Armadillo-IoT ゲートウェイ G4
製品マニュアル

AGX4500-C00D0
AGX4500-C00Z
AGX4500-U00Z
AGX4520-C02D0
AGX4520-C02Z
AGX4520-C03D0
AGX4520-C03Z
AGX4500-C03Z
AGX4500-U03Z

Version 2.8.0
2024/04/04

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-IoT ゲートウェイ G4 製品マニュアル

株式会社アットマークテクノ

製作著作 © 2021-2024 Atmark Techno, Inc.

Version 2.8.0
2024/04/04

目次

1. はじめに 23
1.1. 本書について 23
1.1.1. 本書で扱うこと 23
1.1.2. 本書で扱わないこと 23
1.1.3. 本書で必要となる知識と想定する読者 23
1.1.4. 本書の構成 24
1.1.5. フォント 25
1.1.6. コマンド入力例 25
1.1.7. アイコン 25
1.1.8. ユーザー限定コンテンツ 26
1.1.9. 本書および関連ファイルのバージョンについて 26
1.2. 注意事項 26
1.2.1. 安全に関する注意事項 26
1.2.2. 取扱い上の注意事項 27
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. 電波障害について 30
1.2.6. 保証について 30
1.2.7. 輸出について 31
1.2.8. 商標について 31
1.2.9. 無線モジュールの安全規制について 31
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-IoT ゲートウェイ G4 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 38
2.2. 製品ラインアップ 39
2.2.1. Armadillo-IoT ゲートウェイ G4 開発セット 40
2.2.2. Armadillo-IoT ゲートウェイ G4 量産用、量産ボード 40
2.3. 仕様 41
2.4. インターフェースレイアウト 43
2.5. ブロック図 46
2.6. 使用可能なストレージデバイス 48
2.7. ストレージデバイスのパーティション構成 49
2.8. ソフトウェアのライセンス 49
3. 開発編 50
3.1. アプリケーション開発の流れ 50
3.2. 開発前に知っておくべき Armadillo Base OS の機能・特徴 52
3.2.1. 一般的な Linux OS 搭載組み込み機器との違い 53
3.2.2. Armadillo Base OS 搭載機器のソフトウェア開発手法 54
3.2.3. アップデート機能について 54
3.2.4. ファイルの取り扱いについて 59
3.2.5. インストールディスクについて 61
3.3. 開発の準備 62
3.3.1. 準備するもの 62
3.3.2. 開発環境のセットアップ 63
3.3.3. Armadillo の起動 74

3.3.4. VSCode のセットアップ 84
3.3.5. VSCode を使用して Armadillo のセットアップを行う 86
3.3.6. ユーザー登録 88
3.4. ハードウェアの設計 89
3.4.1. 信頼性試験データについて 89
3.4.2. 放射ノイズ 89
3.4.3. ESD/雷サージ 89
3.4.4. 放熱 90
3.4.5. CON11、 CON12(拡張インターフェース) 91
3.4.6. 拡張ボードの設計 93
3.4.7. 回路設計 95
3.4.8. 電気的仕様 97
3.4.9. 形状図 104
3.4.10. オプション品 106
3.5. Device Tree をカスタマイズする 106
3.5.1. Linux カーネルソースコードの取得 106
3.5.2. at-dtweb のインストール 106
3.5.3. at-dtweb の起動 107
3.5.4. Device Tree をカスタマイズ 109
3.5.5. DT overlay によるカスタマイズ 114
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 117
3.6.1. SD カードを使用する 119
3.6.2. Ethernet を使用する 122
3.6.3. USB デバイスを使用する 125
3.6.4. JTAG デバッガを使用する 129
3.6.5. WLAN を使用する 129
3.6.6. BT デバイスを使用する 131
3.6.7. PCI Express デバイスを使用する 132
3.6.8. UART を使用する 133
3.6.9. HDMI を使用する 134
3.6.10. 音声出力を行う 136
3.6.11. LVDS を使用する 137
3.6.12. MIPI CSI-2 カメラを使用する 140
3.6.13. GPIO を制御する 143
3.6.14. I2C デバイスを使用する 145
3.6.15. SPI デバイスを使用する 147
3.6.16. CAN デバイスを使用する 148
3.6.17. PWM を使用する 149
3.6.18. I2S(SAI) を使用する 150
3.6.19. PDM マイクを使用する 150
3.6.20. RTC を使用する 151
3.6.21. 電源を入力する 153
3.6.22. 起動デバイスを変更する 154
3.6.23. ユーザースイッチを使用する 154
3.6.24. LED を使用する 156
3.6.25. LTE を使用する 158
3.6.26. Wi-SUN デバイスを扱う 160
3.6.27. EnOcean デバイスを扱う 160
3.7. ソフトウェアの設計 161
3.7.1. 開発者が開発するもの、開発しなくていいもの 161
3.7.2. ユーザーアプリケーションの設計 162
3.7.3. ログの設計 162
3.7.4. ウォッチドッグタイマー 163

3.8. ネットワーク設定 163
3.8.1. ABOS Web とは 163
3.8.2. ABOS Web へのアクセス 164
3.8.3. ABOS Web のパスワード登録 167
3.8.4. ABOS Web の設定操作 171
3.8.5. ログアウト 171
3.8.6. WWAN 設定 172
3.8.7. WLAN 設定 174
3.8.8. 各接続設定（各ネットワークインターフェースの設定） 178
3.8.9. DHCP サーバー設定 180
3.8.10. NAT 設定 180
3.8.11. 状態一覧 184
3.9. ABOS Web をカスタマイズする 184
3.10. Network Time Protocol (NTP, ネットワーク・タイム・プロトコル) の設定 187
3.11. Armadillo Twin を体験する 188
3.12. ABOSDE によるアプリケーションの開発 188
3.12.1. ABOSDE の対応言語 188
3.12.2. 参照する開発手順の章の選択 189
3.13. GUI アプリケーションの開発 190
3.13.1. Flutter とは 190
3.13.2. Flutter を用いた開発の流れ 190
3.13.3. ATDE 上でのセットアップ 191
3.13.4. コンテナのディストリビューション 199
3.13.5. Armadillo に転送するディレクトリ及びファイル 199
3.13.6. コンテナ内のファイル一覧表示 199
3.13.7. Armadillo 上でのセットアップ 211
3.13.8. アプリケーション開発 211
3.13.9. 動作確認 215
3.13.10. 製品への書き込み 222
3.13.11. Armadillo 上のコンテナイメージの削除 223
3.14. CUI アプリケーションの開発 223
3.14.1. CUI アプリケーション開発の流れ 223
3.14.2. ATDE 上でのセットアップ 223
3.14.3. アプリケーション開発 224
3.14.4. コンテナのディストリビューション 227
3.14.5. Armadillo に転送するディレクトリ及びファイル 228
3.14.6. コンテナ内のファイル一覧表示 228
3.14.7. Armadillo 上でのセットアップ 240
3.14.8. リリース版のビルド 244
3.14.9. 製品への書き込み 245
3.14.10. Armadillo 上のコンテナイメージの削除 245
3.15. C 言語によるアプリケーションの開発 245
3.15.1. C 言語によるアプリケーション開発の流れ 245
3.15.2. ATDE 上でのセットアップ 246
3.15.3. アプリケーション開発 247
3.15.4. コンテナのディストリビューション 251
3.15.5. コンテナ内のファイル一覧表示 251
3.15.6. Armadillo に転送するディレクトリ及びファイル 263
3.15.7. Armadillo 上でのセットアップ 263
3.15.8. リリース版のビルド 267
3.15.9. 製品への書き込み 268
3.15.10. Armadillo 上のコンテナイメージの削除 268
3.16. システムのテストを行う 268

3.16.1. ランニングテスト 268
3.16.2. 異常系における挙動のテスト 269
4. 量産編 270
4.1. 概略 270
4.1.1. Armadillo Twin を契約する 270
4.1.2. リードタイムと在庫 271
4.1.3. Armaidllo 納品後の製造・量産作業 271
4.2. BTO サービスを使わない場合と使う場合の違い 272
4.2.1. BTO サービスを利用しない(標準ラインアップ品) 272
4.2.2. BTO サービスを利用する 272
4.3. 量産時のイメージ書き込み手法 272
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 273
4.4.1. /etc/swupdate_preserve_file への追記 273
4.4.2. Armadillo Base OS の更新 274
4.4.3. パスワードの確認と変更 274
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 275
4.4.5. VSCode を使用して生成する 275
4.4.6. インストールディスクの動作確認を行う 278
4.4.7. コマンドラインから生成する 279
4.4.8. インストールの実行 283
4.5. SWUpdate を用いてイメージ書き込みする 283
4.5.1. SWU イメージの準備 283
4.5.2. desc ファイルの記述 284
4.6. イメージ書き込み後の動作確認 284
4.7. 量産時の組み立て 285
4.7.1. 拡張ボードの組み付け 285
4.7.2. オプションケース(金属製)への組み付け 285
5. 運用編 289
5.1. Armadillo Twin に Armadillo を登録する 289
5.1.1. Armadillo の設置前に登録する場合 289
5.1.2. Armadillo の設置後に登録する場合 289
5.2. Armadillo を設置する 289
5.2.1. 設置場所 289
5.2.2. ケーブルの取り回し 289
5.2.3. 無線 LAN/BT 用外付けアンテナの指向性 289
5.2.4. LTE 外付け用アンテナの指向性 290
5.2.5. LTE の電波品質に影響する事項 291
5.2.6. サージ対策 291
5.2.7. Armadillo の状態を表すインジケータ 291
5.2.8. 個体識別情報の取得 291
5.2.9. 電源を切る 294
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 294
5.3.1. アプリケーションのアップデート手順 294
5.4. Armadillo のソフトウェアをアップデートする 295
5.4.1. SWU イメージの作成 295
5.4.2. mkswu の desc ファイルを作成する 295
5.4.3. desc ファイルから SWU イメージを生成する 297
5.4.4. イメージのインストール 297
5.5. Armadillo Twin から複数の Armadillo をアップデートする 297
5.6. hawkBit サーバーから複数の Armadillo をアップデートする 298
5.6.1. hawkBit とは 298
5.6.2. データ構造 298
5.6.3. hawkBit サーバーから複数の Armadillo に配信する 298

5.7. eMMC の寿命を確認する 312
5.7.1. eMMC について 312
5.7.2. eMMC 予備領域の確認方法 312
5.8. Armadillo の部品変更情報を知る 312
5.9. Armadillo を廃棄する 313
6. 応用編 314
6.1. persist_file について 314
6.2. コンテナ 316
6.2.1. Podman - コンテナ仮想化ソフトウェアとは 316
6.2.2. コンテナの基本的な操作 316
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 331
6.2.4. コンテナ起動設定ファイルを作成する 333
6.2.5. アットマークテクノが提供するイメージを使う 339
6.2.6. alpine のコンテナイメージをインストールする 341
6.2.7. コンテナのネットワークを扱う 341
6.2.8. コンテナ内にサーバを構築する 343
6.2.9. 画面表示を行う 346
6.2.10. パワーマネジメント機能を使う 355
6.2.11. コンテナからの poweroff 及び reboot 358
6.2.12. 異常検知 358
6.2.13. NPU を扱う 359
6.3. swupdate がエラーする場合の対処 365
6.4. mkswu の .desc ファイルを編集する 365
6.4.1. インストールバージョンを指定する 365
6.4.2. Armadillo へファイルを転送する 366
6.4.3. Armadillo 上で任意のコマンドを実行する 366
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 366
6.4.5. 起動中の Armadillo で任意のコマンドを実行する 367
6.4.6. Armadillo にコンテナイメージを転送する 367
6.4.7. Armadillo のブートローダーを更新する 367
6.4.8. SWU イメージの設定関連 367
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 368
6.4.10. SWUpdate 実行中/完了後の挙動を指定する 368
6.4.11. desc ファイル設定例 368
6.5. swupdate_preserve_files について 371
6.6. SWU イメージの内容の確認 371
6.7. SWUpdate と暗号化について 371
6.8. Web UI から Armadillo をセットアップする (ABOS Web) 372
6.8.1. ABOS Web ではできないこと 372
6.8.2. ABOS Web の設定機能一覧と設定手順 372
6.8.3. コンテナ管理 373
6.8.4. SWU インストール 373
6.8.5. 時刻設定 375
6.8.6. アプリケーション向けのインターフェース (Rest API) 377
6.8.7. カスタマイズ 386
6.9. ABOSDE から ABOS Web の機能を使用する 386
6.9.1. Armadillo の SWU バージョンを取得する 388
6.9.2. Armadillo のコンテナの情報を取得する 388
6.9.3. Armadillo のコンテナを起動・停止する 389
6.9.4. Armadillo のコンテナのログを取得する 391
6.9.5. Armadillo に SWU をインストールする 391
6.10. VPU や NPU を使用する 392
6.10.1. Armadillo へ書き込むためのライブラリイメージを作成する 392

6.10.2. Armadillo にライブラリイメージを書き込む 393
6.10.3. ライブラリイメージのバージョンを確認する 393
6.10.4. コンテナ内からライブラリを使用するための準備 394
6.11. マルチメディアデータを扱う 395
6.11.1. GStreamer - マルチメディアフレームワーク 395
6.11.2. GStreamer 実行用コンテナを作成する 395
6.11.3. GStreamer パイプラインの実行例 396
6.11.4. 動画を再生する 397
6.11.5. ストリーミングデータを再生する 398
6.11.6. USB カメラからの映像を表示する 399
6.11.7. USB カメラからの映像を録画する 399
6.11.8. Video Processing Unit(VPU) 400
6.12. デモアプリケーションを実行する 402
6.12.1. コンテナを作成する 402
6.12.2. デモアプリケーションランチャを起動する 403
6.12.3. mediaplayer 404
6.12.4. video recoder 405
6.12.5. led switch tester 406
6.12.6. rtc tester 406
6.12.7. object detection demo 407
6.12.8. pose estimation demo 408
6.12.9. image segmentation demo 409
6.12.10. super resolution demo 410
6.12.11. hand estimation demo 411
6.12.12. screw detection demo 412
6.13. SMS を利用する 413
6.13.1. 初期設定 413
6.13.2. SMS を送信する 413
6.13.3. SMS を受信する 414
6.13.4. SMS 一覧を表示する 414
6.13.5. SMS の内容を表示する 414
6.13.6. SMS を削除する 415
6.13.7. SMS を他のストレージに移動する 415
6.14. ssh 経由で Armadillo Base OS にアクセスする 415
6.15. コマンドラインからネットワーク設定をする 416
6.15.1. 接続可能なネットワーク 416
6.15.2. IP アドレスの確認方法 416
6.15.3. ネットワークの設定方法 417
6.15.4. nmcli の基本的な使い方 417
6.15.5. 有線 LAN 421
6.15.6. 無線 LAN 422
6.15.7. 3G/LTE 423
6.16. コマンドラインから無線 LAN でネットワークを構成する 432
6.16.1. 無線 LAN アクセスポイント (AP) に接続する 433
6.16.2. 無線 LAN アクセスポイント (AP) として設定する 434
6.16.3. ルータとして設定する 437
6.17. ストレージの操作 439
6.17.1. ストレージ内にアクセスする 440
6.17.2. ストレージを安全に取り外す 440
6.17.3. ストレージのパーティション変更とフォーマット 441
6.18. ボタンやキーを扱う 442
6.18.1. SW1 の短押しと長押しの対応 442
6.18.2. USB キーボードの対応 443

6.18.3. Armadillo 起動時にのみボタンに反応する方法 444
6.19. 動作中の Armadillo の温度を測定する 444
6.19.1. 温度測定の重要性 444
6.19.2. atmark-thermal-profiler をインストールする 445
6.19.3. atmark-thermal-profiler を実行・停止する 445
6.19.4. atmark-thermal-profiler が出力するログファイルを確認する 445
6.19.5. 温度測定結果の分析 446
6.19.6. Armadillo Twin から Armadillo の温度を確認する 448
6.19.7. 温度センサーの仕様 448
6.20. Armadillo Base OS をアップデートする 448
6.21. ロールバック状態を確認する 448
6.22. Armadillo 起動時にコンテナの外でスクリプトを実行する 449
6.23. u-boot の環境変数の設定 449
6.24. SD ブートの活用 451
6.24.1. ブートディスクの作成 452
6.24.2. SD ブートの実行 454
6.25. Armadillo のソフトウェアをビルドする 455
6.25.1. ブートローダーをビルドする 455
6.25.2. Linux カーネルをビルドする 456
6.25.3. Alpine Linux ルートファイルシステムをビルドする 460
6.25.4. ビルドしたルートファイルシステムの SBOM を作成する 463
6.26. eMMC のデータリテンション 464
6.26.1. データリテンションの設定 464
6.26.2. より詳しくデータリテンションの統計情報を確認するには 466
6.26.3. 実装仕様に関する技術情報 467
6.27. Linux カーネルがクラッシュしたときにメモリの状態を保存する 468
6.27.1. Kdump を利用する準備 468
6.27.2. Kdump の動作確認 470
6.27.3. vmcore の確認 471
6.28. 動作ログ 473
6.28.1. 動作ログについて 473
6.28.2. 動作ログを取り出す 473
6.28.3. ログファイルのフォーマット 474
6.28.4. ログ用パーティションについて 474
6.29. vi エディタを使用する 474
6.29.1. vi の起動 474
6.29.2. 文字の入力 475
6.29.3. カーソルの移動 475
6.29.4. 文字の削除 476
6.29.5. 保存と終了 476
6.30. オプション品 476
6.30.1. オプションケース(金属製) 477
6.30.2. 3G/LTE 用外付けアンテナ、アンテナケーブル 483
6.30.3. 無線 LAN/BT 用外付けアンテナ、アンテナケーブル 484
6.30.4. Armadillo-X2、G4 ケースモデル VESA 規格固定用プレート 485

図目次

1.1. 製品化までのロードマップ 24
1.2. 3G/LTE モジュール:EC25-J 認証マーク 32
1.3. WLAN+BT コンボモジュール:AW-XM458 認証マーク 32
2.1. Armadillo-IoT ゲートウェイ G4 とは 34
2.2. エッジ AI 処理、機械学習の例 35
2.3. Armadillo Base OS とは 36
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 37
2.6. Armadillo Twin　とは 38
2.7. インターフェースレイアウト (ケース内部) 43
2.8. インターフェースレイアウト (ケース正面) 43
2.9. インターフェースレイアウト LTE モデル(ケース内部) 45
2.10. インターフェースレイアウト LTE モデル(ケース正面) 45
2.11. インターフェースレイアウト アンテナ接続位置 46
2.12. ブロック図 47
2.13. ブロック図(LTE モデル追加分) 48
3.1. アプリケーション開発の流れ 51
3.2. persist_file コマンド実行例 59
3.3. chattr によって copy-on-write を無効化する例 60
3.4. GNOME 端末の起動 69
3.5. GNOME 端末のウィンドウ 70
3.6. minicom の設定の起動 70
3.7. minicom の設定 70
3.8. minicom のシリアルポートの設定 71
3.9. 例. シリアル通信用 USB ケーブル(A-microB)接続時のログ 71
3.10. minicom のシリアルポートのパラメータの設定 72
3.11. minicom シリアルポートの設定値 72
3.12. minicom 起動方法 73
3.13. minicom 終了確認 73
3.14. Armadillo-IoT ゲートウェイ G4 の接続例 74
3.15. COM7 が競合している状態 75
3.16. Bletooth に割当の COM を変更した状態 76
3.17. 3G/LTE 用アンテナおよび nanoSIM カード接続例 77
3.18. JP1 の位置 78
3.19. ソフトウェアをアップデートする 84
3.20. VSCode を起動する 84
3.21. VSCode に開発用エクステンションをインストールする 85
3.22. ビルドツール実行前の準備 85
3.23. ビルドツールの実行 86
3.24. initial_setup.swu を作成する 87
3.25. initial_setup.swu 初回生成時の各種設定 87
3.26. Armadillo-IoT ゲートウェイ G4 の IC1 とヒートシンク固定穴の位置 90
3.27. Armadillo-IoT ゲートウェイ G4 の拡張インターフェース 94
3.28. Armadillo-IoT ゲートウェイ G4 の拡張ボード例 95
3.29. スイッチ、LED、リレー接続例 96
3.30. DC/DC コンバータ回路(VDD_5V 入力、3.3V 1.5A 出力)例 96
3.31. 1.8V ←→ 3.3V 双方向レベル変換回路の例 97
3.32. 電源回路の構成 100
3.33. 電源シーケンス 101
3.34. リセット回路の構成 102

3.35. リセットシーケンス 102
3.36. ONOFF 回路の構成 103
3.37. 基板形状図 104
3.38. 基板形状図(LTE モデル 追加分) 105
3.39. at-dtweb の起動開始 107
3.40. ボード選択画面 107
3.41. Linux カーネルディレクトリ選択画面 108
3.42. at-dtweb 起動画面 109
3.43. UART3(RXD/TXD) のドラッグ 109
3.44. CON11 8/10 ピンへのドロップ 110
3.45. 信号名の確認 110
3.46. プロパティの設定 111
3.47. プロパティの保存 111
3.48. 全ての機能の削除 112
3.49. I2C5(SCL/SDA) の削除 112
3.50. DTS/DTB の生成 113
3.51. DTS/DTB の生成完了 113
3.52. dtbo/desc の生成完了 114
3.53. /boot/overlays.txt の変更例 115
3.54. DT overlay を作成する例 116
3.55. Armadillo-IoT ゲートウェイ G4 のインターフェース 118
3.56. Armadillo-IoT ゲートウェイ G4 インターフェース LTE モデル追加分 119
3.57. CON1 microSD スロット 取り扱い上の注意事項 121
3.58. CON2 LAN LED 配置 123
3.59. CON3 LAN LED 配置 124
3.60. USB シリアルデバイスを扱うためのコンテナ作成例 126
3.61. setserial コマンドによる USB シリアルデバイス設定の確認例 126
3.62. USB カメラを扱うためのコンテナ作成例 127
3.63. USB メモリをホスト OS 側でマウントする例 127
3.64. 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 128
3.65. USB メモリに保存されているデータの確認例 128
3.66. USB メモリをマウントするためのコンテナ作成例 128
3.67. コンテナ内から USB メモリをマウントする例 128
3.68. WLAN+BT コンボモジュール アンテナケーブル接続用レセプタクルの配置 130
3.69. Bluetooth を扱うコンテナの作成例 131
3.70. Bluetooth を起動する実行例 131
3.71. bluetoothctl コマンドによるスキャンとペアリングの例 132
3.72. 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 134
3.73. 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 134
3.74. 音声出力を行うためのコンテナ作成例 137
3.75. alsa-utils による音声出力を行う例 137
3.76. CON10 接続可能なフレキシブルフラットケーブルの形状 141
3.77. GPIO を扱うためのコンテナ作成例 144
3.78. コンテナ内からコマンドで GPIO を操作する例 144
3.79. gpiodetect コマンドの実行 145
3.80. gpioinfo コマンドの実行 145
3.81. I2C を扱うためのコンテナ作成例 146
3.82. i2cdetect コマンドによる確認例 147
3.83. SPI を扱うためのコンテナ作成例 147
3.84. spi-config コマンドによる確認例 148
3.85. CAN を扱うためのコンテナ作成例 148
3.86. CAN の設定例 148
3.87. PWM を扱うためのコンテナ作成例 149

3.88. PWM の動作設定例 149
3.89. RTC を扱うためのコンテナ作成例 152
3.90. hwclock コマンドによる RTC の時刻表示と設定例 152
3.91. AC アダプタの極性マーク 153
3.92. 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 155
3.93. evtest コマンドによる確認例 155
3.94. LED を扱うためのコンテナ作成例 156
3.95. LED の点灯/消灯の実行例 157
3.96. LED の状態を表示する 157
3.97. 対応している LED トリガを表示 157
3.98. LED のトリガに heartbeat を指定する 158
3.99. EC25-J の状態表示 159
3.100. EC25-J をリセットまたは電源を入れる 160
3.101. EC25-J の電源を切る 160
3.102. Wi-SUN デバイスを扱うためのコンテナ作成例 160
3.103. EnOcean デバイスを扱うためのコンテナ作成例 161
3.104. 開発者が開発するもの、開発しなくていいもの 161
3.105. 現在の面の確認方法 162
3.106. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする 165
3.107. ABOSDE を使用して ABOS Web を開く 166
3.108. ABOSDE に表示されている Armadillo を更新する 167
3.109. パスワード登録画面 168
3.110. パスワード登録完了画面 169
3.111. ログイン画面 170
3.112. トップページ 171
3.113. WWAN 設定画面 173
3.114. WLAN クライアント設定画面 175
3.115. WLAN アクセスポイント設定画面 177
3.116. 現在の接続情報画面 178
3.117. LAN 接続設定で固定 IP アドレスに設定した画面 179
3.118. eth0 に対する DHCP サーバー設定 180
3.119. LTE を宛先インターフェースに指定した設定 181
3.120. LTE からの受信パケットに対するポートフォワーディング設定 182
3.121. VPN 設定 183
3.122. ABOS Web のカスタマイズ設定 185
3.123. メニュー変更画面 (一部) 187
3.124. chronyd のコンフィグの変更例 187
3.125. 参照する開発手順の章を選択する流れ 189
3.126. Flutter アプリケーションの例 190
3.127. Flutter アプリケーション開発の流れ 191
3.128. Flutter Demo アプリケーションの画面 192
3.129. GUI アプリケーションの画面 192
3.130. Signage アプリケーションの画面 192
3.131. Factory Signage アプリケーションの画面 193
3.132. GUI アプリケーションのプロジェクトを作成する 194
3.133. プロジェクト名を入力する 194
3.134. 初期設定を行う 195
3.135. VSCode で初期設定を行う 196
3.136. VSCode のターミナル 196
3.137. SSH 用の鍵を生成する 197
3.138. VSCode でコンテナイメージの作成を行う 198
3.139. コンテナイメージの作成完了 198
3.140. コンテナ内のファイル一覧を表示するタブ 199

3.141. コンテナ内のファイル一覧の例 200
3.142. resources ディレクトリ 201
3.143. コンテナ内のファイル一覧を再表示するボタン 202
3.144. container/resources 下にファイルを追加するボタン 203
3.145. ファイル名を入力 203
3.146. 追加されたファイルの表示 204
3.147. container/resources 下にフォルダーを追加するボタン 205
3.148. container/resources 下にあるファイルを開くボタン 206
3.149. container/resources 下にあるファイルを削除するボタン 207
3.150. コンテナ内のファイルを container/resources 下に保存するボタン 208
3.151. 編集前のファイルを示すマーク 209
3.152. 編集後のファイルを示すマーク 210
3.153. コンテナ内にコピーされないことを示すマーク 211
3.154. my_project へ移動して VSCode を起動する。 212
3.155. ATDE 上で Debug モードでビルドしたアプリケーションを実行する 212
3.156. 起動したサンプルアプリケーション 213
3.157. ATDE 上で Release モードでビルドしたアプリケーションを実行する 214
3.158. dart_periphery パッケージをインストールする例 214
3.159. video_player パッケージをインストールする例 215
3.160. dart_periphery パッケージをアンインストールする例 215
3.161. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする 216
3.162. ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する 217
3.163. ABOSDE に表示されている Armadillo を更新する 218
3.164. ssh_config を編集する 218
3.165. Armadillo 上で Debug モードでビルドしたアプリケーションを実行する 219
3.166. 実行時に表示されるメッセージ 219
3.167. アプリケーションを終了する 220
3.168. Armadillo 上で Release モードでビルドしたアプリケーションを実行する 221
3.169. ホットリロード機能を使う 221
3.170. SWU イメージを作成する 222
3.171. CUI アプリケーション開発の流れ 223
3.172. プロジェクトを作成する 224
3.173. プロジェクト名を入力する 224
3.174. VSCode で my_project を起動する 224
3.175. 初期設定を行う 225
3.176. VSCode で初期設定を行う 226
3.177. VSCode のターミナル 226
3.178. SSH 用の鍵を生成する 226
3.179. VSCode でコンテナイメージの作成を行う 227
3.180. コンテナイメージの作成完了 227
3.181. コンテナ内のファイル一覧を表示するタブ 228
3.182. コンテナ内のファイル一覧の例 229
3.183. resources ディレクトリ 230
3.184. コンテナ内のファイル一覧を再表示するボタン 231
3.185. container/resources 下にファイルを追加するボタン 232
3.186. ファイル名を入力 232
3.187. 追加されたファイルの表示 233
3.188. container/resources 下にフォルダーを追加するボタン 234
3.189. container/resources 下にあるファイルを開くボタン 235
3.190. container/resources 下にあるファイルを削除するボタン 236
3.191. コンテナ内のファイルを container/resources 下に保存するボタン 237
3.192. 編集前のファイルを示すマーク 238
3.193. 編集後のファイルを示すマーク 239

3.194. コンテナ内にコピーされないことを示すマーク 240
3.195. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする 241
3.196. ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する 242
3.197. ABOSDE に表示されている Armadillo を更新する 243
3.198. ssh_config を編集する 243
3.199. Armadillo 上でアプリケーションを実行する 244
3.200. 実行時に表示されるメッセージ 244
3.201. アプリケーションを終了する 244
3.202. リリース版をビルドする 245
3.203. C 言語によるアプリケーション開発の流れ 246
3.204. プロジェクトを作成する 247
3.205. プロジェクト名を入力する 247
3.206. VSCode で my_project を起動する 247
3.207. 初期設定を行う 248
3.208. VSCode で初期設定を行う 249
3.209. VSCode のターミナル 249
3.210. SSH 用の鍵を生成する 249
3.211. C 言語による開発における packages.txt の書き方 250
3.212. VSCode でコンテナイメージの作成を行う 251
3.213. コンテナイメージの作成完了 251
3.214. コンテナ内のファイル一覧を表示するタブ 252
3.215. コンテナ内のファイル一覧の例 252
3.216. resources ディレクトリ 253
3.217. コンテナ内のファイル一覧を再表示するボタン 254
3.218. container/resources 下にファイルを追加するボタン 255
3.219. ファイル名を入力 255
3.220. 追加されたファイルの表示 256
3.221. container/resources 下にフォルダーを追加するボタン 257
3.222. container/resources 下にあるファイルを開くボタン 258
3.223. container/resources 下にあるファイルを削除するボタン 259
3.224. コンテナ内のファイルを container/resources 下に保存するボタン 260
3.225. 編集前のファイルを示すマーク 261
3.226. 編集後のファイルを示すマーク 262
3.227. コンテナ内にコピーされないことを示すマーク 263
3.228. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする 264
3.229. ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する 265
3.230. ABOSDE に表示されている Armadillo を更新する 266
3.231. ssh_config を編集する 266
3.232. Armadillo 上でアプリケーションを実行する 267
3.233. 実行時に表示されるメッセージ 267
3.234. アプリケーションを終了する 267
3.235. リリース版をビルドする 268
3.236. メモリの空き容量の確認方法 269
4.1. Armadillo 量産時の概略図 270
4.2. BTO サービスで対応する範囲 272
4.3. 任意のファイルパスを/etc/swupdate_preserve_files に追記する 274
4.4. Armadillo Base OS をアップデートする 274
4.5. パスワードを変更する 274
4.6. make-installer.swu を作成する 276
4.7. 対象製品を選択する 276
4.8. make-installer.swu 生成時のログ 276
4.9. make-installer.swu インストール時のログ 277
4.10. 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 279

4.11. ip_config.txt の内容 281
4.12. IP アドレスの確認 282
4.13. allocated_ips.csv の内容 282
4.14. インストールログを保存する 283
4.15. インストールログの中身 283
4.16. Armadillo に書き込みたいソフトウェアを ATDE に配置 284
4.17. desc ファイルの記述例 284
4.18. オプションケース(金属製) 放熱シート貼付 285
4.19. オプションケース(金属製) 放熱シート貼付(WLAN+BT コンボモジュール搭載モデル) 286
4.20. オプションケース(金属製) ケース(下)ねじ止め 286
4.21. オプションケース(金属製) ケース嵌め合わせ 287
4.22. オプションケース(金属製) ケース(上)ねじ止め 288
5.1. 無線 LAN/BT 用外付けアンテナの指向性 290
5.2. LTE 外付け用アンテナの指向性 291
5.3. 個体番号の取得方法（device-info） 292
5.4. device-info のインストール方法 292
5.5. 個体番号の取得方法（get-board-info） 292
5.6. 個体番号の環境変数を conf ファイルに追記 293
5.7. コンテナ上で個体番号を確認する方法 293
5.8. MAC アドレスの確認方法 293
5.9. 出荷時の Ethernet MAC アドレスの確認方法 293
5.10. VScode を起動 294
5.11. 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 296
5.12. コンテナイメージアーカイブ作成例 296
5.13. sample_container_update.desc の内容 297
5.14. sample_container_update.desc の内容 297
5.15. hawkBit が扱うソフトウェアのデータ構造 298
5.16. hawkBit コンテナの TLS なしの場合（テスト用）の実行例 299
5.17. hawkBit コンテナの TLS ありの場合の実行例 300
5.18. eMMC の予備領域使用率を確認する 312
6.1. persist_file のヘルプ 314
6.2. persist_file 保存・削除手順例 315
6.3. persist_file ソフトウェアアップデート後も変更を維持する手順例 315
6.4. persist_file 変更ファイルの一覧表示例 315
6.5. persist_file でのパッケージインストール手順例 316
6.6. コンテナを作成する実行例 317
6.7. イメージ一覧の表示実行例 318
6.8. podman images --help の実行例 318
6.9. コンテナ一覧の表示実行例 318
6.10. podman ps --help の実行例 319
6.11. コンテナを起動する実行例 319
6.12. コンテナを起動する実行例(a オプション付与) 319
6.13. podman start --help 実行例 320
6.14. コンテナを停止する実行例 320
6.15. podman stop --help 実行例 320
6.16. my_container を保存する例 320
6.17. podman build の実行例 321
6.18. podman build でのアップデートの実行例 321
6.19. コンテナを削除する実行例 322
6.20. イメージを削除する実行例 323
6.21. podman rmi --help 実行例 323
6.22. Read-Only のイメージを削除する実行例 323
6.23. コンテナ内部のシェルを起動する実行例 324

6.24. コンテナ内部のシェルから抜ける実行例 324
6.25. podman exec --help 実行例 324
6.26. コンテナを作成する実行例 325
6.27. コンテナの IP アドレスを確認する実行例 325
6.28. ping コマンドによるコンテナ間の疎通確認実行例 325
6.29. pod を使うコンテナを自動起動するための設定例 326
6.30. network を使うコンテナを自動起動するための設定例 326
6.31. abos-ctrl podman-rw の実行例 328
6.32. abos-ctrl podman-storage のイメージコピー例 329
6.33. Armadillo 上のコンテナイメージを削除する 332
6.34. abos-ctrl container-clear 実行例 333
6.35. コンテナを自動起動するための設定例 333
6.36. ボリュームを shared でサブマウントを共有する例 335
6.37. /proc/devices の内容例 336
6.38. add_armadillo_env で設定した環境変数の確認方法 337
6.39. 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 340
6.40. Docker ファイルによるイメージのビルドの実行例 340
6.41. ビルド済みイメージを load する実行例 341
6.42. 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 341
6.43. コンテナの IP アドレス確認例 342
6.44. ip コマンドを用いたコンテナの IP アドレス確認例 342
6.45. ユーザ定義のネットワーク作成例 342
6.46. IP アドレス固定のコンテナ作成例 343
6.47. コンテナの IP アドレス確認例 343
6.48. コンテナに Apache をインストールする例 343
6.49. コンテナに lighttpd をインストールする例 344
6.50. コンテナに vsftpd をインストールする例 344
6.51. ユーザを追加する例 344
6.52. 設定ファイルの編集例 345
6.53. vsftpd の起動例 345
6.54. コンテナに samba をインストールする例 345
6.55. ユーザを追加する例 345
6.56. samba の起動例 346
6.57. コンテナに sqlite をインストールする例 346
6.58. sqlite の実行例 346
6.59. Wayland を扱うためのコンテナ作成例 347
6.60. コンテナ内で weston を起動したログの出力とアプリケーションの実行例 347
6.61. weston.ini 348
6.62. weston.ini をボリュームで渡す実行例 349
6.63. X Window System を扱うためのコンテナ起動例 351
6.64. コンテナ内で X Window System を起動する実行例 352
6.65. 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 353
6.66. フレームバッファに直接描画する実行例 353
6.67. タッチパネルを扱うためのコンテナ作成例 353
6.68. VPU を扱うためのコンテナ作成例 353
6.69. weston と GStreamer を扱うためのコンテナ作成例 354
6.70. GStreamer によるデコード実行例 354
6.71. GStreamer によるエンコード実行例 354
6.72. パワーマネジメント機能を使うためのコンテナ作成例 355
6.73. サスペンド状態にする実行例 355
6.74. サスペンド状態にする実行例、rtc で起こす 356
6.75. コンテナから shutdown を行う 358
6.76. 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 359

6.77. 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 359
6.78. 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 359
6.79. 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 359
6.80. NPU を扱うためのコンテナ作成例 360
6.81. ONNX Runtime をインストールする例 360
6.82. python から ONNX Runtime を使う例 361
6.83. TensorFlow Lite をインストールする例 361
6.84. python から TensorFlow Lite を使う例 361
6.85. tflite-runtime のバージョンを確認する 362
6.86. tflite-runtime のバージョンを下げる 363
6.87. at-debian-image のバージョンを確認する 364
6.88. Arm NN をインストールする例 364
6.89. python から Arm NN を使う例 364
6.90. コンテナ管理 373
6.91. SWU インストール 374
6.92. SWU 管理対象ソフトウェアコンポーネントの一覧表示 375
6.93. 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 375
6.94. 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 376
6.95. ネットワークタイムサーバーの設定項目 376
6.96. タイムゾーンの設定項目 376
6.97. 設定管理の Rest API トークン一覧表示 377
6.98. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする 387
6.99. ABOSDE の ABOS Web パスワード入力画面 388
6.100. ABOSDE で Armadillo の SWU バージョンを取得 388
6.101. ABOSDE で Armadillo のコンテナ情報を取得 389
6.102. ABOSDE で Armadillo のコンテナを起動 390
6.103. ABOSDE で Armadillo のコンテナを停止 390
6.104. ABOSDE で Armadillo のコンテナのログを取得 391
6.105. ABOSDE で Armadillo に SWU をインストール 392
6.106. ライブラリイメージ作成ツールの実行 393
6.107. ライブラリイメージ作成ツールの実行 (VPU が不要の場合) 393
6.108. ライブラリイメージを書き込む 393
6.109. ライブラリパーティションのマウント 393
6.110. ライブラリバージョンの確認 393
6.111. imx-mm へのシンボリックリンクを作成する 394
6.112. GStreamer を実行するためのコンテナ作成例 395
6.113. gstreamer のインストール 396
6.114. weston の起動 396
6.115. pulseaudio の起動 396
6.116. GStreamer の実行例 396
6.117. H.264/AVC 動画の再生(音声あり) 397
6.118. H.264/AVC 動画の再生(音声なし) 397
6.119. VP8 動画の再生(音声あり) 397
6.120. VP8 動画の再生(音声なし) 398
6.121. VP9 動画の再生(音声あり) 398
6.122. VP9 動画の再生(音声なし) 398
6.123. HTTP ストリーミングの再生(音声あり) 398
6.124. HTTP ストリーミングの再生(音声なし) 398
6.125. RTSP ストリーミングの再生(音声あり) 398
6.126. RTSP ストリーミングの再生(音声なし) 399
6.127. USB カメラからの映像表示(音声あり) 399
6.128. USB カメラからの映像表示(音声なし) 399
6.129. USB カメラからの映像を H.264 で録画(音声あり) 400

6.130. USB カメラからの映像を H.264 で録画(音声なし) 400
6.131. USB カメラからの映像を表示しながら H.264 で録画(音声あり) 400
6.132. USB カメラからの映像を表示しながら H.264 で録画(音声なし) 400
6.133. デモアプリケーションを実行するためのコンテナ作成例 402
6.134. デモアプリケーションランチャの起動 403
6.135. パッケージのバージョンを確認する 404
6.136. pulseaudio のインストールと起動 405
6.137. pillow のインストールと起動 407
6.138. ビデオデバイスの変更 407
6.139. ビデオデバイスの変更 408
6.140. ビデオデバイスの変更 409
6.141. ビデオデバイスの変更 410
6.142. ビデオデバイスの変更 411
6.143. 各種コンフィグの変更 412
6.144. 言語設定 413
6.145. SMS の作成 413
6.146. SMS 番号の確認 414
6.147. SMS の送信 414
6.148. SMS の一覧表示 414
6.149. SMS の内容を表示 414
6.150. SMS の削除 415
6.151. SIM カードのストレージに SMS を移動 415
6.152. 3G/LTE モジュールの内蔵ストレージに SMS を移動 415
6.153. IP アドレスの確認 416
6.154. IP アドレス(eth0)の確認 417
6.155. nmcli のコマンド書式 417
6.156. コネクションの一覧 418
6.157. コネクションの有効化 418
6.158. コネクションの無効化 418
6.159. コネクションの作成 418
6.160. コネクションファイルの永続化 418
6.161. コネクションの削除 419
6.162. コネクションファイル削除時の永続化 419
6.163. 固定 IP アドレス設定 420
6.164. DNS サーバーの指定 420
6.165. DHCP の設定 420
6.166. コネクションの修正の反映 420
6.167. デバイスの一覧 420
6.168. デバイスの接続 421
6.169. デバイスの切断 421
6.170. 有線 LAN の PING 確認 421
6.171. 無線 LAN の PING 確認 422
6.172. ネットワークインターフェースの優先度確認 423
6.173. ネットワークインターフェースの優先度設定 423
6.174. 3G/LTE のコネクションの作成 425
6.175. 3G/LTE のコネクションの設定の永続化 425
6.176. LTE モジュールの IMS 無効化コマンド 425
6.177. MCC/MNC を指定した 3G/LTE コネクションの作成 425
6.178. PAP 認証を有効にした 3G/LTE コネクションの作成 426
6.179. 3G/LTE モジュール初期化手順 426
6.180. 3G/LTE のコネクション確立 427
6.181. 3G/LTE の PING 確認 427
6.182. 3G/LTE コネクションを切断する 427

6.183. 再接続サービス 旧設定ファイルの削除 428
6.184. 3G/LTE 再接続サービスの設定値を永続化する 429
6.185. 3G/LTE 再接続サービスの状態を確認する 429
6.186. 3G/LTE 再接続サービスを停止する 429
6.187. 3G/LTE 再接続サービスを開始する 429
6.188. 3G/LTE 再接続サービスを無効にする 430
6.189. 3G/LTE 再接続サービスを有効にする 430
6.190. 認識されているモデムの一覧を取得する 431
6.191. モデムの情報を取得する 431
6.192. SIM の情報を取得する 431
6.193. 回線情報を取得する 432
6.194. 無線 LAN AP に接続する 433
6.195. nmtui を起動する 433
6.196. bridge インターフェースを作成する 435
6.197. mlan0 インターフェースを NetworkManager の管理から外す 435
6.198. hostapd.conf を編集する 435
6.199. dnsmasq の設定ファイルを編集する 437
6.200. ip_forward を有効にする 437
6.201. NAT を設定する 438
6.202. bridge に eth0 と eth1 を割り当てる 438
6.203. dnsmasq の設定ファイルを編集する 439
6.204. NAT を設定する 439
6.205. mount コマンド書式 440
6.206. ストレージのマウント 440
6.207. ストレージのアンマウント 440
6.208. fdisk コマンドによるパーティション変更 441
6.209. EXT4 ファイルシステムの構築 442
6.210. buttond で SW1 を扱う 442
6.211. buttond で USB キーボードのイベントを確認する 443
6.212. buttond で USB キーボードを扱う 443
6.213. buttond で SW1 を Armadillo 起動時のみ受け付ける設定例 444
6.214. atmark-thermal-profiler をインストールする 445
6.215. atmark-thermal-profiler を実行する 445
6.216. atmark-thermal-profiler を停止する 445
6.217. ログファイルの内容例 445
6.218. 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) 446
6.219. Armadillo-IoT ゲートウェイ G4 で取得した温度のグラフ 447
6.220. /var/at-log/atlog の内容の例 448
6.221. local サービスの実行例 449
6.222. uboot_env.d のコンフィグファイルの例 450
6.223. 自動マウントされた microSD カードのアンマウント 452
6.224. Linux カーネルを SWU でインストールする方法 458
6.225. Linux カーネルを build_rootfs でインストールする方法 459
6.226. データリテンション開始トリガーの方式 465
6.227. データリテンションの開始トリガーの動作例 466
6.228. 動作ログのフォーマット 474
6.229. vi の起動 474
6.230. 入力モードに移行するコマンドの説明 475
6.231. 文字を削除するコマンドの説明 476
6.232. オプションケース(金属製) 477
6.233. オプションケース(金属製)の加工痕例 478
6.234. LAN モデル ケース(上)形状図 479
6.235. LAN モデル ケース(下)形状図 480

6.236. LTE モデル ケース(上)形状図 480
6.237. LTE モデル ケース(下)形状図 481
6.238. WLAN モデル ケース(上)形状図 481
6.239. WLAN モデル ケース(下)形状図 482
6.240. LTE+WLAN モデル ケース(上)形状図 482
6.241. LTE+WLAN モデル ケース(下)形状図 483
6.242. 3G/LTE 用外付けアンテナ形状図 484
6.243. 3G/LTE 用アンテナケーブル形状図 484
6.244. 無線 LAN/BT 用外付けアンテナ形状図 485
6.245. 無線 LAN/BT 用アンテナケーブル形状図 485
6.246. Armadillo-X2、G4 ケースモデル VESA 規格固定用プレート 486
6.247. ケースに VESA 規格固定用プレートを取り付け 487
6.248. モニターに VESA 規格固定用プレートを取り付け 488
6.249. Armadillo-X2、G4 ケースモデル VESA 規格固定用プレート形状図 489

表目次

1.1. 使用しているフォント 25
1.2. 表示プロンプトと実行環境の関係 25
1.3. コマンド入力例での省略表記 25
1.4. 推奨温湿度環境について 29
1.5. 3G/LTE モジュール:EC25-J 適合証明情報 32
1.6. WLAN+BT コンボモジュール:AW-XM458 適合証明情報 32
2.1. Armadillo-IoT ゲートウェイ G4 ラインアップ 39
2.2. 仕様 41
2.3. 仕様(AGX4500-C03*, AGX4520-C03*) 42
2.4. インターフェース内容 43
2.5. インターフェース内容 45
2.6. ストレージデバイス 48
2.7. eMMC の GPP の用途 48
2.8. eMMC メモリマップ 49
2.9. eMMC ブートパーティション構成 49
2.10. eMMC GPP 構成 49
3.1. 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) 60
3.2. ユーザー名とパスワード 67
3.3. 動作確認に使用する取り外し可能デバイス 69
3.4. シリアル通信設定 70
3.5. CON11、CON12 搭載コネクタと対向コネクタ例 91
3.6. CON11 信号配列 91
3.7. CON12 信号配列 92
3.8. 各インターフェースへの電流供給例 96
3.9. 絶対最大定格 97
3.10. 推奨動作条件 98
3.11. 電源出力仕様 98
3.12. 拡張入出力ピンの電気的仕様(OVDD=VDD_3V3, VDD_1V8) 98
3.13. 拡張入出力ピンの電気的仕様(OVDD=VEXT_3V3) 99
3.14. オン状態、オフ状態を切り替えする際の Low レベル保持時間 103
3.15. Armadillo-IoT ゲートウェイ G4 インターフェース一覧 118
3.16. Armadillo-IoT ゲートウェイ G4 インターフェース一覧 LTE モデル追加分 119
3.17. CON1 信号配列 120
3.18. CON2 信号配列 (10BASE-T/100BASE-TX) 122
3.19. CON2 信号配列 (1000BASE-T) 123
3.20. CON2 LAN LED の動作 123
3.21. CON3 信号配列 (10BASE-T/100BASE-TX) 124
3.22. CON3 信号配列 (1000BASE-T) 124
3.23. CON3 LAN LED の動作 124
3.24. CON4 信号配列 126
3.25. CON7 信号配列 129
3.26. WLAN+BT コンボモジュール アンテナケーブル用レセプタクルの用途 130
3.27. CON6 信号配列 133
3.28. CON8 信号配列 135
3.29. CON9 搭載コネクタと対向コネクタ例 138
3.30. CON9 信号配列 138
3.31. 動作確認済み MIPI CSI-2 カメラデバイス一覧 140
3.32. CON10 搭載コネクタとフレキシブルフラットケーブル例 141
3.33. CON10 信号配列 141
3.34. MIPI CSI-2 カメラ用の DT overlay 143

3.35. I2C デバイス 146
3.36. CON13 信号配列 151
3.37. CON15 搭載コネクタと対向コネクタ例 153
3.38. CON15 信号配列 153
3.39. ジャンパの状態と起動デバイス 154
3.40. JP1 信号配列 154
3.41. SW1 信号配列 154
3.42. キーコード 155
3.43. LED3 の状態 156
3.44. LED4 の状態 156
3.45. LED クラスディレクトリと LED の対応 156
3.46. LED トリガの種類 157
3.47. CON23 信号配列 158
3.48. LED20 の接続 159
3.49. LED20 の状態 159
3.50. 用意する favicon 画像 186
3.51. ABOSDE の対応言語 189
3.52. 組み合わせて使うパッケージ 215
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 273
4.2. インストール中に実行される関数 280
5.1. EXT_CSD_PRE_EOL_INFO の値の意味 312
6.1. add_hotplugs オプションに指定できる主要な文字列 336
6.2. add_armadillo_env で追加される環境変数 337
6.3. 対応するパワーマネジメント状態 355
6.4. 対応するパワーマネジメント状態 357
6.5. ライブラリと tflite-runtime のバージョンと NPU を用いたアプリケーションの動作の関係 . 362
6.6. 2.6.0-1 未満の TensorFlow Lite 関連 deb パッケージ 363
6.7. ライブラリイメージ書き込み済みの製品 392
6.8. H.264/AVC デコーダー仕様 401
6.9. VP8 デコーダー仕様 401
6.10. VP9 デコーダー仕様 401
6.11. H.264/AVC エンコーダー仕様 401
6.12. デモアプリケーション動作のために必要なバージョン 404
6.13. ネジ検出デモのパラメータの詳細 413
6.14. ネットワークとネットワークデバイス 416
6.15. 固定 IP アドレス設定例 419
6.16. APN 情報設定例 424
6.17. 通信モジュールのネットワークデバイス 424
6.18. 再接続サービス設定パラメーター 429
6.19. thermal_profile.csv の各列の説明 446
6.20. u-boot の主要な環境変数 450
6.21. microSD カードのパーティション構成 453
6.22. build-rootfs のファイル説明 460
6.23. データリテンションの挙動 464
6.24. Armadillo のデータリテンションの設定 468
6.25. 入力モードに移行するコマンド 475
6.26. カーソルの移動コマンド 475
6.27. 文字の削除コマンド 476
6.28. 保存・終了コマンド 476
6.29. Armadillo-IoT ゲートウェイ G4 関連のオプション品 476
6.30. Armadillo-X2、G4 ケースモデル VESA 規格固定用プレートについて 486
6.31. Armadillo-X2、G4 ケースモデル VESA 規格固定用プレートの仕様 486

1. はじめに

このたびは Armadillo-IoT ゲートウェイ G4 をご利用いただき、ありがとうございます。Armadillo-IoT ゲートウェイ G4 は、各種センサーとネットワークとの接続を中継する IoT 向けゲートウェイの開発プラットフォームです。エッジ AI 処理、機械学習を低消費電力で実行する専用のハードウェア機能を搭載しており、高度な物体認識や情報の識別をおこなうシステムを開発することが可能です。さらに USB3.0、Gigabit Ethernet、MIPI-CSI、LVDS といった高速なインターフェースをそなえ、高付加価値なシステムの構築に利用いただけます。

Armadillo-IoT ゲートウェイ G4 には Linux ベースのディストリビューションとして専用設計の Armadillo Base OS を搭載しています。Armadillo Base OS はユーザーアプリケーションをコンテナとして管理する機能、Armadillo Base OS 自体とコンテナの両方を安全にリモートアップデートする機能、ネットワークや HW セキュリティに関する機能を集約したコンパクトな Armadillo 専用 OS です。

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

Armadillo Base OS とユーザーアプリケーションを含むコンテナはどちらも、Armadillo Base OS のリモートアップデート機能で安全にアップデートすることができます。Armadillo Base OS はアップデートの状態を 2 面化しているので電源やネットワークの遮断で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-IoT ゲートウェイ G4 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-IoT ゲートウェイ G4 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-IoT ゲートウェイ G4 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-IoT ゲートウェイ G4 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-IoT ゲートウェイ G4 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-IoT ゲートウェイ G4 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。

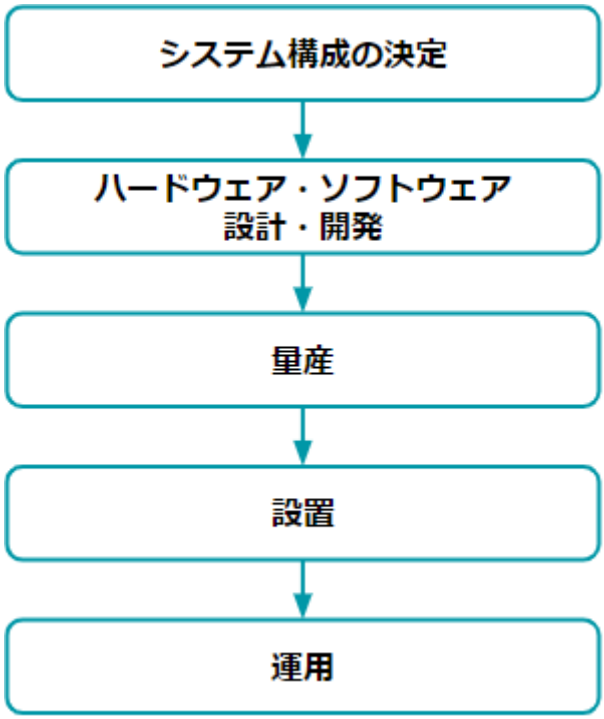


図 1.1 製品化までのロードマップ

- **「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、ハードウェアおよび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-IoT ゲートウェイ G4 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、「3.3.6. 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-IoT ゲートウェイ G4 ドキュメントダウンロード

https://armadillo.atmark-techno.com/armadillo-iot-g4/resources/documents

Armadillo サイト - Armadillo-IoT ゲートウェイ G4 ソフトウェアダウンロード

https://armadillo.atmark-techno.com/armadillo-iot-g4/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	LVDS、MIPI-CSI、HDMI、USB コンソールのコネクタは、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。
設置時の注意事項	人が触れて感電することが無いように 安全な場所に危険のないように設置してください。
発熱に関する注意事項	標準筐体は、筐体内部の熱を逃がす放熱板としての機能があるため内部温度上昇により高温になる場合があります。他の機器と重ねたり、付近に熱に弱い物体を近付けないでください。また、付近に他の熱源がある場合には、筐体を通して熱が伝わり筐体内部が発熱する可能性があるため、他の熱源からは遠ざけるように設置してください。
本製品の改造	本製品に改造[1]を行った場合は保証対象外となりますので十分ご注意ください。また、改造やコネクタ等の増設[2]を行う場合は、作業前に必ず動作確認を行ってください。
電源投入時のコネクタ着脱	本製品や周辺回路に電源が入っている状態で、活線挿抜対応インターフェース(LAN、USB、SD、HDMI) [3]以外へのコネクタ着脱は、絶対に行わないでください。
静電気	本製品には CMOS デバイスを使用しており、静電気により破壊されるおそれがあります。本製品を開封するときは、低湿度状態にならないよう注意し、静電防止用マットの使用、導電靴や人体アースなどによる作業者の帯電防止対策、備品の放電対策、静電気対策を施された環境下で行ってください。また、本製品を保管する際は、静電気を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。
電気通信事業法に関する注意事項について	本製品の有線 LAN を、電気通信事業者の通信回線(インターネットサービスプロバイダーが提供している通信網サービス等)に直接接続

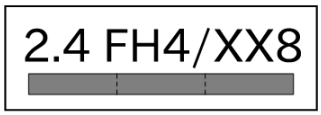
[1]本書を含めた関連マニュアルで改造方法を記載している箇所および、コネクタ非搭載箇所へのコネクタ等の増設は除く。
[2]改造やコネクタを増設する際にはマスキングを行い、周囲の部品に半田くず、半田ボール等付着しないよう十分にご注意ください。
[3]別途、活線挿抜を禁止している場合を除く

することはできません。接続する場合は、必ず電気通信事業法の認定を受けた端末設備(ルーター等)を経由して接続してください。

電波に関する注意事項(2.4GHz帯無線)

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

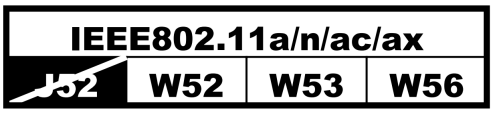
この無線機は 2.4GHz 帯を使用します。全帯域を使用し、かつ移動体識別装置の帯域が回避可能です。変調方式として DS-SS および OFDM 方式を採用し、想定される与干渉距離は 40m 以下です。



この無線機は 2.4GHz 帯を使用します。全帯域を使用し、かつ移動体識別装置の帯域が回避不可です。FH4 は、変調方式として FH-SS 方式を採用し、想定される与干渉距離は 40m 以下です。XX8 は、その他の方式を採用し、想定される与干渉距離は 80m 以下です。

電波に関する注意事項(5GHz帯無線)

この無線機は 5GHz 帯を使用します。W52、W53 の屋外での利用は電波法により禁じられています(5.2GHz 帯高出力データ通信システムのアクセスポイント／中継局と通信する場合を除く)。W53、W56 での AP モードは、2022 年 11 月現在工事設計認証を受けていないため使用しないでください。



1.2.3. 製品の保管について

・ 製品を在庫として保管するときは、高温・多湿、埃の多い環境、水濡れの可能性のある場所、直射日光のあたる場所、有毒ガス (特に腐食性ガス)の発生する場所を避け、精密機器の保管に適した状態で保管してください。

・ 保管環境として推奨する温度・湿度条件は以下のとおりです。

表 1.4 推奨温湿度環境について

推奨温湿度環境	5～35°C/70%RH 以下[a] [b]

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

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

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

1.2.4. ソフトウェア使用に関しての注意事項

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)

1.2.5. 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ください。VCCI-B

1.2.6. 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

1.2.7. 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。

1.2.8. 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。

- HDMI、HDMI ロゴ、High-Definition Multimedia Interface は HDMI Licensing, LLC の登録商標です

1.2.9. 無線モジュールの安全規制について

本製品に搭載されている無線モジュールは、電気通信事業法に基づく設計認証と電波法に基づく工事設計認証を受けています。

これらの無線モジュールを国内で使用するときに無線局の免許は必要ありません。

以下の事項を行うと法律により罰せられることがあります。

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

認証番号は次のとおりです。

表 1.5 3G/LTE モジュール:EC25-J 適合証明情報

項目	内容
型式又は名称	EC25-J
電波法に基づく工事設計認証における認証番号	018-190011
電気通信事業法に基づく設計認証における認証番号	ADF18-0088018

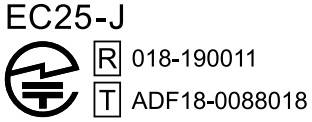


図 1.2 3G/LTE モジュール:EC25-J 認証マーク

表 1.6 WLAN+BT コンボモジュール:AW-XM458 適合証明情報

項目	内容
型式又は名称	AW-XM458
電波法に基づく工事設計認証における認証番号	D210142201
電気通信事業法に基づく設計認証における認証番号	201-210611

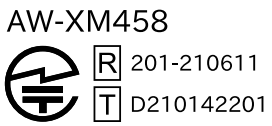


図 1.3 WLAN+BT コンボモジュール:AW-XM458 認証マーク

「表 1.6. WLAN+BT コンボモジュール:AW-XM458 適合証明情報」に示す認証番号が記載されている製品は、Dynamic Frequency Selection(DFS)機能 非対応となります。

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

2.1.2. Armadillo-IoT ゲートウェイ G4 とは

Armadillo-IoT ゲートウェイ G4 は「Armadillo-IoT ゲートウェイ」シリーズの製品です。

図 2.1 Armadillo-IoT ゲートウェイ G4 とは

Armadillo-IoT ゲートウェイ G4 には以下の特長があります。

・ NPU 搭載・エッジ AI 処理にも対応する高性能 IoT ゲートウェイ

Armadillo-IoT ゲートウェイ G4 は、エッジ AI 処理や機械学習にも最適な高性能 IoT ゲートウェイです。Gigabit Ethernet を 2 ポート搭載するほか、USB3.0、HDMI のインターフェースによる画像の入出力に対応。NPU 搭載により、高効率な演算を省電力で実現することができるため、顔認識や人物検知、製造・建築業における AI ソリューションなど、様々な現場で採用いただけます。

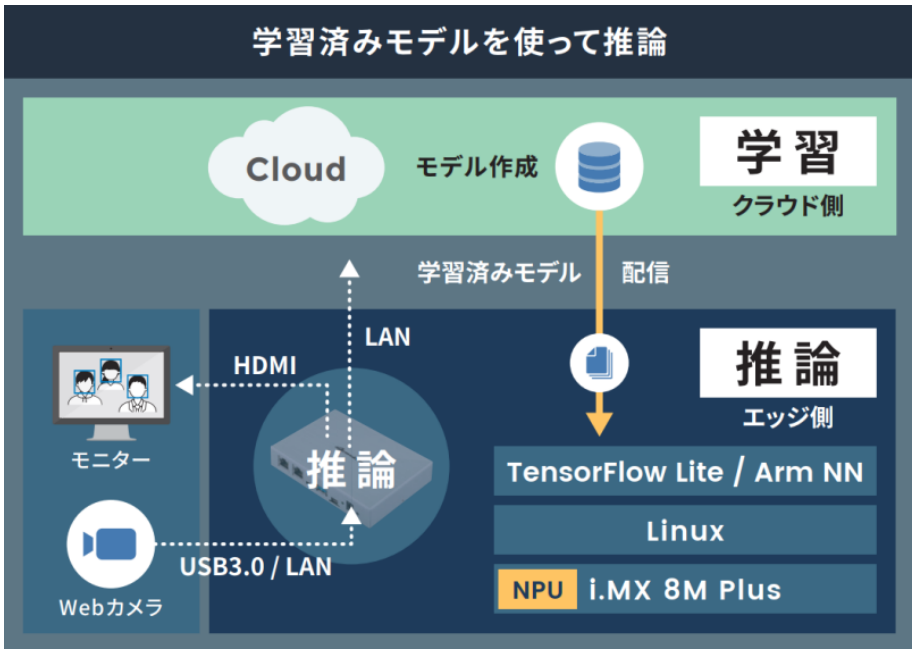


図 2.2 エッジ AI 処理、機械学習の例

- i.MX 8M Plus 搭載・動画を高速処理

  Arm Cortex-A53(1.6GHz)4 コアの SoC「i.MX 8M Plus」(NXP Semiconductors 製)を搭載しています。フル HD サイズ(1080p)の H.264 エンコード/デコード機能も用意されており、動画を記録しながらの AI 処理も可能です。

- 動作温度範囲-20～+70℃の産業設計

  高負荷のかかる AI 処理でも、動作温度範囲内であれば処理能力がほとんど低下しない稼働を見込めます。これまで設置が難しかった環境でも採用いただけるファンレス・小型設計で、産業用 PC よりも安価に導入することができます。

- Armadillo Base OS 搭載

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

- セキュアエレメント搭載

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

2.1.3. Armadillo Base OS とは

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

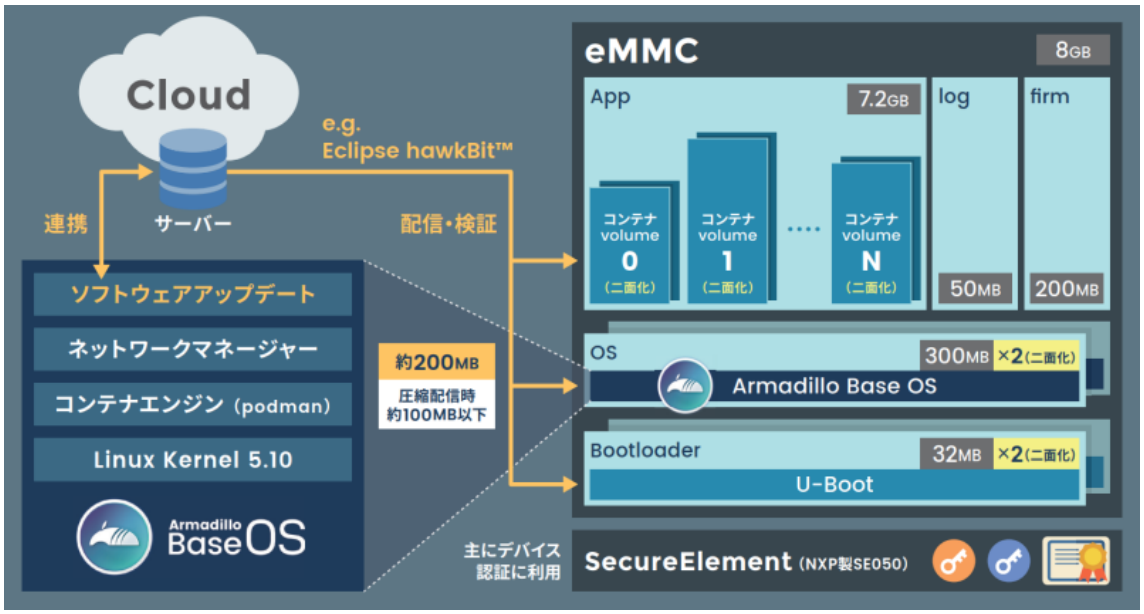


図 2.3 Armadillo Base OS とは

- 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。

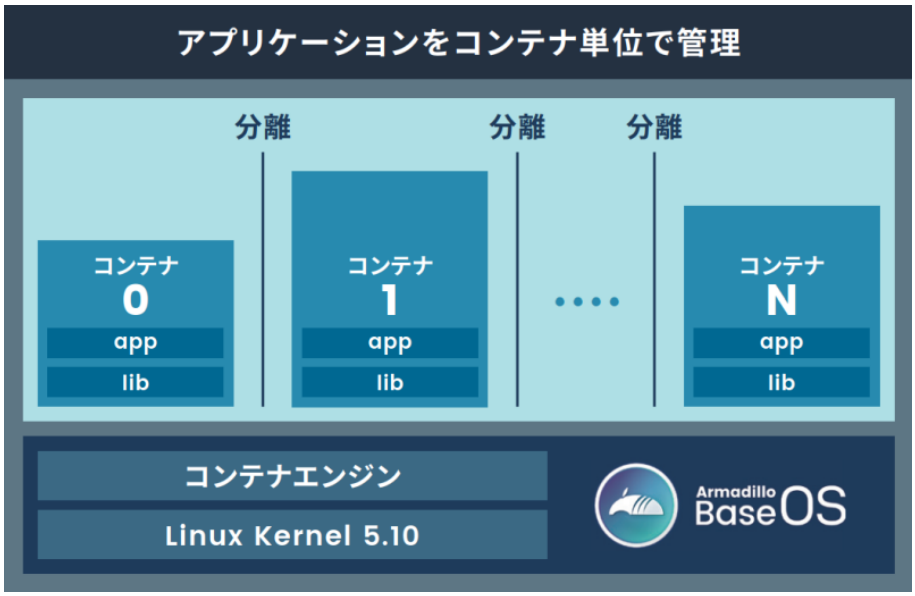


図 2.4 コンテナによるアプリケーションの運用

- 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。

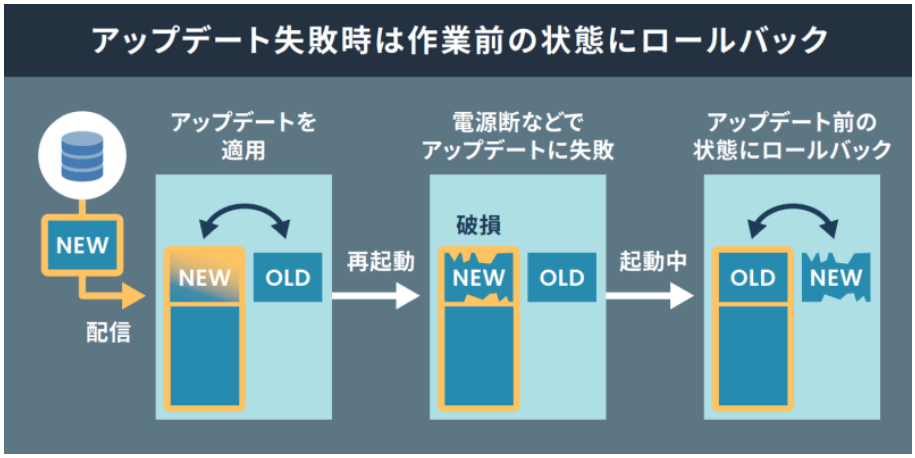


図 2.5 ロールバックの仕組み

- 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」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。

> **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、「3.2. 開発前に知っておくべき Armadillo Base OS の機能・特徴」 を参照してください。

2.1.5. Armadillo Twin とは

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

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

図 2.6 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[a]
個体管理	デバイス登録(デバイス証明書を利用)、ラベル付け、デバイスグループ化機能
ユーザ管理	ユーザーの追加/削除、ユーザー権限の設定
お知らせ	セキュリティアップデート、システム障害通知

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

> **Armadillo Twin**
>
> サービス利用料金など、その他詳細については下記概要ページをご覧ください。
>
> https://armadillo.atmark-techno.com/guide/armadillo-twin

2.2. 製品ラインアップ

Armadillo-IoT ゲートウェイ G4 の製品ラインアップは次のとおりです。

表 2.1 Armadillo-IoT ゲートウェイ G4 ラインアップ

名称	型番
Armadillo-IoT ゲートウェイ G4 LAN モデル開発セット (メモリ 2GB、ストレージ 10GB、WLAN コンボ非搭載)	AGX4500-C00D0
Armadillo-IoT ゲートウェイ G4 LAN モデル量産用 (メモリ 2GB、ストレージ 10GB、WLAN コンボ非搭載)	AGX4500-C00Z
Armadillo-IoT ゲートウェイ G4 LAN モデル量産ボード (メモリ 2GB、ストレージ 10GB、WLAN コンボ非搭載)	AGX4500-U00Z
Armadillo-IoT ゲートウェイ G4 LTE モデル開発セット (メモリ 2GB、ストレージ 10GB、WLAN コンボ非搭載)	AGX4520-C02D0
Armadillo-IoT ゲートウェイ G4 LTE モデル量産用 (メモリ 2GB、ストレージ 10GB、LTE アンテナセット付属、WLAN コンボ非搭載)	AGX4520-C02Z
Armadillo-IoT ゲートウェイ G4 LTE+WLAN モデル 開発セット (メモリ 2GB、ストレージ 10GB)	AGX4520-C03D0
Armadillo-IoT ゲートウェイ G4 LTE+WLAN モデル 量産用 (メモリ 2GB、ストレージ 10GB、LTE アンテナセット付属、WLAN コンボ搭載)	AGX4520-C03Z
Armadillo-IoT ゲートウェイ G4 WLAN モデル 量産用 (メモリ 2GB、ストレージ 10GB、WLAN コンボ搭載)	AGX4500-C03Z
Armadillo-IoT ゲートウェイ G4 WLAN モデル 量産ボード (メモリ 2GB、ストレージ 10GB、WLAN コンボ搭載)	AGX4500-U03Z

2.2.1. Armadillo-IoT ゲートウェイ G4 開発セット

Armadillo-IoT ゲートウェイ G4 を使った開発がすぐに開始できるように、開発に必要なものを一式含んだ製品をラインアップしています。

Armadillo-IoT ゲートウェイ G4 LAN モデル開発セットのセット内容は以下のとおりです。

- Armadillo-IoT ゲートウェイ G4
- オプションケース(金属製)
- USB(A オス-microB)ケーブル
- AC アダプタ(12V/3.0A)
- ジャンパソケット

Armadillo-IoT ゲートウェイ G4 LTE モデル開発セットのセット内容は以下のとおりです。

- Armadillo-IoT ゲートウェイ G4
- オプションケース(金属製)
- USB(A オス-microB)ケーブル
- AC アダプタ(12V/3.0A)
- ジャンパソケット
- 3G/LTE 用外付けアンテナ

Armadillo-IoT ゲートウェイ G4 LTE+WLAN モデル開発セットのセット内容は以下のとおりです。

- Armadillo-IoT ゲートウェイ G4
- オプションケース(金属製)
- USB(A オス-microB)ケーブル
- AC アダプタ(12V/3.0A)
- ジャンパソケット
- 3G/LTE 用外付けアンテナ
- 無線 LAN/BT 用外付けアンテナ

2.2.2. Armadillo-IoT ゲートウェイ G4 量産用、量産ボード

Armadillo-IoT ゲートウェイ G4 の量産用に、必要最小限の内容物に絞った製品をラインアップしています。

「Armadillo-IoT ゲートウェイ G4 LAN モデル量産用」、「Armadillo-IoT ゲートウェイ G4 LTE モデル量産用」、「Armadillo-IoT ゲートウェイ G4 LTE+WLAN モデル 量産用」、「Armadillo-IoT ゲートウェイ G4 WLAN モデル 量産用」がケースに収められた製品、「Armadillo-IoT ゲートウェイ G4 LAN モデル量産ボード」、「Armadillo-IoT ゲートウェイ G4 WLAN モデル 量産ボード」が基板単体の製品となります。

こちらには、AC アダプタ、ケーブル類は付属しておりませんので、適宜必要となるものをご用意ください。

2.3. 仕様

Armadillo-IoT ゲートウェイ G4 の主な仕様は次のとおりです。

表 2.2 仕様

型番	AGX4500-C00D0, AGX4500-C00Z, AGX4500-U00Z	AGX4520-C02D0, AGX4520-C02Z
CPU	NXP Semiconductors i.MX 8M Plus	
	Arm Cortex-A53 × 4 ・命令/データキャッシュ 32kByte/32kByte ・L2 キャッシュ 512kByte ・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 Arm Cortex-M7 × 1 ・命令/データキャッシュ 32kByte/32kByte ・TCM 256kByte	
システムクロック	CPU コアクロック(Arm Cortex-A53): 1.6GHz CPU コアクロック(Arm Cortex-M7): 800MHz DDR クロック: 2GHz 源発振クロック: 32.768kHz、24MHz	
NPU	2.3 TOPS	
RAM	LPDDR4: 2GByte バス幅: 32bit	
ROM	eMMC: 9.8GiB [a] HS400(最大転送速度: 400MB/s)	
LAN(Ethernet)	1000BASE-T × 2 AUTO-MDIX 対応	
モバイル通信	非搭載	3G/LTE モジュール搭載 外付けアンテナ付属 nanoSIM スロット × 1
USB	USB 3.0 Host × 1 (Type-A)	
SD	microSD スロット × 1 UHS-I	
ビデオ	HDMI 出力 × 1 (micro Type-D) LVDS 出力 (4 レーン) × 1 [b]	
オーディオ	HDMI 出力 × 1 (micro Type-D)	
カメラ	MIPI CSI-2 (2 レーン) × 1 [b]	
拡張インターフェース [b] [c]	USB 2.0 × 1、GPIO × 34、SPI × 2、UART × 2、 PDM MIC × 4、I2S × 1、CAN x 2、I2C × 3、PWM × 4	PWM × 1、PDM MIC × 1、GPIO × 13
カレンダー時計	リアルタイムクロック [d]: 平均月差 8 秒(周囲温度-20℃~70℃における参考値)	
スイッチ	ユーザースイッチ × 1	
LED	ユーザー LED × 1 電源 LED × 1 [b]	ユーザー LED × 1 電源 LED × 1 [b] 3G/LTE ステータス確認用 LED × 1
メンテナンスポート	USB micro-B シリアルコンソール	
セキュアエレメント	NXP Semiconductors SE050	
電源電圧	DC 12V±10%	
消費電力(参考値)	2.5W(定常状態) [e]	3.0W(定常状態) [e]、 6.0W(LANx2、LTE 通信時) [e] [f] 8.5W(LANx2、LTE 最大出力時) [e] [g]
動作温度範囲	-20～+70℃(結露なきこと)	-20～+60℃(結露なきこと)
外形サイズ(基板)	135×95mm(突起部を除く)	

