すと、プルダウンメニューで指定されている SWU イメージをダウンロードして、 Armadillo にインストールします。プルダウンメニューに表示される SWU イメージは、 /etc/swupdate.watch に記述されている URL から取得しています。

青色に表示されている「こちら」を押すと、Armadillo Base OS の SWU イメージのダウンロードページにアクセスできます。

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。



図 6.95 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 6.96. 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。



図 6.96 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

時刻が同期されてない状態では 「図 6.97. 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 6.97 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 6.98. 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 6.98 ネットワークタイムサーバーの設定項目

最後に、「図 6.99. 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 6.99 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:

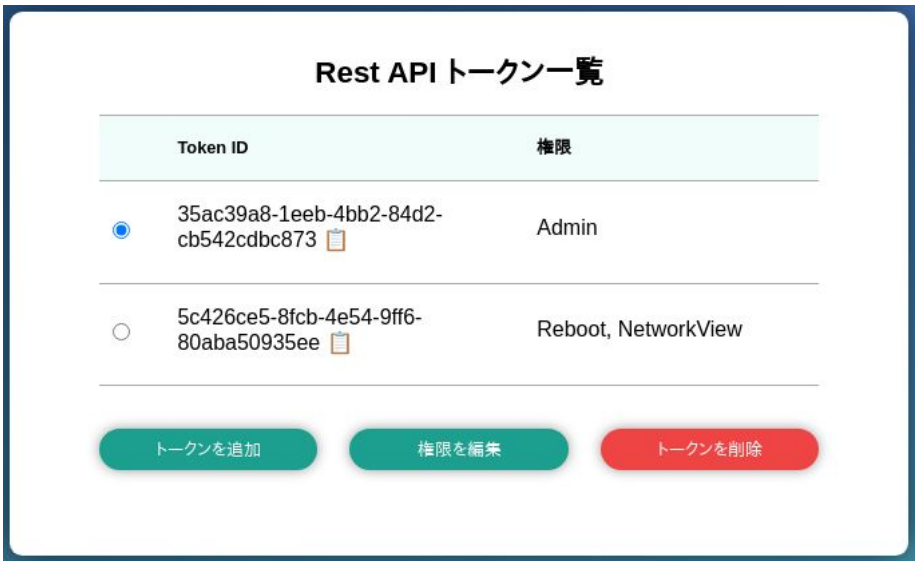


図 6.100 設定管理の Rest API トークン一覧表示



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 でのパラメータ）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"]}
```

- 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$txrw8vLJqEVUreQnZhM0CYMQ5U5B9b58L0mpVRULDiVCh2O46GKscq/
xsDPskjxg.x8ym0ri1/8NqFBu..IZE0
[ATDE ~]$ 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: SwuInstall
  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: SwuInstall
  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"}
{"exit_code":0}

http code: 200
```

6.9.6.6. Rest API : コンテナ操作

- コンテナー覧
  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** (すでに起動されている場合は再起動されます)
  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":"","con
nection.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.ra-
timeout":"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), no_connect (デフォルト false)
  apn 以外は任意です。
  no_connect を true にすると、設定のみを行い接続は行いません。
  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":"XXXXXXXXXXXXXX"}
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/wwan/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/wwan/sms

http code: 200
```

- **LTE モジュールの再起動**
  POST "/api/wwan/force_restart"
  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/wwan/force_restart

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":"1 ⏎
98.51.100.20","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_po ⏎
rt":"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 de
stination=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_po
rt":"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=mypass
http code: 200
```

図 6.101 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 6.102 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
```

- **スリープ (サスペンド)**
  POST "/api/sleep"
  必要権限: Sleep
  パラメータ: 無し
  出力: 無し (リクエストは起床時にリターンします)

```
[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/sleep

http code: 200
```

- **スリープ禁止管理**
  POST "/api/inhibit_sleep"
  必要権限: Sleep
  パラメータ: inhibit: true か空白で禁止し、false で許可します。禁止中に sleep 命令が入った場合は許可した直後に実行されます。禁止命令を複数実行した場合は同じ数の許可命令で再びサスペンド可能になります。
  出力: 無し

```
[ATDE ~]$ curl_rest -d inhibit=true https://armadillo.local:58080/api/inhibit_sleep

http code: 200
```



inhibit_sleep ページに atmark-power-utils サービスが必要なため、Armadillo-610 でエラーします。今後のアップデートで対応する予定です。

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 「3.10. 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 6.103. 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 6.103 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 6.104. 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 6.104 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。

6.9.11. ABOS Web のセキュリティ対策

ABOS Web は開発時には便利ですが、運用時にはできることの多さ故に外部からの攻撃面になり得ます。ここでは、 ABOS Web が具備しているセキュリティ対策機能について紹介します。

6.9.11.1. 同一 LAN 以外からのアクセス禁止

「6.9.1. ABOS Web ではできないこと」でも紹介している通り、 ABOS Web は同一 LAN 内からのみ接続でき、それ以外の接続は拒否します。これによって ABOS Web はインターネット側からの攻撃面となり得ず、攻撃者は同一の LAN 内に侵入してから ABOS Web にアクセスする必要があります。

6.9.11.2. コンテナ以外のアクセスを制限

ABOS Web に設定を追加することで、コンテナからのみ REST API を使用することが可能です。

この設定を行うためには /etc/atmark/abos_web/init.conf を作成し、以下の内容を記載します。

```
command_args="--listen-addr 10.88.0.1:58080 --free-bind"
```

図 6.105 コンテナからのアクセスのみを許可する設定

作成後、ファイルを永続化して Armadillo を再起動します。

コマンド引数の意味は以下の通りです：

オプション	説明
--listen-addr 10.88.0.1:58080	コンテナのデフォルトネットワークでは 10.88.0.0/24 を利用しています。こちらのアドレスにバインドすると、コンテナ以外からアクセスできなくなります。
--free-bind	abos-web 起動時に podman0 インタフェースが存在しなくても正しく起動するためのオプションです。ABOS 3.21.3-at.14 以降が必要です。

6.9.11.3. ABOS Web のユーザ認証

ABOS Web はユーザ認証としてパスワードによる認証を行います。初回起動時には必ずパスワードの設定を求められます。このとき設定するパスワードは 8 文字以上のものに強制しています。

また、ユーザ認証のブルートフォースアタック対策として、パスワードを間違える等でユーザ認証に失敗した場合に、次回再認証までの 2 秒間はパスワードの入力を受け付けない実装になっています。これによって、機械的に総当りでパスワードを解析するまでの時間が大幅に増大することが見込めます。

6.9.11.4. 通信の暗号化

ABOS Web の通信は TLS によって暗号化されています。使用する TLS 証明書は ECDSA (secp384r1) の暗号技術を用いて生成されています。

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 6.106. 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。

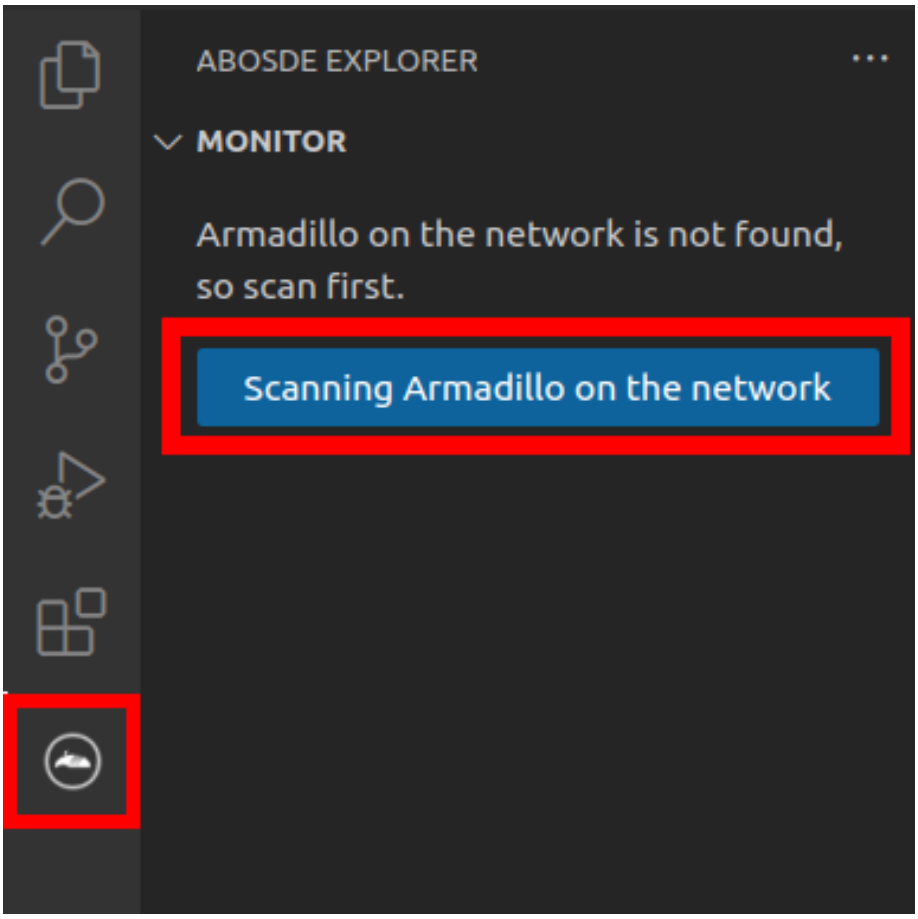


図 6.106 ABOSDE で ローカルネットワーク上の Armadillo をスキャンする

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 6.107. 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。

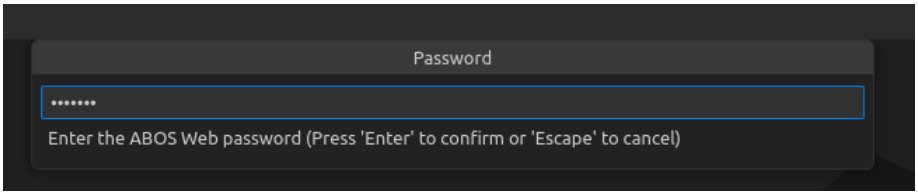


図 6.107 ABOSDE の ABOS Web パスワード入力画面

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 6.108. 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。

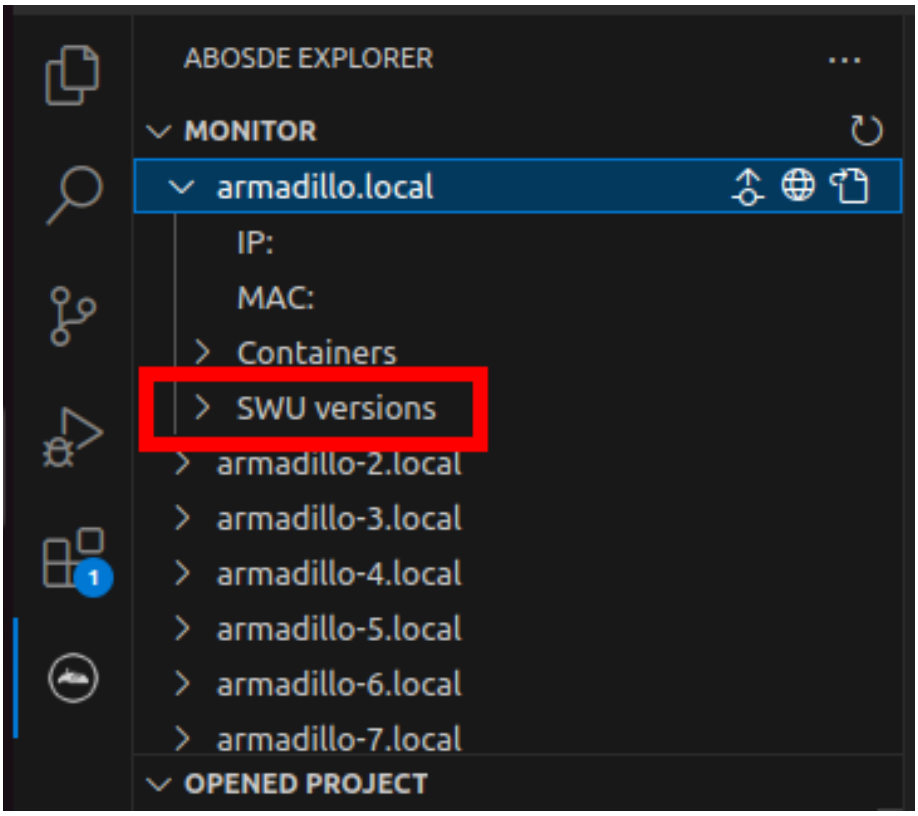


図 6.108 ABOSDE で Armadillo の SWU バージョンを取得

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 6.109. 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

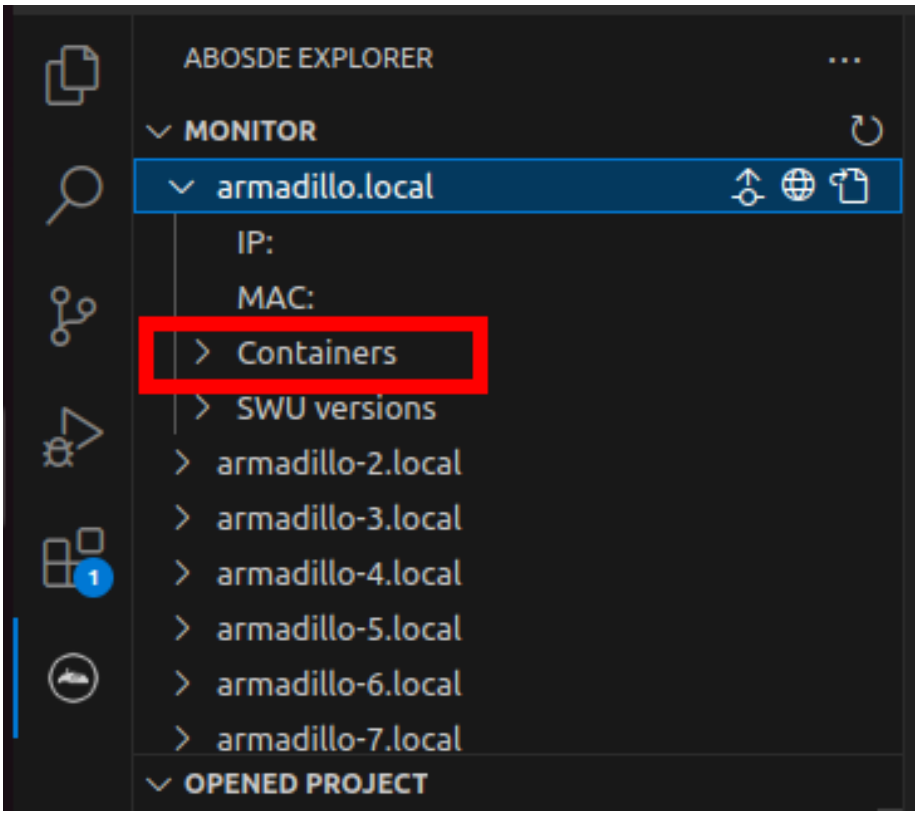


図 6.109 ABOSDE で Armadillo のコンテナ情報を取得

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 6.110. 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 6.111. 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。

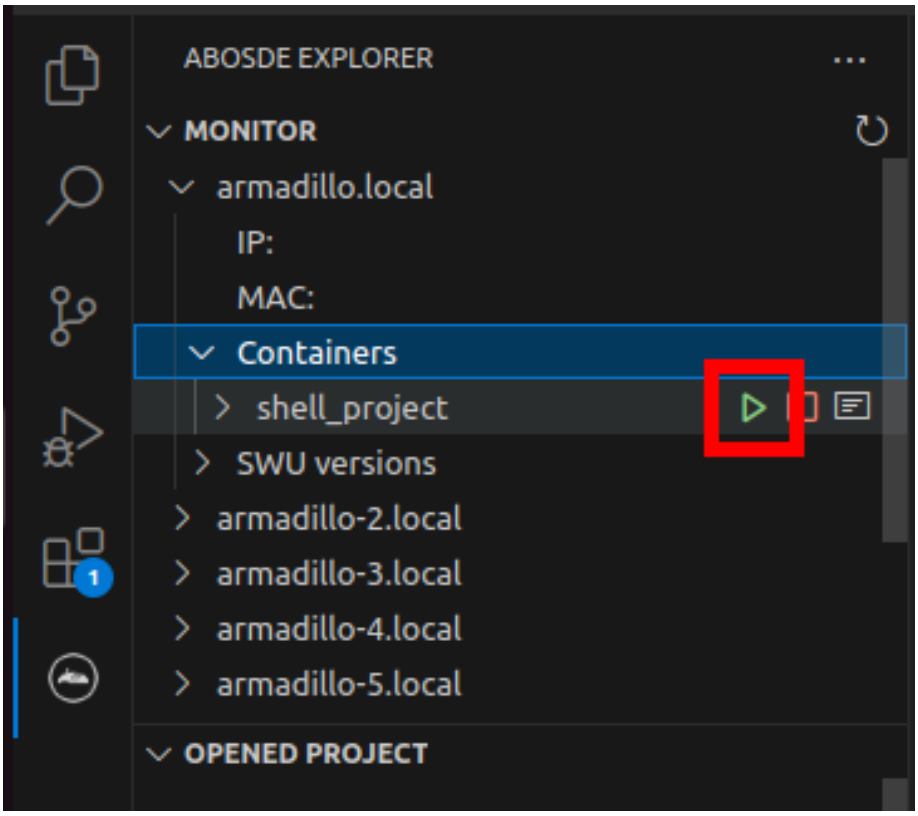


図 6.110 ABOSDE で Armadillo のコンテナを起動

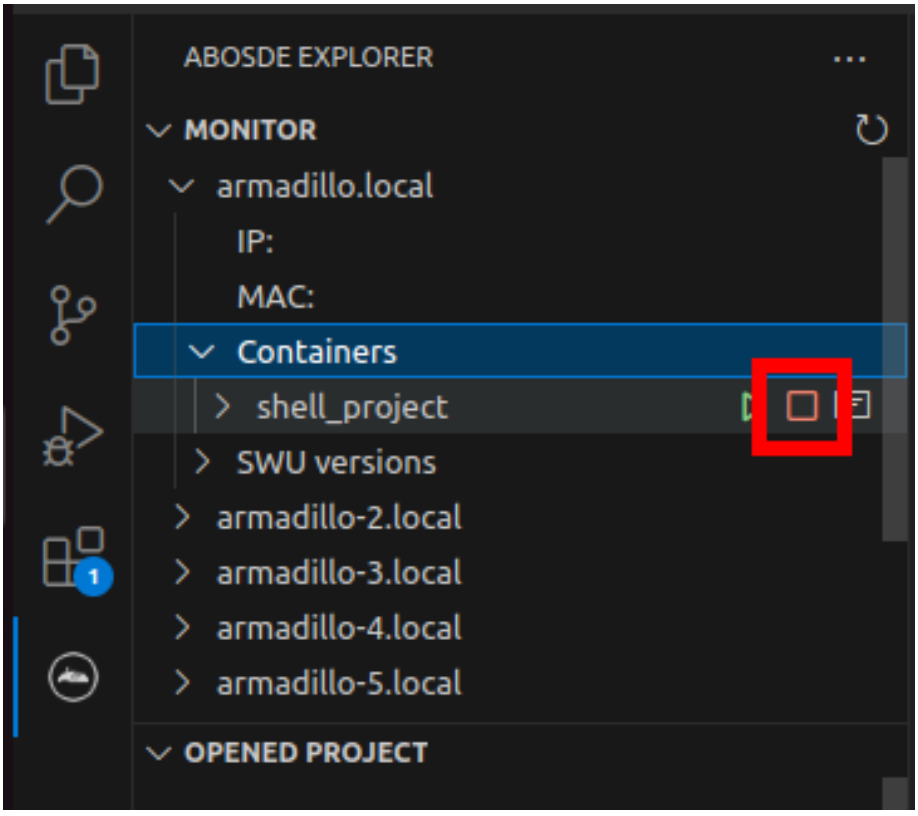


図 6.111 ABOSDE で Armadillo のコンテナを停止

6.10.4. Armadillo のコンテナのログを取得する

「図 6.112. 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。

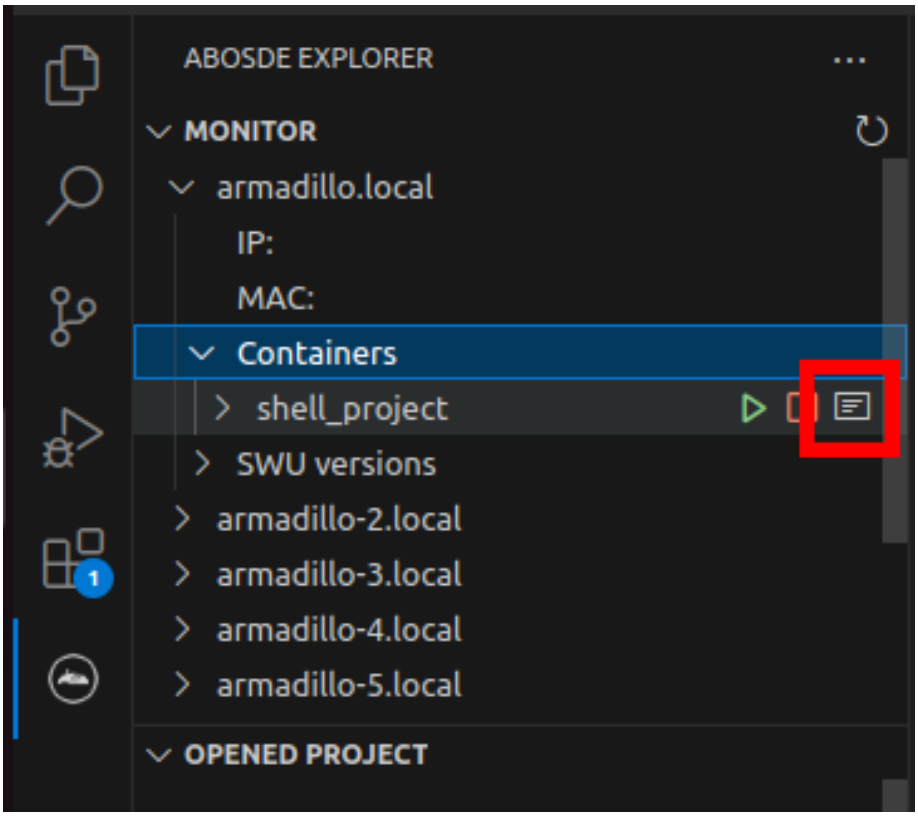


図 6.112 ABOSDE で Armadillo のコンテナのログを取得

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 6.113. 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。

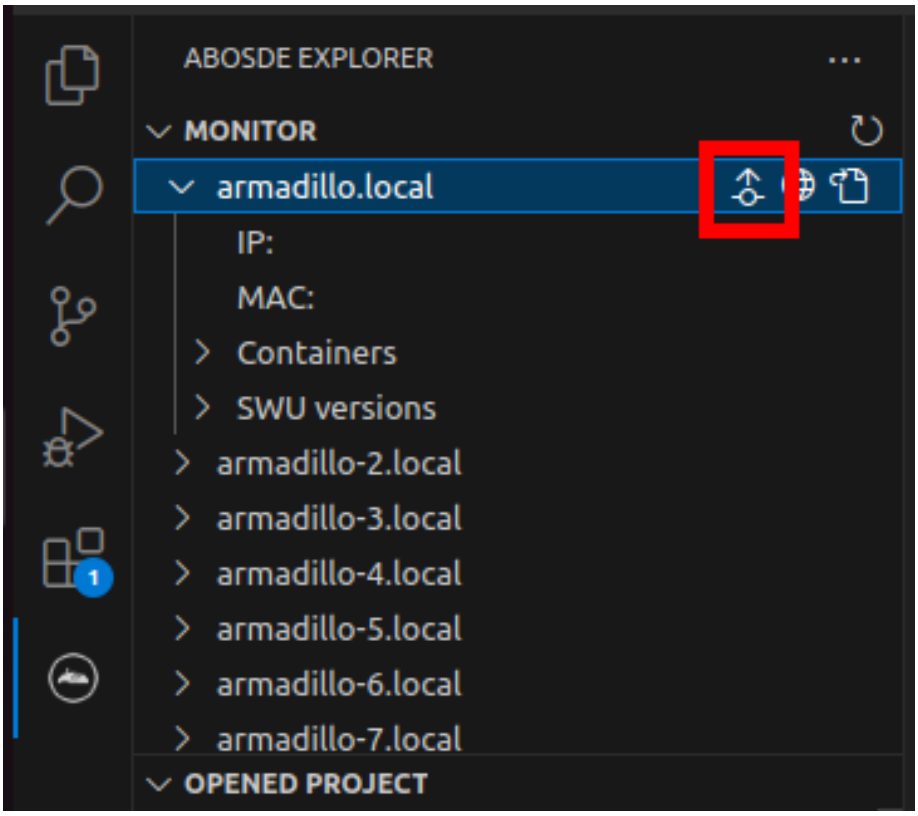


図 6.113 ABOSDE で Armadillo に SWU をインストール

6.11. ssh 経由で Armadillo Base OS にアクセスする

Armadillo-610 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-610 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-610 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-610 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 6.114 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、以下のようにします。

```
[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 6.115 IP アドレス(eth0)の確認

6.12.3. ネットワークの設定方法

Armadillo-610 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 6.116. 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 6.116 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 6.117 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 6.118 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 6.119 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 6.120 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-610 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 6.121 コネクションファイルの永続化



別の Armadillo-610 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込します。

```
[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 6.122 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 6.123 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 6.124. 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 6.124 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 6.125. 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 6.125 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 6.126. DHCP の設定」に示します。

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

図 6.126 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 6.127 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 6.128 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 6.129 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 devicegiven" 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 6.130 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 6.131 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。



デフォルトでは iptables サービスは無効になっています。サービスを有効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 iptables ❶
[armadillo ~]# persist_file /etc/runlevels/default/iptables ❷
```

❶ サービスを有効にします。

❷ サービスの有効を永続化します。

「図 6.132. 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 6.132 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 6.132. 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 6.133. 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 ~]# 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 6.133 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 6.134 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 6.135 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 6.136 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 6.137. 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 6.137 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 6.138 EXT4 ファイルシステムの構築

6.14. ボタンやキーを扱う

buttond 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/buttond.conf に BUTTOND_ARGS を指定することで、動作を指定することができます:

- --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応じた "command" を実行します。(例：buttond --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 buttond 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 buttond 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/buttond.conf ❶
BUTTOND_ARGS="$BUTTOND_ARGS --short prog1 --action 'date >> /tmp/shortpress'"
```

```
BUTTOND_ARGS="$BUTTOND_ARGS --long prog1 --time 5000 --action 'date >> /tmp/longpress'"
[armadillo ~]# persist_file /etc/atmark/buttond.conf ❷
[armadillo ~]# rc-service buttond restart ❸
buttond          | * Stopping button watching daemon ...                                [ ok ]
buttond          | * Starting button watching daemon ...                                [ ok ]
[armadillo ~]# cat /tmp/shortpress ❹
Tue Mar 22 17:16:42 JST 2022
Tue Mar 22 17:16:43 JST 2022
[armadillo ~]# cat /tmp/longpress
Tue Mar 22 17:16:48 JST 2022
```

図 6.139 buttond で SW1 を扱う

❶ buttond の設定ファイルを編集します。この例では、短押しの場合 /tmp/shotpress に、5 秒以上の長押しの場合 /tmp/longpress に日付を出力します。

❷ 設定ファイルを保存します。

❸ buttond サービスを再起動させます。ここでは再起動後短押しを 2 回、長押しを 1 回行ったとします。

❹ 押された回数を確認します。



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-*/* ❶
Skipping directory /dev/input/by-id
Skipping directory /dev/input/by-path
[78972.042] /dev/input/event2 4 4 458976: non-keyboard event ignored
[78972.042] /dev/input/event2 LEFTCTRL (29) pressed: ignored ❷
[78972.042] /dev/input/by-id/usb-0566_3029-event-kbd 4 4 458976: non-keyboard event ignored
[78972.042] /dev/input/by-id/usb-0566_3029-event-kbd LEFTCTRL (29) pressed: ignored
[78972.042] /dev/input/by-path/platform-xhci-hcd.1.auto-usb-0:1:1.0-event-kbd 4 4 458976: ⏎
non-keyboard event ignored
[78972.042] /dev/input/by-path/platform-xhci-hcd.1.auto-usb-0:1:1.0-event-kbd LEFTCTRL ⏎
(29) pressed: ignored
[78972.130] /dev/input/event2 4 4 458976: non-keyboard event ignored
[78972.130] /dev/input/event2 LEFTCTRL (29) released: ignored
[78972.130] /dev/input/by-id/usb-0566_3029-event-kbd 4 4 458976: non-keyboard event ignored
[78972.130] /dev/input/by-id/usb-0566_3029-event-kbd LEFTCTRL (29) released: ignored
[78972.130] /dev/input/by-path/platform-xhci-hcd.1.auto-usb-0:1:1.0-event-kbd 4 4 458976: ⏎
non-keyboard event ignored
```

```
[78972.130] /dev/input/by-path/platform-xhci-hcd.1.auto-usb-0:1:1.0-event-kbd LEFTCTRL
(29) released: ignored
```

図 6.140 buttond で USB キーボードのイベントを確認する

❶ 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。

❷ 希望のキーを押すと、LEFTCTRL が三つのパスで認識されました。 一番安定する by-id のパスを控えておきます。

2. USB デバイスを外すこともありますので、-i (inotify) で管理されてる入力デバイスとして追加します。そうしないとデバイスを外したときに 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 6.141 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 6.142 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. 温度測定の重要性

Armadillo は製品ごとに動作温度範囲が設定されていますが、それらはあ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 6.143 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 6.144. 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 6.144 atmark-thermal-profiler を実行する

「図 6.145. 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 6.145 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 -O jitter:buffer_size: ⏎
4133 -O jitter:refill_thresh:4133 -O 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 6.146 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℃です。
SOC_TEMP	計測時点の SoC 温度を示します。単位は℃です。製品よっては非対応で、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℃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 6.147 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℃で表記されているので、105℃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℃で表記されているので、105℃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 6.148. 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。