型番	AGX4500-C00D0, AGX4500-C00Z, AGX4500-U00Z	AGX4520-C02D0, AGX4520-C02Z
外形サイズ(ケース)	143×100.5×26mm(突起部を除く)	

[a]pSLC モードで動作します。

[b]ケース装着時はケース外部から利用できません。

[c]拡張インターフェース(CON11、CON12)の信号線において、i.MX 8M Plus 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。

[d]コイン電池によるバックアップが可能です。電池は付属していません。

[e]外部接続機器の消費分は含みません。

[f]LTE の電波品質が 80% の時の値となります。

[g]LTE 出力が 23dBm の時の値となります。

表 2.3 仕様(AGX4500-C03*, AGX4520-C03*)

型番	AGX4500-C03Z, AGX4500-U03Z	AGX4520-C03D0, AGX4520-C03Z
CPU	NXP Semiconductors i.MX 8M Plus	
	Arm Cortex-A53 × 4 ・命令/データキャッシュ 32kByte/32kByte ・L2 キャッシュ 512kByte ・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 Arm Cortex-M7 × 1 ・命令/データキャッシュ 32kByte/32kByte ・TCM 256kByte	
システムクロック	CPU コアクロック(Arm Cortex-A53): 1.6GHz CPU コアクロック(Arm Cortex-M7): 800MHz DDR クロック: 2GHz 源発振クロック: 32.768kHz、24MHz	
NPU	2.3 TOPS	
RAM	LPDDR4: 2GByte バス幅: 32bit	
ROM	eMMC: 9.8GiB [a] HS400(最大転送速度: 400MB/s)	
LAN(Ethernet)	1000BASE-T × 2 AUTO-MDIX 対応	
無線 LAN	WLAN + BT コンボモジュール 外付けアンテナ付属 IEEE 802.11a/b/g/n/ac/ax	
モバイル通信	非搭載	3G/LTE モジュール搭載 外付けアンテナ付属 nanoSIM スロット × 1
USB	USB 3.0 Host × 1 (Type-A)	
SD	microSD スロット × 1 UHS-I	
ビデオ	HDMI 出力 × 1 (micro Type-D) LVDS 出力 (4 レーン) × 1 [b]	
オーディオ	HDMI 出力 × 1 (micro Type-D)	
カメラ	MIPI CSI-2 (2 レーン) × 1 [b]	
拡張インターフェース [b] [c]	USB 2.0 × 1、GPIO × 34、SPI × 2、UART × 2、 PDM MIC × 4、I2S × 1、CAN x 2、I2C × 3、PWM × 4	PWM × 1、PDM MIC × 1、GPIO × 13
カレンダー時計	リアルタイムクロック [d]: 平均月差 8 秒(周囲温度-20℃~70℃における参考値)	
スイッチ	ユーザースイッチ × 1	
LED	ユーザー LED × 1 電源 LED × 1 [b]	ユーザー LED × 1 電源 LED × 1 [b] 3G/LTE ステータス確認用 LED × 1
メンテナンスポート	USB micro-B シリアルコンソール	
セキュアエレメント	NXP Semiconductors SE050	

型番	AGX4500-C03Z, AGX4500-U03Z	AGX4520-C03D0, AGX4520-C03Z
電源電圧	DC 12V±10%	
消費電力(参考値)	約 3.9W (定常状態)、 約 5W (WLAN 通信時)	
動作温度範囲	-20～+60℃(結露なきこと)	
外形サイズ(基板)	135×95mm(突起部を除く)	
外形サイズ(ケース)	143×100.5×26mm(突起部を除く)	

[a]pSLC モードで動作します。

[b]ケース装着時はケース外部から利用できません。

[c]拡張インターフェース(CON11、CON12)の信号線において、i.MX 8M Plus 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。

[d]コイン電池によるバックアップが可能です。電池は付属していません。

2.4. インターフェースレイアウト

Armadillo-IoT ゲートウェイ G4 のインターフェースレイアウトです。各インターフェースの配置場所等を確認してください。

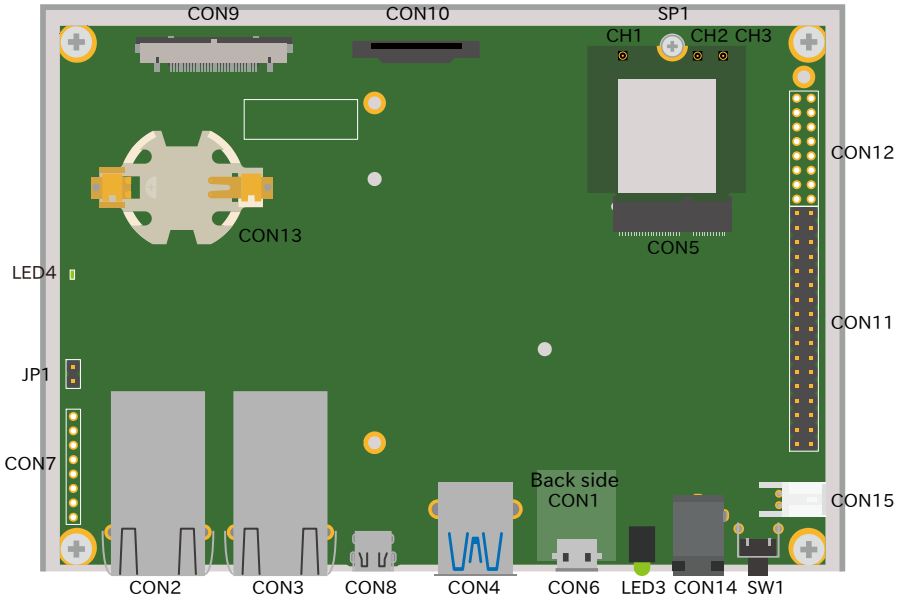


図 2.7 インターフェースレイアウト (ケース内部)

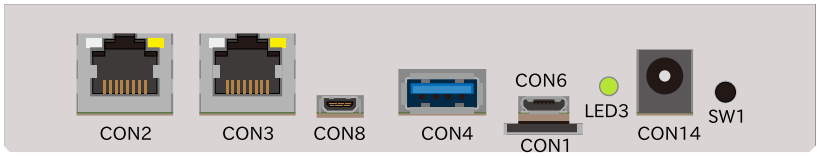


図 2.8 インターフェースレイアウト (ケース正面)

表 2.4 インターフェース内容

部品番号	インターフェース名	説明
CON1	SD インターフェース	microSD カード用の SD スロットです。外部ストレージが必要な場合や、microSD カードに配置したブートディスクイメージから起動する場合などに使用します。

部品番号	インターフェース名	説明
CON2	LAN インターフェース 2	LAN ケーブル接続用の RJ-45 コネクタです。有線 LAN を利用する場合に使用します。
CON3	LAN インターフェース 1	LAN ケーブル接続用の RJ-45 コネクタです。有線 LAN を利用する場合に使用します。
CON4	USB インターフェース	USB メモリや USB 接続デバイス接続用の USB3.0 Type-A コネクタです。外部ストレージが必要な場合や、USB 接続の各種デバイスを使用する場合に使用します。
CON5	M.2 インターフェース	無線 LAN/BT モジュール接続用の M.2 コネクタ(E-Key)です。無線 LAN/BT コンボモジュール搭載モデルの場合は、無線 LAN/BT コンボモジュールが接続されます。
CON6	USB コンソールインターフェース	USB microB ケーブル接続用の USB micro-B コネクタです。コンソール入出力を利用する場合に使用します。
CON7	JTAG インターフェース	1 列 8 ピン(2.54mm ピッチ)のピンヘッダを搭載可能です。JTAG を利用する場合に使用します。
CON8	HDMI インターフェース	HDMI Type-D ケーブル接続用の HDMI Type-D コネクタです。HDMI 対応ディスプレイ等を利用する場合に使用します。
CON9	LVDS インターフェース	LVDS 出力用 31 ピンコネクタです。LVDS 対応ディスプレイ等を利用する場合に使用します。
CON10	MIPI-CSI インターフェース	MIPI CSI-2 対応カメラ接続用の 15 ピン(1mm ピッチ)FFC コネクタです。MIPI CSI-2 対応カメラを利用する場合に使用します。
CON11	拡張インターフェース 1	2 列 17 ピン(2.54mm ピッチ)のピンヘッダが搭載されています。機能拡張する場合に使用します。3G/LTE モジュール搭載モデルでは使用できません。
CON12	拡張インターフェース 2	2 列 8 ピン(2.54mm ピッチ)のピンヘッダを搭載可能です。機能拡張する場合に使用します。
CON13	RTC バックアップインターフェース	CR2032 等のコイン形電池接続用の電池ボックスです。リアルタイムクロックのバックアップ給電が必要な場合に使用します。
CON14	電源入力インターフェース 1	AC アダプタ接続用 DC ジャックです。付属の AC アダプタから Armadillo-IoT ゲートウェイ G4 へ電源供給する場合に使用します[a]。
CON15	電源入力インターフェース 2	電源入力用 2 ピン(2mm ピッチ)ライトアングルコネクタです。AC アダプタ以外の電源装置からら Armadillo-IoT ゲートウェイ G4 へ電源供給する場合に使用します[a]。
JP1	起動デバイス設定ジャンパ	起動モードを設定するための 2 ピン(2.54mm ピッチ)ピンヘッダです。
SW1	ユーザースイッチ	ユーザーが利用可能なタクトスイッチです。
LED3	ユーザー LED	ユーザーが利用可能な砲弾タイプ(Φ3mm)の緑色 LED です。
LED4	電源 LED	電源の入力状態を表示する表面実装タイプの緑色 LED です。
SP1	M.2 用スタッド	無線 LAN/BT モジュール接続用のスペーサー(M2、L=2.45mm)です。無線 LAN/BT コンボモジュール搭載モデルの場合は、無線 LAN/BT コンボモジュールのネジ止めに使用されます。
CH1	無線 LAN アンテナインターフェース 1	無線 LAN/BT 用外付けアンテナケーブルを接続するための小型同軸コネクタです。
CH2	無線 LAN アンテナインターフェース 2	無線 LAN/BT 用外付けアンテナケーブルを接続するための小型同軸コネクタです。
CH3	BT アンテナインターフェース	無線 LAN/BT 用外付けアンテナケーブルを接続するための小型同軸コネクタです。

[a]電源入力インターフェース 1 と電源入力インターフェース 2 の両方から同時に電源供給することはできません。

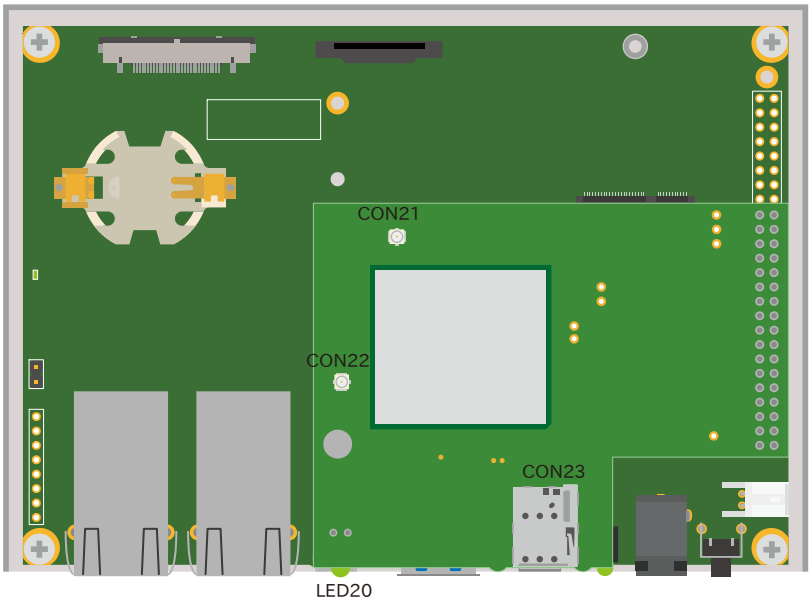


図 2.9 インターフェースレイアウト LTE モデル(ケース内部)

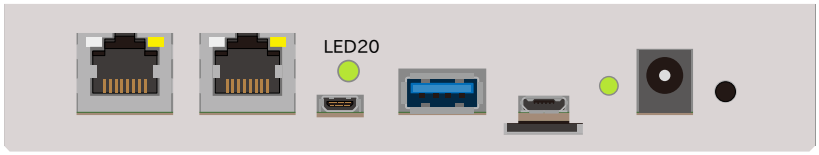
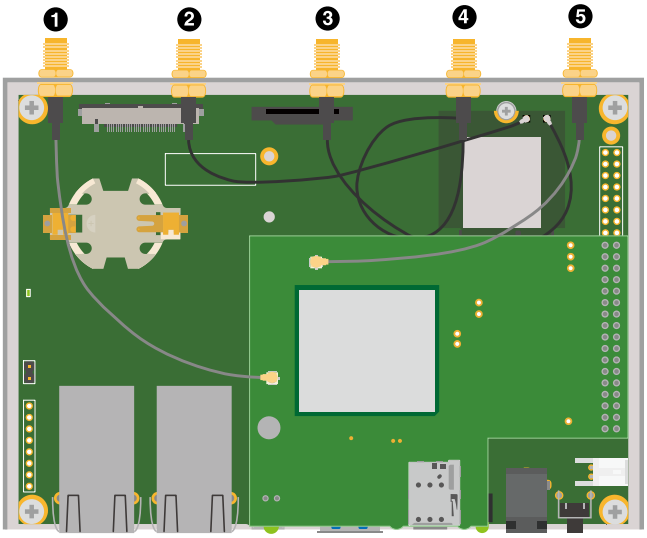


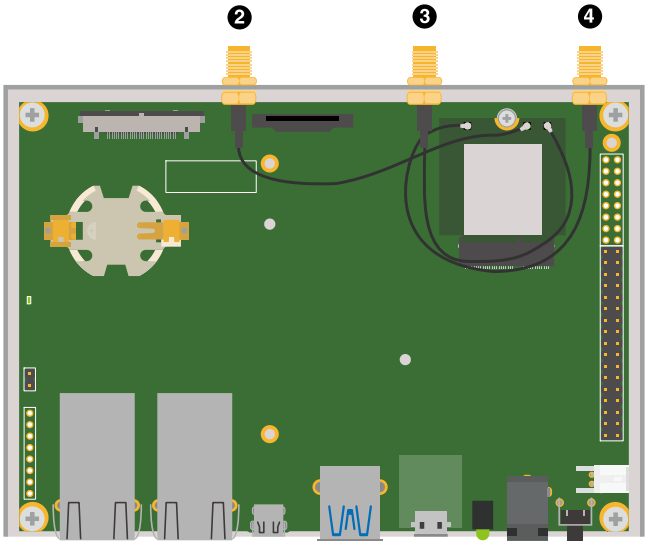
図 2.10 インターフェースレイアウト LTE モデル(ケース正面)

表 2.5 インターフェース内容

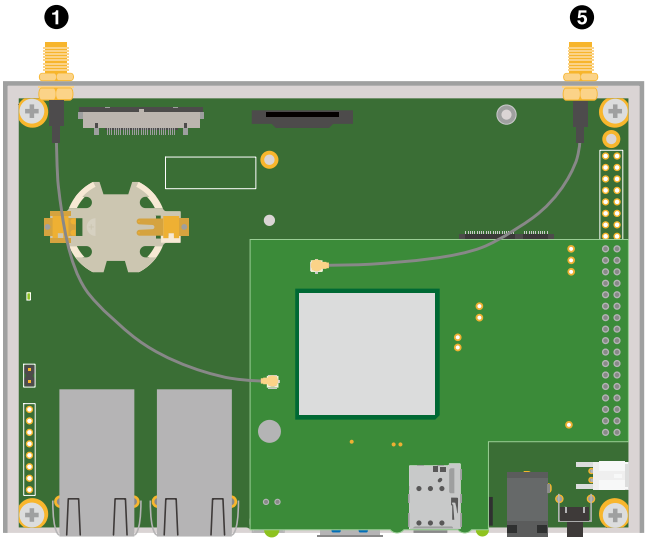
部品番号	インターフェース名	説明
CON21	3G/LTE アンテナインターフェース 1	3G/LTE 用外付けアンテナケーブルを接続するための小型同軸コネクタです。
CON22	3G/LTE アンテナインターフェース 2	3G/LTE 用外付けアンテナケーブルを接続するための小型同軸コネクタです。
CON23	nanoSIM インターフェース	nanoSIM カード用の nanoSIM スロットです。3G/LTE データ通信を利用する場合に使用します。
LED20	3G/LTE ステータス確認用 LED	3G/LTE 通信の状態を表示する砲弾タイプ(Φ3.4mm)の緑色 LED です。



LTE+WLAN/BT



WLAN/BT



LTE

図 2.11 インターフェースレイアウト アンテナ接続位置

❶ 3G/LTE 用外付けアンテナ接続、同軸コネクタ(SMA-J)、3G/LTE 用 Main アンテナ

❷ 無線 LAN/BT 用外付けアンテナ接続、同軸コネクタ(RP-SMA-J)、無線 LAN 用 Main アンテナ

❸ 無線 LAN/BT 用外付けアンテナ接続、同軸コネクタ(RP-SMA-J)、無線 LAN 用 Aux アンテナ

❹ 無線 LAN/BT 用外付けアンテナ接続、同軸コネクタ(RP-SMA-J)、BT 用アンテナ

❺ 3G/LTE 用外付けアンテナ接続、同軸コネクタ(SMA-J)、3G/LTE 用 Diversity アンテナ

2.5. ブロック図

Armadillo-IoT ゲートウェイ G4 のブロック図を「図 2.12. ブロック図」に示します。また、 LTE モデル追加分のブロック図を「図 2.13. ブロック図(LTE モデル追加分)」に示します。

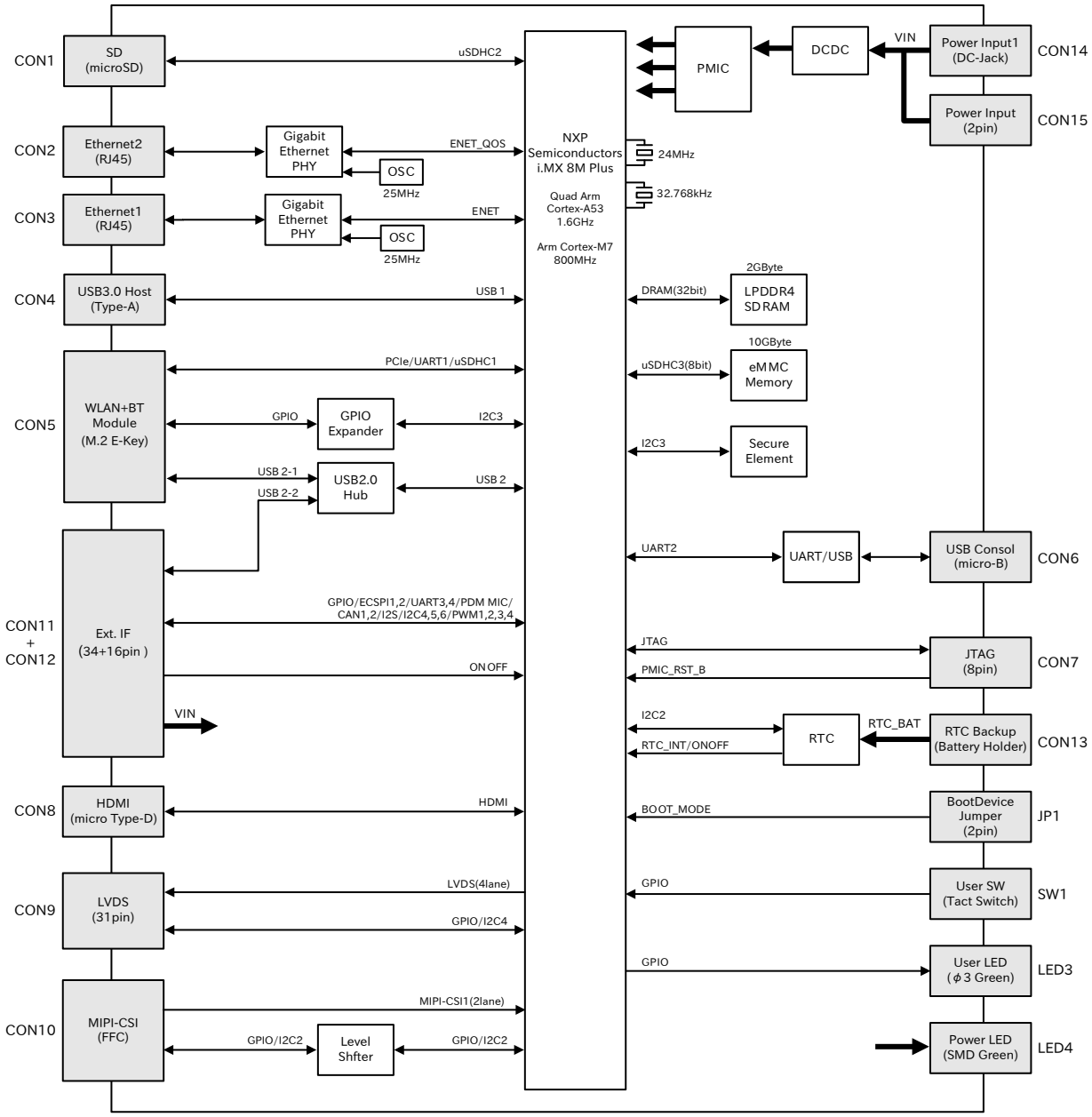

CON1
SD (microSD)
uSDHC2
CON2
Ethernet2 (RJ45)
Gigabit Ethernet PHY
OSC
25MHz
ENET_QOS
CON3
Ethernet1 (RJ45)
Gigabit Ethernet PHY
OSC
25MHz
ENET
CON4
USB3.0 Host (Type-A)
USB 1
CON5
WLAN+BT Module (M.2 E-Key)
PCIe/UART1/uSDHC1
GPIO
GPIO Expander
I2C3
USB 2-1
USB 2-2
USB2.0 Hub
USB 2
CON11 + CON12
Ext. IF (34+16pin)
GPIO/ECSPI1,2/UART3,4/PDM MIC/ CAN1,2/I2S/I2C4,5,6/PWM1,2,3,4
ON OFF
VIN
CON8
HDMI (micro Type-D)
HDMI
CON9
LVDS (31pin)
LVDS(4lane)
GPIO/I2C4
CON10
MIPI-CSI (FFC)
MIPI-CSI1(2lane)
GPIO/I2C2
Level Shfter
GPIO/I2C2
NXP Semiconductors i.MX 8M Plus
Quad Arm Cortex-A53 1.6GHz
Arm Cortex-M7 800MHz
24MHz
32.768kHz
PMIC
DCDC
VIN
Power Input1 (DC-Jack)
CON14
Power Input (2pin)
CON15
2GByte
DRAM(32bit)
LPDDR4 SDRAM
10GByte
uSDHC3(8bit)
eMMC Memory
I2C3
Secure Element
UART2
UART/USB
USB Consol (micro-B)
CON6
JTAG
PMIC_RST_B
JTAG (8pin)
CON7
I2C2
RTC_INT/ONOFF
RTC
RTC_BAT
RTC Backup (Battery Holder)
CON13
BOOT_MODE
BootDevice Jumper (2pin)
JP1
GPIO
User SW (Tact Switch)
SW1
GPIO
User LED (φ3 Green)
LED3
Power LED (SMD Green)
LED4

図 2.12 ブロック図

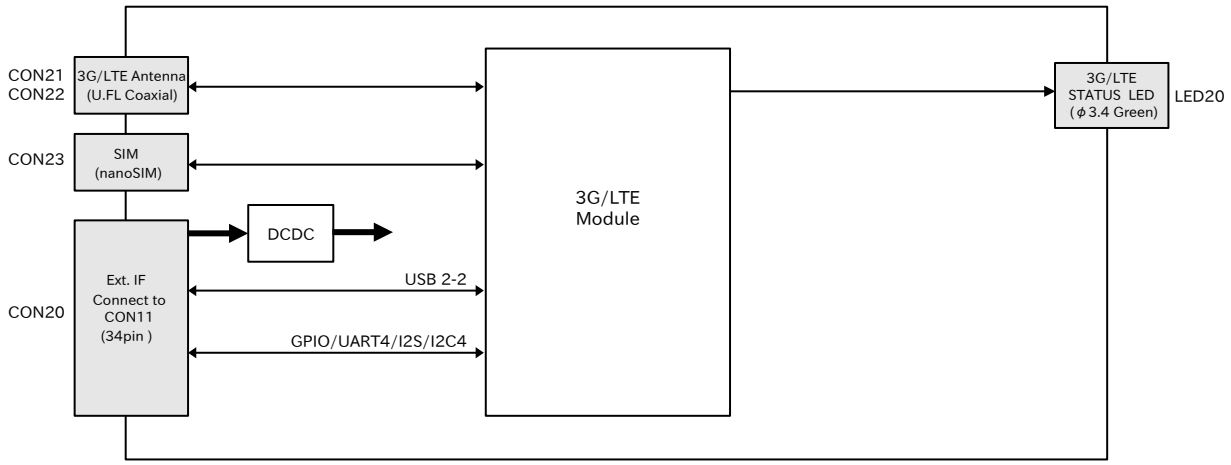


図 2.13 ブロック図(LTE モデル追加分)

2.6. 使用可能なストレージデバイス

Armadillo-IoT ゲートウェイ G4 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/mmcblk2	/dev/mmcblk2p1	オンボード
オンボード eMMC (GPP)	/dev/mmcblk2gp3	なし	オンボード
microSD/microSDHC/ microSDXC カード	/dev/mmcblk1	/dev/mmcblk1p1	SD インターフェース (CON1)
USB メモリ	/dev/sd* [a]	/dev/sd*1	USB ホストインターフェース (CON4)

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

GPP(General Purpose Partition)について

GPP は、eMMC の通常の記憶領域を割譲して eMMC 内部に作られた記憶領域です。 eMMC の通常の記憶領域とはアドレス空間が異なるため、/dev/mmcblk2 および /dev/mmcblk2p* に対してどのような書き込みを行っても /dev/mmcblk2gp* のデータが書き換わることはありません。

Armadillo-IoT ゲートウェイ G4 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/mmcblk2gp0	ライセンス情報等の為の予約領域
/dev/mmcblk2gp1	動作ログ領域
/dev/mmcblk2gp2	動作ログ予備領域[a]
/dev/mmcblk2gp3	ユーザー領域

[a]詳細は「6.28.4. ログ用パーティションについて」を参照ください。

2.7. ストレージデバイスのパーティション構成

Armadillo-IoT ゲートウェイ G4 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	8.95GiB	app	アプリケーション用パーティション

Armadillo-IoT ゲートウェイ G4 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/mmcblk2boot0	31.5 MiB	A/B アップデートの A 面
/dev/mmcblk2boot1	31.5 MiB	A/B アップデートの B 面

Armadillo-IoT ゲートウェイ G4 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/mmcblk2gp0	8 MiB	ライセンス情報等の為の予約領域
/dev/mmcblk2gp1	8 MiB	動作ログ領域
/dev/mmcblk2gp2	8 MiB	動作ログ予備領域[a]
/dev/mmcblk2gp3	8 MiB	ユーザー領域

[a]詳細は「6.28.4. ログ用パーティションについて」を参照ください。

2.8. ソフトウェアのライセンス

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

手順について、詳細は以下の Howto を参照してください。

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

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

3. 開発編

3.1. アプリケーション開発の流れ

Armadillo-IoT ゲートウェイ G4 では基本的に ATDE という Armadillo 専用開発環境と、 Visual Studio Code 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。

基本的な Armadillo-IoT ゲートウェイ G4 でのアプリケーション開発の流れを「図 3.1. アプリケーション開発の流れ」に示します。

本章では、「図 3.1. 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。

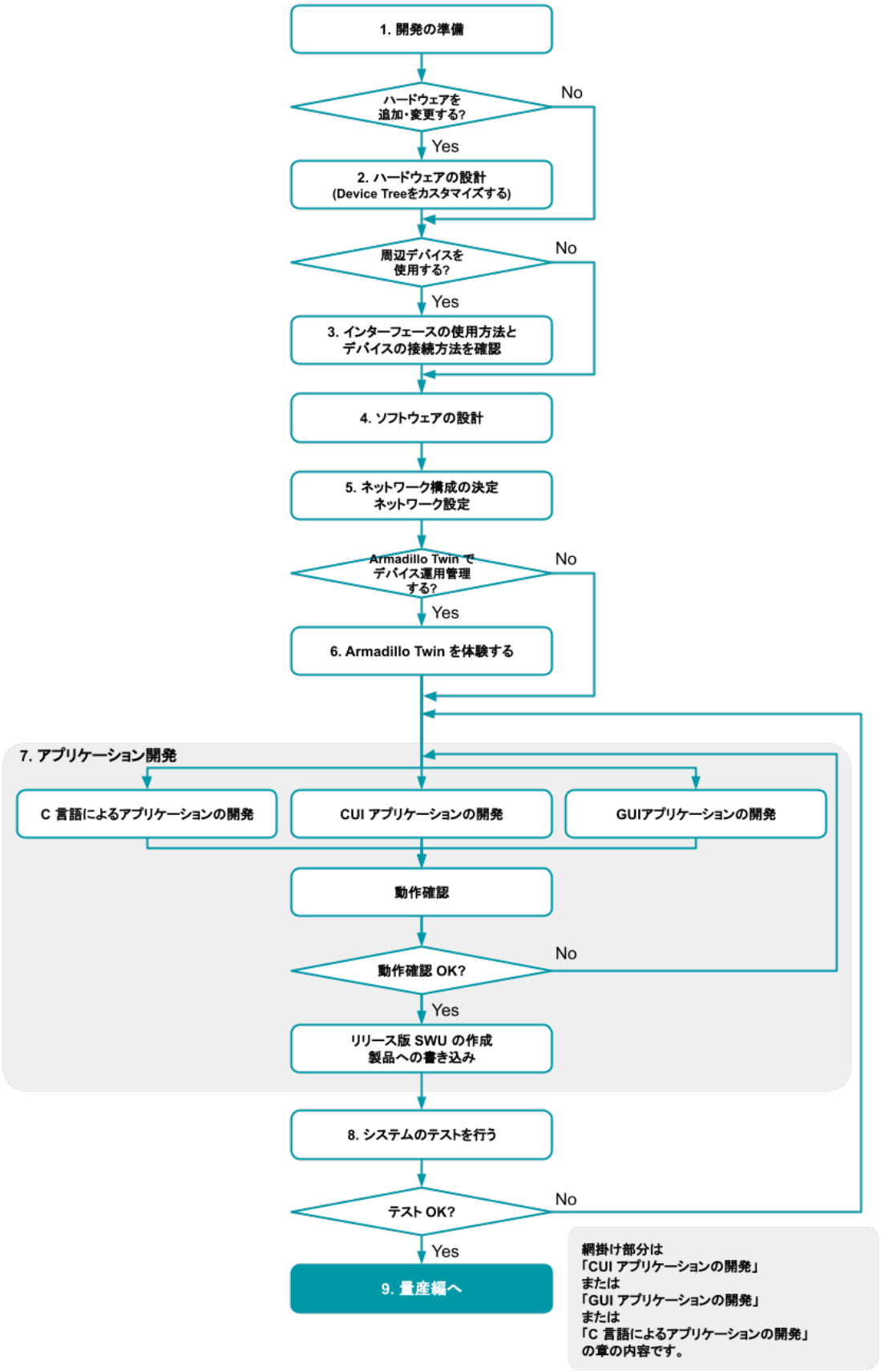


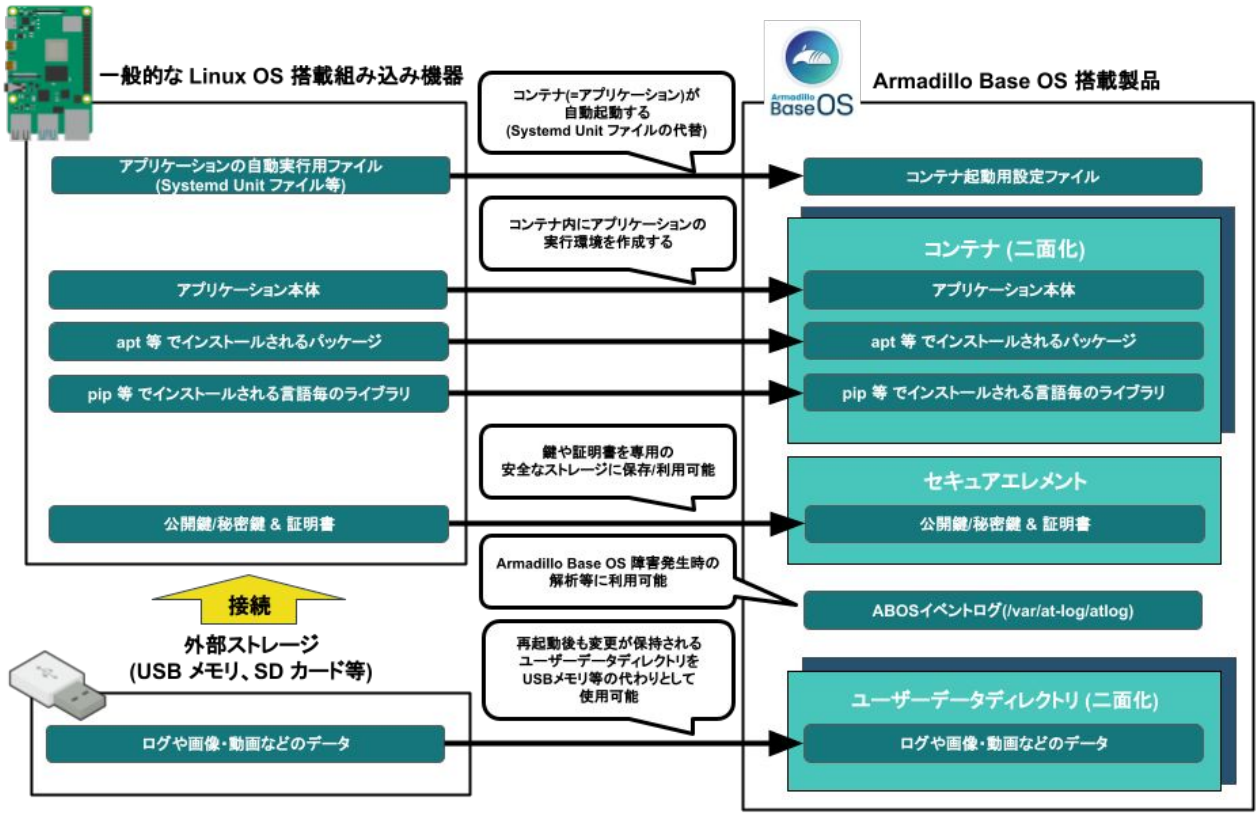
図 3.1 アプリケーション開発の流れ

1. 「3.3. 開発の準備」に従って開発環境の準備を行います。
2. 拡張基板を追加するなど、ハードウェアの追加・変更をする場合、「3.4. ハードウェアの設計」を行います。
    a. 拡張インターフェース(CON11、CON12)のピンを使用する場合「3.5. Device Tree をカスタマイズする」を参考にデバイスツリーのカスタマイズを行います。
3. Armadillo-IoT ゲートウェイ G4 に周辺デバイスを接続して使用する場合は、使用手順を「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」で確認します。
4. 「3.7. ソフトウェアの設計」を行います。
5. 「3.8. ネットワーク設定」を行います。
6. Armadillo Twin を使用したデバイス運用管理を検討する場合、「3.11. Armadillo Twin を体験する」を行います。
7. アプリケーションの開発を行います。「図 3.1. アプリケーション開発の流れ」 の網掛け部分です。
    a. 画面を使用するアプリケーションを開発する場合、「3.13. GUI アプリケーションの開発」を行います。
    b. 画面を必要としないアプリケーションを開発する場合、は、シェスクリプトまたは Python で開発することを推奨します。その場合は「3.14. CUI アプリケーションの開発」を行います。
    c. C 言語で開発された既存のアプリケーションを Armadillo 上で動作させる必要がある、あるいは開発環境の制約によって C 言語でのアプリケーション開発が必要な場合、「3.15. C 言語によるアプリケーションの開発」を行います。
8. 開発したアプリケーションの動作確認が完了しましたら、「3.16. システムのテストを行う」を行います。
9. システムのテストが完了しましたら、「4. 量産編」へ進みます。

3.2. 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。

3.2.1. 一般的な Linux OS 搭載組み込み機器との違い

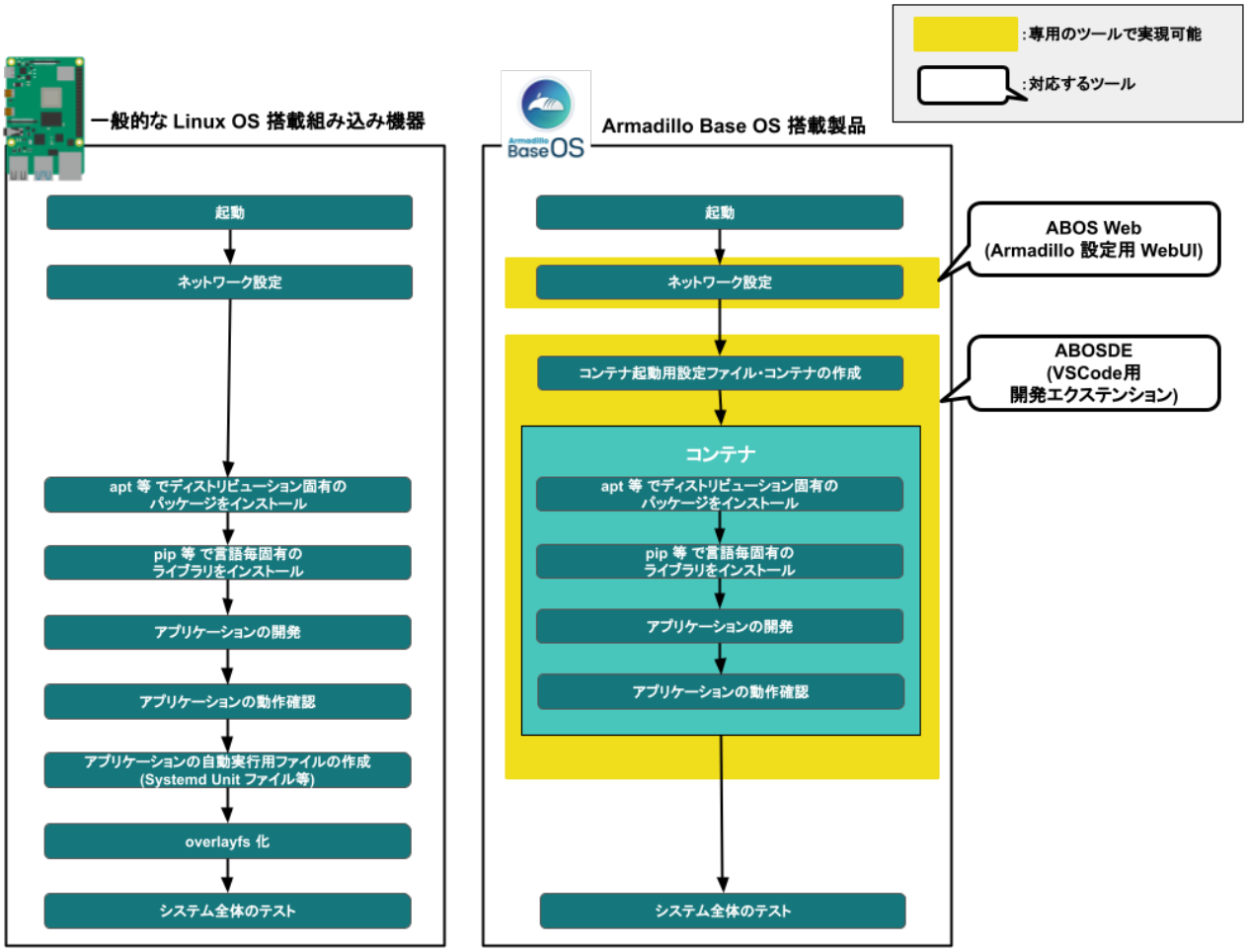


Linux OS 搭載組み込み機器ではアプリケーションの実行環境をユーザーランド上に直接用意し、Systemd などでアプリケーションを自動実行させるのが一般的です。Armadillo Base OS 搭載機器では、アプリケーションの実行環境をコンテナ内に用意して、コンテナ起動用設定ファイルを所定の場所に配置することでコンテナ(=アプリケーション)を自動実行させます。

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。

3.2.2. 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 VSCode 内で行うことができます。

3.2.3. アップデート機能について

Armadillo-IoT ゲートウェイ G4 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。

3.2.3.1. 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-IoT ゲートウェイ G4 では、 SWUpdate を利用することで次の機能を実現しています。

- A/B アップデート(アップデートの 2 面化)
- リカバリーモード
- ソフトウェアの圧縮、暗号化、署名付与
- 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
- hawkBit でのリモートアップデート対応
- ダウングレードの禁止

3.2.3.2. 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、 hawkBit という Web アプリケーションを使うなどです。

3.2.3.3. A/B アップデート(アップデートの 2 面化)

A/B アップデートは、Flash メモリにパーティションを 2 面確保し、アップデート時には交互に利用する仕組みです。

常に使用していない方のパーティションを書き換えるため次の特徴を持ちます。

- ○ アップデートによって動作中のソフトウェアは破壊されない
- ○ 書き込みが電源断などで中断しても、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

3.2.3.4. ロールバック（リカバリー）

システムが起動できなくなった際に、自動的にアップデート前のシステムにロールバックします。

ロールバック状態の確認は 「6.21. ロールバック状態を確認する」 を参照してください。

ロールバックする条件は次の通りです:

- rootfs にブートに必要なファイルが存在しない場合（/boot/Image, /boot/armadillo.dtb)
- 3 回起動を試して「bootcount」サービスが 1 度も起動できなかった場合は、次の起動時にロールバックします。

  bootcount 機能は uboot の「upgrade_available」変数で管理されています。bootcount 機能を利用しないようにするには、「6.23. u-boot の環境変数の設定」 を参照して変数を消します。
- ユーザーのスクリプトなどから、「abos-ctrl rollback」コマンドを実行した場合。

ロールバックが実行されると `/var/at-log/atlog` にログが残ります。

3.2.3.5. 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 SD 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
  - hawkBit を使用した自動インストール

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. 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-IoT ゲートウェイ G4 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。「6.8.4. 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

  VSCode 拡張機能の ABOSDE を使用することで、Armadillo-IoT ゲートウェイ G4 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。「6.9.5. 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 SD カードからの自動インストール

Armadillo-IoT ゲートウェイ G4 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-IoT ゲートウェイ G4 は自動で再起動します。

USB メモリや SD カードを vfat もしくは ext4 形式でフォーマットし、作成した.swu のファイルをディレクトリを作らずに配置してください。

ATDE 上で USB メモリ/microSD カードのパーティションを作成・フォーマットする方法

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

```
[ATDE ~/mkswu]$ df -h
Filesystem           Size  Used Avail Use% Mounted on
:（省略）
/dev/sda1        15G  5.6G  9.1G  39% /media/USBDRIVE ❶
[ATDE ~/mkswu]$ cp initial_setup.swu /media/USBDRIVE/ ❷
[ATDE ~/mkswu]$ umount /media/USBDRIVE ❸
```

❶ USB メモリがマウントされている場所を確認します。

❷ ファイルをコピーします。

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

エラーの場合、/var/log/message に保存されます。例えば、コンソールで証明の間違ったイメージのエラーを表示します：

```
[armadillo ~]# tail /var/log/messages
Nov 19 10:48:42 user.notice swupdate-auto-update: Mounting sda0 on /mnt
Nov 19 10:48:42 user.notice swupdate-auto-update: Trying update /mnt/initial_setup.swu
Nov 19 10:48:42 user.info swupdate: START Software Update started !
Nov 19 10:48:42 user.err swupdate: FAILURE ERROR : Signature verification failed ❶
Nov 19 10:48:42 user.err swupdate: FAILURE ERROR : Compatible SW not found
Nov 19 10:48:42 user.err swupdate: FATAL_FAILURE Image invalid or corrupted. Not installing ...
```

❶ 証明が間違ったメッセージ。

・外部記憶装置からイメージのインストール（手動）

USB メモリや microSD カード等の外部記憶装置のルートディレクトリ以外に swu イメージを保存して、イメージのインストールを行います。ルートディレクトリに保存すると自動アップデートが行われますので、/var/log/messages を確認してください。

以下は外部記憶装置が/dev/mmcblk1p1(microSD カード）として認識された場合に、イメージのインストールを行う例です。

```
[armadillo ~]# mount /dev/mmcblk1p1 /mnt
[armadillo ~]# swupdate -i /mnt/swu/initial_setup.swu
SWUpdate v5f2d8be-dirty

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

[INFO ] : SWUPDATE running :  [main] : Running on AGX4500 Revision at1
[INFO ] : SWUPDATE started :  Software Update started !
[INFO ] : SWUPDATE running :  [read_lines_notify] : No base os update: copying current os over
[INFO ] : SWUPDATE running :  [read_lines_notify] : Removing unused containers
[INFO ] : SWUPDATE running :  [read_lines_notify] : swupdate triggering reboot!
Killed
```

・Armadillo Twin を使用した自動インストール

Armadillo Twin で Armadillo-IoT ゲートウェイ G4 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/{url-product-dir}/image/baseos-x2-latest.swu ¥
                        > /etc/swupdate.watch ❸
[armadillo ~]# echo 'schedule="0 tomorrow"' > /etc/conf.d/swupdate-url
[armadillo ~]# echo 'rdelay="21600"' >> /etc/conf.d/swupdate-url ❹
[armadillo ~]# persist_file /etc/swupdate.watch /etc/conf.d/swupdate-url ❺
```

❶ swupdate-url サービスを有効します。

❷ サービスの有効化を保存します。

❸ イメージの URL を登録します。一行ごとにイメージの URL を設定することができ、複数行にイメージの URL を設定することができます。

❹ チェックやインストールのスケジュールを設定します。

❺ 変更した設定ファイルを保存します。

USB メモリからのアップデートと同様に、ログは/var/log/messages に保存されます。

initial_setup のイメージを作成の際に /usr/share/mkswu/examples/enable_swupdate_url.desc を入れると有効にすることができます。

・ hawkBit を使用した自動インストール

hawkBit で Armadillo-IoT ゲートウェイ G4 を複数台管理してアップデートすることができます。「5.6.3. hawkBit サーバーから複数の Armadillo に配信する」を参考にしてください。

3.2.4. 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 3.2 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。

3.2.4.1. 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)

「3.2.4. 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 3.1. 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 3.1 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)

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の保存に向いています。
/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の保存に向いています。

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

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

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

Copy-on-Write (CoW) について。

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

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

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

図 3.3 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。

3.2.5. 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-IoT ゲートウェイ G4 インストールディスクイメージ [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/disc-image] にある標準イメージです。Armadillo を初期化する際に使用されます。

- 開発が完了した Armadillo-IoT ゲートウェイ G4 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。

3.2.5.1. 初期化インストールディスクの作成

1. 512 MB 以上の microSD カードを用意してください。
2. 標準のインストールディスクイメージを使用する場合は、Armadillo-IoT ゲートウェイ G4 インストールディスクイメージ [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/disc-image] から「Armadillo Base OS」をダウンロードしてください。

   「6.25. Armadillo のソフトウェアをビルドする」 でビルドしたイメージを使用してインストールディスクを作成したい場合は、以下のコマンドを実行して、インストールディスクイメージを作成してください。

   ```
   [ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_image.sh ¥
           --firmware ~/at-imxlibpackage/imx_lib.img
   :（省略）
   [ATDE ~/build-rootfs-[VERSION]]$ ls baseos-x2*img
   baseos-x2-[VERSION].img
   [ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_image.sh ¥
           --boot ~/imx-boot-[VERSION]/imx-boot_armadillo_x2 ¥
           --installer ./baseos-x2-[VERSION].img
   ```

   コマンドの実行が完了すると、baseos-x2-[VERSION]-installer.img というファイルが作成されていますので、こちらを使用してください。

3. ATDE に microSD カードを接続します。詳しくは「3.3.2.7. 取り外し可能デバイスの使用」を参考にしてください。
4. microSD カードのデバイス名を確認します

   ```
   [ATDE ~]$ ls /dev/sd?
   /dev/sda  /dev/sdb
   [ATDE ~]$ sudo fdisk -l /dev/sdb
   Disk /dev/sdb: 7.22 GiB, 7751073792 bytes, 15138816 sectors
   Disk model: SD/MMC
   :（省略）
   ```

5. 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
```

6. ダウンロードしたファイルを展開し、img ファイルを microSD カードに書き込んでください。

   Linux PC の場合、以下のように microSD カードに書き込むことができます。

```
[ATDE ~]$ unzip baseos-x2-installer-[VERSION].zip
[ATDE ~]$ sudo dd if=baseos-x2-installer-[VERSION].img ¥
                 of=/dev/sdb bs=1M oflag=direct status=progress
```

   また、Windows の場合、エクスプローラー等で Zip ファイルから img ファイルを取り出し、「Win32 Disk Imager」などを使用して microSD カードに書き込むことができます。

インストールディスク作成時に SBOM を作成する場合は build_image.sh の引数に --sbom を渡してください。ライセンス情報等を記載するためのコンフィグファイルはデフォルトは baseos_sbom.yaml となっています。コンフィグファイルを変更する場合は --sbom-config <config> に引数を入れてください。また、コンテナイメージを含める場合等に外部の SBOM を入れる必要がある場合は --sbom-external <sbom> に引数を入れてください。SBOM のライセンス情報やコンフィグファイルの設定方法については 「6.25.4. ビルドしたルートファイルシステムの SBOM を作成する」 をご覧ください。

3.2.5.2. インストールディスクを使用する

1. JP1 ジャンパーをショート(SD ブートに設定)し、microSD カードを CON1 に挿入します。
2. 電源を投入すると、1 分程度で eMMC のソフトウェアの初期化が完了します。
3. 完了すると電源が切れます(LED4 が消灯、コンソールに reboot: Power down が表示)。
4. 電源を取り外し、続いて JP1 ジャンパーと microSD カードを外してください。
5. 10 秒以上待ってから再び電源を入れると、初回起動時と同じ状態になります。

3.3. 開発の準備

3.3.1. 準備するもの

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

作業用 PC	Linux または Windows が動作し、ネットワークインターフェースと 1 つ以上の USB ポートを持つ PC です。「開発/動作確認環境の

	構築」を参照して、作業用 PC 上に開発/動作確認環境を構築してください。
ネットワーク環境	Armadillo と作業用 PC をネットワーク通信ができるようにしてください。
microSD カード	microSD スロットの動作を確認する場合などに利用します。
USB メモリ	USB の動作を確認する場合などに利用します。
tar.xz 形式のファイルを展開するソフトウェア	開発/動作確認環境を構築するために利用します。Linux では、tar で展開できます。Windows では、7-Zip や Lhaz などが対応しています。
nanoSIM(UIM カード)と APN 情報	LTE モデルで 3G/LTE の動作を確認する場合に利用します。通信事業者との契約が必要です。SMS の動作を確認する場合は、SMS が利用可能な nanoSIM(UIM カード)が必要です。

3.3.2. 開発環境のセットアップ

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

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

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

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

ATDE は、バージョンにより対応するアットマークテクノ製品が異なります。本製品に対応している ATDE は、ATDE9 の v20211201 以降です。

ATDE9 は Debian GNU/Linux 11 (コードネーム bullseye) をベースに、Armadillo-IoT ゲートウェイ G4 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-IoT ゲートウェイ G4 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。

3.3.2.1. VMware のインストール

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

VMware は、非商用利用限定で無償のものから、商用利用可能な有償のものまで複数の製品があります。製品ごとに異なるライセンス、エンドユー

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

ザー使用許諾契約書(EULA)が存在するため、十分に確認した上で利用目的に合う製品をご利用ください。

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

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

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

3.3.2.2. ATDE のアーカイブを取得

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

本製品に対応している ATDE のバージョンは ATDE9 v20211201 以降です。

作業用 PC の動作環境(ハードウェア、VMware、ATDE の対応アーキテクチャなど)により、ATDE が正常に動作しない可能性があります。VMware 社 Web ページ(http://www.vmware.com/)から、使用している VMware のドキュメントなどを参照して動作環境を確認してください。

3.3.2.3. ATDE のアーカイブを展開

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

Windows での展開方法を「3.3.2.4. Windows で ATDE のアーカイブ展開する」に、Linux での展開方法を手順「3.3.2.5. Linux で tar.xz 形式のファイルを展開する」に示します。

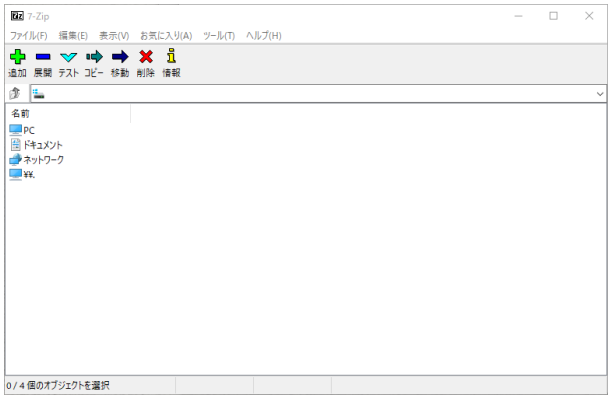
3.3.2.4. Windows で ATDE のアーカイブ展開する

1. 7-Zip のインストール

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

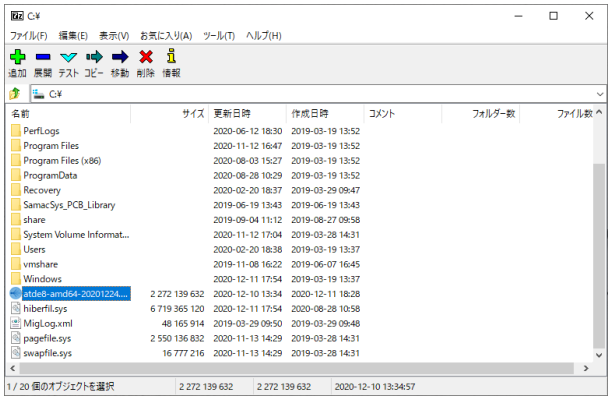
2. 7-Zip の起動

   7-Zip を起動します。



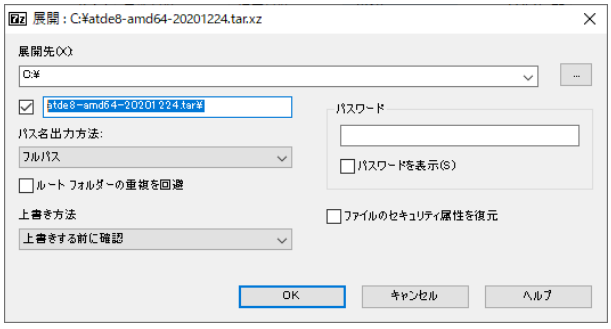
3. xz 圧縮ファイルの選択

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



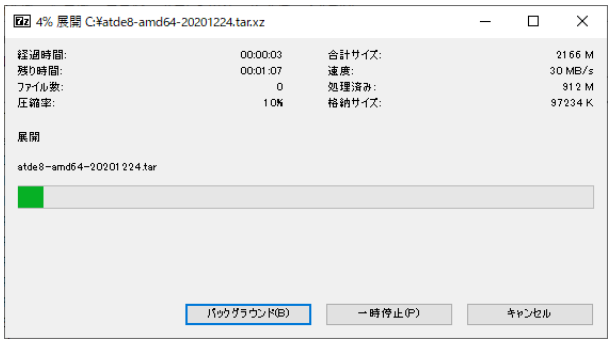
4. xz 圧縮ファイルの展開先の指定

   「展開先」を指定して、「OK」をクリックします。



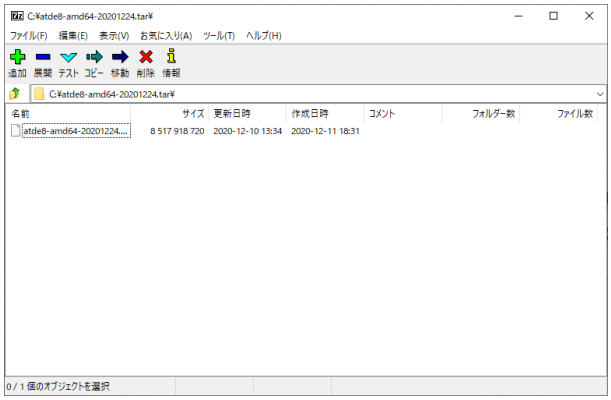
5. xz 圧縮ファイルの展開

   展開が始まります。



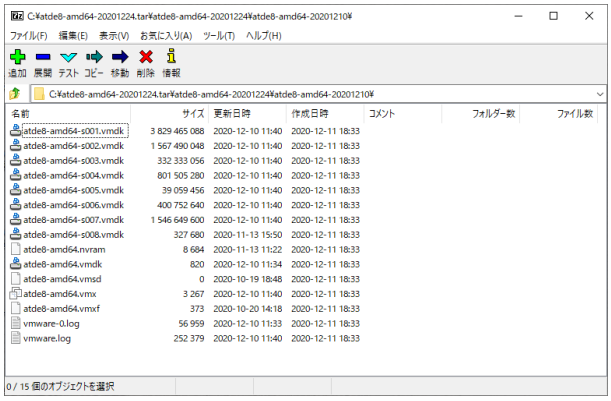
6. tar アーカイブファイルの選択

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



7. 展開の完了確認

   tar アーカイブファイルの展開が終了すると、ATDE アーカイブの展開は完了です。「展開先」に指定したフォルダに ATDE のデータイメージが出力されています。



3.3.2.5. Linux で tar.xz 形式のファイルを展開する

1. tar.xz 圧縮ファイルの展開

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

```
[PC ~]$ tar xf atde9-amd64-[VERSION].tar.xz
```

2. 展開の完了確認

tar.xz 圧縮ファイルの展開が終了すると、ATDE アーカイブの展開は完了です。 **atde9-amd64-[VERSION]** ディレクトリに ATDE のデータイメージが出力されています。

```
[PC ~]$ ls atde9-amd64-[VERSION]/
atde9-amd64-s001.vmdk  atde9-amd64-s008.vmdk
atde9-amd64-s002.vmdk  atde9-amd64-s009.vmdk
atde9-amd64-s003.vmdk  atde9-amd64.nvram
atde9-amd64-s004.vmdk  atde9-amd64.vmdk
atde9-amd64-s005.vmdk  atde9-amd64.vmsd
atde9-amd64-s006.vmdk  atde9-amd64.vmx
atde9-amd64-s007.vmdk  atde9-amd64.vmxf
```

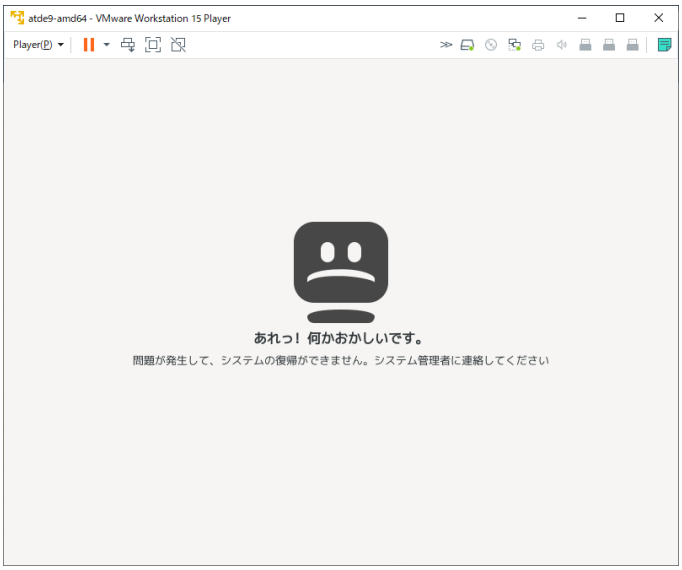
3.3.2.6. ATDE の起動

ATDE のアーカイブを展開したディレクトリに存在する仮想マシン構成(.vmx)ファイルを VMware 上で開くと、ATDE を起動することができます。ATDE9 にログイン可能なユーザーを、「表 3.2. ユーザー名とパスワード」に示します [2]。

表 3.2 ユーザー名とパスワード

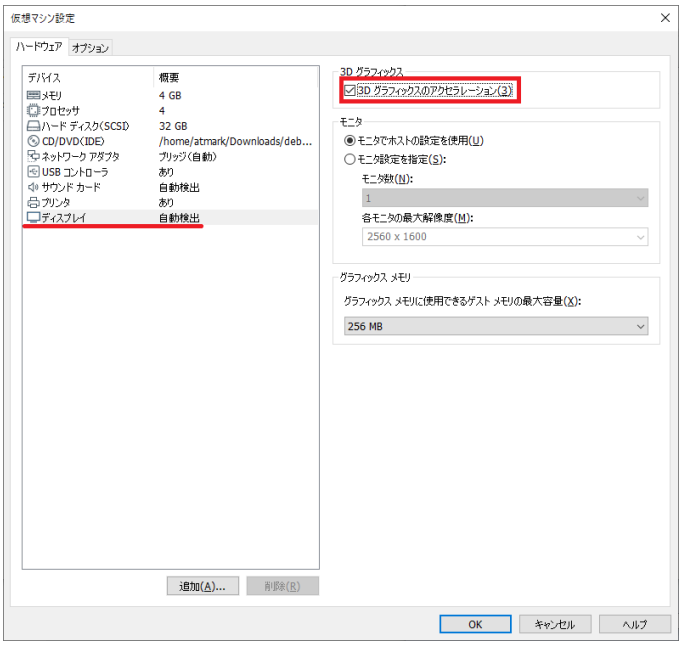
ユーザー名	パスワード	権限
atmark	atmark	一般ユーザー
root	root	特権ユーザー

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



[2]特権ユーザーで GUI ログインを行うことはできません

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



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

3.3.2.7. 取り外し可能デバイスの使用

VMware は、ゲスト OS (ATDE)による取り外し可能デバイス(USB デバイスや DVD など)の使用をサポートしています。デバイスによっては、ホスト OS (VMware を起動している OS)とゲスト OS で同時

に使用することができません。そのようなデバイスをゲスト OS で使用するためには、ゲスト OS にデバイスを接続する操作が必要になります。

取り外し可能デバイスの使用方法については、VMware 社 Web ページ(http://www.vmware.com/)から、使用している VMware のドキュメントなどを参照してください。

Armadillo-IoT ゲートウェイ G4 の動作確認を行うためには、「表 3.3. 動作確認に使用する取り外し可能デバイス」に示すデバイスをゲスト OS に接続する必要があります。

表 3.3 動作確認に使用する取り外し可能デバイス

デバイス	デバイス名
USB シリアル変換 IC	Silicon CP2102N USB to UART Bridge Controller

3.3.2.8. コマンドライン端末(GNOME 端末)の起動

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

GNOME 端末を起動するには、「図 3.4. GNOME 端末の起動」のようにデスクトップ左上のアプリケーションの「ユーティリティ」カテゴリから「端末」を選択してください。

図 3.4 GNOME 端末の起動

「図 3.5. GNOME 端末のウィンドウ」のようにウィンドウが開きます。

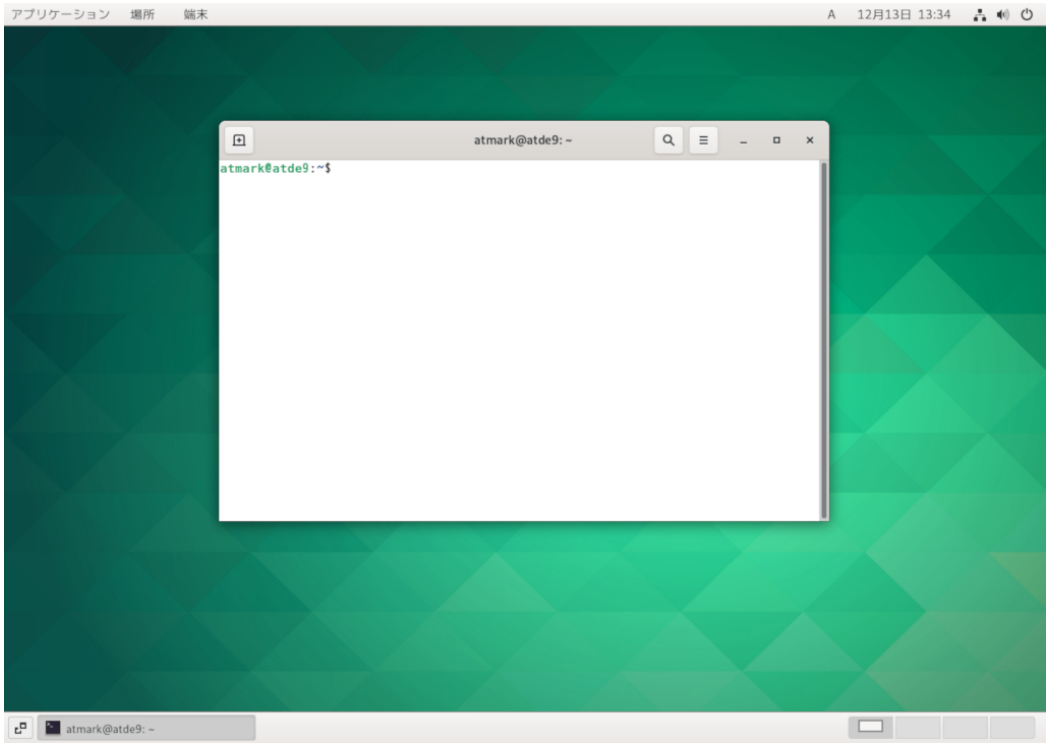


図 3.5 GNOME 端末のウィンドウ

3.3.2.9. シリアル通信ソフトウェア(minicom)の使用

シリアル通信ソフトウェア(minicom)のシリアル通信設定を、「表 3.4. シリアル通信設定」のように設定します。また、minicom を起動する端末の横幅を 80 文字以上にしてください。横幅が 80 文字より小さい場合、コマンド入力中に表示が乱れることがあります。

表 3.4 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 3.6. 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 3.6 minicom の設定の起動

2. 「図 3.7. 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 3.7 minicom の設定

3. 「図 3.8. 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 3.8 minicom のシリアルポートの設定

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

シリアル通信用 USB ケーブル(A-microB)使用時のデバイスファイル確認方法

Linux でシリアル通信用 USB ケーブル(A-microB)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 2-2.1: new full-speed USB device number 4 using uhci_hcd
usb 2-2.1: New USB device found, idVendor=10c4, idProduct=ea60,
bcdDevice= 1.00
usb 2-2.1: New USB device strings: Mfr=1, Product=2,
SerialNumber=3
usb 2-2.1: Product: CP2102N USB to UART Bridge Controller
usb 2-2.1: Manufacturer: Silicon Labs
usb 2-2.1: SerialNumber: 6a9681f80272eb11abb4496e014bf449
usbcore: registered new interface driver usbserial_generic
usbserial: USB Serial support registered for generic
usbcore: registered new interface driver cp210x
```

```
usbserial: USB Serial support registered for cp210x
usb 2-2.1: cp210x converter now attached to ttyUSB0
```

図 3.9 例. シリアル通信用 USB ケーブル(A-microB)接続時のログ

上記のログからシリアル通信用 USB ケーブル(A-microB)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 3.10. 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 3.10 minicom のシリアルポートのパラメータの設定

8. 「図 3.10. 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 3.11. minicom シリアルポートの設定値」を参照してください。

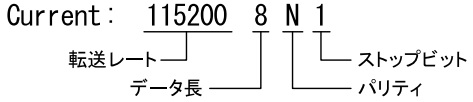


図 3.11 minicom シリアルポートの設定値

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 3.7. minicom の設定」に戻ってください。

13.「図 3.7. minicom の設定」から、「Save setup as dfl」を選択し、設定を保存してください。

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

minicom を起動させるには、「図 3.12. minicom 起動方法」のようにしてください。

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

図 3.12 minicom 起動方法

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

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

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

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

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

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

図 3.13 minicom 終了確認

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

3.3.3. Armadillo の起動

3.3.3.1. Armadillo と開発用 PC を接続

Armadillo-IoT ゲートウェイ G4 と周辺装置の接続例を「図 3.14. Armadillo-IoT ゲートウェイ G4 の接続例」に示します。3G/LTE 用アンテナおよび nanoSIM カード接続例については、「図 3.17. 3G/LTE 用アンテナおよび nanoSIM カード接続例」を参照してください。

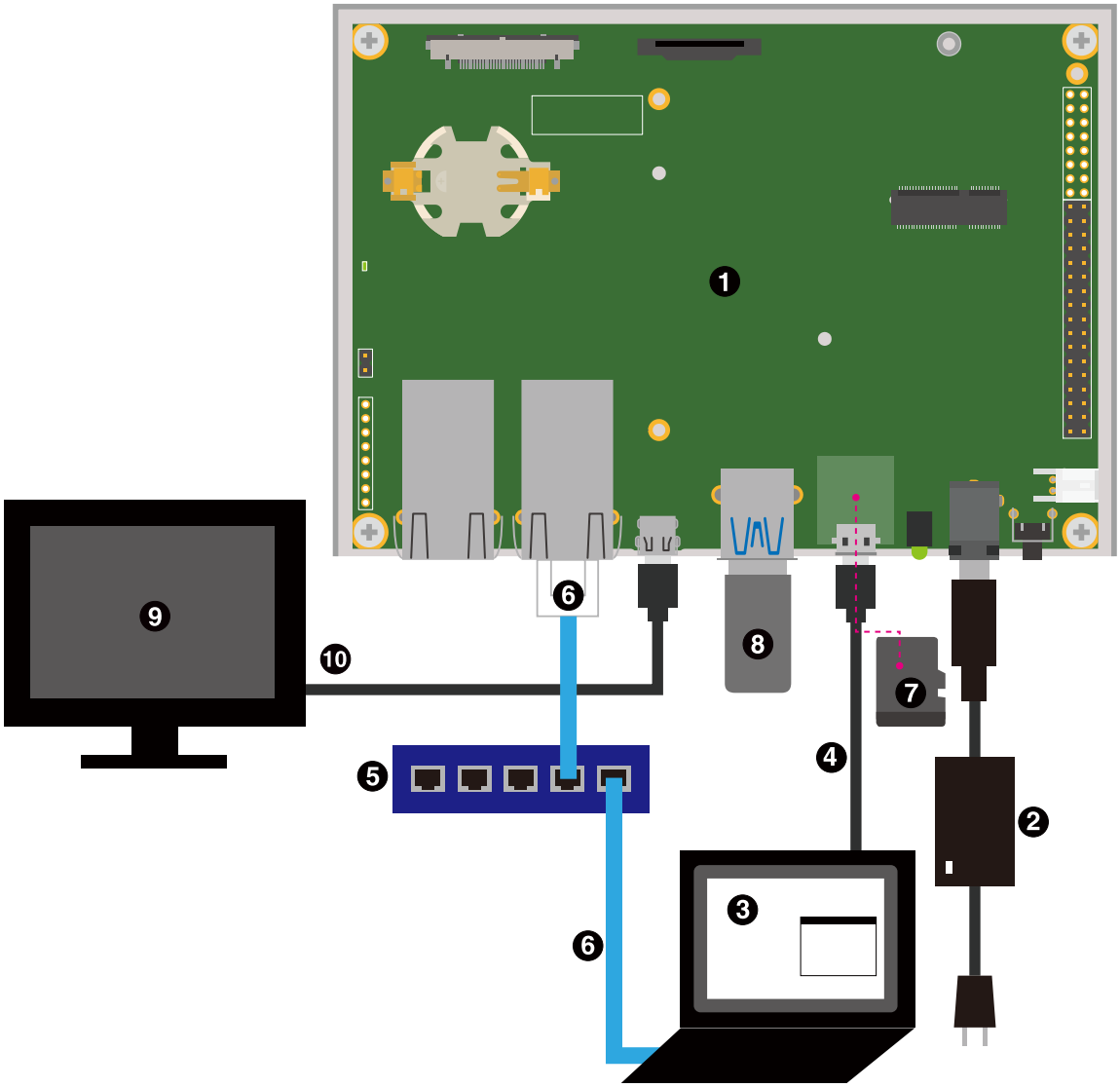


図 3.14 Armadillo-IoT ゲートウェイ G4 の接続例

1. Armadillo-IoT ゲートウェイ G4
2. AC アダプタ(12V/3.0A)
3. 作業用 PC
4. シリアル通信用 USB ケーブル(A-microB)

❺ LAN HUB

❻ Ethernet ケーブル

❼ microSD カード

❽ USB メモリ

❾ ディスプレイ(HDMI 対応)

❿ HDMI ケーブル

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

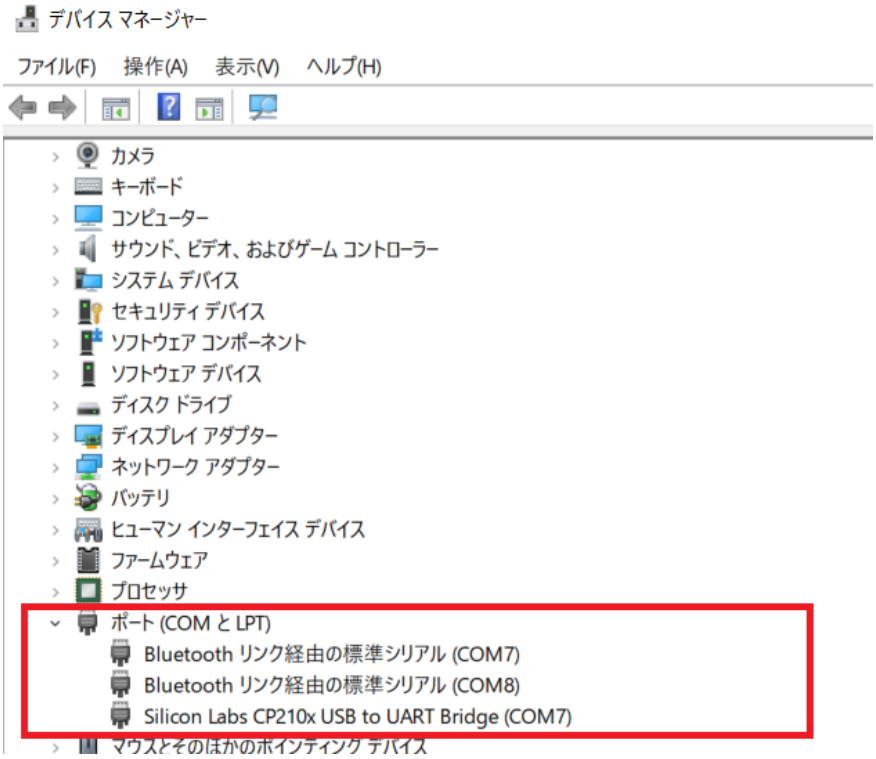


図 3.15 COM7 が競合している状態

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

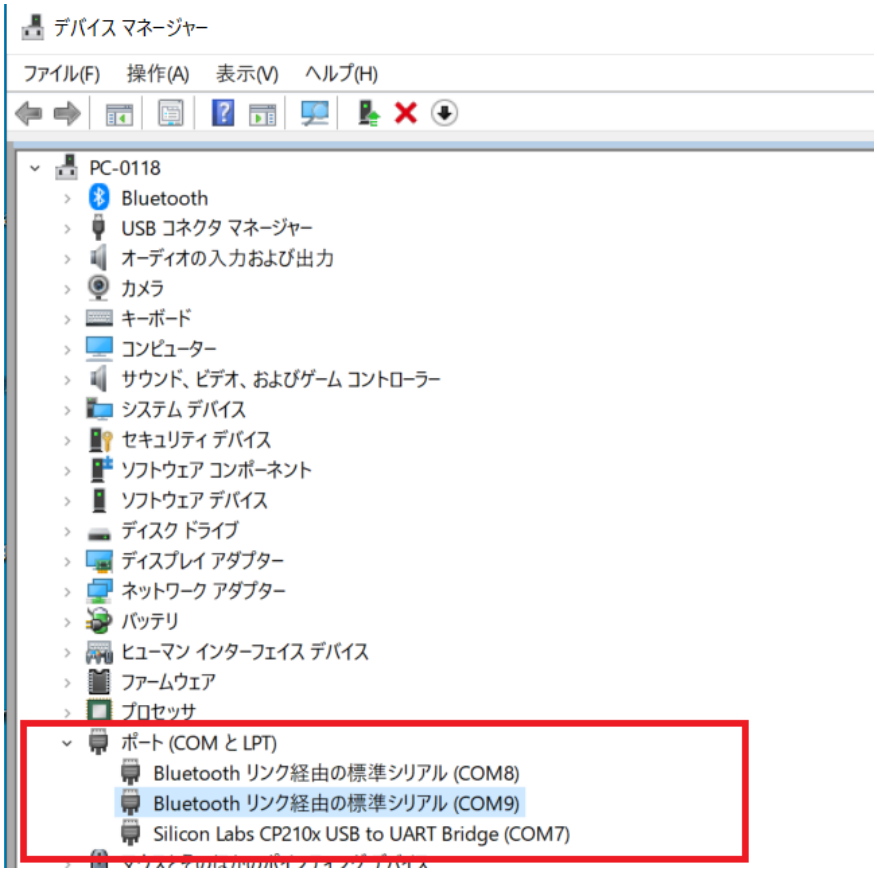


図 3.16 Bletooth に割当の COM を変更した状態

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。

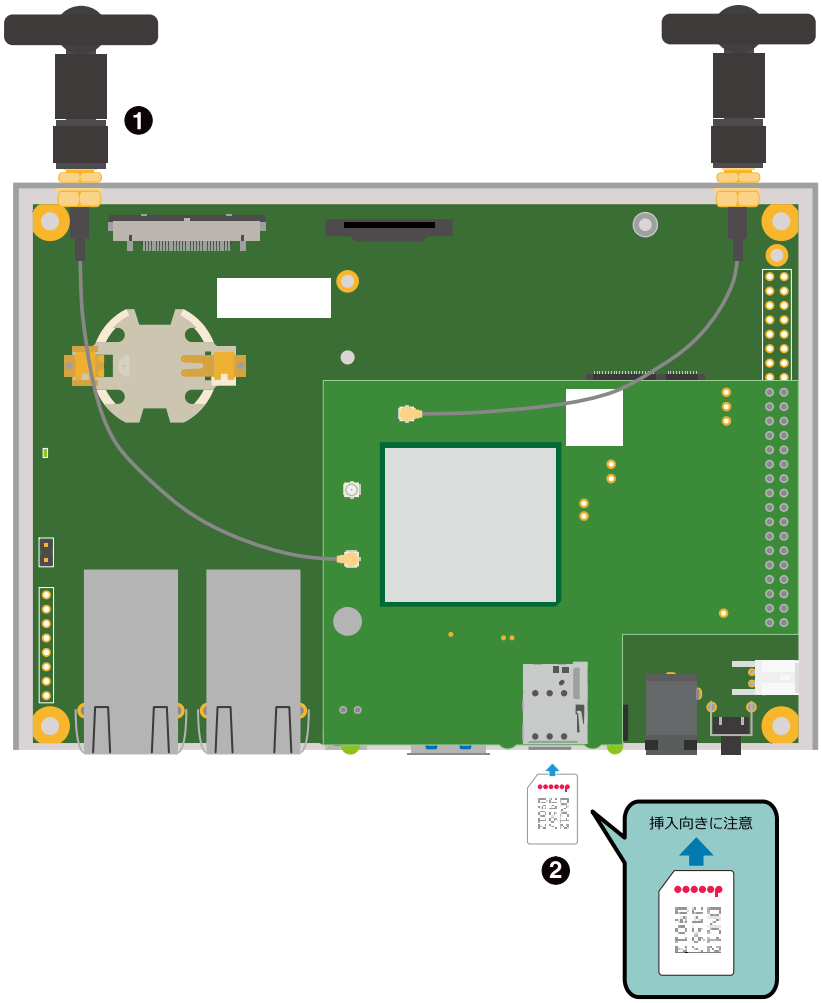


図 3.17 3G/LTE 用アンテナおよび nanoSIM カード接続例

❶ 3G/LTE 用アンテナ

❷ nanoSIM カード ※nanoSIM カードはケースを開けた状態でのみ挿抜可能です。

3.3.3.2. ジャンパピンの設定について

ジャンパの設定を変更することで、Armadillo-IoT ゲートウェイ G4 の動作を変更することができます。

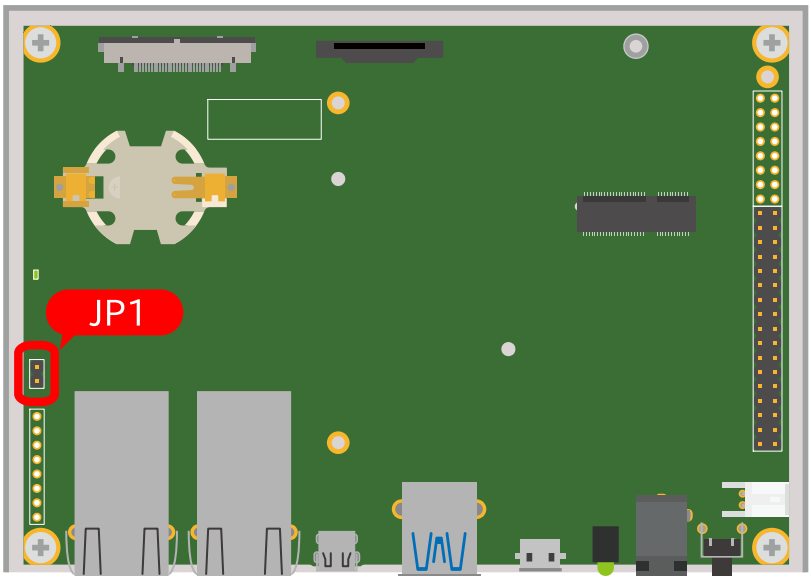


図 3.18 JP1 の位置

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

ジャンパのオープン、ショートとは

「オープン」とはジャンパピンにジャンパソケットを接続していない状態です。

「ショート」とはジャンパピンにジャンパソケットを接続している状態です。

3.3.3.3. 起動

電源入力インターフェースに電源を接続すると Armadillo-IoT ゲートウェイ G4 が起動します。起動すると CON6 (USB コンソールインターフェース) から起動ログが表示されます。

Armadillo-IoT ゲートウェイ G4 の電源投入時点でのジャンパ JP1 の状態によって起動モードが変化します。 詳しくは 「3.6.22. 起動デバイスを変更する」 を参照してください。

以下に起動ログの例を示します。

```
U-Boot SPL 2020.04-at7 (May 21 2022 - 11:21:55 +0900)
DDRINFO: start DRAM init
DDRINFO: DRAM rate 4000MTS
DDRINFO:ddrphy calibration done
```

```
DDRINFO: ddrmix config done
Normal Boot
Trying to boot from BOOTROM
image offset 0x0, pagesize 0x200, ivt offset 0x0
NOTICE:  BL31: v2.4(release):lf-5.10.y-1.0.0-0-gba76d337e956
NOTICE:  BL31: Built : 11:08:22, Apr  6 2022


U-Boot 2020.04-at7 (May 21 2022 - 11:21:55 +0900)

CPU:   i.MX8MP[8] rev1.1 1600 MHz (running at 1200 MHz)
CPU:   Industrial temperature grade (-40C to 105C) at 26C
Model: Atmark-Techno Armadillo X2 Series
DRAM:    Hold key pressed for tests: t (fast) / T (slow)
2 GiB
WDT:   Started with servicing (10s timeout)
MMC:   FSL_SDHC: 1, FSL_SDHC: 2
Loading Environment from MMC... OK
In:    serial
Out:   serial
Err:   serial

 BuildInfo:
  - ATF ba76d33
  - U-Boot 2020.04-at7

first boot since power on
switch to partitions #0, OK
mmc2(part 0) is current device
flash target is MMC:2
Net:   eth0: ethernet@30be0000 [PRIME], eth1: ethernet@30bf0000
Fastboot: Normal
Saving Environment to MMC... Writing to redundant MMC(2)... OK
Normal Boot
Hit any key to stop autoboot:  0
switch to partitions #0, OK
mmc2(part 0) is current device
31051784 bytes read in 675 ms (43.9 MiB/s)
Booting from mmc ...

## Checking Image at 40480000 ...
Unknown image format!
60012 bytes read in 5 ms (11.4 MiB/s)
## Flattened Device Tree blob at 45000000
   Booting using the fdt blob at 0x45000000
   Using Device Tree in place at 0000000045000000, end 0000000045011a6b

Starting kernel ...

[    1.606026] imx6q-pcie 33800000.pcie: invalid resource
[    2.618095] imx6q-pcie 33800000.pcie: failed to initialize host
[    2.624095] imx6q-pcie 33800000.pcie: unable to add pcie port.

   OpenRC 0.44.7.10dab8bfb7 is starting up Linux 5.10.117-r0 (aarch64)

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

```
 * /run/lock: correcting owner
 * Caching service dependencies ... [ ok ]
 * Starting rngd ... * Mounting /sys ... * Remounting devtmpfs on /dev ... [ ok ]
 [ ok ]
 * Mounting security filesystem ... [ ok ]
 * Mounting config filesystem ... [ ok ]
 [ ok ]
 * Mounting fuse control filesystem ... * Mounting /dev/mqueue ... [ ok ]
 [ ok ]
 * Mounting /dev/pts ... [ ok ]
 * Mounting /dev/shm ... [ ok ]
fsck_atlog       | * Checking at-log filesystem /dev/mmcblk2gp1 ...udev             | * Starting
udev ... [ ok ]
 [ ok ]
cryptsetup       | * Decrypting encrypted secondary partitions ...overlayfs        | * Preparing
overlayfs over / ... [ ok ]
 [ ok ]
udev-trigger     | * Generating a rule to create a /dev/root symlink ...hostname          | *
Setting hostname ... [ ok ]
 [ ok ]
udev-trigger     | * Populating /dev with existing devices through uevents ... [ ok ]
fsck             | * Checking local filesystems  ... [ ok ]
root             | * Remounting filesystems ... [ ok ]
localmount       | * Mounting local filesystems ... [ ok ]
urandom          | * Saving 4096 bits of creditable seed for next boot
bootmisc         | * Migrating /var/lock to /run/lock ... [ ok ]
bootmisc         | * Creating user login records ... [ ok ]
bootmisc         | * Wiping /var/tmp directory ... [ ok ]
dbus             | * /run/dbus: creating directory
dbus             | * /run/dbus: correcting owner
micron-emmc-reten | * Starting micron-emmc-reten
dbus            | * Starting System Message Bus ...syslog          | * Starting busybox syslog ...
[ ok ]
 [ ok ]
networkmanager   | * Starting networkmanager ... [ ok ]
buttond          | * Starting button watching daemon ...podman-atmark     | * Starting configured
podman containers ... [ ok ]
 [ ok ]
reset_bootcount  | * Resetting bootcount in bootloader env ...Environment OK, copy 1
chronyd          | * Starting chronyd ...reset_bootcount   | [ ok ]
sshd             | * Starting sshd ... [ ok ]
 [ ok ]
local            | * Starting local ... [ ok ]

Welcome to Alpine Linux 3.15
Kernel 5.10.117-r0 on an aarch64 (/dev/ttymxc1)

armadillo login:
```

U-Boot プロンプト

USB コンソールインターフェース に"Hit any key to stop autoboot:" が出力されている間に何かしらのキー入力を行うと U-Boot のプロンプトが表示されます。この間にキー入力がなければ自動的に起動します。

```
: (省略)
 BuildInfo:
```

```
  - ATF ba76d33
  - U-Boot 2020.04-at7

first boot since power on
switch to partitions #0, OK
mmc2(part 0) is current device
flash target is MMC:2
Net:   eth0: ethernet@30be0000 [PRIME], eth1: ethernet@30bf0000
Fastboot: Normal
Saving Environment to MMC... Writing to redundant MMC(2)... OK
Normal Boot
Hit any key to stop autoboot:  0
u-boot=>
```

3.3.3.4. 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。「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

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

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

   Welcome to Alpine Linux 3.16
   Kernel 5.10.118-1-at on an aarch64 (/dev/ttymxc1)
   ```

```
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。

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。

3.3.3.5. 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!
urandom         | * Saving random seed ...chronyd         | * Stopping chronyd ...syslog              ⏎
| * Stopping busybox syslog ... [ ok ]
overlayfs       | * Unmounting /etc ...rngd             | * Stopping rngd ...podman-atmark    | *     ⏎
Stopping Start podman containers ... [ ok ] [ ok ]

 * start-stop-daemon: no matching processes found
podman-atmark    | [ ok ]
 [ ok ]
udev            | * Stopping udev ...hwclock          | * Setting hardware clock using the system     ⏎
clock [UTC] ... [ ok ]
 * in use but fuser finds nothing
overlayfs       | [ !! ]
overlayfs       | * Unmounting /dev/shm/overlay_etc_lower ... [ ok ]
networkmanager  | * Stopping networkmanager ...overlayfs        | * Unmounting /var ...nm-              ⏎
dispatcher: req:1 'connectivity-change': find-scripts: Cannot execute '/etc/Netw.
 [ ok ]
dbus            | * Stopping System Message Bus ...nm-dispatcher: System bus stopped. Exiting
 * in use but fuser finds nothing
overlayfs       | [ !! ]
 [ ok ]
overlayfs       | * Unmounting /dev/shm/overlay_var_lower ... [ ok ]
```

```
overlayfs        | * Unmounting /root ... [ ok ]
overlayfs        | * Unmounting /dev/shm/overlay_root_lower ... [ ok ]
localmount       | * Unmounting loop devices
overlayfs        | * Unmounting /home ...localmount       | * Unmounting filesystems
localmount       | *   Unmounting /var/tmp ... [ ok ]
localmount       | *   Unmounting /var/app/volumes ... [ ok ]
localmount       | *   Unmounting /var/app/rollback/volumes ... [ ok ]
localmount       | *   Unmounting /var/lib/containers/storage_readonly ... [ ok ]
localmount       | *   Unmounting /var/log ... [ ok ]
localmount       | *   Unmounting /tmp ... [ ok ]
localmount       | *   Unmounting /home ...
 *   in use but fuser finds nothing
localmount       | [ !! ]
localmount       | *   Unmounting /var ... [ ok ]
localmount       | *   Unmounting /etc ... *   in use but fuser finds nothing
localmount       | [ !! ]
 [ ok ]
overlayfs        | * Unmounting /dev/shm/overlay_home_lower ... [ ok ]
killprocs        | * Terminating remaining processes ...mount-ro         | * Remounting remaining
filesystems read-only ... *   Remounting /etc read only ... [ ok ]
mount-ro         | *   Remounting / read only ... [ ok ]
mount-ro         | [ ok ]
The system is going down NOW!
Sent SIGTERM to all processes
Sent SIGKILL to all processes
Requesting system poweroff
[   62.855146] imx2-wdt 30280000.watchdog: Device shutdown: Expect reboot!
[   62.862470] reboot: Power down
```

poweroff コマンドを使用せずに安全に終了できるかは、3G/LTE モジュールを搭載しているかによっても変わります。

- 3G/LTE モジュールが搭載されていない場合

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

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

- 3G/LTE モジュールが搭載されている場合

  3G/LTE モジュールは poweroff 時にシャットダウン処理を行いますので、3G/LTE モジュールが搭載されている Armadillo-IoT ゲートウェイ G4 の場合は、電源を切断する前に必ず poweroff コマンドを実行し、「reboot: Power down」と表示されたのを確認してから電源を切断してください。

  コンソールを確認できない環境で、スイッチなどを使用して電源オフ操作を実施する場合は、電源オフ操作実施後 3G/LTE ステータス確認用 LED (LED20) が消灯されたのを確認してから AC アダプタを抜いてください。3G/LTE ステータス確認用 LED の場所は、 「図 2.10. インターフェースレイアウト LTE モデル(ケース正面)」 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。

3.3.4. VSCode のセットアップ

Armadillo-IoT ゲートウェイ G4 の開発には、 VSCode を使用します。開発前に以下の手順を実施して、 ATDE に VSCode 及び、開発用エクステンションとクロスコンパイル用ライブラリをインストールしてください。

以下の手順は全て ATDE 上で実施します。

3.3.4.1. ソフトウェアのアップデート

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

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

図 3.19 ソフトウェアをアップデートする

VSCode を起動するには code コマンドを実行します。

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

図 3.20 VSCode を起動する

VSCode を起動すると、日本語化エクステンションのインストールを提案して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 VSCode を日本語化できます。

3.3.4.2. VSCode に開発用エクステンションをインストールする

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

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

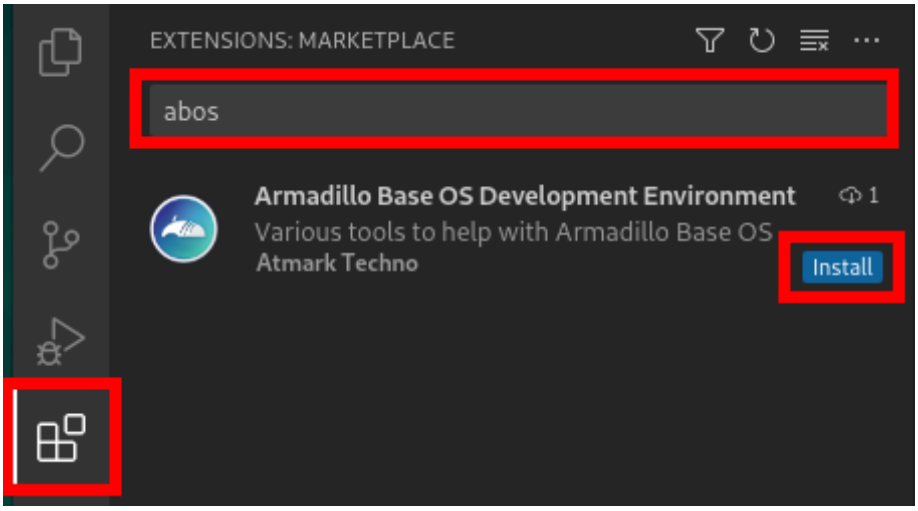


図 3.21 VSCode に開発用エクステンションをインストールする

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

3.3.4.3. クロスコンパイル用ライブラリをインストールする

ATDE のバージョン v20230911 以上では、 クロスコンパイル用ライブラリはインストール済みのため、ここでの手順は不要です。

ライブラリのビルドツールを実行する準備として、git のユーザ名とメールアドレスの設定を行い、ビルドツールである at-imxlibpackage をインストールします。

```
[ATDE ~]$ git config --global user.name "Your name"
[ATDE ~]$ git config --global user.email your@mail.tld
[ATDE ~]$ sudo apt update
[ATDE ~]$ sudo apt install at-imxlibpackage
```

図 3.22 ビルドツール実行前の準備

その後、ビルドツールを実行します。

実行中にライセンスへの同意を求められます。内容を確認の上、同意する場合は y を入力して処理を進めてください。

```
[ATDE ~]$ mkdir at-imxlibpackage
[ATDE ~]$ cd at-imxlibpackage
[ATDE ~/at-imxlibpackage]$ make-imxlibpkg
```

図 3.23 ビルドツールの実行

実行が完了すると、ATDE にクロスコンパイル用のライブラリがインストールされます。

3.3.5. VSCode を使用して Armadillo のセットアップを行う

ここでは VSCode を使用した Armadillo のセットアップ方法を紹介します。

3.3.5.1. initial_setup.swu の作成

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、開発開始時に initial_setup.swu のインストールを行う必要があります。

VSCode の左ペインの [COMMON PROJECT COMMAND] から [Generate initial setup swu] を実行すると、initial_setup.swu が作成されます。

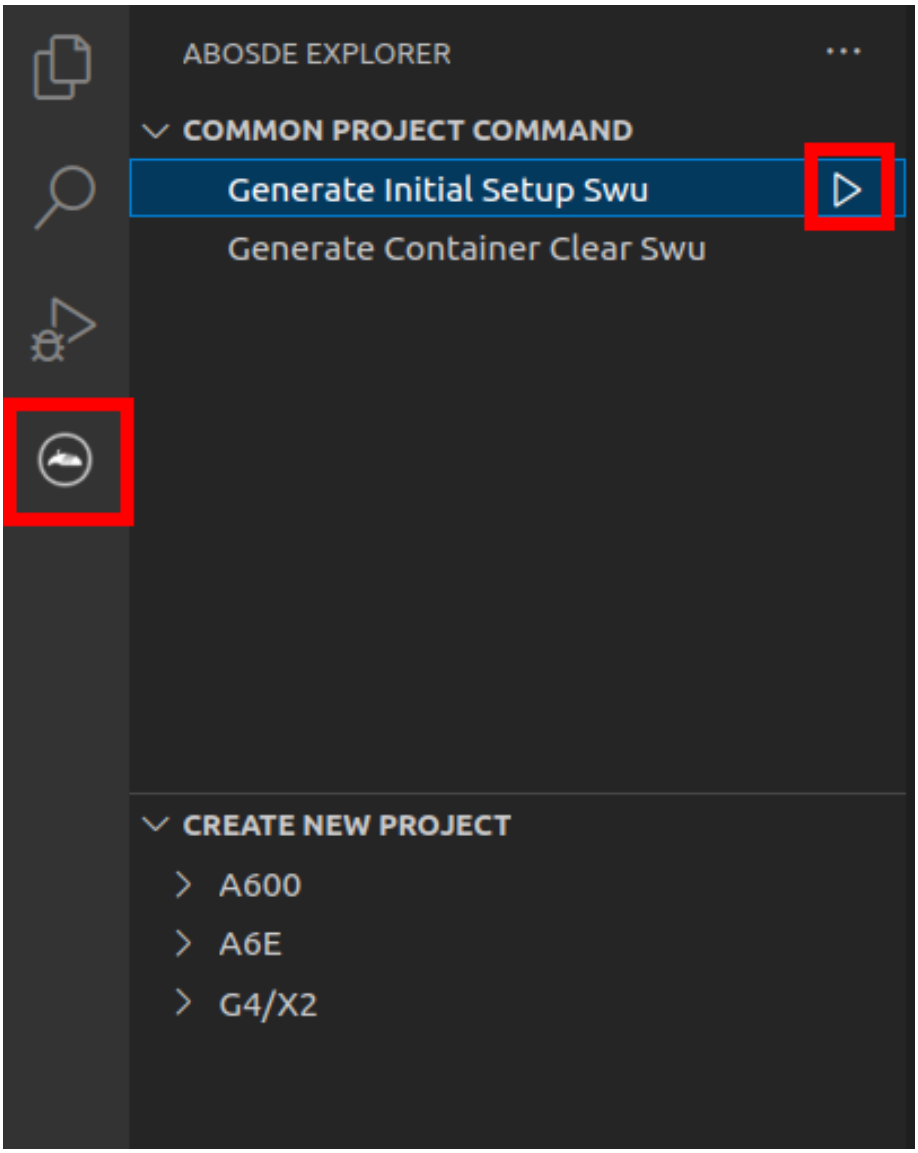


図 3.24 initial_setup.swu を作成する

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 3.25. initial_setup.swu 初回生成時の各種設定」 に示します。

```
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 3.25 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。

❻ atmark ユーザーのパスワードを 2 回入力します。何も入力しない場合はユーザーをロックします。

❼ 自動アップデートを無効のままで進みます。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。

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 に保存されています。この SWU イメージを 「3.2.3.5. 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 ~/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。

3.3.6. ユーザー登録

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

ユーザー登録すると次のようなサービスを受けることができます。

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

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

アットマークテクノ Armadillo サイト

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

3.3.6.1. 購入製品登録

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

購入製品登録の詳しい手順は以下の URL をご参照ください。

Armadillo-IoT ゲートウェイ G4 購入製品登録

https://armadillo.atmark-techno.com/armadillo-iot-g4/register

3.4. ハードウェアの設計

Armadillo-IoT ゲートウェイ G4 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-IoT ゲートウェイ G4 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

LVDS インターフェース(CON9)や HDMI インターフェース(CON8)にディスプレイを接続した場合、放射ノイズが問題になる場合があります。放射ノイズを減らすために、以下の対策が効果的です。

・ シールド付のケーブルを使用する

・ ケーブルは最短で接続する

・ 未使用の拡張ピンは Low レベル出力とする

3.4.3. ESD/雷サージ

Armadillo-IoT ゲートウェイ G4 の ESD 耐性を向上させるために、以下の対策が効果的です。

・ Armadillo-IoT ゲートウェイ G4 を金属筐体に組み込み、GND(固定穴)を金属ねじ等で接続する

・ 金属筐体を接地する

Armadillo-IoT ゲートウェイ G4 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

SoC(基板裏の IC1)の放熱が必要かどうかは、使用状況により違いますので、十分な設計評価の上、ご検討ください。SoC の表面温度が 90℃以上になる場合は、放熱することを推奨いたします。

参考までに、下記条件の場合に SoC の表面温度が 90℃を超えることを確認しています。

- 基板単体
- 周囲温度: 約 65℃
- microSD/HDMI/USB3.0/LANx2 動作

Armadillo-IoT ゲートウェイ G4 の周囲温度の上限は+70℃としていますが、これは下記条件の場合の温度となります。

- 基板をケースに収納(放熱シートあり)
- microSD/HDMI/USB3.0/LANx2 動作

オプションケース(金属製)は、SoC の熱をケースに伝導させて放熱する構造で設計しております。同様の構造でのケース設計をご検討の場合は、「6.30.1. オプションケース(金属製)」をご確認ください。

SoC 近辺にヒートシンク固定用の穴(φ2.5mm × 2)を準備していますので、ヒートシンクからの放熱も可能です。寸法につきましては、「3.4.9. 形状図」をご確認ください。

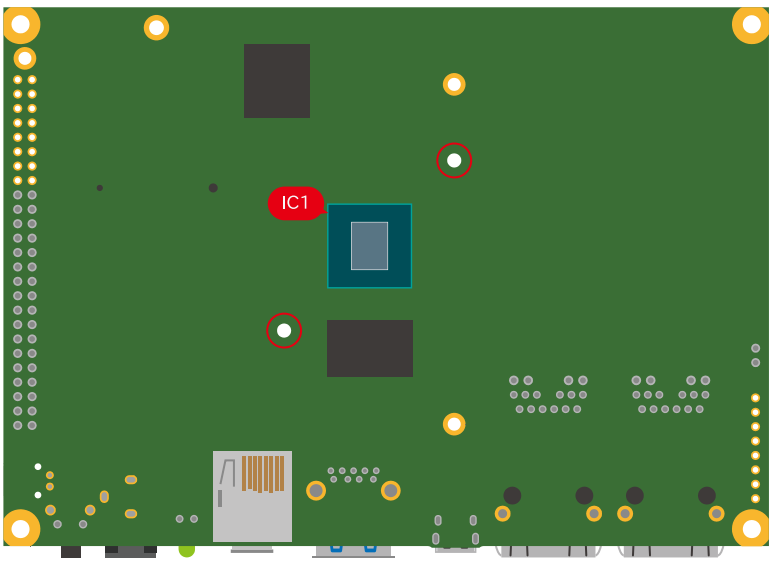


図 3.26 Armadillo-IoT ゲートウェイ G4 の IC1 とヒートシンク固定穴の位置

Armadillo-IoT ゲートウェイ G4 では、温度センサーで CPU(Arm Cortex-A53)周辺温度、SoC(ANAMIX 内部)温度を測定することが可能です。温度センサーの詳細につきましては、「6.19.7. 温度センサーの仕様」をご確認ください。

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「6.19. 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. CON11、 CON12(拡張インターフェース)

CON11、CON12 は機能拡張用のインターフェースです。複数の機能(マルチプレクス)をもつ、i.MX 8M Plus の信号線が接続されており、USB、GPIO、SPI、UART、CAN、I2C、PWM、I2S、PDM MIC 等の機能を拡張することができます。

Armadillo-IoT ゲートウェイ G4 のハードウェアを拡張する際には、主にこの拡張インターフェースに接続していくことになります。

ONOFF 信号等の電源制御用の信号も接続されており、Armadillo-IoT ゲートウェイ G4 の電源を外部からの信号により制御することが可能です。また、電源入出力ピン(VIN)より電源供給することも可能です。

CON11、CON14、CON15 の電源ライン(VIN)は接続されており、同時に電源を供給することはできません。

CON11 の 5、7 ピンと CON9 の 7、8 ピンは同じ I2C バス(I2C4)に接続されています。

表 3.5 CON11、CON12 搭載コネクタと対向コネクタ例

名称	型番	メーカー	備考
搭載コネクタ	6130xx21121 [a]	Wurth Elektronik	許容電流 3A(端子 1 本あたり)
対向コネクタ	6130xx21821 [a]	Wurth Elektronik	-

[a]xx にはピン数が入ります。

表 3.6 CON11 信号配列

ピン番号	ピン名	I/O	説明	電圧グループ
1	VIN	Power	電源入出力(VIN)、CON14、CON15 と共通	-
2	VIN	Power	電源入出力(VIN)、CON14、CON15 と共通	-
3	GND	Power	電源(GND)	-
4	GND	Power	電源(GND)	-
5	I2C4_SCL	In/Out	拡張入出力、i.MX 8M Plus の I2C4_SCL ピン、CON9 7 ピンに接続 基板上で 4.7k プルアップ	VDD_1V8

ピン番号	ピン名	I/O	説明	電圧グループ
6	ECSPI1_MISO	In/Out	拡張入出力、i.MX 8M Plus の ECSPI1_MISO ピンに接続	VDD_1V8
7	I2C4_SDA	In/Out	拡張入出力、i.MX 8M Plus の I2C4_SDA ピン、CON9 8 ピンに接続 基板上で 4.7k プルアップ	VDD_1V8
8	ECSPI1_MOSI	In/Out	拡張入出力、i.MX 8M Plus の ECSPI1_MOSI ピンに接続	VDD_1V8
9	ECSPI2_MISO	In/Out	拡張入出力、i.MX 8M Plus の ECSPI2_MISO ピンに接続	VDD_1V8
10	ECSPI1_SCLK	In/Out	拡張入出力、i.MX 8M Plus の ECSPI1_SCLK ピンに接続	VDD_1V8
11	ECSPI2_MOSI	In/Out	拡張入出力、i.MX 8M Plus の ECSPI2_MOSI ピンに接続	VDD_1V8
12	ECSPI1_SS0	In/Out	拡張入出力、i.MX 8M Plus の ECSPI1_SS0 ピンに接続	VDD_1V8
13	ECSPI2_SCLK	In/Out	拡張入出力、i.MX 8M Plus の ECSPI2_SCLK ピンに接続	VDD_1V8
14	SAI3_TXFS	In/Out	拡張入出力、i.MX 8M Plus の SAI3_TXFS ピンに接続	VDD_1V8
15	ECSPI2_SS0	In/Out	拡張入出力、i.MX 8M Plus の ECSPI2_SS0 ピンに接続	VDD_1V8
16	SAI3_TXC	In/Out	拡張入出力、i.MX 8M Plus の SAI3_TXC ピンに接続	VDD_1V8
17	SAI5_RXC	In/Out	拡張入出力、i.MX 8M Plus の SAI5_RXC ピンに接続	VDD_1V8
18	SAI3_TXD	In/Out	拡張入出力、i.MX 8M Plus の SAI3_TXD ピンに接続	VDD_1V8
19	SAI5_RXD0	In/Out	拡張入出力、i.MX 8M Plus の SAI5_RXD0 ピンに接続	VDD_1V8
20	SAI3_RXD	In/Out	拡張入出力、i.MX 8M Plus の SAI3_RXD ピンに接続	VDD_1V8
21	SAI5_RXD1	In/Out	拡張入出力、i.MX 8M Plus の SAI5_RXD1 ピンに接続	VDD_1V8
22	SAI3_MCLK	In/Out	拡張入出力、i.MX 8M Plus の SAI3_MCLK ピンに接続	VDD_1V8
23	SAI5_RXD2	In/Out	拡張入出力、i.MX 8M Plus の SAI5_RXD2 ピンに接続	VDD_1V8
24	GPIO1_IO15	In/Out	拡張入出力、i.MX 8M Plus の GPIO1_IO15 ピンに接続	VDD_1V8
25	SAI5_RXD3	In/Out	拡張入出力、i.MX 8M Plus の SAI5_RXD3 ピンに接続	VDD_1V8
26	USBDM_DN2	In/Out	USB 2.0 データ(-)、USB HUB 経由で i.MX 8M Plus の USB2 に接続	-
27	SAI5_MCLK	In/Out	拡張入出力、i.MX 8M Plus の SAI5_MCLK ピンに接続	VDD_1V8
28	USBDP_DN2	In/Out	USB 2.0 データ(+)、USB HUB 経由で i.MX 8M Plus の USB2 に接続	-
29	SAI5_RXFS	In/Out	拡張入出力、i.MX 8M Plus の SAI5_RXFS ピンに接続	VDD_1V8
30	VDD_1V8	Power	電源出力(VDD_1V8)	-
31	VDD_5V	Power	電源出力(VDD_5V)	-
32	VDD_5V	Power	電源出力(VDD_5V)	-
33	GND	Power	電源(GND)	-
34	GND	Power	電源(GND)	-

表 3.7 CON12 信号配列

ピン番号	ピン名	I/O	説明	電圧グループ
1	PWR_OFF	In/Out	Armadillo Base OS で使用、ユーザーによる変更可能、 i.MX 8M Plus の GPIO1_IO01 ピンに接続	VDD_1V8

ピン番号	ピン名	I/O	説明	電圧グループ
2	GPIO2_IO11	In/Out	拡張入出力、i.MX 8M Plus の GPIO2_IO11 ピンに接続	VDD_1V8
3	REBOOT	In/Out	Armadillo Base OS で使用、ユーザーによる変更可能、 i.MX 8M Plus の GPIO1_IO05 ピンに接続	VDD_1V8
4	GPIO4_IO27	In/Out	拡張入出力、i.MX 8M Plus の GPIO4_IO27 ピンに接続	VDD_1V8
5	ONOFF	In	ONOFF 信号[a]、i.MX 8M Plus の ONOFF ピンに接続、基板上で 100k プルアップ(NVCC_SNVS_1V8)	NVCC_SNVS_1V8
6	GPIO4_IO28	In/Out	拡張入出力、i.MX 8M Plus の SAI3_RXFS ピンに接続	VDD_1V8
7	FW_UPDATE_IND	In/Out	Armadillo Base OS で使用、ユーザーによる変更可能、 i.MX 8M Plus の GPIO1_IO00 ピンに接続	VDD_1V8
8	GPIO4_IO29	In/Out	拡張入出力、i.MX 8M Plus の GPIO4_IO29 ピンに接続	VDD_1V8
9	STDWN_IND	In/Out	Armadillo Base OS で使用、ユーザーによる変更可能、 i.MX 8M Plus の GPIO1_IO07 ピンに接続	VDD_1V8
10	GPIO3_IO08	In/Out	拡張入出力、i.MX 8M Plus の NAND_DATA02 ピンに接続	VDD_1V8
11	PWR_IND	In/Out	Armadillo Base OS で使用、ユーザーによる変更可能、 i.MX 8M Plus の GPIO1_IO06 ピンに接続	VDD_1V8
12	GPIO3_IO09	In/Out	拡張入出力、i.MX 8M Plus の NAND_DATA03 ピンに接続	VDD_1V8
13	GPIO2_IO08	In/Out	拡張入出力、i.MX 8M Plus の SD1_DATA6 ピンに接続	VDD_1V8
14	GPIO3_IO14	In/Out	拡張入出力、i.MX 8M Plus の NAND_DQS ピンに接続	VDD_1V8
15	GND	Power	電源(GND)	-
16	GND	Power	電源(GND)	-

[a]ONOFF 信号の詳細につきましては、「3.4.8.9. 外部からの電源制御」をご確認ください。

拡張できる機能の詳細につきましては、「Armadillo-IoT ゲートウェイ G4 マルチプレクス表」 [https://armadillo.atmark-techno.com/resources/documents/armadillo-iot-g4/manual-multiplex] をご参照ください。

3.4.6. 拡張ボードの設計

Armadillo-IoT ゲートウェイ G4 の拡張インターフェース(CON11、CON12)には、複数の機能をもった信号線が接続されており、様々な機能拡張が可能です。

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

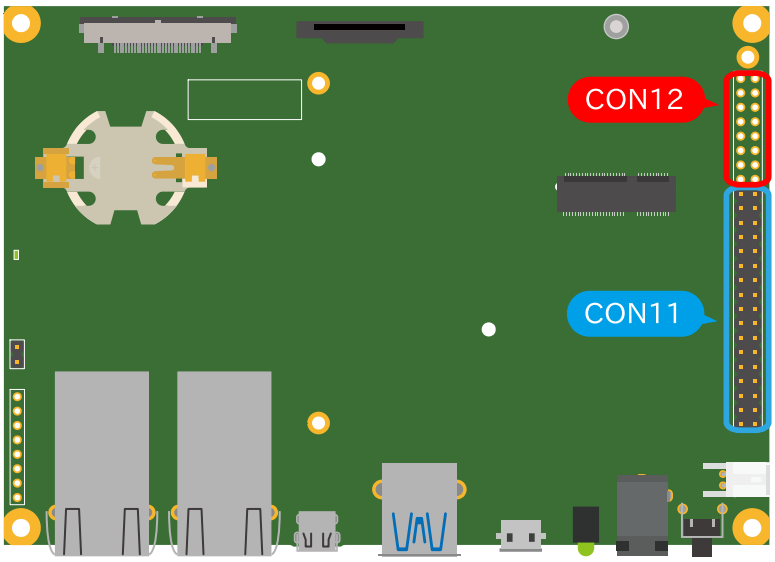


図 3.27 Armadillo-IoT ゲートウェイ G4 の拡張インターフェース

3.4.6.1. ピンアサイン

Armadillo-IoT ゲートウェイ G4 では、「表 2.2. 仕様」の拡張インターフェースの欄にあるとおりの機能が拡張できます。ただし、ここに記載の拡張数は、優先的に機能を割り当てた場合の最大数ですので、必要な機能がすべて実現できるかは、『Armadillo-IoT ゲートウェイ G4 マルチプレクス表』で検討する必要があります。

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

各機能の詳細な仕様が必要な場合は、NXP Semiconductors のホームページからダウンロード可能な、『i.MX 8M Plus Applications Processor Reference Manual』、『i.MX 8M Plus Applications Processor Datasheet for Industrial Products』をご確認ください。Armadillo-IoT ゲートウェイ G4 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. CON11、 CON12(拡張インターフェース)」に i.MX 8M Plus 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-IoT ゲートウェイ G4 マルチプレクス表は 「Armadillo-IoT ゲートウェイ G4 マルチプレクス表」 [https://armadillo.atmark-techno.com/resources/documents/armadillo-iot-g4/manual-multiplex]からダウンロードしてください。

3.4.6.2. 基板形状

Armadillo-IoT ゲートウェイ G4 の拡張ボードを設計する際の推奨形状は「図 3.28. Armadillo-IoT ゲートウェイ G4 の拡張ボード例」のとおりです。拡張ボード側にピンソケットを実装して Armadillo-IoT ゲートウェイ G4 と接続します。

一般的なピンソケットを実装した場合、嵌合高さは約 11mm となります。LAN コネクタの高さは 13.5mm ですので、LAN コネクタの上に基板を重ねることはできません。

拡張ボード固定用に、φ2.3mm の穴を 3 箇所用意しており、M2 のスペーサーとねじを接続可能です。

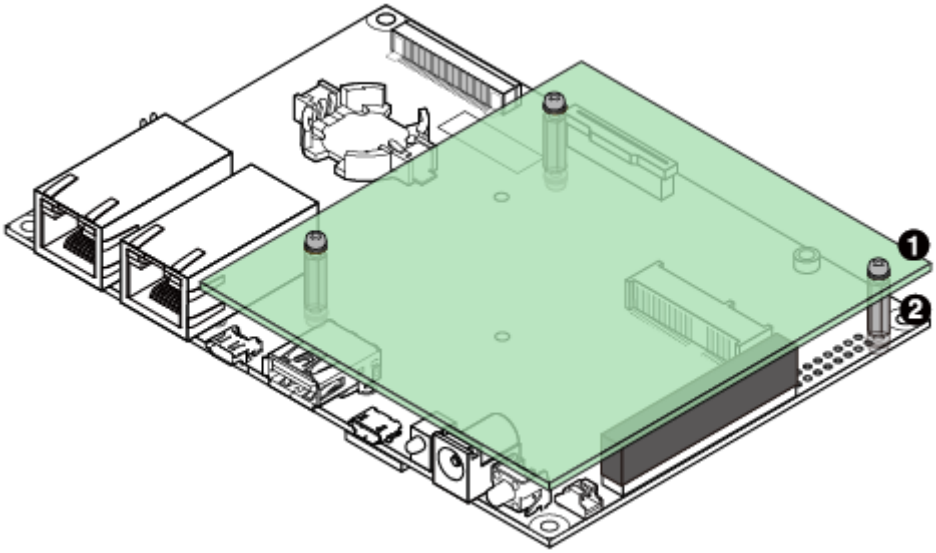


図 3.28 Armadillo-IoT ゲートウェイ G4 の拡張ボード例

❶ なべ小ねじ、ワッシャ、スプリングワッシャ付(M2、L=6mm) × 6

❷ 金属スペーサ(M2、L=11mm) × 3

基板の詳細寸法につきましては、「3.4.9.1. 基板形状図」をご確認ください。

3.4.7. 回路設計

拡張インターフェース(CON11、CON12)を使用する際の参考回路を紹介します。

参考回路は動作を保証するものではありません。実際のアプリケーションで十分な評価をお願いいたします。

3.4.7.1. 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には、i.MX 8M Plus の信号線が直接接続されています。静電気等による内部回路の故障を防ぐため、スイッチと i.MX 8M Plus の間に、電流制限抵抗等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。

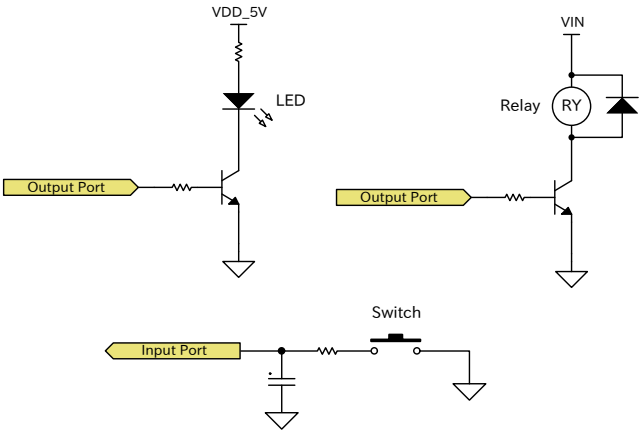


図 3.29 スイッチ、LED、リレー接続例

3.4.7.2. 電源

拡張インターフェース(CON11)から拡張ボード用に、12V 電圧(VIN)、5V 電源(VDD_5V)、1.8V 電源(VDD_1V8)を出力しています。その他の電源が必要な場合は、別途外部から入力するか、DC/DC コンバータ、LDO 等で生成してください。電源シーケンス、出力電流につきましては、「3.4.8.5. 電源回路の構成」をご確認ください。

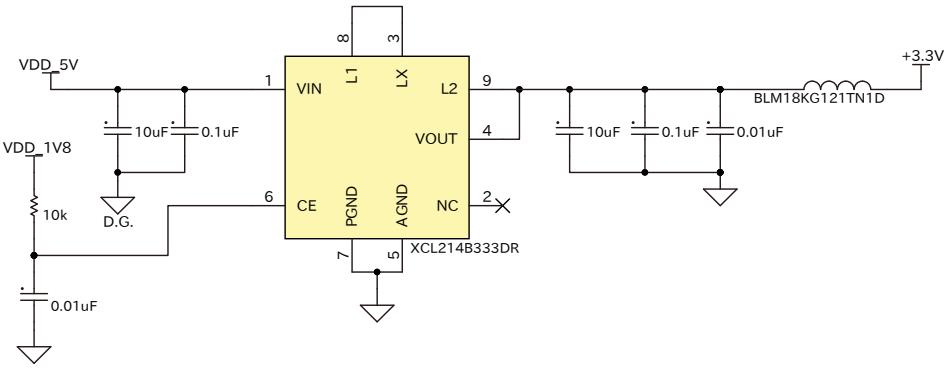


図 3.30 DC/DC コンバータ回路(VDD_5V 入力、3.3V 1.5A 出力)例

「図 3.32. 電源回路の構成」のインターフェース名(LVDS I/F 等)の左横にはコネクタもしくはノイズフィルタの定格電流値を最大値として記載しています。また、パワースイッチの下には、パワースイッチの制限電流値を最大値として記載しています。スイッチングレギュレータの供給能力を超えてしまうため、インターフェースすべての最大値まで電流供給することはできません。それぞれのインターフェースへの推奨の電流供給値は以下のとおりです。

表 3.8 各インターフェースへの電流供給例

部品番号	インターフェース名	電圧グループ	電流値
CON4	USB インターフェース	USB1_VBUS	900mA
CON9	LVDS インターフェース	VIN	入力電源に依存
		VDD_1V8	500mA
CON10	MIPI-CSI インターフェース	VEXT3V3	500mA

部品番号	インターフェース名	電圧グループ	電流値
CON11	拡張インターフェース	VIN	入力電源に依存
		VDD_5V	1A
		VDD_1V8	500mA

動作させるアプリケーションにより、内部で消費する電流値は大きく変わりますので、動作検証の上、供給電源の設計を行なってください。

3.4.7.3. レベル変換

拡張インターフェース(CON11、CON12)の拡張入出力ピンの電圧レベルは 1.8V(VDD_1V8)です。異なる電圧レベルのデバイスを接続する場合は、レベル変換が必要となります。CON11 には VDD_1V8、VDD_5V ピンがありますので、適宜ご活用ください。レベル変換 IC は、立ち上がり、立ち下がり速度、遅延時間、ドライブ能力等を考慮し、適切なものを選定してください。

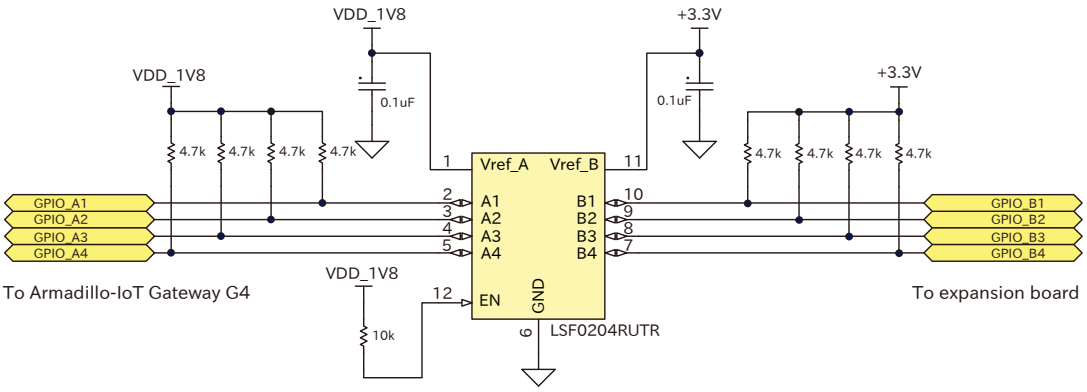


図 3.31 1.8V ←→ 3.3V 双方向レベル変換回路の例

上記レベル変換 IC は 1.8V ←→ 5V でも使用可能です。

3.4.8. 電気的仕様

3.4.8.1. 絶対最大定格

表 3.9 絶対最大定格

項目	記号	Min.	Max.	単位	備考
電源電圧	VIN	-0.3	13.2	V	-

項目	記号	Min.	Max.	単位	備考
入出力電圧	VI,VO (VDD_3V3, VDD_1V8, NVCC_SNVS_1V8)	-0.3	OVDD+0.3	V	OVDD=VDD_3V3, VDD_1V8, NVCC_SNVS_1V8
	VI,VO(M2_3V3)	-0.5	4.6	V	-
	VI,VO(VEXT_3V3)	-0.5	7.0	V	-
USB コンソール電源電圧	VBUS_CNSL	-0.3	5.8	V	-
RTC バックアップ電源電圧	RTC_BAT	-0.3	5.5	V	-
動作温度範囲(LAN モデル)	Topr	-20	70	℃	結露なきこと
動作温度範囲(LTE モデル)	Topr	-20	60	℃	結露なきこと

絶対最大定格は、あらゆる使用条件や試験状況において、瞬時でも超えてはならない値です。上記の値に対して余裕をもってご使用ください。

3.4.8.2. 推奨動作条件

表 3.10 推奨動作条件

項目	記号	Min.	Typ.	Max.	単位	備考
電源電圧	VIN	10.8	12	13.2	V	-
USB コンソール電源電圧	VBUS_CNSL	3.0	-	5.25	V	-
RTC バックアップ電源電圧	RTC_BAT	2.4	3	3.6	V	Topr=+25℃
使用温度範囲(LAN モデル)	Ta	-20	25	70	℃	結露なきこと
使用温度範囲(LTE モデル)	Ta	-20	25	60	℃	結露なきこと

3.4.8.3. 電源出力仕様

表 3.11 電源出力仕様

項目	記号	Min.	Typ.	Max.	単位	備考
5V 電源	VDD_5V USB1_VBUS HDMI_5V	4.85	5	5.15	V	-
3.3V 電源	VDD_3V3 VDD_SD VEXT_3V3 M2_3V3	3.135	3.3	3.465	V	-
1.8V 電源	VDD_1V8 NVCC_SNVS_1V8	1.71	1.8	1.89	V	-

3.4.8.4. 入出力インターフェースの電気的仕様

表 3.12 拡張入出力ピンの電気的仕様(OVDD=VDD_3V3, VDD_1V8)

項目	記号	Min.	Typ.	Max.	単位	備考
ハイレベル出力電圧	VOH (VDD_1V8)	0.8xOVDD	-	OVDD	V	IOH = 1.6/3.2/6.4/9.6mA
	VOH (VDD_3V3)	0.8xOVDD	-	OVDD	V	IOH = 2/4/8/12mA
ローレベル出力電圧	VOL (VDD_1V8)	0	-	0.2xOVDD	V	IOL = 1.6/3.2/6.4/9.6mA
	VOL (VDD_3V3)	0	-	0.2xOVDD	V	IOL = 2/4/8/12mA

項目	記号	Min.	Typ.	Max.	単位	備考
ハイレベル入力電圧	VIH	0.7×OVDD	-	OVDD+0.3	V	-
ローレベル入力電圧	VIL	-0.3	-	0.3×OVDD	V	-
Pull-up 抵抗 (VDD_1V8)	-	12	22	49	kΩ	-
Pull-down 抵抗 (VDD_1V8)	-	13	23	48	kΩ	-
Pull-up 抵抗 (VDD_3V3)	-	18	37	72	kΩ	-
Pull-down 抵抗 (VDD_3V3)	-	24	43	87	kΩ	-

表 3.13 拡張入出力ピンの電気的仕様(OVDD=VEXT_3V3)

項目	記号	Min.	Typ.	Max.	単位	備考
ハイレベル出力電流	IOH	-	-	0.7	mA	-
ローレベル出力電流	IOL	-	-	1	mA	-
ハイレベル入力電圧	VIH	0.7×OVDD	-	OVDD+0.3	V	-
ローレベル入力電圧	VIL	-0.3	-	0.3×OVDD	V	-

3.4.8.5. 電源回路の構成

Armadillo-IoT ゲートウェイ G4 の電源回路の構成は「図 3.32. 電源回路の構成」のとおりです。

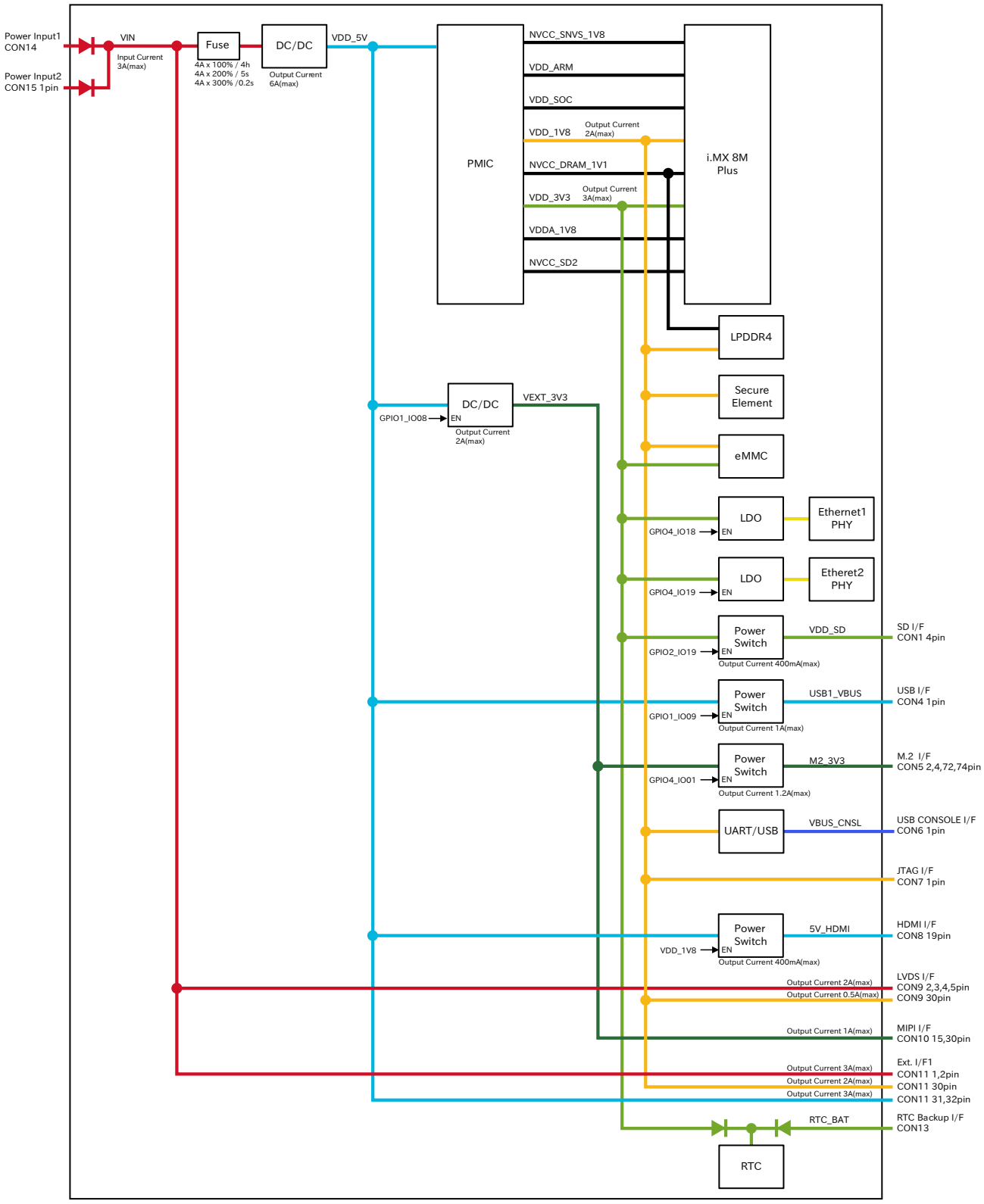


図 3.32 電源回路の構成

入力電圧(VIN)を電源 IC 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。

外部インターフェースへの電源は GPIO によりオンオフ制御できるようになっており、不要な場合はオフすることで、省電力化が可能です。

3.4.8.6. 電源シーケンス

電源シーケンスは「図 3.33. 電源シーケンス」のとおりです。

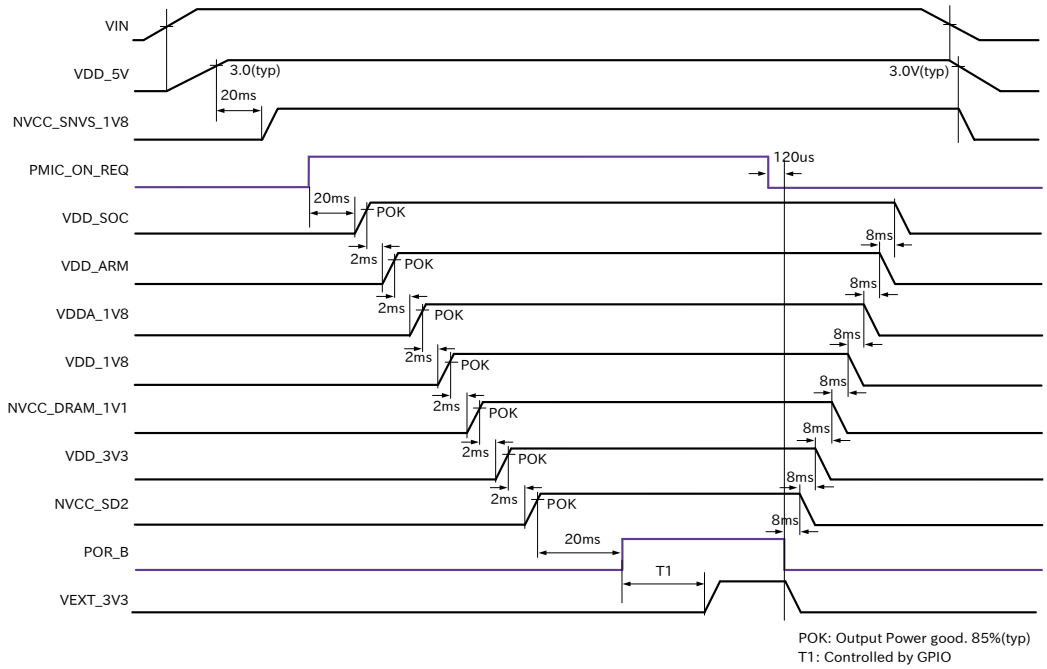


図 3.33 電源シーケンス

- 電源オン時

  Armadillo-IoT ゲートウェイ G4 に電源(VIN)を投入すると、VDD_5V、NVCC_SNVS_1V8 の順で電源が立ち上がり、i.MX 8M Plus からパワーマネジメント IC に PMIC_ON_REQ 信号が出力されます。パワーマネジメント IC は PMIC_ON_REQ 信号のアサートを検知後、電源オンシーケンスを開始し、VDD_SOC、VDD_ARM、VDDA_1V8、VDD_1V8、NVCC_DRAM_1V1、VDD_3V3、NVCC_SD2 の順に電源を立ち上げます。POR_B 信号が解除されると、ソフトウェアにより、VEXT_3V3 を任意のタイミングで立ち上げることが可能です。

- 電源オフ時

  poweroff コマンドにより、POR_B 信号がアサートされると、パワーマネジメント IC は電源オフシーケンスを開始し、電源オンシーケンスとは逆の順番で電源を立ち下げます。Armadillo-IoT ゲートウェイ G4 の電源(VIN)を切断すると、VDD_5V、NVCC_1V8 の順で電源が立ち下がります。

3.4.8.7. リセット回路の構成

Armadillo-IoT ゲートウェイ G4 のリセット回路の構成は「図 3.34. リセット回路の構成」のとおりです。

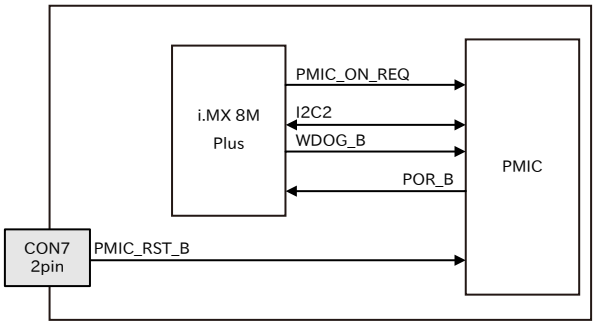


図 3.34 リセット回路の構成

3.4.8.8. リセットシーケンス

リセットシーケンスは「図 3.35. リセットシーケンス」のとおりです。

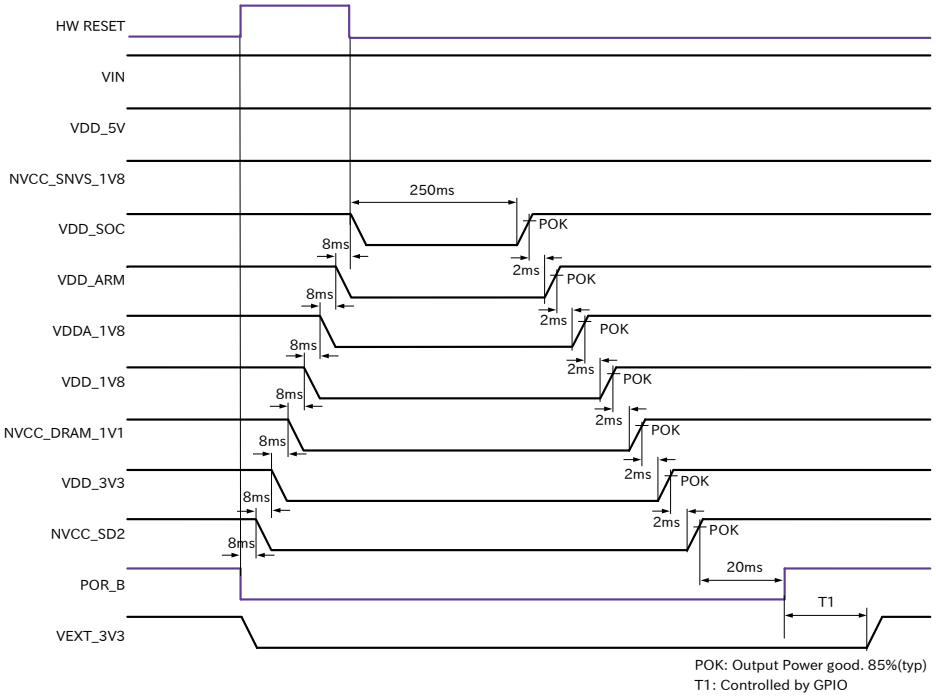


図 3.35 リセットシーケンス

Armadillo-IoT ゲートウェイ G4 のハードウェアリセットには、I2C によるリセット[3]、ウォッチドックタイマーによるリセット、JTAG インターフェースの PMIC_RST_B 信号(CON7 2 ピン)によるリセットの 3 つがあります。

パワーマネジメント IC が、ハードウェアリセットを検知すると、POR_B 信号をアサートして電源オフシーケンスを開始し、VIN、VDD_5V、NVCC_SNVS_1V8 以外の電源を切断します。I2C によるリセット、ウォッチドックタイマーによるリセットの場合、電源オフシーケンスが終わった 250ms 後に電源オンシーケンスを開始し、電源が再投入されます。PMIC_RST_B 信号によるリセットの場合、PMIC_RST_B 信号がディアサートされた 250ms 後に電源オンシーケンスを開始し、電源が再投入され

[3]reboot コマンド時は、I2C によりリセットされます。

ます。PMIC_RST_B 信号によりリセットするためには、50 ミリ秒以上、Low レベルを保持する必要があります。

3.4.8.9. 外部からの電源制御

・ ONOFF ピンからの電源制御

拡張インターフェースの ONOFF 信号(CON12 5 ピン)およびリアルタイムクロックの割り込み信号は、i.MX 8M Plus の ONOFF ピンに接続されています。

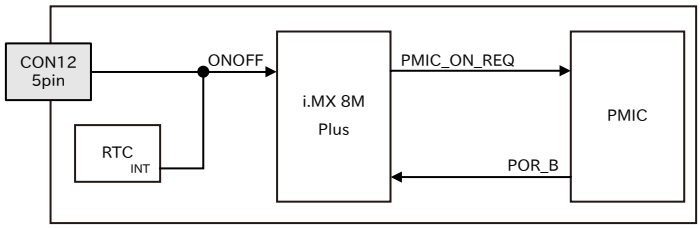


図 3.36 ONOFF 回路の構成

ONOFF 信号を一定時間以上、Low レベルとすることで、i.MX 8M Plus のオン状態、オフ状態を切り替えることができます。オン状態になると、PMIC_ON_REQ 信号がアサートされ、ソフトウェアからの制御で電源切断しているものを除いて、すべての電源が供給されます。オフ状態になると、PMIC_ON_REQ 信号がディアサートされ、VIN、VDD_5V、NVCC_SNVS_1V8 以外の電源が切断されます。オン状態からオフ状態に切り替える場合は 5 秒以上、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。

表 3.14 オン状態、オフ状態を切り替えする際の Low レベル保持時間

状態	Low レベル保持時間
オン状態からオフ状態	5 秒以上
オフ状態からオン状態	500 ミリ秒以上

ONOFF 信号を制御する場合は、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。オン状態およびオフ状態は、NVCC_SNVS_1V8 が供給されている限り、保持されます。

オフ状態にして Armadillo-IoT ゲートウェイ G4 の電源(VIN)を切断した場合、コンデンサに蓄えられた電荷が抜けるまではオフ状態であることが保持されます。オフ状態を保持した状態で電源を投入したくない場合は、一定時間以上空けて電源を投入する必要があります。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

3.4.9. 形状図

3.4.9.1. 基板形状図

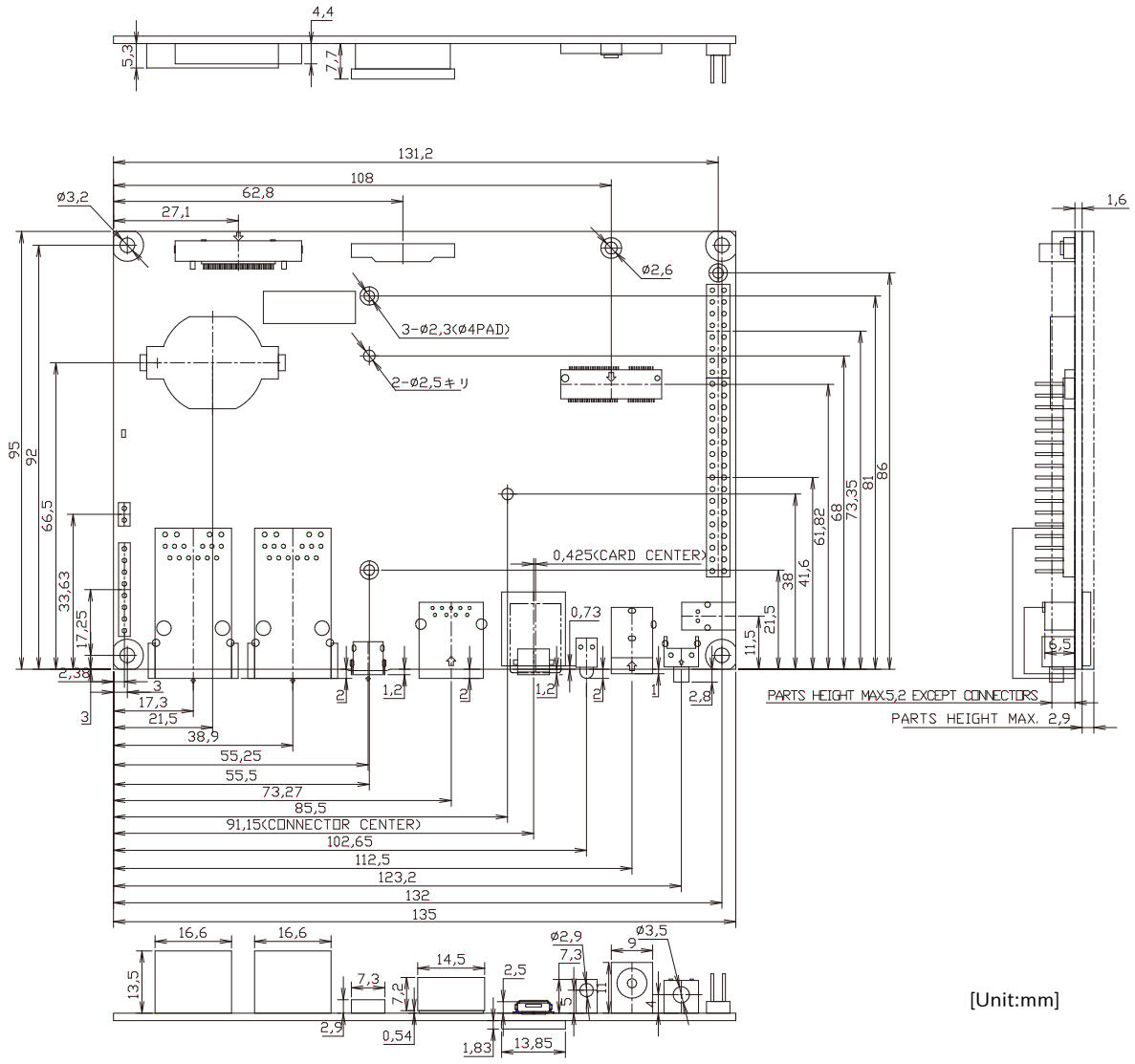


図 3.37 基板形状図

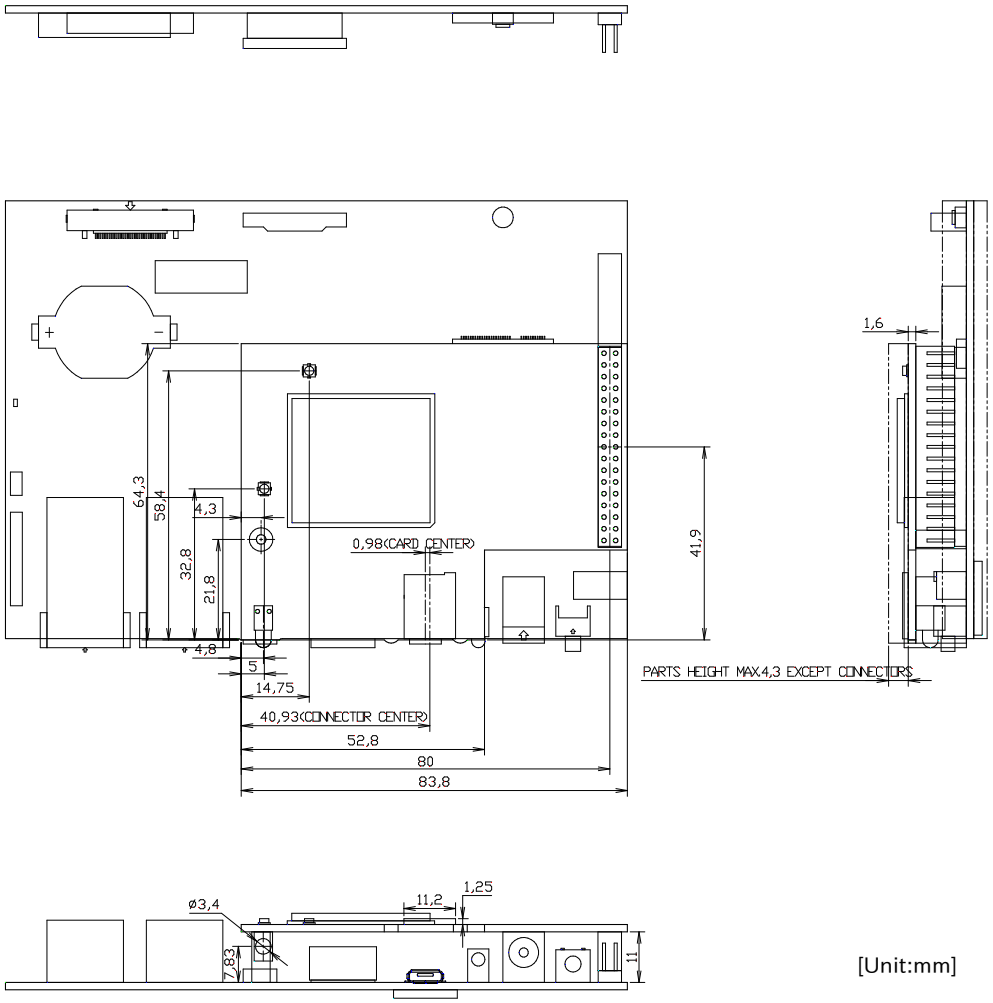


図 3.38 基板形状図(LTE モデル 追加分)

基板改版や部品変更により、基板上の部品位置、高さは変更になることがあります。ケースや拡張基板を設計する場合、ある程度の余裕をもった寸法での設計をお願いいたします。

DXF 形式の形状図を「アットマークテクノ Armadillo サイト」から「購入者向けの限定公開データ」としてダウンロード可能です。

3.4.10. オプション品

Armadillo-IoT ゲートウェイ G4 のオプション品については、「6.30. 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 Device Tree の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(CON11、CON12)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、「6.25.2. Linux カーネルをビルドする」を参照してください。

Armadillo-IoT ゲートウェイ G4 Linux カーネル [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/linux-kernel] から「Linux カーネル」ファイル (linux-at-x2-[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-x2-[VERSION].tar
[ATDE ~]$ tar xf linux-at-x2-[VERSION]/linux-[VERSION].tar.gz
[ATDE ~]$ cd linux-[VERSION]
```

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

```
[ATDE ~/linux-[VERSION]]$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- x2_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。

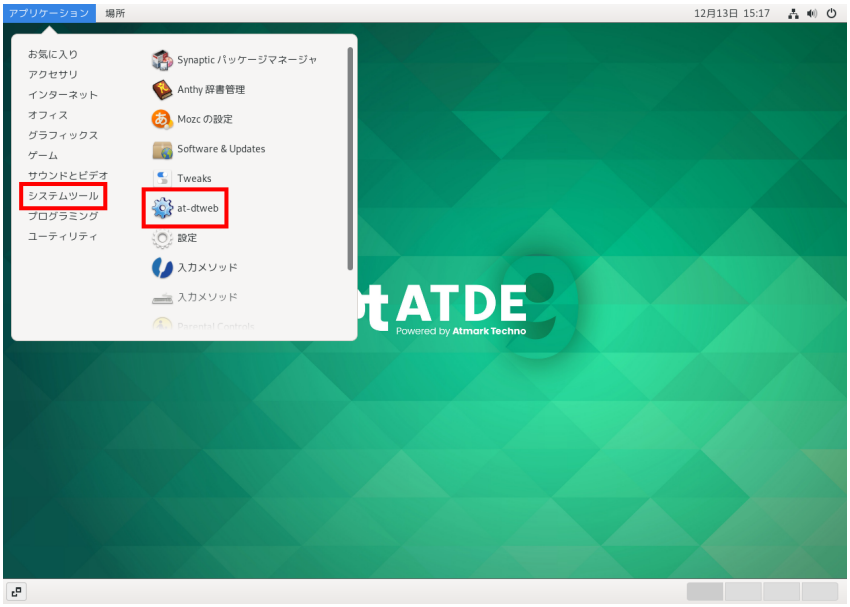


図 3.39 at-dtweb の起動開始

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-IoT_G4 を選択して、「OK」をクリックします。

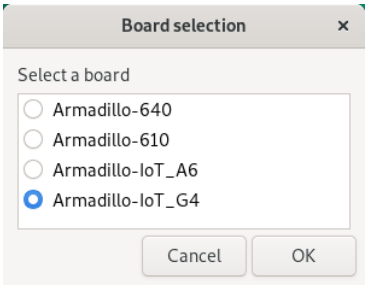


図 3.40 ボード選択画面

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。「3.5.1. Linux カーネルソースコードの取得」で準備した 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。