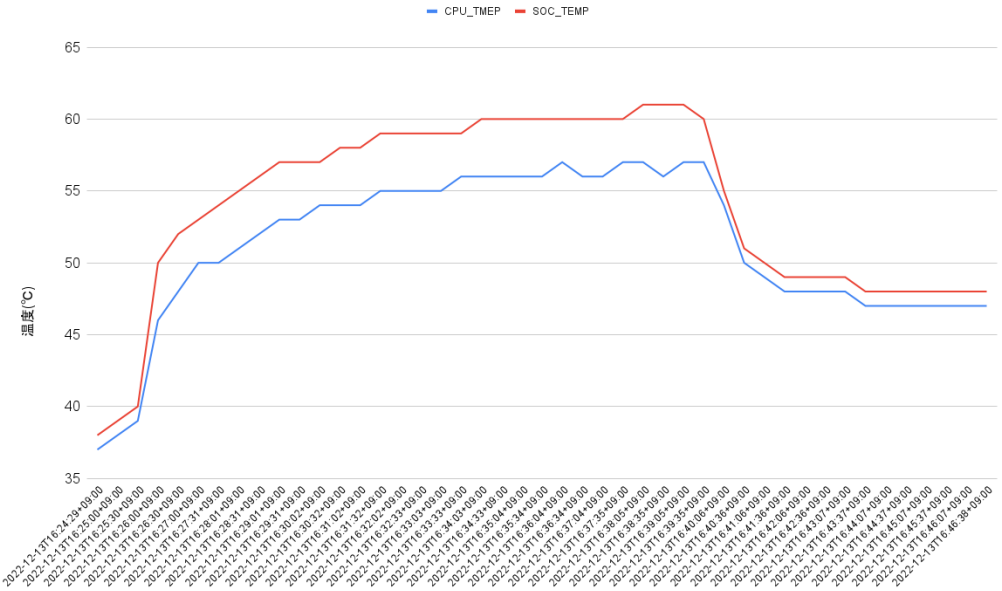


図 6.148 Armadillo-IoT ゲートウェイ G4 で取得した温度のグラフ

グラフの縦軸は温度(℃)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℃ 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しない処理が行なわれていないかなどを確認することをおすすめします。

6.15.6. Armadillo Twin から Armadillo の温度を確認する

atmark-thermal-profiler の他に、Armadillo Twin からも温度や 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-610 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/mmcblk0p1
```

```
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 6.149 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 6.150 /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 6.151 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 ❹
bootdelay=-2
```

図 6.152 uboot_env.d のコンフィグファイルの例

❶ コンフィグファイルを生成します。

❷ ファイルを永続化します。

❸ 変数を書き込みます。

❹ 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

環境変数	説明	デフォルト値
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_available	1 以上の場合は 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
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。 ・ -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/uImage
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。 ・ 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

6.20. 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. ブートディスクイメージの書き込み

   ブートディスクイメージの書き込みは「3.1.4.1. 初期化インストールディスクの作成」を参照してください。参照先では初期化インストールディスクの場合の手順を示していますが、ここでビルドしたイメージについても同じ手順になります。



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/mmcblk1: 15319040 sectors, 7.3 GiB
Sector size (logical/physical): 512/512 bytes
Disk identifier (GUID): 309AD967-470D-4FB2-835E-7963578102A4
Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
First usable sector is 34, last usable sector is 15319006
Partitions will be aligned on 2048-sector boundaries
Total free space is 20446 sectors (10.0 MiB)

Number  Start (sector)    End (sector)  Size       Code  Name
   1           20480          634879   300.0 MiB   8300  rootfs_0
   2          634880         1249279   300.0 MiB   8300  rootfs_1
   3         1249280         1351679   50.0 MiB    8300  logs
   4         1351680         1761279   200.0 MiB   8300  firm
   5         1761280        60485632   28.0 GiB    8300  app
```

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-610 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 をジャンパでショートします。

2. 電源を投入します。

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

CPU:   i.MX6ULL rev1.1 at 396 MHz
Model: Atmark Techno Armadillo-610
DRAM:  512 MiB
setup_rtc_disarm_alarm: Can't find bus
WDT:   Started with servicing (10s timeout)
PMIC: PFUZE3000 DEV_ID=0x30 REV_ID=0x11
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
Loading Environment from MMC... OK
In:    mxc_serial
Out:   mxc_serial
Err:   mxc_serial
switch to partitions #0, OK
mmc1 is current device
flash target is MMC:1
Net:   eth0: ethernet@2188000
Fastboot: Normal
Saving Environment to MMC... Writing to MMC(1)... OK
Normal Boot
Hit any key to stop autoboot:  0
switch to partitions #0, OK
mmc1 is current device
Cannot lookup file boot/boot.scr
6859976 bytes read in 1420 ms (4.6 MiB/s)
Booting from mmc ...
```

```
37363 bytes read in 93 ms (391.6 KiB/s)
Loading fdt boot/armadillo.dtb
Cannot lookup file boot/overlays.txt
## Booting kernel from Legacy Image at 80800000 ...
   Image Name:   Linux-5.10.180-2-at
   Created:      2023-06-09   9:48:24 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    6859912 Bytes = 6.5 MiB
   Load Address: 82000000
   Entry Point:  82000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 83500000
   Booting using the fdt blob at 0x83500000
   Loading Kernel Image
   Loading Device Tree to 9ef1d000, end 9ef49fff ... OK

Starting kernel ...

...中略...

Welcome to Alpine Linux 3.17
Kernel 5.10.180-2-at on an armv7l (/dev/ttymxc0)

armadillo login:
```

6.21. Armadillo のソフトウェアをビルドする

ここでは、Armadillo-610 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-610 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-610 ブートローダー [https://armadillo.atmark-techno.com/resources/software/armadillo-610/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 ~]$ wgets https://armadillo.atmark-techno.com/files/downloads/armadillo-610/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 6.153. 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 6.153 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-610 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-610 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-610 Linux カーネル [https://armadillo.atmark-techno.com/resources/software/armadillo-610/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-at-a6-[VERSION]/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  --->                                     |  |
    |  |          Library routines  --->                                      |  |
    |  |          Kernel hacking  --->                                        |  |
    |  |                                                                      |  |
    |  +----------------------------------------------------------------------+  |
    |                                                                            |
    +----------------------------------------------------------------------------+
    |              <Select>    < Exit >    < Help >    < Save >    < Load >      |
    +----------------------------------------------------------------------------+
```



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 6.154 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 6.155. 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 を展開した後に以下の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 6.155 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無くなります。

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-610 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-610 開発用ツール [https://armadillo.atmark-techno.com/resources/software/armadillo-610/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-610/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. 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. 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ーにコピーされます。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-610 開発用ツール [https://armadillo.atmark-techno.com/resources/software/armadillo-610/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 「6.27.2.3. 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。



アットマークテクノがリリースするファームウェアはバージョンのサフィックスとして"-at.[数字]"を含めています。オリジナルのサフィックスをつける場合は、"-at.[数字]"を使用しないことを強く推奨します。

6.22. 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。

6.22.1. 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 mksquashfs コマンドを使用して squashfs イメージを作成します。

```
[ATDE]$ mkdir sample
[ATDE]$ echo ”complete mounting squashfs on eMMC(GPP)” > sample/README
[ATDE]$ mksquashfs sample squashfs.img
```

図 6.156 squashfs イメージの作成

6.22.2. squashfs イメージを書き込む

以降の作業は Armadillo 上で行います。

「6.22.1. 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-610 にコピーし、GPP に書き込みます。



ユーザー領域として使用可能な GPP は /dev/mmcblk0gp3 のみです。

GPP への書き込みを行う際は、誤って /dev/mmcblk0gp0 などに書き込みを行わないよう、十分に注意してください。

```
[armadillo]# mount /dev/sda1 /mnt
[armadillo]# dd if=/mnt/squashfs.img of=/dev/mmcblk0gp3 conv=fsync
[armadillo]# umount /mnt
```

6.22.3. 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 6.157 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 6.158 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
```

6.22.4. 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)
```

6.23. 動作ログ

6.23.1. 動作ログについて

Armadillo-610 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-610 で /var/log 配下に出力するログに関しては 「6.23.5. /var/log/ 配下のログに関して」を参照ください。

6.23.2. 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。

6.23.3. 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 6.159 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

6.23.4. 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。

6.23.5. /var/log/ 配下のログに関して

「表 6.16. /var/log/ 配下のログ」 に Armadillo-610 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 6.16. /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 6.16. /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 6.16 /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
/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

6.23.6. ABOS Web でログを確認する

ログは ABOS Web の「設定管理」ページにある「ログ情報」でも確認およびダウンロードすることができます。



- **ログを表示する**

  /var/log/messages の最新の 200 行を表示します。

- **ログをダウンロードする**

  /var/log ディレクトリと /var/at-log ディレクトリをまとめて圧縮したファイルをダウンロードできます。より詳細なログを確認したい場合は、こちらを使用してください。

6.24. ATDE・Linux でインストールディスクを作成する

ATDE や Linux でインストールディスクイメージからインストールディスクを作成する方法を紹介します。(Windows でインストールディスクを作成する方法については「3.1.4.1. 初期化インストールディスクの作成」を参照してください。参照先は初期化インストールディスクイメージを使用した方法ですが、それ以外のインストールディスクイメージでも同様の手順で行うことができます。)

ここでは、例として ATDE で初期化インストールディスクを作成する 2 種類の方法（CUI・GUI）を記載しています。初期化インストールディスクイメージ以外のインストールディスクイメージでも同様の手順で行うことができます。

いずれの方法でも、まずは次の共通の手順を実施してください。

1. 1 GB 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ダウンロードします。Armadillo-610 インストールディスクイメージ [https://armadillo.atmark-techno.com/resources/software/armadillo-610/abos-disc-image] から「Armadillo Base OS インストールディスクイメージ」をダウンロードしてください。

6.24.1. GUI でインストールディスクを作成する

1. ダウンロードした zip ファイルを展開します。図のとおり、zip ファイルを選択して右クリックし、「ここで展開」をしてください。

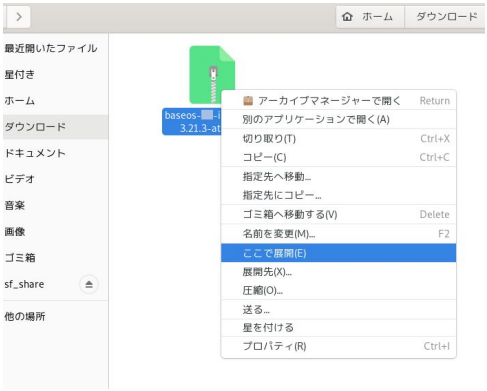


図 6.160 zip ファイルを展開

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. 展開したフォルダ内にあるインストールディスクイメージ（.img）をダブルクリックして、「ディスクイメージをリストア」のウィンドウを表示します。



図 6.161 展開したフォルダ内にある img ファイルをダブルクリック



図 6.162 ディスクイメージをリストア

4. microSD カードを指定します。[転送先]から接続した microSD カードに対応するものを選びます。



[転送先]は細心の注意を払って選んでください。[転送先]には ATDE のシステムディスク（右側に VBOX HARDDISK や /dev/atde9-vg* と記載されているもの）も表示されていますが、**これらは決して選択しないでください。**誤って選択して書き込んでしまった場合、ATDE が破壊される可能性があります。

また、他に接続されているデバイスに意図せず書き込んでしまった場合、そのデバイスのデータが上書きされてしまいます。選んだ[転送先]が目的の microSD カードなのかどうか、しっかりと確認してください。microSD カードやカードリーダーの挿抜による[転送先]の表示の変化で確かめることもできます。



図 6.163 microSD カードを指定

5. [リストアを開始]ボタンをクリックして、書き込みを開始します。
6. 確認のウィンドウが現れるので、今一度確認した上で[リストア]をクリックします。



図 6.164 確認のウィンドウ

7. パスワードが要求されるので入力します。



図 6.165 パスワードの要求

8. 書き込みが完了したら、 microSD カードを取り外します。

6.24.2. CUI でインストールディスクを作成する

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,iocharset
=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、以下の dd コマンドで img ファイルを microSD カードに書き込んでください。

```
[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
```

5. 書き込みが完了したら、 microSD カードを取り外します。

6.25. シリアル通信ソフトウェア(minicom)

Linux や ATDE で使用できるシリアル通信ソフトウェアとして minicom の使用方法を紹介します。

6.25.1. 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 6.17. シリアル通信設定」 のとおりに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 6.17 シリアル通信設定

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

1. 「図 6.166. 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 6.166 minicom の設定の起動

2. 「図 6.167. 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 6.167 minicom の設定

3. 「図 6.168. 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 6.168 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 6.169 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 6.170. 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 6.170 minicom のシリアルポートのパラメータの設定

8. 「図 6.170. 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 6.171. minicom シリアルポートの設定値」を参照してください。

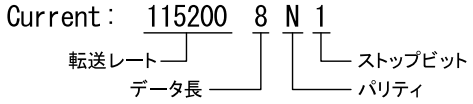


図 6.171 minicom シリアルポートの設定値

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 6.167. minicom の設定」に戻ってください。

13.「図 6.167. 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。

6.25.2. minicom の起動

ATDE の場合は、minicom を起動する前に 「表 6.18. シリアルコンソールとして使用する取り外し可能デバイス」に示すデバイスを ATDE に接続してください。

表 6.18 シリアルコンソールとして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 6.172. 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 6.172 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
```

6.25.3. 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 6.173 minicom 終了確認

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 6.174 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起動します。

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 6.175. 入力モードに移行するコマンドの説明」に示します。

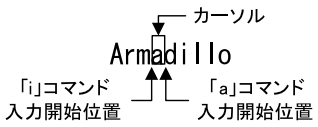


図 6.175 入力モードに移行するコマンドの説明

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 6.176. 文字を削除するコマンドの説明」に示します。

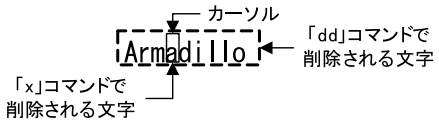


図 6.176 文字を削除するコマンドの説明

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. セキュリティ

近年、組み込みデバイスの増加に伴い、ハッキングによる脅威も増加の一途を辿っています。サイバー攻撃による被害はそのデバイスの使用者はもちろんですが、信用の失墜や損害賠償といった開発元の企業が被る損害も計り知れません。そのようなリスクを最小限に抑えるために、開発及び運用時にセキュリティを意識することは重要です。

6.27.1. 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。

6.27.2. 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 「6.27.2.3. 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 「6.27.2.4. SWU イメージと同時に SBOM を作成する」 を参照ください。

6.27.2.1. 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。

6.27.2.2. 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 を用いた SBOM のスキャンについては「3.18. 生成した SBOM をスキャンする」をご参照ください。

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。

6.27.2.3. 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-610/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。

6.27.2.4. 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。

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。

「6.27.2.4. SWU イメージと同時に SBOM を作成する」 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。

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 6.23 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	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
	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	
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 6.177 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。

6.27.3. 不正な USB デバイスの接続を拒否する



USB 接続制御機能は、 ABOS バージョン 3.20.3-at.8 以降で対応しています。



この機能に関して、ABOS バージョン 3.21.3-at.12 以降であれば ABOS Web で設定することも可能です。

「3.9. USB デバイスの接続を許可する」をご参照ください。

```
[armadillo ~]# abos-ctrl usb-filter help
Usage: abos-ctrl usb-filter [action [arguments]]

Possible actions:
  enable: enabling the USB filter function
  disable: disabling the USB filter
  list-devices [--verbose]: list currently connected USB devices
  list-rules [--verbose]: list currently USB device allowlist
  reset-rules [--force]: reset USB device allowlist
                         with --force option, do not prompt
  allow-device [--vendor-id VID] [--product-id PID]
               [--model MODEL] [--usb-interfaces INTERFACES]
               [--serial SERIAL]: add device to the allowlist
                                  Omitted parameters are stored as wildcard
                                  parameters can be checked with list-devices and
                                  list-rules --verbose flag
  allow-device {DEVID}: allow connected USB devices specified by ID to connect,
                        and add device's information to the allowlist
                        ID can be checked with `abos-ctrl usb-filter list-devices`
  block-device {DEVID}: block alllowed USB devices specified by ID to connect,
                        and delete device's information from the allowlist
                        ID can be checked with `abos-ctrl usb-filter list-devices`
  allow-class [CLASS]...: create allow rule for each USB device class
                          the string for 'CLASS' can be found by running
                          `abos-ctrl usb-filter allow-class`
  remove-rule {RULEID}...: remove the allow rule specified by ID
                           ID can be checked with `abos-ctrl usb-filter list-rules`
  help: show this message
```

図 6.178 USB 接続制御機能を管理するコマンド

SWUpdate から USB 接続制御機能の設定を変更する手順を Armadillo サイトの Howto で公開していますので、必要な場合はご参照ください。

Armadillo サイト Howto 「SWUpdate を用いて USB 接続制御機能の設定を行う」

https://armadillo.atmark-techno.com/howto/setting-usb-filter-with-swupdate

6.27.3.1. USB 接続制御機能を有効/無効化する

現在 USB 接続制御機能が有効か無効かは「図 6.179. USB 接続制御機能の状態を確認する」に示すコマンドを実行することで確認できます。

```
[armadillo ~]# abos-ctrl usb-filter
Currently USB filter is disabled.
```

図 6.179 USB 接続制御機能の状態を確認する

「図 6.180. USB 接続制御機能を有効化する」に示すコマンドを実行することで、 USB 接続制御機能を有効化できます。

```
[armadillo ~]# abos-ctrl usb-filter enable
USB filter enabled.
please reboot to apply rules to currently connected devices
```

図 6.180 USB 接続制御機能を有効化する

有効化したあとに接続された USB デバイスは、設定した許可ルールにしたがって許可/拒否されます。デフォルトでは全てのデバイスは拒否されます。

「図 6.181. USB 接続制御機能を無効化する」に示すコマンドを実行することで、 USB 接続制御機能を無効化できます。

```
[armadillo ~]# abos-ctrl usb-filter disable
USB filter disabled.
please reboot to apply rules to currently connected devices
```

図 6.181 USB 接続制御機能を無効化する

6.27.3.2. 接続済みの USB デバイスの一覧を表示する

「図 6.182. 接続されている USB デバイスをリストする」に示すコマンドを実行することで、 Armadillo に接続されている全ての USB デバイスのリストを表示します。

```
[armadillo ~]# abos-ctrl usb-filter list-devices
     1  block "001:003" "046d" "08e5" "HD_Pro_Webcam_C920" ":0e0100:0e0200:010100:010200:" ⏎
"046d_HD_Pro_Webcam_C920" "/devices/platform/soc@0/32f10100.usb/38100000.dwc3/xhci-hcd.1.auto/ ⏎
usb1/1-1"
```

図 6.182 接続されている USB デバイスをリストする

1 行につき 1 つのデバイスの情報をスペース区切りで示しています。各列が何を示しているかは「表 6.24. デバイスリストの各列の意味」を参照してください。

表 6.24 デバイスリストの各列の意味

1 列目	そのデバイスに割り当たっている ID です。USB デバイスを許可/拒否する際に識別子として使用されます
2 列目	そのデバイスが現在許可(allow)されているか。拒否(block)されているかを示します
3 列目	そのデバイスに割り当たっている USB バス番号とデバイス番号のペアです
4 列目	そのデバイスのベンダー ID です
5 列目	そのデバイスのモデル ID です
6 列目	そのデバイスのモデル名です

7 列目	そのデバイスの USB クラスコードです。デバイスによっては複数存在するものもあり、":"(コロン)区切りで表示されます
8 列目	そのデバイスのシリアル番号です。一般に同じ製品であっても個体ごとに一意な値です
9 列目	そのデバイスに割り当たっている/sys 以下のディレクトリパスです

6.27.3.3. USB デバイスの接続を許可する

「図 6.182. 接続されている USB デバイスをリストする」のコマンドを実行して、2 列目が「block」となっているデバイスは拒否されているデバイスです。このデバイスに対して、「図 6.183. 指定した USB デバイスを許可する」に示すコマンドを実行することで接続を許可し、今後再接続されたとしても許可します。

```
[armadillo ~]# abos-ctrl usb-filter allow-device 1 ❶
allowed the following device:
    "001:003" "046d" "08e5" "HD_Pro_Webcam_C920" ":0e0100:0e0200:010100:010200:"
"046d_HD_Pro_Webcam_C920" "/devices/platform/soc@0/32f10100.usb/38100000.dwc3/xhci-hcd.1.auto/
usb1/1-1" ❷
```

図 6.183 指定した USB デバイスを許可する

❶ この例では、「図 6.182. 接続されている USB デバイスをリストする」のコマンドを実行して ID が 1 のデバイスを許可しています

❷ 許可されたデバイスの情報が表示されます

また、ベンダー/モデル ID やシリアル番号などの情報を前もって知っている場合は、それらの情報を用いて USB デバイスの接続を許可することも可能です。

```
[armadillo ~]# abos-ctrl usb-filter allow-device ¥
--vendor-id "046d" ¥
--product-id "08e5" ¥
--model "HD_Pro_Webcam_C920" ¥
--usb-interfaces ":0e0100:0e0200:010100:010200:" ¥
--serial "046d_HD_Pro_Webcam_C920"
allowed the following device:
    "046d" "08e5" "HD_Pro_Webcam_C920" ":0e0100:0e0200:010100:010200:" "046d_HD_Pro_Webcam_C920"
```

図 6.184 パラメータで指定した USB デバイスを許可する

「図 6.184. パラメータで指定した USB デバイスを許可する」において、1 つ以上のパラメータが指定されていない場合、その指定しなかったパラメータはワイルドカードとして扱われます。この場合、デバイスを接続したときに、指定したパラメータさえ完全に一致していれば、ワイルドカードとして扱われているパラメータが何であってもデバイスの接続が許可されます。



Armadillo に接続されている USB デバイスであれば 「図 6.185. Armadillo に接続している USB デバイスのパラメータを調べる」に示すコマンドを実行することで、「図 6.184. パラメータで指定した USB デバイスを許可する」に指定すべきパラメータを調べることができます。

```
[armadillo ~]# abos-ctrl usb-filter list-devices -v
     1  block "001:003" --vendor-id "046d" --product-id "08e5" --model
```

```
"HD_Pro_Webcam_C920" --usb-interfaces ":0e0100:0e0200:010100:010200:" --
serial "046d_HD_Pro_Webcam_C920" "/devices/platform/soc@0/32f10100.usb/
38100000.dwc3/xhci-hcd.1.auto/usb1/1-1"
```

図 6.185 Armadillo に接続している USB デバイスのパラメータを調べる

6.27.3.4. USB デバイスの接続を拒否する

「図 6.182. 接続されている USB デバイスをリストする」のコマンドを実行して、2 列目が「allow」となっているデバイスは許可されているデバイスです。このデバイスに対して、「図 6.186. 指定した USB デバイスを拒否する」に示すコマンドを実行することで接続を拒否し、今後再接続されたとしても拒否します。

```
[armadillo ~]# abos-ctrl usb-filter block-device 1 ❶
blocked the following device:
    "001:003" "046d" "08e5" "HD_Pro_Webcam_C920" ":0e0100:0e0200:010100:010200:"
"046d_HD_Pro_Webcam_C920" "/devices/platform/soc@0/32f10100.usb/38100000.dwc3/xhci-hcd.1.auto/
usb1/1-1" ❷
```

図 6.186 指定した USB デバイスを拒否する

❶ この例では、「図 6.182. 接続されている USB デバイスをリストする」のコマンドを実行して ID が 1 のデバイスを拒否しています

❷ 拒否されたデバイスの情報が表示されます

6.27.3.5. USB デバイスクラス単位で USB デバイスの接続を許可する

「6.27.3.3. USB デバイスの接続を許可する」の手順では、 USB デバイスのベンダー ID やプロダクト ID、シリアル番号などが完全一致したデバイスのみを許可します。そのため同じメーカーの同じデバイスであっても、シリアル番号が一致しないと許可されません。ここでは、「USB メモリの接続はどのメーカーのどの製品であっても全て許可するが、 USB カメラなどの接続は拒否する」といったルールを手軽に作成する機能を紹介します。

abos-ctrl usb-filter allow-class コマンドを用いて、指定した USB デバイスクラスのみを許可することができます。例として、「図 6.187. 指定した USB デバイスクラスを許可する」では、 USB MassStorage クラスのデバイス(USB メモリなど)を許可します。

```
[armadillo ~]# abos-ctrl usb-filter allow-class MassStorage
```

図 6.187 指定した USB デバイスクラスを許可する

「図 6.187. 指定した USB デバイスクラスを許可する」の例では、引数として"MassStorage"を指定していますが、他にも指定できるデバイスクラスがあります。「図 6.188. 指定可能な USB デバイスクラスを確認する」に示すコマンドを実行することで、引数として指定できる文字列を確認できます。

```
[armadillo ~]# abos-ctrl usb-filter allow-class
supported USB device classes:
        Audio
        CDC
```

```
        HID
        Physical
        Image
        Printer
        MassStorage
        Hub
        CDCdata
        SmartCard
        ContentSecurity
        Video
        PersonalHealthCare
```

図 6.188 指定可能な USB デバイスクラスを確認する

各 USB デバイスクラスについては、 Defined Class Codes [https://www.usb.org/defined-class-codes] を参照してください。

1 つのデバイスで複数のデバイスクラスを持っている場合、そのデバイスの全てのデバイスクラスが許可されていなければ認識されません。例えば、「図 6.182. 接続されている USB デバイスをリストする」の 1 番目に認識されている USB カメラは Video と Audio の 2 つのデバイスクラスを持っています。このデバイスは、 Video と Audio 両方のデバイスクラスを許可している場合に認識されます。

6.27.3.6. 定義済みの USB デバイス許可ルールを表示する

「図 6.183. 指定した USB デバイスを許可する」や「図 6.186. 指定した USB デバイスを拒否する」、「図 6.187. 指定した USB デバイスクラスを許可する」を実行すると、 USB デバイスの許可ルールが更新されます。許可ルールに記載されているデバイスは、接続されたときに認識され、使用できます。

「図 6.189. 定義済みの USB デバイス許可ルールを表示する」に示すコマンドを実行することで現在の許可ルールの一覧を表示できます。

```
[armadillo ~]# abos-ctrl usb-filter list-rules
     1  class MassStorage
     2  device "046d" "08e5" "HD_Pro_Webcam_C920" ":0e0100:0e0200:010100:010200:" ⏎
"046d_HD_Pro_Webcam_C920"
```

図 6.189 定義済みの USB デバイス許可ルールを表示する

各行の最初の数値は、そのルールに割り当たった ID です。この ID は ルールを個別に削除する際に使用されます。詳細は「6.27.3.7. 定義済みの USB デバイス許可ルールを削除する」を参照してください。

2 列目の文字列は、そのルールがデバイスクラス単位の許可ルールであるか、デバイス固有の情報に基づいた許可ルールであるかを示します。
この列が class であれば、当該のルールはデバイスクラス単位の許可ルール(「6.27.3.5. USB デバイスクラス単位で USB デバイスの接続を許可する」によって追加されたルール)であり、3 列目の文字列は許可する USB デバイスクラス名です。
この列が device であれば、当該のルールはデバイス固有の情報に基づいた許可ルール(「6.27.3.3. USB デバイスの接続を許可する」によって追加されたルール)であり、以降の文字列はその条件に一致したデバイスが接続されると接続が許可されるルールです。この場合の 3 列目以降の文字列の意味を「表 6.25. 2 列目が device のときの許可ルールリストの各列の意味」に示します。

表 6.25 2 列目が device のときの許可ルールリストの各列の意味

3 列目	このルールによって許可されるベンダー ID です
4 列目	このルールによって許可されるモデル ID です
5 列目	このルールによって許可されるモデル名です
6 列目	このルールによって許可される USB クラスコードです。複数ある場合も全てが完全一致した場合のみ許可されます
7 列目	このルールによって許可されるシリアル番号です

1 つのルールにつき 1 行で表記され、接続したデバイスが全てのルールのうちどれか 1 つでも満たしていれば許可されます。

6.27.3.7. 定義済みの USB デバイス許可ルールを削除する

定義した許可ルールは削除することができます。削除することでそのデバイスは再接続時に接続が拒否され使用できなくなります。

「図 6.189. 定義済みの USB デバイス許可ルールを表示する」に示す状況のときに、"MassStorage"の許可を削除する例を「図 6.190. 定義済みの USB デバイス許可ルールを削除する」に示します。

```
[armadillo ~]# abos-ctrl usb-filter remove-rule 1
[armadillo ~]# abos-ctrl usb-filter list-rules
     1  device "046d" "08e5" "HD_Pro_Webcam_C920" ":0e0100:0e0200:010100:010200:"
"046d_HD_Pro_Webcam_C920"
```

図 6.190 定義済みの USB デバイス許可ルールを削除する

6.27.4. EdgeLock SE050 を利用したキーストレージ

NXP の EdgeLock SE050 は IoT アプリケーション向けのセキュアエレメントです。フラッシュメモリを内蔵しており、保存された秘密鍵を外部に露出することなく暗号処理に利用できます。ここでは Egelock SE050 をキーストレージとして利用する方法を説明します。



EdgeLock SE050 の詳細については以下のデータシートを参照してください。

SE050 datasheet

https://www.nxp.com/docs/en/data-sheet/SE050-DATASHEET.pdf

6.27.4.1. EdgeLock SE05x Plug & Trust Middleware

EdgeLock SE050 を利用するためのソフトウェアとして、NXP から ミドルウェア EdgeLock SE05x Plug & Trust Middleware (以下、Plug & Trust Middleware) が提供されています。ただし、ビルド済みバイナリは提供されないため利用環境に合わせてビルドの必要があります。

Armadillo Base OS では Alpine Linux と Debian GNU/Linux 向けに Plug & Trust Middleware のビルド済みパッケージを提供しています。以下は Plug & Trust Middleware 周辺のソフトウェアスタックです。提供中のビルド済みパッケージは赤い四角のブロックになります。

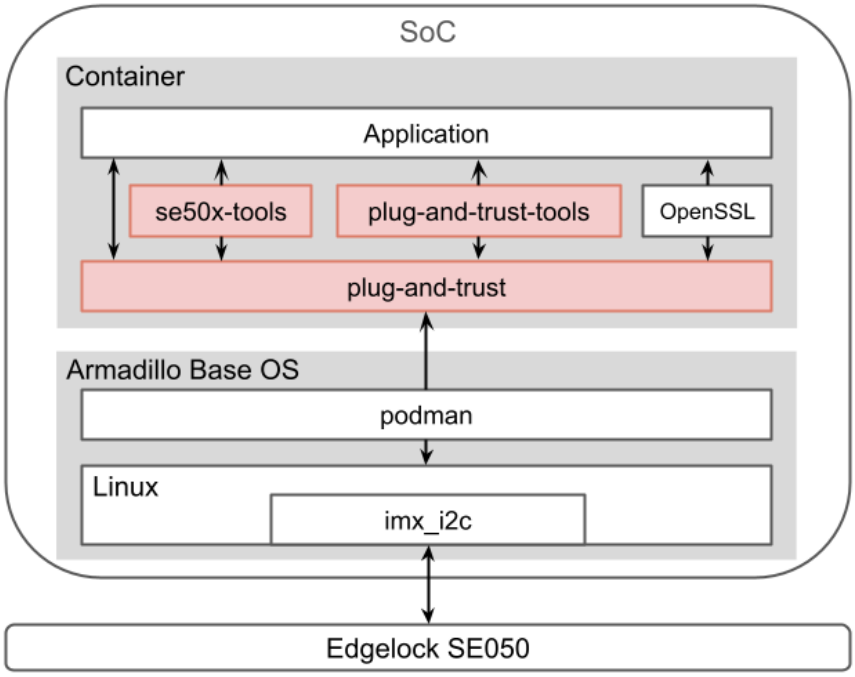


図 6.191 Plug & Trust Middleware の周辺のソフトウェアスタック

提供しているパッケージは以下のとおりです。

表 6.26 Plug & Trust Middle のパッケージ

パッケージ	内容
plug-and-trust	Plug & Trust Middleware のビルド済みライブラリ群。OpenSSL engine を含む。
plug-and-trust-dev	Plug & Trust Middleware の開発用ファイル。アプリケーション開発に利用できます。
plug-and-trust-tools	Plug & Trust Middleware に含まれるサンプルアプリケーション。最小限の動作確認ツール (se05x_Minimal) や機能や固有情報を取得するツール (se05x_GetInfo) など。
se05x-tools	非対称暗号向けの鍵の読み書きツール。

Armadillo Base OS が対応するディストリビューションとバージョンは以下のとおりです。

表 6.27 Plug & Trust Middleware のサポート

ディストリビューション	バージョン
Alpine Linux	3.21
Debian GNU/Linux	11 (bullseye)
Debian GNU/Linux	12 (bookworm)



本ガイド作成時点では利用したバージョンは以下のとおりです。

- Plug & Trust Middleware : 04.02.00
- plug-and-trust : 4.2.0

・ se05x-tools : 1.2.2

6.27.4.2. EdgeLock SE050 を有効にする

EdgeLock SE050 は ENA ピンがアサートされると有効化されます。

Armadillo-610 の EdgeLock SE050 はデフォルトで自動的に有効化され、スリープモード中は無効化されるようになっているため、設定は不要です。

6.27.4.3. Plug & Trust Middleware のインストール

Debian と Alpine Linux に対応しますが、ここでは Alpine Linux で作業を進めます。

コンテナを立ち上げるために、/etc/atmark/containers ディレクトリに、plug-and-trust.conf ファイルを以下の内容で作成します。