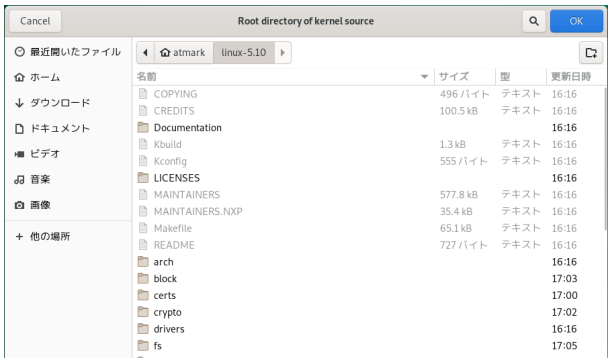


図 3.41 Linux カーネルディレクトリ選択画面

at-dtweb のバージョン が 2.7.0 以降の場合は、Linux カーネルソースのバージョン v5.10.112-r0 以降を、mkswu のバージョンは 3.15.at.6 以降を使用してください。

at-dtweb と mkswu のバージョンは以下のコマンドで調べることができます。

```
[ATDE ~]$ dpkg -l at-dtweb mkswu
要望=(U)不明/(I)インストール/(R)削除/(P)完全削除/(H)保持
| 状態=(N)無/(I)インストール済/(C)設定/(U)展開/(F)設定失敗/(H)半インストール/(W)トリガ待ち/(T)トリガ保留
|/ エラー?=(空欄)無/(R)要再インストール（状態,エラーの大文字=異常）
||/ 名前           バージョン    アーキテクチ 説明
+++-==============-============-============-=================================================
ii  at-dtweb       2.7.0        all          Device Tree editor for Atmark Techno products
ii  mkswu          3.15.at.6-1  all          build SWU images for armadillo base OS
```

1. 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。

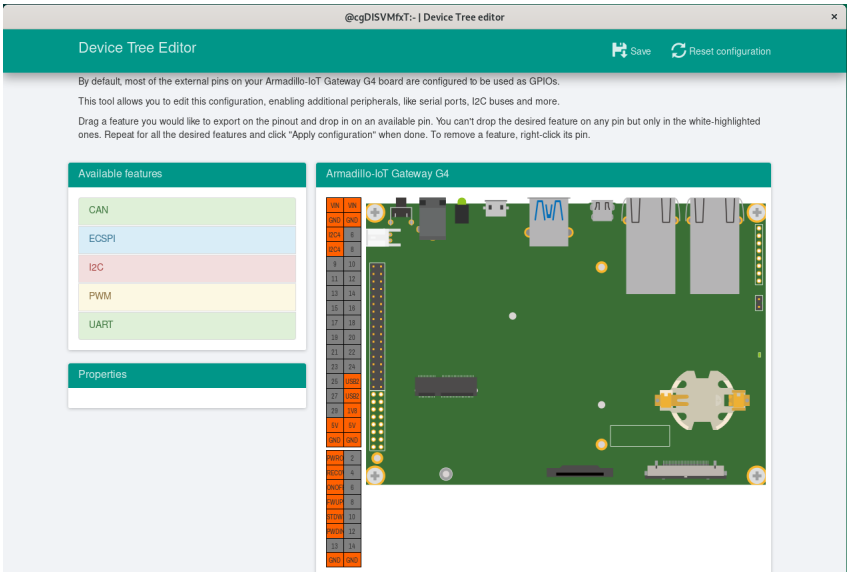


図 3.42 at-dtweb 起動画面

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-IoT Gateway G4」の白色に変化したピンにドロップします。例として CON11 8/10 ピンを UART3(RXD/TXD) に設定します。

何も機能が選択されていないピンには GPIO の機能が割り当てられます。

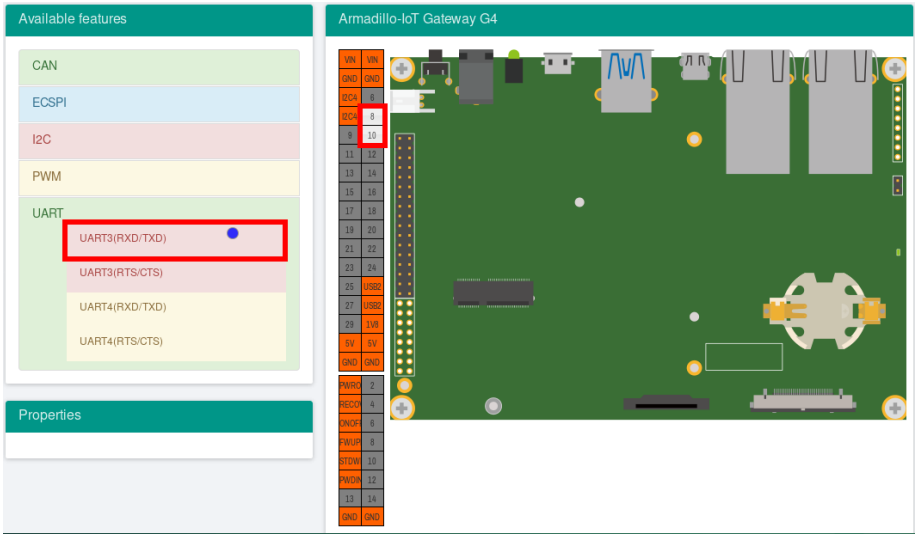


図 3.43 UART3(RXD/TXD) のドラッグ

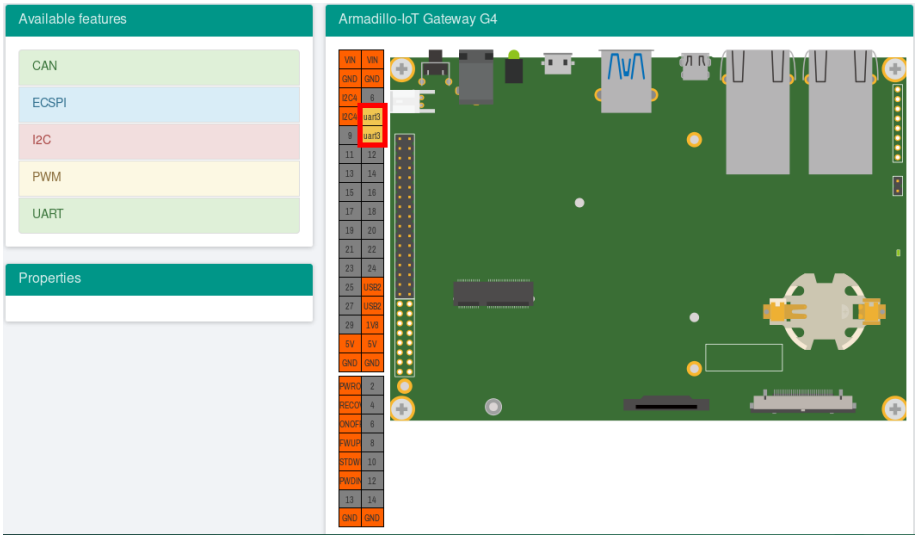


図 3.44 CON11 8/10 ピンへのドロップ

3.5.4.2. 信号名の確認

画面右側の「Armadillo-IoT Gateway G4」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 UART3(RXD/TXD) の信号名を確認します。

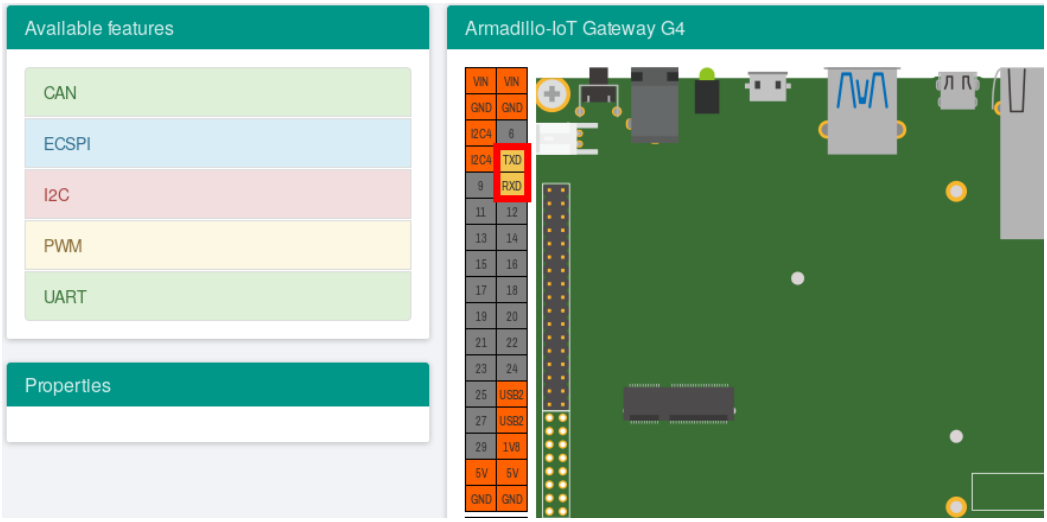


図 3.45 信号名の確認

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-IoT Gateway G4」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 CON11 19/27 ピンの I2C5(SCL/SDA) の clock_frequency プロパティを設定します。

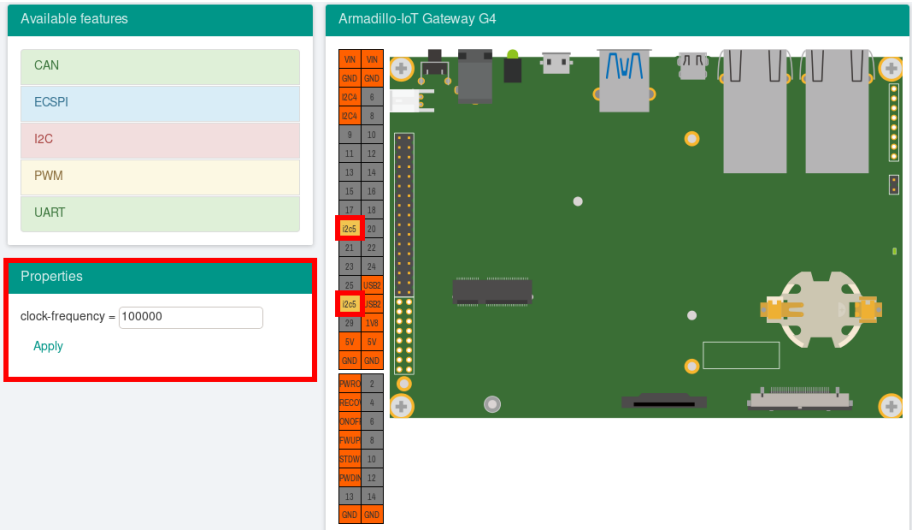


図 3.46 プロパティの設定

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。

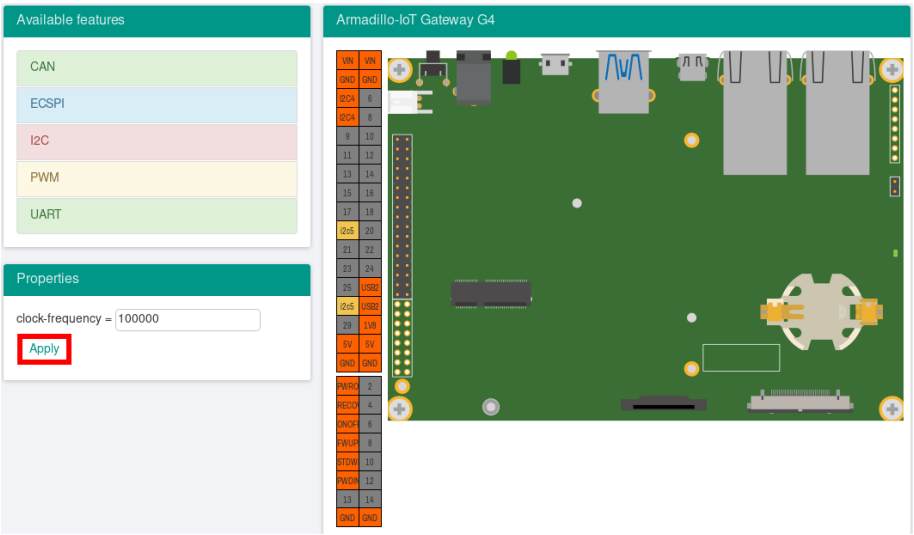


図 3.47 プロパティの保存

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-IoT Gateway G4」のピンを右クリックして「Remove」をクリックします。

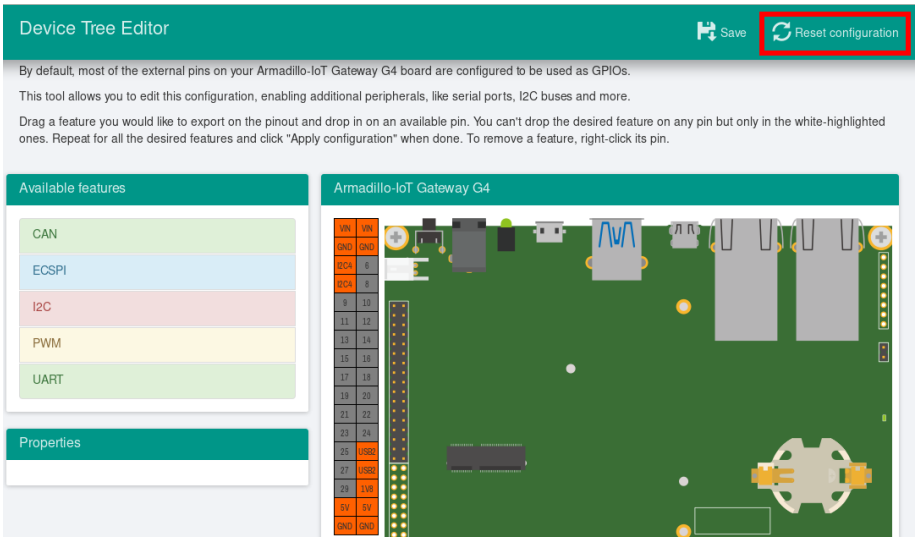


図 3.48 全ての機能の削除

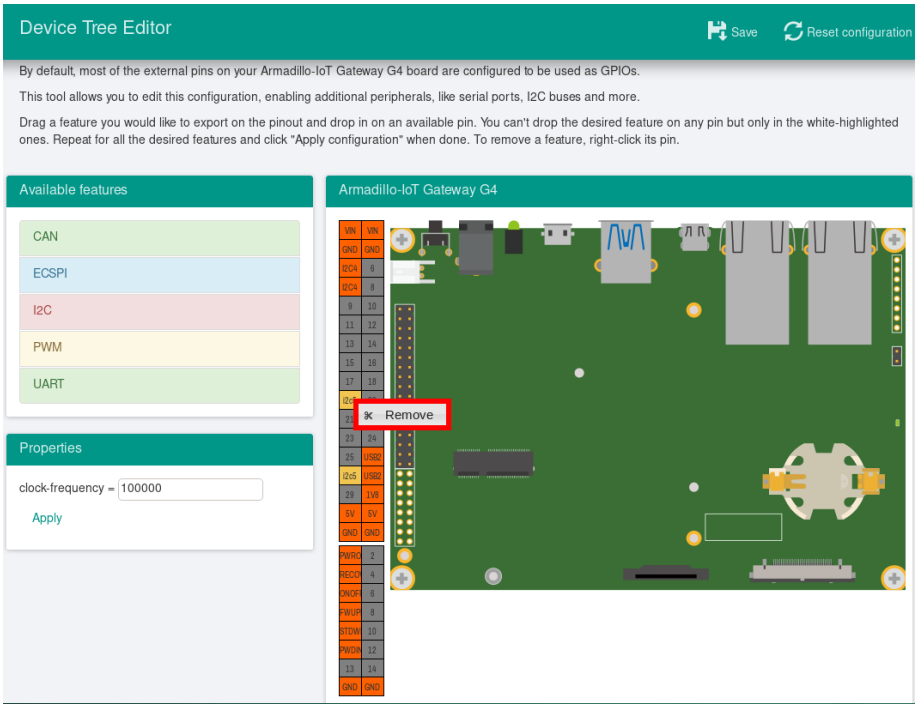


図 3.49 I2C5(SCL/SDA) の削除

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。

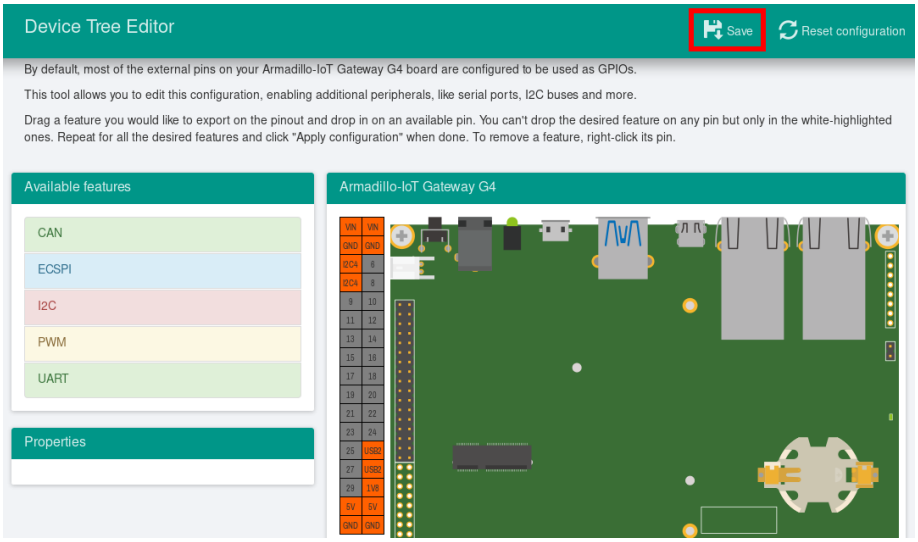


図 3.50 DTS/DTB の生成

at-dtweb のバージョンによって Device Tree のファイル生成の完了画面が異なります。

・ バージョン 2.6.2 未満の場合

以下の画面ようなメッセージが表示されると、DTS および DTB の生成は完了です。

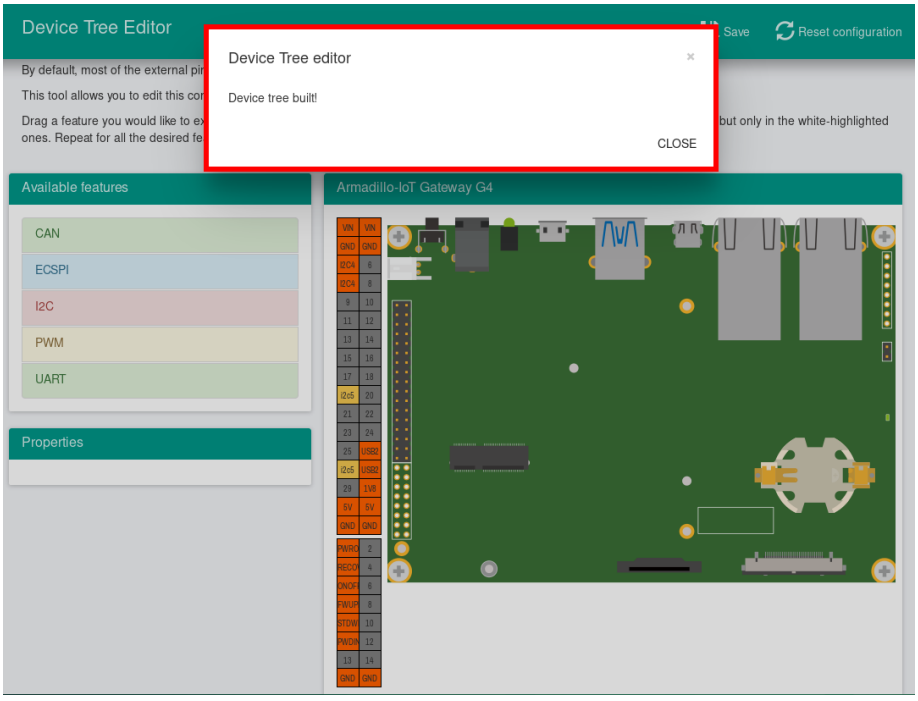


図 3.51 DTS/DTB の生成完了

ビルドが終了すると、arch/arm64/boot/dts/freescale 以下に DTS/DTB が作成されています。

```
[ATDE ~/linux-5.10]$ ls arch/arm64/boot/dts/freescale/armadillo_iotg_g4-expansion-interface.dtsi
arch/arm64/boot/dts/freescale/armadillo_iotg_g4-expansion-interface.dtsi
```

```
[ATDE ~/linux-5.10]$ ls arch/arm64/boot/dts/freescale/armadillo_iotg_g4-at-dtweb.dtb
arch/arm64/boot/dts/freescale/armadillo_iotg_g4-at-dtweb.dtb
```

・バージョン 2.7.0 以降の場合

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。

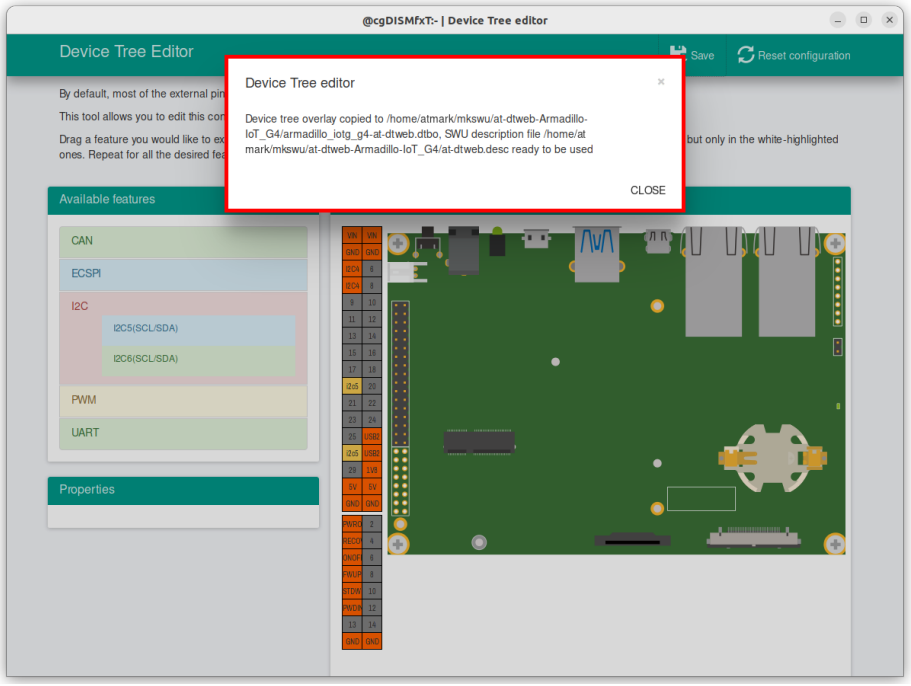


図 3.52 dtbo/desc の生成完了

at-dtweb のバージョン 2.7.0 以降では、ビルドが完了するとホームディレクトリ下の mkswu/at-dtweb-Armadillo-IoT_G4 ディレクトリに、DTB overlays ファイル(dtbo ファイル)と desc ファイルが生成されます。Armadillo-IoT ゲートウェイ G4 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-IoT_G4
armadillo_iotg_g4-at-dtweb.dtbo  at-dtweb.desc.old   update_preserve_files.sh
at-dtweb.desc                    update_overlays.sh
[ATDE ~]$ cd ~/mkswu/at-dtweb-Armadillo-IoT_G4
[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 「3.2.3. 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_iotg_g4-nousb.dtbo armadillo_iotg_g4-sw1-wakeup.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_iotg_g4-nousb.dtbo ❹
Applying fdt overlay: armadillo_iotg_g4-sw1-wakeup.dtbo
:（省略）

[armadillo ~]# cat /sys/firmware/devicetree/base/regulator-usb1-vbus/status; echo
broken ❺
[armadillo ~]# cat /sys/devices/platform/gpio-keys/power/wakeup ❻
enabled
```

図 3.53 /boot/overlays.txt の変更例

❶ /boot/overlays.txt ファイルに「armadillo_iotg_g4-sw1-wakeup.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。

❺ Linux からも「nousb」overlay の確認ができます。USB の regulator を無効にしたため、USB を使えないようになりました。

❻ sw1-wakeup も有効になっていることを確認できます。

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_iotg_g4-nousb.dtbo: USB の電源を切ります。
- armadillo_iotg_g4-sw1-wakeup.dtbo: SW1 の起床要因を有効にします。
- armadillo_iotg_g4-con10-arducam.dtbo: arducam カメラを MIPI CSI-2 で接続する場合にご使用ください。

- armadillo_iotg_g4-con10-imx219.dtbo: Raspberry Pi 向けの imx219 カメラを MIPI CSI-2 で接続する場合にご使用ください。
- armadillo_iotg_g4-con10-ox01f10.dtbo: OMNIVISION の OX01F10 カメラを MIPI CSI-2 で接続する場合にご使用ください。
- armadillo_iotg_g4-lte-ext-board.dtbo: LTE モデルで自動的に使用します。

3.5.5.2. カスタマイズした DT overlay の作成

at-dtweb では対応できない変更を行いたい場合にカスタマイズした DT overlay を作成することでができます。

overlay を使用することで、今後のアップデートで overlay される側の dts に変更があっても自動的に適用され続けます。

1. 「6.25.2. Linux カーネルをビルドする」 を参照して、カーネルのソースコードを取得します。
2. ソースディレクトリの arch/arm64/boot/dts/freescale/armadillo_iotg_g4-customize.dts を編集します。
3. make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- dtbs で DT overlay をビルドします。
4. arch/arm64/boot/dts/freescale/armadillo_iotg_g4-customize.dtbo ファイルを Armadillo の /boot に配置し、/boot/overlays.txt に記載します。

```
[PC ~]$ cd linux-[VERSION] ❶
[PC ~/linux-[VERSION]]$ vim ¥
    arch/arm64/boot/dts/freescale/armadillo_iotg_g4-customize.dts ❷
/dts-v1/;
/plugin/;

#include <dt-bindings/gpio/gpio.h>
#include <dt-bindings/clock/imx8mp-clock.h>
#include <dt-bindings/input/input.h>

#include "imx8mp-pinfunc.h"

&pwm2 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_pwm2>;
        status = "okay";
};

&iomuxc {
        pinctrl_pwm2: pwm2grp {
                fsl,pins = <
                        MX8MP_IOMUXC_SPDIF_RX__PWM2_OUT  0x186
                >;
        };
};
[PC ~/linux-[VERSION]]$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- dtbs ❸
: (省略)
  DTC     arch/arm64/boot/dts/freescale/armadillo_iotg_g4-customize.dtbo
: (省略)
[PC ~/linux-[VERSION]]$ scp ¥
```

```
    arch/arm64/boot/dts/freescale/armadillo_iotg_g4-customize.dtbo ¥
    armadillo:/boot ❹
armadillo_iotg_g4-customize.dtbo          100%  551   207.5KB/s   00:00
[armadillo ~]# cd /boot
[armadillo /boot]# vi /boot/overlays.txt ❺
fdt_overlays=armadillo_iotg_g4-customize.dtbo
[armadillo /boot]# persist_file -vp overlays.txt ¥
                                  armadillo_iotg_g4-customize.dtbo ❻
'/boot/overlays.txt' -> '/mnt/boot/overlays.txt'
'/boot/armadillo_iotg_g4-customize.dtbo' -> '/mnt/boot/armadillo_iotg_g4-customize.dtbo'
Added "/boot/armadillo_iotg_g4-customize.dtbo" to /etc/swupdate_preserve_files
[armadillo /boot]# reboot ❼
:（省略）
Applying fdt overlay: armadillo_iotg_g4-customize.dtbo
```

図 3.54 DT overlay を作成する例

❶ 取得したカーネルのソースディレクトリに入ります。

❷ dts ファイルを編集します。この例では pwm2 を SPDIF_RX (CON9.28) ピンを有効にします。

❸ DT overlay をビルドします。

❹ ビルドされたファイルを Armadillo にコピーします。この例では scp を使いましたが、USB ドライブでのコピーや SWUpdate でも可能です。

❺ overlays.txt にこの DT overlay をロードするように記載します。

❻ ファイルを永続化します。DT overlay は swupdate_preserve_files のデフォルトには記載されていないため、SWUpdate で更新する場合は必ず swupdate_preserve_files も更新してください。

❼ 再起動して、u-boot の出力で DT overlay がロードされてることを確認します。

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 3.55. Armadillo-IoT ゲートウェイ G4 のインターフェース」に Armadillo-IoT ゲートウェイ G4 の各インターフェースの位置を、「表 3.15. Armadillo-IoT ゲートウェイ G4 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。LTE モデルで追加となるインターフェースに関しては、「図 3.56. Armadillo-IoT ゲートウェイ G4 インターフェース LTE モデル追加分」と「表 3.16. Armadillo-IoT ゲートウェイ G4 インターフェース一覧 LTE モデル追加分」を参照してください。

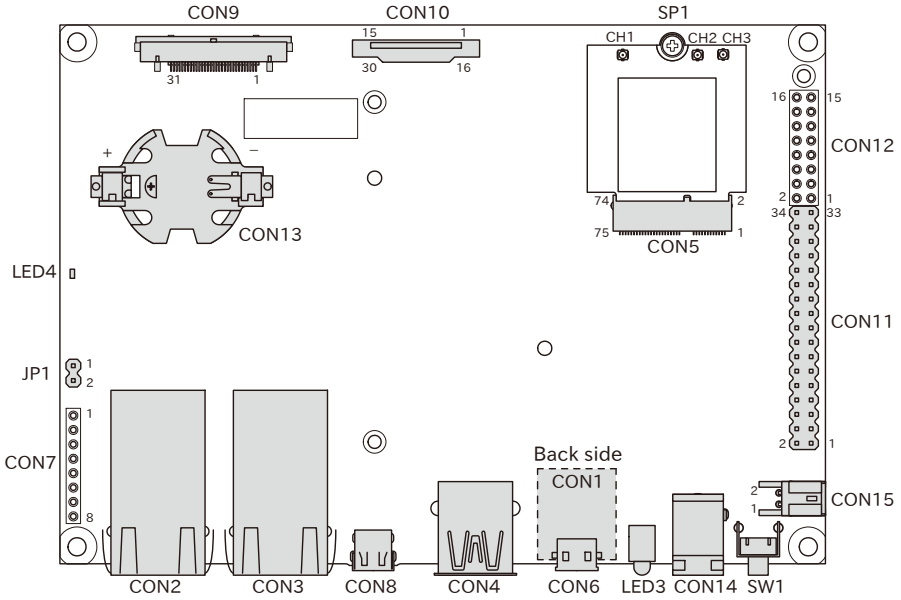


図 3.55 Armadillo-IoT ゲートウェイ G4 のインターフェース

表 3.15 Armadillo-IoT ゲートウェイ G4 インターフェース一覧

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	DM3BT-DSF-PEJS	HIROSE ELECTRIC
CON2	LAN インターフェース 2	56F-1304DYDZ2NL	YUAN DEAN SCIENTIFIC
CON3	LAN インターフェース 1	56F-1304DYDZ2NL	YUAN DEAN SCIENTIFIC
CON4	USB インターフェース	GSB3111311HR	Amphenol ICC
CON5	M.2 インターフェース	SM3ZS067U410AER1000	Japan Aviation Electronics Industry
CON6	USB コンソールインターフェース	UB-MC5BR3-SD204-4S-1-TB NMP	J.S.T.Mfg.
CON7	JTAG インターフェース	A2-8PA-2.54DSA(71)	HIROSE ELECTRIC
CON8	HDMI インターフェース	DC3RX19JA2R1700	Japan Aviation Electronics Industry
CON9	LVDS インターフェース	FX15S-31S-0.5SH	HIROSE ELECTRIC
CON10	MIPI-CSI インターフェース	1-1734248-5	TE Connectivity
CON11	拡張インターフェース 1	61303421121	Wurth Elektronik
CON12	拡張インターフェース 2	61301621121	Wurth Elektronik
CON13	RTC バックアップインターフェース	BU2032SM-FH-GTR	Memory Protection Devices
CON14	電源入力インターフェース 1	PJ-102AH	CUI
CON15	電源入力インターフェース 2	S02B-PASK-2(LF)(SN)	J.S.T.Mfg.
JP1	起動デバイス設定ジャンパ	61300211121	Wurth Elektronik
SW1	ユーザースイッチ	SKHHLUA010	ALPS ELECTRIC
LED3	ユーザー LED	L-710A8CB/1GD	Kingbright Electronic
LED4	電源 LED	SML-D12M8WT86	ROHM
SP1	M.2 用スタッド	SM3ZS067U410-NUT1-R1200	Japan Aviation Electronics Industry

「表 3.15. Armadillo-IoT ゲートウェイ G4 インターフェース一覧」には部品の実装、未実装を問わず、搭載可能な代表型番を記載しています。お手元の製品に搭載されている実際の部品情報につきましては、「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納入仕様書および変更履歴表をご確認ください。

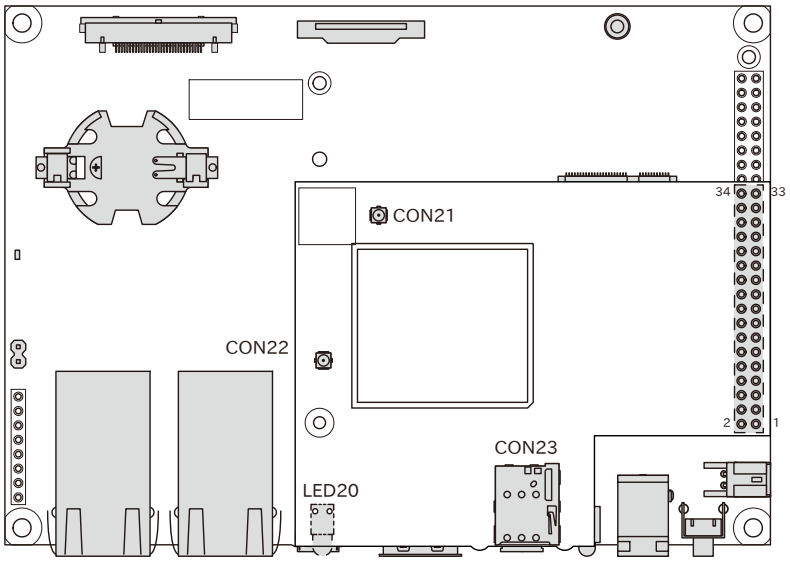


図 3.56 Armadillo-IoT ゲートウェイ G4 インターフェース LTE モデル追加分

表 3.16 Armadillo-IoT ゲートウェイ G4 インターフェース一覧 LTE モデル追加分

部品番号	インターフェース名	型番	メーカー
CON21	3G/LTE アンテナインターフェース 1	U.FL-R-SMT-1	HIROSE ELECTRIC
CON22	3G/LTE アンテナインターフェース 2	U.FL-R-SMT-1	HIROSE ELECTRIC
CON23	nanoSIM インターフェース	SF72S006VBDR2500	Japan Aviation Electronics Industry
LED20	3G/LTE ステータス確認用 LED	L-1384AD/1GD	Kingbright Electronic

「表 3.16. Armadillo-IoT ゲートウェイ G4 インターフェース一覧 LTE モデル追加分」には部品の実装、未実装を問わず、搭載可能な代表型番を記載しています。お手元の製品に搭載されている実際の部品情報につきましては、「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納入仕様書および変更履歴表をご確認ください。

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-IoT ゲートウェイ G4 の SD ホストは、i.MX 8M Plus の uSDHC(Ultra Secured Digital Host Controller)を利用しています。

Armadillo-IoT ゲートウェイ G4 では、SD インターフェース(CON1)が uSDHC2 を利用しています。

機能
- カードタイプ: microSD/microSDHC/microSDXC/microSDIO
- バス幅: 1bit or 4bit
- スピードモード: Default Speed(26MHz), High Speed(52MHz), UHS-I (50MHz)
- カードディテクトサポート

スピードモードが UHS-I モードで動作した場合、VCCI ClassB 規格準拠のため、SD カードの対応スピードが DDR50(最大クロック 50MHz)に制限されます。

インターフェース仕様

CON1 は UHS-I に対応した SD インターフェースです。信号線は i.MX 8M Plus の SD ホストコントローラ(uSDHC2)に接続されています。

表 3.17 CON1 信号配列

ピン番号	ピン名	I/O	説明
1	SD_DAT2	In/Out	SD データバス(bit2)、i.MX 8M Plus の SD2_DATA2 ピンに接続
2	SD_DAT3	In/Out	SD データバス(bit3)、i.MX 8M Plus の SD2_DATA3 ピンに接続
3	SD_CMD	In/Out	SD コマンド/レスポンス、i.MX 8M Plus の SD2_CMD ピンに接続
4	VDD_SD	Power	電源出力(VDD_SD)
5	SD_CLK	Out	SD クロック、i.MX 8M Plus の SD2_CLK ピンに接続
6	GND	Power	電源(GND)
7	SD_DAT0	In/Out	SD データバス(bit0)、i.MX 8M Plus の SD2_DATA0 ピンに接続
8	SD_DAT1	In/Out	SD データバス(bit1)、i.MX 8M Plus の SD2_DATA1 ピンに接続
-	SD_CD	In	SD カード検出、i.MX 8M Plus の SD2_CD_B ピンに接続 (Low: カード挿入、High: カード未挿入)

microSD カードを挿入すると、スロット内部の端子が飛び出します。引っかける等で破損する可能性がございますので、取り扱いにはご注意ください。

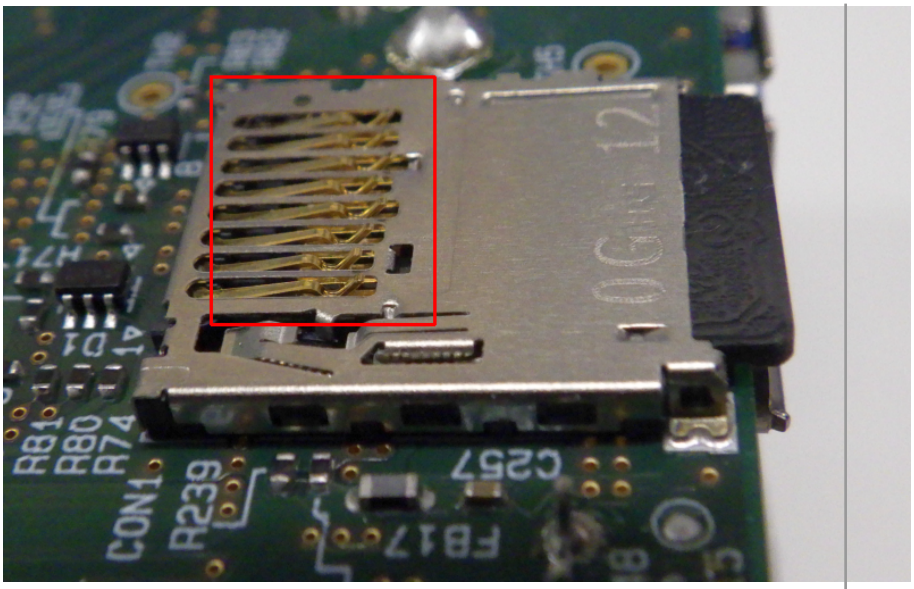


図 3.57 CON1 microSD スロット 取り扱い上の注意事項

3.6.1.2. 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、「6.17. 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-IoT ゲートウェイ G4 の Ethernet(LAN)は、i.MX 8M Plus の ENET(Ethernet MAC)および ENET_QOS(Ethernet Quality Of Service)を利用しています。

Armadillo-IoT ゲートウェイ G4 では、LAN インターフェース 1(CON3)が ENET を、LAN インターフェース 2(CON2)が ENET_QOS 利用しています。

LAN インターフェース 2(CON2)は 10Mbps(10BASE-T)に非対応です。10Mbps で通信を行う場合は、LAN インターフェース 1(CON3)をご利用ください。

機能

- 通信速度： 1000Mbps(1000BASE-T)， 100Mbps(100BASE-TX)，10Mbps(10BASE-T)
- 通信モード: Full-Duplex(全二重), Half-Duplex(半二重) [4]
- Auto Negotiation サポート
- キャリア検知サポート
- リンク検出サポート

インターフェース仕様(CON2)

CON2 は 100BASE-TX/1000BASE-T に対応した LAN インターフェースです。カテゴリ 5e 以上のイーサネット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 Ethenet PHY(KSZ9131RNXI-TR/Microchip Technology)を経由して i.MX 8M Plus の Ethernet Quality Of Service(ENET_QOS)に接続されています。

表 3.18 CON2 信号配列 (10BASE-T/100BASE-TX)

ピン番号	ピン名	I/O	説明
1	LAN2_TX+	In/Out	送信データ(+)
2	LAN2_TX-	In/Out	送信データ(-)
3	LAN2_RX+	In/Out	受信データ(+)
4	-	-	-
5	-	-	-

[4] 1000Mbps(1000BASE-T)は Half-Duplex に非対応です。

ピン番号	ピン名	I/O	説明
6	LAN2_RX-	In/Out	受信データ(-)
7	-	-	-
8	-	-	-

表 3.19 CON2 信号配列 (1000BASE-T)

ピン番号	ピン名	I/O	説明
1	LAN2_TRD0+	In/Out	送受信データ 0(+)
2	LAN2_TRD0-	In/Out	送受信データ 0(-)
3	LAN2_TRD1+	In/Out	送受信データ 1(+)
4	LAN2_TRD2+	In/Out	送受信データ 2(+)
5	LAN2_TRD2-	In/Out	送受信データ 2(-)
6	LAN2_TRD1-	In/Out	送受信データ 1(-)
7	LAN2_TRD3+	In/Out	送受信データ 3(+)
8	LAN2_TRD3-	In/Out	送受信データ 3(-)

表 3.20 CON2 LAN LED の動作

名称	状態	説明
LAN リンクアクティビティ LED	消灯	リンクが確立されていない
	点灯(黄)	リンクが確立されている
	点滅(黄)	リンクが確立されており、データを送受信している
LAN スピード LED	消灯	リンクが確立されていない
	点灯(緑)	100Mbps で接続されている
	点灯(橙)	1000Mbps で接続されている

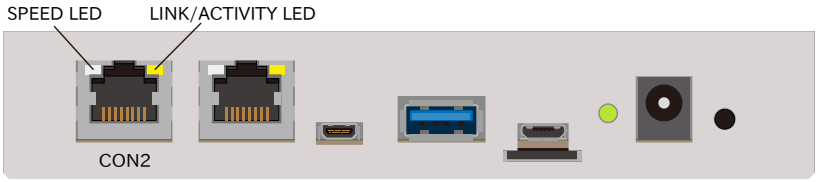


図 3.58 CON2 LAN LED 配置

インターフェース仕様 (CON3)

CON3 は 10BASE-T/100BASE-TX/1000BASE-T に対応した LAN インターフェースです。カテゴリ 5e 以上のイーサネット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 Ethenet PHY(KSZ9131RNXI-TR/Microchip Technology)を経由して i.MX 8M Plus の Ethernet MAC(ENET)に接続されています。

表 3.21 CON3 信号配列 (10BASE-T/100BASE-TX)

ピン番号	ピン名	I/O	説明
1	LAN1_TX+	In/Out	送信データ(+)
2	LAN1_TX-	In/Out	送信データ(-)
3	LAN1_RX+	In/Out	受信データ(+)
4	-	-	-
5	-	-	-
6	LAN1_RX-	In/Out	受信データ(-)
7	-	-	-
8	-	-	-

表 3.22 CON3 信号配列 (1000BASE-T)

ピン番号	ピン名	I/O	説明
1	LAN1_TRD0+	In/Out	送受信データ 0(+)
2	LAN1_TRD0-	In/Out	送受信データ 0(-)
3	LAN1_TRD1+	In/Out	送受信データ 1(+)
4	LAN1_TRD2+	In/Out	送受信データ 2(+)
5	LAN1_TRD2-	In/Out	送受信データ 2(-)
6	LAN1_TRD1-	In/Out	送受信データ 1(-)
7	LAN1_TRD3+	In/Out	送受信データ 3(+)
8	LAN1_TRD3-	In/Out	送受信データ 3(-)

表 3.23 CON3 LAN LED の動作

名称	状態	説明
LAN リンクアクティビティ LED	消灯	リンクが確立されていない
	点灯(黄)	リンクが確立されている
	点滅(黄)	リンクが確立されており、データを送受信している
LAN スピード LED	消灯	10Mbps で接続されている、またはリンクが確立されていない
	点灯(緑)	100Mbps で接続されている
	点灯(橙)	1000Mbps で接続されている

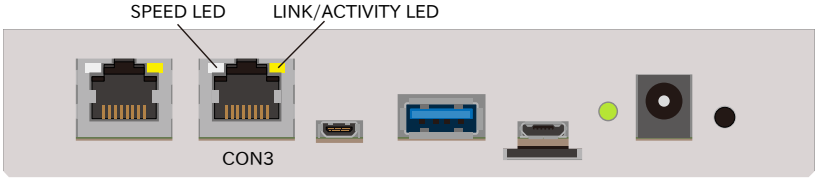


図 3.59 CON3 LAN LED 配置

3.6.2.2. ソフトウェア仕様

ネットワークデバイス
- `eth0` (LAN インターフェース 1)
- `eth1` (LAN インターフェース 2)

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. USB デバイスを使用する

3.6.3.1. ハードウェア仕様

- USB ホスト

  Armadillo-IoT ゲートウェイ G4 の USB ホストは、i.MX 8M Plus の USB(Universal Serial Bus Controller)および USB_PHY(Universal Serial Bus PHY)を利用しています。

  Armadillo-IoT ゲートウェイ G4 では、USB インターフェース(CON4)が USB1 を利用しています。

  機能
  - USB specification rev 3.0 準拠
  - xHCI(eXtensible Host Controller Interface)互換
  - 転送レート: Super-speed(5 Gbps), high-speed(480 Mbps), full-speed(12 Mbps), low-speed(1.5 Mbps)

- USB ハブ

  Armadillo-IoT ゲートウェイ G4 には、Microchip 製 USB2422 が搭載されています。USB2422 は、「3.4.5. CON11、 CON12(拡張インターフェース)」に接続されています。

  機能
  - USB specification rev 2.0 準拠
  - 転送レート: high-speed(480 Mbps), full-speed(12 Mbps), low-speed(1.5 Mbps)

  インターフェース仕様 CON4 は USB 3.0 に対応した USB インターフェースです。信号線は i.MX 8M Plus の USB コントローラ(USB1)に接続されています。

  USB デバイスに供給される電源(USB1_VBUS)は、i.MX 8M Plus の GPIO1_IO09 ピンで制御しており、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
    - Super Speed(5Gbps)
    - High Speed(480Mbps)
    - Full Speed(12Mbps)
    - Low Speed(1.5Mbps)

表 3.24 CON4 信号配列

ピン番号	ピン名	I/O	説明
1	USB1_VBUS	Power	電源出力(USB1_VBUS)
2	USB1_D-	In/Out	USB 2.0 データ(-)、i.MX 8M Plus の USB1_D_N ピンに接続
3	USB1_D+	In/Out	USB 2.0 データ(+)、i.MX 8M Plus の USB1_D_P ピンに接続
4	GND	Power	電源(GND)
5	USB1_SSRX-	In	USB 3.0 受信データ(-)、i.MX 8M Plus の USB1_RX_N ピンに接続
6	USB1_SSRX+	In	USB 3.0 受信データ(+)、i.MX 8M Plus の USB1_RX_P ピンに接続
7	GND	Power	電源(GND)
8	USB1_SSTX-	Out	USB 3.0 送信データ(-)、i.MX 8M Plus の USB1_TX_N ピンに接続
9	USB1_SSTX+	Out	USB 3.0 送信データ(+)、i.MX 8M Plus の USB1_TX_P ピンに接続

3.6.3.2. 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。

3.6.3.3. 使用方法

コンテナ内で動作するアプリケーションから 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 3.60 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 3.61 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 3.62 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 3.63 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 3.64 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 3.65 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 3.66 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 3.67 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。

3.6.4. JTAG デバッガを使用する

CON7 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。

JTAG 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 8M Plus Applications Processor Reference Manual』をご参照ください。

表 3.25 CON7 信号配列

ピン番号	ピン名	I/O	説明	電圧グループ
1	VDD_1V8	Power	電源出力(VDD_1V8)	-
2	PMIC_RST_B	In	PMIC リセット信号[a]、PMIC の PMIC_RST_B ピンに接続、PMIC 内部でプルアップ(NVCC_SNVS_1V8)	NVCC_SNVS_1V8
3	JTAG_TDI	In	テストデータ入力、i.MX 8M Plus の JTAG_TDI ピンに接続、i.MX 8M Plus 内部でプルアップ(VDD_1V8)	VDD_1V8
4	JTAG_TMS	In	テストモード選択、i.MX 8M Plus の JTAG_TMS ピンに接続、i.MX 8M Plus 内部でプルアップ(VDD_1V8)	VDD_1V8
5	JTAG_TCK	In	テストクロック、i.MX 8M Plus の JTAG_TCK ピンに接続、i.MX 8M Plus 内部でプルアップ(VDD_1V8V)	VDD_1V8
6	JTAG_TDO	Out	テストデータ出力、i.MX 8M Plus の JTAG_TDO ピンに接続	VDD_1V8
7	GND	Power	電源(GND)	-
8	JTAG_MOD	In	モード設定、i.MX 8M Plus の JTAG_MOD ピンに接続、基板上で 10kΩ プルダウン	VDD_1V8

[a]PMIC リセット信号の詳細につきましては、「3.4.8.8. リセットシーケンス」をご確認ください。

3.6.5. WLAN を使用する

3.6.5.1. ハードウェア仕様

Armadillo-IoT ゲートウェイ G4 の M.2 インターフェース(CON5)に、AzureWave Technologies 製 AW-XM458 を接続可能です。AW-XM458 の WLAN 機能は、 PCI Express に接続されます。「3.6.7. PCI Express デバイスを使用する」も合わせて参照してください。

機能
- IEEE 802.11a/b/g/n/ac/ax 準拠
- 最大通信速度(2.4GHz): 458 Mbps(理論値)
- 最大通信速度(5GHz): 1.2 Gbps(理論値)
- 動作モード: インフラストラクチャモード(STA/AP), アドホックモード
- チャンネル(2.4GHz): 1-13
- チャンネル(5GHz): 36-48(W52), 52-64(W53), 100-144(W56)

インターフェース仕様

CON5 は WLAN+BT コンボモジュール用の M.2 インターフェースです。極性キーは E タイプです。

M.2 インターフェースに供給される電源(M2_3V3)は、i.MX 8M Plus の SAI1_RXC ピン(GPIO4_IO01)で制御しており、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。

CON5 の信号配列につきましては、公開していません。

LTE+WLAN モデル、WLAN モデルの場合は、WLAN+BT コンボモジュールが接続されます。WLAN+BT コンボモジュールには、アンテナケーブル接続用のレセプタクルが搭載されています。MHF4 のプラグを接続可能で、各レセプタクルの用途は以下のとおりです。

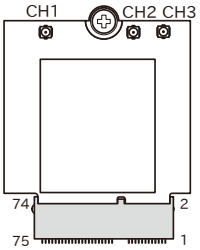


図 3.68 WLAN+BT コンボモジュール アンテナケーブル接続用レセプタクルの配置

表 3.26 WLAN+BT コンボモジュール アンテナケーブル用レセプタクルの用途

インターフェース名	用途
CH1	無線 LAN 用(Main)
CH2	無線 LAN 用(Aux)
CH3	BT 用

3.6.5.2. ソフトウェア仕様

ネットワークデバイス

- `mlan0` (STA)
- `uap0` (AP)

AW-XM458 のファームウェアは、Armadillo-IoT ゲートウェイ G4 にインストールされている linux-firmware-nxpwifi-9098 パッケージに含まれています。

linux-firmware-nxpwifi-9098 の著作権およびライセンス情報については、Armadillo-IoT ゲートウェイ G4 上で/usr/share/licenses/linux-firmware-nxpwifi-9098/LICENSE を参照してください。

3.6.5.3. 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。

3.6.6. BT デバイスを使用する

3.6.6.1. ハードウェア仕様

Armadillo-IoT ゲートウェイ G4 の M.2 インターフェース(CON5)に、AzureWave Technologies 製 AW-XM458 を接続可能です。AW-XM458 の BT 機能は、「3.6.8. UART を使用する」に示す UART1 に接続されます。

AW-XM458 は、Bluetooth® version 5.3 に対応しており、BLE(Bluetooth Low Energy)、EDR(Enhanced Data Rate)が利用できます。

Bluetooth® version 5.0 以降で追加された Coded PHY(Long Range)などの機能は工場出荷状態のソフトウェアで利用することができません。これは、BlueZ が非対応の為です。

3.6.6.2. ソフトウェア仕様

デバイス
- hci0

BT 機能を利用するには、Devicetree Overlay の設定が必要です。

```
[armadillo~]# vi /boot/overlays.txt
fdt_overlays=armadillo_iotg_g4-aw-xm458.dtbo
[armadillo~]# persist_file /boot/overlays.txt
[armadillo~]# reboot
```

3.6.6.3. 使用方法

コンテナ内から Bluetooth を扱うには、コンテナ作成時にホストネットワークを使用するために、NET_ADMIN の権限を渡す必要があります。「図 3.69. Bluetooth を扱うコンテナの作成例」に、alpine イメージから Bluetooth を扱うコンテナを作成する例を示します。

```
[armadillo ~]# vi /etc/atmark/containers/bt_example.conf
set_image docker.io/alpine
set_command sleep infinity
set_network host
add_args --cap-add=NET_ADMIN
[armadillo ~]# podman_start bt_example
Starting 'bt_example'
45fe1eb6b25529f0c84cd4b97ca1aef8451785fc9a87a67d54873c1ed45b70a4
```

図 3.69 Bluetooth を扱うコンテナの作成例

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

```
[armadillo ~]# podman exec -it bt_example sh
[container ~]# apk upgrade
```

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

図 3.70 Bluetooth を起動する実行例

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

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

図 3.71 bluetoothctl コマンドによるスキャンとペアリングの例

❶ コントローラを起動します。

❷ 周辺機器をスキャンします。

❸ ペアリングしたい機器の MAC アドレスを指定してペアリングします。

❹ exit で bluetoothctl のプロンプトを終了します。

3.6.7. PCI Express デバイスを使用する

3.6.7.1. ハードウェア仕様

Armadillo-IoT ゲートウェイ G4 の PCI Express は、i.MX 8M Plus の PCIe(PCI Express)および PCIe_PHY(PCI Express PHY)を利用しています。

機能
- PCI Express Base Specification, Revision 4.0, Version 0.7 準拠
- PCI Local Bus Specification, Revision 3.0 準拠
- PCI Bus Power Management Specification, Revision 1.2 3.0 準拠
- PCI Express Card Electromechanical Specification, Revision 1.1 準拠
- リンク幅: x1

- 転送レート: 8.0GT/s
- 割り込み通知方式: MSI

3.6.8. UART を使用する

3.6.8.1. ハードウェア仕様

Armadillo-IoT ゲートウェイ G4 の UART は、i.MX 8M Plus の UART(Universal Asynchronous Receiver/Transmitter)を利用しています。

Armadillo-IoT ゲートウェイ G4 では、USB シリアル変換 IC(CP2102N/Silicon Labs)経由で UART2 に接続されています。

フォーマット
- データビット長: 7 or 8 ビット
- ストップビット長: 1 or 2 ビット
- パリティ: 偶数 or 奇数 or なし
- フロー制御: CTS/RTS or XON/XOFF or なし
- 最大ボーレート:4Mbps

USB コンソールインターフェース(CON6)は 4Mbps で利用することができません。USB シリアル変換 IC(CP2102N/Silicon Labs)の最大ボーレートが 3Mbps である為です。

拡張インターフェース(CON11)でシリアル(UART)を最大 2 ポート拡張することが可能です。信号線は i.MX 8M Plus の UART(UART3、UART4)に接続されています。

- 信号レベル: VDD_1V8

インターフェース仕様

CON6 は USB コンソール用インターフェースです。

信号線は USB シリアル変換 IC(CP2102N/Silicon Labs)経由で i.MX 8M Plus の UART コントローラ(UART2)に接続されています。

表 3.27 CON6 信号配列

ピン番号	ピン名	I/O	説明
1	VBUS_CNSL	Power	電源入力(VBUS_CNSL)
2	CNSL_USB_D-	In/Out	コンソール用 USB のマイナス側信号、USB シリアル変換 IC に接続
3	CNSL_USB_D+	In/Out	コンソール用 USB のプラス側信号、USB シリアル変換 IC に接続
4	CNSL_USB_ID	-	未接続
5	GND	Power	電源(GND)

3.6.8.2. 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
UART2	/dev/ttymxc1

3.6.8.3. 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/ttymxc0 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/ttymxc0
[armadillo ~]# podman_start serial_example
Starting 'serial_example'
3999f09d51253371cacffd68967c90fdd5250770888a82f59d7810b54fcc873e
```

図 3.72 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/ttymxc0
/dev/ttymxc0, Line 0, UART: undefined, Port: 0x0000, IRQ: 29
        Baud_base: 5000000, close_delay: 50, divisor: 0
        closing_wait: 3000
        Flags: spd_normal
```

図 3.73 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

3.6.9. HDMI を使用する

3.6.9.1. ハードウェア仕様

Armadillo-IoT ゲートウェイ G4 の HDMI は、i.MX 8M Plus の HDMI TX Controller、HDMI TX PHY、HDMI TX BLK_CTRL、HTX_PVI(HDMI TX Parallel Video Interface)および LCDIF(LCD Interface)を利用しています。LCDIF は、LCDIF3 を利用します。

Armadillo-IoT ゲートウェイ G4 は、HDMI 対応ディスプレイへの画像出力及び、音声出力をサポートしています。 Linux では、それぞれ DRM(Direct Rendering Manager)デバイス [5]、ALSA(Advanced Linux Sound Architecture)デバイスとして利用することができます。

機能(画像出力)

- 最大解像度: 4096x2160 ピクセル
- 最大ドットクロック: 297MHz
- カラーフォーマット: RGB888(24bit)

[5]フレームバッファデバイスとして利用することもできます。

・走査方式: プログレッシブ

上記を満していても、画像出力できない場合があります。次の VIC [6] は非対応である為、画像出力できません。

・DAR(Display Aspect Ratio)が 64：27 または 256：135 の VIC

機能(音声出力)

・サンプリング周波数: 32kHz, 44.1kHz, 48kHz, 88.2kHz, 96kHz, 176.4kHz, 192kHz

・チャンネル数: 2

・フォーマット: Signed 24/32 bit, Little-endian

インターフェース仕様

CON8 は HDMI 出力インターフェースです。

信号線は i.MX 8M Plus の HDMI TX コントローラに接続されています。

表 3.28 CON8 信号配列

ピン番号	ピン名	I/O	説明
1	HDMI_HPD	In	ホットプラグ検出、HEAC(-)、i.MX 8M Plus の EARC_N_HPD ピン、HDMI_HPD ピンに接続
2	HDMI_Utility	In/Out	Utility、HEAC(+)、i.MX 8M Plus の EARC_P_UTIL ピンに接続
3	HDMI_TX2+	Out	TMDS データ 2(+)、i.MX 8M Plus の HDMI_TX2_P ピンに接続
4	HDMI_TX2_Shield	-	TMDS データ 2 シールド
5	HDMI_TX2-	Out	TMDS データ 2(-)、i.MX 8M Plus の HDMI_TX2_N ピンに接続
6	HDMI_TX1+	Out	TMDS データ 1(+)、i.MX 8M Plus の HDMI_TX1_P ピンに接続
7	HDMI_TX1_Shield	-	TMDS データ 1 シールド
8	HDMI_TX1-	Out	TMDS データ 1(-)、i.MX 8M Plus の HDMI_TX1_N ピンに接続
9	HDMI_TX0+	Out	TMDS データ 0(+)、i.MX 8M Plus の HDMI_TX0_P ピンに接続
10	HDMI_TX0_Shield	-	TMDS データ 0 シールド
11	HDMI_TX0-	Out	TMDS データ 0(-)、i.MX 8M Plus の HDMI_TX0_N ピンに接続
12	HDMI_TXC+	Out	TMDS クロック(+)、i.MX 8M Plus の HDMI_TXC_P ピンに接続
13	HDMI_TXC_Shield	-	TMDS クロックシールド
14	HDMI_TXC-	Out	TMDS クロック(-)、i.MX 8M Plus の HDMI_TXC_N ピンに接続
15	HDMI_CEC	In/Out	CEC 信号、i.MX 8M Plus の HDMI_CEC ピンに接続
16	HDMI_GND	Power	電源(GND)

[6]CEA-861 規格の VIC コード。

ピン番号	ピン名	I/O	説明
17	HDMI_SCL	In/Out	DDC クロック、i.MX 8M Plus の HDMI_DDC_SCL ピンに接続
18	HDMI_SDA	In/Out	DDC データ、i.MX 8M Plus の HDMI_DDC_SDA ピンに接続
19	5V_HDMI	Power	電源出力(5V_HDMI)

3.6.9.2. ソフトウェア仕様

デバイスファイル
- /dev/dri/card1 (DRM)
- /dev/fb0 (フレームバッファ)
- hw:0 (ALSA)

フレームバッファデバイスは「3.6.11. LVDS を使用する」と共通です。

sysfs DRM クラスディレクトリ
- /sys/class/drm/card1-HDMI-A-1

以下のコマンドを実行することで映像出力の信号を停止することができます。

```
[armadillo ~]# echo 1 > /sys/class/graphics/fb0/blank
```

映像出力を行いたい場合は以下のコマンドを実行します。

```
[armadillo ~]# echo 0 > /sys/class/graphics/fb0/blank
```

3.6.9.3. 使用方法

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。

特別な理由が無い限りは Wayland で画面表示を行うことを推奨しています。

3.6.10. 音声出力を行う

Armadillo-IoT ゲートウェイ G4 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 3.74 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 3.75 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。

3.6.11. LVDS を使用する

3.6.11.1. ハードウェア仕様

Armadillo-IoT ゲートウェイ G4 の LVDS は、i.MX 8M Plus の LDB(LVDS Display Bridge)および LCDIF(LCD Interface)を利用しています。LCDIF は、LCDIF2 を利用します。

Armadillo-IoT ゲートウェイ G4 では、LVDS インターフェース(CON9)が LVDS0 を利用しています。

Linux では、画像出力を DRM(Direct Rendering Manager)デバイス [7]として利用することができます。

機能

- 最大解像度: 1366x768p60
- 最大ピクセルクロック: 80MHz
- レーン数: 4(データ), 1(クロック)
- カラーフォーマット: RGB666(18bit), RGB888(24bit)
- カラーマッピング: JEIDA, VESA

[7]フレームバッファデバイスとして利用することもできます。

インターフェース仕様

CON9 は 1 チャンネル(4 レーン)の LVDS 出力インターフェースです。

信号線は i.MX 8M Plus の LVDS Display Bridge(LDB)に接続されています。

CON11 の 5、7 ピンと CON9 の 7、8 ピンは同じ I2C バス(I2C4)に接続されています。

表 3.29 CON9 搭載コネクタと対向コネクタ例

名称	型番	メーカー	備考
搭載コネクタ	FX15S-31S-0.5SH	HIROSE ELECTRIC	許容電流 0.5A(端子 1 本あたり)
対向コネクタ	FX15S-31P-C	HIROSE ELECTRIC	シェル付きシールド強化タイプ (AWG30～AWG32 対応)
	FX15SW-31P-C	HIROSE ELECTRIC	シェル付きシールド強化タイプ (AWG28～AWG30 対応)
コンタクト	FX15-3032PCFB	HIROSE ELECTRIC	適用電線 AWG30～AWG32
	FX15-2830PCFB	HIROSE ELECTRIC	適用電線 AWG28～AWG30

接触不良や断線を防ぐため、コンタクトは汎用工具ではなく専用工具で圧着することをお勧めします。コンタクトとコネクタの選定前に、工具をご確認ください。

表 3.30 CON9 信号配列

ピン番号	ピン名	I/O	説明	電圧グループ
1	GND	Power	電源(GND)	-
2	VIN	Power	電源出力(VIN)	-
3	VIN	Power	電源出力(VIN)	-
4	VIN	Power	電源出力(VIN)	-
5	VIN	Power	電源出力(VIN)	-
6	GND	Power	電源(GND)	-
7	I2C4_SCL	In/Out	I2C クロック、i.MX 8M Plus の I2C4_SCL ピン、CON11 5 ピンに接続、基板上で 4.7k プルアップ(VDD_1V8)	VDD_1V8
8	I2C4_SDA	In/Out	I2C データ、i.MX 8M Plus の I2C4_SDA ピン、CON11 7 ピンに接続、 基板上で 4.7k プルアップ(VDD_1V8)	VDD_1V8
9	GND	Power	電源(GND)	-
10	LVDS0_TX0N	Out	LVDS データ 0(-)、i.MX 8M Plus の LVDS0_D0_N ピンに接続	-
11	LVDS0_TX0P	Out	LVDS データ 0(+)、i.MX 8M Plus の LVDS0_D0_P ピンに接続	-

ピン番号	ピン名	I/O	説明	電圧グループ
12	GND	Power	電源(GND)	-
13	LVDS0_TX1N	Out	LVDS データ 1(-)、i.MX 8M Plus の LVDS0_D1_N ピンに接続	-
14	LVDS0_TX1P	Out	LVDS データ 1(+)、i.MX 8M Plus の LVDS0_D1_P ピンに接続	-
15	GND	Power	電源(GND)	-
16	LVDS0_CLKN	Out	LVDS クロック(-)、i.MX 8M Plus の LVDS0_CLK_N ピンに接続	-
17	LVDS0_CLKP	Out	LVDS クロック(+)、i.MX 8M Plus の LVDS0_CLK_P ピンに接続	-
18	GND	Power	電源(GND)	-
19	LVDS0_TX2N	Out	LVDS データ 2(-)、i.MX 8M Plus の LVDS0_D2_N ピンに接続	-
20	LVDS0_TX2P	Out	LVDS データ 2(+)、i.MX 8M Plus の LVDS0_D2_P ピンに接続	-
21	GND	Power	電源(GND)	-
22	LVDS0_TX3N	Out	LVDS データ 3(-)、i.MX 8M Plus の LVDS0_D3_N ピンに接続	-
23	LVDS0_TX3P	Out	LVDS データ 3(+)、i.MX 8M Plus の LVDS0_D3_P ピンに接続	-
24	GND	Power	電源(GND)	-
25	GPIO3_IO00	In/Out	拡張入出力、i.MX 8M Plus の NAND_ALE ピンに接続	VDD_1V8
26	GPIO3_IO01	In/Out	拡張入出力、i.MX 8M Plus の NAND_CE0_B ピンに接続	VDD_1V8
27	GPIO5_IO03	In/Out	拡張入出力、i.MX 8M Plus の SPDIF_TX ピンに接続	VDD_1V8
28	GPIO5_IO04	In/Out	拡張入出力、i.MX 8M Plus の SPDIF_RX ピンに接続	VDD_1V8
29	GND	Power	電源(GND)	-
30	VDD_1V8	Power	電源出力(VDD_1V8)	-
31	GND	Power	電源(GND)	-

3.6.11.2. ソフトウェア仕様

デバイスファイル
- /dev/dri/card1 (DRM)
- /dev/fb0 (フレームバッファ)

フレームバッファデバイスは「3.6.9. HDMI を使用する」と共通です。

sysfs DRM クラスディレクトリ
- /sys/class/drm/card1-LVDS-1

3.6.11.3. 使用方法

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。

特別な理由が無い限りは Wayland で画面表示を行うことを推奨しています。

3.6.12. MIPI CSI-2 カメラを使用する

3.6.12.1. 動作確認済み MIPI CSI-2 カメラデバイス

「表 3.31. 動作確認済み MIPI CSI-2 カメラデバイス一覧」に、アットマークテクノで動作確認済みの MIPI CSI-2 カメラデバイスの一覧を示します。

表 3.31 動作確認済み MIPI CSI-2 カメラデバイス一覧

型番・名称	メーカー	リンク
KBCR-S08MM	シキノハイテック	https://armadillo.atmark-techno.com/node/10889
Arducam 2MP*2 Stereo Camera	Arducam	https://www.arducam.com/product/arducam-2mp-stereo-camera-for-raspberry-pi-nvidia-jetson-nano-xavier-nx-dual-ov2311-monochrome-global-shutter-camera-module/

3.6.12.2. ハードウェア仕様

Armadillo-IoT ゲートウェイ G4 の MIPI CSI-2 は、i.MX 8M Plus の MIPI_CSI(MIPI CSI Host Controller)を利用しています。

Armadillo-IoT ゲートウェイ G4 では、MIPI-CSI インターフェース(CON10)が MIPI_CSI1 を利用しています。

Linux では、カメラ [8] からの画像入力を V4L2(Video4Linux2)デバイスとして利用することができます。

機能
- MIPI D-PHY specification V1.2 準拠
- MIPI CSI2 Specification V1.3 準拠(C-PHY feature を除く)
- レーン数: 2(データ), 1(クロック)
- 最大ピクセルクロック: 400MHz
- データレート: 80Mbps - 1.5Gbps(1 レーンあたり)
- カラーフォーマット(YUV): YUV420 8/10bit, YUV420 8bit Legacy, YUV420 8/10bit CSPS, YUV422 8/10bit
- カラーフォーマット(RGB): RGB565, RGB666, RGB888
- カラーフォーマット(RAW): RAW6, RAW7, RAW8, RAW10, RAW12, RAW14

インターフェース仕様: CON10 はカメラ接続用の 1 チャンネル(2 レーン)の MIPI-CSI インターフェースです。

[8] Armadillo-IoT ゲートウェイ G4 にカメラは付属していません。

信号線は i.MX 8M Plus の MIPI Camera Serial Interface(MIPI CSI1)に接続されています。

表 3.32 CON10 搭載コネクタとフレキシブルフラットケーブル例

名称	型番	メーカー	備考
搭載コネクタ	1-1734248-5	TE Connectivity	許容電流 1A(端子 1 本あたり)

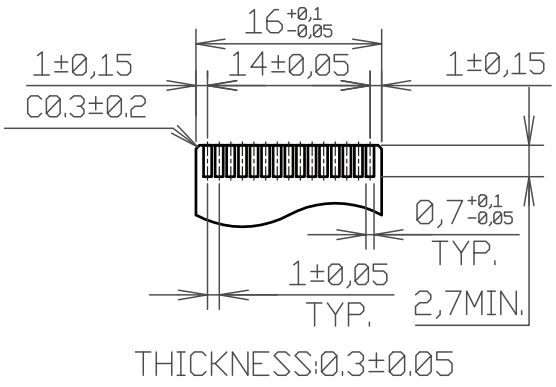


図 3.76 CON10 接続可能なフレキシブルフラットケーブルの形状

表 3.33 CON10 信号配列

ピン番号	ピン名	I/O	説明	電圧グループ
1	GND	Power	電源(GND)	-
2	CSI1_DN_0	In	MIPI データ 0(-)、i.MX 8M Plus の MIPI_CSI1_D0_N ピンに接続、 17 ピンと共通	-
3	CSI1_DP_0	In	MIPI データ 0(+)、i.MX 8M Plus の MIPI_CSI1_D0_P ピンに接続、 18 ピンと共通	-
4	GND	Power	電源(GND)	-
5	CSI1_DN_1	In	MIPI データ 1(-)、i.MX 8M Plus の MIPI_CSI1_D1_N ピンに接続、 20 ピンと共通	-
6	CSI1_DP_1	In	MIPI データ 1(+)、i.MX 8M Plus の MIPI_CSI1_D1_P ピンに接続、 21 ピンと共通	-
7	GND	Power	電源(GND)	-
8	CSI1_CK_N	In	MIPI クロック(-)、i.MX 8M Plus の MIPI_CSI1_CLK_N ピンに接続、 23 ピンと共通	-
9	CSI1_CK_P	In	MIPI クロック(+)、i.MX 8M Plus の MIPI_CSI1_CLK_P ピンに接続、 24 ピンと共通	-
10	GND	Power	電源(GND)	-
11	CSI1_GPIO0_3V3	In/Out	拡張入出力、レベル変換 IC を経由して i.MX 8M Plus の NAND_DATA00 ピンに接続、 26 ピンと共通	VEXT_3V3

ピン番号	ピン名	I/O	説明	電圧グループ
12	CSI1_GPIO1_3V3	In/Out	拡張入出力、レベル変換 IC を経由して i.MX 8M Plus の NAND_DATA01 ピンに接続、 27 ピンと共通	VEXT_3V3
13	I2C2_SCL_3V3	Out	I2C クロック、レベル変換 IC を経由して i.MX 8M Plus の I2C2_SCL ピンに接続、基板上で 4.7k プルアップ(VEXT_3V3)、 28 ピンと共通	VEXT_3V3
14	I2C2_SDA_3V3	In/Out	I2C データ、レベル変換 IC を経由して i.MX 8M Plus の I2C2_SDA ピンに接続、基板上で 4.7k プルアップ(VEXT_3V3)、 29 ピンと共通	VEXT_3V3
15	VEXT_3V3	Power	電源出力(VEXT_3V3)	-
16	GND	Power	電源(GND)	-
17	CSI1_DN_0	In	MIPI データ 0(-)、i.MX 8M Plus の MIPI_CSI1_D0_N ピンに接続、 2 ピンと共通	-
18	CSI1_DP_0	In	MIPI データ 0(+)、i.MX 8M Plus の MIPI_CSI1_D0_P ピンに接続、 3 ピンと共通	-
19	GND	Power	電源(GND)	-
20	CSI1_DN_1	In	MIPI データ 1(-)、i.MX 8M Plus の MIPI_CSI1_D1_N ピンに接続、 5 ピンと共通	-
21	CSI1_DP_1	In	MIPI データ 1(+)、i.MX 8M Plus の MIPI_CSI1_D1_P ピンに接続、 6 ピンと共通	-
22	GND	Power	電源(GND)	-
23	CSI1_CK_N	In	MIPI クロック(-)、i.MX 8M Plus の MIPI_CSI1_CLK_N ピンに接続、 8 ピンと共通	-
24	CSI1_CK_P	In	MIPI クロック(+)、i.MX 8M Plus の MIPI_CSI1_CLK_P ピンに接続、 9 ピンと共通	-
25	GND	Power	電源(GND)	-
26	CSI1_GPIO0_3V3	In/Out	拡張入出力、レベル変換 IC を経由して i.MX 8M Plus の NAND_DATA00 ピンに接続、基板上で 4.7k プルアップ(VEXT_3V3)、 11 ピンと共通	VEXT_3V3
27	CSI1_GPIO1_3V3	In/Out	拡張入出力、レベル変換 IC を経由して i.MX 8M Plus の NAND_DATA01 ピンに接続、基板上で 4.7k プルアップ(VEXT_3V3)、 12 ピンと共通	VEXT_3V3
28	I2C2_SCL_3V3	Out	I2C クロック、レベル変換 IC を経由して i.MX 8M Plus の I2C2_SCL ピンに接続、基板上で 4.7k プルアップ(VEXT_3V3)、 13 ピンと共通	VEXT_3V3
29	I2C2_SDA_3V3	In/Out	I2C データ、レベル変換 IC を経由して i.MX 8M Plus の I2C2_SDA ピンに接続、基板上で 4.7k プルアップ(VEXT_3V3)、 14 ピンと共通	VEXT_3V3
30	VEXT_3V3	Power	電源(VEXT_3V3)	-

3.6.12.3. ソフトウェア仕様

デバイスファイル
- /dev/video2 [9]

3.6.12.4. 使用方法

MIPI CSI-2 カメラを使用する場合、 Device Tree の変更が必要です。

Armadillo-IoT ゲートウェイ G4 では標準で以下の MIPI CSI-2 カメラの DT overlay を提供しています。

DT overlay の使用方法については「3.5.5. DT overlay によるカスタマイズ」を参照してください。

表 3.34 MIPI CSI-2 カメラ用の DT overlay

DT overlay ファイル	対応デバイス例	説明
armadillo_iotg_g4-con10-arducam.dtbo	Arducam 2MP*2 Stereo Camera	arducam カメラを MIPI CSI-2 で接続する場合にご使用ください。
armadillo_iotg_g4-con10-imx219.dtbo	-	Raspberry Pi 向けの imx219 カメラを MIPI CSI-2 で接続する場合にご使用ください。
armadillo_iotg_g4-con10-ox01f10.dtbo	KBCR-S08MM	OMNIVISION の OX01F10 カメラを MIPI CSI-2 で接続する場合にご使用ください。

3.6.13. GPIO を制御する

3.6.13.1. ハードウェア仕様

GPIO は、i.MX 8M Plus の GPIO(General Purpose Input/Output)および、ON Semiconductor 製 FXL6408(GPIO エキスパンダー)を利用しています。

拡張インターフェース(CON11、CON12)で GPIO を最大 34 ボート拡張することが可能です。拡張インターフェースの仕様については「3.4.5. CON11、 CON12(拡張インターフェース)」を参照してください。

- 信号レベル : VDD_1V8

3.6.13.2. ソフトウェア仕様

デバイスファイル

デバイスファイル	GPIO 番号
/dev/gpiochip0	0～31(GPIO1_IO00～GPIO1_IO31)
/dev/gpiochip1	32～63(GPIO2_IO00～GPIO2_IO31)
/dev/gpiochip2	64～95(GPIO3_IO00～GPIO3_IO31)
/dev/gpiochip3	96～127(GPIO4_IO00～GPIO4_IO31)
/dev/gpiochip4	128～159(GPIO5_IO00～GPIO5_IO31)
/dev/gpiochip5	504～511 [a] (FXL6408)

[a]GPIO エキスパンダーを追加した場合は、番号が異なる可能性があります。

sysfs GPIO クラスディレクトリ
- /sys/class/gpio/

[9]UVC カメラなどを接続して V4L2 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。

sysfs GPIO クラスは旧バージョンの Linux カーネルとの互換性維持の為に残っています。新しくアプリケーションを開発する際の利用はおすすめしません。

新しくアプリケーションを開発する場合は、libgpiod パッケージに含まれるアプリケーションまたは Linux カーネルのソースコードに含まれているサンプル(tools/gpio/)を参考にしてください。

3.6.13.3. 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/gpiochip2 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/gpiochip2
[armadillo ~]# podman_start gpio_example
Starting 'gpio_example'
956a0fecc48d5ea1210069910f7bb48b9e90b2dadb12895064d9776dae0360b5
```