```
[armadillo ~]# vi /etc/atmark/containers/plug-and-trust.conf
set_image docker.io/alpine
add_armadillo_env
add_devices "${AT_SE_PARAM%:*}"
add_volumes /etc/apk:/etc/apk:ro
set_command sleep infinity
```



Debian を利用する場合は、at-debian の利用をお勧めします。at-debian はアットマークテクノによる動作確認済みの環境です。apt-get install を利用してインストールしてください。

「6.2.5. アットマークテクノが提供するイメージを使う」を参考にしてください。

ベースとなる alpine コンテナをインストールします。

```
[armadillo ~]# abos-ctrl podman-rw pull docker.io/alpine
```

コンテナを起動し、コンテナの中に入ります。

```
[armadillo ~]# podman_start plug-and-trust
[armadillo ~]# podman exec -it plug-and-trust sh
```



以下のエラーが出る場合は、Armadillo Base OS のバージョンを最新にするか、plug-and-trust.conf を書き換えて再度実行してみてください。

```
[armadillo ~]# podman_start plug-and-trust
error: plug-and-trust: device  does not exist
[armadillo ~]# vi /etc/atmark/containers/plug-and-trust.conf
```

```
set_image docker.io/alpine
export AT_SE_PARAM="$(device-info --se-param)"
add_args --env=AT_SE_PARAM
add_devices "${AT_SE_PARAM%:*}"
add_volumes /etc/apk:/etc/apk:ro
set_command sleep infinity
```

パッケージをインストールします。

```
[container /]# cd
[container ~]# apk add se05x-tools plug-and-trust-tools
```

利用のために環境変数を設定します。

```
[container ~]# export OPENSSL_CONF=/etc/plug-and-trust/openssl11_sss_se050.cnf
[container ~]# export EX_SSS_BOOT_SSS_PORT="$AT_SE_PARAM"
```

インストールと環境設定が終わったら、se05x_GetInfo で EdgeLock SE050 の動作確認を確認します。アクセスに成功すると以下のようなログが出力されます。

```
[container ~]# se05x_GetInfo
App   :INFO :PlugAndTrust_v04.02.00_20220524
App   :INFO :Running se05x_GetInfo
App   :INFO :Using PortName='/dev/i2c-0:0x48' (ENV: EX_SSS_BOOT_SSS_PORT=/dev/i2c-0:0x48)
sss   :INFO :atr (Len=35)
      00 A0 00 00    03 96 04 03    E8 00 FE 02    0B 03 E8 08
      01 00 00 00    00 64 00 00    0A 4A 43 4F    50 34 20 41
      54 50 4F
App   :WARN :No SemsLite Applet Available.
App   :INFO :Running se05x_GetInfo
App   :INFO :Using PortName='/dev/i2c-0:0x48' (ENV: EX_SSS_BOOT_SSS_PORT=/dev/i2c-0:0x48)
sss   :INFO :atr (Len=35)
      00 A0 00 00    03 96 04 03    E8 00 FE 02    0B 03 E8 08
      01 00 00 00    00 64 00 00    0A 4A 43 4F    50 34 20 41
      54 50 4F
sss   :WARN :Communication channel is Plain.
sss   :WARN :!!!Not recommended for production use.!!!
App   :WARN :#####################################################
App   :INFO :uid (Len=18)
      04 00 50 01    EA 31 EF DE    83 23 DA 04    0F 81 AA 2A
      1E 90
App   :WARN :#####################################################
App   :INFO :Applet Major = 3
App   :INFO :Applet Minor = 1
App   :INFO :Applet patch = 1
App   :INFO :AppletConfig = 6FFF
App   :INFO :With    ECDSA_ECDH_ECDHE
App   :INFO :With    EDDSA
App   :INFO :With    DH_MONT
App   :INFO :With    HMAC
App   :INFO :With    RSA_PLAIN
App   :INFO :With    RSA_CRT
```

```
App   :INFO :With    AES
App   :INFO :With    DES
App   :INFO :With    PBKDF
App   :INFO :With    TLS
App   :INFO :With    MIFARE
App   :INFO :With    I2CM
:（省略）
```

以上で Plug & Trust Middleware を利用するための準備を終わります。

6.27.4.4. Plug & Trust Middleware を活用する

ここからは実際に Plug & Trust Middleware を用いて EdgeLock SE050 を利用する方法を説明していきます。

EdgeLock SE050 の保存された鍵を利用する

plug-and-trust パッケージには OpenSSL engine のライブラリが含まれており、OpenSSL を利用して間接的に EdgeLock SE050 を操作することができます。ここでは例として、EdgeLock SE050 のチップ製造時に書き込まれた鍵を使って OpenSSL で署名と署名の検証を行います。

まず、次のコマンドでリファレンスキーを取得します。

- keyid = 0xF0000100
- Cloud connection key 0 (prime256v1)

```
[container ~]# se05x_getkey 0xF0000100 refkey.pem "$AT_SE_PARAM"
:(省略)
[container ~]# ls
refkey.pem ❶
```

❶ refkey.pem が生成されます。



チップ製造時に書き込まれた鍵について

「6.27.4.5. 補足説明」 を参照してください。



リファレンスキーについて

「6.27.4.5. 補足説明」 を参照してください。

次のコマンドで署名して sig.bin を生成します。message.txt は任意のファイルを用意してください。取得したリファレンスキー refkey.pem を利用します。

```
[container ~]# echo "This is test" > message.txt
[container ~]# openssl dgst -sha256 -sign refkey.pem -out sig.bin message.txt
```

```
ssse-flw: EmbSe_Init(): Entry
App   :INFO :Using PortName='/dev/i2c-0:0x48' (ENV: EX_SSS_BOOT_SSS_PORT=/dev/i2c-0:0x48)
sss   :INFO :atr (Len=35)
      00 A0 00 00    03 96 04 03    E8 00 FE 02    0B 03 E8 08
      01 00 00 00    00 64 00 00    0A 4A 43 4F    50 34 20 41
      54 50 4F
sss   :WARN :Communication channel is Plain.
sss   :WARN :!!!Not recommended for production use.!!!
ssse-flw: Version: 1.0.5
ssse-flw: EmbSe_Init(): Exit
ssse-flw: Control Command EMBSE_LOG_LEVEL; requested log level = 4
[container ~]# ls
message.txt  refkey.pem   sig.bin ❶
```

❶ sig.bin が生成されます。

署名が正しいかどうかを確認するために署名を検証します。

```
[container ~]# openssl dgst -sha256 -prverify refkey.pem -signature sig.bin message.txt
ssse-flw: EmbSe_Init(): Entry
App   :INFO :Using PortName='/dev/i2c-0:0x48' (ENV: EX_SSS_BOOT_SSS_PORT=/dev/i2c-0:0x48)
sss   :INFO :atr (Len=35)
      00 A0 00 00    03 96 04 03    E8 00 FE 02    0B 03 E8 08
      01 00 00 00    00 64 00 00    0A 4A 43 4F    50 34 20 41
      54 50 4F
sss   :WARN :Communication channel is Plain.
sss   :WARN :!!!Not recommended for production use.!!!
ssse-flw: Version: 1.0.5
ssse-flw: EmbSe_Init(): Exit
ssse-flw: Control Command EMBSE_LOG_LEVEL; requested log level = 4
Verified OK
```



クラウドサービスに接続するために事前書き込みされた鍵と証明書を利用する方法は以下を参照してください(Armadillo-IoT ゲートウェイ G4 向けの内容ですが、本機種でも同じ手順でご利用いただけます)。

EdgeLock SE050 を使用して AWS IoT Core へ接続する

https://armadillo.atmark-techno.com/howto/connect_to_aws_iot_core_for_aiot_g4

EdgeLock SE050 を使用して Azure IoT Hub へ接続する

https://armadillo.atmark-techno.com/howto/connect_azure_iot_hub_for_aiot_g4

ユーザーが生成した鍵を保存して利用する

EdgeLock SE050 ではユーザーが生成した鍵を保存することもできます。一度保存された鍵は、そのまま暗号処理に利用することができるようになります。ここでは例として、ユーザー鍵を保存して openssl cms で暗号化と復号を実行してみます。

以降の例で生成されるファイルの役割を説明します。

- key.pem: openssl で生成された秘密鍵
- cert.pem: openssl で生成された自己証明書（公開鍵）
- refkey.pem: SE050 に保存した秘密鍵 key.pem を使用するためのリファレンスキー
- message.txt: 任意のファイル
- message.enc: message.txt を暗号化したファイル
- message.dec: message.enc を復号したファイル

以降の作業のため、「6.27.4.4. Plug & Trust Middleware を活用する」で生成したファイルは削除します。

```
[container ~]# rm -rf *
```

まず、ECC の曲線 prime256v1 の鍵を生成します。証明書の情報はお任せします。

```
[container ~]# openssl req -x509 -nodes -days 3650 -newkey ec ¥
-pkeyopt ec_paramgen_curve:prime256v1 -keyout key.pem -out cert.pem
```

生成した鍵と自己証明書を EdgeLock SE050 に保存します。EdgeLock SE050 では keyid が 0x00000001 から 0x7BFFFFFF の範囲で開放されており、 50kB のストレージが搭載されています。ただし、今後アットマークテクノが一部ストレージを利用する予定であるため、Armadillo-610 で利用できる keyid およびストレージ容量は以下の通りとなります。

- keyid: 0x00000001 ～ 0x7BFEFFFF
- ストレージ容量: 35kB

```
[container ~]# se05x_setkey -f 0x10 key.pem "$AT_SE_PARAM"
[container ~]# se05x_setkey -f 0x11 cert.pem "$AT_SE_PARAM"
```

SE050 に秘密鍵 key.pem を保存したので、流出を防ぐために key.pem を削除します。

```
[container ~]# rm -rf key.pem
```



Secure Object についての詳しい情報は以下を参照してください。

SE050A/B/C/D/F APDU Specification

https://www.nxp.com/search?keyword=AN12413



秘密鍵を EdgeLock SE050 に保存に成功したあとは、利用する際に EdgeLock SE050 にアクセスすることになります。そのため、ファイルシステム上の key.pem は基本的に必要ありません。セキュリティ上、削除することをお勧めします。

0x10 に保存した秘密鍵を使用するためのリファレンスキー refkey.pem を生成します。

```
[container ~]# se05x_getkey 0x10 refkey.pem "$AT_SE_PARAM"
```



リファレンスキーについて

「6.27.4.5. 補足説明」 を参照してください。

自己証明書に含まれる公開鍵 cert.pem を使って暗号化します。message.txt は任意のファイルを用意してください。

```
[container ~]# echo "This is test" > message.txt
[container ~]# openssl cms -encrypt -binary -aes256 -in message.txt ¥
-out message.enc cert.pem
[container ~]# ls
cert.pem  message.enc  message.txt  refkey.pem ❶
```

❶ message.txt を暗号化した message.enc が生成されます。

以下はあくまで例ですが、暗号化されたファイルは以下のような内容になります。

```
[container ~]# cat message.enc
MIME-Version: 1.0
Content-Disposition: attachment; filename="smime.p7m"
Content-Type: application/pkcs7-mime; smime-type=enveloped-data; name="smime.p7m"
Content-Transfer-Encoding: base64

MIIBWgYJKoZIhvcNAQcDoIIBSzCCAUcCAQIxggECoYH/AgEDoFGhTzAJBgcqhkjO
PQIBA0IABPFaZ4KiT2MfZYLMgyhKmTRMeU8+dOwLVW9qe5l6QuFgPUmIVq5Q/jC1
teAvZVli90gmGmvH+GqkFy2rVKpvYEowGAYJK4EFEIZIPwACMAsGCWCGSAFlAwQB
LTCBjDCBiTBdMEUxCzAJBgNVBAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEw
HwYDVQQKDBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGQCFB7+NC1Xtvz6O3ptiaPN
hHqnkocSBCgVCiONik9gab2iWroRv4VLp35RKBGY/YRv4OytUgEzOV7ejHTzoFPa
MDwGCSqGSIb3DQEHATAdBglghkgBZQMEASoEEL4YFn0m8QCBHSBJZHeWV16AEEWr
AlzrhnvA3jiOYZCX4e4=
```

EdgeLock SE050 のキーストレージにある秘密鍵を使って復号します。

```
[container ~]# openssl cms -decrypt -in message.enc -out message.dec ¥
-inkey refkey.pem
```

```
[container ~]# cat message.dec
This is test
```

ユーザーが保存した鍵を削除する

「6.27.4.4. Plug & Trust Middleware を活用する」で EdgeLock SE050 にユーザーが生成した鍵を削除する方法を紹介します。注意点としまして、複数保存した鍵の中から 1 つだけを選んで削除することはできません。EdgeLock SE050 内に保存されているユーザーが保存した鍵をすべて削除し、その後復元できませんのでよく理解した上でご使用ください。



「6.27.4.5. 補足説明」で紹介している、事前に書き込まれた鍵と証明書は削除されません。

EdgeLock SE050 内に保存されているユーザーが保存した鍵をすべて削除するコマンドを「図 6.192. EdgeLock SE050 内に保存されているユーザーが保存した鍵をすべて削除する」に示します。

```
[container ~]# export EX_SSS_BOOT_SSS_PORT="$AT_SE_PARAM"
[container ~]# se05x_reset -y
:(省略)
secure objects have been successfully removed.
```

図 6.192 EdgeLock SE050 内に保存されているユーザーが保存した鍵をすべて削除する

削除後、保存されていた keyid から鍵を読み出そうとしても失敗します。

```
[container ~]# se05x_getkey 0x10 refkey "$AT_SE_PARAM"
:(省略)
failed to retrieve handle with keyid.
keyid may be invalid. please check keyid.
failure occurred in sss api (kStatus_SSS_Fail)
```

図 6.193 削除したユーザーが保存した鍵を読み出そうとしてみる

6.27.4.5. 補足説明

事前書き込みされた鍵と X.509 証明書

EdgeLock SE050 のフラッシュメモリにはチップ製造時に書き込まれたチップ固有な鍵と NXP に署名された X.509 証明書を保持しています。それらの鍵や証明書は Armadillo-610 を購入後にすぐにそのままの状態でクラウドサービスなどの PKI に利用できます。事前に書き込みがされた状態で空き容量はあるので、ユーザーの鍵や証明書を追加することも可能です。



事前書き込みされた鍵と証明書については以下のドキュメントを参照してください。利用可能な keyid、鍵の種類、想定する用途が記載されています。

なお、Armadillo-610 に搭載されている EdgeLock SE050 は Variant C です。

SE050 configuration

https://www.nxp.com/search?keyword=AN12436

リファレンスキー

EdgeLock SE050 は秘密鍵が外部に露出しません。これは、いったん鍵を EdgeLock SE050 に書き込むと秘密鍵を抜き出すことができない (削除や書き換えは可能) という仕様で実現されています。攻撃者がチップの中にある、どこかのサイトの認証資格などを有する証明書を得るためには、物理的にチップ自体も必要ということになってきます。

秘密鍵を抜き出すことができない仕様で、どのように秘密鍵を利用するかというと、リファレンスキーと呼ばれる特殊なファイルで代用します。リファレンスキーは秘密鍵のフォーマットで保存されますが、公開鍵部分のみ有効で、それ以外の値は管理用やダミーの情報になっています。リファレンスキーをファイルシステムに置いておいて、EdgeLock SE050 を利用する時にそのファイルを秘密鍵のように使うと、Plug & Trust Middleware に含まれる OpenSSL の engine のライブラリがフックして EdgeLock SE050 にアクセスします。

e05x-tools

se05x-tools は EdgeLock SE050 をキーストレージとして利用するためのツールです。アットマークテクノから Armadillo Base OS 向けにリリースされています。se05x-tools の内部では Plug & Trust Middleware を利用しています。

対応する非対称暗号鍵の種類は以下のとおりです。

se05x-tools でサポートされる ECC カーブ

- prime192v1, secp224r1, prime256v1, secp384r1, secp521r1

se05x-tools でサポートされる RSA 鍵長

- 1024bit, 2048bit, 3072bit, 4096bit



Secure Object についての詳しい情報は以下を参照してください。

SE050A/B/C/D/F APDU Specification

https://www.nxp.com/search?keyword=AN12413

6.27.5. デバッグ機能を閉じる

ここでは、デバッグ機能を閉じる方法とその影響について説明します。

デバッグ機能は開発の効率を向上させるために開発フェーズではなくてはならないものです。製品が市場に出荷されると市場不良の解析にも効果を発揮します。しかし、開発者にとって便利であるということは、同時に攻撃者にとっても便利な解析手段となり得ます。想定される製品の運用形態によって、デバッグ機能が必須である運用形態もあります。セキュリティリスクとのトレードオフになる可能性があるので、有効にするかどうかを検討することをお勧めします。



デバッグ機能を閉じることは開発者と攻撃者だけでなく、**アットマークテクノによる解析についてもトレードオフになります。閉じられたインターフェースを利用した解析ができなくなることをご留意いただきますようお願いします。**

6.27.5.1. JTAG と SD ブートを無効化する

Armadillo-610 の出荷時は、JTAG ポートは有効なままで出荷されます。JTAG が有効なままですと攻撃者が悪意のあるコードを実行する、メモリをダンプして鍵などセキュアな情報を取り出すなどの行為が可能となります。そのため、市場に出荷される最終段階では JTAG を無効化することをお勧めします。

次に、SD ブートは SD メディアを挿すだけで起動する便利な機能です。その反面、SD メディアの盗難や流出によってシステムへの侵入、SD ブートを利用したセキュアブート鍵に対する攻撃が考えられます。これらのリスクは、SD ブートを無効にすることで排除することが可能です。しかし、SD ブートはシステムの復旧の役割も担っています。SD ブートを無効化することによって、eMMC ブートでは起動できない状態に陥った場合、合わせて JTAG の無効化が合わせて設定されていると、二度と復旧することができないデバイスになる (廃棄するしかない) 可能性があることを考慮してください。

ソフトウェアの開発が完了し量産段階に入ると、量産用 Armadillo にソフトウェアを書き込むためのインストールディスクが必要になります。Armadillo Base OS では開発が完了した Armadillo 上でコマンドを実行することにより、その Armadillo の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作成することができます。詳細な情報は「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」を参照してください。

JTAG と SD ブートを無効化したインストールディスクを作成するには `abos-ctrl make-installer` コマンドを実行する前に、`abos-ctrl installer-setting` コマンドで設定します。



作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 6.194 インストールディスクの 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 6.195 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 6.196 JTAG と SD ブートの設定値をリセットする

JTAG と SD ブートの無効化設定を実行した後に、 abos-ctrl make-installer コマンドを実行してインストールディスクを作成します。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。コマンド実行例は以下のようになります。

```
[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
y
WARNING: it will overwrite your SD card!!

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
Upcase 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
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 6.197 インストールディスクを作成する

❶ y を入力します。

❷ 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作成します。

「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ています。 Armadillo から microSD カードを抜去してください。この microSD カードを使って量産用 Armadillo にインストールを行うと、その Armadillo は JTAG と SD ブートが無効化された状態となります。

6.27.5.2. U-Boot の環境変数の変更を制限する

Armadillo Base OS の U-Boot は eMMC の固定の領域から環境変数を読み取っています。

本来の使い方ではユーザーはその eMMC の領域に書込みできませんが、eMMC が外部から変更されたとしても Linux が正しいシーケンスで起動するようにしたい場合は、 U-Boot の環境変数をある程度ロックした方が安全です。

imx-boot バージョン 2020.04-at23 以降では、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環境変数以外は変更できなくなります。

その変更可能の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。CFG_ENV_FLAGS_LIST_STATIC にリストされている環境変数以外は変更できなくなります。

提供しているコンフィグでは、以下の環境変数が変更可能です：

- 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て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。

また、Linux を起動して fw_printenv 等を使用しても Linux 側ではデフォルト値や変更可能の環境変数リストは把握していないので信頼性が低いです。変数の値に疑いがある場合は U-Boot の prompt で確認してください。

6.27.5.3. U-Boot プロンプトを無効にする

Armadillo Base OS の U-Boot はデフォルトで autoboot が有効になっています。autoboot が有効な状態では決まったディレイ (bootdelay) の間キー入力を待ち、入力がなければ自動的に bootcmd を実行して Linux を起動します。一方、決まった時間にキー入力があるとプロンプトが表示されて、様々なコマンドを実行することができます。これらのコマンドはとても便利なものですが、攻撃者にとっても攻撃を仕掛けるために有効な手段となり得ます。ここでは、決まった時間待つ処理を無効化する方法を説明します。

bootdelay
- 2 : デフォルト。2 秒待つ
- 0 : 待ち時間なし。ただしキー入力でプロンプトが表示される
- -1 : autoboot が無効
- -2 : 待ち時間なし。キー入力も無効

「6.27.5.2. U-Boot の環境変数の変更を制限する」 の手順を行った場合は imx-boot の uboot-imx/configs/x2_defconfig に「CONFIG_BOOTDELAY=-2」を追加して変更してください。

環境変数が変更可能な場合は Armadillo Base OS の /boot/uboot_env.d/no_prompt の様なファイルを作って、そちらに変数を設定すれば今後のアップデートにも適用されます。

詳細は「6.19. u-boot の環境変数の設定」を参考にしてください。

```
[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 ❹
bootdelay=-2
```

❶ コンフィグファイルを生成します。

❷ ファイルを永続化します。

❸ 変数を書き込みます。swupdate で書き込む場合は自動的に行われています。

❹ 書き込んだ変数を確認します。



「6.27.5.2. U-Boot の環境変数の変更を制限する」 を行ってない場合に CONFIG_BOOTDELAY は無視されます。必ず

CONFIG_ENV_WRITEABLE_LIST も設定するか、/boot/uboot_env.d で設定してください。



ここで説明した以外にもいくつかの方法があります。ソースコードにドキュメントがあります。以下を参照してください。

imx-boot-[VERSION]/uboot-imx/doc/README.autoboot

6.27.6. コンテナの最小化

開発時に使用するパッケージが最新でセキュリティに問題がないと考えられていても、後に脆弱性が発見される可能性があります。

コンテナ内のパッケージに脆弱性が見つかった場合、開発者は公開されたパッチを適用する必要があります。しかし、インストールされているパッケージが多いと、アップデートが必要なパッケージを把握するのが困難になることがあります。

その結果、メンテナンスされていないパッケージが残り、セキュリティリスクが高まる可能性があります。

このようなリスクを回避するためには、コンテナにインストールするパッケージを必要最低限に抑えることが重要です。

コンテナを最小化する主な利点は以下の通りです。

- セキュリティリスクの低減：不要なパッケージやサービスを減らすことで、脆弱性の発生箇所が少なくなり、攻撃対象領域が縮小します。
- 保守性の向上：インストールされているパッケージが少ないため、アップデートや脆弱性対応が容易になります。
- コストの削減：コンテナイメージが小さくなることで、ネットワーク経由でのダウンロードや転送が速くなり、ストレージの消費も抑えられます。これにより、転送コストや時間の削減につながります。
- ライセンス管理の簡素化：含まれるソフトウェアが少ないため、ライセンス管理や監査が容易になります。

VS Code の拡張機能「Armadillo Base OS Development Environment（ABOSDE）」では、コンテナにインストールされるパッケージを最小化する工夫がされています。

もし ABOSDE を使用しない場合でも、以下の内容を参考にすることで、コンテナのセキュリティを向上させることができます。

6.27.6.1. コンテナに使用する OS の選択

ABOSDE が提供する Shellscript/Python/C 言語のプロジェクトでは、arm64v8 と arm32v7 の両方で Debian または Alpine コンテナを選択できます。

Debian コンテナでは、ディストリビューションとして bullseye-slim を使用しています。

以下は、Debian と Alpine のコンテナ内パッケージを比較した表です。(以降、2025/7/1 時点の情報です。)

表 6.28 Debian と Alpine の比較

比較項目	Debian	Alpine
パッケージ管理システム	apt	apk
インストール済みパッケージ数	88 個	16 個
利用可能パッケージ数	豊富	Debian に比べて少ない

Alpine は利用可能なパッケージ数が少ないため、開発難易度が高くなる可能性があります。しかし、Alpine を採用することで、Debian よりもコンテナ内のパッケージ数を減らすことができます。

以下は、Python プロジェクトを Debian と Alpine でビルドした場合のコンテナイメージサイズの比較例です。どちらも LED を点滅させるアプリケーションを動作させています。

表 6.29 Python プロジェクトにおける Debian と Alpine コンテナのサイズ比較

アーキテクチャ	Debian	Alpine
arm32v7	117 MB	64.8 MB
arm64v8	139 MB	76.1 MB

この例では、どちらのアーキテクチャにおいても Python プロジェクトのコンテナサイズは Alpine コンテナベースが Debian コンテナベースの約半分であることが分かります。

ABOSDE におけるコンテナ OS の選択方法は「3.15.3.4. ABOSDE におけるコンテナ OS の選択方法」を参照してください。

6.27.6.2. Debian コンテナにインストールするパッケージを減らす方法

Debian を使用する場合、依存パッケージを減らすことでコンテナサイズを小さくできます。

ABOSDE が提供する Dockerfile では、デフォルトで必要最低限のパッケージのみをインストールするオプションが指定されています。

```
:(省略)

# Add extra packages to containers/packages.txt
ARG PACKAGES
RUN apt-get update && apt-get upgrade -y ¥
    && apt-get install -y --no-install-recommends ${PACKAGES} ¥ ❶
    && apt-get clean

:(省略)
```

図 6.198 Debian コンテナの Dockerfile の例

❶ --no-install-recommends オプションを指定しています。

--no-install-recommends オプションを使用すると、必要最低限のパッケージのみをインストールできます。

以下は、Python プロジェクトのサンプルプログラムを使用して、--no-install-recommends の有無による Debian コンテナのサイズを比較した結果です。(2025/7/1 時点)

表 6.30 Python プロジェクトにおける --no-install-recommends の有無による Debian コンテナのサイズ比較

アーキテクチャ	オプション無し	オプションあり
arm32v7	317 MB	117 MB
arm64v8	418 MB	139 MB

この例から、どちらのアーキテクチャでも --no-install-recommends オプションを指定することで、コンテナサイズを約 1/3 に削減できることが分かります。

6.28. eFuse を変更する

Armadillo-610 で採用している CPU (i.MX6ULL) には、一度しか書き込むことのできない eFuse が搭載されています。 eFuse には、 CPU がブートする時の設定や MAC アドレスなどが書かれます。Armadillo-610 は組み込み機器を作り込むエンジニアを対象にした製品ですので、 eFuse もユーザーに開放し、細かな制御を可能にしています。しかし eFuse はその性質上、一度書き間違うと直すことができません。十分に注意してください。



eFUSE は一度書き込むと元に戻すことができません。eFUSE の設定によっては Armadillo-610 が正常に動作しなくなる可能性がありますので、書き込みを行う際には細心の注意を払うようお願いいたします。eFUSE の設定によって異常が起こった場合は保証対象外となります。

MAC アドレスは Armadillo-610 の出荷時に書き込まれているので、新たに書き込む必要はありません。この章では U-Boot を使って eFuse の書き換えを行い、ブートモードを制御する方法を説明します。

eFuse を変更する場合は、必ず「i.MX 6ULL Applications Processor Reference Manual [https://www.nxp.com/docs/en/reference-manual/IMX6ULLRM.pdf]」を参照してください。重要な章は、以下の 4 つです。

- Chapter 5: Fusemap
- Chatper 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。

6.28.1. ブートモード

i.MX6ULL にはブートモードを決める BOOT_MODE0 と BOOT_MODE1 というピンがあります。Armadillo-610 では、BOOT_MODE0 は 0 、BOOT_MODE1 は 1 となるよう回路が設計されており、ブートモードは必ず Internal Boot モードとなります。

6.28.1.1. Internal Boot モード

Internal Boot モードでは、 on-chip boot ROM に書き込まれているコードが実行し、ブート可能なデバイスを検索します。リファレンスマニュアル「8.5 Boot devices (internal boot)」に、i.MX6ULL がブートできるデバイスの一覧が記載されています。Armadillo-610 では、そのうちオンボード eMMC と microSD カードに対応しています。

Internal Boot モードでは、GPIO によって eFuse の設定を上書き (override) できるようになっています。この機能は eFuse の BT_FUSE_SEL が 0 の場合のみ有効となります。eFuse の設定とは異なり何度も再設定できる点では便利ですが、override に対応したピンには i.MX6ULL の電源投入時に決まった信号を入力しておかなければいけないため、ハードウェア設計上は不便になります。