図 3.77 GPIO を扱うためのコンテナ作成例

コンテナ内に入ってコマンドで GPIO を操作する例を以下に示します。この例では GPIO3_IO21 を操作しています。

```
[armadillo ~]# podman exec -it gpio_example sh
[container ~]# apk upgrade
[container ~]# apk add libgpiod
[container ~]# gpioget gpiochip2 21 ❶
0 ❷
[container ~]# gpioset gpiochip2 21=1 ❸
```

図 3.78 コンテナ内からコマンドで GPIO を操作する例

❶ GPIO 番号 21 の値を取得します。

❷ 取得した値を表示します。

❸ GPIO 番号 21 に 1(High) 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/gpiochip2 が認識されていることが確認できます。

```
[container ~]# gpiodetect
gpiochip2 [30220000.gpio] (32 lines)
```

図 3.79 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 gpiochip2
gpiochip2 - 32 lines:
        line   0:      unnamed         "?"  output  active-high [used]
        line   1:      unnamed       unused   input  active-High
        line   2:      unnamed       unused   input  active-high
        line   3:      unnamed       unused   input  active-high
        line   4:      unnamed       unused   input  active-high
        line   5:      unnamed       unused   input  active-high
        line   6:      unnamed       unused   input  active-high
        line   7:      unnamed       unused   input  active-high
        line   8:      unnamed       unused   input  active-high
        line   9:      unnamed       unused   input  active-high
        line  10:      unnamed       unused   input  active-high
        line  11:      unnamed       unused   input  active-high
        line  12:      unnamed       unused   input  active-high
        line  13:      unnamed       unused   input  active-high
        line  14:      unnamed       unused   input  active-high
        line  15:      unnamed       unused   input  active-high
        line  16:      unnamed       unused   input  active-high
        line  17:      unnamed       unused   input  active-high
        line  18:      unnamed       unused   input  active-high
        line  19:      unnamed       unused   input  active-high
        line  20:      unnamed       unused   input  active-high
        line  21:      unnamed       unused   input  active-high
        line  22:      unnamed       unused   input  active-high
        line  23:      unnamed       unused   input  active-high
        line  24:      unnamed       unused   input  active-high
        line  25:      unnamed       unused   input  active-high
        line  26:      unnamed       unused   input  active-high
        line  27:      unnamed       unused   input  active-high
        line  28:      unnamed       unused   input  active-high
        line  29:      unnamed       unused   input  active-high
        line  30:      unnamed       unused   input  active-high
        line  31:      unnamed       unused   input  active-high
```

図 3.80 gpioinfo コマンドの実行

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。

3.6.14. I2C デバイスを使用する

3.6.14.1. ハードウェア仕様

I2C インターフェースは、i.MX 8M Plus の I2C(I2C Controller)を利用しています。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。

主に拡張インターフェース(CON11、CON12)で I2C を最大 3 ポート拡張することが可能です。拡張インターフェースの仕様については「3.4.5. CON11、 CON12(拡張インターフェース)」を参照してください。信号線は i.MX 8M Plus の I2C コントローラ(I2C4、I2C5、I2C6)に接続されています。

- 最大データ転送レート: 384kbps
- 信号レベル: VDD_1V8

Armadillo-IoT ゲートウェイ G4 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 3.35 I2C デバイス

I2C バス	I2C デバイス	
	アドレス	デバイス名
0(I2C1)	0x25	PCA9450(PMIC)
1(I2C2)	0x2c	USB2422(USB ハブ)
	0x32	RV-8803-C7(RTC)
2(I2C3)	0x43	FXL6408(GPIO エキスパンダー)
	0x48	SE050(セキュアエレメント)
3(I2C4)	接続デバイス無し	

3.6.14.2. ソフトウェア仕様

Armadillo-IoT ゲートウェイ G4 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
- 最大クロック: 384kHz

デバイスファイル
- /dev/i2c-0 (I2C1)
- /dev/i2c-1 (I2C2)
- /dev/i2c-2 (I2C3)
- /dev/i2c-3 (I2C4)

/dev/i2c-6 は、HDMI DDC です。

3.6.14.3. 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 3.81 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: -- -- -- -- -- -- -- -- -- -- -- -- -- UU -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- UU -- -- --
50: UU -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- 72 -- -- -- -- --
```

図 3.82 i2cdetect コマンドによる確認例

3.6.15. SPI デバイスを使用する

3.6.15.1. ハードウェア仕様

拡張インターフェース(CON11)で SPI を最大 2 ボート拡張することが可能です。拡張インターフェースの仕様については「3.4.5. CON11、 CON12(拡張インターフェース)」を参照してください。

- 最大クロック周波数: 66MHz(リード)/23MHz(ライト)
- 信号レベル : VDD_1V8

3.6.15.2. 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 3.83 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 3.84 spi-config コマンドによる確認例

3.6.16. CAN デバイスを使用する

3.6.16.1. ハードウェア仕様

拡張インターフェース(CON11)で CAN を最大 2 ポート拡張することが可能です。拡張インターフェースの仕様については「3.4.5. CON11、 CON12(拡張インターフェース)」を参照してください。信号線は i.MX 8M Plus の FLEXCAN(FLEXCAN1、FLEXCAN2)に接続されています。

- CAN FD、CAN 2.0B プロトコル対応
- 信号レベル: VDD_1V8

3.6.16.2. 使用方法

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

```
[armadillo ~]# vi /etc/atmark/containers/can_example.conf
set_image dockage.io/alpine
set_command sleep infinity
set_network host
add_args --cap-add=NET_ADMIN
[armadillo ~]# podman_start can_example
Starting 'can_example'
73e7dbce86e84eef337bbc5c580a747948b94b87015bb34143da341b8301c16a
```

図 3.85 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 3.86 CAN の設定例

❶ CAN の設定のために必要な iproute2 をインストールします。すでにインストール済みの場合は不要です。

❷ CAN の通信速度を 125000 kbps に設定します。

❸ can0 インターフェースを起動します。

❹ can0 インターフェースの現在の使用状況を表示します。

3.6.17. PWM を使用する

3.6.17.1. ハードウェア仕様

拡張インターフェース(CON11)で PWM を最大 4 ポート拡張することが可能です。拡張インターフェースの仕様については「3.4.5. CON11、 CON12(拡張インターフェース)」を参照してください。

- 最大周波数: 66MHz
- 信号レベル: VDD_1V8

3.6.17.2. 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 3.87 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/pwmchip2 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/pwmchip2/export ❶
[container ~]# echo 1000000000 > /sys/class/pwm/pwmchip2/pwm0/period ❷
```

```
[container ~]# echo 500000000 > /sys/class/pwm/pwmchip2/pwm0/duty_cycle ❸
[container ~]# echo 1 > /sys/class/pwm/pwmchip2/pwm0/enable ❹
```

図 3.88 PWM の動作設定例

❶ pwmchip2 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。

3.6.18. I2S(SAI) を使用する

3.6.18.1. ハードウェア仕様

I2S を最大 1 ポート拡張することが可能です。信号線は i.MX 8M Plus の同期式オーディオインターフェース(SAI3)に接続されています。

・ 信号レベル: VDD_1V8

3.6.18.2. 使用方法

Armadillo サイトの Howto にて I2S を使用する例を公開していますので、そちらを参照してください。

Armadillo サイト - Howto I2S 接続のオーディオコーデックを使用する方法 (Armadillo-IoT ゲートウェイ G4/X2)

https://armadillo.atmark-techno.com/index.php/howto/aiotg4-i2s-tlv320aic3110

3.6.19. PDM マイクを使用する

3.6.19.1. ハードウェア仕様

L と R が対になった PDM MIC を最大 4 ポート拡張することが可能です。信号線は i.MX 8M Plus の PDM マイクロフォンインターフェース(MICFIL)に接続されています。

・ 信号レベル: VDD_1V8

3.6.19.2. 使用方法

Armadillo サイトの Howto にて PDM マイクを使用する例を公開していますので、そちらを参照してください。

Armadillo サイト - Howto PDM マイクを使用する方法 (Armadillo-IoT ゲートウェイ G4/X2)

https://armadillo.atmark-techno.com/index.php/howto/aiotg4-use-pdm-mic

3.6.20. RTC を使用する

3.6.20.1. ハードウェア仕様

Armadillo-IoT ゲートウェイ G4 のリアルタイムクロックは、Armadillo-IoT ゲートウェイ G4 に搭載された Micro Crystal 製 RV-8803-C7 および、i.MX 8M Plus の SNVS_HP Real Time Counter を利用しています。

機能
- アラーム割り込みサポート

インターフェース仕様

CON13 は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。長時間電源が切断されても時刻データを保持させたい場合にご使用ください。

CON13 には CR2032、BR2032 等の電池を接続することができます。リアルタイムクロックの時刻保持時の平均消費電流は、データシート上、240nA(Typ.)となっておりますので、電池寿命までの時刻保持が期待できます。

温度補償タイプのリアルタイムクロックを実装しており、平均月差は周囲温度-20℃～70℃で 8 秒(参考値)です。

表 3.36 CON13 信号配列

ピン番号	ピン名	I/O	説明
1	RTC_BAT	Power	リアルタイムクロックのバックアップ用電源入力(RTC_BAT)
2	GND	Power	電源(GND)

電池をホルダーへ装着する際は、異物の挟み込みや不完全な装着がないように、目視での異物確認や装着状態の確認を行ってください。

3.6.20.2. ソフトウェア仕様

デバイスファイル
- `/dev/rtc` (`/dev/rtc0` へのシンボリックリンク)
- `/dev/rtc0` (RV-8803-C7)
- `/dev/rtc1` (SNVS_HP Real Time Counter)

RV-8803-C7 が `/dev/rtc0` 、SNVS_HP Real Time Counter が `/dev/rtc1` となるよう、Device Tree でエイリアスを設定しています。

Linux カーネルのバージョン v5.10.86-r0 以降では、NTP サーバーと RTC を時刻同期した場合、rtc0 (RV-8803-C7)にのみ時刻が保存されます。

Linux カーネルのバージョン v5.10.52-r1 では、NTP サーバーと RTC を時刻同期した場合、rtc0 (RV-8803-C7)と rtc1 (SVNS) の両方に時刻が保存されていました。

RV-8803-C7 は、毎分 0 秒にしかアラーム割り込みを発生させることができません。0 時 0 分 30 秒の時に、1 秒後にアラームが鳴るように設定しても、実際にアラーム割り込みが発生するのは 0 時 1 分 0 秒となります。

アラーム割り込みは、デバイスファイル経由で利用することができます。

詳細な情報については、Linux カーネルのソースコードに含まれているドキュメント(Documentation/admin-guide/rtc.rst)やサンプルプログラム(tools/testing/selftests/rtc/rtctest.c)を参照してください。

3.6.20.3. 使用方法

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

```
[armadillo ~]# vi /etc/atmark/containers/rtc_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_args --cap-add=SYS_TIME
add_devices /dev/rtc0
[armadillo ~]# podman_start rtc_example
Starting 'rtc_example'
025209e0d96f43c2911239a8397b7002c3eaab057e031d8abb765df5707d75bd
```

図 3.89 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 3.90 hwclock コマンドによる RTC の時刻表示と設定例

❶ RTC に設定されている現在時刻を表示します。

❷ システム時刻を 2021 年 4 月 1 日 9 時 0 分 0 秒に設定します。

❸ システム時刻を RTC に反映させます。

❹ RTC に設定されている時刻が変更されていることを確認します。

3.6.21. 電源を入力する

3.6.21.1. ハードウェア仕様

CON14、CON15 は電源入力用のインターフェースです。

インターフェース仕様 (CON14)

CON14 には DC ジャックが実装されており、「図 3.91. 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。対応プラグは内径 2.1mm、外形 5.5mm のものとなります。

図 3.91 AC アダプタの極性マーク

インターフェース仕様 (CON15)

CON15 には 2mm ピッチのライトアングルコネクタを実装しています。

表 3.37 CON15 搭載コネクタと対向コネクタ例

名称	型番	メーカー	備考
搭載コネクタ	S02B-PASK-2(LF)(SN)	J.S.T.Mfg.	許容電流 3A(端子 1 本あたり)
対向コネクタ	PAP-02V-S	J.S.T.Mfg.	-
コンタクト	SPHD-001T-P0.5	J.S.T.Mfg.	適用電線 AWG26～AWG22
	SPHD-002T-P0.5	J.S.T.Mfg.	適用電線 AWG28～AWG24

表 3.38 CON15 信号配列

ピン番号	ピン名	I/O	説明
1	VIN	Power	電源入力(VIN)
2	GND	Power	電源(GND)

CON11、CON14、CON15 の電源ライン(VIN)は接続されており、同時に電源を供給することはできません。

AC アダプタを使用する際は、AC アダプタの DC プラグを Armadillo-IoT ゲートウェイ G4 に接続してから AC プラグをコンセントに挿してください。

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。

3.6.22. 起動デバイスを変更する

3.6.22.1. ハードウェア仕様

機能

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。

表 3.39 ジャンパの状態と起動デバイス

JP1 の状態	起動デバイス
オープン	eMMC
ショート	microSD(CON1)

インターフェース仕様

表 3.40 JP1 信号配列

ピン番号	ピン名	I/O	説明
1	JP1	In	起動デバイス設定用信号、i.MX 8M Plus の BOOT_MODE0 ピンに接続、基板上で 100kΩ プルダウン
2	JP1_PU	Out	基板上で 4.7kΩ プルアップ(VDD_1V8)

3.6.23. ユーザースイッチを使用する

3.6.23.1. ハードウェア仕様

Armadillo-IoT ゲートウェイ G4 に搭載されているユーザースイッチには、GPIO が接続されています。

インターフェース仕様

SW1 は、ユーザー側で自由に利用できる押しボタンスイッチです。

表 3.41 SW1 信号配列

部品番号	名称	説明
SW1	ユーザースイッチ	i.MX 8M Plus の GPIO1_IO13 ピンに接続、基板上で 10kΩ プルアップ(VDD_1V8) (Low: 押された状態、High: 押されていない状態)

3.6.23.2. ソフトウェア仕様

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

ユーザースイッチと信号には、次に示すキーコードが割り当てられています。

表 3.42 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_PROG1	148	XF86Launch1
EC25-J RI	KEY_PROG2	149	XF86Launch2
予約	KEY_PROG3	202	XF86Launch3
予約	KEY_PROG4	203	XF86Launch4
PWR_OFF	KEY_POWER	116	XF86PowerOff
REBOOT	KEY_RESET	408	なし

デバイスファイル
- /dev/input/by-path/platform-gpio-keys-event [10]

3.6.23.3. 使用方法

Armadillo-IoT ゲートウェイ G4 にはユーザースイッチが実装されています。これらのスイッチのプッシュ/リリースイベントを取得するためには、Podman のイメージからコンテナを作成する際にホスト OS 側の /dev/input ディレクトリを渡す必要があります。以下は、/dev/input を渡して alpine イメージからコンテナを作成する例です。ここで渡された /dev/input ディレクトリはコンテナ内の /dev/input にマウントされます。

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

図 3.92 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/event1
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 1612849227.554456, type 1 (EV_KEY), code 28 (KEY_ENTER), value 1  ❶
Event: time 1612849227.554456, -------------- SYN_REPORT ------------
Event: time 1612849229.894444, type 1 (EV_KEY), code 28 (KEY_ENTER), value 0  ❷
Event: time 1612849229.894444, -------------- SYN_REPORT ------------
```

図 3.93 evtest コマンドによる確認例

[10]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

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「6.18. ボタンやキーを扱う」を参照してください。

3.6.24. LED を使用する

3.6.24.1. ハードウェア仕様

Armadillo-IoT ゲートウェイ G4 に搭載されているユーザー LED には、GPIO が接続されています。

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

表 3.43 LED3 の状態

部品番号	名称(色)	説明
LED3	ユーザー LED(緑)	トランジスタを経由して i.MX 8M Plus の GPIO1_IO14 ピンに接続 (Low: 消灯、High: 点灯)

インターフェース仕様 (LED4)　LED4 は、Armadillo-IoT ゲートウェイ G4 の電源確認用の LED です。

表 3.44 LED4 の状態

部品番号	名称(色)	状態	説明
LED4	電源 LED(緑)	点灯	VDD_3V3 が供給されている
		消灯	VDD_3V3 が供給されていない

3.6.24.2. 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/led1`

3.6.24.3. 使用方法

Armadillo-IoT ゲートウェイ G4 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。LED クラスディレクトリと LED の対応を次に示します。

表 3.45 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/led1/	ユーザー LED 緑	none

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 3.94 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/led1/brightness
[container ~]# echo 1 > /sys/class/leds/led1/brightness
```

図 3.95 LED の点灯/消灯の実行例

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

```
[container ~]# cat /sys/class/leds/led1/brightness
```

図 3.96 LED の状態を表示する

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

表 3.46 LED トリガの種類

設定	説明
none	トリガを設定しません
mmc1	microSD スロットのアクセスランプにします
mmc2	eMMC のアクセスランプにします
heartbeat	心拍のように点灯/消灯を行います
default-on	主に Linux カーネルから使用します。LED が点灯します

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

```
[container ~]# cat /sys/class/leds/led1/trigger
[none] rc-feedback bluetooth-power rfkill-any rfkill-none kbd-scrolllock kbd-num
lock kbd-capslock kbd-kanalock kbd-shiftlock kbd-altgrlock kbd-ctrllock kbd-altl
ock kbd-shiftllock kbd-shiftrlock kbd-ctrlllock kbd-ctrlrlock rfkill0 rfkill1 di
sk-activity disk-read disk-write ide-disk heartbeat cpu cpu0 cpu1 cpu2 cpu3 mmc2
 default-on panic mmc1
```

図 3.97 対応している LED トリガを表示

以下のコマンドを実行すると、心拍のように点灯/消灯を行います。

```
[container ~]# echo heartbeat > /sys/class/leds/led1/trigger
```

図 3.98 LED のトリガに heartbeat を指定する

3.6.25. LTE を使用する

3.6.25.1. ハードウェア仕様

Armadillo-IoT ゲートウェイ G4 の LTE 搭載モデルには、3G/LTE モデム Quectel 製 EC25-J が搭載されています。

機能

- 3G/LTE 通信
- リセットドライバによる EC25-J の電源制御
- LED による状態確認

インターフェース仕様(CON21, CON22)

CON21 及び CON22 は 3G/LTE モジュール用アンテナケーブルを接続するためのインターフェースです。

アンテナ端子にアンテナケーブルを接続する際、無理な力を加えると破壊の原因となりますので、十分にご注意ください。

インターフェース仕様(CON23)

CON23 は、3G/LTE モジュール用 nanoSIM インターフェースです。

表 3.47 CON23 信号配列

ピン番号	ピン名	I/O	説明
1	GND	Power	電源(GND)
2	SIM_VCC	Power	SIM 電源出力、3G/LTE モジュールの SIM インターフェースに接続
3	SIM_RST	Out	SIM リセット、3G/LTE モジュールの SIM インターフェースに接続
4	SIM_CLK	Out	SIM クロック、3G/LTE モジュールの SIM インターフェースに接続
5	SIM_VPP	-	未接続
6	SIM_I/O	In/Out	SIM データ、3G/LTE モジュールの SIM インターフェースに接続

CON23 は活線挿抜に対応しておりません。nanoSIM カードの挿抜は、本製品の電源を切断した状態で行ってください。

インターフェース仕様(LED20)　LED20 は、3G/LTE モジュールの通信状態を示す LED です。

表 3.48 LED20 の接続

部品番号	説明
LED20	LTE モジュールに接続

表 3.49 LED20 の状態

状態	説明
消灯	パワーオフ状態
点滅((200ms 点灯/1800ms 消灯)	ネットワーク探索状態
点滅(1800ms 点灯/200ms 消灯)	待機状態
点滅(125ms 点灯/125ms 消灯)	通信状態

3.6.25.2. ソフトウェア仕様

デバイスファイル
- /dev/ttyUSB0
- /dev/ttyUSB1
- /dev/ttyUSB2
  - ModemManager が /dev/ttyCommModem のシンボリックリンクを作成し AT コマンド用ポートとして使用します。
- /dev/ttyUSB3
- /dev/ttymxc3

ttyUSB[0-3] は、他の USB デバイスを接続している場合、番号が変わる可能性があります。

ネットワークデバイス
- ppp0

3.6.25.3. 使用方法

3G/LTE モデム Quectel 製 EC25-J に対して、以下の制御が可能です。

EC25-J は、Armadillo 起動時に自動的に電源が投入され、 Armadillo が終了する際には自動的に電源が切られます。

また、「6.15.7.9. 3G/LTE 再接続サービス」 でも、通信状態に応じて EC25-J のリセットなどを実施しますので処理が重複しないように、下記制御を実施する際には、「図 6.186. 3G/LTE 再接続サービスを停止する」の手順を参考に再接続サービスを停止してから実施してください。

```
[armadillo ~]# cat /sys/devices/platform/ec25-reset/reset/status
running ❶
```

```
[armadillo ~]# cat /sys/devices/platform/ec25-reset/reset/status
off ❷
```

図 3.99 EC25-J の状態表示

❶ 動作中は **running** が表示されます。

❷ 停止中は **off** が表示されます。

```
[armadillo ~]# wwan-force-restart
```

図 3.100 EC25-J をリセットまたは電源を入れる

```
[armadillo ~]# wwan-poweroff
```

図 3.101 EC25-J の電源を切る

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。

3.6.26. 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/ttymxc0 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/ttymxc0
[armadillo ~]# podman_start wisun_example
Starting 'wisun_example'
ef9a5a2f7eee4236cb28c1fbf5090a6f0db9d6dfe7f3a34573867e0833dd3122
[armadillo ~]# podman exec -it wisun_example sh
[container ~]# ls /dev/ttymxc0
/dev/ttymxc0
```

図 3.102 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/ttymxc0 を使って Wi-SUN データの送受信ができるようになります。

3.6.27. 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/ttymxc0 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/ttymxc0
[armadillo ~]# podman_start enocean_example
Starting 'enocean_example'
a808b491a100f9078d8c72a7b36966d9182614f3657fe054fb8d7f87b0d4b31c
[armadillo ~]# podman exec -it enocean_example sh
[container ~]# ls /dev/ttymxc0
/dev/ttymxc0
```

図 3.103 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/ttymxc0 を使って EnOcean データの送受信ができるようになります。

3.7. ソフトウェアの設計

Armadillo-IoT ゲートウェイ G4 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 3.104. 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。

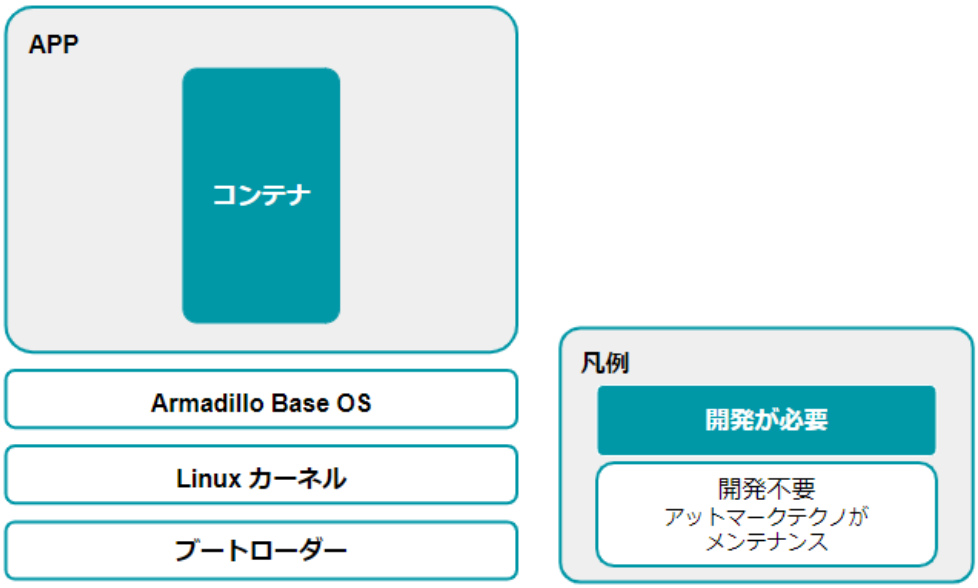


図 3.104 開発者が開発するもの、開発しなくていいもの

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。

VPU 及び NPU を使用する場合は、 アットマークテクノが提供する Debian GNU/Linux ベースのコンテナイメージ [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/container] の使用を推奨しますが、それ以外は 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。

3.7.3. 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、「3.2.4.1. 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 3.105. 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
Currently booted on /dev/mmcblk2p1 ❶
:（省略）
```

図 3.105 現在の面の確認方法

❶ この実行結果から今の面は/dev/mmcblk2p1 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-IoT ゲートウェイ G4 のウォッチドッグタイマーは、i.MX 8M Plus 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。Linux カーネルは、ウォッチドッグタイマードライバの初期化時にタイムアウト時間を 60 秒に再設定します。

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。

3.8. 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「6.8. 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 以前の ABOS から、v3.17.4-at.7 以降へアップデートした場合、アップデートよって新しく avahi サービスが追加されますが、新しく追加されたサービスが自動起動されることによる影響を防ぐため avahi は自動起動しない設定になっています。ABOS Web にアクセスできるようにするためには、この avahi サービスを自動起動する必要があるため、以下の手順で有効にしてください。

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

また、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 --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おらず、自動起動しません。この場合は ABOS Web のパスワードを以下のように設定してください。

```
[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、VSCode 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 3.106. 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。

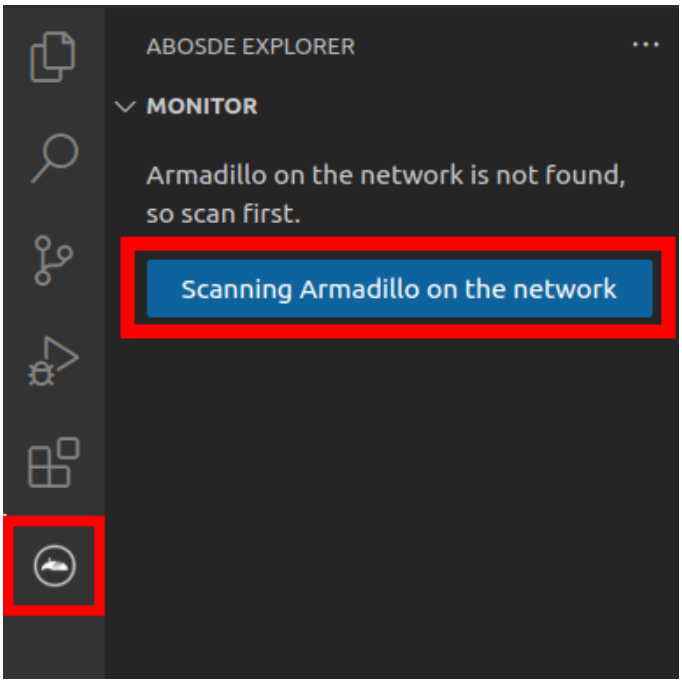


図 3.106 ABOSDE で ローカルネットワーク上の Armadillo をスキャンする

「図 3.107. 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。

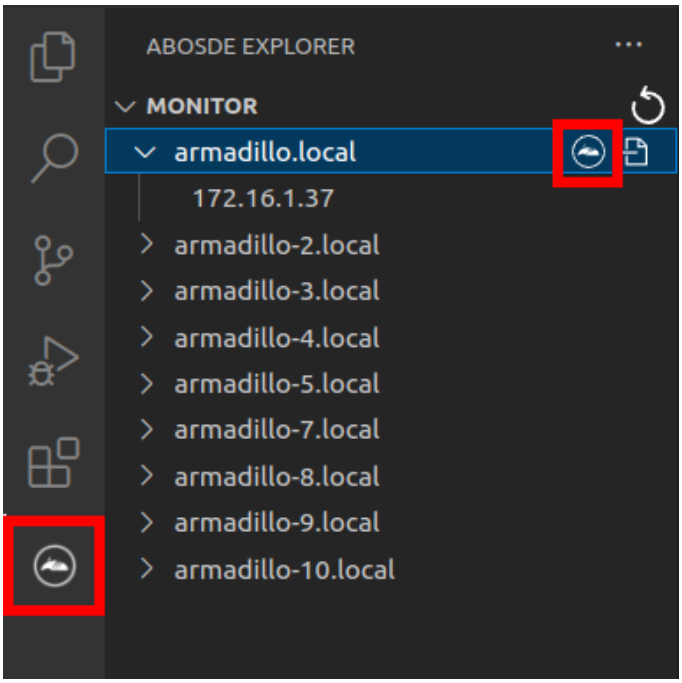


図 3.107 ABOSDE を使用して ABOS Web を開く

「図 3.108. 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。

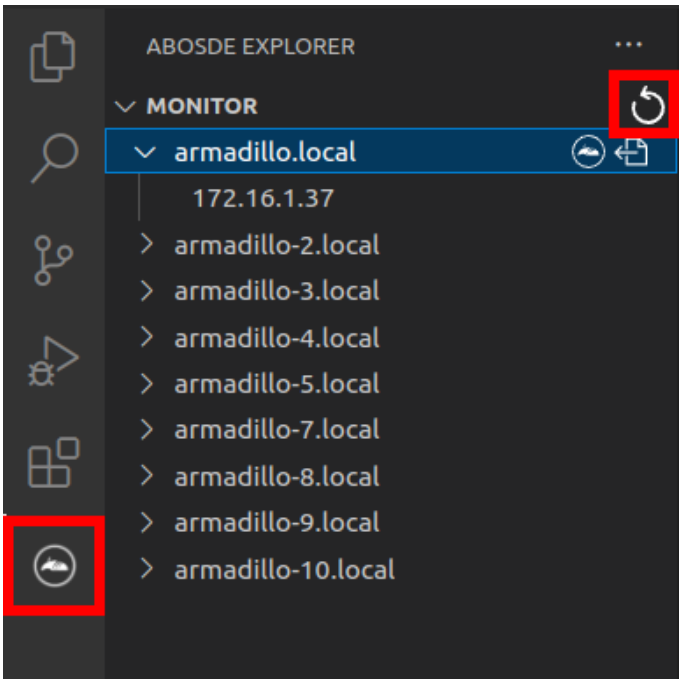


図 3.108 ABOSDE に表示されている Armadillo を更新する

3.8.3. ABOS Web のパスワード登録

「3.3.5.1. 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 3.109 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 3.110 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 3.111 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。

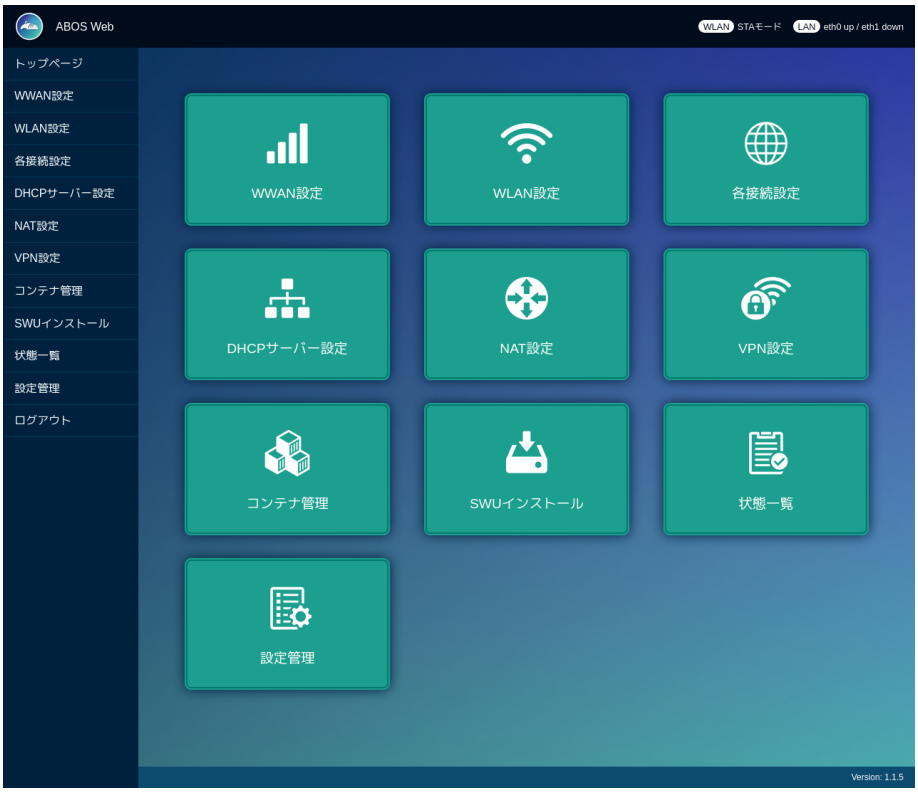


図 3.112 トップページ

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。

3.8.4. 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。

3.8.5. 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。

3.8.6. 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 3.113. WWAN 設定画面」に、WWAN 設定を行った状態を示します。

図 3.113 WWAN 設定画面

閉域 LTE 網を使用する料金プランをご契約で本サービスをご利用になられる際の注意点。

「6.15.7.9. 3G/LTE 再接続サービス」をご利用になられる場合、コネクション状態確認時 PING 送付先の初期値は 8.8.8.8 ですが、この IP アドレスに対して ping 導通ができない場合、ping 導通が可能となる IP アドレスを指定する必要があります。設定方法は、「6.15.7.9. 3G/LTE 再接続サービス」を参照ください。

3.8.7. 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。

3.8.7.1. 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 3.114. 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 3.114 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。

3.8.7.2. 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 3.115. 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 3.115 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。

3.8.8. 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 3.116 現在の接続情報画面

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

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

3.8.8.1. LAN 接続設定

LAN 接続の接続名は、デフォルトでは "Wired conneciton 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 3.117. 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 3.117 LAN 接続設定で固定 IP アドレスに設定した画面

3.8.8.2. WWAN 接続設定

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

3.8.8.3. 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。

3.8.9. 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 3.118. 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 3.118 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。

3.8.10. 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。

3.8.10.1. 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 3.119. 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 3.119 LTE を宛先インターフェースに指定した設定

3.8.10.2. 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 3.120. 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 3.120 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

3.8.10.3. 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 3.121. VPN 設定」に、VPN 接続設定を行った状態を示します。

図 3.121 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。

3.8.11. 状態一覧

各設定画面で行った設定の現在状態を、設定ごとに区切って一覧表示します。

3.9. ABOS Web をカスタマイズする

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

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

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

カスタマイズは、「設定管理」で行うことができます。

図 3.122 ABOS Web のカスタマイズ設定

- **ロゴ画像**

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

- **ヘッダロゴアイコン画像**

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

- **ヘッダタイトル**

  画面左上に常に表示されている文字列です。24 文字まで入力できます。

- **favicon 画像**

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

表 3.50 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。

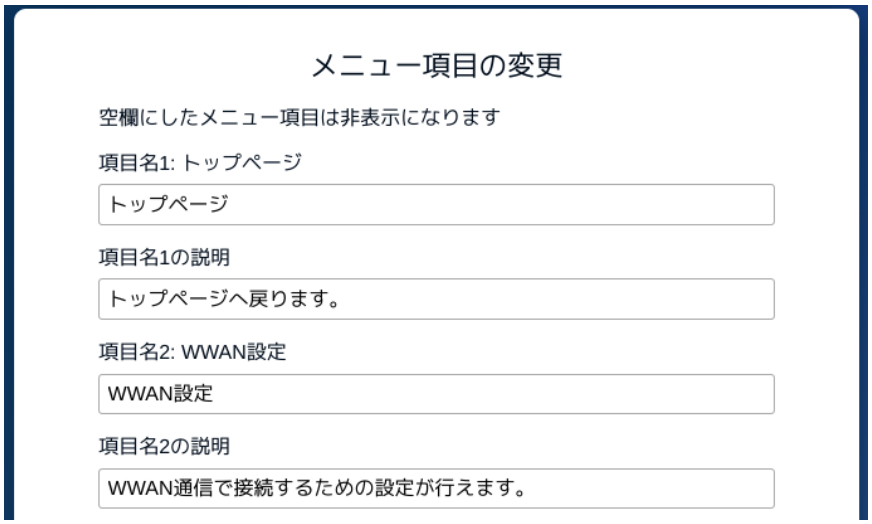


図 3.123 メニュー変更画面 (一部)

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

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

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

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

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

Armadillo Base OS では chronyd を使っています。

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、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/ と同じファイル名の設定ファイルを置いておくことで、デフォルトのファイルを読まないようになります。

例えば、 NTP サーバーの設定は servers.conf に記載されてますので、変更する際は/etc/chrony/conf.d/servers.conf に記載します：

```
[armadillo ~]# vi /etc/chrony/conf.d/servers.conf ❶
pool my.ntp.server iburst
[armadillo ~]# persist_file /etc/chrony/conf.d/servers.conf ❷
[armadillo ~]# rc-service chronyd restart ❸
```

```
chronyd          | * Stopping chronyd ... [ ok ]
chronyd          | * Starting chronyd ... [ ok ]
armadillo:~# chronyc sources ❹
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
^? my.ntp.server                 1   6     3     2    +88ms[  +88ms] +/-  173ms
```

図 3.124 chronyd のコンフィグの変更例

❶ コンフィグファイルを作成します。

❷ ファイルを保存します。

❸ chronyd サービスを再起動します。

❹ chronyc で新しいサーバーが使用されていることを確認します。

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、「6.8.5. 時刻設定」および 「6.8.6.8. Rest API : 時刻の設定」 を参照してください。

3.11. Armadillo Twin を体験する

Armadillo Twin を利用したデバイス運用管理を検討する場合は、一度 Armadillo Twin をお試し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。

3.12. ABOSDE によるアプリケーションの開発

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

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

- GUI アプリケーション
- CUI アプリケーション
- C 言語アプリケーション

3.12.1. ABOSDE の対応言語

「表 3.51. 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 3.51 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）
GUI アプリケーション	Dart (Flutter)
CUI アプリケーション	シェルスクリプト
	Python
C 言語アプリケーション	C 言語

3.12.2. 参照する開発手順の章の選択

どのようなアプリケーションを開発するかによって ABOSDE による開発手順が異なります。「図 3.125. 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。

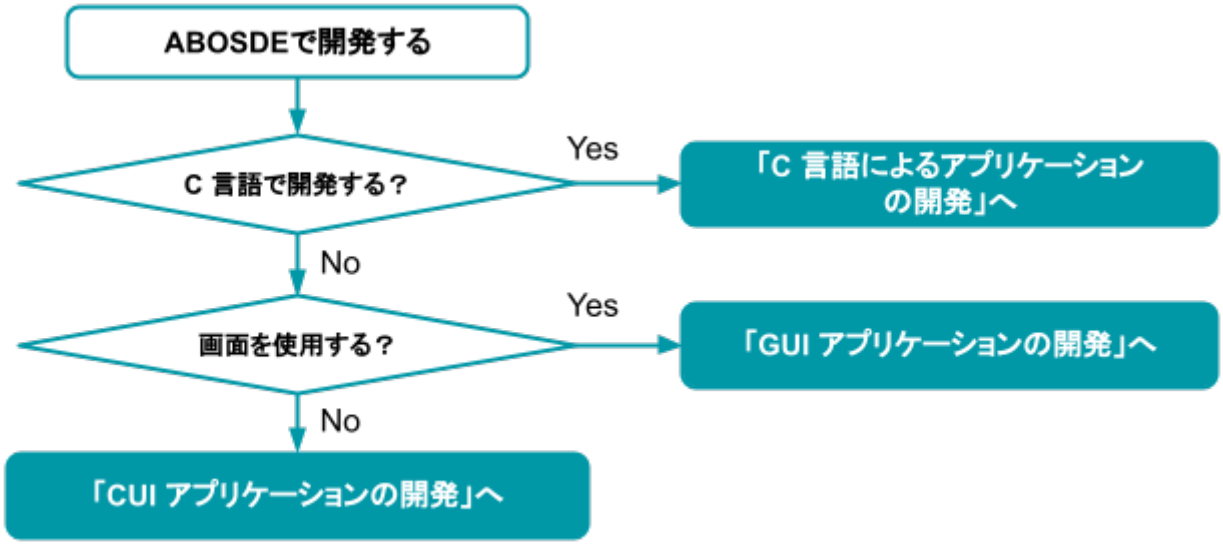


図 3.125 参照する開発手順の章を選択する流れ

GUI アプリケーション

- 対象ユーザー
  - 画面を使用するアプリケーションを開発したい
- マニュアルの参照先
  - 「3.13. GUI アプリケーションの開発」を参照

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
  - 「3.14. 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
  - 「3.15. C 言語によるアプリケーションの開発」を参照

3.13. GUI アプリケーションの開発

ここでは Armadillo の性能を最大限に生かした GUI アプリケーションを作ることのできる Flutter を使った開発方法を紹介します。

3.13.1. Flutter とは

Flutter とはモバイルアプリケーションや Web アプリケーションの開発に使われる GUI アプリケーション開発ツールキットです。マルチプラットフォームなので、ソースコードの大部分を共通化可能で一度開発したアプリケーションは最小限の工数で別のプラットフォームへ移植できます。さらに、プラットフォーム間でアプリケーションの見た目も統一することができます。アプリケーション開発言語として Dart を使用しています。

Flutter を使うことで Armadillo 上でも GUI アプリケーションを開発することができます。以下は Flutter で開発したアプリケーションを Armadillo 上で動かしている例です。

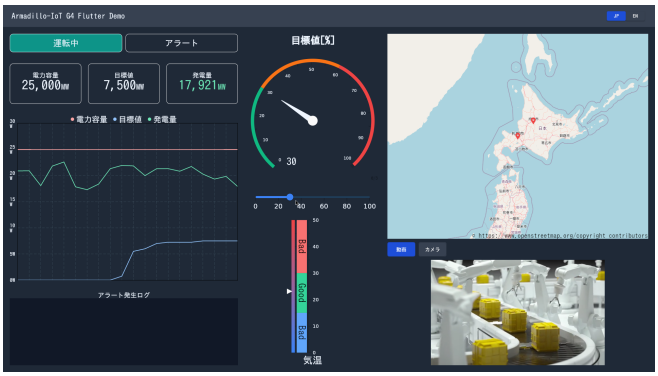


図 3.126 Flutter アプリケーションの例

3.13.2. Flutter を用いた開発の流れ

Armadillo 向けに Flutter 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。

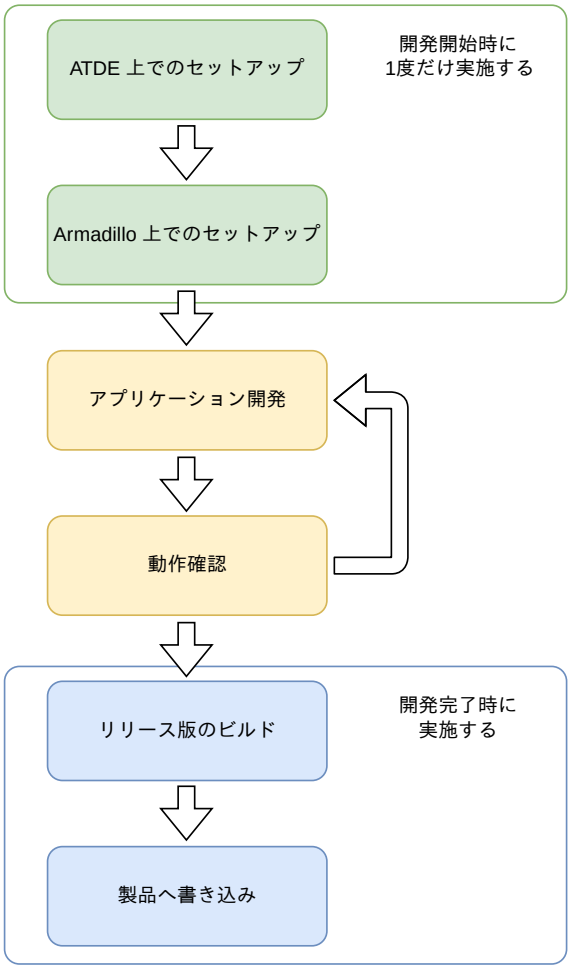


図 3.127 Flutter アプリケーション開発の流れ

3.13.3. 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 「3.3. 開発の準備」 を参照して ATDE のセットアップを完了してください。

3.13.3.1. プロジェクトの作成

Flutter アプリケーションのサンプルとして以下を用意しております。

- Flutter Demo アプリケーション
- GUI アプリケーション
- Signage アプリケーション
- Factory Signage アプリケーション

各プロジェクトは以下のようなアプリケーションの画面となります。

図 3.128 Flutter Demo アプリケーションの画面

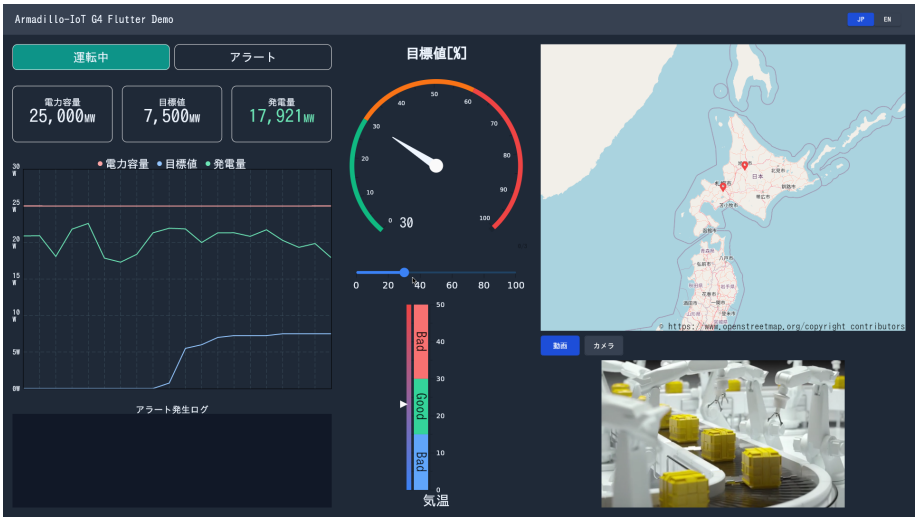


図 3.129 GUI アプリケーションの画面

図 3.130 Signage アプリケーションの画面

図 3.131 Factory Signage アプリケーションの画面

以降の手順でサンプルアプリケーション毎に VSCode でクリックする箇所や生成されるファイル名等が変わります。

VSCode の左ペインの [G4/X2] から [<アプリケーション名> 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。ここでは、ホームディレクトリ直下に `my_project` として保存しています。以下では例として [GUI アプリケーション] の作成を行っています。

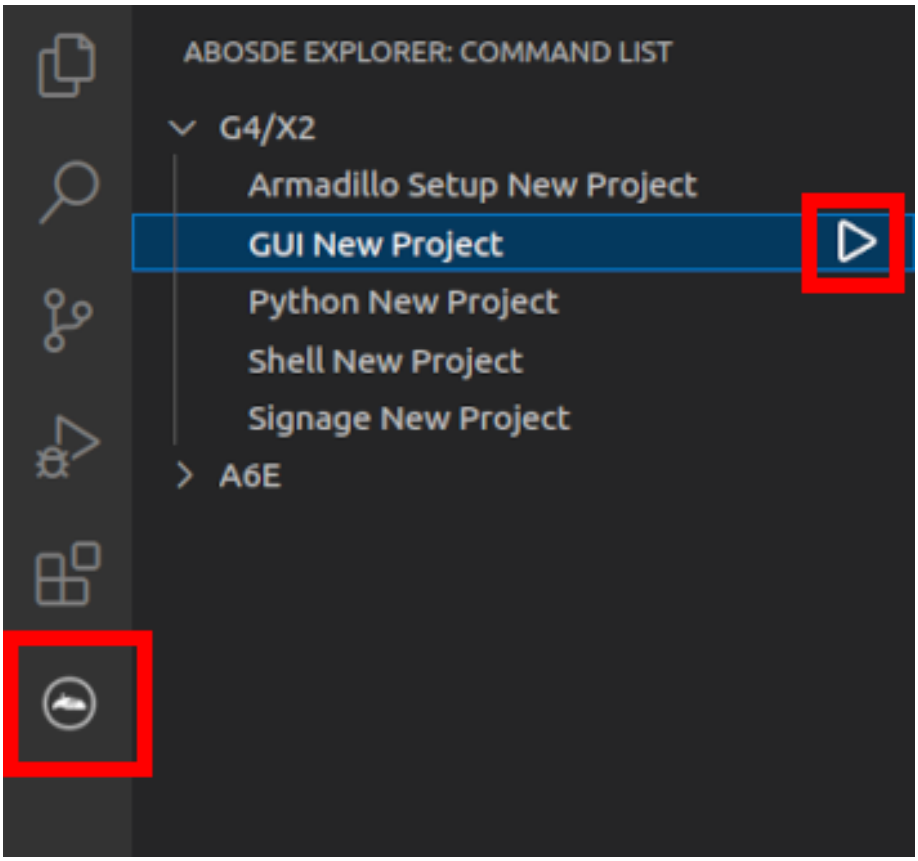


図 3.132 GUI アプリケーションのプロジェクトを作成する

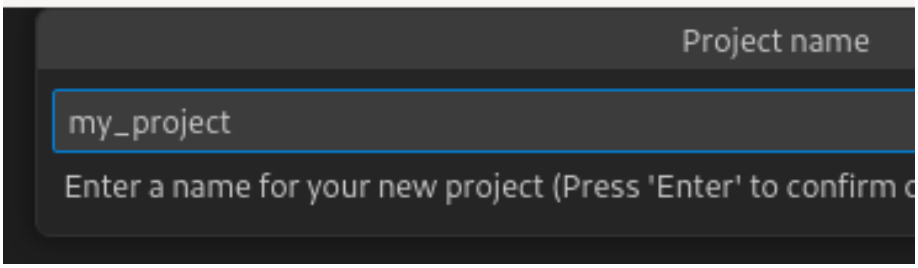


図 3.133 プロジェクト名を入力する

3.13.3.2. ディレクトリ構成

プロジェクトには下記のディレクトリがあります。

- app: Flutter アプリケーションです。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。
- config: 設定ファイルです。各ファイルが設定するものは以下のとおりです
  - app.conf: コンテナのコンフィグです。記載内容については 「6.2.4. コンテナ起動設定ファイルを作成する」 を参照してください。
  - app.desc: SWU イメージを生成するための .desc ファイルです。記載内容については 「6.4. mkswu の .desc ファイルを編集する」 を参照してください。

- ssh_config: Armadillo への ssh 接続に使用します。「3.14.7.2. ssh 接続に使用する IP アドレスの設定」 を参照してください。

- container: スクリプトを実行するコンテナの設定ファイルです。 packages.txt に記載されているパッケージがインストールされます。 Dockerfile を直接編集することも可能です。

3.13.3.3. 初期設定

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

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

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

図 3.134 初期設定を行う

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

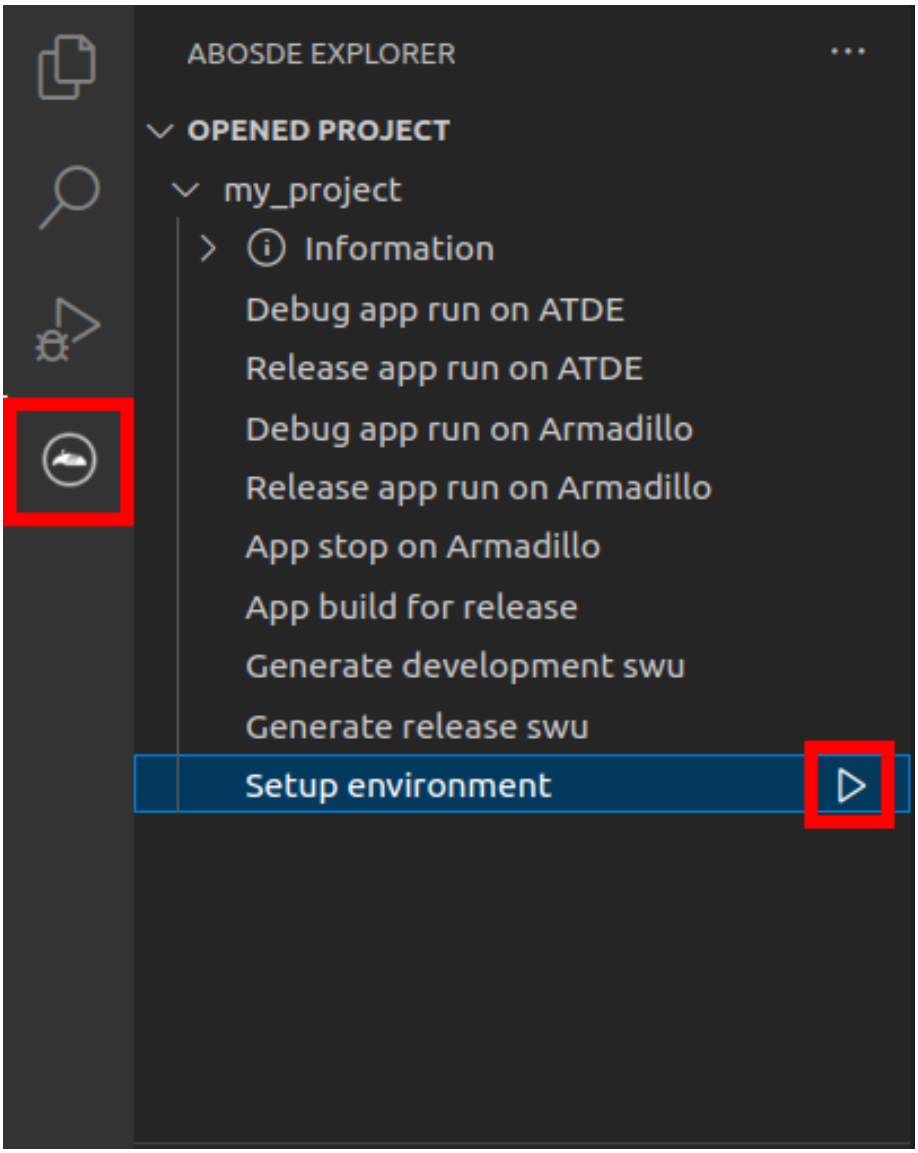


図 3.135 VSCode で初期設定を行う

選択すると、 VSCode の下部に以下のようなターミナルが表示されます。

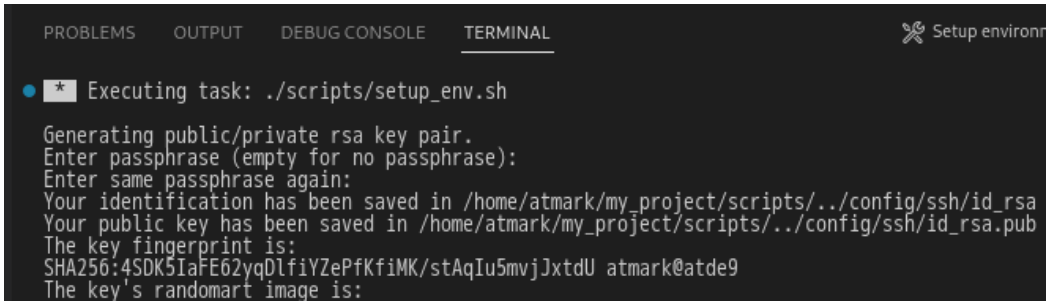


図 3.136 VSCode のターミナル

このターミナル上で以下のように入力してください。

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

Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase): ❶
Enter same passphrase again:
Your identification has been saved in config/ssh/id_rsa
Your public key has been saved in config/ssh/id_rsa.pub
:(省略)

 *  Terminal will be reused by tasks, press any key to close it.  ❷
```

図 3.137 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　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。