Armadillo-610 では、GPIO による override を利用することで、仕様が確定していない段階ではブートデバイスを自由に何度も切り替えることを可能にしつつ、BT_FUSE_SEL を 1 にして GPIO による override を無効化することで、仕様が確定した段階では自由なハードウェア設計が可能になるよう配慮しています。また、GPIO による override を無効化することで、フィールドに出した製品が悪意ある人によって意図していないブートをし、被害が出ることを防ぐことができます。(もちろん、ブート後に root アカウントを乗っ取られるような作りでは、意味がありませんが…)

6.28.2. ブートデバイス

Internal Boot モードでは、GPIO によって eFuse の設定を上書き (override) できるようになってると紹介しましたが、Armadillo-610 では、Armadillo-610 拡張ボードの JP1 はまさにこの機能を使っています。 JP1 は BJP1(Armadillo-610 CON2_42 ピン) に接続されており、 LCD1_DATA05 と LCD1_DATA11 の制御をしていますが、これらのピンはそれぞれ BOOT_CFG1[5] と BOOT_CFG2[3] を override しています。「8.3.2 GPIO boot overrides」の 表「8-3. GPIO override contact assignments」を確認してください。

ややこしい事に、この BOOT_CFG で始まる eFUSE は、リファレンスマニュアルの中では eFuse のアドレスでも表記されています。 BOOT_CFG1 は eFuse のアドレスで言うと 0x450 の下位 8 bit つまり 0x450[7:0] であり、 BOOT_CFG2 は上位 8 bit つまり 0x450[15:8] にあたります。これは「5.1 Boot Fusemap」の表「5-5. SD/eSD Boot Fusemap」または表「5-6. MMC/eMMC Boot Fusemap」を確認することでわかります。

さらにややこしい事に、eFuse を書き込む場合にはこれら全ての値が使えず、On-Chip OTP Controller の bank と word の値が必要になります。これらの値は リファレンスマニュアルの「On-Chip OTP Controller」を参照してください。後で出てきますが Boot From Fuses で使用する BT_FUSE_SEL という eFuse のように GPIO による override ができないものもあります。

表 6.31 GPIO override と eFuse

信号名	eFuse 名	eFuse アドレス	OCOTP 名	Bank	Word
LCD1_DATA05	BOOT_CFG1[5]	0x450[5]	OCOTP_CFG4	0	5
LCD1_DATA11	BOOT_CFG2[3]	0x450[11]	OCOTP_CFG4	0	5
N/A	BT_FUSE_SEL	0x460[4]	OCOTP_CFG5	0	6

Armadillo-610 では SD カード または eMMC からのブートになるので、ブートデバイスを選択する eFuse BOOT_CFG1[7:4] は、010x または 011x になります。

リファレンスマニュアル「8.5.3.1 Expansion device eFUSE configuration」には、さらに詳しく SD/MMC デバイスの設定について記載されています。テーブル「8-15. USDHC boot eFUSE descriptions」によれば、eFuse の 0x450[7:6] が 01 の場合に SD/MMC デバイスからブートすることを決めています。さらに 0x450[5] が 0 なら SD が、 0x450[5] が 1 なら MMC が選択されます。つまり、4 から 7 bit までの間で 5 bit 目だけが MMC か SD かを決めています。 BOOT_CFG1[5] が 0 の場

合はコントローラーは SD デバイスが繋がっている前提で、 BOOT_CFG1[5] が 1 の場合は MMC デバイスが繋っている前提で動作します。

i.MX6ULL には、SD/MMC のコントローラーである uSDHC が 2 つ搭載されています。Armadillo-610 では、eMMC が uSDHC1 に、 microSD カードが uSDHC2 に接続されています。ブート時にどちらのコントローラーからブートするかを決めている eFuse が 0x450[12:11] です。0x450[12:11] が 00 であれば uSDHC1 つまりオンボード eMMC から、01 であれば uSDHC2 つまり microSD カードからブートします。言い換えると Armadillo-610 でオンボード eMMC からブートしたい場合は、0x450[5] を 1 に、 0x450[12:11] を 00 にします。逆に microSD カードから起動したい場合は 0x450[5] を 0 に、0x450[12:11] を 01 にします。

表 6.32 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

6.28.3. eFuse の書き換え

Armadillo-610 では、U-Boot のコマンドによって eFuse の書き換えをサポートしています。 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

6.28.4. eFuse の設定によるブートデバイスの選択

6.28.4.1. BT_FUSE_SEL

eFuse の設定によるブートデバイスの選択を可能にするには、 eFuse に書き込んだ値が正しいことを i.MX6ULL に教える必要があります。そのための eFuse が BT_FUSE_SEL (0x460[4]) です。Armadillo-610 では、このビットが 1 であれば、GPIO による override が無効になり eFuse の設定にしたがってブートデバイスが選択されるようになります。

6.28.4.2. eMMC からのブートに固定

オンボード eMMC からだけブートさせたい場合は、ブートデバイスの種類で MMC と、コントローラーで 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 6.33 オンボード eMMC のスペック

名前	Bit	eFuse	値	bit 列	可変
BOOT_CFG2	[15:13]	Bus Width	8 bit	010	不
	[12:11]	Port Select	uSDHC1	00	不
	[10]	Boot Frequencies	500 / 400 MHz	00	可
	[9]	SD Voltage Selection	3.3 V	0	不
	[8]	-	-	0	-
BOOT_CFG1	[7:5]	eMMC	-	011	不
	[4]	Fast Boot	Regular	0	可
	[3]	SD/MMC Speed	High	0	不
	[2]	Fast Boot Acknowledge Disable	Enabled	0	可
	[1]	SD Power Cycle Enable	Disabled	0	可
	[0]	SD Loopback Clock Source Sel	SD Pad	0	不

値を見易いように、 BOOT_CFG2 を上にしています。 BOOT_CFG1 と BOOT_CFG2 は、OCOTP_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-610 ができあがりました。



eMMC からしか起動しないので、あやまって eMMC に書き込まれている U-Boot を消してしまうと、二度と起動しないようになります。注意してください。



eMMC Fast Boot 機能を使う場合や Power Cycle を Enable にする場合は、当該ビットを 1 に変更してください。

同じ要領で、SD からだけしかブートしないようにすることも可能です。しかし eFuse によるブートデバイスの固定は、意図しないブートを防ぐことが目的です。 Armadillo-610 で microSD からのブートに固定することは可能ですが、別の microSD カードを挿入されてしまうと、その別の microSD カードからブートしてしまうので目的を達成できません。理解してお使いください。

6.28.4.3. eFuse のロック

書き込んだ eFuse の値を変更されてしまっては、Boot From Fuse モードにしている意味がありません。i.MX6ULL では eFuse を変更できなくするビットも用意されています。

リファレンスマニュアル「5.3 Fusemap Descriptions Table」を確認してください。

6.29. オプション品

本章では、Armadillo-610 のオプション品について説明します。

表 6.34 Armadillo-610 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	SA-SCUSB-00	Armadillo-610 開発セットに同梱
Armadillo-610 拡張ボード	ー	Armadillo-610 開発セットに同梱
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	7 インチタッチパネル WVGA 液晶が付属
Armadillo-400 シリーズ LCD オプションセット	OP-A400-LCD43EXT-L01	4.3 インチタッチパネル WQVGA 液晶が付属

6.29.1. USB シリアル変換アダプタ

6.29.1.1. 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。

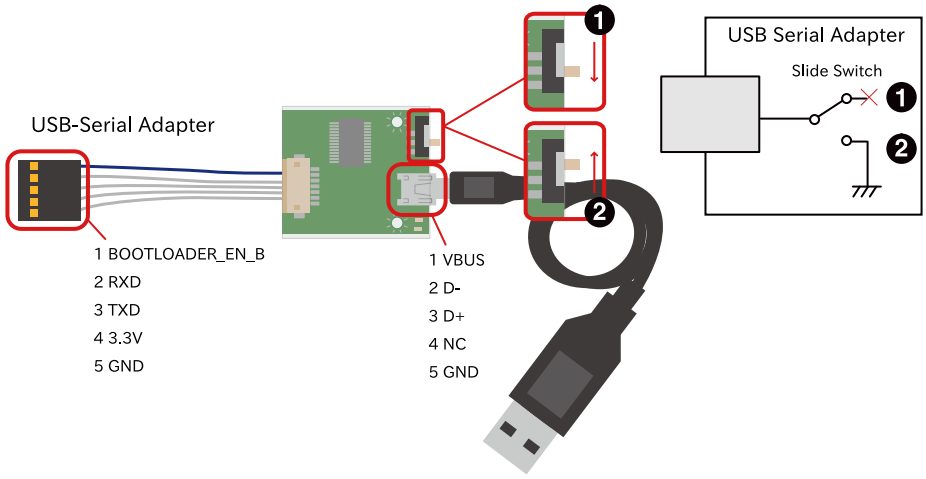


図 6.199 USB シリアル変換アダプタの配線

❶ オープン

❷ GND ショート

シリアルインターフェース(Armadillo-610 拡張ボード: CON3)に接続した場合、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 6.35 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-610 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。

6.29.2. Armadillo-610 拡張ボード

6.29.2.1. 概要

Armadillo-610 拡張ボードは Armadillo-610 を搭載する拡張ボードを設計開発するためのリファレンスボードです。電源、LAN、USB、SD、LCD [2]、RS-485、オーディオ、絶縁デジタル入出力、リアルタイムクロック、スイッチ、LED 等の動作を確認することが可能です。Armadillo-610 拡張ボードは Armadillo-610 開発セットに同梱されます。



Armadillo-610 拡張ボードの回路図、部品表は購入者向けの限定データとして link:「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。



Armadillo-610 拡張ボードは Armadillo-610 がないと機能しない製品ですので、Armadillo-610 を搭載した状態での仕様を説明します。

6.29.2.2. 仕様

Armadillo-610 拡張ボードの主な仕様は次のとおりです。

表 6.36 Armadillo-610 拡張ボードの仕様

LAN(Ethernet)	100BASE-TX/10BASE-T x 1、AUTO-MDIX 対応
シリアル(UART)	3.3V CMOS レベル x 1、RS-485 x 1
USB	USB 2.0 Host(High Speed) x 2、USB 2.0 OTG(High Speed) x 1
SD	SD スロット x 1 [a]

[2]動作を確認するためには、別途オプション品を購入する必要があります。

カレンダ時計	リアルタイムクロック搭載、バックアップ用コネクタ搭載 [b]
オーディオ	モノラルスピーカー出力 x 1
ビデオ	LCD オプションセット(7 インチタッチパネル WVGA 液晶)接続可能 [c]
接点入出力	入力 x 2、出力 x 2
Grove インターフェース	Grove コネクタ x 4 [d] UART x 1、I2C x 1、A/D x 2
拡張インターフェース	UART、SD、LCD、I2S、S/PDIF、MQS、I2C、SPI、CAN、A/D、PWM、GPIO 等 [e]
スイッチ	ユーザースイッチ x 1、 リセットスイッチ x 1、パワースイッチ x 1
LED	ユーザー LED x 1
電源電圧	DC 9～24V±10%(メイン電源)、DC 2.0～3.5V(RTC バックアップ)、DC 2.75～3.3V(i.MX6ULL 内蔵 RTC バックアップ)
消費電力	約 1.2W(待機時)、約 1.8W(LAN 通信時) [f]
使用温度範囲	+10～+40℃(結露なきこと)
外形サイズ	115 x 160 mm(突起部を除く)

[a]Armadillo-610 上の microSD スロットと排他利用となります。
[b]電池は付属しません。
[c]Armadillo-610 拡張ボードに LCD オプションセット(7 インチタッチパネル WVGA 液晶)は付属しません。
[d]LCD インターフェースと排他利用となります。
[e]シリアル、SD スロット、オーディオ、LCD 等のインターフェースと排他利用となります。
[f]電源電圧 DC 12V 時の消費電力です。外部接続機器の消費分は含みません。



Armadillo-610 開発ボードは設計開発用のリファレンスボードです。仕様や外観を予告なく変更する場合があります。

6.29.2.3. ブロック図

Armadillo-610 拡張ボードのブロック図は次のとおりです。

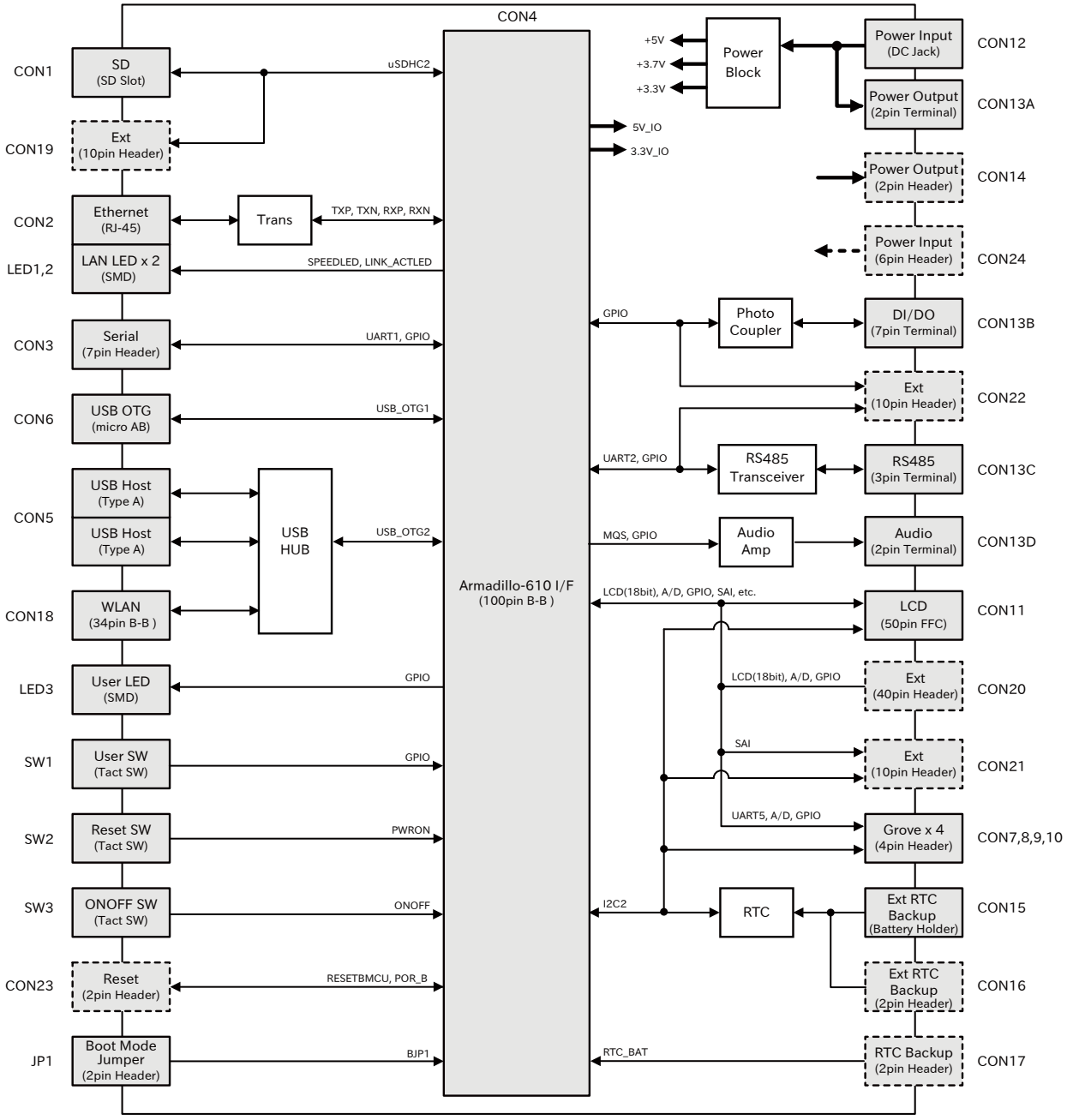


図 6.200 Armadillo-610 拡張ボードのブロック図

Armadillo-610 拡張ボードの電源回路の構成は次のとおりです。

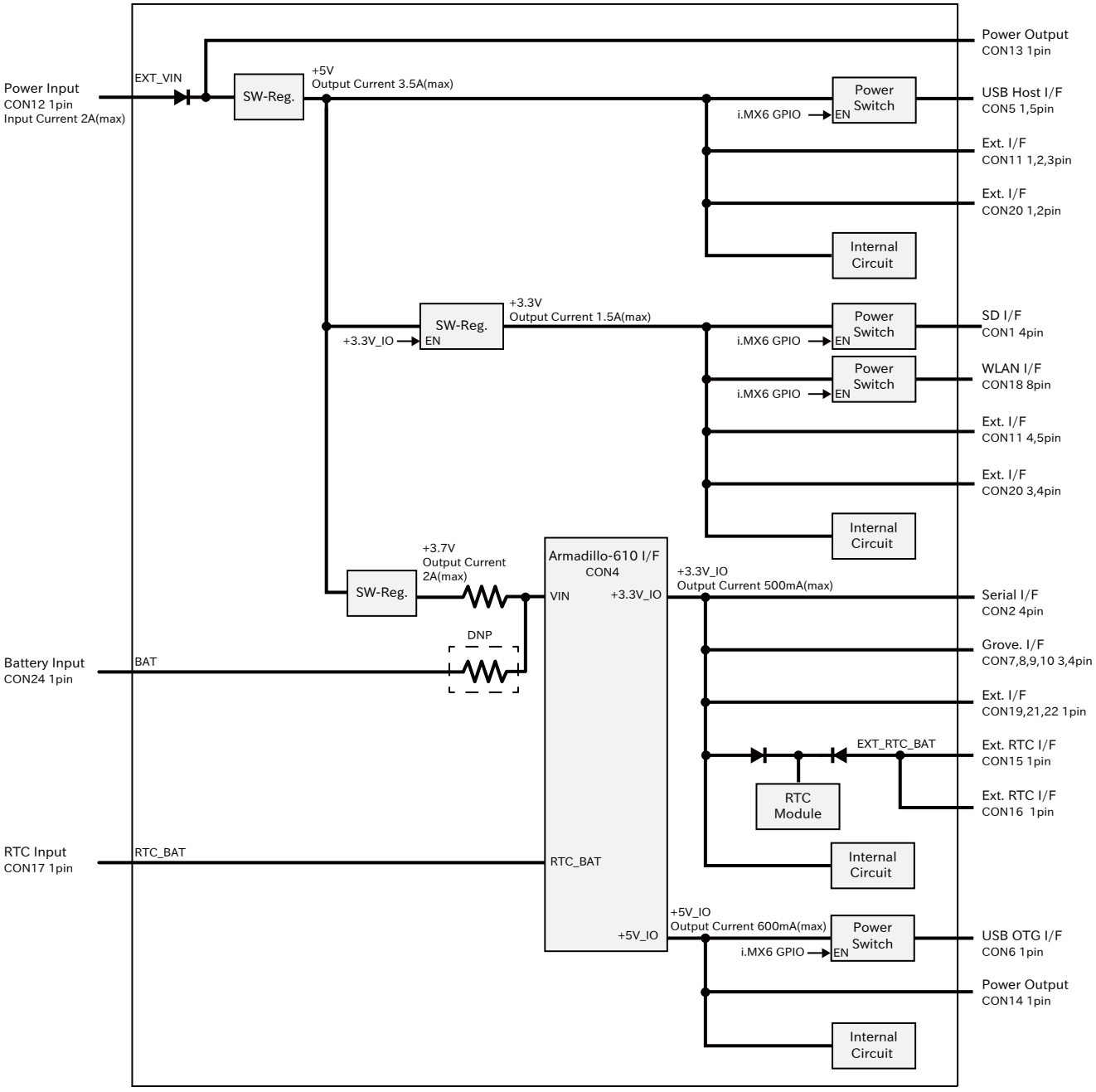


図 6.201 Armadillo-610 拡張ボードの電源回路の構成

6.29.2.4. インターフェース仕様

Armadillo-610 拡張ボードのインターフェース仕様について説明します。

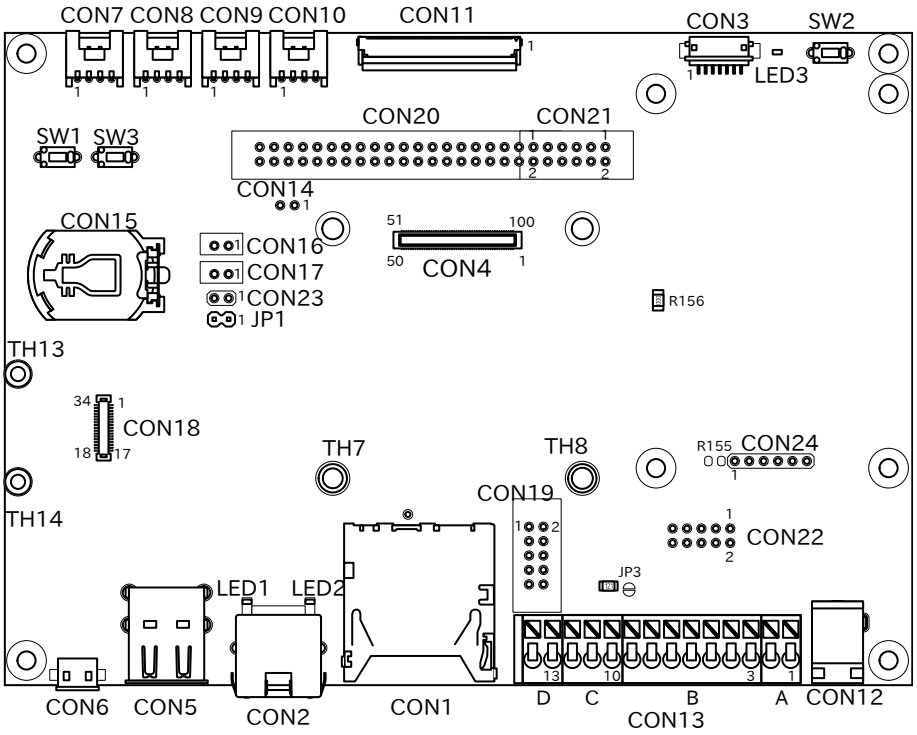


図 6.202 Armadillo-610 拡張ボードのインターフェースの概要

表 6.37 Armadillo-610 拡張ボードの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	CIM-K03NS	MITSUMI ELECTRIC
CON2	LAN インターフェース	TM11R-5M2-88-LP	HIROSE ELECTRIC
CON3	シリアルインターフェース	DF13A-7P-1.25H(51)	HIROSE ELECTRIC
CON4	Armadillo-610 インターフェース	DF40HC(3.0)-100DS-0.4V(51)	HIROSE ELECTRIC
CON5	USB ホストインターフェース	UBA-4RS-D14T-4D(LF)(SN)	J.S.T. Mfg.
CON6	USB OTG インターフェース	UB-MC5ABR3-SD204-4S-1	J.S.T. Mfg.
CON7	Grove インターフェース	1125R-4P	Shenzhen NS-TECH Co.,Ltd
CON8		1125R-4P	Shenzhen NS-TECH Co.,Ltd
CON9		1125R-4P	Shenzhen NS-TECH Co.,Ltd
CON10		1125R-4P	Shenzhen NS-TECH Co.,Ltd
CON11	LCD 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	PJ-102AH	Same Sky
CON13A	電源出力インターフェース	TBL002A-350-14GY-2GY	Same Sky
CON13B	DIDO インターフェース		
CON13C	RS-485 インターフェース		
CON13D	オーディオインターフェース		
CON14	電源出力インターフェース	A2-2PA-2.54DSA(71)	HIROSE ELECTRIC
CON15	RTC バックアップインターフェース	CH7410-2032LF	TAKACHI
CON16		B2B-EH(LF)(SN)	J.S.T. Mfg.
CON17	内蔵 RTC バックアップインターフェース	B2B-EH(LF)(SN)	J.S.T. Mfg.
CON18	WLAN インターフェース	AXK6F34347YG	Panasonic

部品番号	インターフェース名	型番	メーカー
CON19	拡張インターフェース	XG4C-1031	OMRON
CON20		XG4C-4031	OMRON
CON21		XG4C-1031	OMRON
CON22		XG4C-1031	OMRON
CON23	リセットインターフェース	A2-2PA-2.54DSA(71)	HIROSE ELECTRIC
CON24	電源入力インターフェース	A2-6PA-2.54DSA(71)	HIROSE ELECTRIC
JP1	起動デバイス設定ジャンパ	A2-2PA-2.54DSA(71)	HIROSE ELECTRIC
SW1	ユーザースイッチ	SKHLACA010	ALPS ELECTRIC
SW2	リセットスイッチ	SKHLACA010	ALPS ELECTRIC
SW3	ONOFF スイッチ	SKHLACA010	ALPS ELECTRIC
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	SML-310MTT86	ROHM
TH7	Armadillo-610 用スタッド	TH-1.6-3.0-M3	Mac-Eight
TH8		TH-1.6-3.0-M3	Mac-Eight
TH13	WLAN モジュール用スタッド	TH-1.6-1.5-M2	Mac-Eight
TH14		TH-1.6-1.5-M2	Mac-Eight

[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。



「表 6.37. Armadillo-610 拡張ボードのインターフェース一覧 」には搭載可能な代表型番を記載しており、実際に搭載されている型番と違うことがあります。

6.29.2.5. CON1 (SD インターフェース)

CON1 はハイスピード(最大クロック周波数: 49.5MHz)に対応した SD インターフェースです。信号線は i.MX6ULL の SD ホストコントローラ(uSDHC2)に接続されます。

SD カードに供給される電源は i.MX6ULL の UART2_RTS_B ピン(GPIO1_IO23)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。



SD コントローラ(uSDHC2)は SD インターフェース(Armadillo-610: CON1)でも使用しており、同時に使用することはできません。こちらの SD を有効にした場合、SD インターフェース(Armadillo-610: CON1)はブート時のみ利用され、ブート以降はこちらが利用されます。

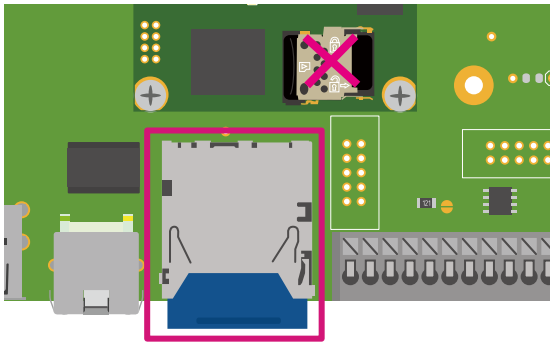


図 6.203 Armadillo-610 拡張ボード CON1

表 6.38 CON1 信号配列

ピン番号	ピン名	I/O	説明
1	CD/DAT3	In/Out	SD データバス(bit3)、i.MX6ULL の LCD_DATA23 ピンに接続
2	CMD	In/Out	SD コマンド/レスポンス、i.MX6ULL の LCD_DATA18 ピンに接続
3	VSS	Power	電源(GND)
4	VDD	Power	電源(+3.3V)
5	CLK	Out	SD クロック、i.MX6ULL の LCD_DATA19 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 LCD_DATA20 ピンに接続
8	DAT1	In/Out	SD データバス(bit1)、i.MX6ULL の LCD_DATA21 ピンに接続
9	DAT2	In/Out	SD データバス(bit2)、i.MX6ULL の LCD_DATA22 ピンに接続
10	CD1	In	カード検出、i.MX6ULL の UART3_RTS_B ピンに接続 (Low: カード挿入、High: カード未挿入)
11	CD2		
12	WP1	Power	電源(GND)
13			
14	WP2	In	ライトプロテクト検出、i.MX6ULL の UART3_CTS_B ピンに接続 (Low: 書き込み可能、High: 書き込み不可能)
15	GND	Power	電源(GND)
16			
17			
18			
19			

6.29.2.6. CON2(LAN インターフェース)

CON2 は 10BASE-T/100BASE-TX に対応した LAN インターフェースです。カテゴリ 5 以上の Ethernet ケーブルを接続することができます。AUTO-MDIX 機能を搭載しており、ストレートケーブルまたはクロスケーブルを自動認識して送受信端子を切り替えます。

信号線は Ethernet PHY(LAN8720AI-CP/Microchip Technology) を経由して i.MX6ULL の Ethernet コントローラ(ENET1)に接続されます。

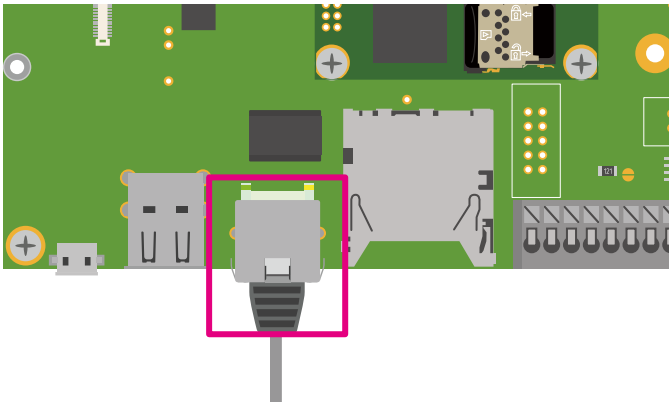


図 6.204 Armadillo-610 拡張ボード CON2

表 6.39 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(+)
2	TX-	In/Out	送信データ(-)
3	RX+	In/Out	受信データ(+)
4	—	—	CON2 の 5 ピンと接続後に 75Ω 終端

ピン番号	ピン名	I/O	説明
5	—	—	CON2 の 4 ピンと接続後に 75Ω 終端
6	RX-	In/Out	受信データ(-)
7	—	—	CON2 の 8 ピンと接続後に 75Ω 終端
8	—	—	CON2 の 7 ピンと接続後に 75Ω 終端

6.29.2.7. CON3(シリアルインターフェース)

CON3 は非同期(調歩同期)シリアルインターフェースです。信号は i.MX6ULL の UART コントローラ(UART1)に接続されます。CON3 の 6 ピンは i.MX6ULL の UART2_CTS_B ピン(GPIO1_IO22)に接続されており、Low レベル入力で保守モード、High レベル入力でオートブートモードで起動します。

シリアルインターフェース(Armadillo-610 拡張ボード: CON3)に USB シリアル変換アダプタを接続する際は、ケーブルの根本を軽く握り、指先でコネクタを押すようにして挿入してください。取り外しの際は、全ケーブルが均等に引きぬかれるようにケーブルをつかみ、引き抜いてください。また、両コネクタを水平にして挿入・抜去してください。30°以上傾けた状態での斜め挿入・抜去は、端子変形、ケース破損の原因となります。