3.13.3.4. アプリケーション実行用コンテナイメージの作成

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

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

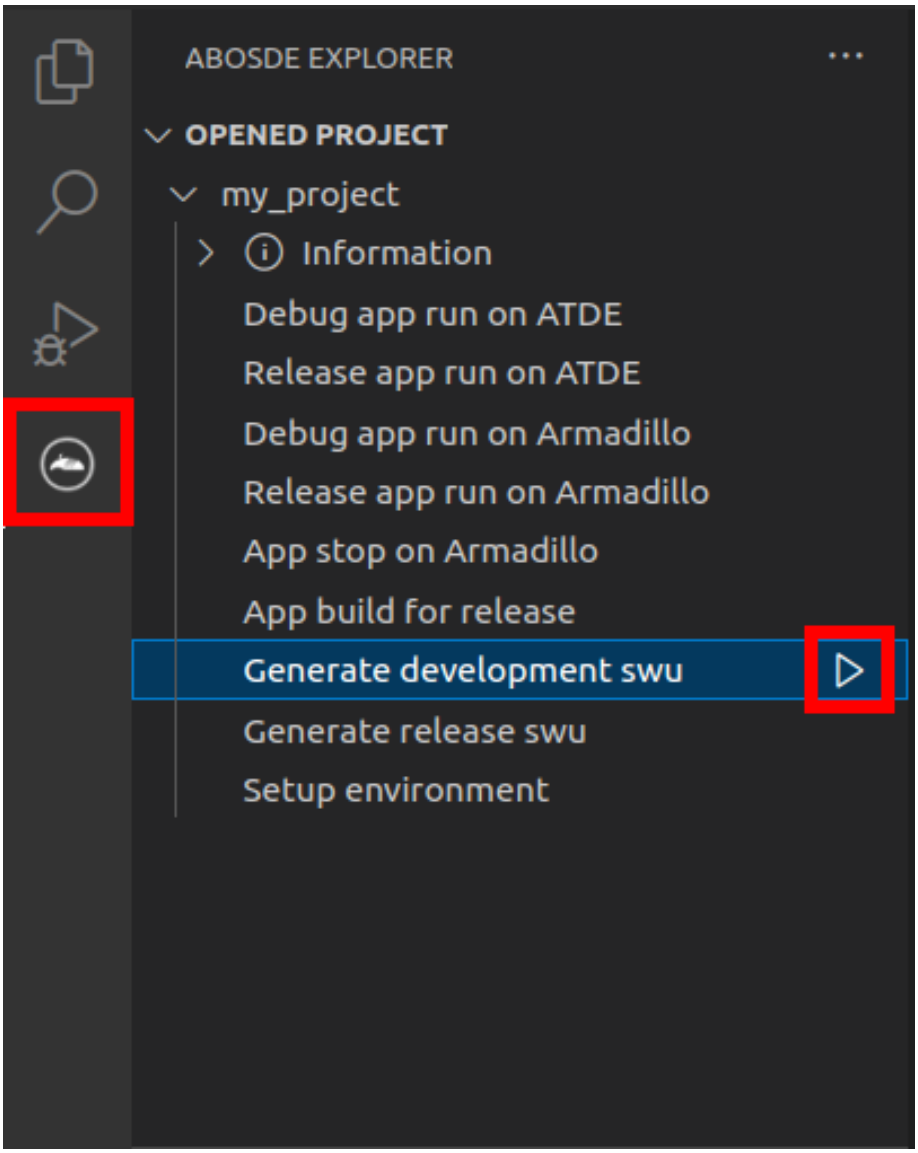


図 3.138 VSCode でコンテナイメージの作成を行う

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

```
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 3.139 コンテナイメージの作成完了

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

3.13.4. コンテナのディストリビューション

使用するコンテナのディストリビューションを以下のとおりです。

ディストリビューション	・ debian:bullseye

3.13.5. Armadillo に転送するディレクトリ及びファイル

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

Armadillo に転送するディレクトリ及びファイル	・ my_project/swu/app ・ my_project/app/build/elinux/arm64/[debug または release]/bundle

3.13.6. コンテナ内のファイル一覧表示

「図 3.140. コンテナ内のファイル一覧を表示するタブ」 の赤枠で囲われているタブをクリックすることで、`development.swu` または「3.13.10. 製品への書き込み」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。

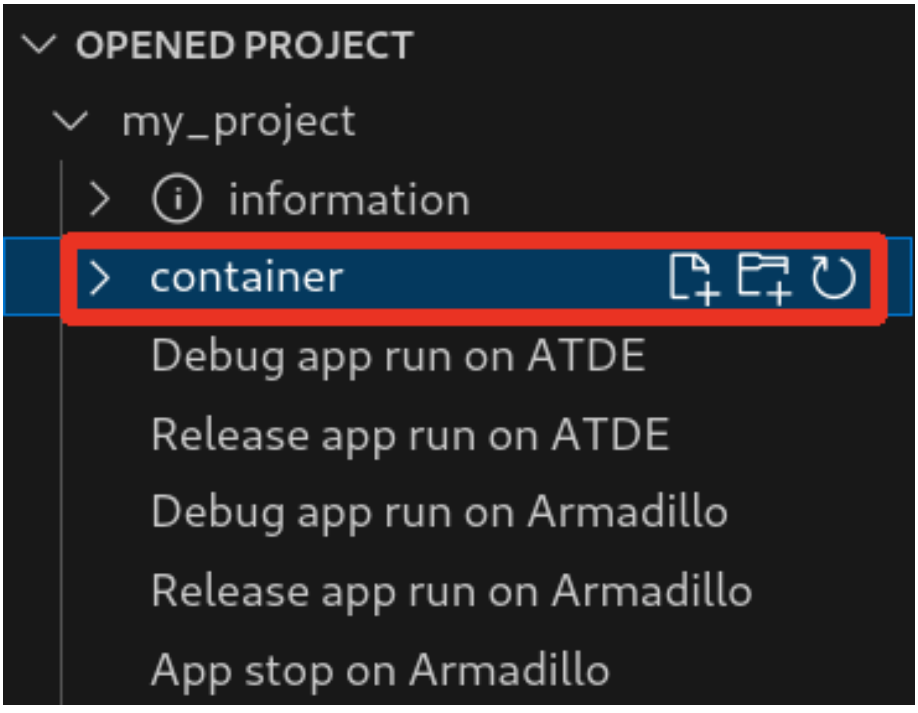


図 3.140 コンテナ内のファイル一覧を表示するタブ

クリック後の表示例を 「図 3.141. コンテナ内のファイル一覧の例」 に示します。

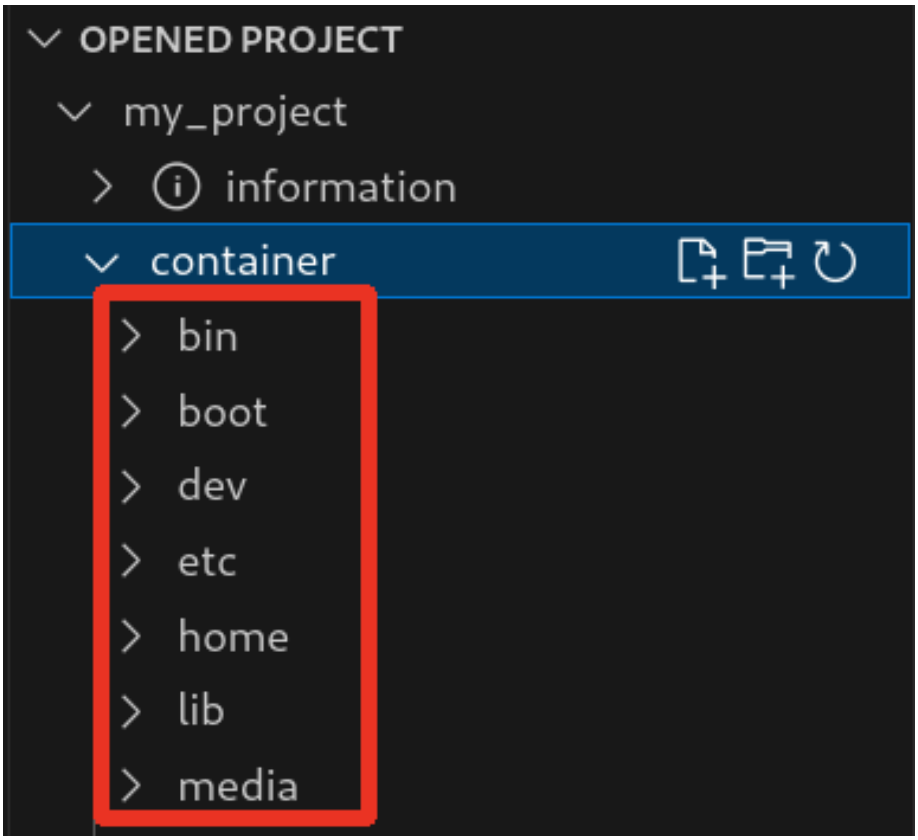


図 3.141 コンテナ内のファイル一覧の例

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

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

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

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

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

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

3.13.6.1. resources ディレクトリについて

「図 3.142. 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。

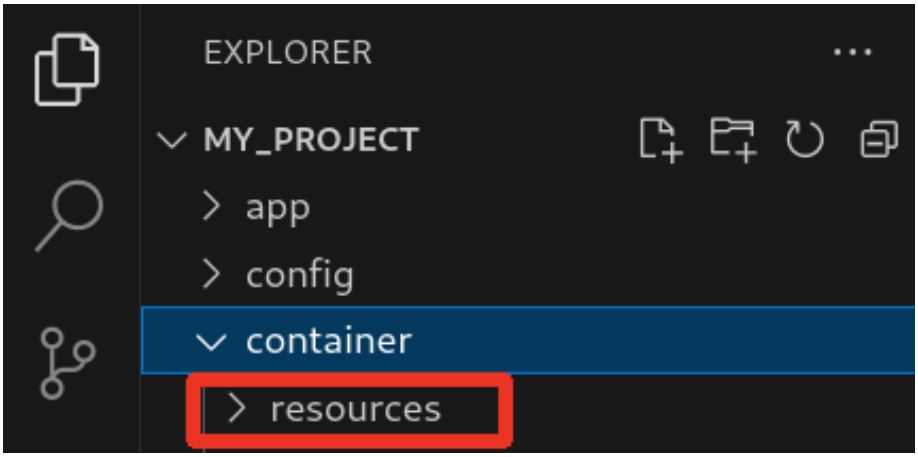


図 3.142 resources ディレクトリ

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

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

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

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

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

3.13.6.2. コンテナ内のファイル一覧の再表示

「図 3.140. 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。

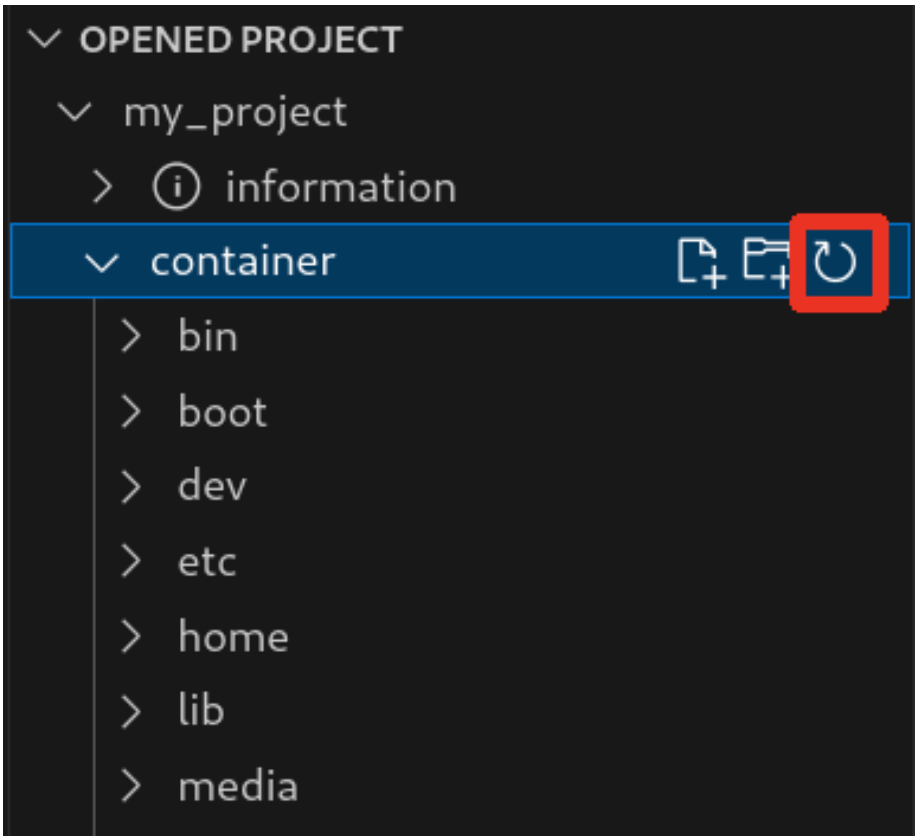


図 3.143 コンテナ内のファイル一覧を再表示するボタン

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

「図 3.144. 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。

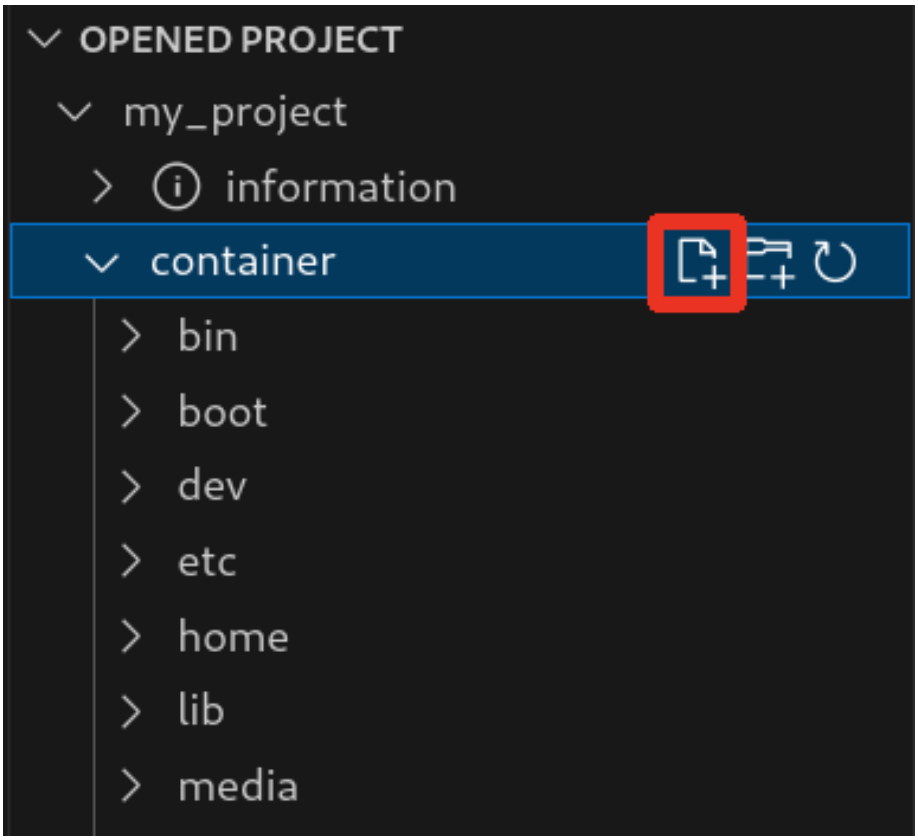


図 3.144 container/resources 下にファイルを追加するボタン

「図 3.145. 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。

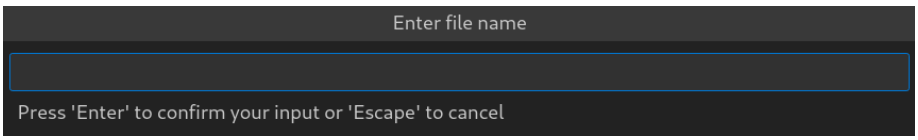


図 3.145 ファイル名を入力

例として、「add_file」というファイル名を入力したとします。「図 3.146. 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。

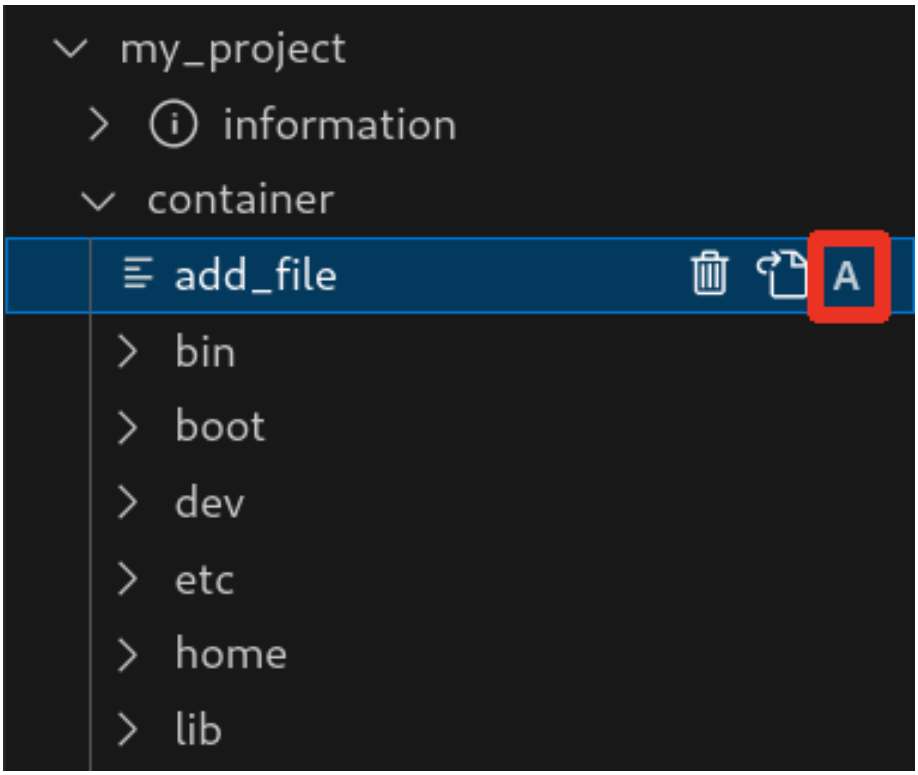


図 3.146 追加されたファイルの表示

また、「図 3.147. 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。

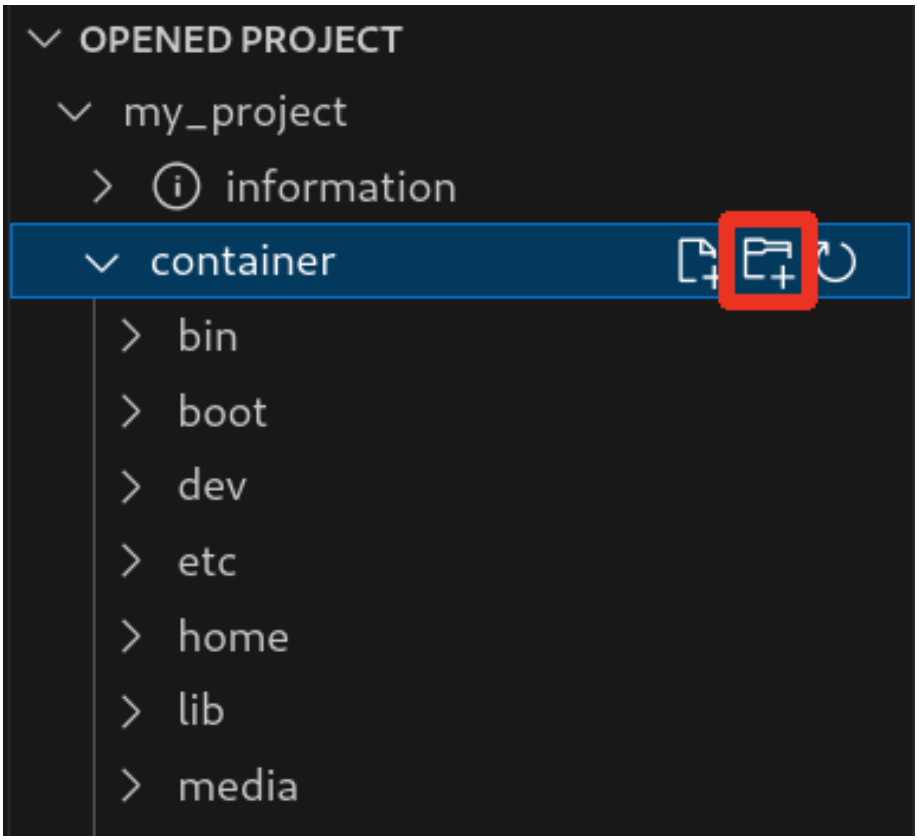


図 3.147 container/resources 下にフォルダーを追加するボタン

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

「図 3.148. 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。

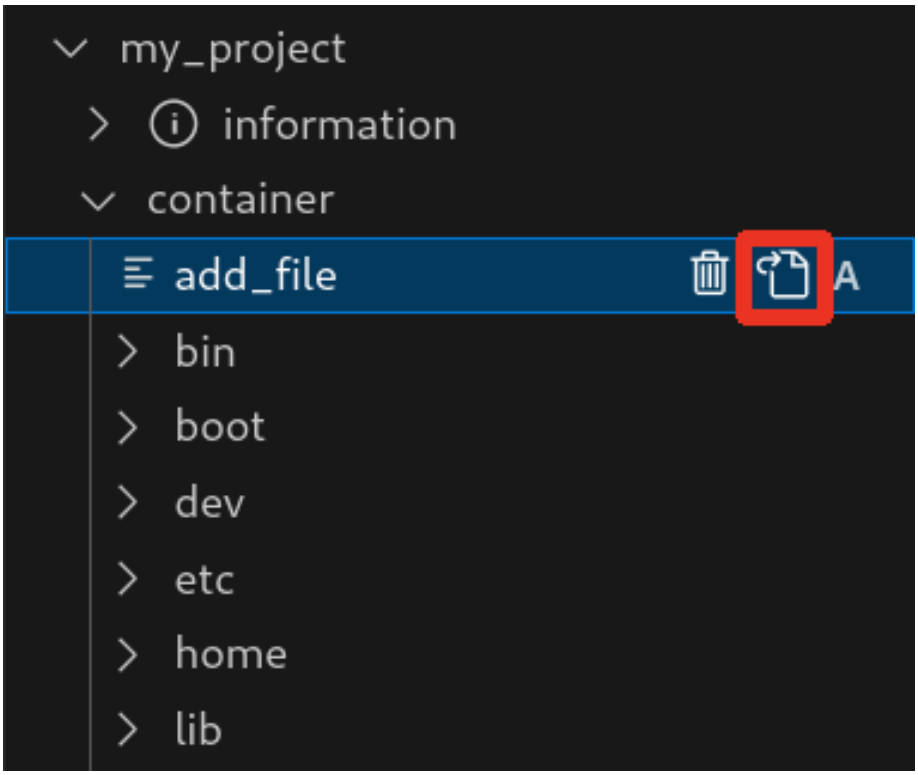


図 3.148 container/resources 下にあるファイルを開くボタン

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

「図 3.148. 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。

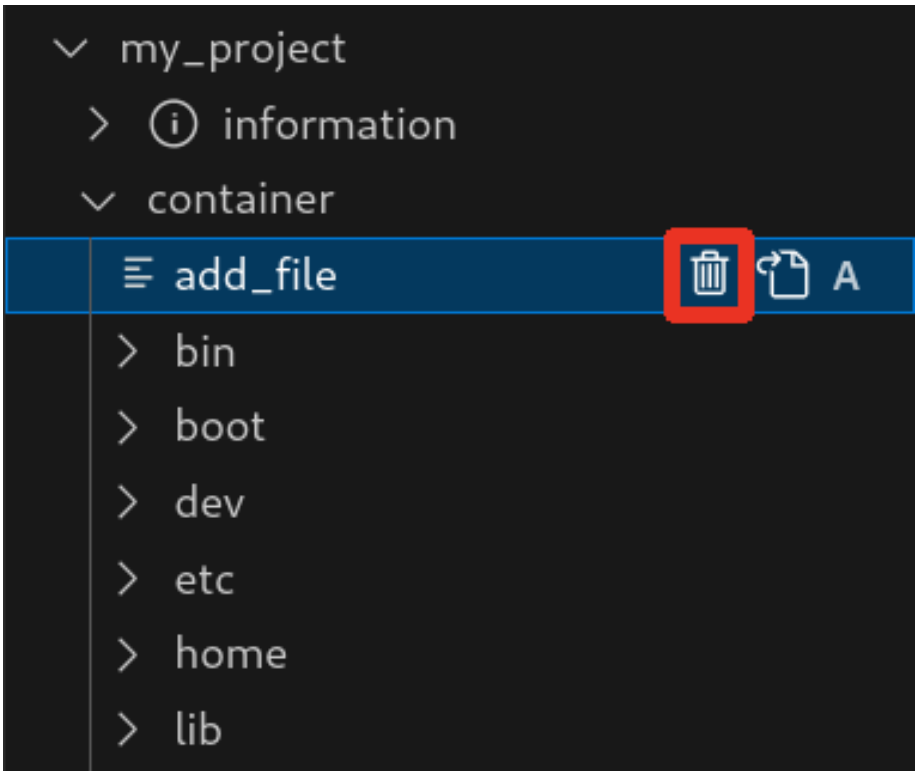


図 3.149 container/resources 下にあるファイルを削除するボタン

ディレクトリも同様に「図 3.148. container/resources 下にあるファイルを開くボタン」 の赤枠で囲われている表記のボタンをクリックすることで削除することができます。

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

「図 3.150. 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。

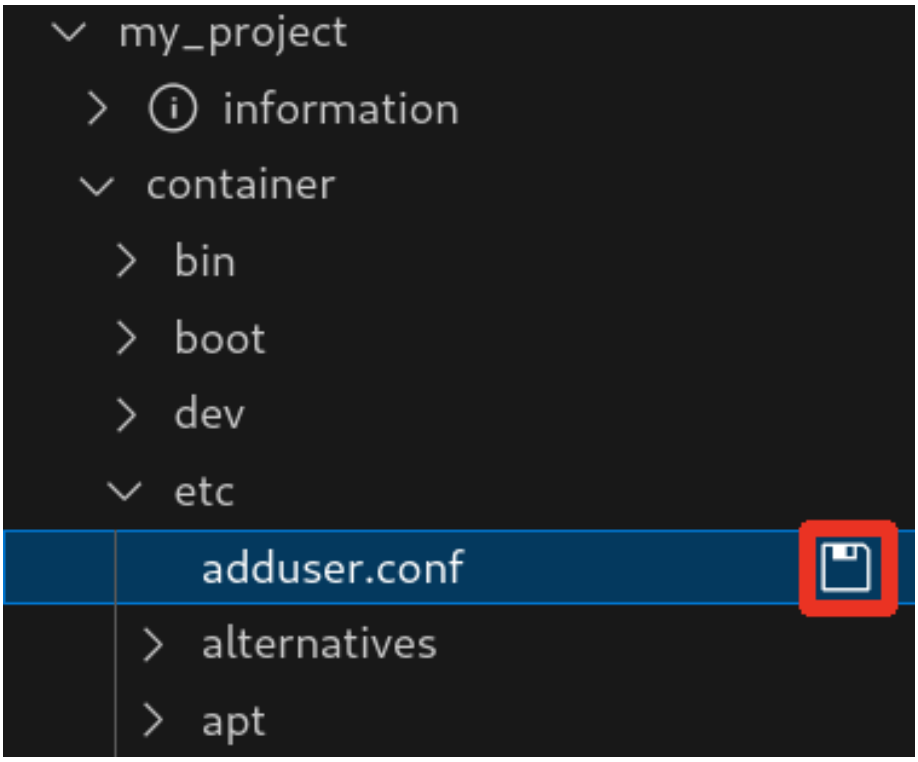


図 3.150 コンテナ内のファイルを container/resources 下に保存するボタン

ファイルが container/resources 下に保存されると、「図 3.151. 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。

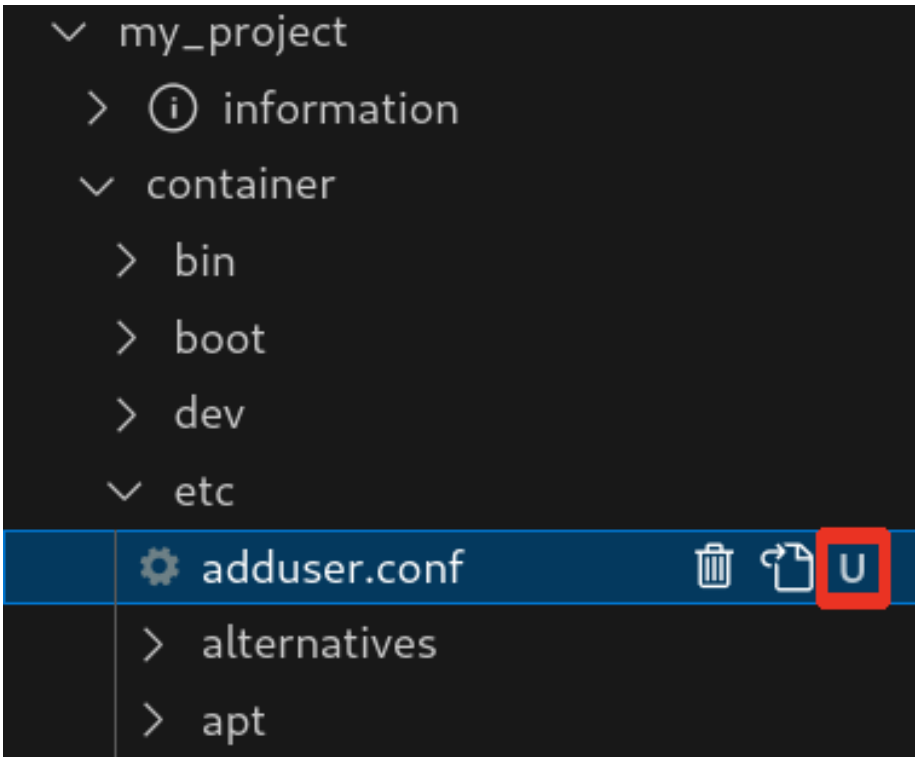


図 3.151 編集前のファイルを示すマーク

container/resources 下にあるファイルを編集して再表示すると、「図 3.152. 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。

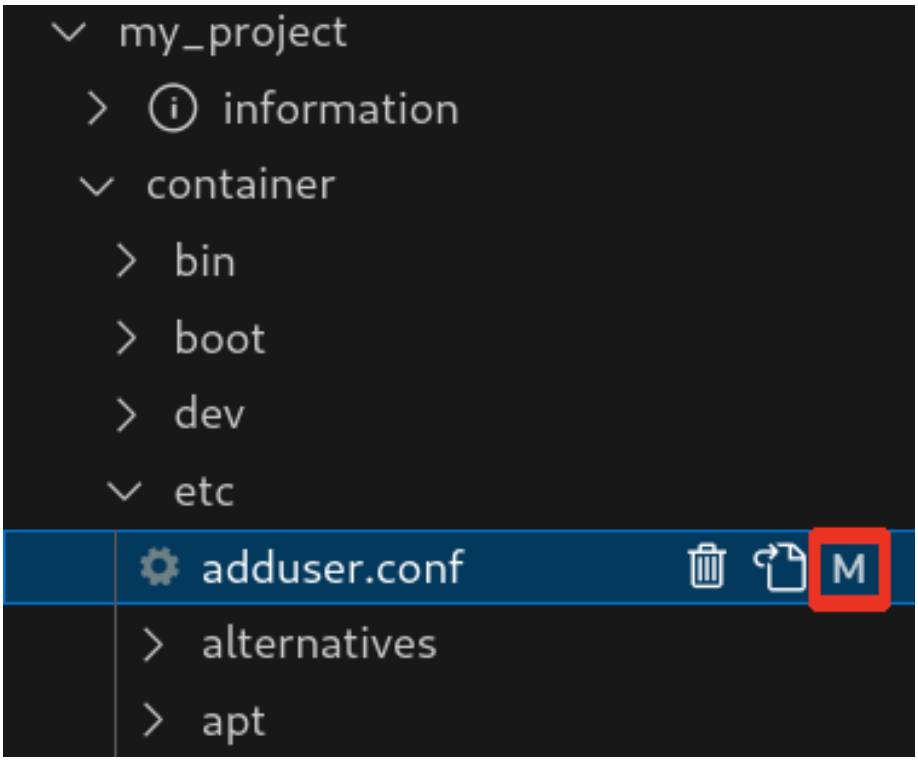


図 3.152 編集後のファイルを示すマーク

3.13.6.7. エラー表示

プロジェクトディレクトリにある 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 3.153. 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。

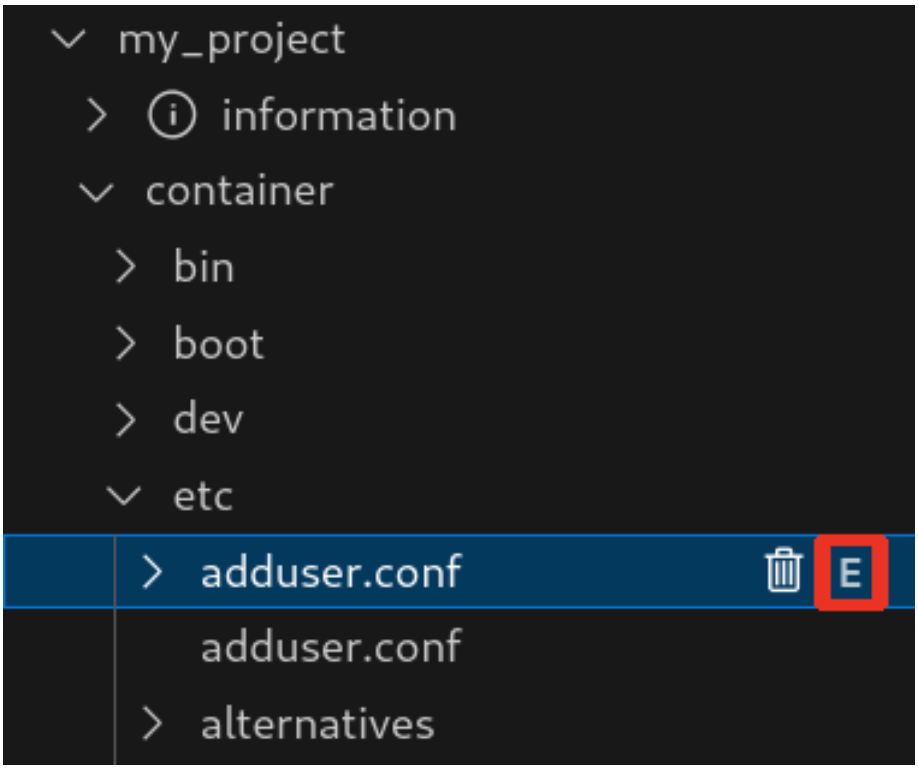


図 3.153 コンテナ内にコピーされないことを示すマーク

3.13.7. Armadillo 上でのセットアップ

3.13.7.1. ディスプレイの接続

「3.3.3.1. Armadillo と開発用 PC を接続」 を参照して Armadillo にディスプレイを接続してください。

3.13.7.2. アプリケーション実行用コンテナイメージのインストール

「3.13.3.4. アプリケーション実行用コンテナイメージの作成」 で作成した `development.swu` を「3.2.3.5. 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。この際、 `weston` も自動起動します。

3.13.8. アプリケーション開発

3.13.8.1. アプリケーションのビルドモード

Flutter アプリケーションのビルドモードには Debug、 Profile、 Release の 3 種類があり、VSCode からは Debug、Release モードの実行が可能です。Debug モードでビルドしたアプリケーションは後述するホットリロード等のデバッグ機能を用いて、効率的に開発が可能ですが、アプリケーションの動作が重くなります。特に動画やアニメーションの動作に大きく影響が出ますので、その場合は Release モードで動作を確認してください。

3.13.8.2. サンプルアプリケーションのビルド

Flutter のサンプルアプリケーションのビルド方法を説明します。プロジェクトディレクトリへ移動し VSCode を起動します。

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

図 3.154 my_project へ移動して VSCode を起動する。

VSCode の左ペインの [my_project] から [Debug app run on ATDE] を実行すると、Debug モードでアプリケーションがビルドされ ATDE 上で起動します。

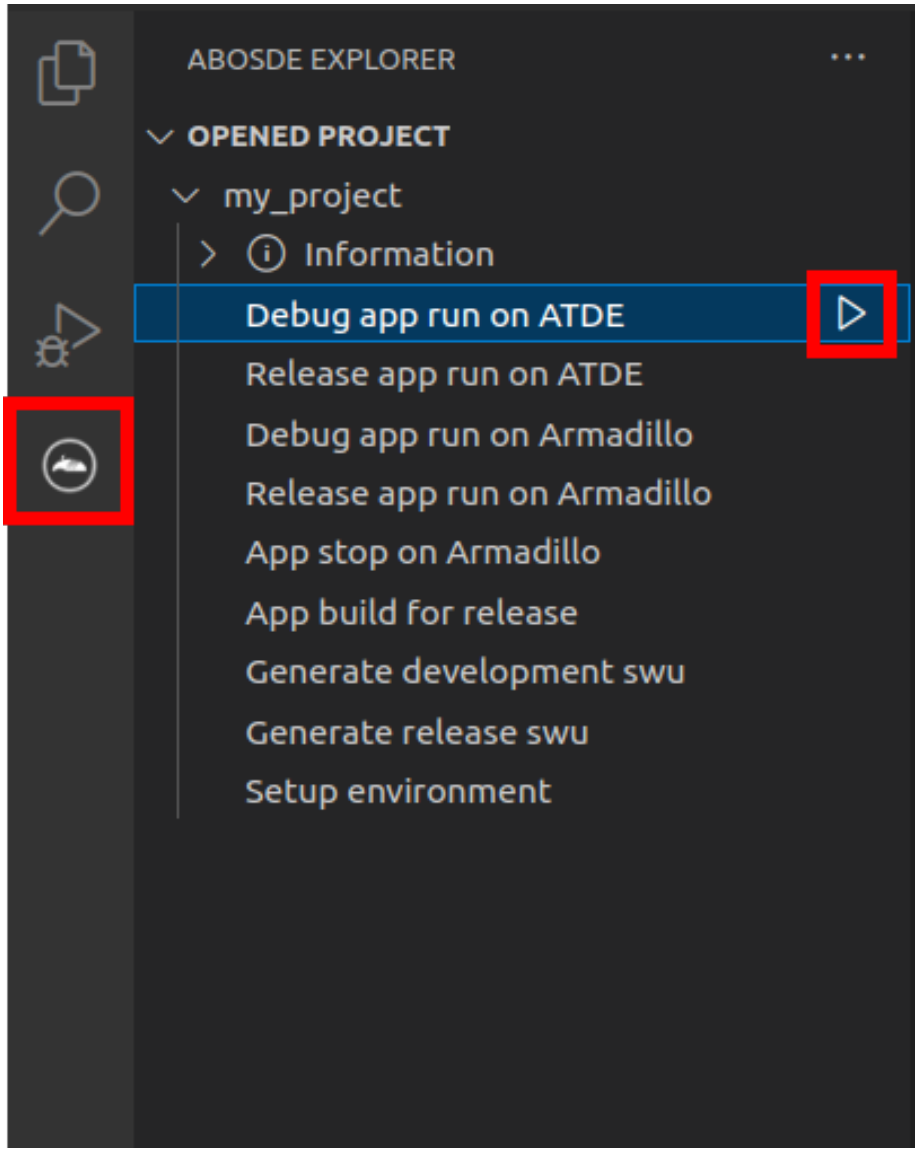


図 3.155 ATDE 上で Debug モードでビルドしたアプリケーションを実行する

flutter-elinux をイントール後に初めてビルドを実行する時は、必要なファイルのダウンロード処理が行われるため、アプリケーションが起動するまでに時間がかかります。

GUI アプリケーションの場合は以下のようなアプリケーションが起動します。

図 3.156 起動したサンプルアプリケーション

アプリケーションを終了するにはウィンドウ右上の X ボタンを押してください。

また、Release モードでアプリケーションを実行するには、VSCode の左ペインの [my_project] から [Release app run on ATDE] を実行してください。

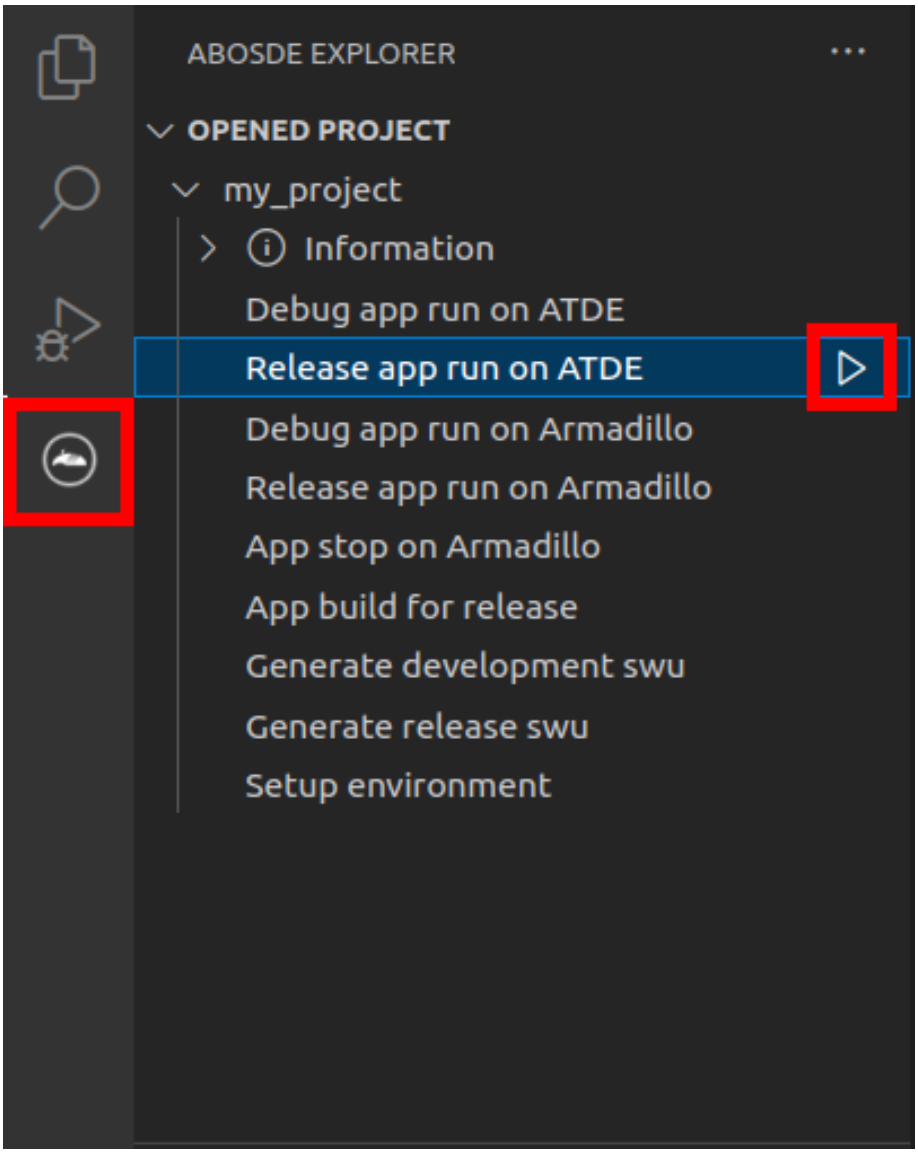


図 3.157 ATDE 上で Release モードでビルドしたアプリケーションを実行する

サンプルアプリケーションのソースコードは、 app/lib にあります。サンプルアプリケーションをベースとして開発を進める場合は、 app/lib 下にソースコードを保存してください。

3.13.8.3. パッケージをインストールする

Flutter には様々な機能を実現するためのパッケージが豊富に存在しており、主に こちらのサイト[https://pub.dev/]で見つけることができます。

目的のパッケージをアプリケーションで使えるようにするためには、アプリケーションディレクトリの中で以下のコマンドを実行します。例として dart_periphery パッケージをインストールします。

```
[ATDE ~/my_project]$ cd app
[ATDE ~/my_project/app]$ flutter-elinux pub add dart_periphery
```

図 3.158 dart_periphery パッケージをインストールする例

video_player や camera など以下に挙げたパッケージは、ATDE 内の /opt/flutter-elinux-packages にあるパッケージと組み合わせて使う必要があります。

表 3.52 組み合わせて使うパッケージ

パッケージ名	/opt/flutter-elinux-package 内のパッケージ名
video_player	video_player_elinux
camera	camera_elinux
path_provider	path_provider_elinux
shared_preferences	shared_preferences_elinux
なし	joystick

これらのパッケージをインストールする場合は以下のようにインストールしてください。

```
[ATDE ~/my_project/app]$ flutter-elinux pub add video_player
[ATDE ~/my_project/app]$ flutter-elinux pub add video_player_elinux ¥
--path /opt/flutter-elinux-plugins/packages/video_player
```

図 3.159 video_player パッケージをインストールする例

パッケージをアンインストールする場合は pub remove を実行します。

```
[ATDE ~/my_project/app]$ flutter-elinux pub remove dart_periphery
```

図 3.160 dart_periphery パッケージをアンインストールする例

3.13.9. 動作確認

ここでは、実際に Armadillo 上でアプリケーションを起動する場合の手順を説明します。

3.13.9.1. ssh 接続に使用する IP アドレスの設定

VSCode 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 3.161. 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。

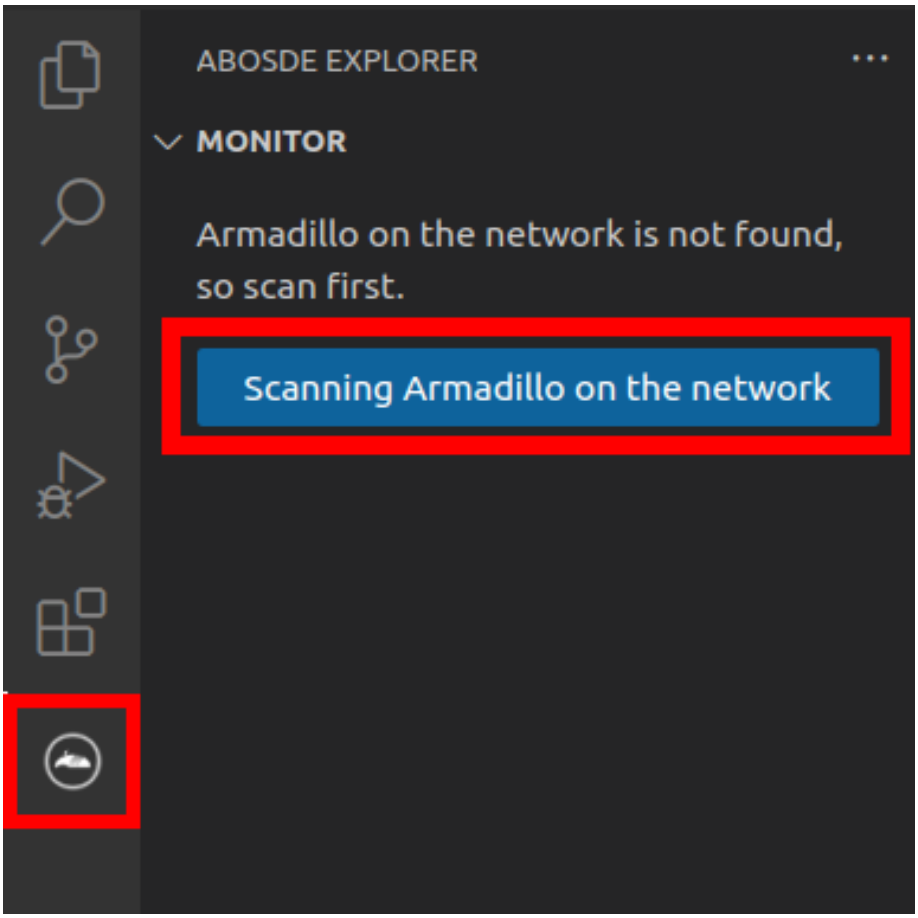


図 3.161 ABOSDE で ローカルネットワーク上の Armadillo をスキャンする

「図 3.162. 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。

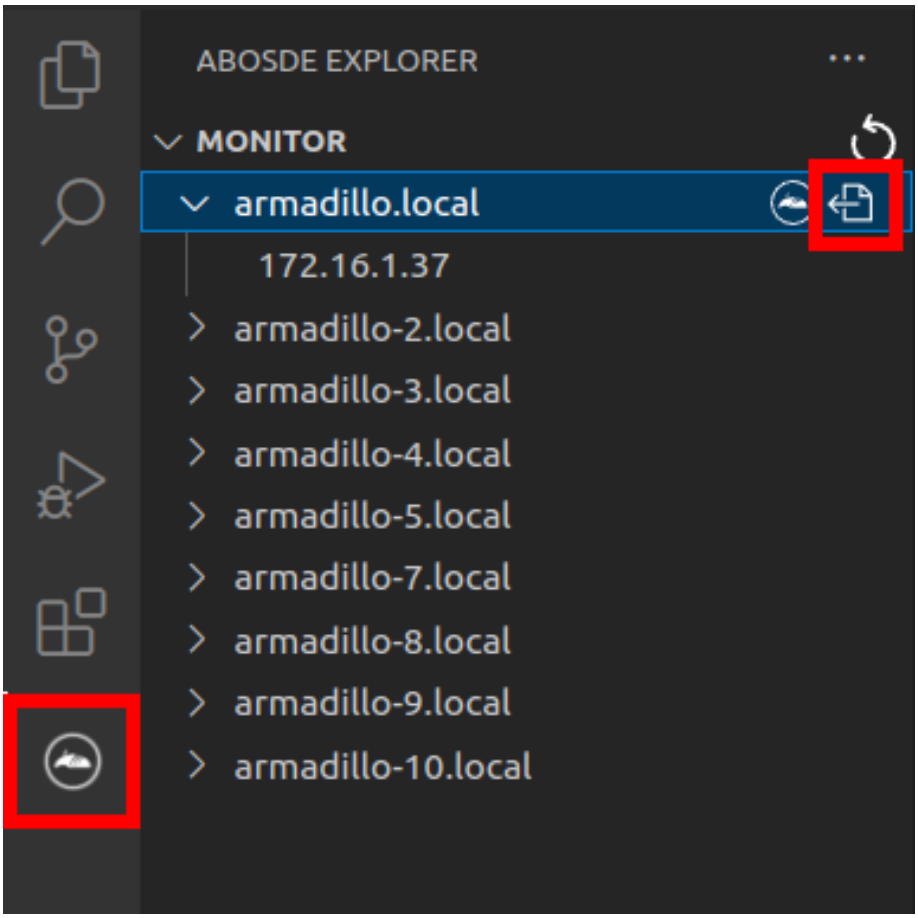


図 3.162 ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する

「図 3.163. 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。

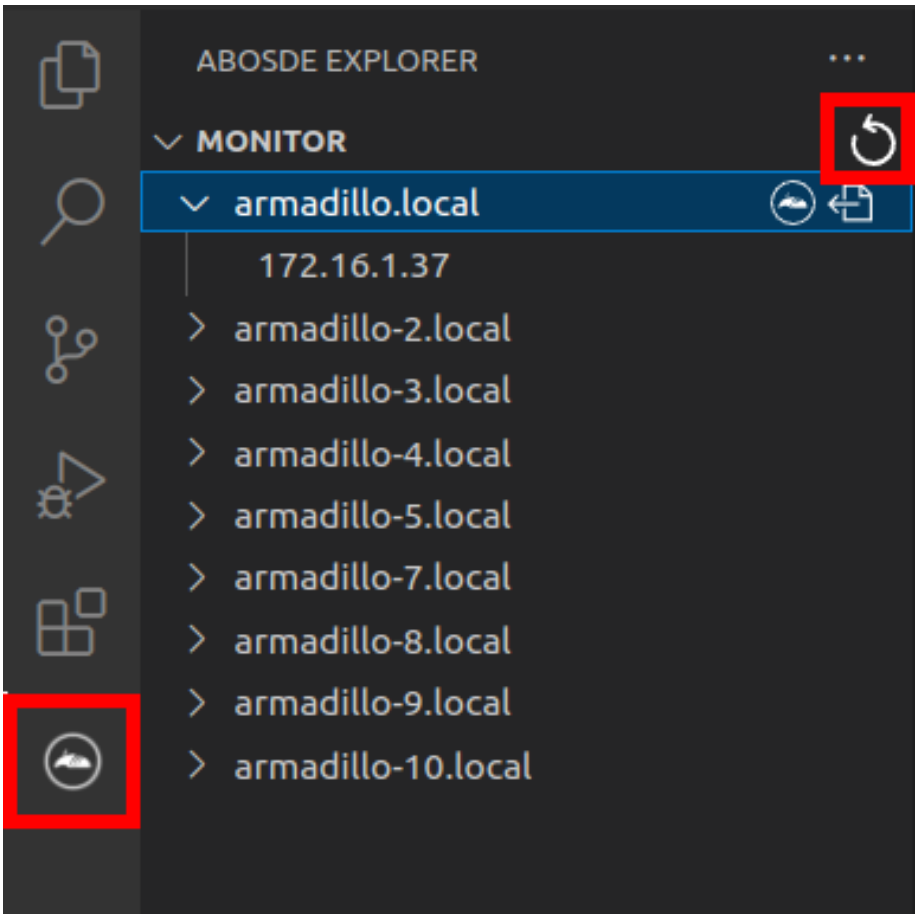


図 3.163 ABOSDE に表示されている Armadillo を更新する

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
    Port 2222
    IdentityFile ../config/ssh/id_rsa
```

図 3.164 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。

3.13.9.2. アプリケーションの実行

VSCode の左ペインの [my_project] から [Debug app run on Armadillo] を実行すると、Debug モードでビルドされたアプリケーションが Armadillo へ転送されて起動します。

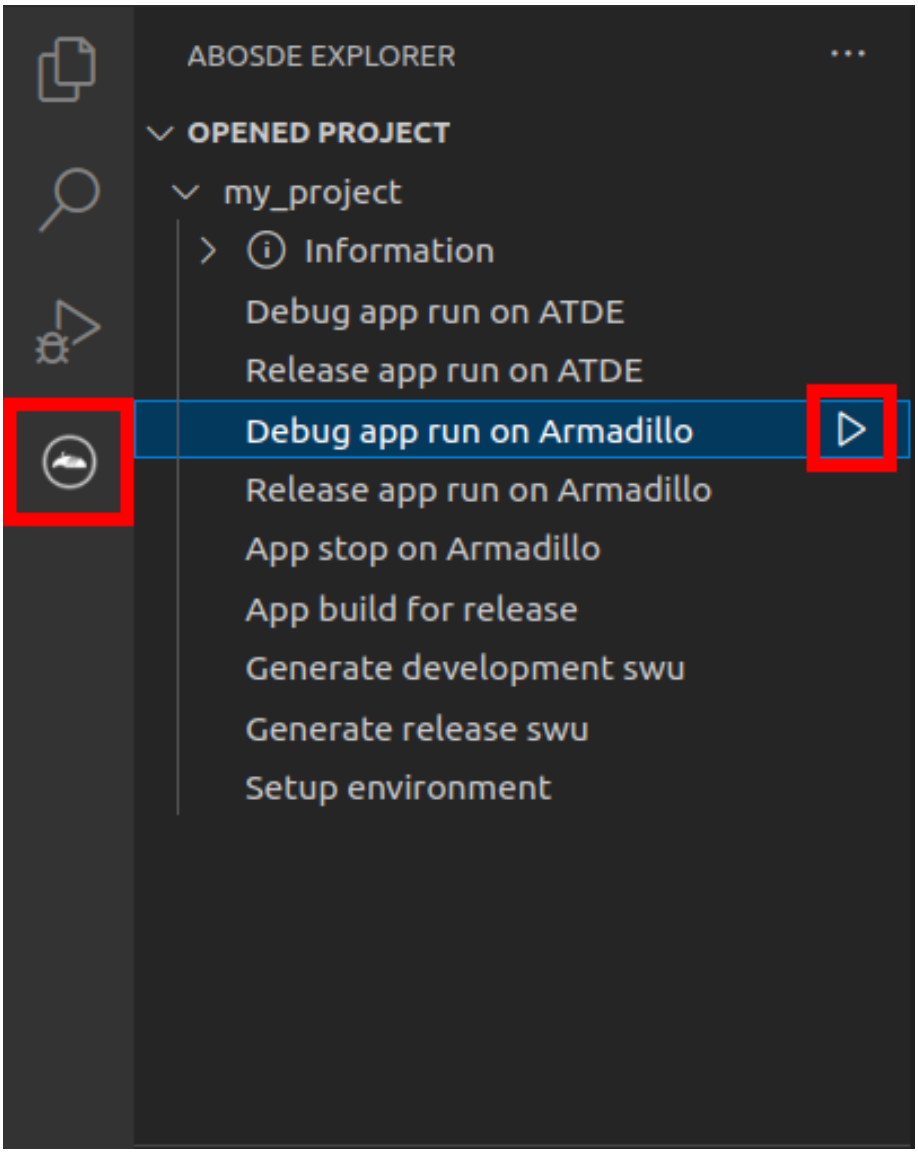


図 3.165 Armadillo 上で Debug モードでビルドしたアプリケーションを実行する

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

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

図 3.166 実行時に表示されるメッセージ

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

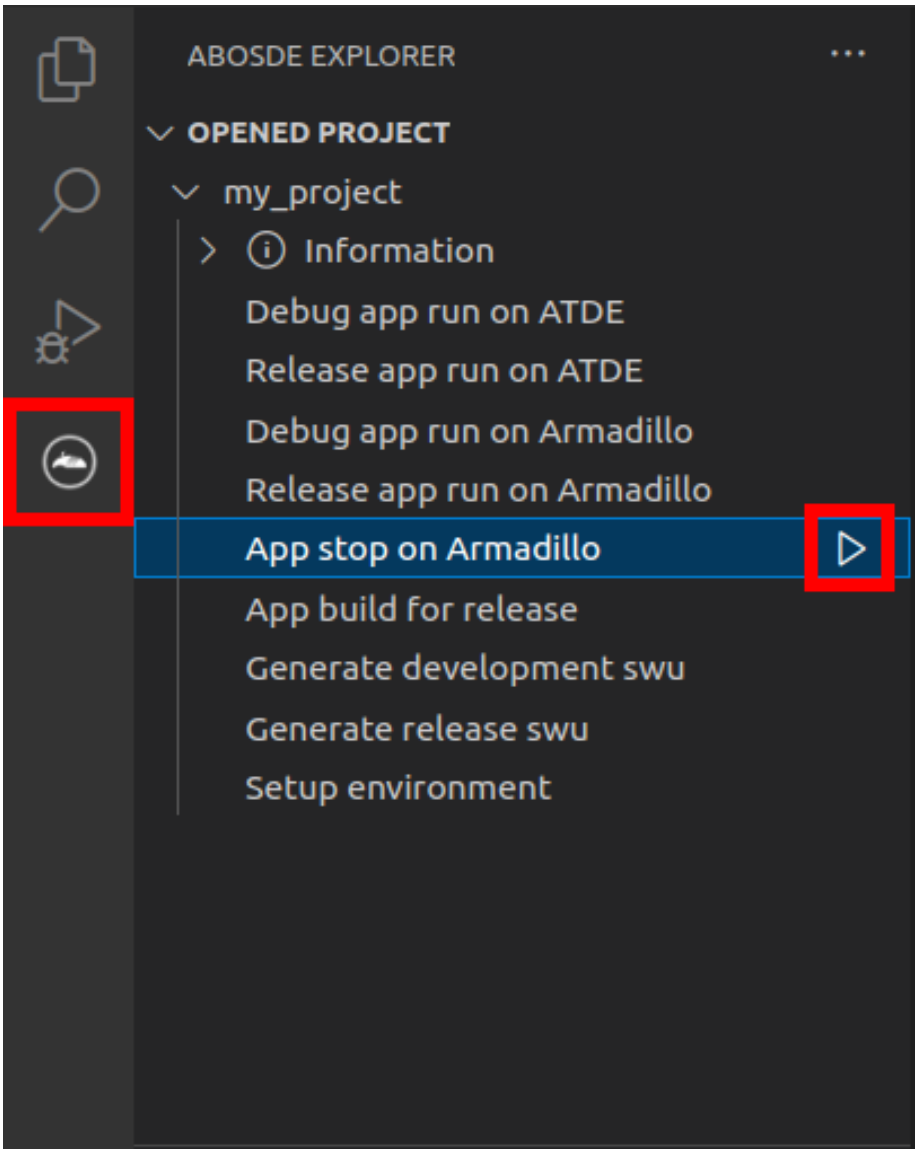


図 3.167 アプリケーションを終了する

また、Release モードでアプリケーションを実行するには、VSCode の左ペインの [my_project] から [Release app run on Armadillo] を実行してください。

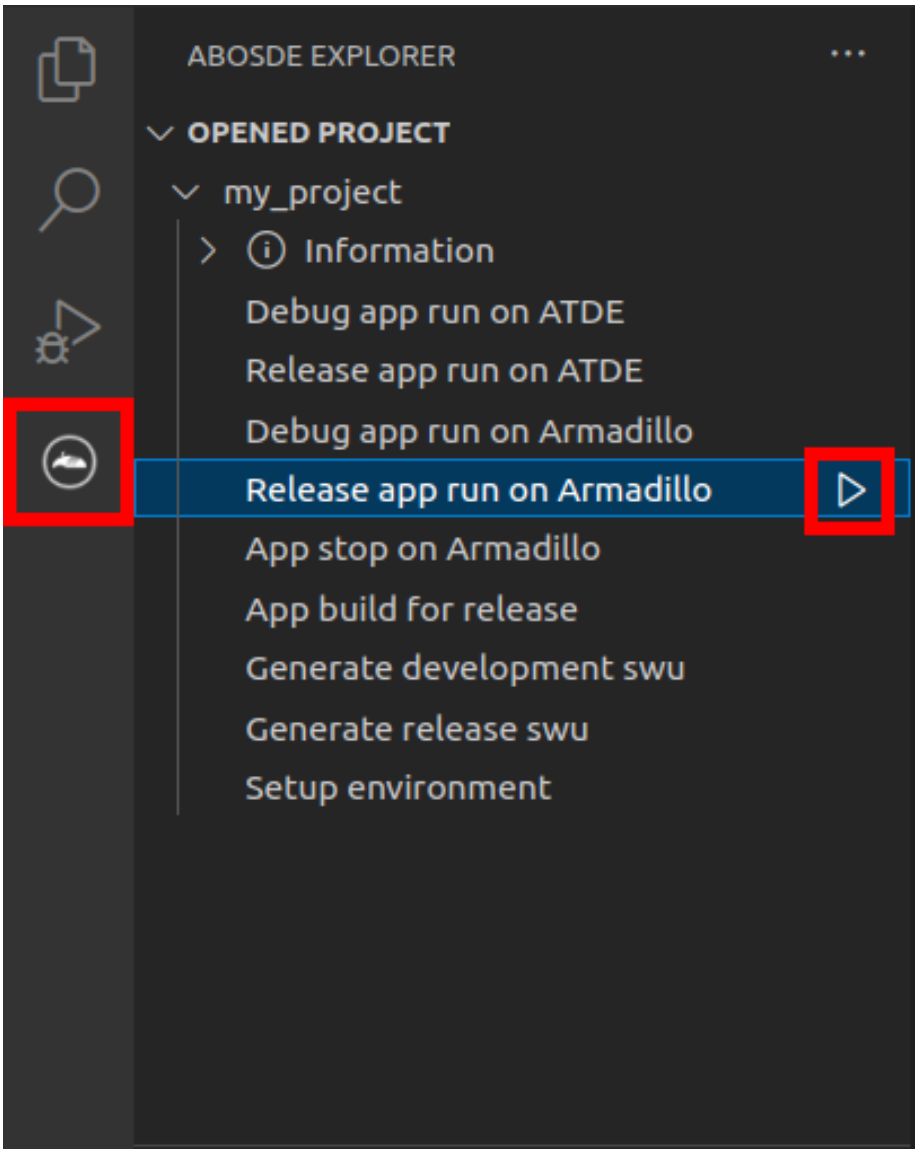


図 3.168 Armadillo 上で Release モードでビルドしたアプリケーションを実行する

3.13.9.3. ホットリロード

アプリケーションのソースコードに修正を加えた後にコンパイルをせずに即座に動作確認をしたい場合、ホットリロード機能を使うことができます。この機能を使うには Debug モードでアプリケーションをビルドしている必要があります。

ホットリロード機能を使うには、アプリケーション実行時に表示される VSCode のターミナルで r を入力してください。その後、以下のようなメッセージが表示され修正が反映されます。

```
Performing hot reload...
Reloaded 1 of 1349 libraries in 2,752ms (compile: 172 ms, reload: 984 ms, reassemble: 1291 ms).
```

図 3.169 ホットリロード機能を使う

3.13.10. 製品への書き込み

リリース版のアプリケーションを含んだ SWU イメージを作成します。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。

VSCode の左ペインの [my_project] から [Generate release swu] を実行すると SWU イメージが作成されます。

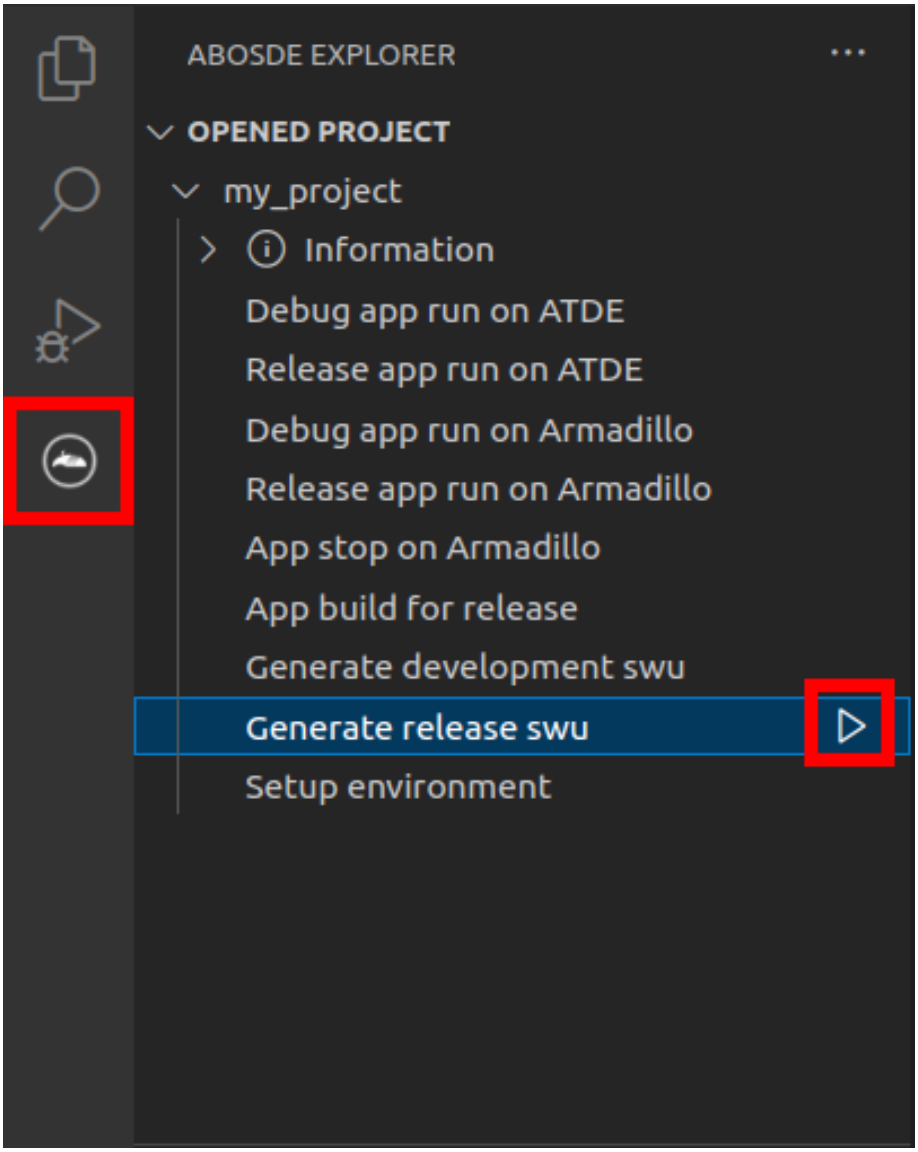


図 3.170 SWU イメージを作成する

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

3.13.11. Armadillo 上のコンテナイメージの削除

`development.swu` または `release.swu` を Armadillo にインストールすることで保存されたコンテナイメージを削除する方法は、 「6.2.3.1. VSCode から実行する」 を参照してください。

3.14. 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。

3.14.1. 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。

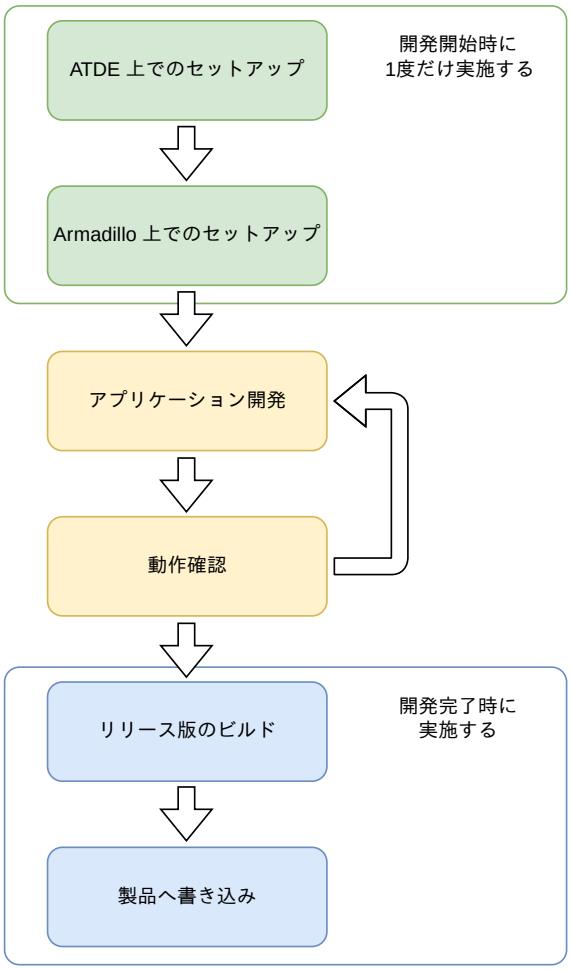


図 3.171 CUI アプリケーション開発の流れ

3.14.2. ATDE 上でのセットアップ

ここでは、開発開始時の ATDE 上でのセットアップ手順について説明します。ATDE をお使いでない場合は、先に 「3.3. 開発の準備」 を参照して ATDE 及び、 VSCode のセットアップを完了してください。

3.14.2.1. プロジェクトの作成

VSCode の左ペインの [G4/X2] から [Shell New Project] を実行し、表示されるディレクトリ選択画面からプロジェクトを保存するディレクトリを選択してください。実行するためには右に表示されている三角形ボタンを押してください。Python の場合は [Python New Project] を実行してください。保存先を選択すると、プロジェクト名を入力するダイアログが表示されるので、任意のプロジェクト名を入力してエンターキーを押してください。ここでは、ホームディレクトリ直下に `my_project` として保存しています。

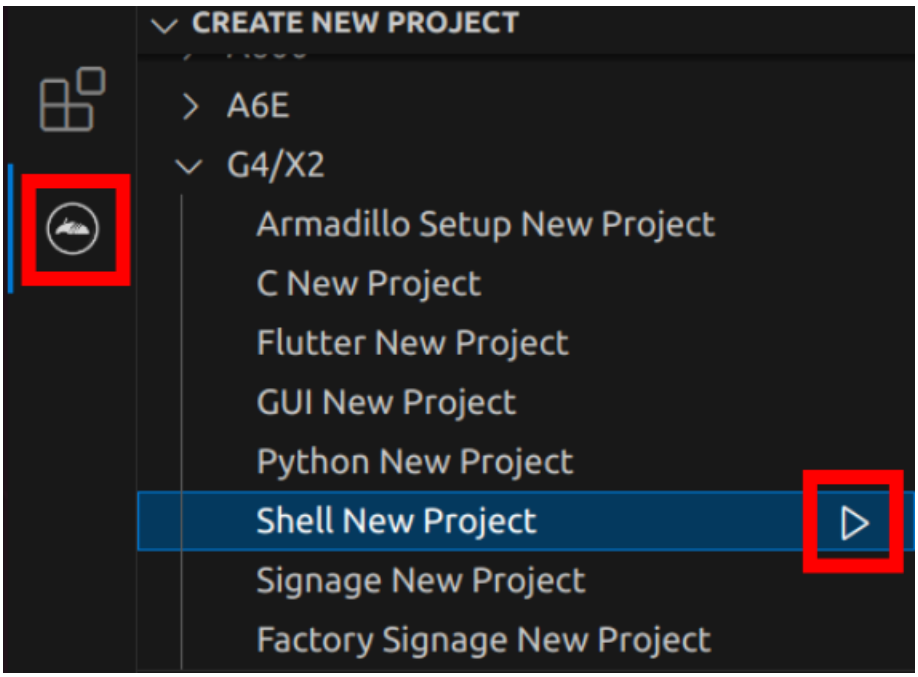


図 3.172 プロジェクトを作成する

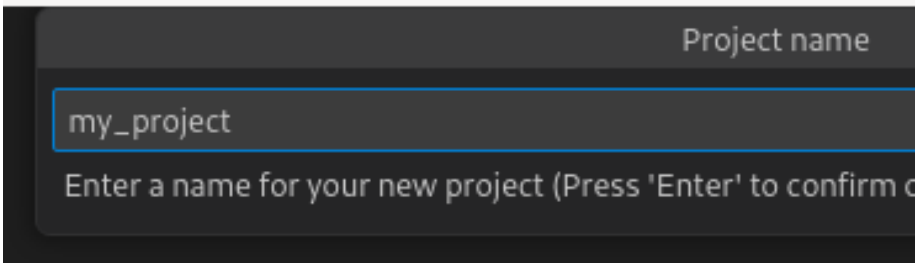


図 3.173 プロジェクト名を入力する

3.14.3. アプリケーション開発

3.14.3.1. VSCode の起動

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

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

図 3.174 VSCode で `my_project` を起動する

3.14.3.2. ディレクトリ構成

プロジェクトには下記のディレクトリがあります。

- **app** : 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** : 設定ファイルです。各ファイルが設定するものは以下のとおりです
  - **app.conf** : コンテナのコンフィグです。記載内容については 「6.2.4. コンテナ起動設定ファイルを作成する」 を参照してください。
  - **app.desc** : SWU イメージを生成するための .desc ファイルです。記載内容については 「6.4. mkswu の .desc ファイルを編集する」 を参照してください。
  - **ssh_config** : Armadillo への ssh 接続に使用します。「3.14.7.2. ssh 接続に使用する IP アドレスの設定」 を参照してください。
- **container** : スクリプトを実行するコンテナの設定ファイルです。 packages.txt に記載されているパッケージがインストールされます。 Dockerfile を直接編集することも可能です。

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。

3.14.3.3. 初期設定

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

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

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

図 3.175 初期設定を行う

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

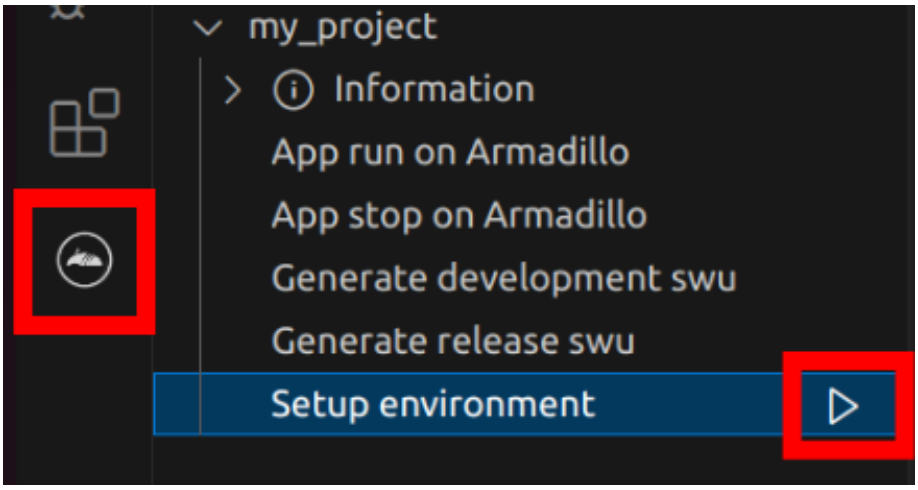


図 3.176 VSCode で初期設定を行う

選択すると、 VSCode の下部に以下のようなターミナルが表示されます。

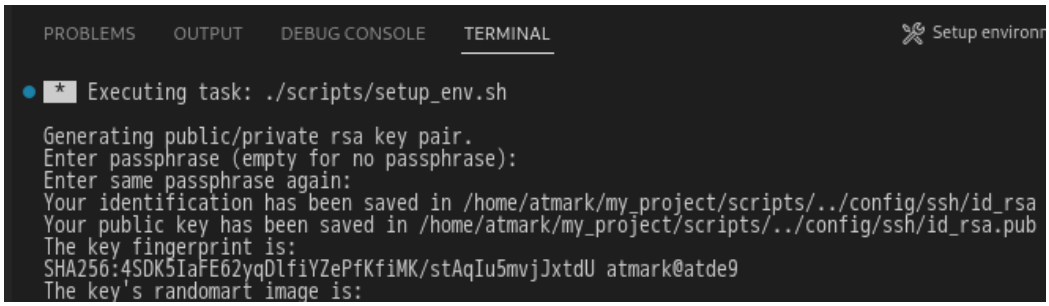


図 3.177 VSCode のターミナル

このターミナル上で以下のように入力してください。

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

Generating public/private ed25519 key pair.
Enter passphrase (empty for no passphrase): ❶
Enter same passphrase again: ❷
Your identification has been saved in /home/atmark/.ssh/id_ed25519_vscode
:(省略)

 *  Terminal will be reused by tasks, press any key to close it.  ❸
```

図 3.178 SSH 用の鍵を生成する

❶ パスフレーズを設定します。設定しない場合は何も入力せず Enter を押します。

❷ 1 でパスフレーズを設定した場合は、確認のため再度入力してください。

❸ ここで何か任意のキーを押すとターミナルが閉じます。

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

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

3.14.3.4. 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 VSCode で行います。VSCode の左ペインの [my_project] から [Generate development swu] を実行します。

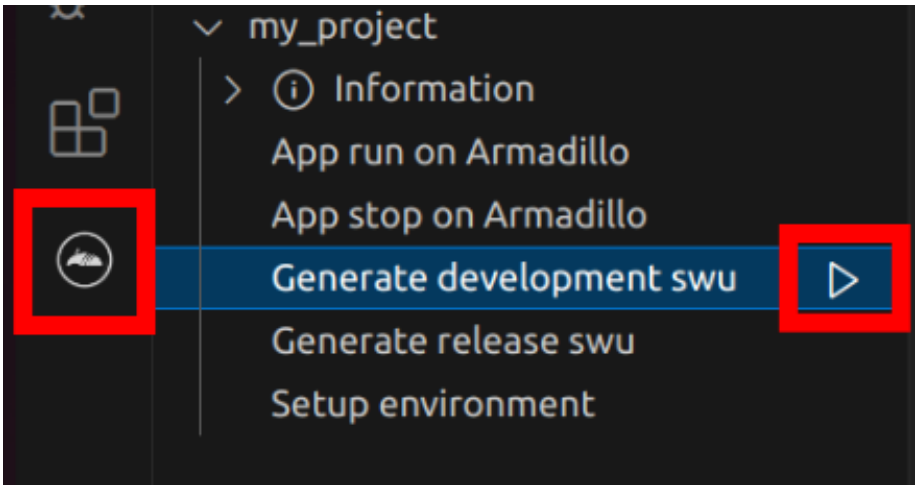


図 3.179 VSCode でコンテナイメージの作成を行う

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

```
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 3.180 コンテナイメージの作成完了

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

3.14.4. コンテナのディストリビューション

使用するコンテナのディストリビューションを以下のとおりです。

ディストリビューション	・debian:bullseye-slim

3.14.5. 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

3.14.6. コンテナ内のファイル一覧表示

「図 3.181. コンテナ内のファイル一覧を表示するタブ」 の赤枠で囲われているタブをクリックすることで、`development.swu` または「3.14.8. 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。

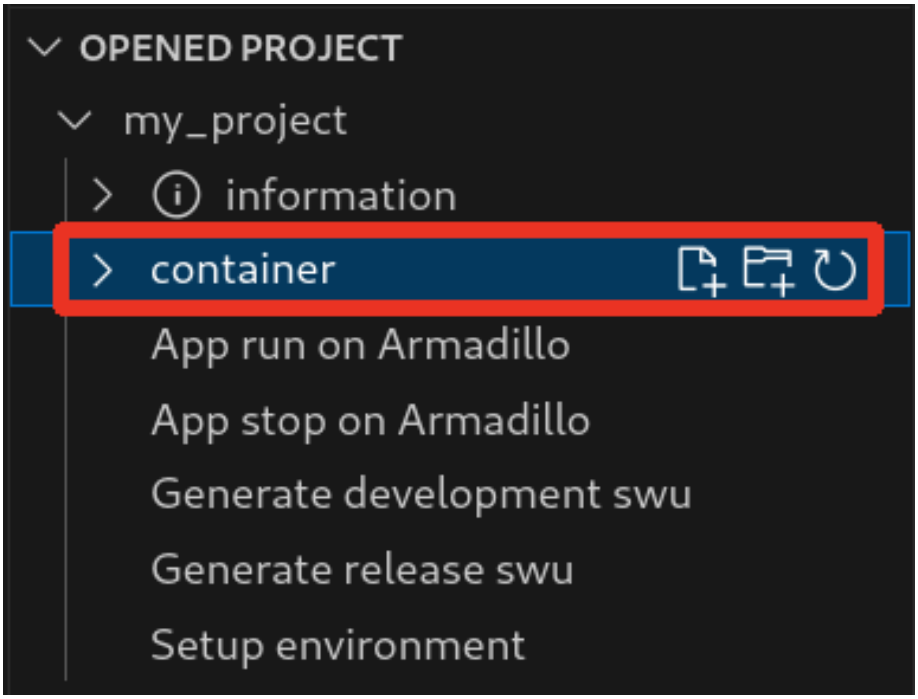


図 3.181 コンテナ内のファイル一覧を表示するタブ

クリック後の表示例を 「図 3.182. コンテナ内のファイル一覧の例」 に示します。

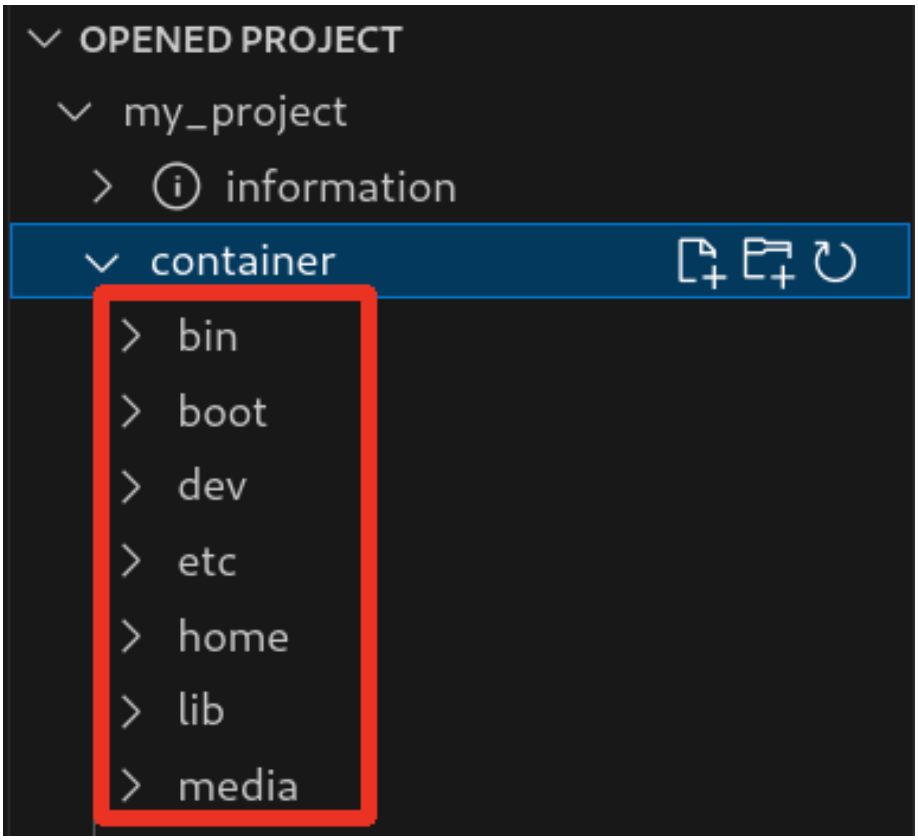


図 3.182 コンテナ内のファイル一覧の例

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

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

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

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

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

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

3.14.6.1. resources ディレクトリについて

「図 3.183. 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。

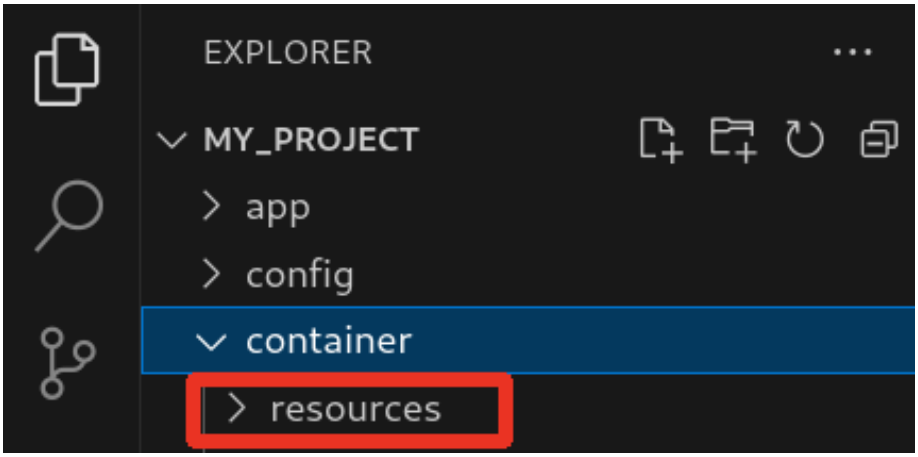


図 3.183 resources ディレクトリ

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

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

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

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

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

3.14.6.2. コンテナ内のファイル一覧の再表示

「図 3.181. 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。

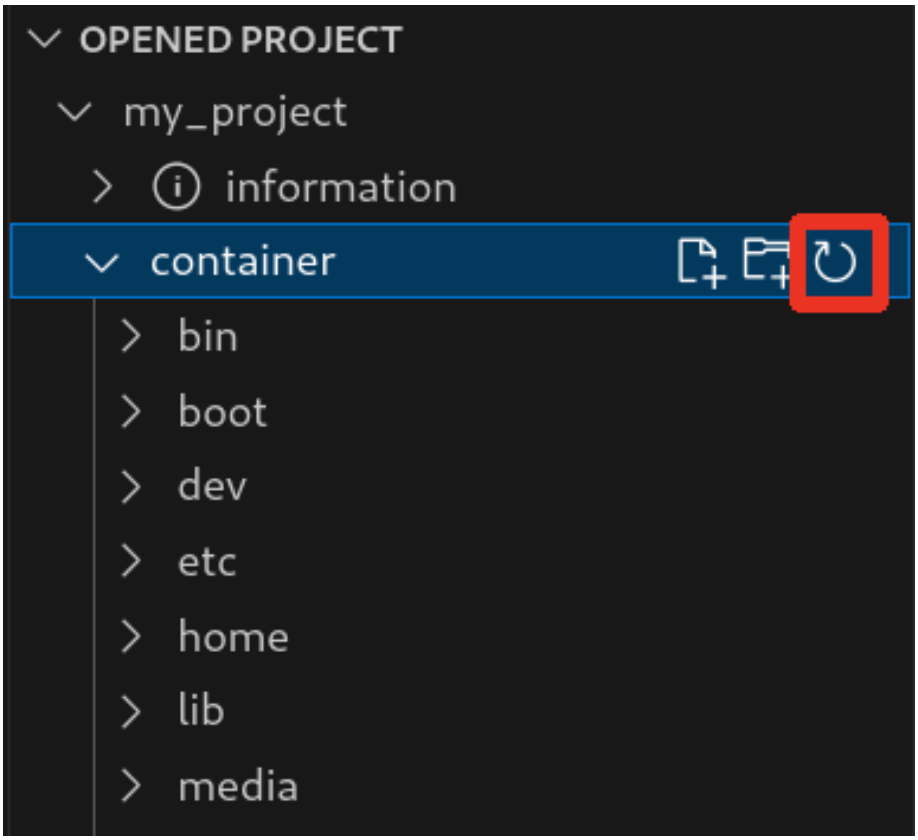


図 3.184 コンテナ内のファイル一覧を再表示するボタン

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

「図 3.185. 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。

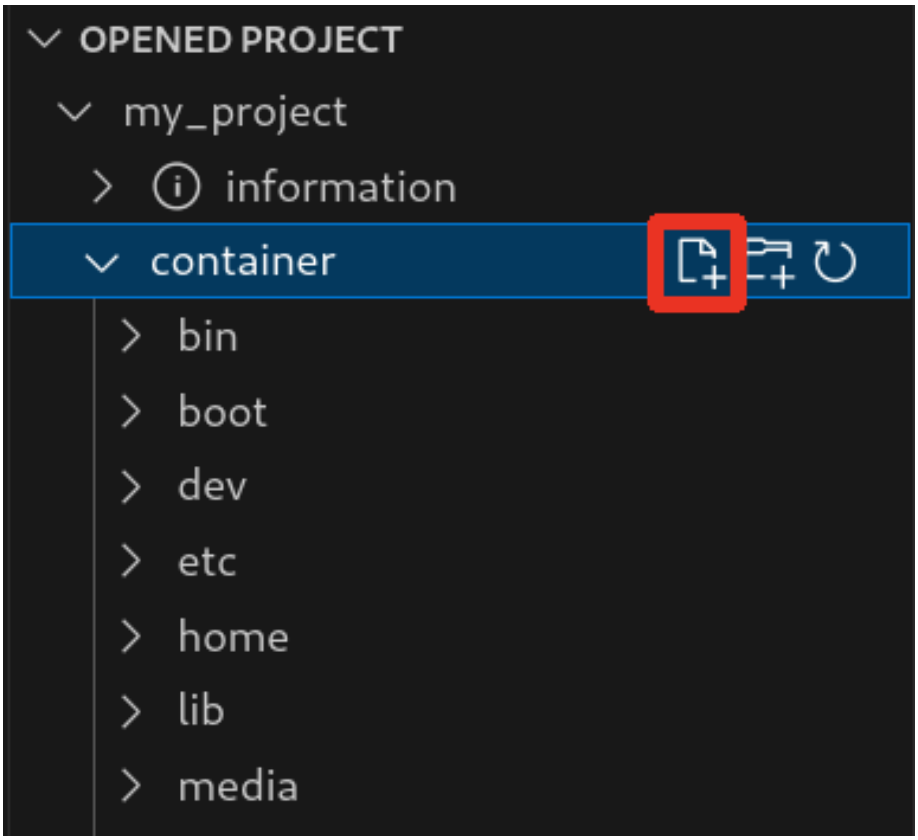


図 3.185 container/resources 下にファイルを追加するボタン

「図 3.186. 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。

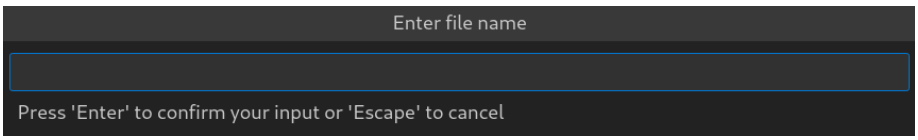


図 3.186 ファイル名を入力

例として、「add_file」というファイル名を入力したとします。

「図 3.187. 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。

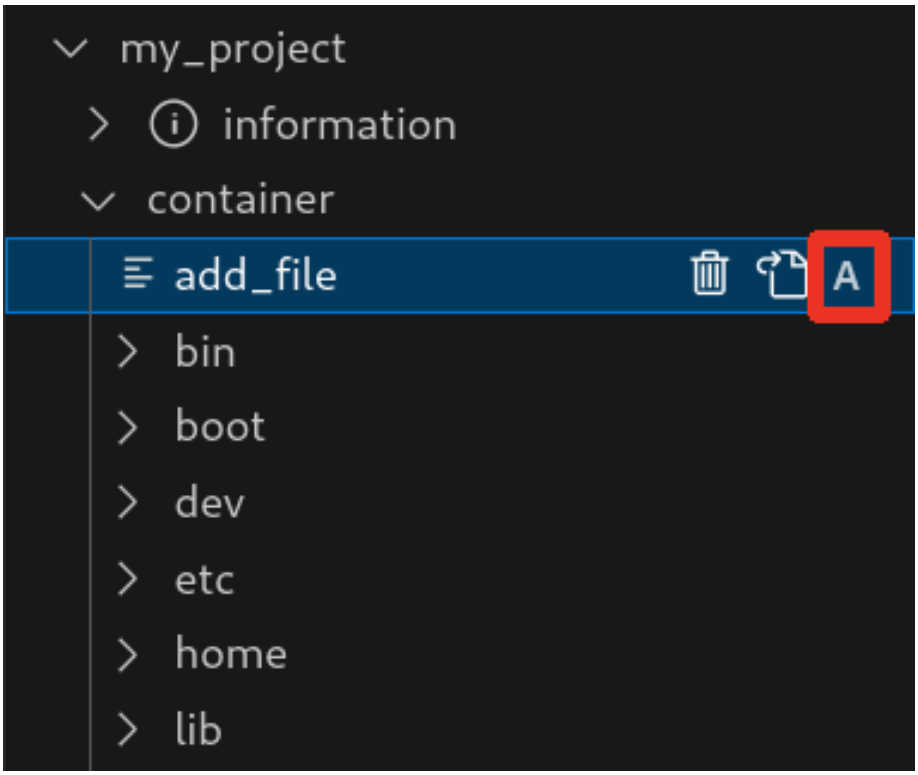


図 3.187 追加されたファイルの表示

また、「図 3.188. 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。

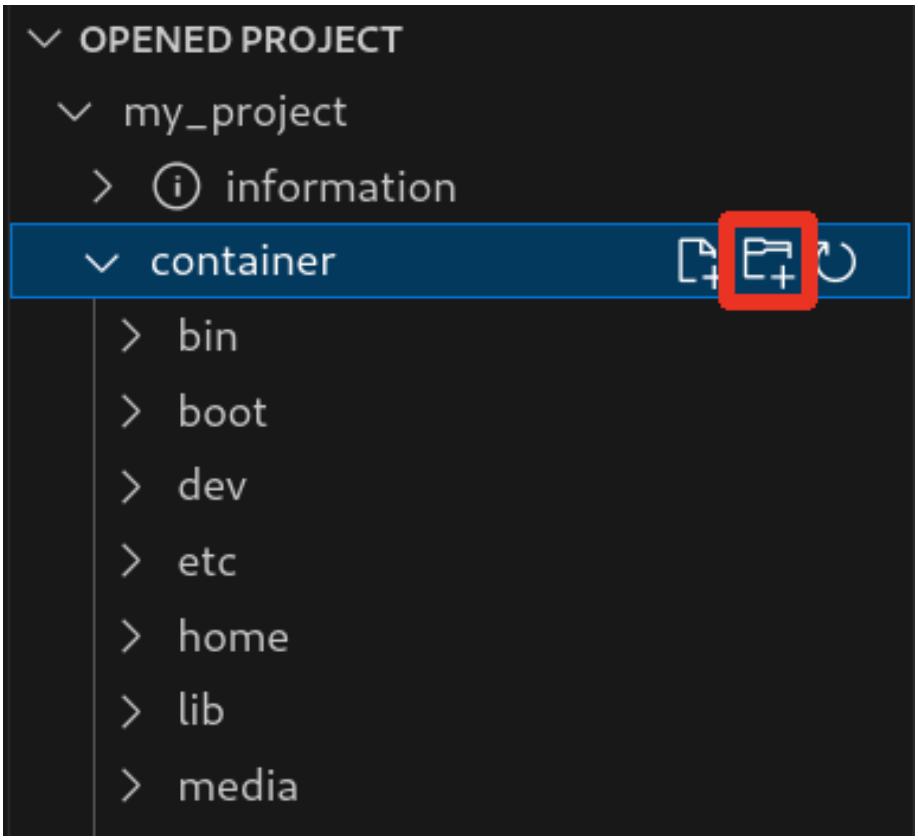


図 3.188 container/resources 下にフォルダーを追加するボタン

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

「図 3.189. 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。

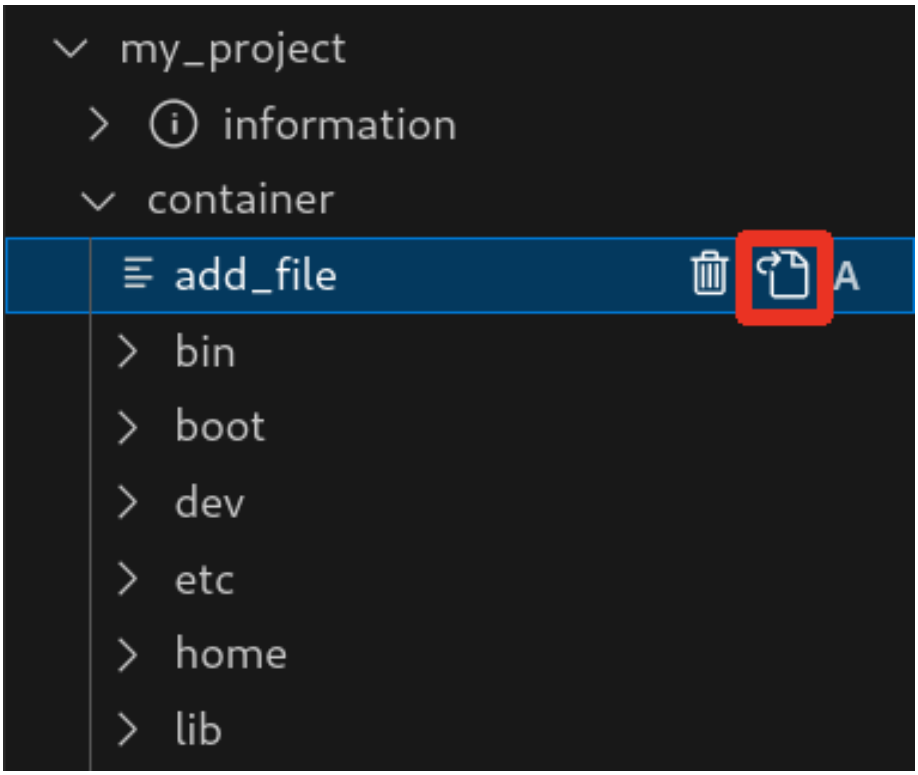


図 3.189 container/resources 下にあるファイルを開くボタン

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

「図 3.189. 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。

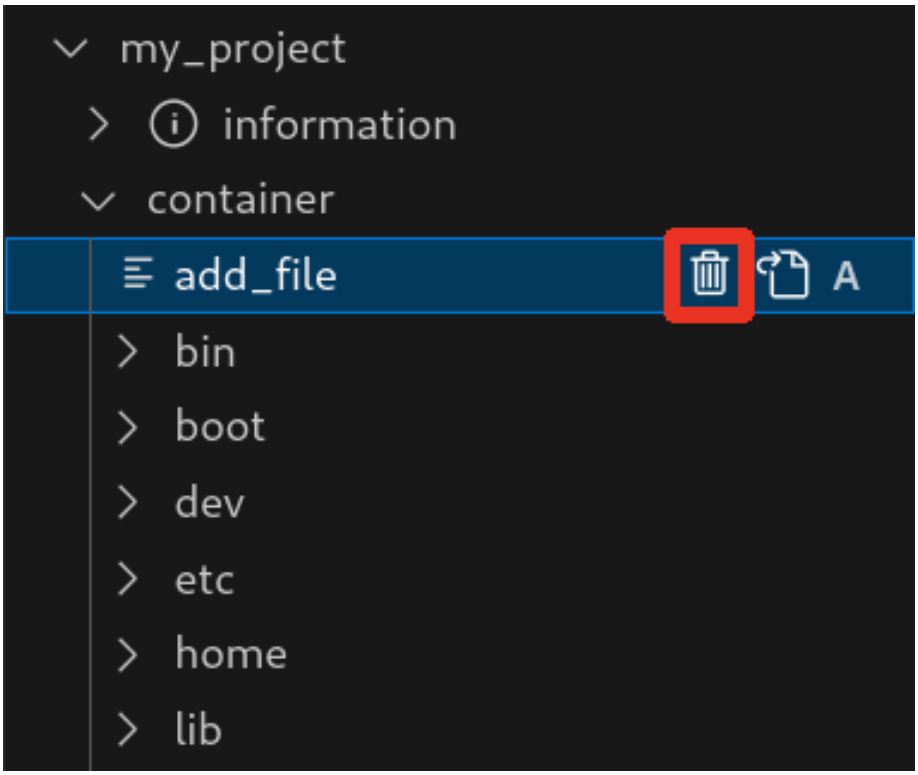


図 3.190 container/resources 下にあるファイルを削除するボタン

ディレクトリも同様に「図 3.189. container/resources 下にあるファイルを開くボタン」 の赤枠で囲われている表記のボタンをクリックすることで削除することができます。