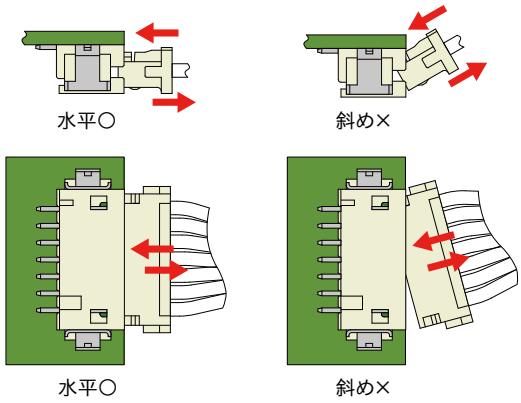


図 6.205 USB シリアル変換アダプタの挿抜角度

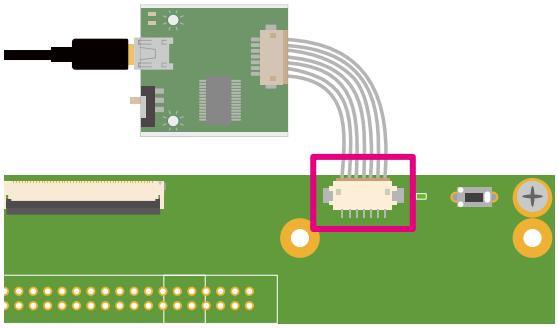


図 6.206 Armadillo-610 拡張ボード CON3

表 6.40 CON3 信号配列

ピン番号	ピン名	I/O	説明
1	UART_RXD	In	受信データ、i.MX6ULL の UART1_RX_DATA ピンに接続

ピン番号	ピン名	I/O	説明
2	GND	Power	電源(GND)
3	UART_TXD	Out	送信データ、i.MX6ULL の UART1_TX_DATA ピンに接続
4	+3.3V_IO	Power	電源(+3.3V_IO)
5	UART_CTS	In	送信可能、CON3 の 7 ピンと接続
6	BOOTLOADER_EN_B	In	起動モード設定、i.MX6ULL の UART2_CTS_B ピンに接続 (Low: 保守モード、High: オートブートモード)
7	UART_RTS	Out	送信要求、CON3 の 5 ピンと接続

6.29.2.8. CON4(Armadillo-610 インターフェース)

CON4 は Armadillo-610 と接続するためのインターフェースです。

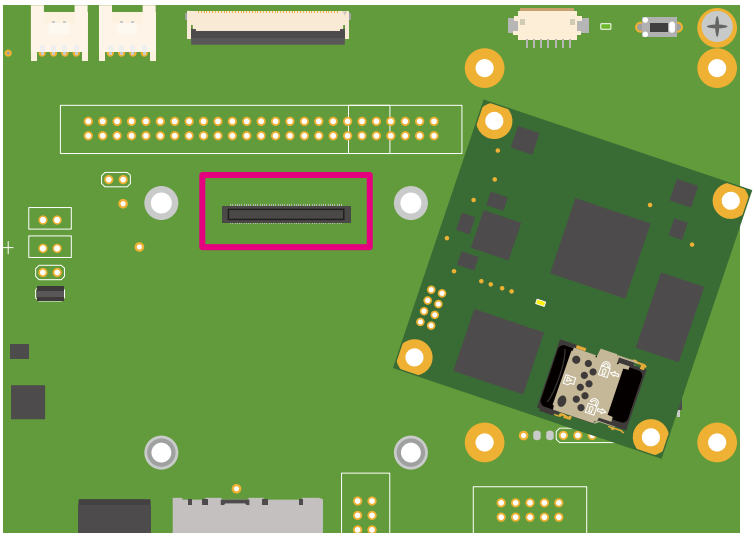


図 6.207 Armadillo-610 拡張ボード CON4

表 6.41 CON4 信号配列

ピン番号	ピン名	I/O	説明
1	USB_OTG1_DP	In/Out	USB1 のプラス側信号、CON6 の 3 ピンに接続
2	USB_OTG1_DN	In/Out	USB1 のマイナス側信号、CON6 の 2 ピンに接続
3	GND	Power	電源(GND)
4	USB_OTG2_DN	In/Out	USB2 のマイナス側信号、USB HUB の USBUP_DM ピンに接続
5	USB_OTG2_DP	In/Out	USB2 のプラス側信号、USB HUB の USBUP_DP ピンに接続
6	GND	Power	電源(GND)
7	USB_OTG1_VBUS	Power	電源(USB_OTG1_VBUS)、CON6 の 1 ピンに接続
8	USB_OTG2_VBUS	Power	電源(USB_OTG2_VBUS)、CON5 の 1 ピンと 5 ピンに接続
9	SPEEDLED	In	LAN スピード LED 用信号、Ethernet PHY の LED2 ピンに接続
10	LINK_ACTLED	In	LAN リンクアクティビティ LED 用信号、Ethernet PHY の LED1 ピンに接続
11	USB1_PWREN	In	USB1 用パワースイッチ切り替え信号、パワースイッチのイネーブルピンに接続 (High: 電源供給、Low: 電源切断)
12	USB2_PWREN	In	USB2 用パワースイッチ切り替え信号、パワースイッチのイネーブルピンに接続 (High: 電源供給、Low: 電源切断)

ピン番号	ピン名	I/O	説明
13	RTC_INT_B	Out	リアルタイムクロック割り込み信号、リアルタイムクロックの割り込みピンに接続
14	ADC_IN4	In/Out	拡張入出力、CON11 の 34 ピン、CON20 の 37 ピン、CON10 の 1 ピンに接続
15	ADC_IN3	In/Out	拡張入出力、CON11 の 35 ピン、CON20 の 38 ピン、CON9 の 1 ピンに接続
16	ADC_IN2	In/Out	拡張入出力、CON11 の 36 ピン、CON20 の 39 ピン、CON9 の 2 ピンに接続
17	ADC_IN1	In/Out	拡張入出力、CON11 の 37 ピン、CON20 の 40 ピン、CON10 の 2 ピンに接続
18	LCD_DATA00	In/Out	拡張入出力、CON11 の 13 ピン、CON20 の 13 ピンに接続
19	LCD_DATA01	In/Out	拡張入出力、CON11 の 14 ピン、CON20 の 14 ピンに接続
20	LCD_DATA02	In/Out	拡張入出力、CON11 の 15 ピン、CON20 の 15 ピンに接続
21	LCD_DATA03	In/Out	拡張入出力、CON11 の 16 ピン、CON20 の 16 ピンに接続
22	LCD_DATA04	In/Out	拡張入出力、CON11 の 17 ピン、CON20 の 17 ピンに接続
23	LCD_DATA05	In/Out	拡張入出力、CON11 の 18 ピン、CON20 の 18 ピンに接続
24	LCD_DATA06	In/Out	拡張入出力、CON11 の 20 ピン、CON20 の 20 ピンに接続
25	LCD_DATA07	In/Out	拡張入出力、CON11 の 21 ピン、CON20 の 21 ピンに接続
26	LCD_DATA08	In/Out	拡張入出力、CON11 の 22 ピン、CON20 の 22 ピンに接続
27	LCD_DATA09	In/Out	拡張入出力、CON11 の 23 ピン、CON20 の 23 ピンに接続
28	LCD_DATA10	In/Out	拡張入出力、CON11 の 24 ピン、CON20 の 24 ピンに接続
29	LCD_DATA11	In/Out	拡張入出力、CON11 の 25 ピン、CON20 の 25 ピンに接続
30	LCD_DATA12	In/Out	拡張入出力、CON11 の 27 ピン、CON20 の 27 ピンに接続
31	LCD_DATA13	In/Out	拡張入出力、CON11 の 28 ピン、CON20 の 28 ピンに接続
32	LCD_DATA14	In/Out	拡張入出力、CON11 の 29 ピン、CON20 の 29 ピンに接続
33	LCD_DATA15	In/Out	拡張入出力、CON11 の 30 ピン、CON20 の 30 ピンに接続
34	LCD_DATA16	In/Out	拡張入出力、CON11 の 31 ピン、CON20 の 31 ピンに接続
35	LCD_DATA17	In/Out	拡張入出力、CON11 の 32 ピン、CON20 の 32 ピンに接続
36	GND	Power	電源(GND)
37	LCD_CLK	In/Out	拡張入出力、CON11 の 8 ピン、CON20 の 7 ピンに接続
38	LCD_HSYNC	In/Out	拡張入出力、CON11 の 9 ピン、CON20 の 8 ピンに接続
39	LCD_VSYNC	In/Out	拡張入出力、CON11 の 10 ピン、CON20 の 9 ピンに接続
40	LCD_ENABLE	In/Out	拡張入出力、CON11 の 11 ピン、CON20 の 10 ピンに接続
41	PWM5_OUT	In/Out	拡張入出力、CON11 の 12 ピン、CON20 の 11 ピンに接続
42	BJP1	Out	起動デバイス設定用信号、JP1 に接続
43	EXT_SW1	Out	ユーザースイッチ、SW1 に接続
44	EXT_RESET_B	Out	システムリセット、CON23 の 1 ピンに接続
45	+3.3V_IO	Power	電源(+3.3V_IO)
46	+3.3V_IO	Power	電源(+3.3V_IO)
47	VIN	Power	電源(VIN)
48	VIN	Power	電源(VIN)
49	VIN	Power	電源(VIN)
50	VIN	Power	電源(VIN)
51	GND	Power	電源(GND)
52	GND	Power	電源(GND)
53	+5V_IO	Power	電源(+5V_IO)
54	+5V_IO	Power	電源(+5V_IO)
55	I2C2_SDA	In/Out	I2C データ信号、CON11 の 49 ピン、CON21 の 10 ピン、CON8 の 2 ピン、リアルタイムクロックの SDA ピンに接続
56	I2C2_SCL	In	I2C クロック信号、CON11 の 48 ピン、CON21 の 9 ピン、CON8 の 1 ピン、リアルタイムクロックの SCL ピンに接続
57	SAI1_TX_SYNC	In/Out	拡張入出力、CON11 の 47 ピン、CON21 の 8 ピンに接続
58	SAI1_TX_BCLK	In/Out	拡張入出力、CON11 の 46 ピン、CON21 の 7 ピンに接続
59	SAI1_RX_DATA	In/Out	拡張入出力、CON11 の 45 ピン、CON21 の 6 ピンに接続
60	SAI1_TX_DATA	In/Out	拡張入出力、CON11 の 44 ピン、CON21 の 5 ピンに接続

ピン番号	ピン名	I/O	説明
61	SAI1_RX_SYNC	In/Out	拡張入出力、CON11 の 43 ピン、CON21 の 4 ピンに接続
62	SAI1_MCLK	In/Out	拡張入出力、CON11 の 42 ピン、CON21 の 3 ピン、CON7 の 1 ピンに接続
63	GPIO4_IO24	In/Out	拡張入出力、CON11 の 41 ピン、CON20 の 36 ピンに接続
64	GPIO4_IO21	In/Out	拡張入出力、CON11 の 40 ピン、CON20 の 35 ピンに接続
65	GPIO4_IO18	In/Out	拡張入出力、CON11 の 39 ピン、CON20 の 34 ピンに接続
66	RS485_DE	In	RS-485 送信イネーブル信号、RS-485 トランシーバの DE ピン、CON22 の 6 ピンに接続
67	RS485_RE_N	In	RS-485 受信イネーブル信号、RS-485 トランシーバの RE ピン、CON22 の 5 ピンに接続
68	RS485_RX	Out	RS-485 受信データ、RS-485 トランシーバの RO ピン、CON22 の 4 ピンに接続
69	RS485_TX	In	RS-485 送信データ、RS-485 トランシーバの DI ピン、CON22 の 3 ピンに接続
70	SD2_DATA3	In/Out	SD データバス(bit3)、CON1 の 1 ピンに接続
71	SD2_DATA2	In/Out	SD データバス(bit2)、CON1 の 9 ピンに接続
72	SD2_DATA1	In/Out	SD データバス(bit1)、CON1 の 8 ピンに接続
73	SD2_DATA0	In/Out	SD データバス(bit0)、CON1 の 7 ピンに接続
74	SD2_CLK	In	SD クロック、CON1 の 5 ピンに接続
75	GND	Power	電源(GND)
76	SD2_CMD	In/Out	SD コマンド/レスポンス、CON1 の 2 ピンに接続
77	PWRON	Out	パワーマネジメント IC の PWRON 信号、SW2 に接続
78	ONOFF	Out	i.MX6ULL の ON/OFF 信号、SW3 に接続
79	RTC_BAT	Power	電源(RTC_BAT)、CON17 の 1 ピンに接続
80	EXT_LED1	In	LED3 に接続(High: 点灯、Low: 消灯)
81	AMP_SD_B	In	オーディオアンプのシャットダウンピンに接続 (High: オーディオ開始、Low: オーディオ停止)
82	DO1	In	CON13 の DO1 制御ピンに接続 (High: DO1 ショート、Low: DO1 オープン)
83	DEBUG_UART_TX	In	送信データ、CON3 の 3 ピンに接続
84	DO2	In	CON13 の DO 制御ピンに接続 (High: DO2 ショート、Low: DO2 オープン)
85	DEBUG_UART_RX	Out	受信データ、CON3 の 1 ピンに接続
86	SD_PWREN	In	SD 用パワースイッチ切り替え信号、パワースイッチのイネーブルピンに接続 (High: 電源供給、Low: 電源切断)
87	MAINT_EN_B	Out	起動モード設定、CON3 の 6 ピンに接続
88	DI2	Out	DI2 入力、CON13 の 4 ピンに接続
89	DI1	Out	DI1 入力、CON13 の 5 ピンに接続
90	MQS	In	オーディオ入力、オーディオアンプに接続
91	SD2_WP	Out	ライトプロテクト検出、CON1 の 14 ピン、CON19 の 10 ピンに接続
92	SD2_CD_B	Out	カード検出、CON1 の 10 ピン、CON19 の 9 ピンに接続
93	WLAN_PWREN	In	WLAN 用パワースイッチ切り替え信号、パワースイッチのイネーブルピンに接続 (High: 電源供給、Low: 電源切断)
94	USB1_OTG_ID	Out	CON6 の 4 ピンに接続
95	GND	Power	電源(GND)
96	Ether_RXN	In/Out	Ethernet 送信/受信データ(-) CH2、Ethernet トランスに接続
97	Ether_RXP	In/Out	Ethernet 送信/受信データ(+) CH2、Ethernet トランスに接続
98	GND	Power	電源(GND)
99	Ether_TXN	In/Out	Ethernet 送信/受信データ(-) CH1、Ethernet トランスに接続
100	Ether_TXP	In/Out	Ethernet 送信/受信データ(+) CH1、Ethernet トランスに接続

6.29.2.9. CON5(USB ホストインターフェース)

CON5 は USB ホストインターフェースです。2 段のコネクタを実装しており、信号線は USB HUB を経由して i.MX6ULL の USB コントローラ(USB OTG2)に接続されます。

供給される電源は 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。

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

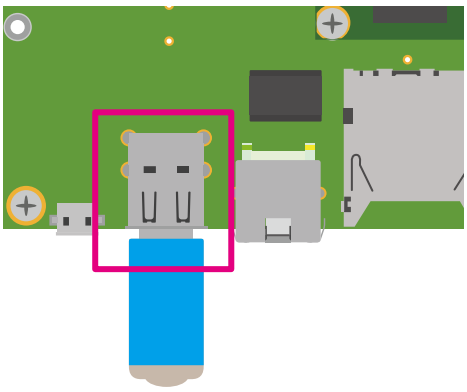


図 6.208 Armadillo-610 拡張ボード CON5

表 6.42 CON5 信号配列

ピン番号	ピン名	I/O	説明
1	+5V	Power	電源(+5V)
2	USB_L_DN	In/Out	USB 下段のマイナス側信号、USB HUB(Port2)を経由して i.MX6ULL の USB_OTG2_DN ピンに接続
3	USB_L_DP	In/Out	USB 下段のプラス側信号、USB HUB(Port2)を経由して i.MX6ULL の USB_OTG2_DP ピンに接続
4	GND	Power	電源(GND)
5	+5V	Power	電源(+5V)
6	USB_U_DN	In/Out	USB 上段のマイナス側信号、USB HUB(Port3)を経由して i.MX6ULL の USB_OTG2_DN ピンに接続
7	USB_U_DP	In/Out	USB 上段のプラス側信号、USB HUB(Port3)を経由して i.MX6ULL の USB_OTG2_DP ピンに接続
8	GND	Power	電源(GND)

6.29.2.10. CON6(USB OTG インターフェース)

CON6 は USB OTG インターフェースです。信号線は i.MX6ULL の USB コントローラ(USB OTG1)に接続されます。

供給される電源は i.MX6ULL の UART1_RTS_B ピン(GPIO1_IO19)および CON6 の 4 ピン(USB_ID)により制御が可能です。i.MX6ULL のの UART1_RTS_B ピン(GPIO1_IO19)から High レベル出力かつ CON6 の 4 ピン(USB_ID)から Low レベル入力で電源が供給され、CON6 の 4 ピン(USB_ID)がオープンで電源切断されます。

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

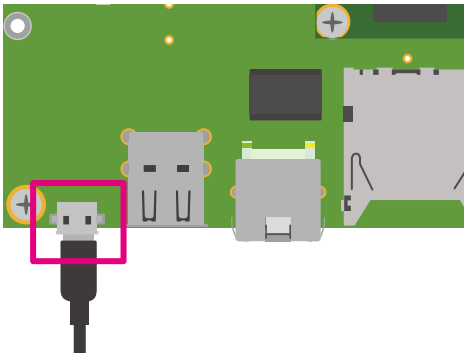


図 6.209 Armadillo-610 拡張ボード CON6

表 6.43 CON6 信号配列

ピン番号	ピン名	I/O	説明
1	+5V	Power	電源(+5V_IO)
2	USB_DN	In/Out	USB のマイナス側信号、USB HUB を経由して i.MX6ULL の USB_OTG1_DN ピンに接続
3	USB_DP	In/Out	USB のプラス側信号、USB HUB を経由して i.MX6ULL の USB_OTG1_DP ピンに接続
4	USB_ID	In	USB の ID 信号、i.MX6ULL の GPIO1_IO24 ピンに接続
5	GND	Power	電源(GND)

6.29.2.11. CON7、CON8、CON9、CON10(Grove インターフェース)

CON7、CON8、CON9、CON10 は Seeed 社が推奨するコネクタ規格「Grove システム」に対応した Grove モジュール接続用のインターフェースです。

マルチプレクスの設定で機能を割り当てることで、GPIO、UART、I2C、A/D で拡張する Grove モジュールを接続することができます。



CON11(LCD インターフェース)、CON20(拡張インターフェース)、CON21(拡張インターフェース)と共通の信号線が接続されているため、同時に使用できません。また、CON8 の I2C 信号は基板上のリアルタイムクロックにも接続されておりますので、マルチプレクスの設定を変更する際には、ご注意ください。

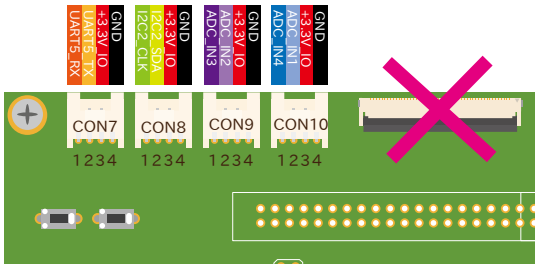


図 6.210 Armadillo-610 拡張ボード CON7、CON8、CON9、CON10

表 6.44 CON7 信号配列

ピン番号	ピン名	I/O	説明
1	UART5_RX	I/O	拡張入出力、i.MX6ULL の CSI_DATA01 ピンに接続
2	UART5_TX	I/O	拡張入出力、i.MX6ULL の CSI_DATA00 ピンに接続
3	+3.3V_IO	Power	電源(+3.3V_IO)
4	GND	Power	電源(GND)

表 6.45 CON8 信号配列

ピン番号	ピン名	I/O	説明
1	I2C2_CLK	I/O	拡張入出力、i.MX6ULL の CSI_HSYNC ピンに接続
2	I2C2_SDA	I/O	拡張入出力、i.MX6ULL の CSI_VSYNC ピンに接続
3	+3.3V_IO	Power	電源(+3.3V_IO)
4	GND	Power	電源(GND)

表 6.46 CON9 信号配列

ピン番号	ピン名	I/O	説明
1	ADC_IN3	I/O	拡張入出力、i.MX6ULL の GPIO1_IO03 ピンに接続
2	ADC_IN2	I/O	拡張入出力、i.MX6ULL の GPIO1_IO02 ピンに接続
3	+3.3V_IO	Power	電源(+3.3V_IO)
4	GND	Power	電源(GND)

表 6.47 CON10 信号配列

ピン番号	ピン名	I/O	説明
1	ADC_IN4	I/O	拡張入出力、i.MX6ULL の GPIO1_IO04 ピンに接続
2	ADC_IN1	I/O	拡張入出力、i.MX6ULL の GPIO1_IO01 ピンに接続
3	+3.3V_IO	Power	電源(+3.3V_IO)
4	GND	Power	電源(GND)

6.29.2.12. CON11(LCD インターフェース)

CON11 はデジタル RGB 入力を持つ液晶パネルモジュールなどを接続することができる、LCD インターフェースです。信号線は i.MX6ULL の LCD インターフェース(eLCDIF)等に接続されます。

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-00)、Armadillo-400 シリーズ LCD オプションセット(4.3 インチタッチパネル WQVGA 液晶)(型番: OP-A400-LCD43EXT-L01)を接続可能です。

オプションセットの詳細につきましては「6.29.3. LCD オプションセット(7 インチタッチパネル WVGA 液晶)」、「6.29.4. Armadillo-400 シリーズ LCD オプションセット」をご確認ください。

LCD オプションセット(7 インチタッチパネル WVGA 液晶)を使用する場合、I2C2_SCL 信号がバックライト用の PWM 信号として使用されるため、基板上のリアルタイムクロックが使用できなくなります。

Armadillo-400 シリーズ LCD オプションセットを使用する場合、LCD オプションセット側にもリアルタイムクロックが接続されており、アドレスが被っているため、アクセスすると信号が衝突します。リアルタイムクロックにアクセスしない、もしくはどちらかのリアルタイムクロックを切り離してご使用ください。Armadillo-610 拡張ボード側は R146、R147(基板裏)の抵抗を未実装にすることでリアルタイムクロックを切り離すことが可能です。

CON7、CON8、CON9、CON10(Grove インターフェース)、CON20(拡張インターフェース)、CON21(拡張インターフェース)と共通の信号線が接続されているため、同時に使用できません。また、48、49 ピンの信号線は基板上のリアルタイムクロックにも接続されており、I2C で使用しておりますので、マルチプレクスの設定を変更する際には、ご注意ください。

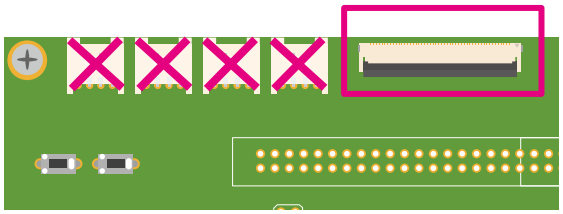


図 6.211 Armadillo-610 拡張ボード CON11

表 6.48 CON11 信号配列

ピン番号	ピン名	I/O	説明
1	+5V	Power	電源出力(+5V)
2	+5V	Power	電源出力(+5V)
3	+5V	Power	電源出力(+5V)
4	+3.3V	Power	電源出力(+3.3V)
5	+3.3V	Power	電源出力(+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

ピン番号	ピン名	I/O	説明
14	LCD_DATA01	In/Out	拡張入出力、i.MX6ULL の LCD_DATA01 ピンに接続
15	LCD_DATA02	In/Out	拡張入出力、i.MX6ULL の LCD_DATA02 ピンに接続
16	LCD_DATA03	In/Out	拡張入出力、i.MX6ULL の LCD_DATA03 ピンに接続
17	LCD_DATA04	In/Out	拡張入出力、i.MX6ULL の LCD_DATA04 ピンに接続
18	LCD_DATA05	In/Out	拡張入出力、i.MX6ULL の LCD_DATA05 ピンに接続
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
21	LCD_DATA07	In/Out	拡張入出力、i.MX6ULL の LCD_DATA07 ピンに接続
22	LCD_DATA08	In/Out	拡張入出力、i.MX6ULL の LCD_DATA08 ピンに接続
23	LCD_DATA09	In/Out	拡張入出力、i.MX6ULL の LCD_DATA09 ピンに接続
24	LCD_DATA10	In/Out	拡張入出力、i.MX6ULL の LCD_DATA10 ピンに接続
25	LCD_DATA11	In/Out	拡張入出力、i.MX6ULL の LCD_DATA11 ピンに接続
26	GND	Power	電源(GND)
27	LCD_DATA12	In/Out	拡張入出力、i.MX6ULL の LCD_DATA12 ピンに接続
28	LCD_DATA13	In/Out	拡張入出力、i.MX6ULL の LCD_DATA13 ピンに接続
29	LCD_DATA14	In/Out	拡張入出力、i.MX6ULL の LCD_DATA14 ピンに接続
30	LCD_DATA15	In/Out	拡張入出力、i.MX6ULL の LCD_DATA15 ピンに接続
31	LCD_DATA16	In/Out	拡張入出力、i.MX6ULL の LCD_DATA16 ピンに接続
32	LCD_DATA17	In/Out	拡張入出力、i.MX6ULL の LCD_DATA17 ピンに接続
33	GND	Power	電源(GND)
34	ADC_IN4	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	ADC_IN3	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	ADC_IN2	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	ADC_IN1	In/Out	拡張入出力、i.MX6ULL の GPIO1_IO01 ピンに接続、0.01uF のコンデンサが接続されています。
38	GND	Power	電源(GND)
39	GPIO4_IO18	In/Out	拡張入出力、i.MX6ULL の CSI_PIXCLK ピンに接続
40	GPIO4_IO21	In/Out	拡張入出力、i.MX6ULL の CSI_DATA00 ピンに接続
41	GPIO4_IO24	In/Out	拡張入出力、i.MX6ULL の CSI_DATA03 ピンに接続
42	SAI1_MCLK	In/Out	拡張入出力、i.MX6ULL の CSI_DATA01 ピンに接続
43	SAI1_RX_SYNC	In/Out	拡張入出力、i.MX6ULL の CSI_DATA02 ピンに接続
44	SAI1_TX_DATA	In/Out	拡張入出力、i.MX6ULL の CSI_DATA07 ピンに接続
45	SAI1_RX_DATA	In/Out	拡張入出力、i.MX6ULL の CSI_DATA06 ピンに接続
46	SAI1_TX_BCLK	In/Out	拡張入出力、i.MX6ULL の CSI_DATA05 ピンに接続
47	SAI1_TX_SYNC	In/Out	拡張入出力、i.MX6ULL の CSI_DATA04 ピンに接続
48	I2C2_SCL	In/Out	拡張入出力、i.MX6ULL の CSI_HSYNC ピンに接続
49	I2C2_SDA	In/Out	拡張入出力、i.MX6ULL の CSI_VSYNC ピンに接続
50	GND	Power	電源(GND)

6.29.2.13. CON12(電源入力インターフェース)

CON12 は電源供給用のインターフェースです。DC ジャックが実装されており、「図 6.212. 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。



図 6.212 AC アダプタの極性マーク

AC アダプタから電源を供給する際、DC プラグを Armadillo-610 拡張ボードの DC ジャックに接続してから、AC プラグをコンセントに接続してください。突入電流により、故障する可能性があります。

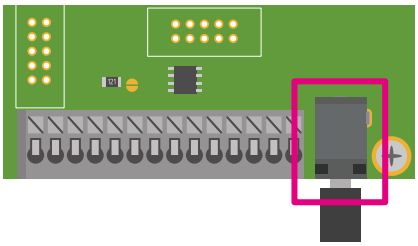


図 6.213 Armadillo-610 拡張ボード CON12

表 6.49 CON12 信号配列

ピン番号	ピン名	I/O	説明
1	EXT_VIN	Power	電源(EXT_VIN)
2	GND	Power	電源(GND)
3	GND	Power	電源(GND)

6.29.2.14. CON13A(電源出力インターフェース)

CON13A は電源出力用インターフェースです。端子台が実装されています。CON12 から入力した電源が出力されます。

電源入力として使用することも可能ですが、同時に CON12 から電源供給しないようご注意ください。

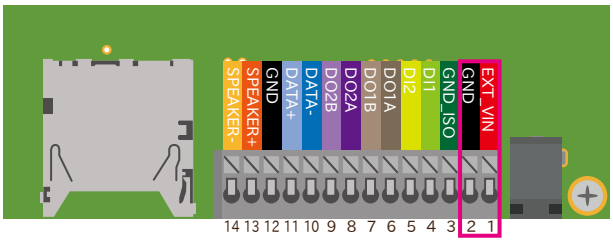


図 6.214 CON13A の配置

表 6.50 CON13A 信号配列

ピン番号	ピン名	I/O	説明
1	EXT_VIN	Power	電源(EXT_VIN)
2	GND	Power	電源(GND)

6.29.2.15. CON13B(DIDO インターフェース)

CON13B は絶縁デジタル入出力インターフェースです。

デジタル入力部はフォトカプラによる絶縁入力となっています。入力部を駆動するためには外部に電源(定格電圧 DC 3.3～12V)が必要となります。

デジタル出力部はフォトリレーによる絶縁出力(無極性)となっています。出力部を駆動するためには外部に電源が必要となります。出力 1 点につき最大電流 200mA(最大電圧 DC 48V)まで駆動可能です。



動作確認時には CON13A(電源出力インターフェース)から電源を取るのが便利です。

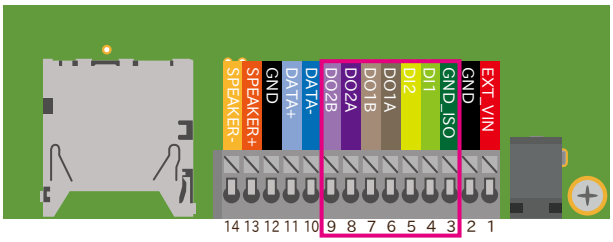


図 6.215 CON13B の配置

表 6.51 CON13B 信号配列

ピン番号	ピン名	I/O	説明
3	GND_ISO	Power	電源(GND_ISO)
4	DI1	In	デジタル入力 1
5	DI2	In	デジタル入力 2
6	DO1A	―	デジタル出力 1A
7	DO1B	―	デジタル出力 1B
8	DO2A	―	デジタル出力 2A
9	DO2B	―	デジタル出力 2B



CON13 の 3 ピンの GND_ISO は絶縁されています。

6.29.2.16. CON13C(RS-485 インターフェース)

CON13C は RS-485(半二重)のシリアルインターフェースです。信号線は RS-485 トランシーバを経由して、i.MX6ULL の UART コントローラ(UART2)に接続されています。

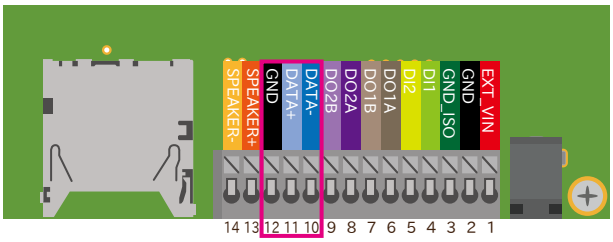


図 6.216 CON13C の配置

表 6.52 CON13C 信号配列

ピン番号	ピン名	I/O	説明
10	DATA-	In/Out	送受信データ(-)
11	DATA+	In/Out	送受信データ(+)
12	GND	Power	電源(GND)

6.29.2.17. CON13D(オーディオインターフェース)

CON13D はモノラルのオーディオ出力インターフェースです。1.4W オーディオアンプを経由して i.MX6ULL の Medium Quality Sounc(MQS)に接続されています。8Ω スピーカーが接続可能です。

Armadillo-610 開発セットに付属しているのは、8Ω スピーカーです。このスピーカーを直接端子台に取り付ける場合、8〜9mm ワイヤーストリッパー等で剥いてください。剥かずに取り付けた場合、音がでない等のトラブルの原因となる可能性があります。

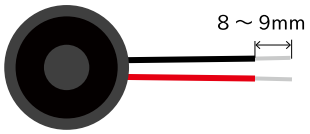


図 6.217 スピーカーのリード線

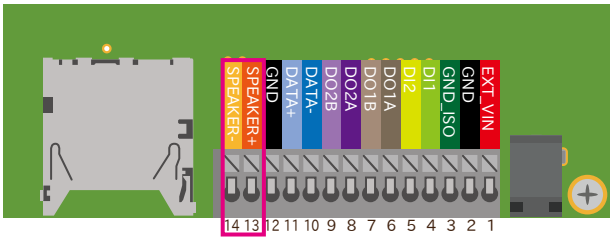


図 6.218 CON13D の配置

表 6.53 CON13D 信号配列

ピン番号	ピン名	I/O	説明
13	SPEAKER+	Out	スピーカー出力(+)
14	SPEAKER-	Out	スピーカー出力(-)

6.29.2.18. CON14(電源出力インターフェース)

CON14 は電源出力インターフェースです。Armadillo-610 で生成する+5V_IO が接続されています。

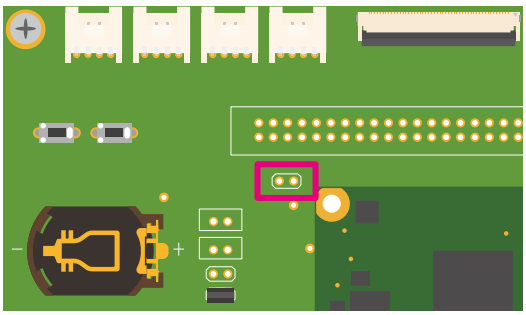


図 6.219 Armadillo-610 拡張ボード CON14

表 6.54 CON14 信号配列

ピン番号	ピン名	I/O	説明
1	+5V_IO	Power	電源(+5V_IO)
2	GND	Power	電源(GND)

6.29.2.19. CON15、CON16(RTC バックアップインターフェース)

CON15、CON16 は Armadillo-610 拡張ボードに搭載しているリアルタイムクロックのバックアップ用インターフェースです。

別途バックアップ用の電源を接続することで、Armadillo-610 の電源(VIN)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。

Armadillo-610 拡張ボードに搭載しているリアルタイムクロックは、i.MX6ULL 内蔵のリアルタイムクロックよりも消費電力が少なく、精度が良いものとなっております。

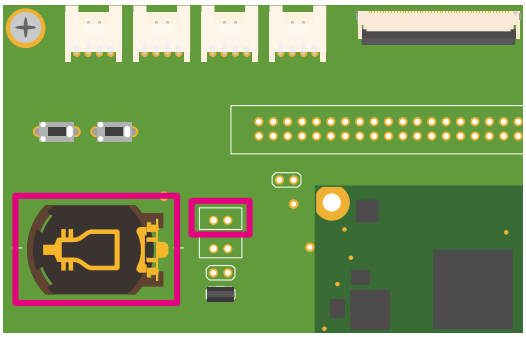


図 6.220 Armadillo-610 拡張ボード CON15、CON16

表 6.55 CON16 信号配列

ピン番号	ピン名	I/O	説明
1	EXT_RTC_BAT	Power	電源(EXT_RTC_BAT)
2	GND	Power	電源(GND)

表 6.56 CON15、CON16 対応電池の例とバックアップ時間

対応電池	バックアップ時間(参考値)
CR2032	6.2 年



CON15、CON16 は共通の端子に接続されており、同時に使用できません。



リアルタイムクロックの平均月差は周囲温度 25℃で±8 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。



CON15、CON16 はリチウムコイン電池からの電源供給を想定しています。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。

6.29.2.20. CON17(内蔵 RTC バックアップインターフェース)

CON17 は i.MX6ULL の低消費電力ドメインにある SRTC(Secure Real Time Clock)のデータ等を保持するためのバックアップ用インターフェースです。

別途バックアップ用の電源を接続することで、Armadillo-610 の電源(VIN)が切断された場合でも、データを保持することが可能です。

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には、以下を参照してください。

- RTC_BAT ピンからバックアップ電源を切り離し、5V 電源を ON にする。
- ONOFF ピンの制御による電源の ON

詳細は、「3.4.6.6. 外部からの電源制御」参照してください。

- RTC(i.MX6ULL)のアラーム割り込みによる電源の ON

詳細は、「6.2.10.2. 起床要因を有効化する」を参照してください。

- RTC(NR3225SA)のアラーム割り込みによる電源の ON

詳細は、「6.2.10.2. 起床要因を有効化する」を参照してください。

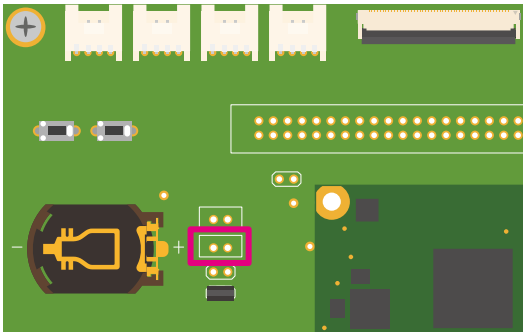


図 6.221 Armadillo-610 拡張ボード CON17

表 6.57 CON17 信号配列

ピン番号	ピン名	I/O	説明
1	RTC_BAT	Power	電源(RTC_BAT)、Armadillo-610 のパワーマネジメント IC の LICELL ピンに接続
2	GND	Power	電源(GND)

表 6.58 CON17 対応電池の例とバックアップ時間

対応電池	バックアップ時間(参考値)
CR2032	約 4 か月 [a]

[a]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、外付けバッテリの消耗が速くなります。

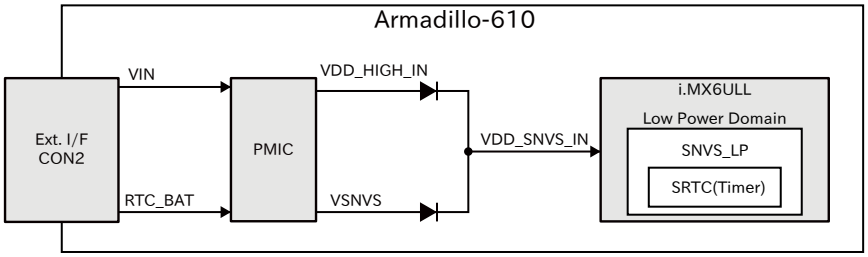


図 6.222 バックアップ電源供給回路

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、Armadillo-610 への供給電源(VIN)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 2.75～3.3V 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℃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。

6.29.2.21. CON18(WLAN インターフェース)

CON18 は Armadillo-WLAN モジュール(AWL13)接続用のインターフェースです。AWL13 は Armadillo Base OS で非対応の為、利用することができません。

無線 LAN 機能を拡張する場合、動作確認済みデバイスである、Ezurio 製の Sterling LWB5+での拡張がおすすめです。詳細につきましては、「3.4.5.4. 無線 LAN/BT」をご確認ください。



Armadillo-WLAN モジュール(AWL13)は販売を終了いたしました。詳細は変更通知: No.2023-001 をご確認ください。

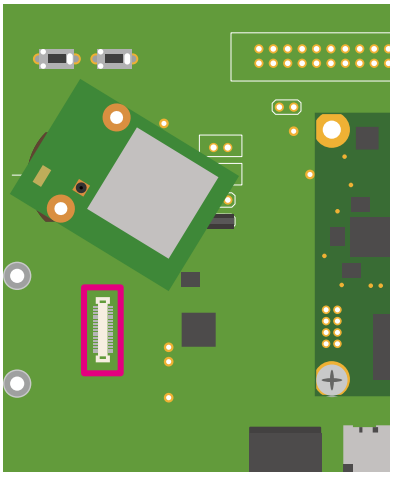


図 6.223 Armadillo-610 拡張ボード CON18

表 6.59 CON18 信号配列

ピン番号	ピン名	I/O	説明
1	SDDATA1	-	未接続
2	SDDATA0	-	未接続
3	GND	Power	電源(GND)
4	GND	Power	電源(GND)
5	USB_DM	In/Out	USB マイナス側信号
6	USB_DP	In/Out	USB プラス側信号
7	SDCLK	-	未接続
8	+3.3V	Power	電源(+3.3V)
9	NC	-	未接続
10	SDCMD	-	未接続
11	SDDATA3	-	未接続
12	SDDATA2	-	未接続
13	UART_RXD	-	未接続
14	UART_TXD	-	未接続
15	BOOT_SEL1	Out	起動モード設定、USB 起動モードに設定
16	BOOT_SEL0	Out	
17	HOST_SEL	Out	
18	FLASH_RXD	-	未接続
19	FLASH_CSB	-	未接続
20	FLASH_CLK	-	未接続

ピン番号	ピン名	I/O	説明
21	FLASH_TXD	Out	47kΩ プルダウン
22	FLASH_SEL	-	未接続
23	GPIO0	-	未接続
24	GPIO1	-	未接続
25	M_ANA	-	未接続
26	GPIO2	-	未接続
27	GPIO6	-	未接続
28	HRST	Out	+3.3V に接続
29	PRST	-	未接続
30	TMS	-	未接続
31	TCK	-	未接続
32	TDI	-	未接続
33	TDO	-	未接続
34	TRSTB	-	未接続

6.29.2.22. CON19(拡張インターフェース)

CON19 は SD 用の機能を割り当て可能な信号線を接続した、SD 拡張用のインターフェースです。

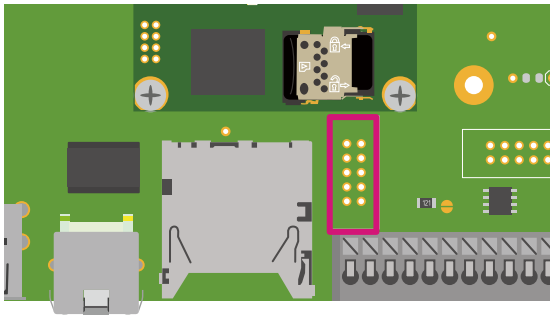


図 6.224 Armadillo-610 拡張ボード CON19



CON1(SD インターフェース)と共通の信号線が接続されているため、同時に使用できません。R78～R83、R86、R87(CON1 周辺、R79 以外は基板裏)の抵抗を未実装にすることで、CON1(SD インターフェース)と切り離すことが可能です。

表 6.60 CON19 信号配列

ピン番号	ピン名	I/O	説明
1	+3.3V_IO	Power	電源(+3.3V_IO)
2	GND	Power	電源(GND)
3	SD2_CLK	In/Out	拡張入出力、i.MX6ULL の LCD_DATA19 ピンに接続
4	SD2_CMD	In/Out	拡張入出力、i.MX6ULL の LCD_DATA18 ピンに接続
5	SD2_DATA0	In/Out	拡張入出力、i.MX6ULL の LCD_DATA20 ピンに接続
6	SD2_DATA1	In/Out	拡張入出力、i.MX6ULL の LCD_DATA21 ピンに接続
7	SD2_DATA2	In/Out	拡張入出力、i.MX6ULL の LCD_DATA22 ピンに接続
8	SD2_DATA3	In/Out	拡張入出力、i.MX6ULL の LCD_DATA23 ピンに接続
9	SD2_CD_B	In/Out	拡張入出力、i.MX6ULL の UART3_RTS_B ピンに接続
10	SD2_WP	In/Out	拡張入出力、i.MX6ULL の UART3_CTS_B ピンに接続

6.29.2.23. CON20(拡張インターフェース)

CON20 は主に LCD やタッチパネル用の機能を割り当て可能な信号線を接続した、LCD 拡張用インターフェースです。



CON7、CON9、CON10(Grove インターフェース)、CON11(LCD インターフェース)と共通の信号線が接続されているため、同時に使用できません。

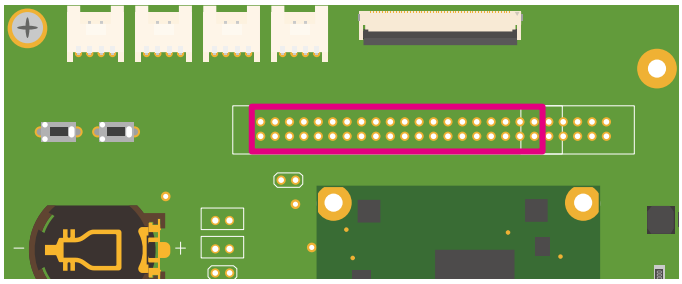


図 6.225 Armadillo-610 拡張ボード CON20

表 6.61 CON20 信号配列

ピン番号	ピン名	I/O	説明
1	+5V	Power	電源出力(+5V)
2	+5V	Power	電源出力(+5V)
3	+3.3V	Power	電源出力(+3.3V)
4	+3.3V	Power	電源出力(+3.3V)
5	GND	Power	電源(GND)
6	GND	Power	電源(GND)
7	LCD_CLK	In/Out	拡張入出力、i.MX6ULL の LCD_CLK ピンに接続
8	LCD_HSYNC	In/Out	拡張入出力、i.MX6ULL の LCD_HSYNC ピンに接続
9	LCD_VSYNC	In/Out	拡張入出力、i.MX6ULL の LCD_VSYNC ピンに接続
10	LCD_ENABLE	In/Out	拡張入出力、i.MX6ULL の LCD_ENABLE ピンに接続
11	PWM5_OUT	In/Out	拡張入出力、i.MX6ULL の NAND_DQS ピンに接続
12	GND	Power	電源(GND)
13	LCD_DATA00	In/Out	拡張入出力、i.MX6ULL の LCD_DATA00 ピンに接続
14	LCD_DATA01	In/Out	拡張入出力、i.MX6ULL の LCD_DATA01 ピンに接続
15	LCD_DATA02	In/Out	拡張入出力、i.MX6ULL の LCD_DATA02 ピンに接続
16	LCD_DATA03	In/Out	拡張入出力、i.MX6ULL の LCD_DATA03 ピンに接続
17	LCD_DATA04	In/Out	拡張入出力、i.MX6ULL の LCD_DATA04 ピンに接続
18	LCD_DATA05	In/Out	拡張入出力、i.MX6ULL の LCD_DATA05 ピンに接続
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
21	LCD_DATA07	In/Out	拡張入出力、i.MX6ULL の LCD_DATA07 ピンに接続
22	LCD_DATA08	In/Out	拡張入出力、i.MX6ULL の LCD_DATA08 ピンに接続
23	LCD_DATA09	In/Out	拡張入出力、i.MX6ULL の LCD_DATA09 ピンに接続
24	LCD_DATA10	In/Out	拡張入出力、i.MX6ULL の LCD_DATA10 ピンに接続
25	LCD_DATA11	In/Out	拡張入出力、i.MX6ULL の LCD_DATA11 ピンに接続
26	GND	Power	電源(GND)
27	LCD_DATA12	In/Out	拡張入出力、i.MX6ULL の LCD_DATA12 ピンに接続
28	LCD_DATA13	In/Out	拡張入出力、i.MX6ULL の LCD_DATA13 ピンに接続

ピン番号	ピン名	I/O	説明
29	LCD_DATA14	In/Out	拡張入出力、i.MX6ULL の LCD_DATA14 ピンに接続
30	LCD_DATA15	In/Out	拡張入出力、i.MX6ULL の LCD_DATA15 ピンに接続
31	LCD_DATA16	In/Out	拡張入出力、i.MX6ULL の LCD_DATA16 ピンに接続
32	LCD_DATA17	In/Out	拡張入出力、i.MX6ULL の LCD_DATA17 ピンに接続
33	GND	Power	電源(GND)
34	GPIO4_IO18	In/Out	拡張入出力、i.MX6ULL の CSI_PIXCLK ピンに接続
35	GPIO4_IO21	In/Out	拡張入出力、i.MX6ULL の CSI_DATA00 ピンに接続
36	GPIO4_IO24	In/Out	拡張入出力、i.MX6ULL の CSI_DATA03 ピンに接続
37	ADC_IN4	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。
38	ADC_IN3	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。
39	ADC_IN2	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。
40	ADC_IN1	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。

6.29.2.24. CON21(拡張インターフェース)

CON21 は主にオーディオ用の機能を割り当て可能な信号線を接続した、オーディオ拡張用インターフェースです。



CON7(Grove インターフェース)、CON11(LCD インターフェース)と共通の信号線が接続されているため、同時に使用できません。また、9、10 ピンの信号線は基板上のリアルタイムクロックにも接続されており、I2C で使用しておりますので、マルチプレクスの設定を変更する際には、ご注意ください。

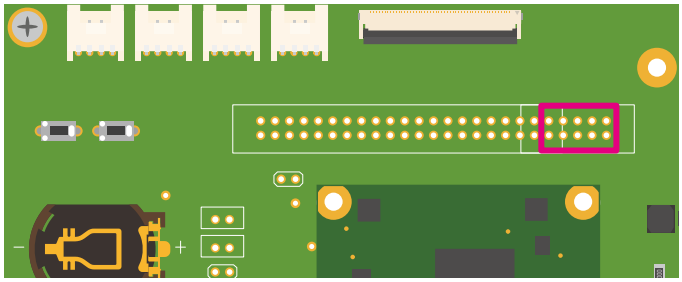


図 6.226 Armadillo-610 拡張ボード CON21

表 6.62 CON21 信号配列

ピン番号	ピン名	I/O	説明
1	+3.3V_IO	Power	電源出力(+3.3V_IO)
2	GND	Power	電源(GND)
3	SAI1_MCLK	In/Out	拡張入出力、i.MX6ULL の CSI_DATA01 ピンに接続
4	SAI1_RX_SYNC	In/Out	拡張入出力、i.MX6ULL の CSI_DATA02 ピンに接続
5	SAI1_TX_DATA	In/Out	拡張入出力、i.MX6ULL の CSI_DATA07 ピンに接続
6	SAI1_RX_DATA	In/Out	拡張入出力、i.MX6ULL の CSI_DATA06 ピンに接続
7	SAI1_TX_BCLK	In/Out	拡張入出力、i.MX6ULL の CSI_DATA05 ピンに接続
8	SAI1_TX_SYNC	In/Out	拡張入出力、i.MX6ULL の CSI_DATA04 ピンに接続
9	I2C2_SCL	In/Out	拡張入出力、i.MX6ULL の CSI_HSYNC ピンに接続

ピン番号	ピン名	I/O	説明
10	I2C2_SDA	In/Out	拡張入出力、i.MX6ULL の CSI_VSYNC ピンに接続

6.29.2.25. CON22(拡張インターフェース)

CON22 は SPI、I2C、UART 等に割り当て可能な信号線を接続した、拡張用インターフェースです。



CON13B(DIDO インターフェース)、CON13C(RS-485 インターフェース)と共通の信号線が接続されているため、同時に使用できません。R93～R96、R105、R108、R110、R112(CON13 周辺)の抵抗を未実装にすることで、CON13 と切り離すことが可能です。

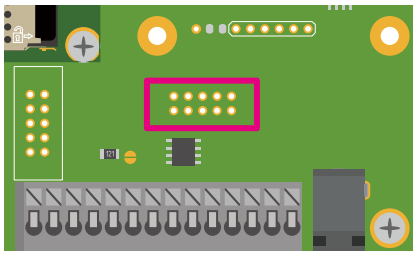


図 6.227 Armadillo-610 拡張ボード CON22

表 6.63 CON22 信号配列

ピン番号	ピン名	I/O	説明
1	+3.3V_IO	Power	電源出力(+3.3V_IO)
2	GND	Power	電源(GND)
3	RS485_TX	In/Out	拡張入出力、i.MX6ULL の NAND_DATA04 ピンに接続
4	RS485_RX	In/Out	拡張入出力、i.MX6ULL の NAND_DATA05 ピンに接続
5	RS485_RE_N	In/Out	拡張入出力、i.MX6ULL の NAND_DATA06 ピンに接続
6	RS485_DE	In/Out	拡張入出力、i.MX6ULL の NAND_DATA07 ピンに接続
7	DI1	In/Out	拡張入出力、i.MX6ULL の UART2_TX_DATA ピンに接続
8	DI2	In/Out	拡張入出力、i.MX6ULL の UART2_RX_DATA ピンに接続
9	DO1	In/Out	拡張入出力、i.MX6ULL の UART5_TX_DATA ピンに接続
10	DO2	In/Out	拡張入出力、i.MX6ULL の UART5_RX_DATA ピンに接続

6.29.2.26. CON23(リセットインターフェース)

CON23 はリセット入出力用のインターフェースです。EXT_RESET_N ピンは Armadillo-610 上のパワーマネジメント IC の RESETBMCU ピン、i.MX6ULL の POR_B ピンに接続されています。

リセット回路の詳細につきましては、「図 3.69. リセット回路の構成」をご参照ください。

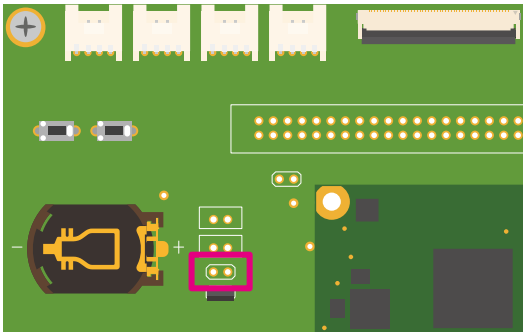


図 6.228 Armadillo-610 拡張ボード CON23

表 6.64 CON23 信号配列

ピン番号	ピン名	I/O	説明
1	EXT_RESET_N	In/Out	リセット信号、パワーマネジメント IC の RESETBMCU ピン、i.MX6ULL の POR_B ピンに接続
2	GND	Power	電源(GND)

6.29.2.27. CON24(電源入力インターフェース)

CON24 は電源供給用のインターフェースです。電源回路の詳細につきましては、「図 6.201. Armadillo-610 拡張ボードの電源回路の構成」をご参照ください。



出荷時、CON24(電源入力インターフェース)から電源供給することはできません。R155 を実装、R156 未実装にすることで、CON24 の 1 ピンから Armadillo-610 の電源(VIN)に電源を供給することが可能になります。回路図をご確認の上、絶対定格値を超えない範囲でご使用ください。Armadillo-610 の入力電圧範囲(VIN)は 3.6～4.5V です。

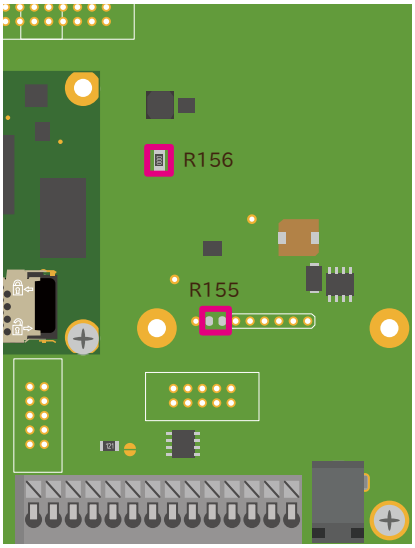


図 6.229 Armadillo-610 拡張ボード CON24

表 6.65 CON24 信号配列

ピン番号	ピン名	I/O	説明
1	BAT_IN	Power	電源(BAT_IN)、R155(出荷時未実装)を経由してパワーマネジメント IC の VIN ピンに接続
2	GND	Power	電源(GND)
3	+5V	Power	電源(+5V)
4	GND	Power	電源(GND)
5	NC	-	未接続
6	NC	-	未接続

6.29.2.28. JP1 (起動デバイス設定ジャンパ)

JP1 は起動デバイス設定ジャンパです。JP1 の状態で、起動デバイスを設定することができます。

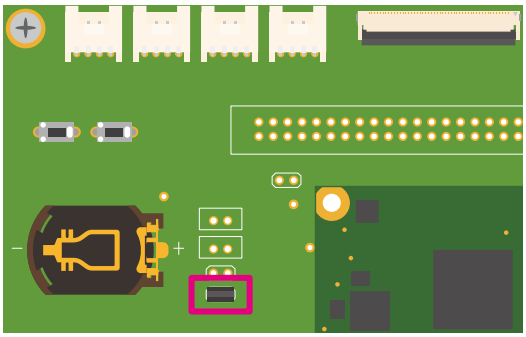


図 6.230 Armadillo-610 拡張ボード JP1

表 6.66 JP1 信号配列

部品番号	説明
JP1	ロジック IC を経由して、i.MX6ULL の LCD_DATA05 ピン、LCD_DATA_DATA11 ピンに接続

表 6.67 ジャンパの設定と起動デバイス

JP1	起動デバイス
オープン	eMMC
ショート	microSD



eFUSE で起動デバイスを設定している場合、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。



eFUSE は一度書き込むと元に戻すことができません。eFUSE の設定によっては Armadillo-610 が正常に動作しなくなる可能性がありますので、細心の注意を払って書き込みを行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。

6.29.2.29. SW1(ユーザースイッチ)

SW1 はユーザー側で自由に利用できる押しボタンスイッチです。

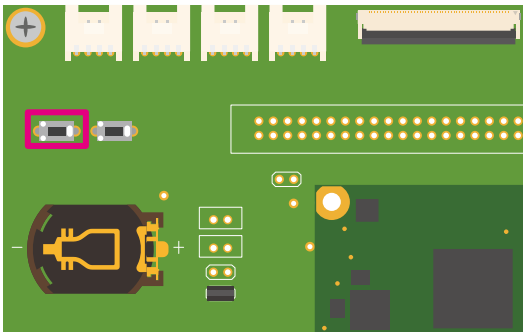


図 6.231 Armadillo-610 拡張ボード SW1

表 6.68 SW1 信号配列

部品番号	説明
SW1	i.MX6ULL の JTAG_MOD ピンに接続(押されていない状態: Low、押された状態: High)

6.29.2.30. SW2(リセットスイッチ)

SW2 はリセット用の押しボタンスイッチです。ボタンを押すとパワーマネジメント IC からの電源が切断されます。

動作の詳細については、「3.4.6.6. 外部からの電源制御」をご確認ください。

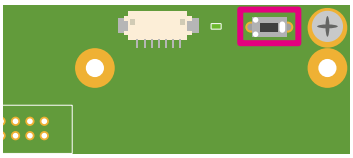


図 6.232 Armadillo-610 拡張ボード SW2

表 6.69 SW2 信号配列

部品番号	説明
SW2	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(押されていない状態: リセット解除、 押された状態: リセット状態)

6.29.2.31. SW3(ONOFF スイッチ)

SW3 はボタンの長押しで電源を制御する押しボタンスイッチです。

動作の詳細については、「3.4.6.6. 外部からの電源制御」をご確認ください。

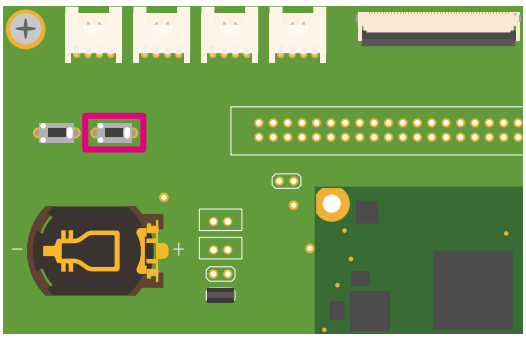


図 6.233 Armadillo-610 拡張ボード SW3

表 6.70 SW3 信号配列

部品番号	説明
SW3	i.MX6ULL の ONOFF ピンに接続

6.29.2.32. LED1、LED2(LAN LED)

LED1、LED2 は CON2(LAN インターフェース)のステータス LED です。CON2(LAN インターフェース)の上部に表示されます。信号線は Ethernet PHY(LAN8720AI-CP/Microchip Technology)の LED ピンに接続されます。

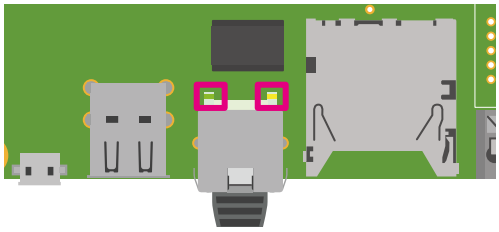


図 6.234 Armadillo-610 拡張ボード LED1、LED2

表 6.71 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

6.29.2.33. LED3(ユーザー LED)

LED3 ユーザー側で自由に利用できる LED です。

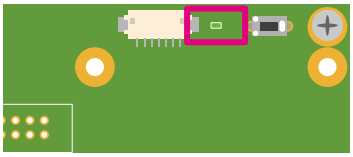


図 6.235 Armadillo-610 拡張ボード LED3

表 6.72 LED3

部品番号	名称(色)	説明
LED3	ユーザー LED(緑)	i.MX6ULL の GPIO1_IO08 ピンに接続、(Low: 消灯、High: 点灯)