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

「図 3.191. 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。

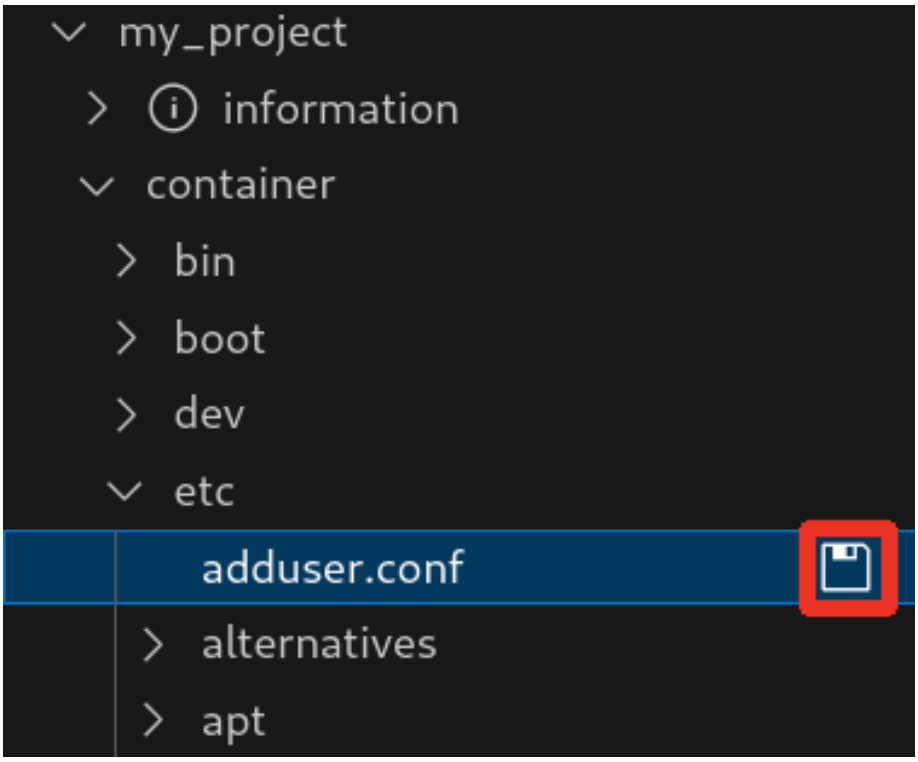


図 3.191 コンテナ内のファイルを container/resources 下に保存するボタン

ファイルが container/resources 下に保存されると、「図 3.192. 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。

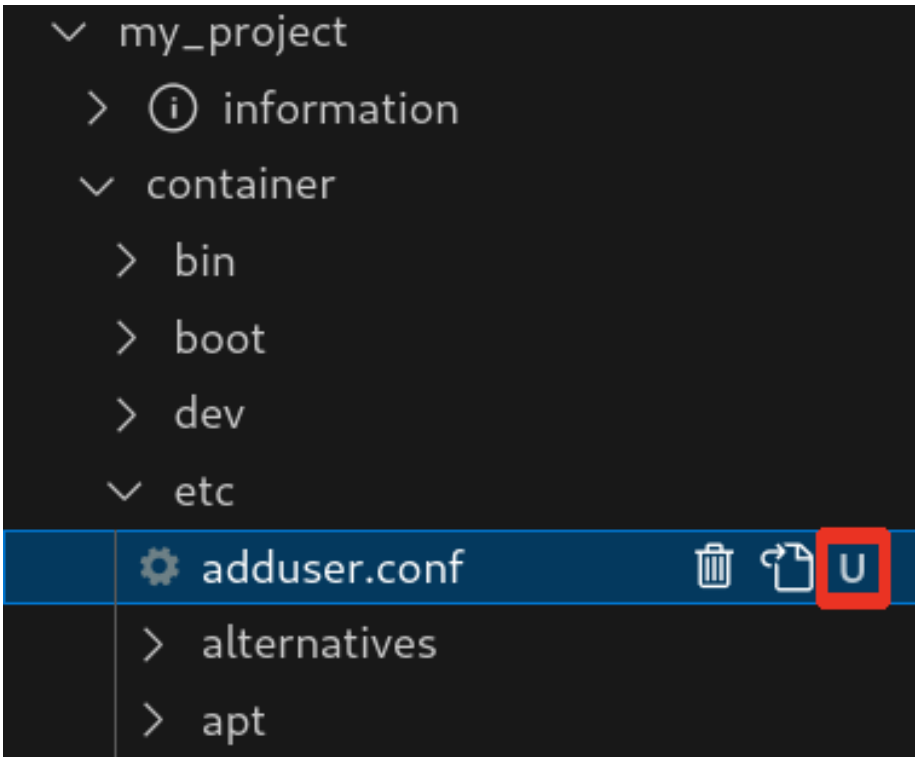


図 3.192 編集前のファイルを示すマーク

container/resources 下にあるファイルを編集して再表示すると、「図 3.193. 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。

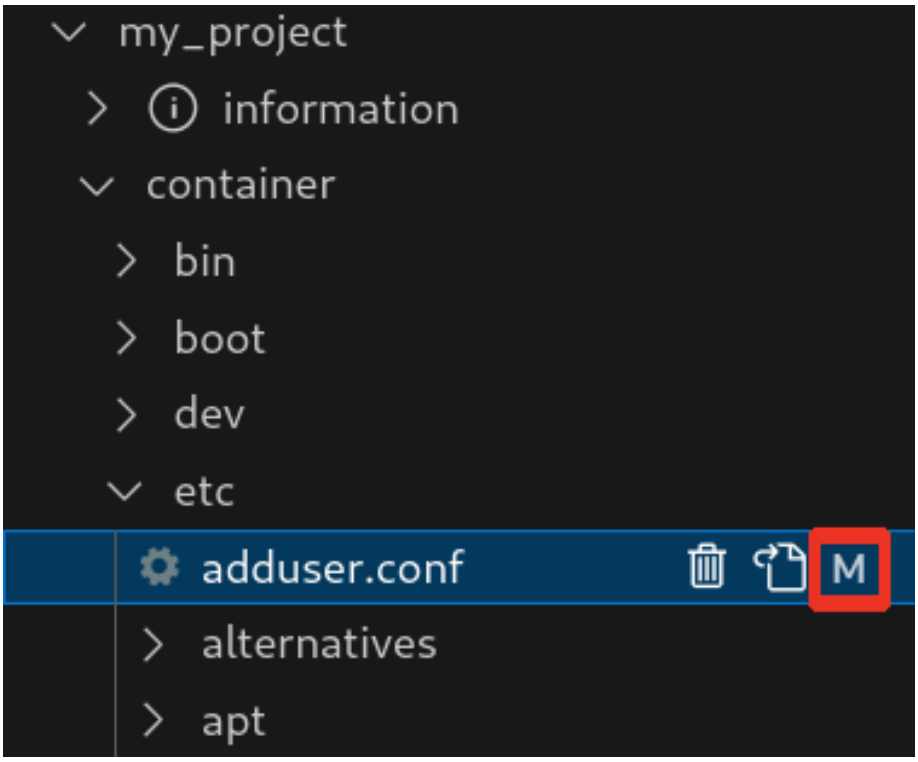


図 3.193 編集後のファイルを示すマーク

3.14.6.7. エラー表示

container/resources 下とコンテナ内にあるファイルまたはディレクトリを比較して、同名でかつファイルの種類が異なる場合、「図 3.194. 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。

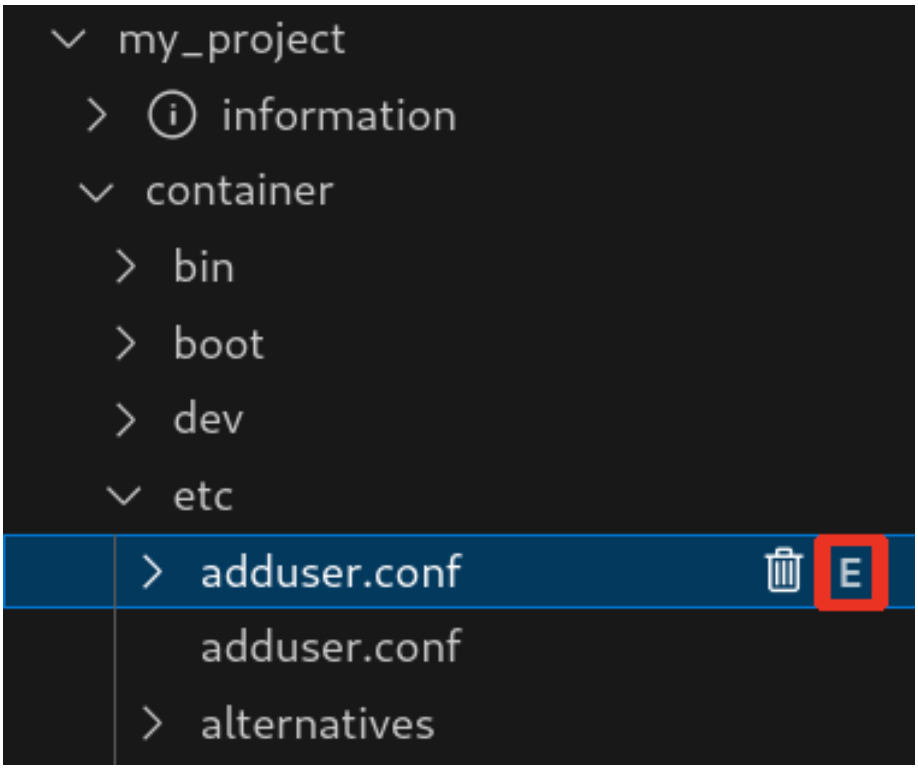


図 3.194 コンテナ内にコピーされないことを示すマーク

3.14.7. Armadillo 上でのセットアップ

3.14.7.1. アプリケーション実行用コンテナイメージのインストール

「3.14.3.4. アプリケーション実行用コンテナイメージの作成」 で作成した `development.swu` を「3.2.3.5. SWU イメージのインストール」 を参照して Armadillo へインストールしてください。

インストール後に自動で Armadillo が再起動します。

3.14.7.2. ssh 接続に使用する IP アドレスの設定

VSCode 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 3.195. 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。

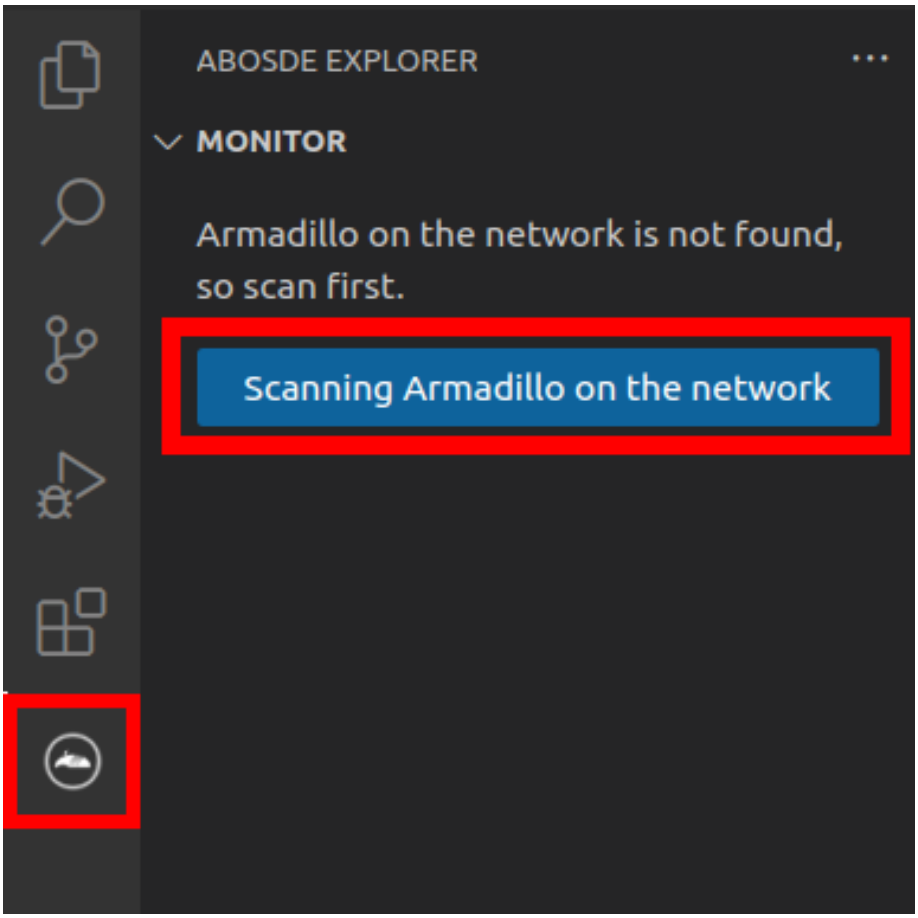


図 3.195 ABOSDE で ローカルネットワーク上の Armadillo をスキャンする

「図 3.196. 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。

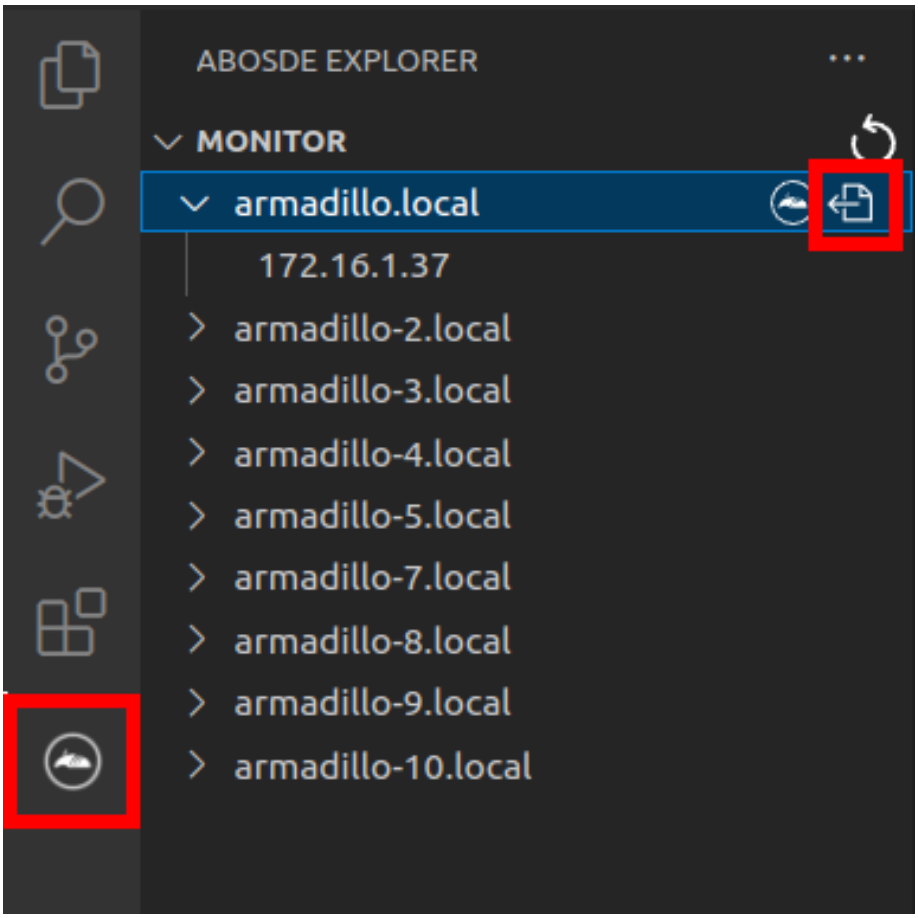


図 3.196 ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する

「図 3.197. 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。

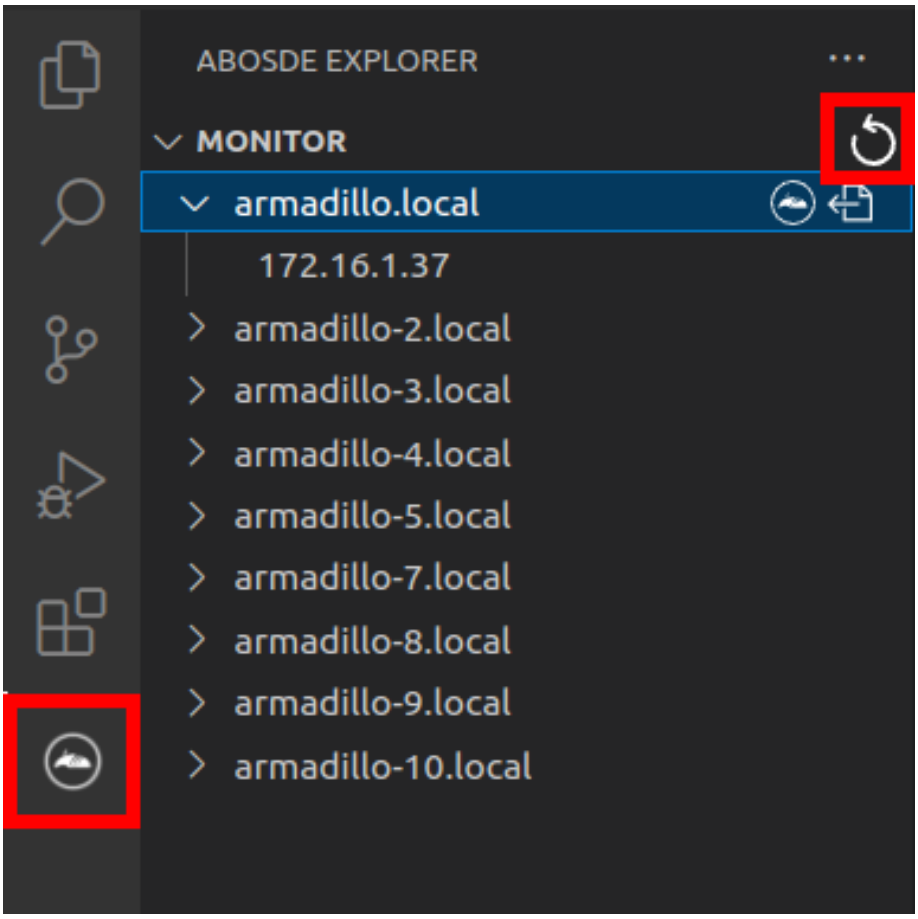


図 3.197 ABOSDE に表示されている Armadillo を更新する

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 3.198 ssh_config を編集する

❶ Armadillo の IP アドレスに置き換えてください。

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

3.14.7.3. アプリケーションの実行

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

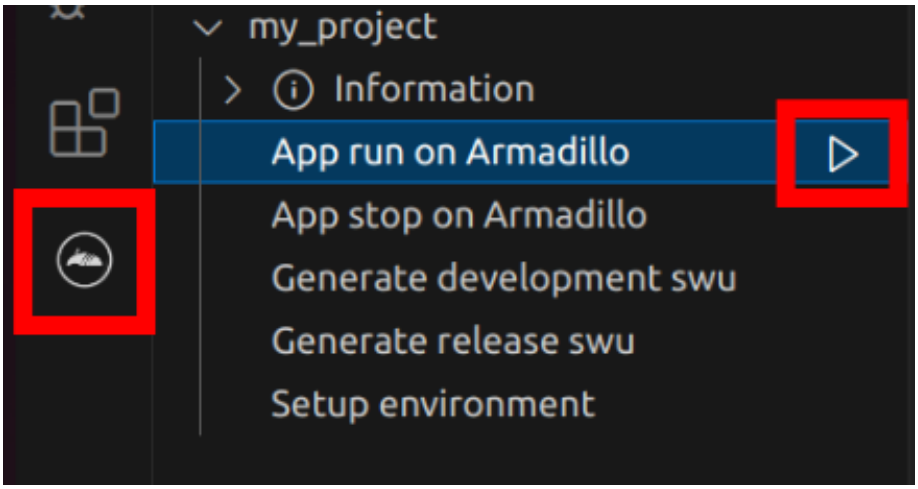


図 3.199 Armadillo 上でアプリケーションを実行する

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

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

図 3.200 実行時に表示されるメッセージ

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

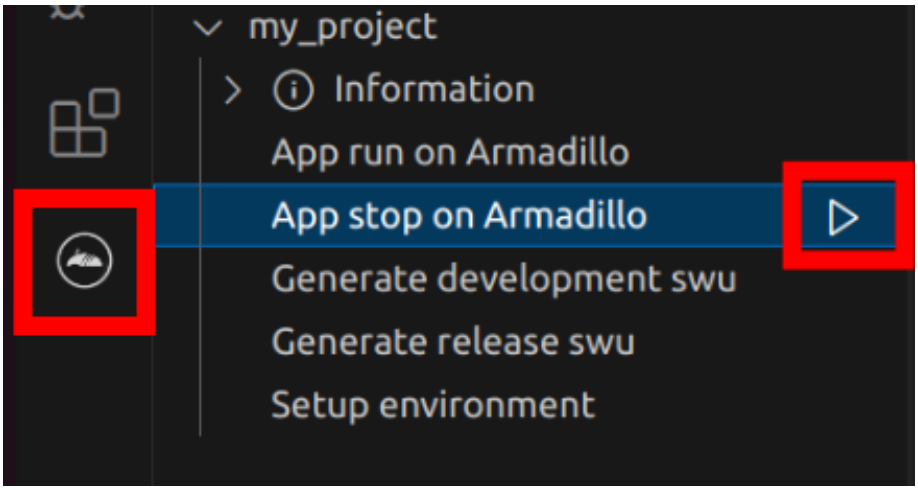


図 3.201 アプリケーションを終了する

3.14.8. 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。

VSCode の左ペインの [my_project] から [Generate release swu] を実行すると、リリース版のアプリケーションを含んだ SWU イメージが作成されます。事前に 「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。

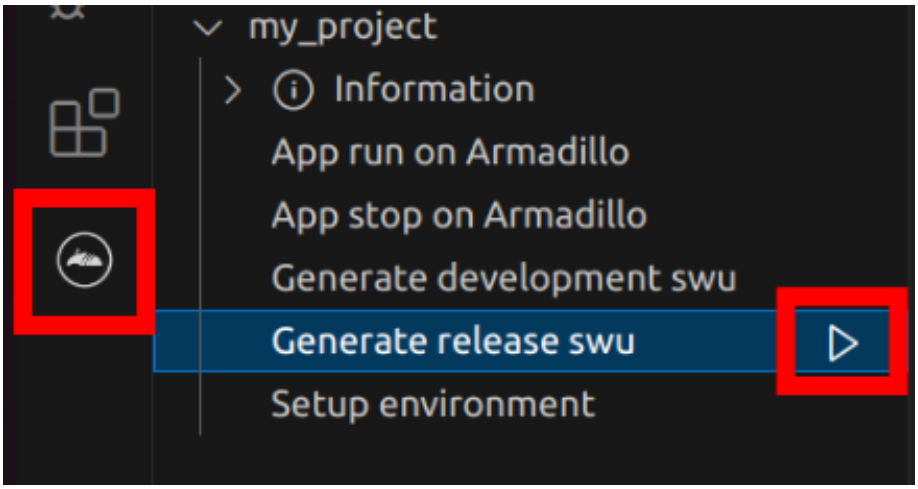


図 3.202 リリース版をビルドする

3.14.9. 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 「3.2.3.5. 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。

3.14.10. Armadillo 上のコンテナイメージの削除

`development.swu` または `release.swu` を Armadillo にインストールすることで保存されたコンテナイメージを削除する方法は、 「6.2.3.1. VSCode から実行する」 を参照してください。

3.15. 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、「3.14. CUI アプリケーションの開発」を参照してください。GUI アプリケーションを開発したい場合、flutter で実現可能ならば、「3.13. GUI アプリケーションの開発」を参照してください。

3.15.1. 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。

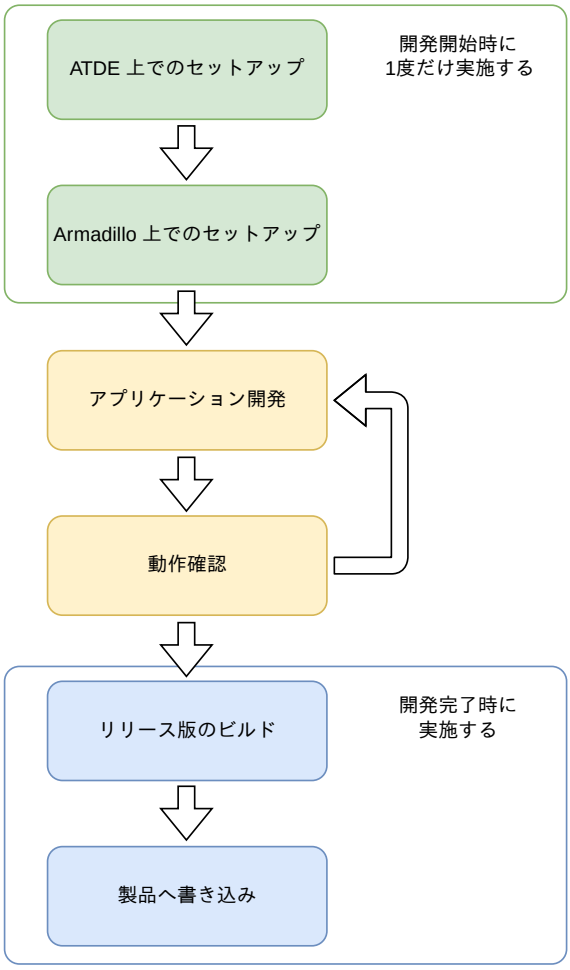


図 3.203 C 言語によるアプリケーション開発の流れ

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 「3.3. 開発の準備」 を参照して ATDE 及び、 VSCode 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

VSCode の左ペインの [G4/X2] 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。ここでは、ホームディレクトリ直下に `my_project` として保存しています。

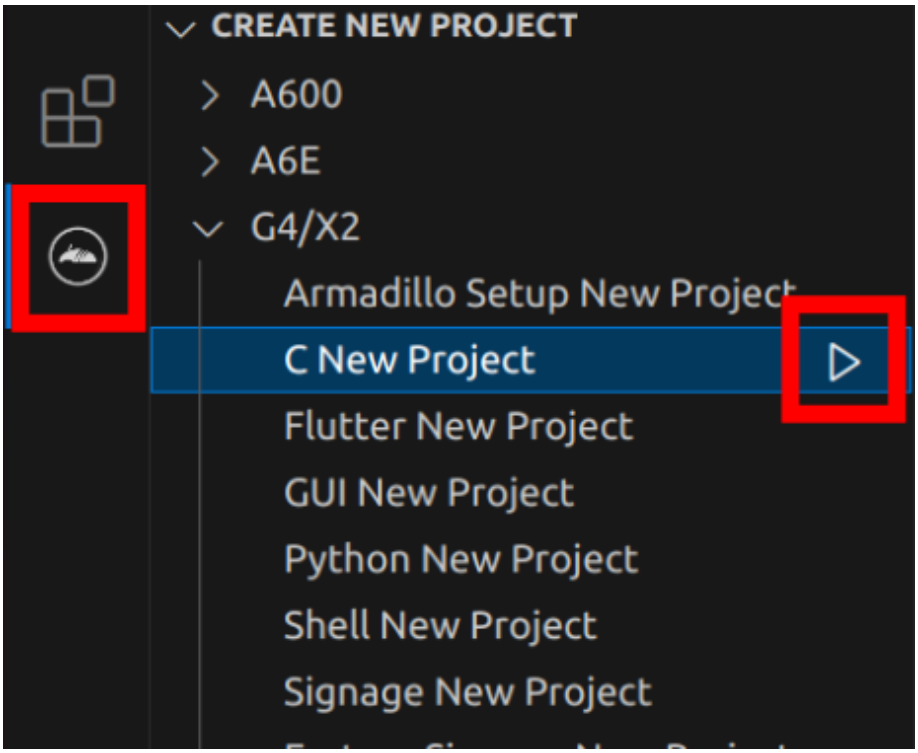


図 3.204 プロジェクトを作成する

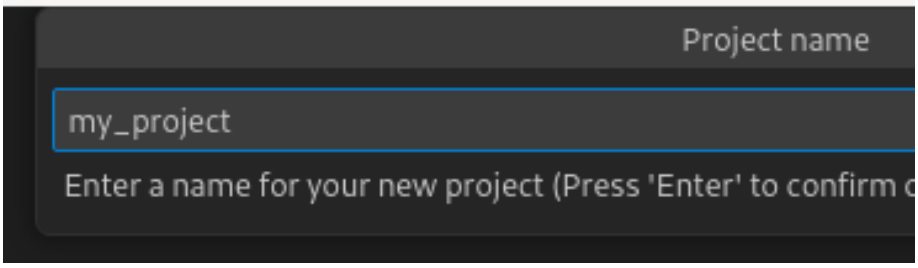


図 3.205 プロジェクト名を入力する

3.15.3. アプリケーション開発

3.15.3.1. VSCode の起動

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

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

図 3.206 VSCode 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**：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** : 設定ファイルです。各ファイルが設定するものは以下のとおりです
  - **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。「3.15.7.2. ssh 接続に使用する IP アドレスの設定」 を参照してください。
- **container** : スクリプトを実行するコンテナの設定ファイルです。 packages.txt に記載されているパッケージがインストールされます。 Dockerfile を直接編集することも可能です。

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。

3.15.3.3. 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 VSCode を起動してください。

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

図 3.207 初期設定を行う

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

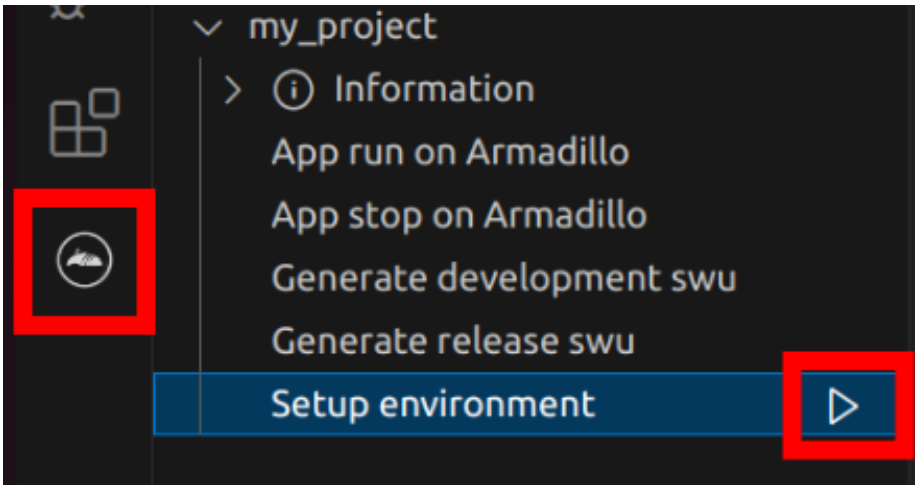


図 3.208 VSCode で初期設定を行う

選択すると、 VSCode の下部に以下のようなターミナルが表示されます。

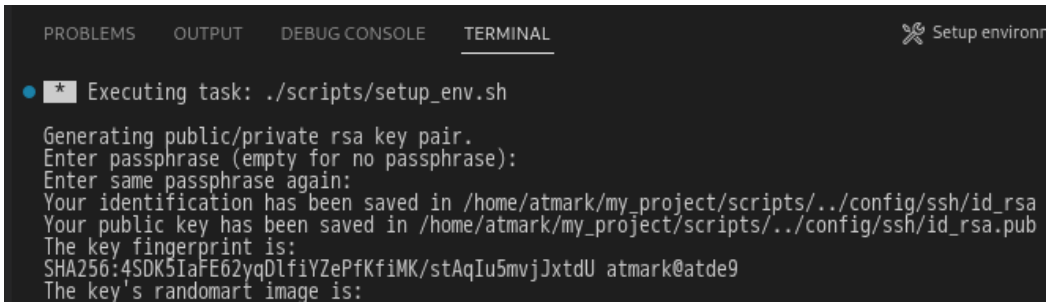


図 3.209 VSCode のターミナル

このターミナル上で以下のように入力してください。

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

Generating public/private ed25519 key pair.
Enter passphrase (empty for no passphrase): ❶
Enter same passphrase again: ❷
Your identification has been saved in /home/atmark/.ssh/id_ed25519_vscode
:(省略)

 *  Terminal will be reused by tasks, press any key to close it.  ❸
```

図 3.210 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. 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 3.211. 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 3.211 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。

3.15.3.5. ABOSDE での開発における制約

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

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

3.15.3.6. 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、 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 VSCode で行います。VSCode の左ペインの [my_project] から [Generate development swu] を実行します。

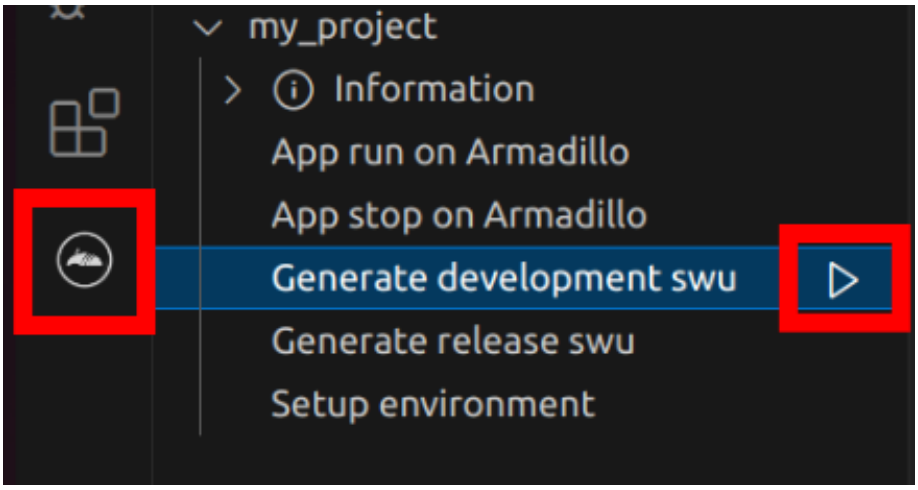


図 3.212 VSCode でコンテナイメージの作成を行う

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

```
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 3.213 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。

3.15.4. コンテナのディストリビューション

使用するコンテナのディストリビューションを以下のとおりです。

ディストリビューション	・ debian:bullseye-slim

3.15.5. コンテナ内のファイル一覧表示

「図 3.214. 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。

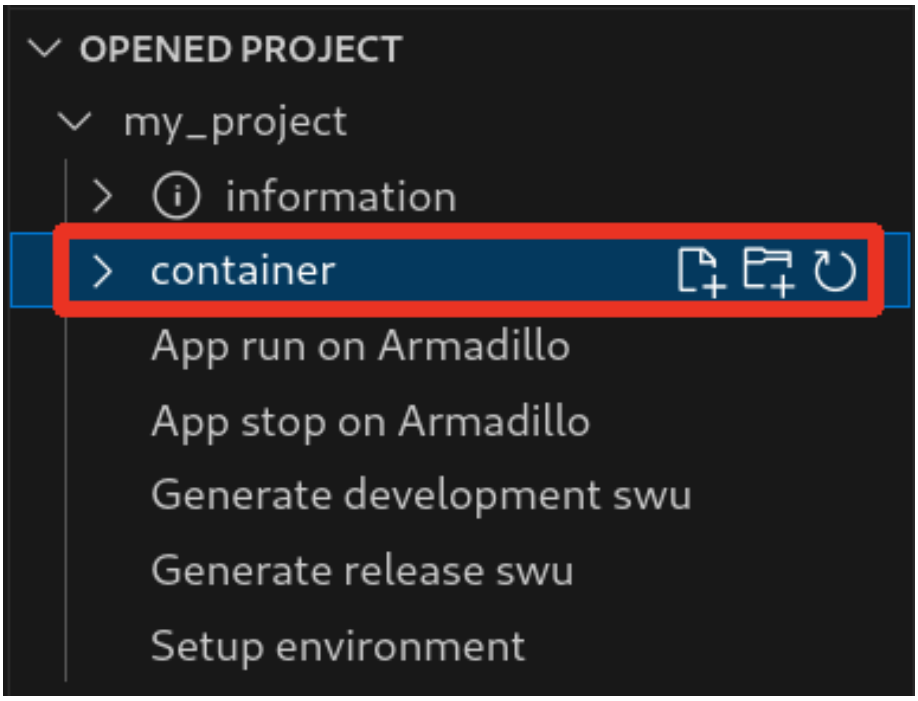


図 3.214 コンテナ内のファイル一覧を表示するタブ

クリック後の表示例を 「図 3.215. コンテナ内のファイル一覧の例」 に示します。

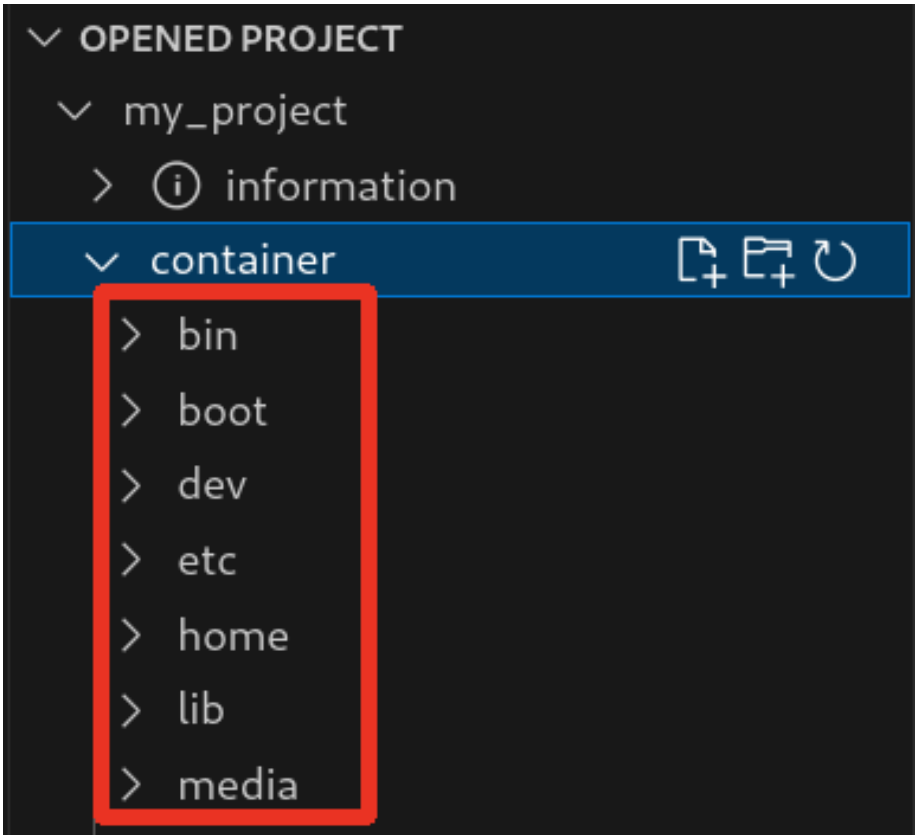


図 3.215 コンテナ内のファイル一覧の例

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 3.216. 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。

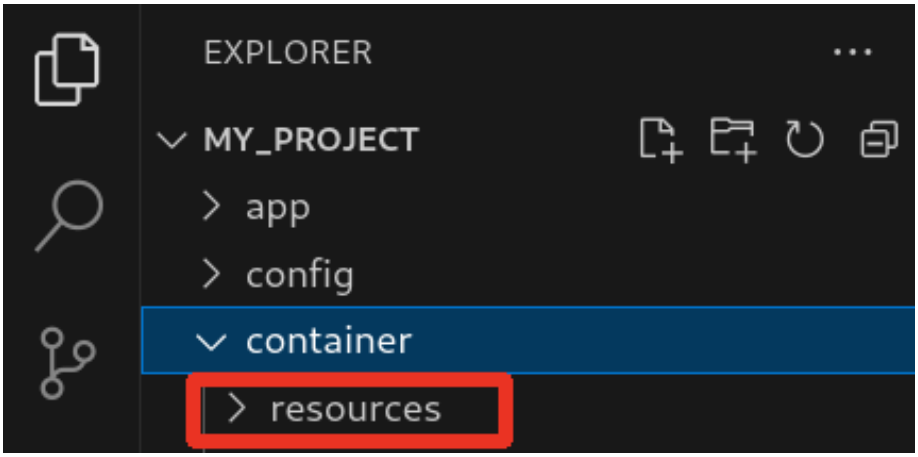


図 3.216 resources ディレクトリ

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 3.214. 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。

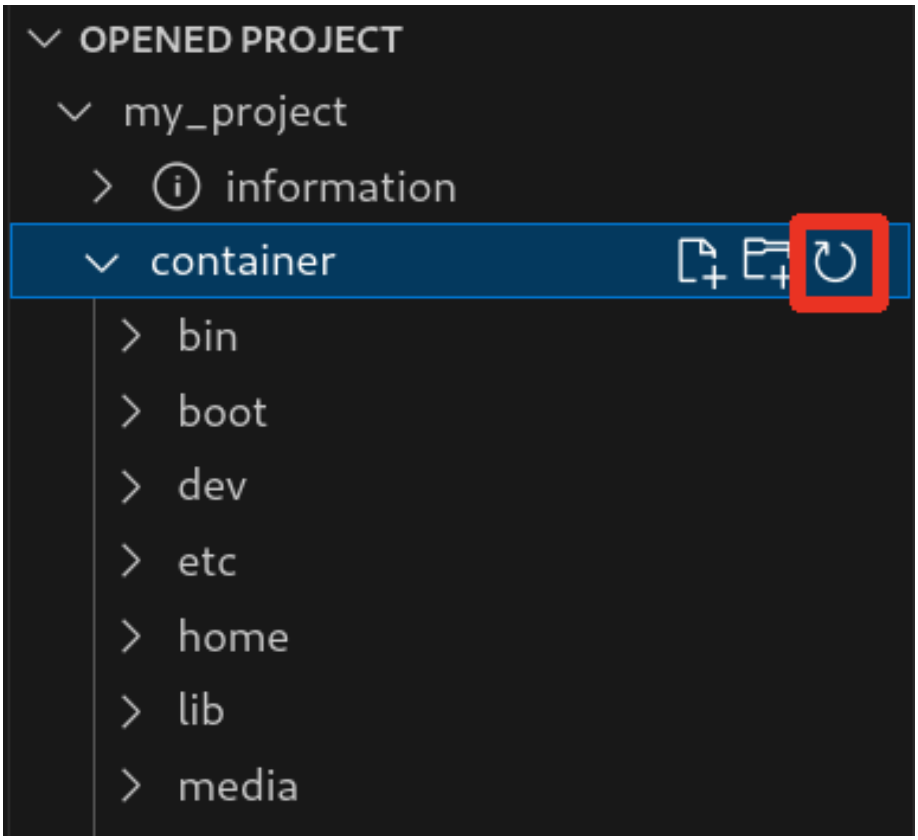


図 3.217 コンテナ内のファイル一覧を再表示するボタン

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

「図 3.218. 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。

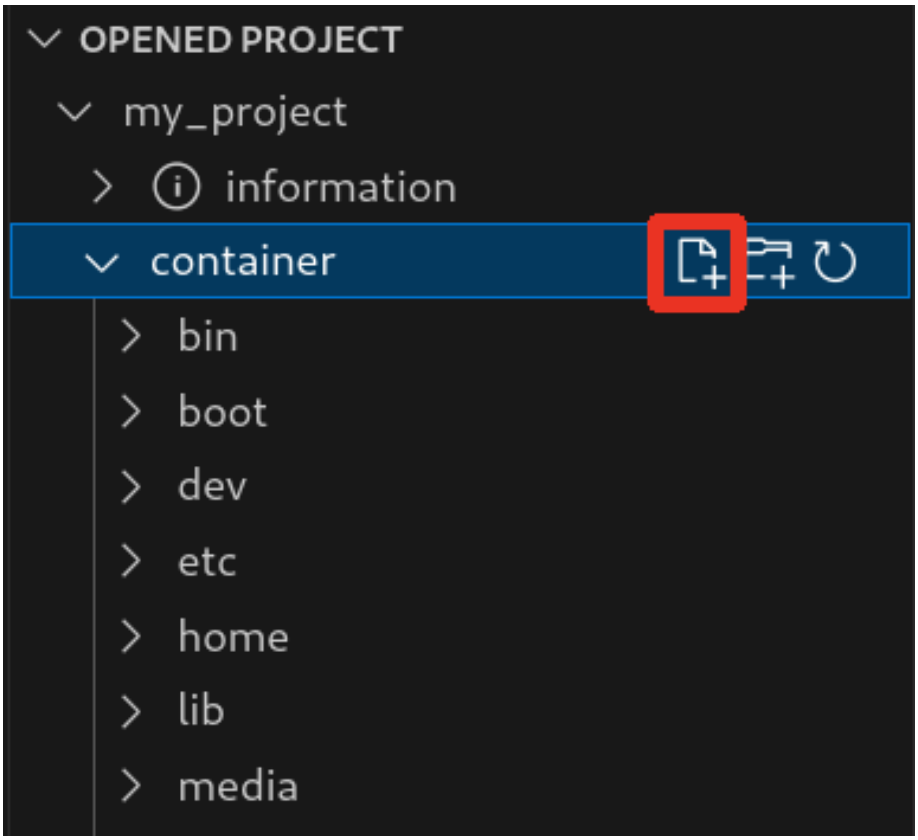


図 3.218 container/resources 下にファイルを追加するボタン

「図 3.219. 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。

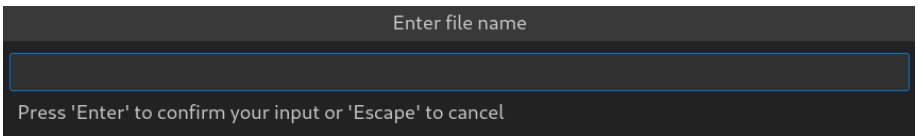


図 3.219 ファイル名を入力

例として、「add_file」というファイル名を入力したとします。「図 3.220. 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。

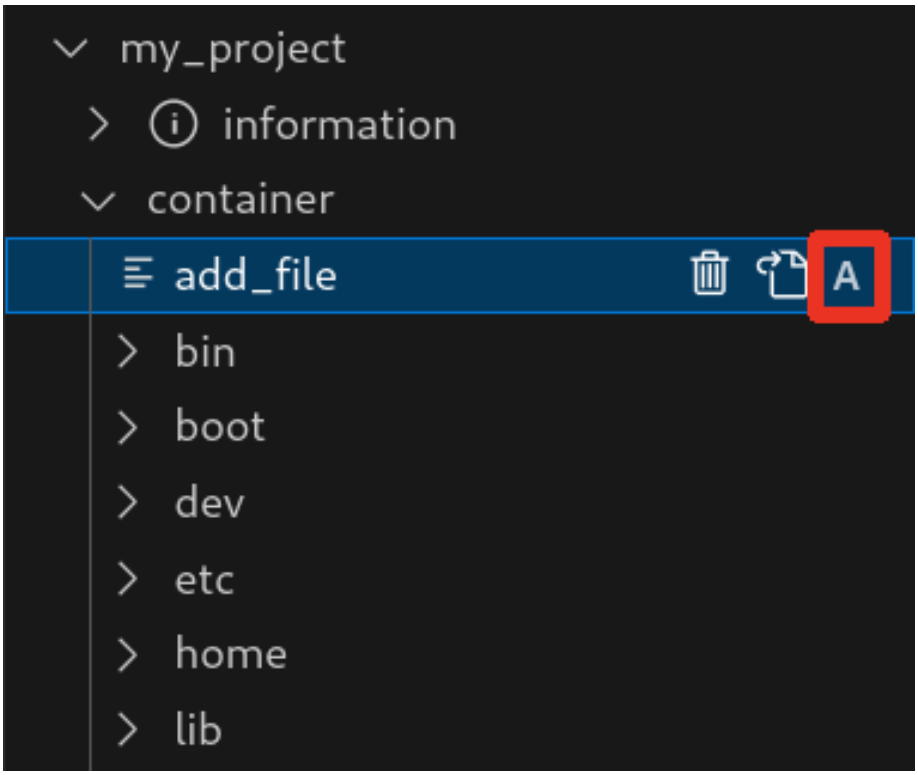


図 3.220 追加されたファイルの表示

また、「図 3.221. 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。

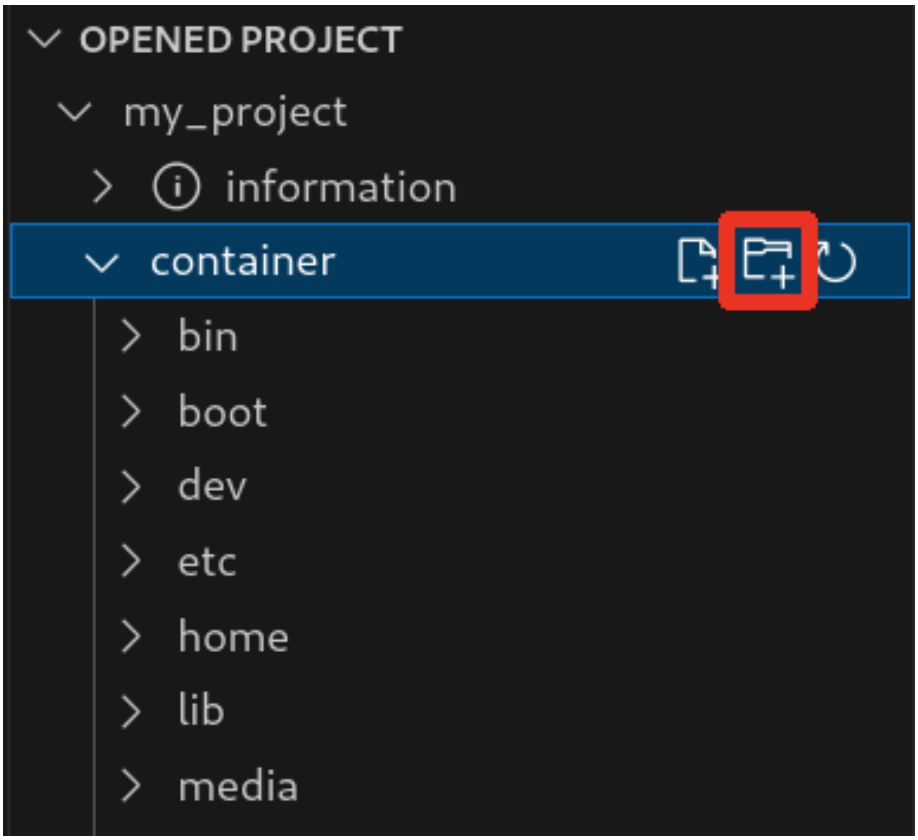


図 3.221 container/resources 下にフォルダーを追加するボタン

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

「図 3.222. 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。

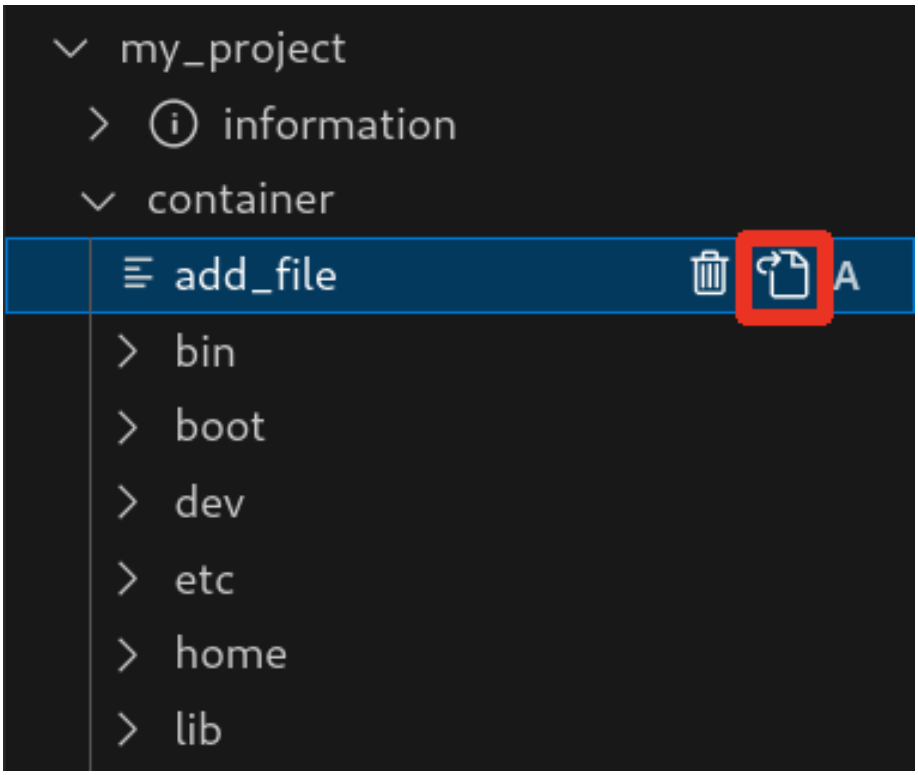


図 3.222 container/resources 下にあるファイルを開くボタン

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 3.222. 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。

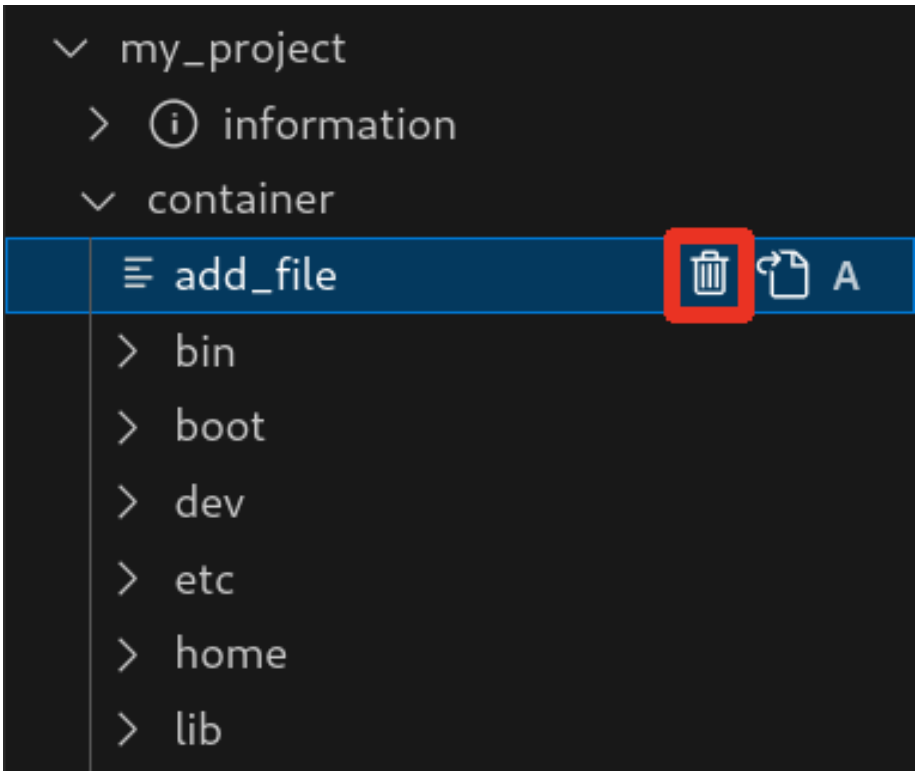


図 3.223 container/resources 下にあるファイルを削除するボタン

ディレクトリも同様に「図 3.222. 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 3.224. 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。

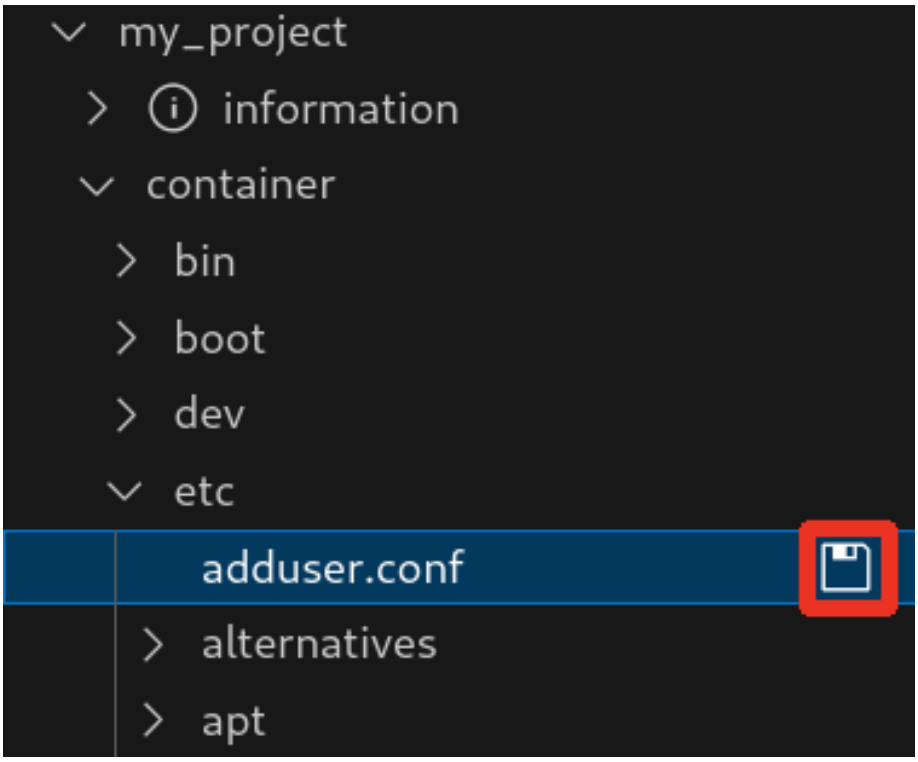


図 3.224 コンテナ内のファイルを container/resources 下に保存するボタン

ファイルが container/resources 下に保存されると、「図 3.225. 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。

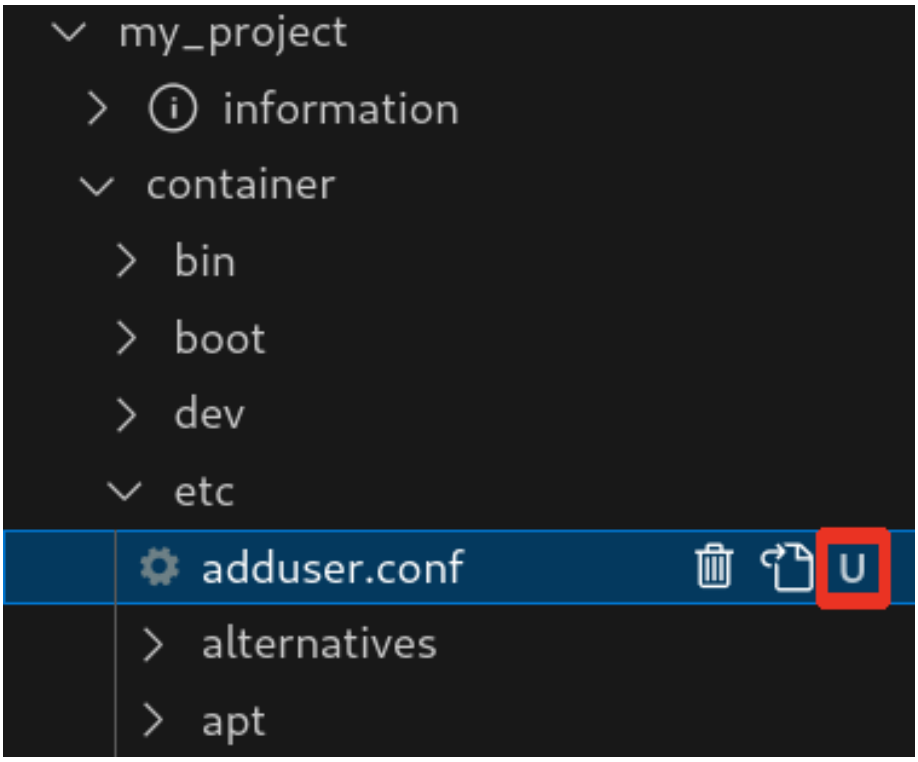


図 3.225 編集前のファイルを示すマーク

container/resources 下にあるファイルを編集して再表示すると、「図 3.226. 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。

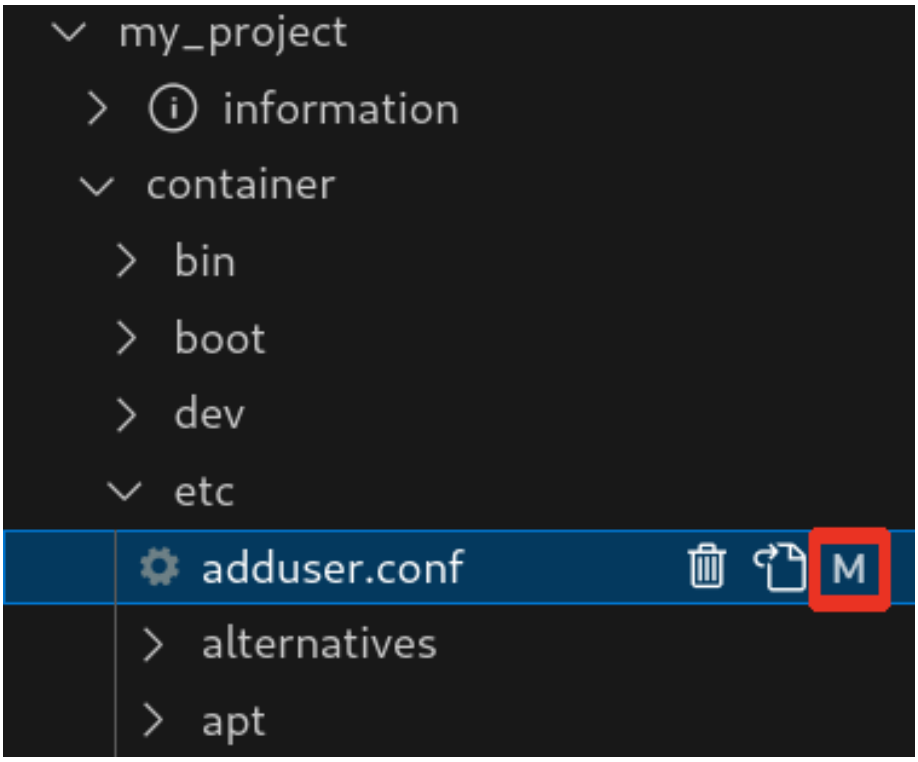


図 3.226 編集後のファイルを示すマーク

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 3.227. 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。

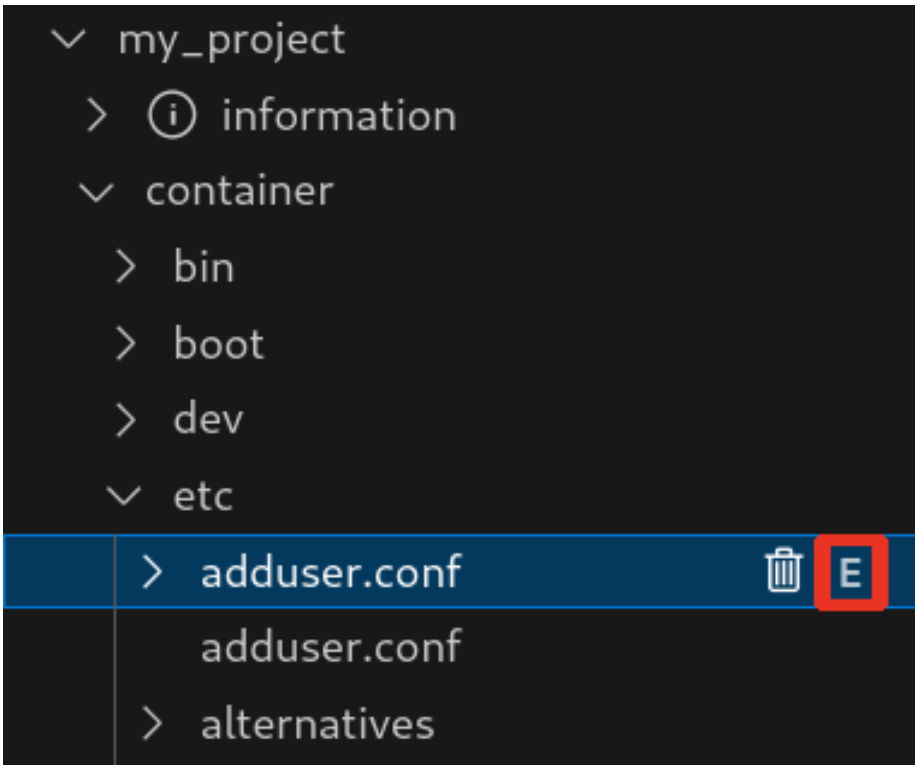


図 3.227 コンテナ内にコピーされないことを示すマーク

3.15.6. Armadillo 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

3.15.7. Armadillo 上でのセットアップ

3.15.7.1. アプリケーション実行用コンテナイメージのインストール

「3.15.3.6. 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「3.2.3.5. 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。

3.15.7.2. ssh 接続に使用する IP アドレスの設定

VSCode 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 3.228. 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。

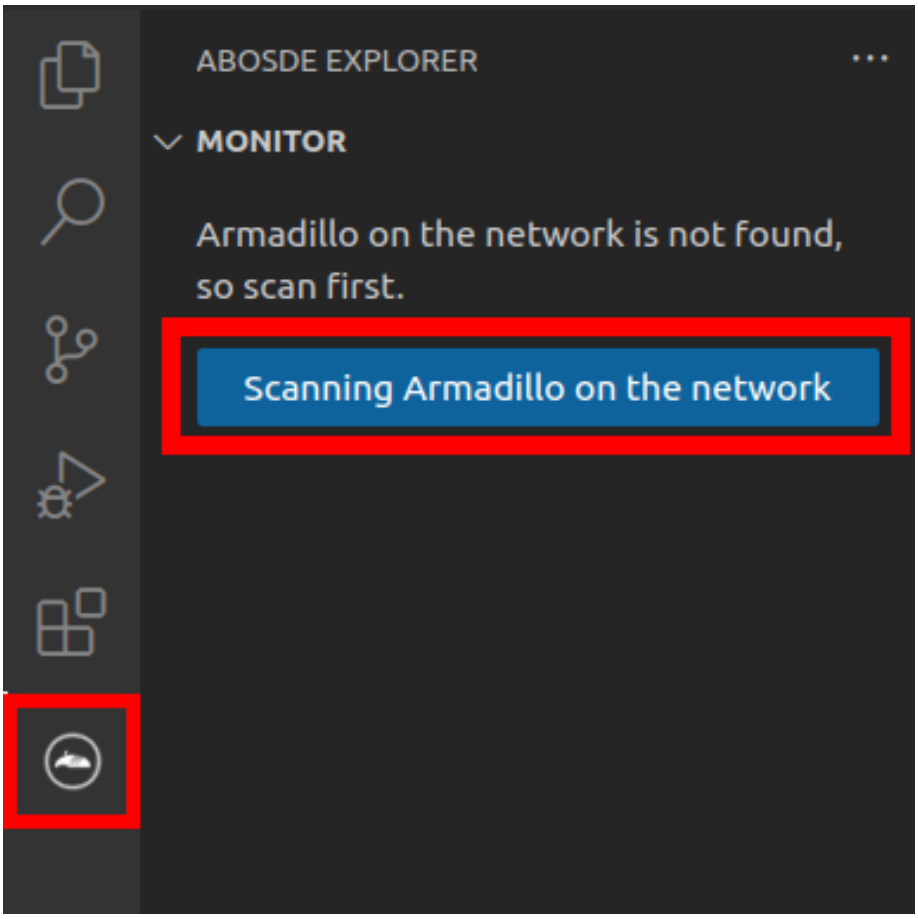


図 3.228 ABOSDE で ローカルネットワーク上の Armadillo をスキャンする

「図 3.229. 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。

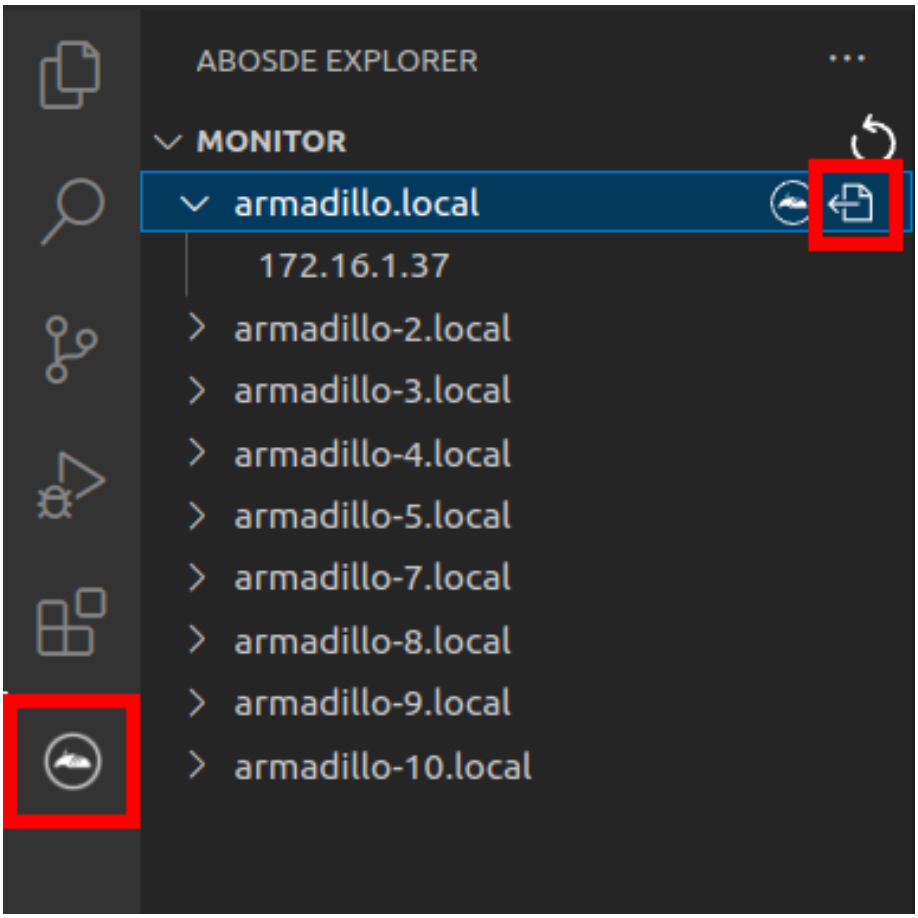


図 3.229 ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する

「図 3.230. 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。

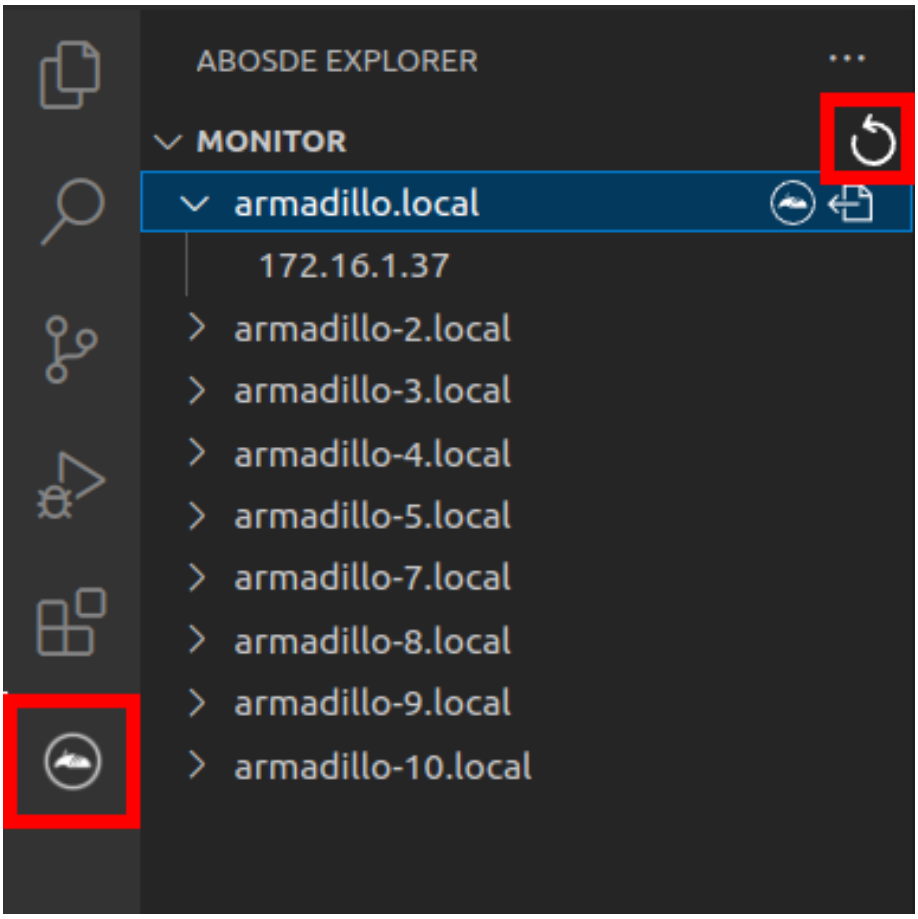


図 3.230 ABOSDE に表示されている Armadillo を更新する

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 3.231 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。