6.29.2.34. 基板形状図

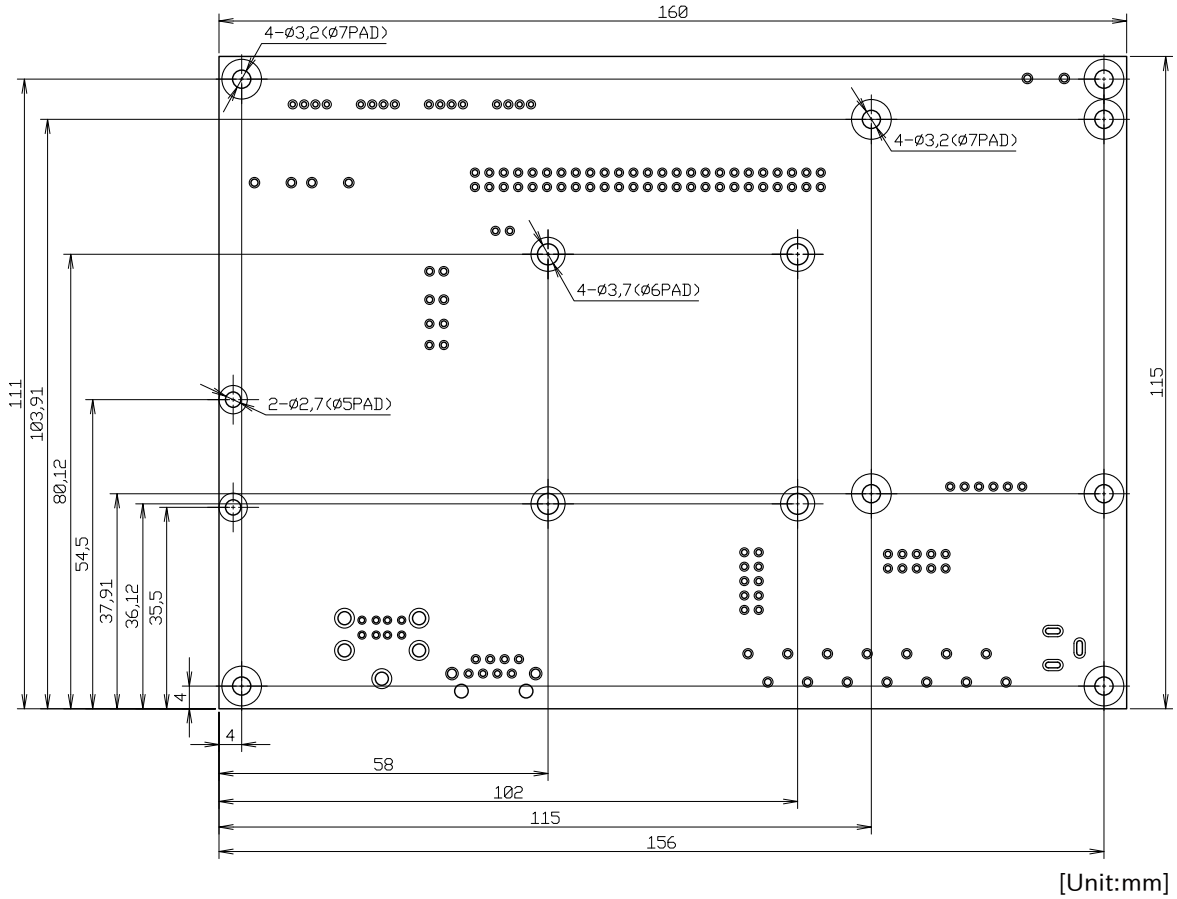


図 6.236 Armadillo-610 拡張ボード 基板形状および固定穴寸法

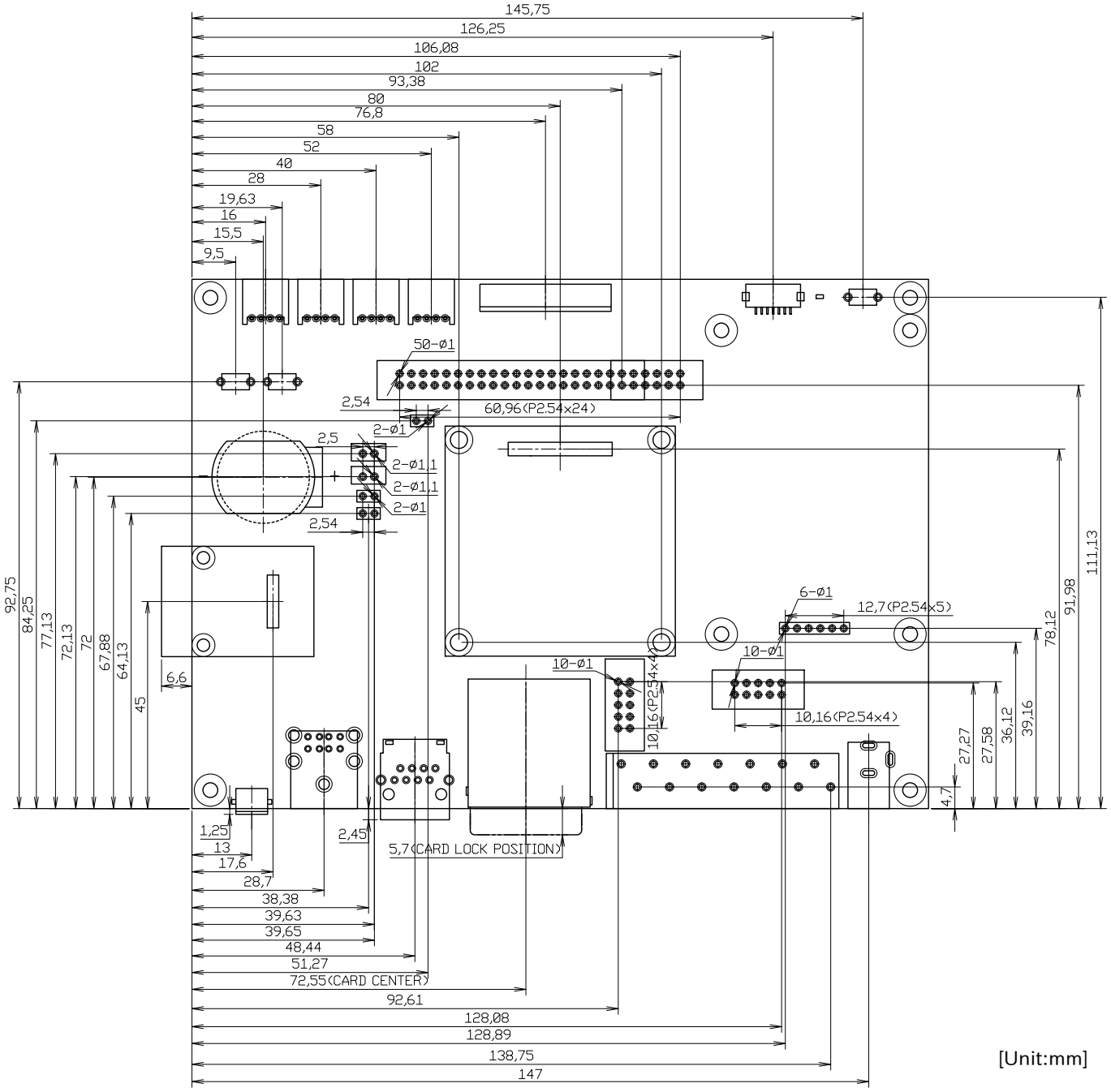


図 6.237 Armadillo-610 拡張ボード コネクタ中心寸法および穴寸法

6.29.3. LCD オプションセット(7 インチタッチパネル WVGA 液晶)

6.29.3.1. 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。LCD インターフェース(Armadillo-610 拡張ボード: CON11)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、「3.6.12. LCD を使用する」を参照してください。



LCD オプションセット(7 インチタッチパネル WVGA 液晶)を使用する場合、Armadillo-610 拡張ボード搭載のリアルタイムクロックが使用できません。

表 6.73 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 6.74 LCD の仕様

型番	GTWV070VRB00P
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℃
輝度	850cd/m^2 (Typ.) 25℃
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]LCD インターフェース(Armadillo-610 拡張ボード: CON11)は 18bit 対応です。



タッチパネル LCD をご使用になる前に、『GTWV070VRB00P 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。

『GTWV070VRB00P 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。

6.29.3.2. 組み立て

「3.6.12. LCD を使用する」を参照してください。

6.29.4. Armadillo-400 シリーズ LCD オプションセット

6.29.4.1. 概要

4.3 インチタッチパネル LCD、LCD 拡張ボード、フレキシブルフラットケーブル(FFC)のセットです。

LCD 拡張ボードには LCD 接続インターフェースの他にオーディオコーデック、リアルタイムクロック(RTC)を搭載しています。LCD インターフェース(Armadillo-610 拡張ボード: CON11)に接続して使用することが可能です。



LCD 拡張ボードの回路図、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。



本製品は Armadillo-400 シリーズ用に準備された LCD オプションセットですが、Armadillo-600 シリーズでも利用可能です。

表 6.75 Armadillo-400 シリーズ LCD オプションセットについて

商品名	Armadillo-400 シリーズ LCD オプションセット
型番	OP-A400-LCD43EXT-L01
内容	4.3 インチ タッチパネル LCD、FFC、LCD 固定用両面テープ

表 6.76 LCD 拡張ボードの仕様

LCD	型番	FG040346DSSWBG04 [a]
	メーカー	Data Image
LCD バックライト	バックライト用 LED ドライバ搭載	
オーディオ	型番	WM8978GEFL/V
	メーカー	Wolfson
リアルタイムクロック	型番	S-35390A
	メーカー	ABLIC
RTC バックアップ	300 秒(Typ.)、60 秒(Min.)、電池ホルダ搭載、外部バッテリ接続コネクタ搭載可能(対応電池: CR2032 等)	
RTC 平均月差(参考値)	約 30 秒@25 [mailto:約 30 秒@25]℃	
電源電圧	DC 3.3±0.2V(メイン電源)、DC 2.8～5.5V(LCD バックライト)、DC 2.0～3.5V(RTC バックアップ)	
消費電力	約 0.8W(LCD 消費分を含む)	
使用温度範囲	-20～+70℃(結露なきこと)	
外形サイズ	106 x 82 mm(突起部を除く)	

[a]LCD 拡張インターフェース(汎用)により、その他の LCD も接続可能です。



RTC の時間精度やバックアップ時間は周囲温度、電圧印加時間等に大きく影響を受けます。ご使用の際には十分に特性の確認をお願いします。

表 6.77 LCD の仕様

型番	FG040346DSSWBG04
メーカー	Data Image
タイプ	TFT-LCD
表示サイズ	4.3 インチ
外形サイズ	105.5(W) x 67.2(H) x 4.3(D) mm
解像度	480 x 272 pixels

表示色数	約 1677 万色
使用温度範囲	-20～+70℃
輝度	320cd/m^2 (Typ.)
映像入力インターフェース	RGB パラレル(24bit)[a]
タッチ方式	4 線抵抗膜方式

[a]Armadillo-610 拡張ボードの CON11 は 18bit 対応です。



LCD オプションセットは製品リビジョンによって一部仕様が違いますのでご注意ください。本章には、「製品リビジョン F」以降についての情報を記載しています。製品リビジョンにつきましては、「アットマークテクノ Armadillo サイト」[https://armadillo.atmark-techno.com/]からダウンロードできる「Armadillo-400 シリーズ LCD オプションセット変更履歴表」にてご確認ください。

6.29.4.2. ブロック図

LCD 拡張ボードのブロック図は次のとおりです。

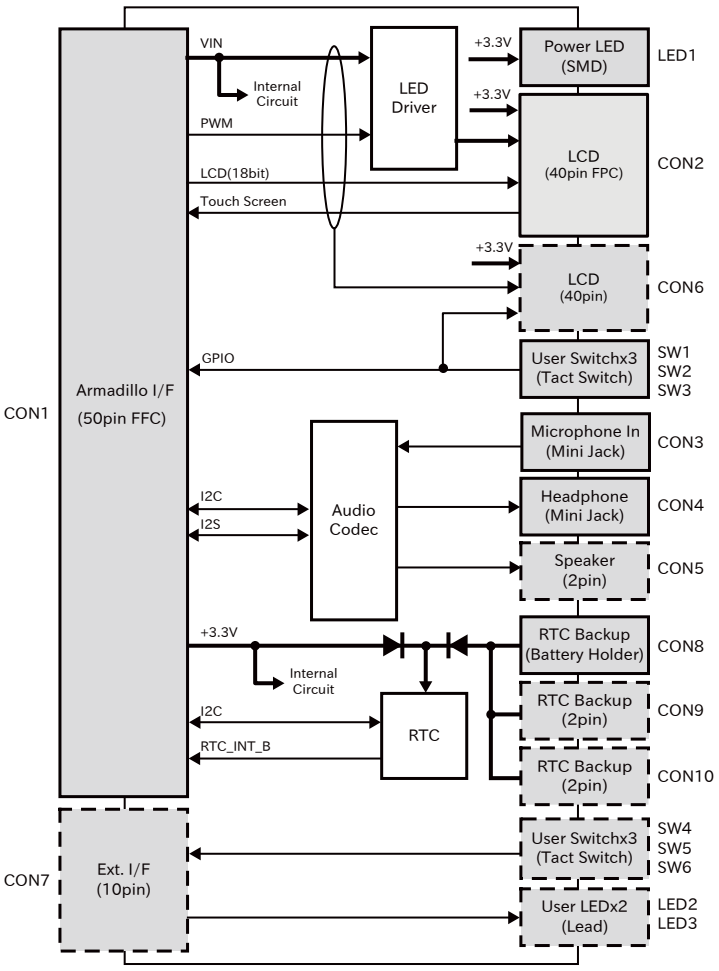


図 6.238 LCD 拡張ボードのブロック図

6.29.4.3. インターフェースの概要

LCD 拡張ボードのインターフェース仕様について説明します。

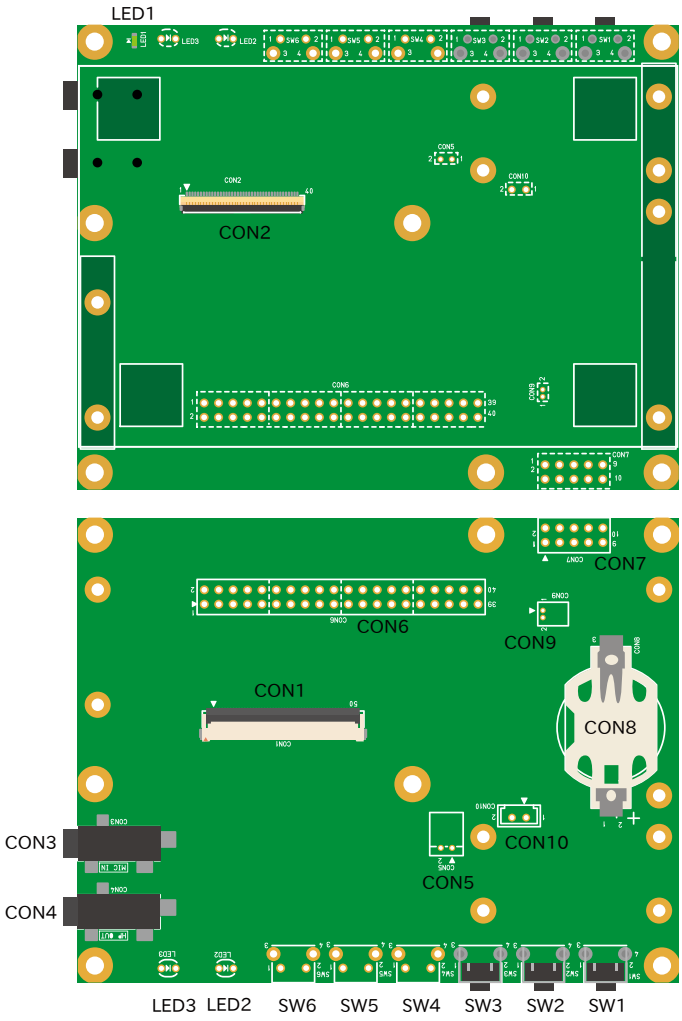


図 6.239 LCD 拡張ボードのインターフェース

表 6.78 LCD 拡張ボード 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 接続インターフェース	XF2M-5015-1A	OMRON
CON2	LCD 拡張インターフェース(専用)	FH19SC-40S-0.5SH(05)	HIROSE ELECTRIC
CON3	モノラルマイク入力	SJ-3524-SMT	Same Sky
CON4	ステレオヘッドホン出力	SJ-3524-SMT	Same Sky
CON5	スピーカー出力	S2B-PH-K-S(LF)(SN)	J.S.T.Mfg.
CON6	LCD 拡張インターフェース(汎用)	RF-H402TD-1130	J.S.T.Mfg.
CON7	拡張インターフェース	RF-H102TD-1130	J.S.T.Mfg.
CON8	RTC バックアップインターフェース	HU2032	TAKACHI
CON9		DF13-2P-1.25DS(20)	HIROSE ELECTRIC
CON10		B2B-EH(LF)(SN)	J.S.T.Mfg.

部品番号	インターフェース名	型番	メーカー
SW1	ユーザースイッチ	SKHHLMA010	ALPS
SW2		SKHHLMA010	ALPS
SW3		SKHHLMA010	ALPS
SW4		SKHHLMA010	ALPS
SW5		SKHHLMA010	ALPS
SW6		SKHHLMA010	ALPS
LED1	電源 LED	SML-310MT	ROHM
LED2	ユーザー LED	OSNG3133A	OPTO SUPPLY
LED3		OSNG3133A	OPTO SUPPLY

[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。

6.29.4.4. CON1 (Armadillo 接続インターフェース)

CON1 は Armadillo と接続するためのインターフェースです。50 ピン(0.5mm ピッチ)のフレキシブルフラットケーブルにより、LCD インターフェース(Armadillo-610 拡張ボード: CON11)と接続可能です。

LCD、タッチパネル、オーディオコーデック、リアルタイムクロックの各信号線が接続されています。

表 6.79 CON1 信号配列

ピン番号	ピン名	I/O	説明
1	GND	Power	電源(GND)
2	I2C3_SDA	In/Out	コーデック I2C データ、コーデックの SDIN ピンに接続、1kΩ プルアップ(+3.3V)
3	I2C3_SCL	In	コーデック I2C クロック、コーデック IC の SCLK ピンに接続、1kΩ プルアップ(+3.3V)
4	AUD5_TXFS	In	コーデック TXFS、コーデックの LRC ピンに接続
5	AUD5_TXC	In	コーデック TXC、コーデックの BCLK ピンに接続
6	AUD5_RXD	Out	コーデック RXD、コーデックの ADCDAT ピンに接続
7	AUD5_TXD	In	コーデック TXD、コーデックの DACDAT ピンに接続
8	AUD_RXFS	In/Out	R50 の実装によりリアルタイムクロックの割り込みピンに接続
9	AUD5_SYSCLK	In	コーデック SYSCLK、コーデックの MCLK ピンに接続
10	GPIO2_30	In/Out	タクトスイッチ出力、SW3 の 2 ピン、CON6 の 7 ピンに接続
11	GPIO2_29	In/Out	タクトスイッチ出力、SW2 の 2 ピン、CON6 の 8 ピンに接続
12	GPIO2_20	In/Out	タクトスイッチ出力、SW1 の 2 ピン、CON6 の 9 ピンに接続
13	GND	Power	電源(GND)
14	TOUCH_YN	In/Out	タッチパネル YN、CON2 の 40 ピン、CON6 の 10 ピンに接続
15	TOUCH_YP	In/Out	タッチパネル YP、CON2 の 38 ピン、CON6 の 11 ピンに接続
16	TOUCH_XN	In/Out	タッチパネル XN、CON2 の 39 ピン、CON6 の 12 ピンに接続
17	TOUCH_XP	In/Out	タッチパネル XP、CON2 の 37 ピン、CON6 の 13 ピンに接続
18	GND	Power	電源(GND)
19	LCD_LD17	In	LCD 拡張 I/F LD17、CON2 の 12 ピン、CON6 の 15 ピンに接続
20	LCD_LD16	In	LCD 拡張 I/F LD16、CON2 の 11 ピン、CON6 の 16 ピンに接続
21	LCD_LD15	In	LCD 拡張 I/F LD15、CON2 の 10 ピン、CON6 の 17 ピンに接続
22	LCD_LD14	In	LCD 拡張 I/F LD14、CON2 の 9 ピン、CON6 の 18 ピンに接続
23	LCD_LD13	In	LCD 拡張 I/F LD13、CON2 の 8 ピン、CON6 の 19 ピンに接続
24	LCD_LD12	In	LCD 拡張 I/F LD12、CON2 の 7 ピン、CON6 の 20 ピンに接続
25	GND	Power	電源(GND)
26	LCD_LD11	In	LCD 拡張 I/F LD11、CON2 の 20 ピン、CON6 の 22 ピンに接続
27	LCD_LD10	In	LCD 拡張 I/F LD10、CON2 の 19 ピン、CON6 の 23 ピンに接続
28	LCD_LD9	In	LCD 拡張 I/F LD9、CON2 の 18 ピン、CON6 の 24 ピンに接続
29	LCD_LD8	In	LCD 拡張 I/F LD8、CON2 の 17 ピン、CON6 の 25 ピンに接続
30	LCD_LD7	In	LCD 拡張 I/F LD7、CON2 の 16 ピン、CON6 の 26 ピンに接続
31	LCD_LD6	In	LCD 拡張 I/F LD6、CON2 の 15 ピン、CON6 の 27 ピンに接続

ピン番号	ピン名	I/O	説明
32	GND	Power	電源(GND)
33	LCD_LD5	In	LCD 拡張 I/F LD4、CON2 の 28 ピン、CON6 の 29 ピンに接続
34	LCD_LD4	In	LCD 拡張 I/F LD4、CON2 の 27 ピン、CON6 の 30 ピンに接続
35	LCD_LD3	In	LCD 拡張 I/F LD3、CON2 の 26 ピン、CON6 の 31 ピンに接続
36	LCD_LD2	In	LCD 拡張 I/F LD2、CON2 の 25 ピン、CON6 の 32 ピンに接続
37	LCD_LD1	In	LCD 拡張 I/F LD1、CON2 の 24 ピン、CON6 の 33 ピンに接続
38	LCD_LD0	In	LCD 拡張 I/F LD0、CON2 の 23 ピン、CON6 の 34 ピンに接続
39	PWMO1	In	LCD 拡張 I/F PWMO1、CON6 の 36 ピン、LED ドライバの FB ピンに接続
40	LCD_OE_ACD	In	LCD 拡張 I/F OE_ACD、CON2 の 34 ピン、CON6 の 37 ピンに接続
41	LCD_VSYN	In	LCD 拡張 I/F VSYN、CON2 の 33 ピン、CON6 の 38 ピンに接続
42	LCD_HSYN	In	LCD 拡張 I/F HSYN、CON2 の 32 ピン、CON6 の 39 ピンに接続
43	LCD_LSCLK	In	LCD 拡張 I/F LSCLK、CON2 の 30 ピン、CON6 の 40 ピンに接続
44	GND	Power	電源(GND)
45	GND	Power	電源(GND)
46	+3.3V	Power	電源(+3.3V)
47	+3.3V	Power	電源(+3.3V)
48	VIN	Power	電源(VIN)
49	VIN	Power	電源(VIN)
50	VIN	Powe	電源(VIN)

6.29.4.5. CON2(LCD 拡張インターフェース)

CON2 は LCD(FG040346DSSWBG04/Data Image)と接続するためのインターフェースです。



CON2 と CON6 には、共通の信号が接続されていますので同時に使用できません。CON6 を使用する場合は、CON2 から LCD を取り外してください。

表 6.80 CON2 信号配列

ピン番号	ピン名	I/O	説明
1	BL_LED_K	Power	LED ドライバ電源(一端子)
2	BL_LED_A	Power	LED ドライバ電源(＋端子)
3	GND	Power	電源(GND)
4	+3.3V	Power	電源(+3.3V)
5	GND	Power	電源(GND)
6	GND	Power	電源(GND)
7	LCD_LD12	Out	LCD 拡張 I/F LD12、CON1 の 24 ピン、CON6 の 20 ピンに接続
8	LCD_LD13	Out	LCD 拡張 I/F LD13、CON1 の 23 ピン、CON6 の 19 ピンに接続
9	LCD_LD14	Out	LCD 拡張 I/F LD14、CON1 の 22 ピン、CON6 の 18 ピンに接続
10	LCD_LD15	Out	LCD 拡張 I/F LD15、CON1 の 21 ピン、CON6 の 17 ピンに接続
11	LCD_LD16	Out	LCD 拡張 I/F LD16、CON1 の 20 ピン、CON6 の 16 ピンに接続
12	LCD_LD17	Out	LCD 拡張 I/F LD17、CON1 の 19 ピン、CON6 の 15 ピンに接続
13	GND	Power	電源(GND)
14	GND	Power	電源(GND)
15	LCD_LD6	Out	LCD 拡張 I/F LD6、CON1 の 31 ピン、CON6 の 27 ピンに接続
16	LCD_LD7	Out	LCD 拡張 I/F LD7、CON1 の 30 ピン、CON6 の 26 ピンに接続
17	LCD_LD8	Out	LCD 拡張 I/F LD8、CON1 の 29 ピン、CON6 の 25 ピンに接続
18	LCD_LD9	Out	LCD 拡張 I/F LD9、CON1 の 28 ピン、CON6 の 24 ピンに接続
19	LCD_LD10	Out	LCD 拡張 I/F LD10、CON1 の 27 ピン、CON6 の 23 ピンに接続
20	LCD_LD11	Out	LCD 拡張 I/F LD11、CON1 の 26 ピン、CON6 の 22 ピンに接続

ピン番号	ピン名	I/O	説明
21	GND	Power	電源(GND)
22	GND	Power	電源(GND)
23	LCD_LD0	Out	LCD 拡張 I/F LD0、CON1 の 38 ピン、CON6 の 34 ピンに接続
24	LCD_LD1	Out	LCD 拡張 I/F LD1、CON1 の 37 ピン、CON6 の 33 ピンに接続
25	LCD_LD2	Out	LCD 拡張 I/F LD2、CON1 の 36 ピン、CON6 の 32 ピンに接続
26	LCD_LD3	Out	LCD 拡張 I/F LD3、CON1 の 35 ピン、CON6 の 31 ピンに接続
27	LCD_LD4	Out	LCD 拡張 I/F LD4、CON1 の 34 ピン、CON6 の 30 ピンに接続
28	LCD_LD5	Out	LCD 拡張 I/F LD5、CON1 の 33 ピン、CON6 の 29 ピンに接続
29	GND	Power	電源(GND)
30	DISP	In/Out	10kΩ プルアップ(+3.3V)
31	LCD_LSCLK	Out	LCD 拡張 I/F LSCLK、CON1 の 43 ピン、CON6 の 40 ピンに接続
32	LCD_HSYN	Out	LCD 拡張 I/F HSYN、CON1 の 42 ピン、CON6 の 39 ピンに接続
33	LCD_VSYN	Out	LCD 拡張 I/F VSYN、CON1 の 41 ピン、CON6 の 38 ピンに接続
34	LCD_OE_ACD	Out	LCD 拡張 I/F OE_ACD、CON1 の 40 ピン、CON6 の 37 ピンに接続
35	NC	ー	ー
36	GND	Power	電源(GND)
37	TOUCH_XP	In/Out	タッチパネル XP、 CON1 の 17 ピン、CON6 の 13 ピンに接続
38	TOUCH_YP	In/Out	タッチパネル YP、 CON1 の 15 ピン、CON6 の 11 ピンに接続
39	TOUCH_XN	In/Out	タッチパネル XN、 CON1 の 16 ピン、CON6 の 12 ピンに接続
40	TOUCH_YN	In/Out	タッチパネル YN、 CON1 の 14 ピン、CON6 の 10 ピンに接続

6.29.4.6. CON3(モノラルマイク入力)

CON3 はモノラルマイク入力です。

表 6.81 CON3 信号配列

ピン番号	ピン名	I/O	説明
1	GND	Power	電源(GND)
2	MIC_IN	In	コーデックの LIP ピンに接続
3	ー	ー	ー
10	ー	ー	ー

6.29.4.7. CON4(ステレオヘッドホン出力)

CON4 はステレオヘッドホン出力です。

表 6.82 CON4 信号配列

ピン番号	ピン名	I/O	説明
1	GND	Power	電源(GND)
2	HP_L_OUT	Out	コーデックの LOUT1 ピンに接続
3	HP_R_OUT	Out	コーデックの ROUT1 ピンに接続
10	HP_DET	In	コーデックの L2/GPIO2 ピンに接続

6.29.4.8. CON5(スピーカー出力)

CON5 はスピーカー出力です。部品は実装されていません。

表 6.83 CON5 信号配列

ピン番号	ピン名	I/O	説明
1	SPK_N	Out	コーデックの LOUT2 ピンに接続
2	SPK_P	Out	コーデックの ROUT2 ピンに接続

6.29.4.9. CON6(LCD 拡張インターフェース)

CON6 は LCD と接続するためのインターフェースです。部品は実装されていません。



CON2 と CON6 には、共通の信号が接続されていますので同時に使用できません。CON6 を使用する場合は、CON2 から LCD を取り外してください。

表 6.84 CON6 信号配列

ピン番号	ピン名	I/O	説明
1	VIN	Power	電源(VIN)
2	VIN	Power	電源(VIN)
3	+3.3V	Power	電源(+3.3V)
4	+3.3V	Power	電源(+3.3V)
5	GND	Power	電源(GND)
6	GND	Power	電源(GND)
7	GPIO2_30	In/Out	タクトスイッチ出力、SW3 の 2 ピン、CON1 の 10 ピンに接続
8	GPIO2_29	In/Out	タクトスイッチ出力、SW2 の 2 ピン、CON1 の 11 ピンに接続
9	GPIO2_20	In/Out	タクトスイッチ出力、SW1 の 2 ピン、CON1 の 12 ピンに接続
10	TOUCH_YN	In/Out	タッチパネル YN、 CON1 の 14 ピン、CON1 の 40 ピンに接続
11	TOUCH_YP	In/Out	タッチパネル YP、 CON1 の 15 ピン、CON1 の 38 ピンに接続
12	TOUCH_XN	In/Out	タッチパネル XN、 CON1 の 16 ピン、CON1 の 39 ピンに接続
13	TOUCH_XP	In/Out	タッチパネル XP、 CON1 の 17 ピン、CON1 の 37 ピンに接続
14	GND	Power	電源(GND)
15	LCD_LD17	Out	LCD 拡張 I/F LD17、CON2 の 12 ピン、CON1 の 19 ピンに接続
16	LCD_LD16	Out	LCD 拡張 I/F LD16、CON2 の 11 ピン、CON1 の 20 ピンに接続
17	LCD_LD15	Out	LCD 拡張 I/F LD15、CON2 の 10 ピン、CON1 の 21 ピンに接続
18	LCD_LD14	Out	LCD 拡張 I/F LD14、CON2 の 9 ピン、CON1 の 22 ピンに接続
19	LCD_LD13	Out	LCD 拡張 I/F LD13、CON2 の 8 ピン、CON1 の 23 ピンに接続
20	LCD_LD12	Out	LCD 拡張 I/F LD12、CON2 の 7 ピン、CON1 の 24 ピンに接続
21	GND	Power	電源(GND)
22	LCD_LD11	Out	LCD 拡張 I/F LD11、CON2 の 20 ピン、CON1 の 26 ピンに接続
23	LCD_LD10	Out	LCD 拡張 I/F LD10、CON2 の 19 ピン、CON1 の 27 ピンに接続
24	LCD_LD9	Out	LCD 拡張 I/F LD9、CON2 の 18 ピン、CON1 の 28 ピンに接続
25	LCD_LD8	Out	LCD 拡張 I/F LD8、CON2 の 17 ピン、CON1 の 29 ピンに接続
26	LCD_LD7	Out	LCD 拡張 I/F LD7、CON2 の 16 ピン、CON1 の 30 ピンに接続
27	LCD_LD6	Out	LCD 拡張 I/F LD6、CON2 の 15 ピン、CON1 の 31 ピンに接続
28	GND	Power	電源(GND)
29	LCD_LD5	Out	LCD 拡張 I/F LD5、CON2 の 28 ピン、CON1 の 33 ピンに接続
30	LCD_LD4	Out	LCD 拡張 I/F LD4、CON2 の 27 ピン、CON1 の 34 ピンに接続
31	LCD_LD3	Out	LCD 拡張 I/F LD3、CON2 の 26 ピン、CON1 の 35 ピンに接続
32	LCD_LD2	Out	LCD 拡張 I/F LD2、CON2 の 25 ピン、CON1 の 36 ピンに接続
33	LCD_LD1	Out	LCD 拡張 I/F LD1、CON2 の 24 ピン、CON1 の 37 ピンに接続
34	LCD_LD0	Out	LCD 拡張 I/F LD0、CON2 の 23 ピン、CON1 の 38 ピンに接続
35	GND	Power	電源(GND)
36	PWMO1	Out	LCD 拡張 I/F PWMO1、CON1 の 39 ピン、LED ドライバの FB ピンに接続
37	LCD_OE_ACD	Out	LCD 拡張 I/F OE_ACD、CON1 の 40 ピン、CON1 の 34 ピンに接続
38	LCD_VSYN	Out	LCD 拡張 I/F VSYN、CON1 の 41 ピン、CON1 の 33 ピンに接続
39	LCD_HSYN	Out	LCD 拡張 I/F HSYN、CON1 の 42 ピン、CON1 の 32 ピンに接続
40	LCD_LSCLK	Out	LCD 拡張 I/F LSCLK、CON1 の 43 ピン、CON1 の 31 ピンに接続

6.29.4.10. CON7(拡張インターフェース)

CON7 はユーザー側で自由に利用できるスイッチ、LED 拡張用のインターフェースです。部品は実装されていません。

表 6.85 CON7 信号配列

ピン番号	ピン名	I/O	説明
1	LED2	In	LED2 に接続 (Low:消灯、 High:点灯)
2	NC	—	—
3	LED3	In	LED3 に接続 (Low:消灯、 High:点灯)
4	NC	—	—
5	SW4	In/Out	SW4 の 2 ピンに接続
6	NC	—	—
7	SW5	In/Out	SW5 の 2 ピンに接続
8	NC	—	—
9	SW6	In/Out	SW6 の 2 ピンに接続
10	NC	—	—

6.29.4.11. CON8、CON9、CON10(RTC バックアップインターフェース)

CON8、CON9、CON10 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。電源(+3.3V)が切断されても長期間時刻データを保持させたい場合に、別途バックアップ用のバッテリを接続することで、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、お使いのバッテリに合わせてご使用ください。CON9、CON10 に部品は実装されていません。

表 6.86 CON8 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	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
3	GND	Power	電源(GND)



CON8、CON9、CON10 は共通の端子に接続されており、同時に使用することはできません。



CON8、CON9、CON10 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。

表 6.87 CON9、CON10 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)

6.29.4.12. SW1、SW2、SW3、SW4、SW5、SW6(ユーザースイッチ)

SW1、SW2、SW3、SW4、SW5、SW6 はユーザー側で自由に利用できるスイッチです。SW4、SW5、SW6 に部品は実装されていません。

表 6.88 SW1、SW2、SW3、SW4、SW5、SW6 の機能

部品番号	説明
SW1	CON1 の 12 ピン、CON6 の 9 ピンに接続 (Low:押された状態、 Open:押されていない状態)
SW2	CON1 の 11 ピン、CON6 の 8 ピンに接続 (Low:押された状態、 Open:押されていない状態)
SW3	CON1 の 10 ピン、CON6 の 7 ピンに接続 (Low:押された状態、 Open:押されていない状態)
SW4	CON7 の 5 ピンに接続 (Low:押された状態、 Open:押されていない状態)
SW5	CON7 の 7 ピンに接続 (Low:押された状態、 Open:押されていない状態)
SW6	CON7 の 9 ピンに接続 (Low:押された状態、 Open:押されていない状態)

6.29.4.13. LED1(電源 LED)

LED1 は電源 LED です。

表 6.89 LED1 の機能

部品番号	名称(色)	説明
LED1	電源 LED(緑)	電源(+3.3V)ON: 点灯、電源(+3.3V)OFF: 消灯

6.29.4.14. LED2、LED3 ユーザー(LED)

LED2、LED3 はユーザー側で自由に利用できる LED です。部品は実装されていません。

表 6.90 LED2、LED3 の機能

部品番号	名称(色)	説明
LED2	ユーザー LED(ー)	CON7 の 1 ピンと接続(High: 点灯、Low: 消灯)
LED3		CON7 の 3 ピンと接続(High: 点灯、Low: 消灯)

6.29.4.15. 組み立て

タッチパネル LCD と LCD 拡張ボードは「図 6.240. タッチパネル LCD と LCD 拡張ボードの接続」のように接続します。

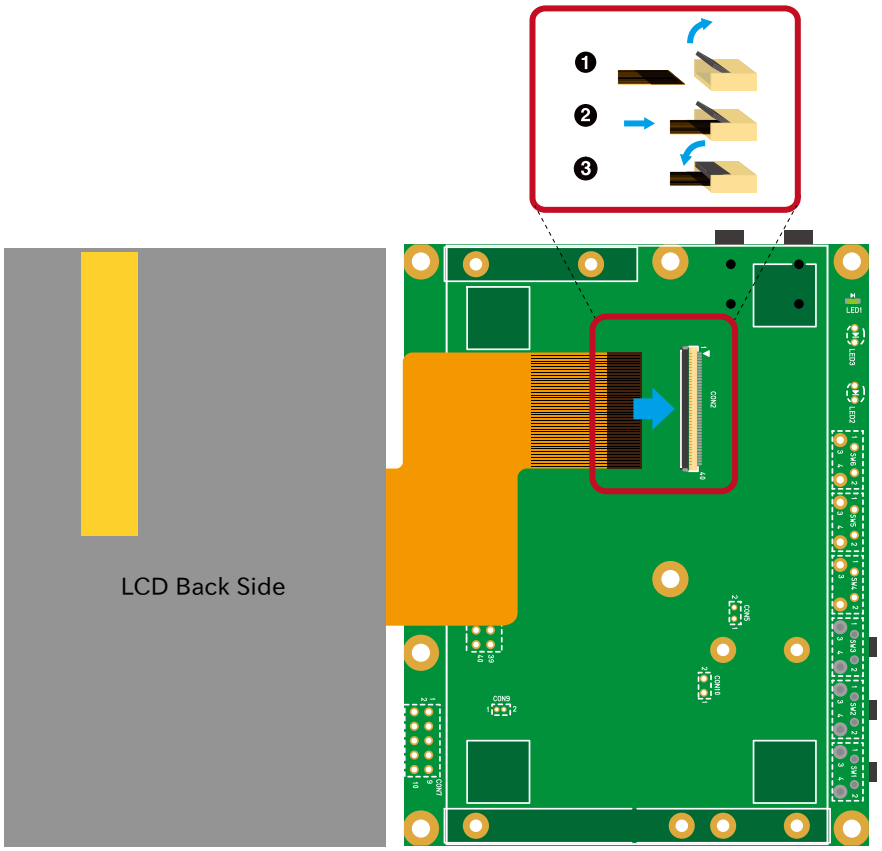


図 6.240 タッチパネル LCD と LCD 拡張ボードの接続

❶ LCD 拡張ボード CON2 のロックレバーを上げる

❷ タッチパネル LCD のフレキシブル基板(FPC)が止まるところまで挿入

❸ LCD 拡張ボード CON2 のロックレバーを下げる



LCD 拡張ボード CON2 のロックレバーは指でつまむ等の操作は避け、親指や人指し指の爪により、跳ね上げる感じで操作してください。強い力を加えると、コネクタが破損する恐れがあります。

タッチパネル LCD を LCD 拡張ボードに両面テープで固定する場合、「図 6.241. 両面テープの貼付位置」の位置に貼付するのがおすすめです。

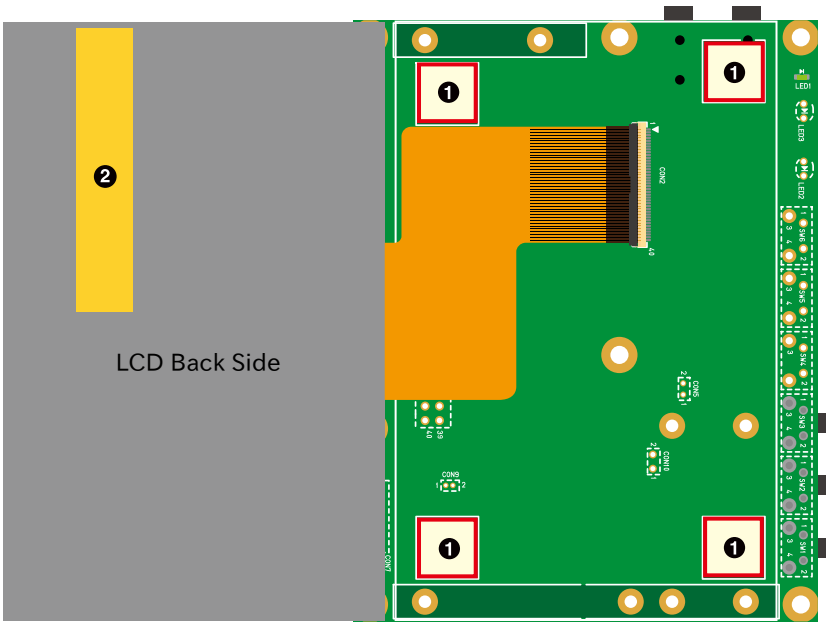


図 6.241 両面テープの貼付位置

❶ 両面テープ貼付位置

❷ 絶縁テープ

付属の両面テープは以下の理由から量産時の使用を推奨しておりません。

- 経年劣化により両面テープの粘着力が低下し、タッチパネル LCD が剥がれる可能性があります。
- 弾力性のある両面テープでタッチパネル LCD を LCD 拡張ボードに固定した場合、タッチパネルに強い力が加わった際に、タッチパネル LCD のフレームと LCD 拡張ボードの基板配線が接触し、故障の原因となる可能性があります。

LCD 拡張ボードには LCD の固定に利用可能な穴を複数設けておりますので、ご利用ください。

タッチパネル LCD 裏面の絶縁テープは LCD 拡張ボード CON2 とのショートを防止するために貼付していますので、剥がさないでください。

Armadillo-610 拡張ボードと LCD 拡張ボードは「図 6.242. Armadillo-610 と LCD 拡張ボードの接続」を参考にし、Armadillo-610 拡張ボードの CON11 の 1 ピンと LCD 拡張ボードの CON1 の 50 ピンが対応するように、FFC を接続します。FFC は電極が上になるようにします。

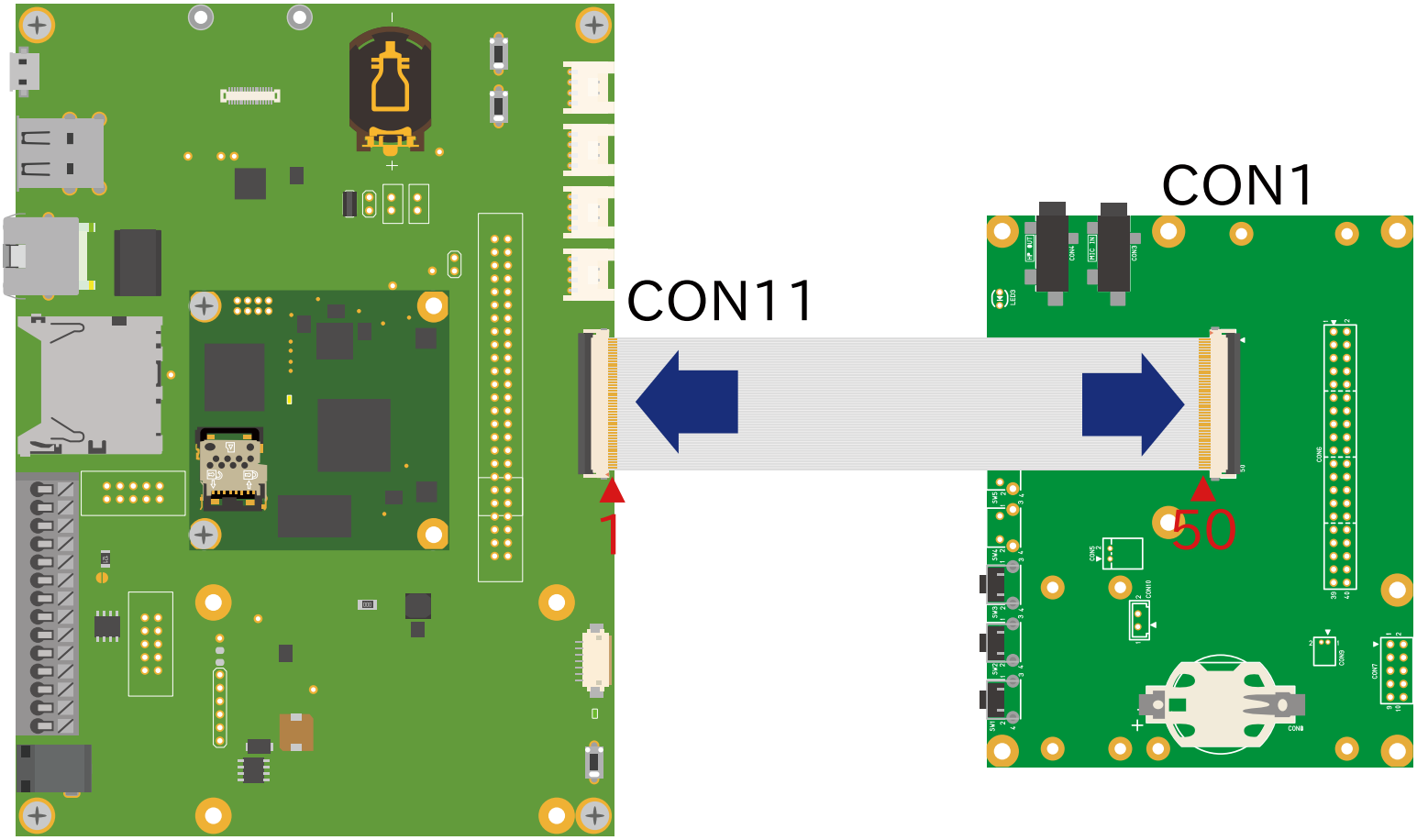


図 6.242 Armadillo-610 と LCD 拡張ボードの接続

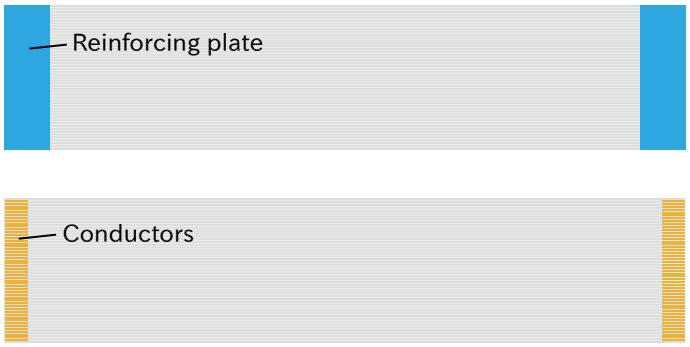


図 6.243 フレキシブルフラットケーブルの形状

必ず 1 ピンと 50 ピンが対応するように、接続してください。1 ピンと 1 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 の電極を上下逆に接続した場合、実装部品と電極が接触し、故障する可能性があります。

6.29.4.16. 形状図

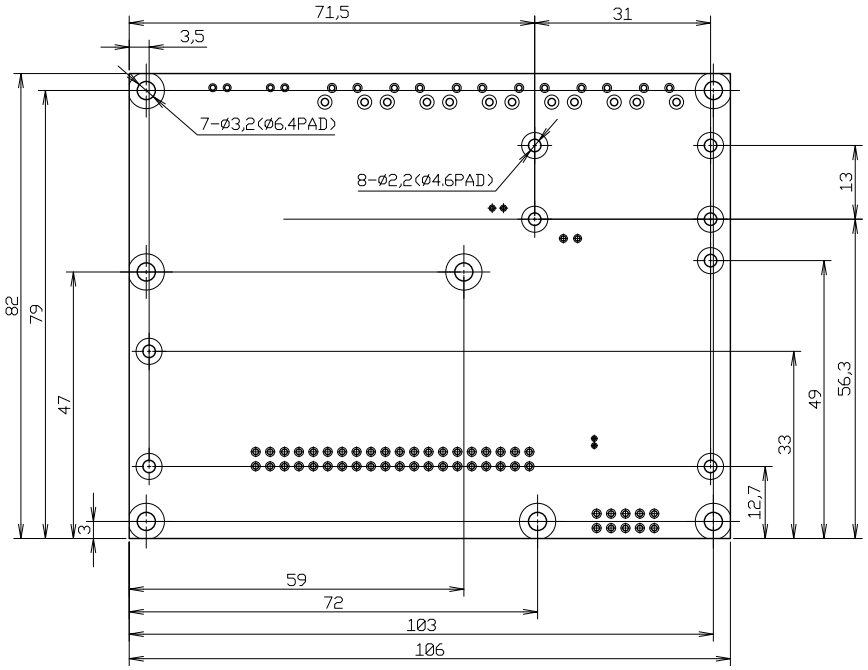


図 6.244 LCD 拡張ボードの基板形状および固定穴寸法

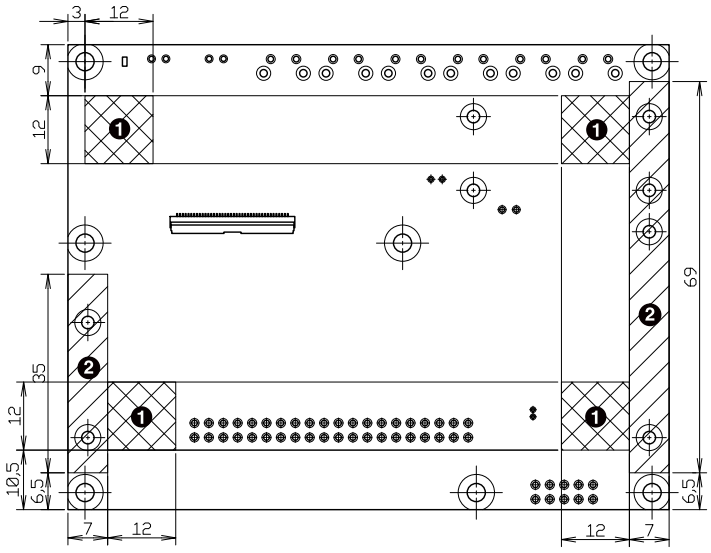


図 6.245 LCD 拡張ボードの両面テープと固定部品配置可能位置

❶ 両面テープ

❷ 固定部品

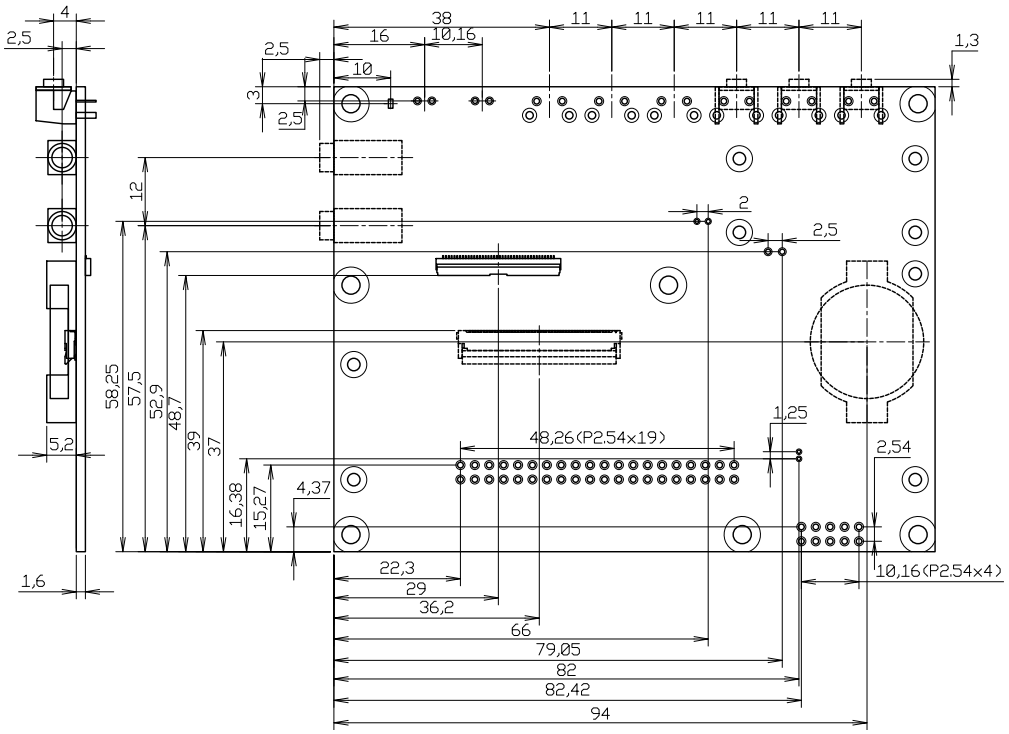


図 6.246 LCD 拡張ボードのコネクタ位置寸法

改訂履歴

バージョン	年月日	改訂内容
3.0.0	2023/06/29	・初版発行
3.1.0	2023/07/26	・Armadillo-610 拡張ボード参考回路の情報を Sterling LWB5+ 向けのものに修正 ・「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 ・「3.15. 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 ・「3.15. 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	・文章全体の構成を変更 ・「3.6.7. I2C デバイスを使用する」 の最大転送レートを 384kbps に修正 ・「3.15.6.2. 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	・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(バージョン以外の変更は無し)
4.0.1	2023/09/27	・「図 3.83. プロパティの保存」 を修正
4.1.0	2023/10/30	・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 ・「3.15. CUI アプリケーションの開発」 内にある 「3.15.3.2. 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 ・「3.12. 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	・「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

		・「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 ・「6.27.2.3. 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	・「表 2.3. 仕様」 にセキュアエレメントの項目を追加 ・「図 2.9. Armadillo-610 ブロック図」 のセキュアエレメントを I2C3→I2C1 へ修正 ・「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 ・「5.2.4. 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 ・「6.29.2.20. CON17(内蔵 RTC バックアップインターフェース)」に電源制御の方法を追記 ・「図 6.72. poweroff コマンドで電源を切断し、180 秒後に i.mx6ull の RTC 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	・「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 ・「3.15.3.2. ディレクトリ構成」 に requirements.txt に関する説明を追記 ・「3.16. 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	・「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	・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 ・「3.14. ABOSDE によるアプリケーションの開発」 を追加 ・「3.15. CUI アプリケーションの開発」 に 「3.15.3.5. Debian コンテナのディストリビューション」 を追加 ・「3.16. C 言語によるアプリケーションの開発」 に 「3.16.3.5. Debian 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	・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

		・「3.10. 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	・「3.15. CUI アプリケーションの開発」 に「3.15.5. コンテナ内のファイル一覧表示」 を追加 ・「3.16. C 言語によるアプリケーションの開発」 に「3.16.4. コンテナ内のファイル一覧表示」 を追加
4.9.0	2024/04/23	・「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 ・「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	・「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	・「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 ・「3.10. ABOS Web をカスタマイズする」 に対応バージョンを追記 ・「3.12. 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 ・「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 ・「表 6.37. Armadillo-610 拡張ボードのインターフェース一覧 」の電源入力インターフェース及び電源出力インターフェースのメーカー名を CUI Devices に変更

		・「表 6.78. LCD 拡張ボード インターフェース一覧 」のモノラルマイク入力及びステレオヘッドホン出力のメーカー名を CUI Devices に変更 ・「6.23.5. /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	・「3.8.4. ABOS Web のパスワード変更」 を追加 ・「3.15.3.7. 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	・「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 3.17. 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 ・「6.27.5.2. U-Boot の環境変数の変更を制限する」 を追加 ・「6.27.2. SBOM の提供」 を追加 ・ 誤記・わかりにくい文章の修正
4.14.0	2024/09/25	・「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 ・「3.1.4.3. インストールディスクを使用する」 の手順を変更 ・「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 ・「3.15.7. SBOM 生成に関する設定」 を追加 ・「3.16.7. SBOM 生成に関する設定」 を追加 ・「3.17. SBOM 生成に関わる設定を行う」 を追加 ・「3.20. 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 ・「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 6.33. 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	・「1.2.4. ソフトウェア使用に関しての注意事項」 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しての注意事項」 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 ・「5.9. Armadillo を廃棄する」 に詳細情報を追加 ・「表 6.37. Armadillo-610 拡張ボードのインターフェース一覧 」の「CON12:電源入力インターフェース」のメーカー名を Same Sky に変更 ・「表 6.37. Armadillo-610 拡張ボードのインターフェース一覧 」の「CON13A:電源出力インターフェース」「CON13B:DIDO インターフェース」「CON13C:RS-485 インターフェース」「CON13D:オーディオインターフェース」のメーカー名を Same Sky に変更 ・「表 6.78. LCD 拡張ボード インターフェース一覧 」 の「CON3:モノラルマイク入力」のメーカー名を Same Sky に変更 ・「表 6.78. LCD 拡張ボード インターフェース一覧 」 の「CON4:ステレオヘッドホン出力」のメーカー名を Same Sky に変更 ・「6.9.6.12. Rest API : VPN の設定」 を追加 ・誤記修正
4.16.0	2024/11/27	・「3.15.8. 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 ・「3.16.8. 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
4.17.0	2024/12/19	・「3.18. 生成した SBOM をスキャンする」 追加 ・誤記修正
4.18.0	2025/01/29	・「6.27.3. 不正な USB デバイスの接続を拒否する」 追加
4.19.0	2025/02/26	・「4.4.2. Armadillo Base OS の更新」 内の推奨手順を変更 ・libgpiod のバージョンアップに伴って、「3.6.6. GPIO を制御する」 内の手順を変更 ・「6.29.3. LCD オプションセット(7 インチタッチパネル WVGA 液晶)」 内で紹介している LCD の型番を変更 ・その他軽微な修正及び誤記修正
4.20.0	2025/03/26	・「3.6.6. GPIO を制御する」内で紹介している gpioset コマンドのオプションとして --daemonize の代わりに -t0 を使用する手順に変更 ・「3.1.4.1. 初期化インストールディスクの作成」 内のインストールディスクの作成方法を GUI アプリケーションを使用する手順に変更(従来の CUI で作成する手順は 「6.24. ATDE・Linux でインストールディスクを作成する」 に移動しました) ・「6.12.6. ファイアウォールの設定方法」に iptables サービスの有効化手順を追加 ・誤記修正
4.21.0	2025/04/23	・「6.9.11. ABOS Web のセキュリティ対策」 を追加 ・「2.1.3. Armadillo Base OS とは」 に OP-TEE に関する注釈を追加

		・VirtualBox の名称やライセンス情報などを更新 ・「4.4.7.1. JTAG と SD ブートを無効化する」 に JTAG と SD ブート無効化に対する注意喚起を追加 ・「周辺装置との接続」の節を削除し、必要な内容を適切な位置に移動 ・「5.6. Armadillo Twin を利用してソフトウェアの脆弱性チェックを行う」 を追加 ・ソフトウェアのバージョンの付け方に関する注意喚起を追記 ・誤記などの軽微な修正 ・html 版のマニュアルにあるロゴをクリックするとマニュアルのトップに戻るように変更 ・html 版のマニュアルの検索機能を削除
4.22.0	2025/05/28	・「3.9. USB デバイスの接続を許可する」 を追加 ・「3.1.5. Armadillo に初期設定をインストールする」 内に、Armadillo をインターネットに接続できない環境についての記述を追加 ・「6.1. persist_file について」 に、 OverlayFS の説明を追加 ・「6.27.3. 不正な USB デバイスの接続を拒否する」 内の、各種コマンドの使用方法を新しいバージョン向けに修正 ・「3.6.16. BT デバイスを使用する」 内に標準出荷状態では Bluetooth のプロトコルスタックを実装していないことを記載 ・その他軽微な修正
4.23.0	2025/06/25	・「6.27. セキュリティ」 を追加 ・「6.9.6.14. Rest API : 電源制御」 に sleep と inhibit_sleep に関する説明を追記 ・「3.11. ABOS Web から Armadillo の電源を操作する」 の節を記載 ・「3.5. Device Tree をカスタマイズする」 の節に GPIO の機能が自動的に割り当てられるのを防ぐ方法を記載 ・その他軽微な修正
4.24.0	2025/07/30	・「3.1.4.1. 初期化インストールディスクの作成」 で紹介する手順を Windows での手順に限定。従来の ATDE 及び Linux での作成方法は 「6.24. ATDE・Linux でインストールディスクを作成する」 を参照してください。 ・「3.1.7. シリアルコンソールを使用する」 で紹介するツールを Tera Term に変更。従来の minicom を用いたシリアルコンソールの操作方法については 「6.25. シリアル通信ソフトウェア (minicom)」 を参照してください。 ・「6.9.11.2. コンテナ以外のアクセスを制限」 を追加 ・「3.16.3.7. 実行ファイルのビルド」 を追加 ・「3.15.3.2. ディレクトリ構成」 に resources_python と python_launch に関する説明を追記 ・「3.8.4. ABOS Web のパスワード変更」に パスワードリセットに関する説明を追記 ・「3.10. ABOS Web をカスタマイズする」 に 製品型番のカスタマイズに関する説明を追記 ・「6.23.6. ABOS Web でログを確認する」 を追加 ・「6.27.6. コンテナの最小化」 を追加 ・誤記修正 ・その他軽微な修正
4.24.1	2025/08/28	・軽微な修正
4.25.0	2025/09/24	・「3.6.6. GPIO を制御する」 において、 sysfs GPIO クラスの使用が非推奨になったのに伴って当該の記載を削除

Armadillo-610 製品マニュアル

		・「6.9.11.2. コンテナ以外のアクセスを制限」 に command_args に渡すコマンド引数の説明を追加 ・「6.9.6.9. Rest API : WWAN の設定」 に 「LTE モジュールの再起動」 を追加 ・「6.9.4. SWU インストール」 に ABOS を最新バージョンに更新する機能の説明を追加 ・「3.9. USB デバイスの接続を許可する」 に 標準出荷状態で許可されている USB デバイスについての説明を追加 ・軽微な修正

Armadillo-610 製品マニュアル
Version 4.25.0
2025/09/24