3.15.7.3. アプリケーションの実行

VSCode の左ペインの [my_project] から [App run on Armadillo] を実行すると、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。

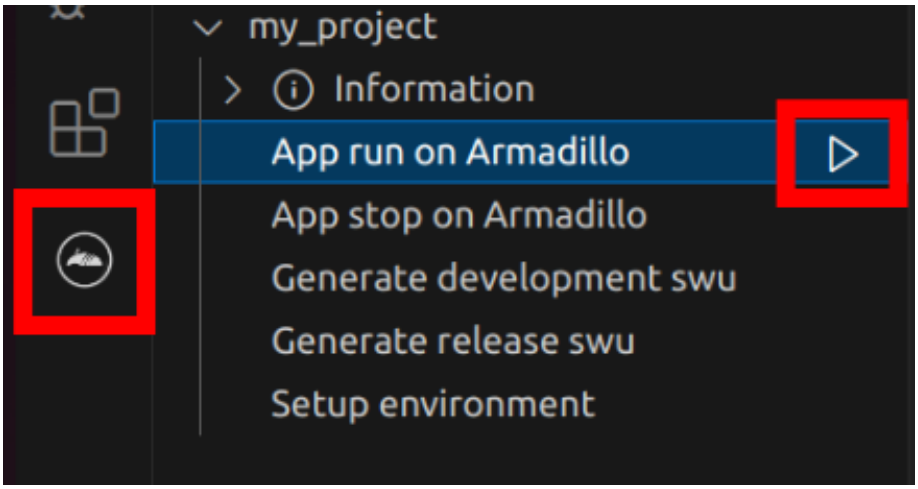


図 3.232 Armadillo 上でアプリケーションを実行する

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

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

図 3.233 実行時に表示されるメッセージ

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

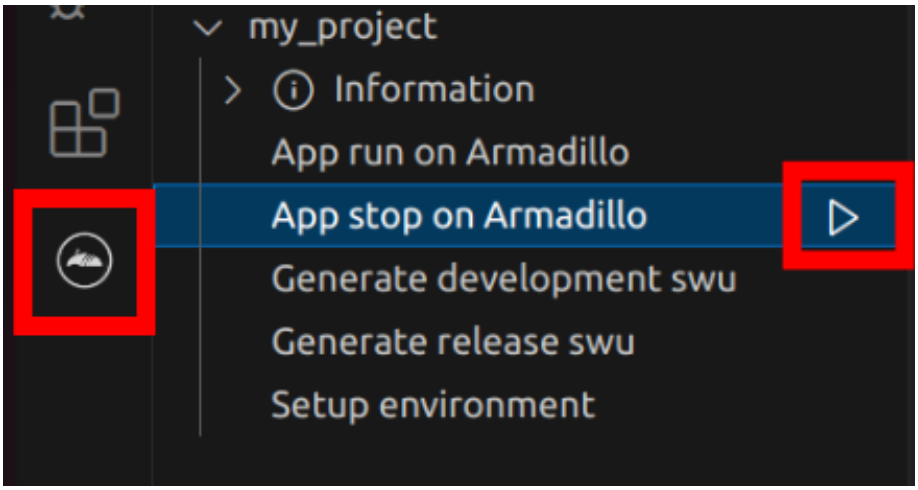


図 3.234 アプリケーションを終了する

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。

VSCode の左ペインの [my_project] から [Generate release swu] を実行すると、リリース版のアプリケーションを含んだ SWU イメージが作成されます。事前に 「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。

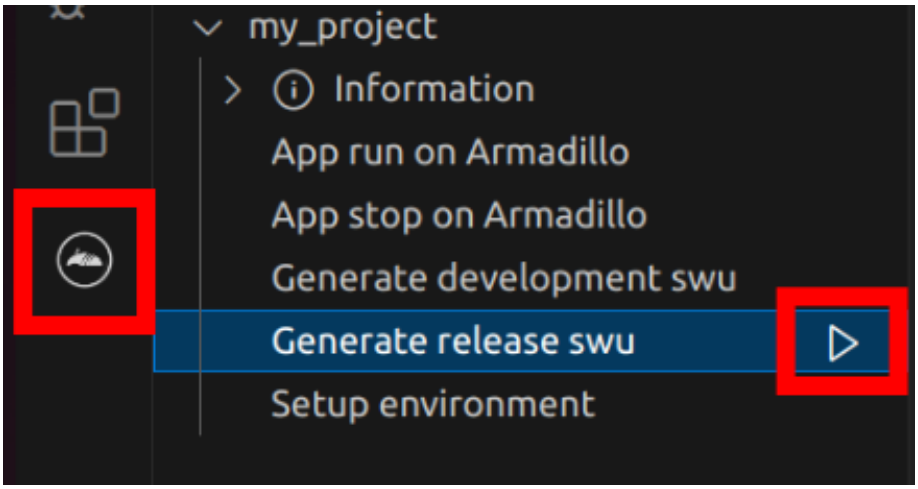


図 3.235 リリース版をビルドする

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 「3.2.3.5. 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

`development.swu` または `release.swu` を Armadillo にインストールすることで保存されたコンテナイメージを削除する方法は、 「6.2.3.1. VSCode から実行する」 を参照してください。

3.16. システムのテストを行う

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

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

3.16.1. ランニングテスト

長期間のランニングテストは実施すべきです。

ランニングテストで発見できる現象としては、以下のような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 3.236. 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 3.236 メモリの空き容量の確認方法

3.16.2. 異常系における挙動のテスト

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

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

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

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

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

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。
- 「4.7. 量産時の組み立て」では、Armadillo-IoT ゲートウェイ G4 の組み立て手順を説明します。

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。

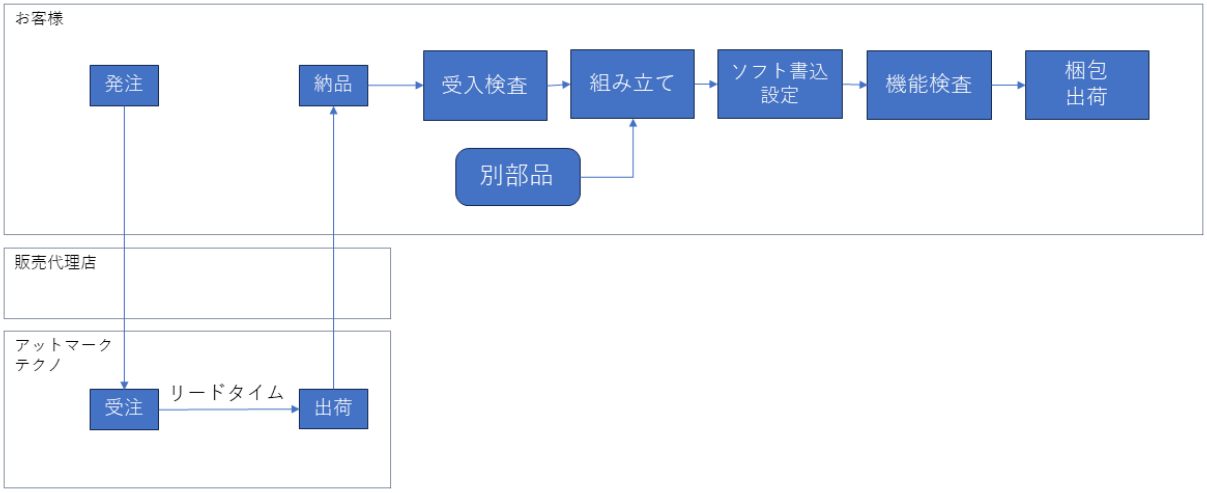


図 4.1 Armadillo 量産時の概略図

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. Armaidllo 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

    eMMC のデータリテンションを設定する場合は設定ファイル書き込み前に行ってください。eMMC のデータリテンションの設定に関しては 「6.26. eMMC のデータリテンション」 を参照してください。
- 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

	Pros	Cons
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、SD 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ジャンパピンをショートにするために Armadillo のケースを開ける必要がある ・ 動いているシステムをそのままインストールディスクにするため、出荷時の標準イメージから手動で同じ環境を構築する手順が残らない
SWUpdate	・ ジャンパピンをショートにせずに実行できるため、Armadillo のケースを開ける必要がない ・ 必ず必要となる初回アップデートを別途実行する必要がない	・ 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

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

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

4.4.1. /etc/swupdate_preserve_file への追記

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

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

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

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

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

4.4.2. Armadillo Base OS の更新

Armadillo-IoT ゲートウェイ G4 Armadillo Base OS [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/baseos]から「Armadillo-IoT ゲートウェイ G4 用 SWU イメージイメージファイル」の URL をコピーして、「図 4.4. Armadillo Base OS をアップデートする」に示すコマンドを実行することで Armadillo Base OS を最新版にアップデートできます。

```
[armadillo /]# swupdate -d '-u https://armadillo.atmark-techno.com/files/downloads/{url-product-
dir}/image/baseos-x2-[VERSION].swu'
```

図 4.4 Armadillo Base OS をアップデートする

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

「3.3.5.1. 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。

- VSCode を使用して生成

ATDE と VSCode 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。VSCode 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。

4.4.5. VSCode を使用して生成する

ATDE と VSCode 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。「3.3.4. VSCode のセットアップ」 を参考に、 ATDE に VSCode 開発用エクステンションをインストールしてください。VSCode 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。

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

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

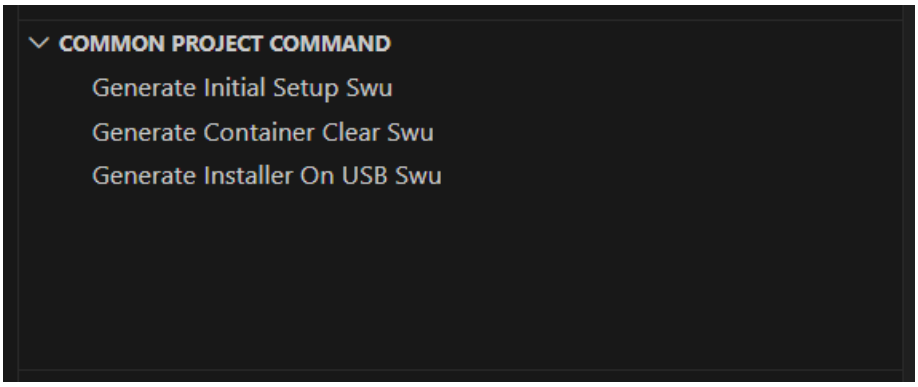


図 4.6 make-installer.swu を作成する

次に、対象製品を選択します。

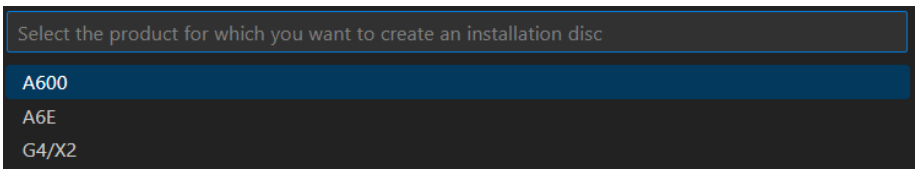


図 4.7 対象製品を選択する

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-IoT ゲートウェイ G4 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。「6.8.4. 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-IoT ゲートウェイ G4 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、VSCode 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

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
An installer system is already available on SD card. Use it? [Y/n] ❶

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] - 30026): 500 ❷
Checking and growing installer main partition
GPT data structures destroyed! You may now partition the disk using fdisk or
other utilities.
The operation has completed successfully.
Trying to install mkfs.exfat (exfatprogs) in memory from internet
fetch https://download.atmark-techno.com/alpine/v3.16/atmark/aarch64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.16/main/aarch64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.16/community/aarch64/APKINDEX.tar.gz
(1/1) Installing exfatprogs (1.1.3-r0)
Executing busybox-1.35.0-r14.trigger
OK: 159 MiB in 215 packages
exfatprogs version : 1.1.3
Creating exFAT filesystem(/dev/mmcblk1p2, cluster size=32768)

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 400.00MiB to max
Currently booted on /dev/mmcblk2p1
Copying boot image
Copying rootfs
314572800 bytes (315 MB, 300 MiB) copied, 11 s, 28.5 MB/s
300+0 records in
```

```
300+0 records out
314572800 bytes (315 MB, 300 MiB) copied, 11.0192 s, 28.5 MB/s
Copying /opt/firmware filesystem
Copying appfs
At subvol app/snapshots/volumes
At subvol app/snapshots/boot_volumes
At subvol app/snapshots/boot_containers_storage
Cleaning up and syncing changes to disk...
Installer updated successfully!
```

図 4.10 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 Enter キーを押下します。
❶
❶

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。詳細は「4.4.7.1. 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。

4.4.7.1. 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。

4.4.7.2. 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 4.11. 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 4.11 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 4.12 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 4.13 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。

4.4.7.3. 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 4.14. 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 4.14 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 4.15. 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 4.15 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。

「3.2.5.2. 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。

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 4.16 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 4.17 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-IoT ゲートウェイ G4 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-IoT ゲートウェイ G4 の拡張インターフェース(CON11、CON12)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「図 3.28. Armadillo-IoT ゲートウェイ G4 の拡張ボード例」に示す図のように組み付けを行ってください。

4.7.2. オプションケース(金属製)への組み付け

Armadillo-IoT ゲートウェイ G4 と 専用のオプションケース(金属製)を組み付ける際には、以下の組み立て図に従って行ってください。

オプションケース(金属製)の詳細な仕様については、「6.30.1. オプションケース(金属製)」を参照してください。

オプションケース(金属製)は、SoC の熱をケースに伝導させて放熱する構造で設計しています。基板裏の IC1 に放熱シートを貼り付け、ケース(下)と放熱シートを接触させた状態で基板をねじ止めします。ねじの締め付けトルクは 31.5cN·m です。

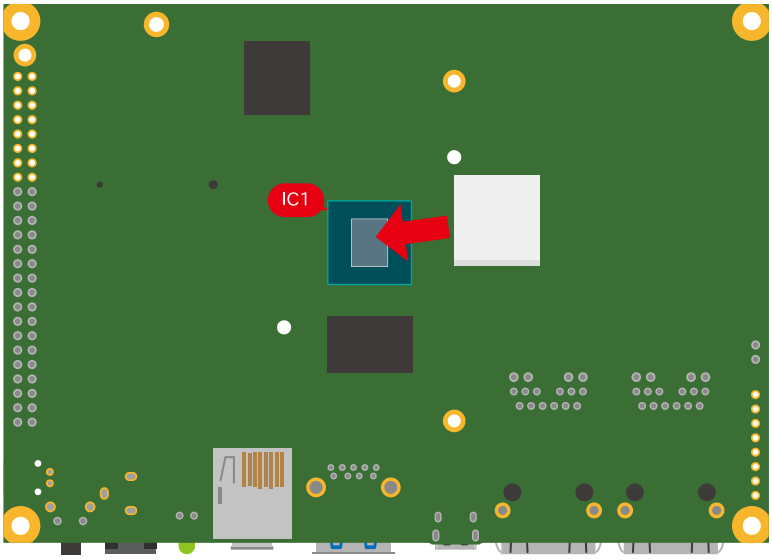


図 4.18 オプションケース(金属製) 放熱シート貼付

WLAN+BT コンボモジュールが搭載されたモデルの場合は、基板裏にもう 1 箇所放熱シートを貼り付けます。

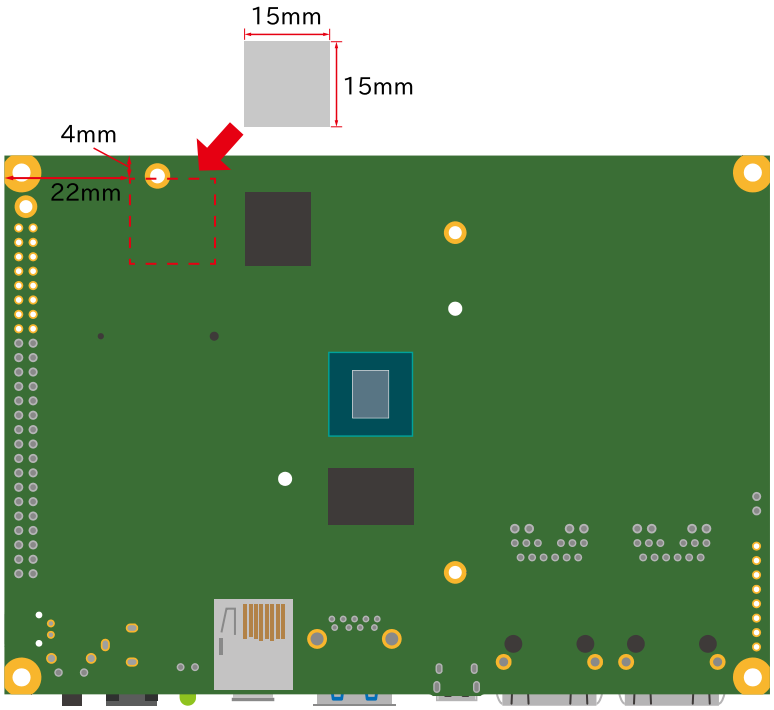


図 4.19 オプションケース(金属製) 放熱シート貼付(WLAN+BT コンボモジュール搭載モデル)

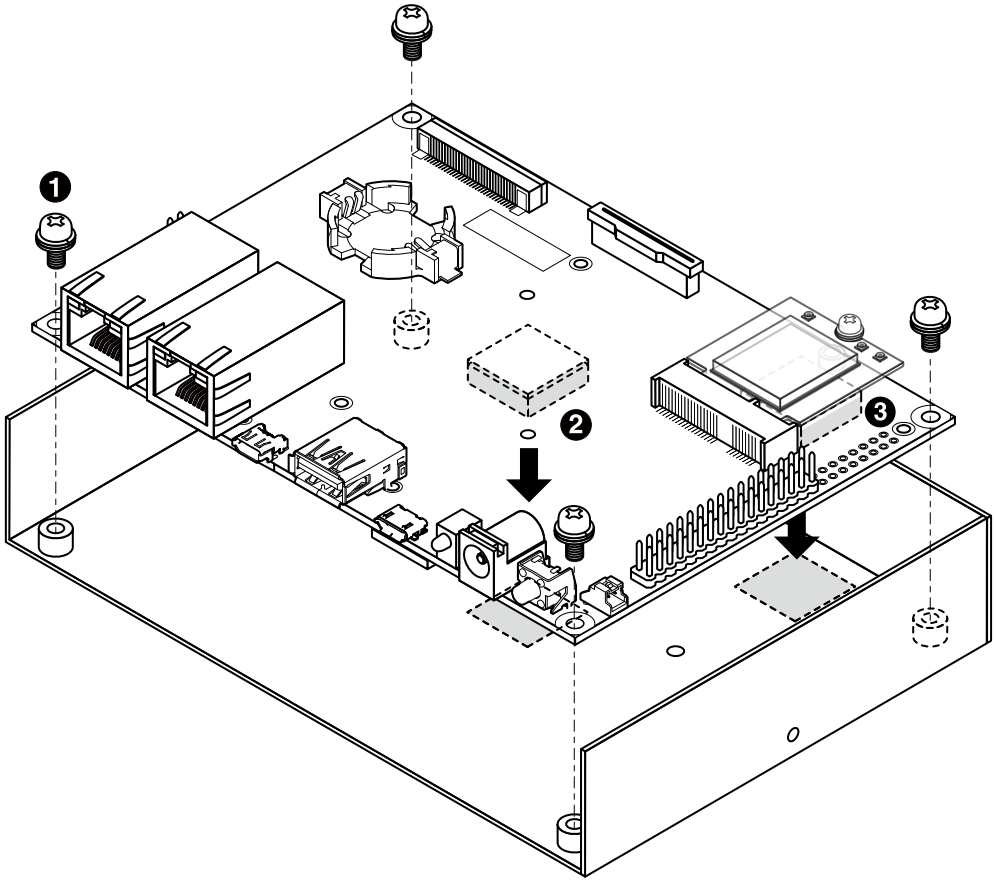


図 4.20 オプションケース(金属製) ケース(下)ねじ止め

❶ なべ小ねじ、ワッシャ、スプリングワッシャ付き(M3、L=6mm) × 4

❷ 放熱シート(15×15×4mm)

❸ 放熱シート(15×15×4.5mm) ※WLAN+BT コンボモジュール搭載モデルのみ

ケース(上)は、ケース(下)とフロント側を支点に嵌め合わせます。このとき、コネクタ、スイッチ、LEDが曲がらないように注意しながら、フロントに空いた穴に嵌めつつ閉めます。LTE モデルの場合には、3G/LTE モジュールの熱をケースに伝導させて放熱する構造で設計しています。3G/LTE モジュール上部に貼り付けられた放熱シートをケース(上)に接触させた状態で嵌め合わせます。

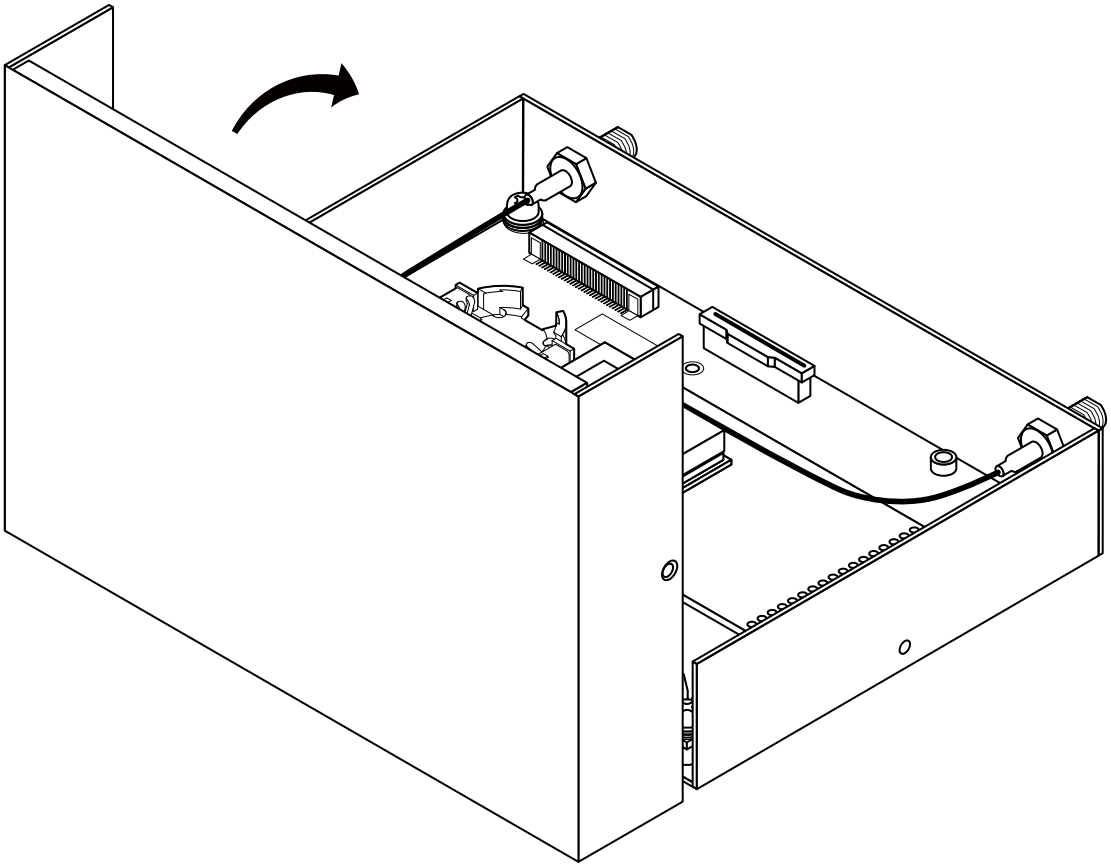


図 4.21 オプションケース(金属製) ケース嵌め合わせ

ケース(上)とケース(下)は 2 箇所ねじ止めします。ねじの締め付けトルクは 17.5cN·m です。

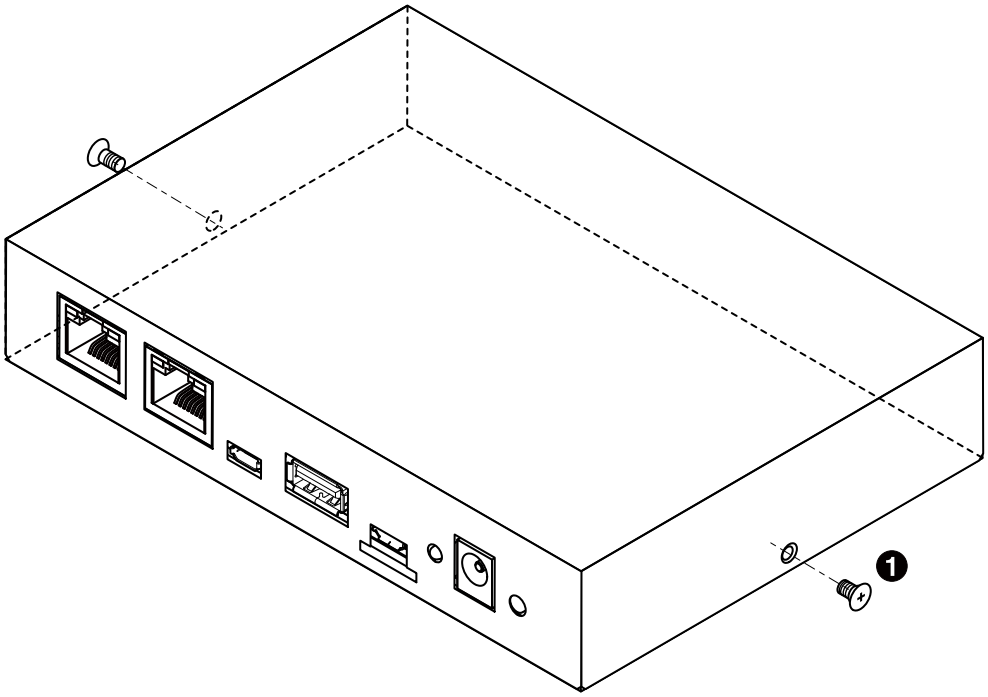


図 4.22 オプションケース(金属製) ケース(上)ねじ止め

❶ 皿ねじ(M2.6、L=4mm) × 2

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。

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。

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. 無線 LAN/BT 用外付けアンテナの指向性

無線 LAN/BT 用外付けアンテナは「図 5.1. 無線 LAN/BT 用外付けアンテナの指向性」に示す指向性があります。

一般的な λ/2 ダイポールアンテナ、λ/4 モノポールアンテナの指向性イメージです。実際のアンテナの特性を正確に表しているものではありません

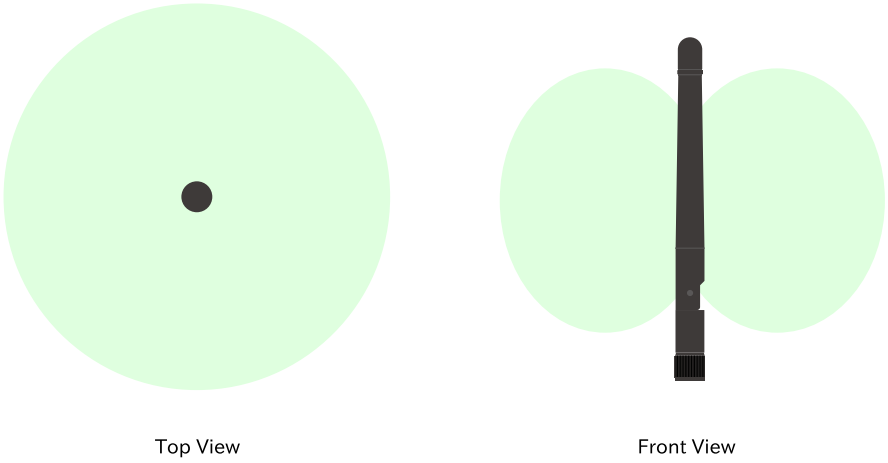


図 5.1 無線 LAN/BT 用外付けアンテナの指向性

5.2.4. LTE 外付け用アンテナの指向性

LTE アンテナは「図 5.2. LTE 外付け用アンテナの指向性」に示す指向性があります。

一般的な λ/2 ダイポールアンテナ、λ/4 モノポールアンテナの指向性イメージです。実際のアンテナの特性を正確に表しているものではありません

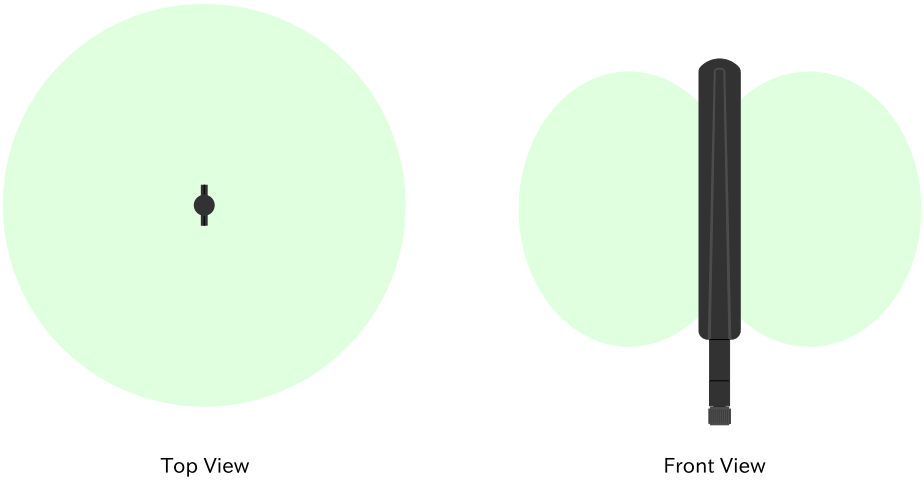


図 5.2 LTE 外付け用アンテナの指向性

5.2.5. LTE の電波品質に影響する事項

一般的には、周辺に障害物、特に鉄板や鉄筋コンクリート、電波シールドフィルムの貼られたガラスが存在する場合電波を大きく妨げます。また、周辺機器の電波出力、人通り(周辺のスマートフォン=機器が増える)、基地局との距離・方向など多くの要素によって変化します。

WWAN LED にて電波状況をチェックできますので、設置時に電波状況を確認の上運用ください。WWAN LED の状態と意味は「表 3.49. LED20 の状態」を参照ください。

5.2.6. 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。

5.2.7. Armadillo の状態を表すインジケータ

LED にて状態を表示しています。

有線 LAN の状態は「表 3.20. CON2 LAN LED の動作」及び「表 3.23. CON3 LAN LED の動作」を、 3G/LTE モジュールの状態を表示する LED に関しては「表 3.49. LED20 の状態」を参照ください。

5.2.8. 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

5.2.8.1. 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。

5.2.8.2. 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 5.3. 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 5.3 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 5.4. 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 5.4 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 5.5. 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 5.5 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 5.6. 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 5.6 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 5.7. 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 5.7 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 5.8. 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）
3: eth1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN qlen 1000
    link/ether 00:11:0c:33:01:9f brd ff:ff:ff:ff:ff:ff
:（省略）
```

図 5.8 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 5.9. 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
eth1: 00:11:0C:12:34:57
```

図 5.9 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 5.9. 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 5.8. 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。

5.2.9. 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、「3.3.3.5. 終了方法」を参照してください。

5.3. ABOSDE で開発したアプリケーションをアップデートする

ABOSDE で開発したアプリケーションのアップデートは、開発時と同様に ABOSDE を用いて行うことが出来ます。

「3.12. ABOSDE によるアプリケーションの開発」で示したように、開発時にはリリース版のアプリケーションを Armadillo にインストールするために、 VScode 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 VScode で開く

「図 5.10. VScode 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 VSCode を起動してください。

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

図 5.10 VScode を起動

5.3.1.2. アップデート前のバージョンのプロジェクトを管理する

ABOSDE では、プロジェクトのバージョン管理は行っていません。必要な場合はユーザー自身でアップデート前のプロジェクトを管理してください。

アップデート前のプロジェクトの release.swu のバージョンを知りたい場合は「6.6. SWU イメージの内容の確認」を参照してください。

5.3.1.3. アプリケーションのソースコードを編集しテストする

既存のアプリケーションのソースコードを編集した後、「3.12. ABOSDE によるアプリケーションの開発」を参考に、アプリケーションが Armadillo 上で問題なく動作するかテストを行ってください。

5.3.1.4. アップデート用の swu を作成する

VScode 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「3.2.3.5. 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。「3.3. 開発の準備」で作成した環境で作業してください。

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。

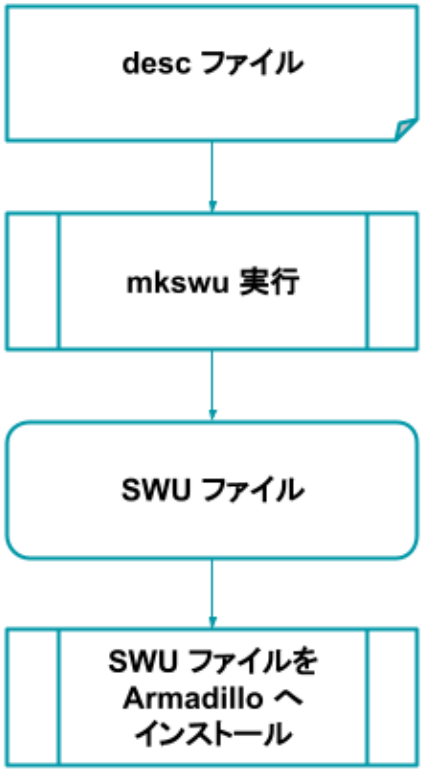


図 5.11 desc ファイルから Armadillo へ SWU イメージをインストールする流れ

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 5.12. 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 5.12 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 5.13. 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 5.13 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 5.14 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、「3.2.3.5. 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. hawkBit サーバーから複数の Armadillo をアップデートする

5.6.1. hawkBit とは

hawkBit は、サーバー上で実行されるプログラムで、ネットワーク経由で 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。

hawkBit は次のような機能を持っています。

- ソフトウェアの管理
- デバイスの管理
  - デバイス認証 (セキュリティトークン、証明書)
  - デバイスのグループ化
- アップデート処理の管理
  - 進捗のモニタリング
  - スケジューリング、強制アップデート
- RESTful API での直接操作

5.6.2. データ構造

hawkBit は、配信するソフトウェアを次のデータ構造で管理します。

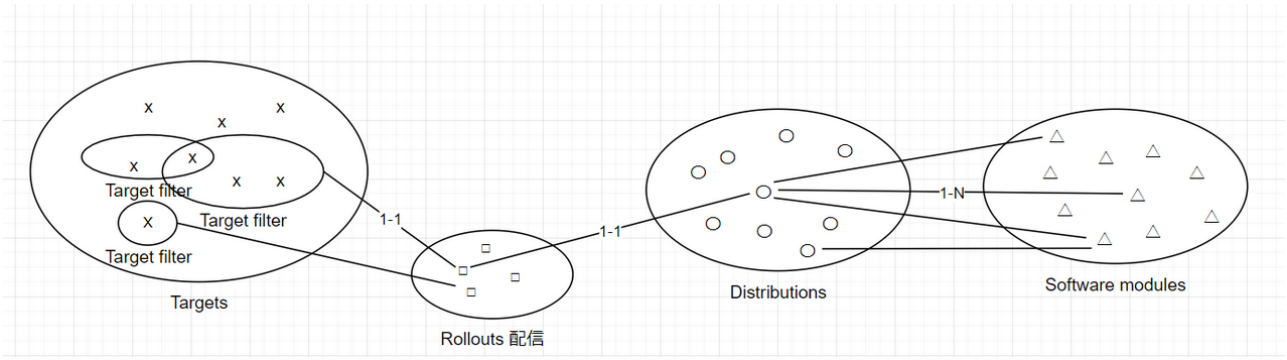


図 5.15 hawkBit が扱うソフトウェアのデータ構造

5.6.3. hawkBit サーバーから複数の Armadillo に配信する

hawkBit サーバーを利用することで複数の Armadillo のソフトウェアをまとめてアップデートすることができます。

手順は次のとおりです。

1. コンテナ環境の準備

   Docker を利用すると簡単にサーバーを準備できます。Docker の準備については https://docs.docker.com/get-docker/ を参照してください。

Docker の準備ができたら、要件に合わせてコンテナの設定を行います。

- ATDE の場合
  - apt update && apt install mkswu で最新のバージョンを確認してください。
  - ポート転送も必要です。一番シンプルな、プロキシを使用しない場合は 8080、TLS を使う場合は 443 を転送してください。

    vmware を使う場合は vmware の NAT モードのネットワークを使用している仮想マシン上で Web サーバを構成する [https://kb.vmware.com/s/article/2006955?lang=ja]ページを参考にしてください。
  - ホスト PC の IP アドレスを控えておいてください。
- ATDE 以外の場合
  - Armadillo-IoT ゲートウェイ G4 開発用ツール [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/tools] から「Hawkbit docker-compose コンテナ」 をダウンロードして展開してください。この場合、以下に /usr/share/mkswu/hawkbit-compose を使う際に展開先のディレクトリとして扱ってください。
  - docker がアクセスできるホストネームやアドレスを控えておいてください。

2. hawkBit サーバーの準備

   /usr/share/mkswu/hawkbit-compose/setup_container.sh を実行して、質問に答えてください。

   以下に簡単な（TLS を有効にしない）テスト用の場合と、TLS を有効にした場合の例を示します。

   setup_container.sh を一度実行した場合はデータのディレクトリにある setup_container.sh のリンクを実行して、ユーザーの追加等のオプション変更を行うこともできます。詳細は`--help`を参考にしてください。

```
[ATDE ~]$ /usr/share/mkswu/hawkbit-compose/setup_container.sh
docker-compose の設定ファイルと hawkBit のデータをどこに保存しますか？ [/home/atmark/hawkbit-compose] ❶
setup_container.sh へのリンクを /home/atmark/hawkbit-compose に作ります。
docker サービスに接続できませんでした。sudo でもう一度試します。
[sudo] atmark のパスワード: ❷
OK!
Hawkbit admin user name [admin] ❸
admin ユーザーのパスワード: ❹
パスワードを再入力してください:
追加の管理人アカウントのユーザーネーム（空にすると追加しません） ❺
hawkBit の「device」ユーザーを登録しますか？（自動登録用） [Y/n] ❻
device ユーザーのパスワード:
パスワードを再入力してください:
hawkBit の「mkswu」ユーザーを登録しますか？（swu のアップロード用） [Y/n] ❼
ユーザーにロールアウトの権限を与えますか？（インストール要求を出すこと） [Y/n] ❽
mkswu ユーザーのパスワード:
パスワードを再入力してください:
Setup TLS reverse proxy? [y/N] ❾
```

```
コンテナの設定が完了しました。docker-compose コマンドでコンテナの管理が可能です。
/home/atmark/hawkbit-compose/setup_container.sh を再び実行すると設定の変更が可能です。
hawkBit コンテナを起動しますか？ [Y/n] ❿
Creating network "hawkbit-compose_default" with the default driver
Pulling mysql (mysql:5.7)...
:（省略）
Creating hawkbit-compose_hawkbit_1 ... done
Creating hawkbit-compose_mysql_1   ... done
```

図 5.16 hawkBit コンテナの TLS なしの場合（テスト用）の実行例

❶ コンテナのコンフィグレーションとデータベースの場所を設定します。

❷ docker の設定によって sudo が必要な場合もあります。

❸ admin ユーザーのユーザー名を入力します。

❹ admin ユーザーのパスワードを二回入力します。

❺ 追加のユーザーが必要な場合に追加できます。

❻ examples/hawkbit_register.desc で armadillo を登録する場合に作っておいてください。詳細は 「5.6.3.2. SWU で hawkBit を登録する」 を参考にしてください。

❼ hawkbit_push_update でアップデートを CLI で扱う場合は、「Y」を入力してください。詳細は「5.6.3.1. hawkBit のアップデート管理を CLI で行う」 を参照してください。

❽ hawkbit_push_update でアップデートを実行する場合は、「Y」を入力してください。

❾ ここでは http でテストのコンテナを作成するので、「N」のままで進みます。

❿ コンテナを起動します。初期化が終わったら <IP>:8080 でアクセス可能になります。

```
[ATDE ~]$ /usr/share/mkswu/hawkbit-compose/setup_container.sh
docker-compose の設定ファイルと hawkBit のデータをどこに保存しますか？ [/home/atmark/hawkbit-compose]
setup_container.sh へのリンクを /home/atmark/hawkbit-compose に作ります。
docker サービスに接続できませんでした。sudo でもう一度試します。
OK!
Hawkbit admin user name [admin]
admin ユーザーのパスワード:
パスワードを再入力してください:
パスワードが一致しません。
admin ユーザーのパスワード:
パスワードを再入力してください:
追加の管理人アカウントのユーザーネーム（空にすると追加しません）
hawkBit の「device」ユーザーを登録しますか？（自動登録用） [Y/n]
device ユーザーのパスワード:
パスワードを再入力してください:
hawkBit の「mkswu」ユーザーを登録しますか？（swu のアップロード用） [Y/n]
ユーザーにロールアウトの権限を与えますか？（インストール要求を出すこと） [Y/n]
mkswu ユーザーのパスワード:
パスワードを再入力してください:
Setup TLS reverse proxy? [y/N] y ❶
lighttpd が起動中で、リバースプロキシ設定と競合しています。
lighthttpd サービスを停止しますか？ [Y/n] ❷
Synchronizing state of lighttpd.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install disable lighttpd
```

```
Removed /etc/systemd/system/multi-user.target.wants/lighttpd.service.
リバースプロキシの設定に証明書の domain name が必要です。
この domain はこのままデバイスからアクセスできる名前にしてください。
例えば、https://hawkbit.domain.tld でアクセスしたら hawkbit.domain.tld、
https://10.1.1.1 でしたら 10.1.1.1 にしてください。
証明書の domain name: 10.1.1.1 ❸
証明書の有効期限を指定する必要があります。Let's encrypt を使用する場合、
この値は新しい証明書が生成されるまでしか使用されないので、デフォルトの値
のままにしておくことができます。Let's encrypt を使用しない場合、
数年ごとに証明書を新しくすることが最も好ましです。
証明書の有効期間は何日間にしますか？ [3650] ❹
クライアントの TLS 認証を設定するために CA が必要です。
署名 CA のファイルパス（空にするとクライアント TLS 認証を無効になります） [] ❺
サーバーが直接インターネットにアクセス可能であれば、Let's Encrypt の証明書
を設定することができます。TOS への同意を意味します。
https://letsencrypt.org/documents/LE-SA-v1.2-November-15-2017.pdf
certbot コンテナを設定しますか？ [y/N] ❻

/home/atmark/hawkbit-compose/data/nginx_certs/proxy.crt を /usr/local/share/ca-
certificates/ にコピーして、 update-ca-certificates を実行する必要があります。
この base64 でエンコードされたコピーを examples/hawkbit_register.sh の
SSL_CA_BASE64 に指定する手順が推奨されます。

LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUJlekNDQVNHZ0F3SUJBZ0lVQTByZ0cwcTJF
SFNnampmb0tUZWg3aGlaSVVVd0NnWUlLb1pJemowRUF3SXcKRXpFUk1BOEdBMVVFQXd3SU1UQXVN
UzR4TGpFd0hoY05Nakl3TXpJMU1EVXhOVFU0V2hjTk16SXdNekl5TURVeApOVFU0V2pBVE1SRXdE
d1lEVlFRRERBZ3hNQzR4TGpFdU1UQlpNQk1HQnlxR1NNNDlBZ0VHQ0NxR1NNNDlBd0VICkEwSUFC
SDFFREhBN3NOTlFJUDlTdlhlUnNmWjl2dVVFWkRkMVE2TzViRlV2RTh4UjUwUlBCLzNlajMzd0VI
NEoKYmZqb296bEpXaExlSG5SbGZsaHExVDlKdm5TalV6QlJNQjBHQTFVZERnUVdCQlFBUmYvSkdT
dkVJek5xZ2JMNQpQamY2VGRpSk1EQWZCZ05WSFNNRUdEQVdnQlFBUmYvSkdTdkVJek5xZ2JMNVBq
ZjZUZGlKTURBUEJnTlZIUk1CCkFmOEVCVEFEQVFIL01Bb0dDQ3FHU0000OUJBTUNBMGdBTUVVQ0lD
Nis3ZzJlZk1SRXl0RVk5WDhDNC8vUEw1U1kKWUlGZHUxVFZiUEZrSlV0SUFpRUE4bm1VSnVQSFlz
SHg2N2ErZFRwSXZ1QmJUSG1KbWd6dUl3bTJ2RXppRnZRPQotLS0tLUVORCBDRVJUSUZJQ0FURS0t
LS0tCg== ❼


Let's encrypt の設定は後で足したい場合に setup_container.sh を--letsencrypt で実行してください。

コンテナの設定が完了しました。docker-compose コマンドでコンテナの管理が可能です。
/home/atmark/hawkbit-compose/setup_container.sh を再び実行すると設定の変更が可能です。
hawkBit コンテナを起動しますか？ [Y/n]
```

図 5.17 hawkBit コンテナの TLS ありの場合の実行例

❶ 今回は TLS を有効にするので、「y」を入力します。

❷ lighttpd サービスが起動している場合に聞かれます。不要なので、停止します。

❸ 証明書の common name を入力してください。ATDE の場合、ポート転送によってホストの IP アドレスで接続しますのでそのアドレスを入力します。Let's encrypt を使用する場合には外部からアクセス可能な DNS を入力してください。

❹ 証明書の有効期間を設定します。デフォルトでは 10 年になっています。Let's encrypt を使用する場合には使われていません。

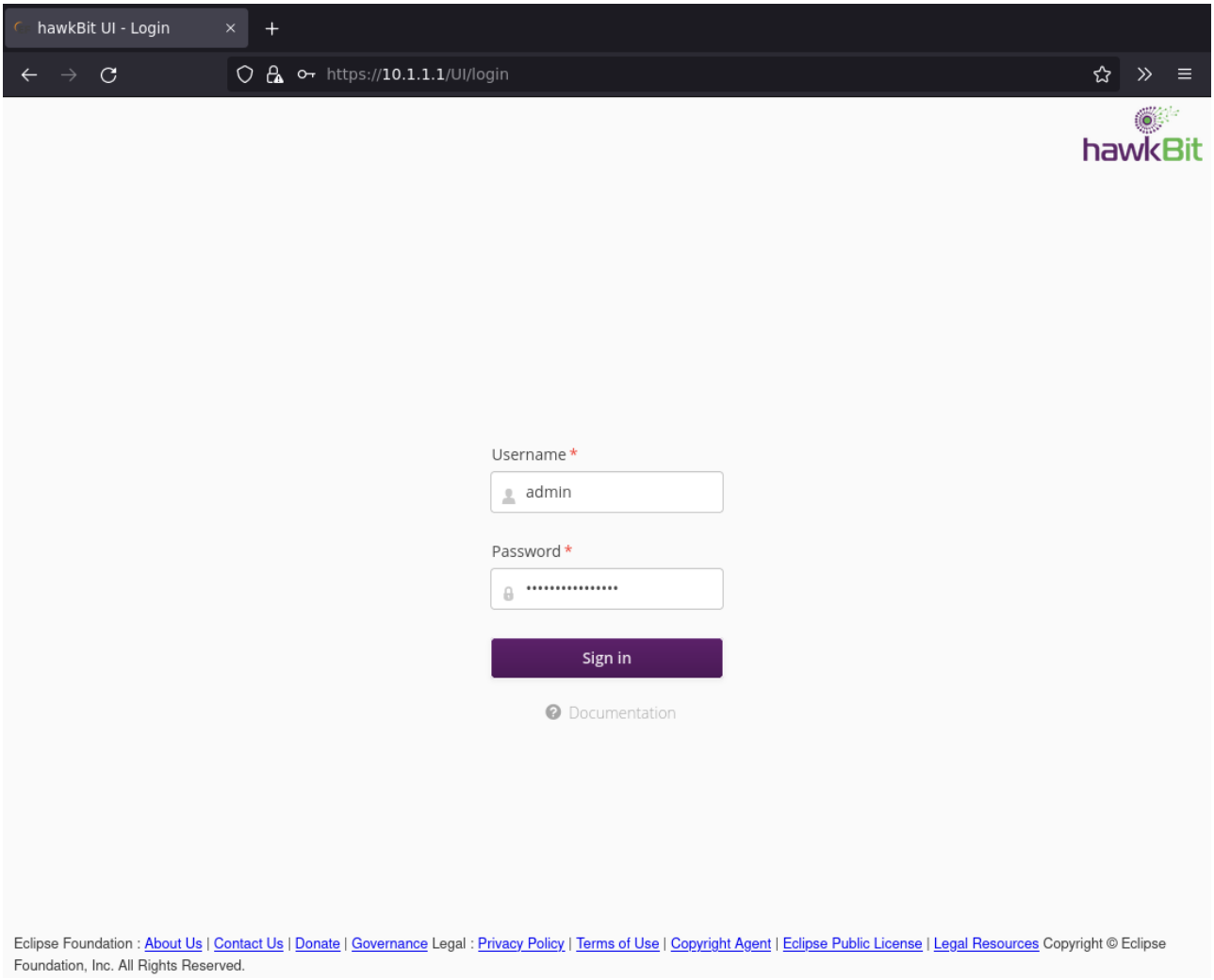
❺ クライアント側では x509 証明書で認証をとることができますが、この例では使用しません。

❻ Let's encrypt による証明書を作成できます。ATDE の場合は外部からのアクセスが難しいので、この例では使用しません。

❼ 自己署名証明書を作成したので、 Armadillo に設置する必要があります。この証明書の取扱いは 「5.6.3.2. SWU で hawkBit を登録する」 を参照してください。

3. hawkBit へのログイン

作成したコンテナによって `http://<サーバーの IP アドレス>:8080` か `https://<サーバーのアドレス>` にアクセスすると、ログイン画面が表示されます。

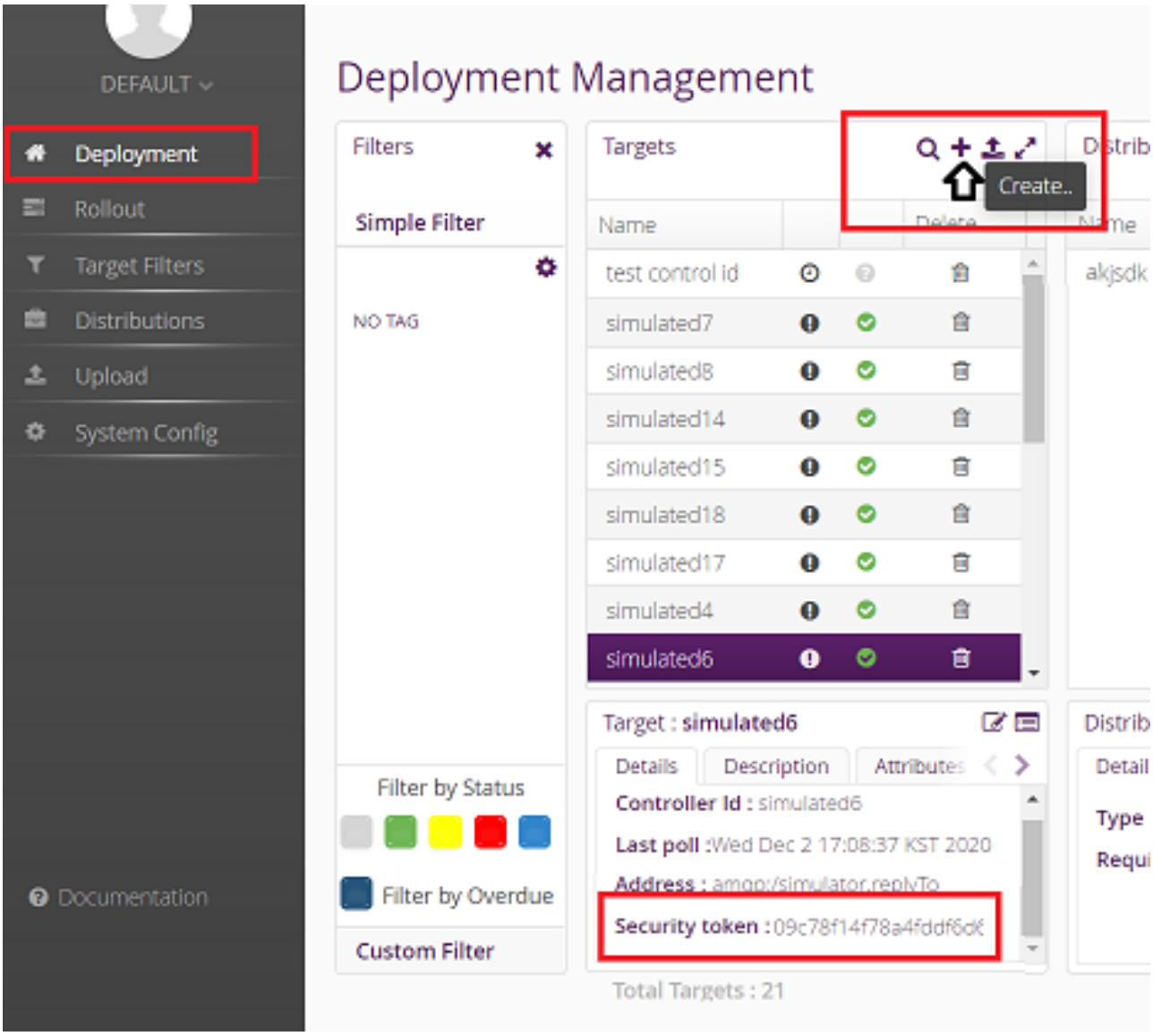


デフォルトでは次のアカウントでログインできます。

ユーザー	admin
パスワード	admin

4. Armadillo を Target に登録する

左側のメニューから Deployment をクリックして、Deployment の画面に移ります。



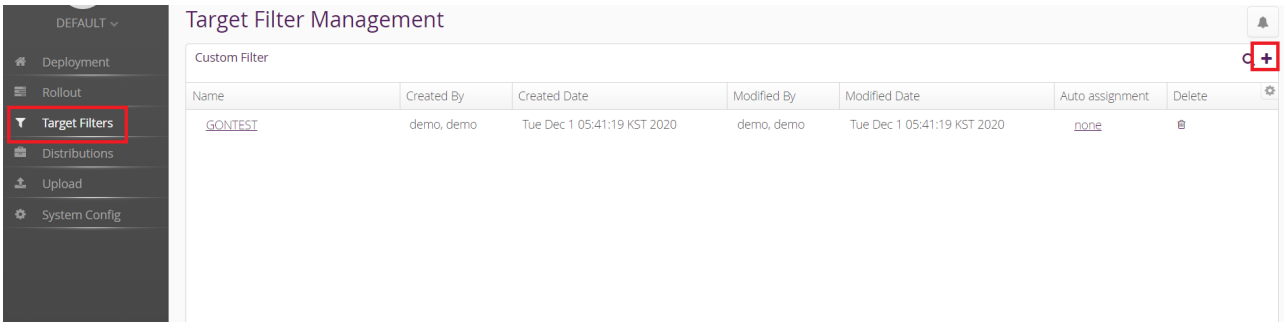
"+"をクリックして Target を作成します。

作成したターゲットをクリックすると、下のペインに "Security token:<文字列>" と表示されるので、<文字列>の部分をメモします。

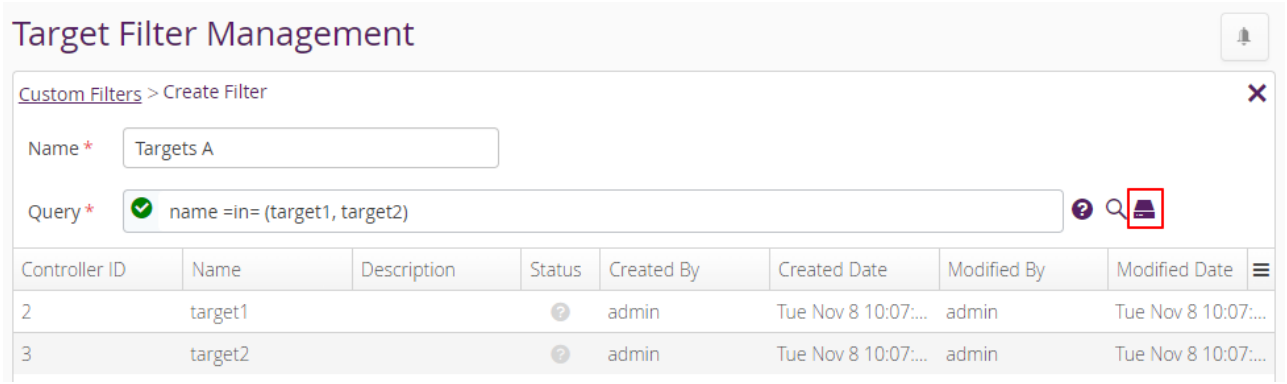
メモした<文字列>を Armadillo の /etc/swupdate.cfg に設定すると Hawkbit への接続認証が通るようになります。

5. Target Filter を作成する

左側のメニューから"Target Filters"をクリックして、Target Filters の画面に移ります。



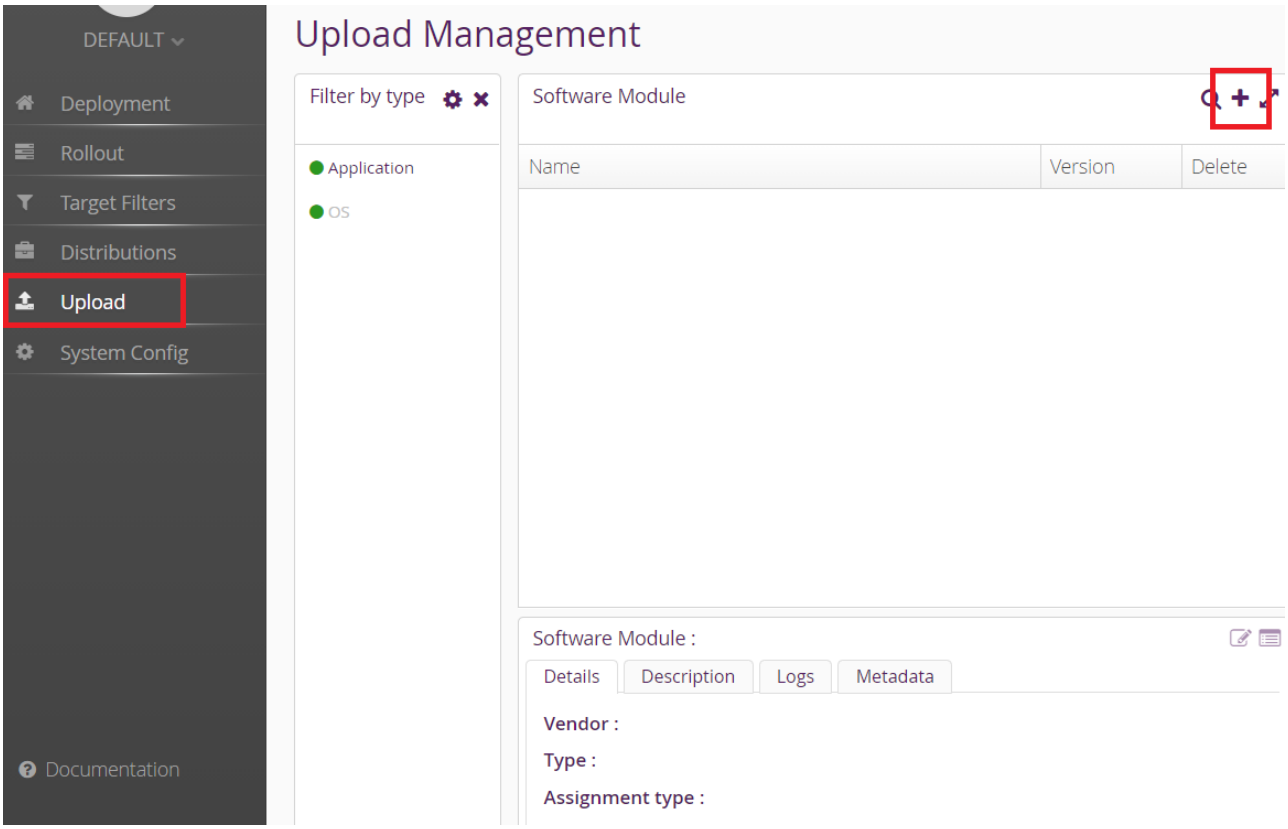
"+" をクリックして新規に Target Filter を作成します。



Filter name と フィルタリング条件を入力して保存します。

6. Software module を作成する

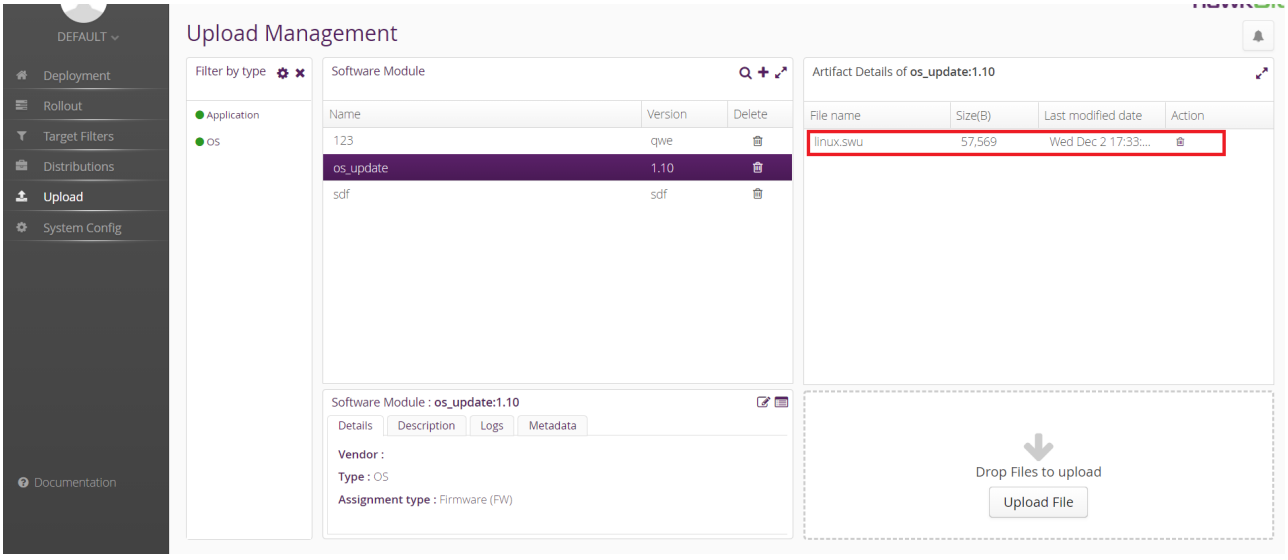
   左側のメニューから"Upload"をクリックして、Upload Management の画面に移ります。



"+" をクリックして Software module を作成します。type には OS/Application、version には任意の文字列を指定します。

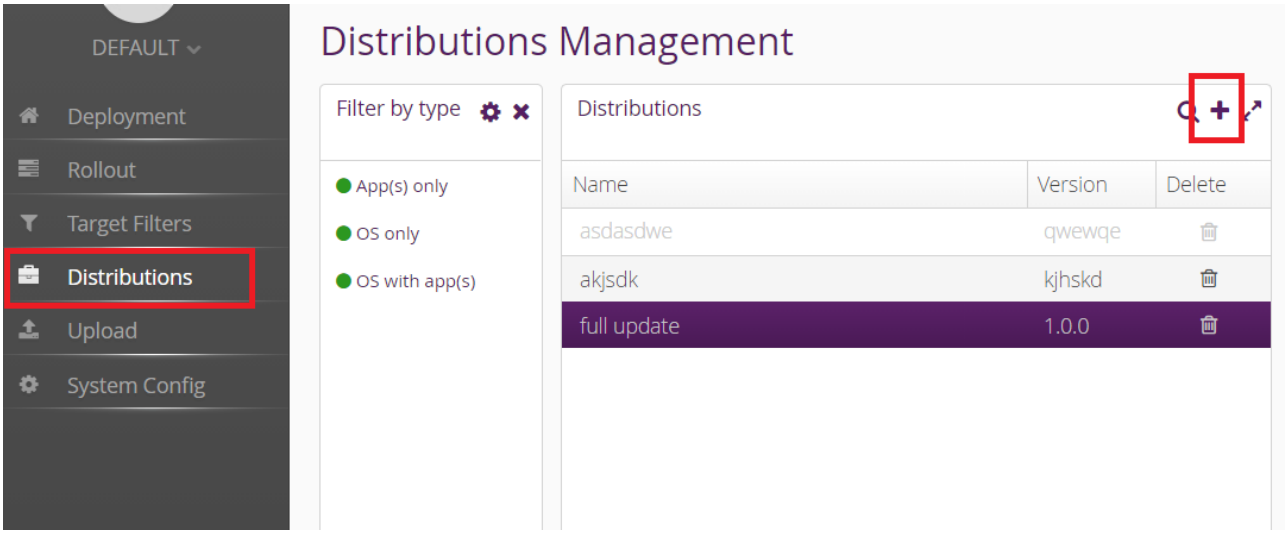
7. swu パッケージをアップロードして Software module に関連付ける

   先程作成した Software module を選択して、ハイライトされた状態で、"Upload File"ボタンをクリックするか、ファイルをドラッグアンドドロップしてアップロードします。

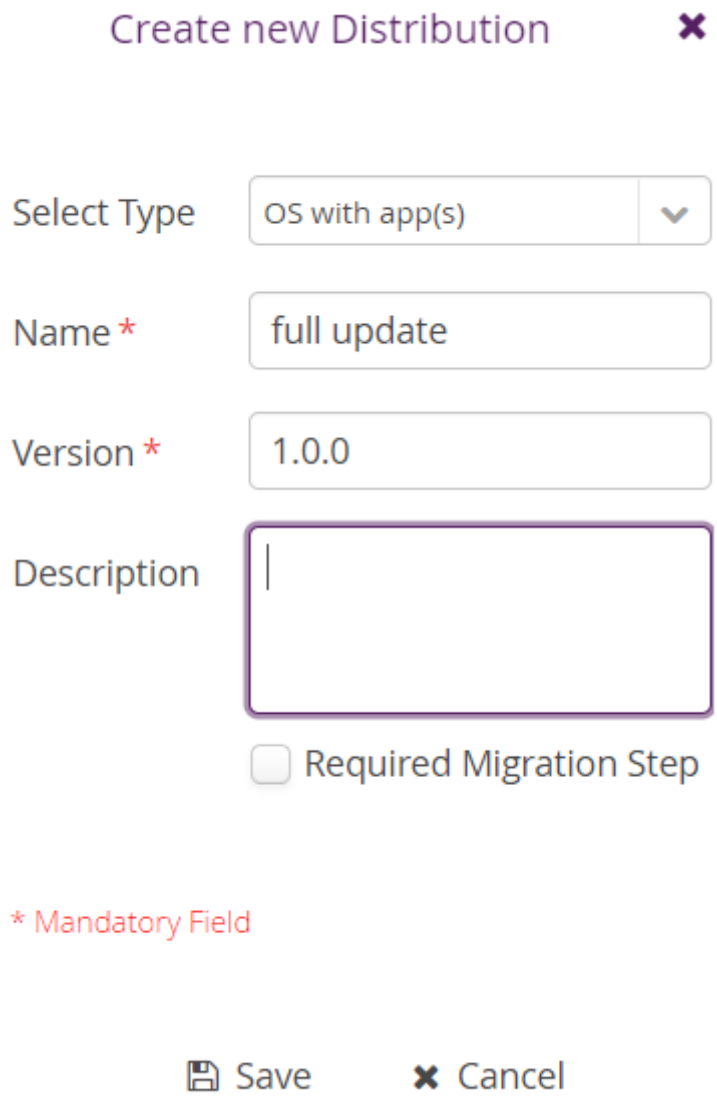


8. Distribution を作成して Software module を関連付ける

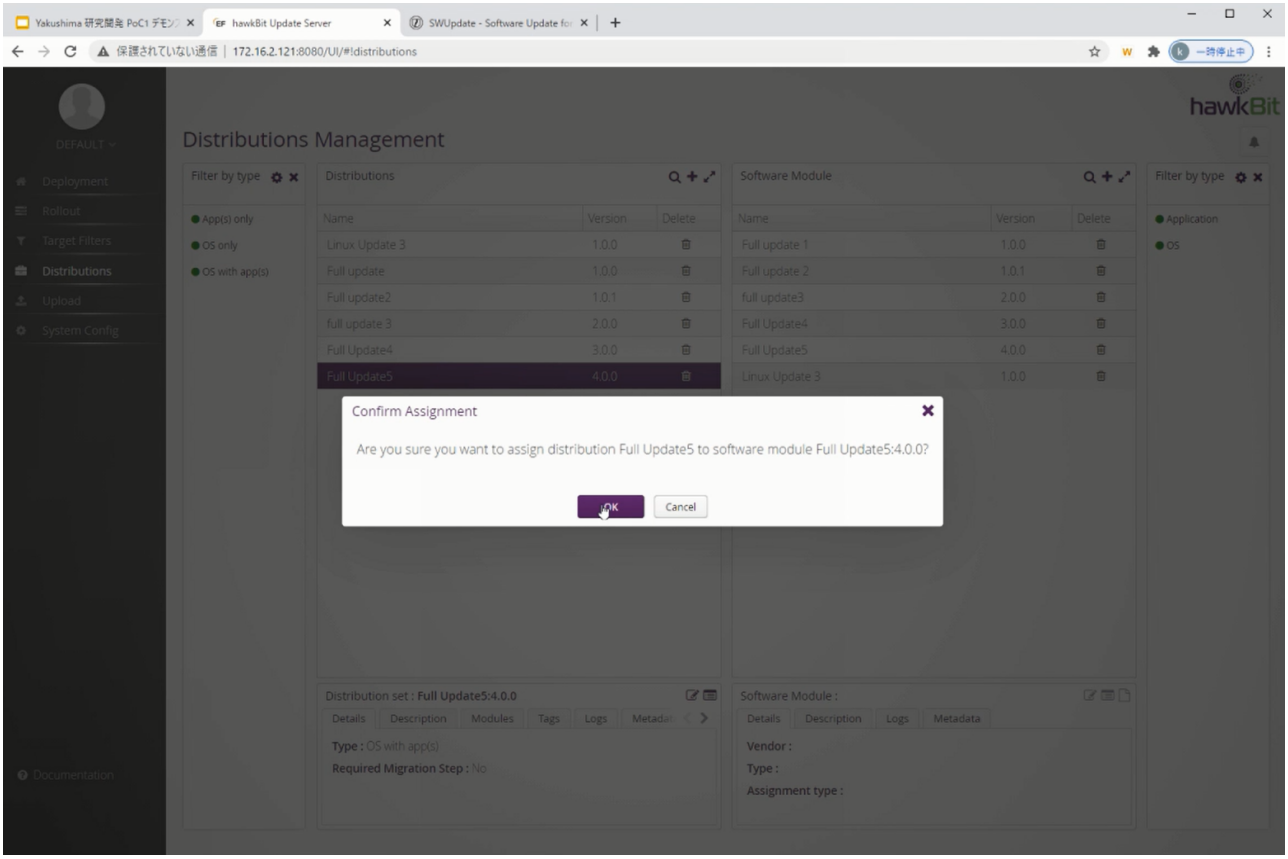
左側のメニューから"Distribution"をクリックして、Distribution Management の画面に移ります。



"+" をクリックして Distribution を作成します。type には OS/OSwithApp/Apps、version には任意の文字列を指定します。

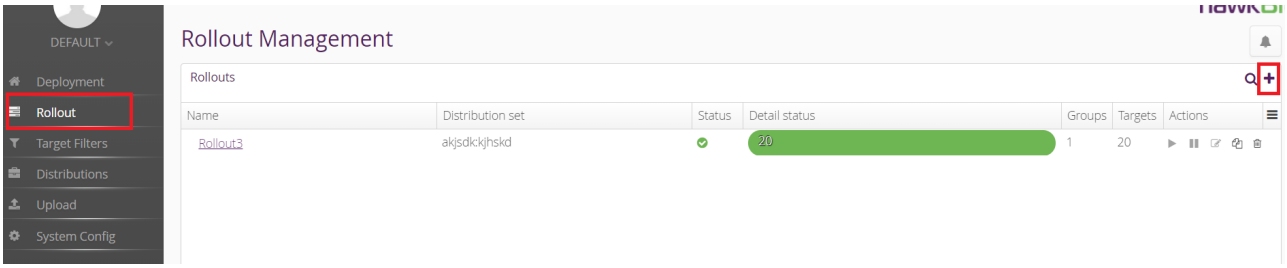


"Software module"のペインから先程作成した Software をドラッグして、作成した Distribution の上にドロップします。

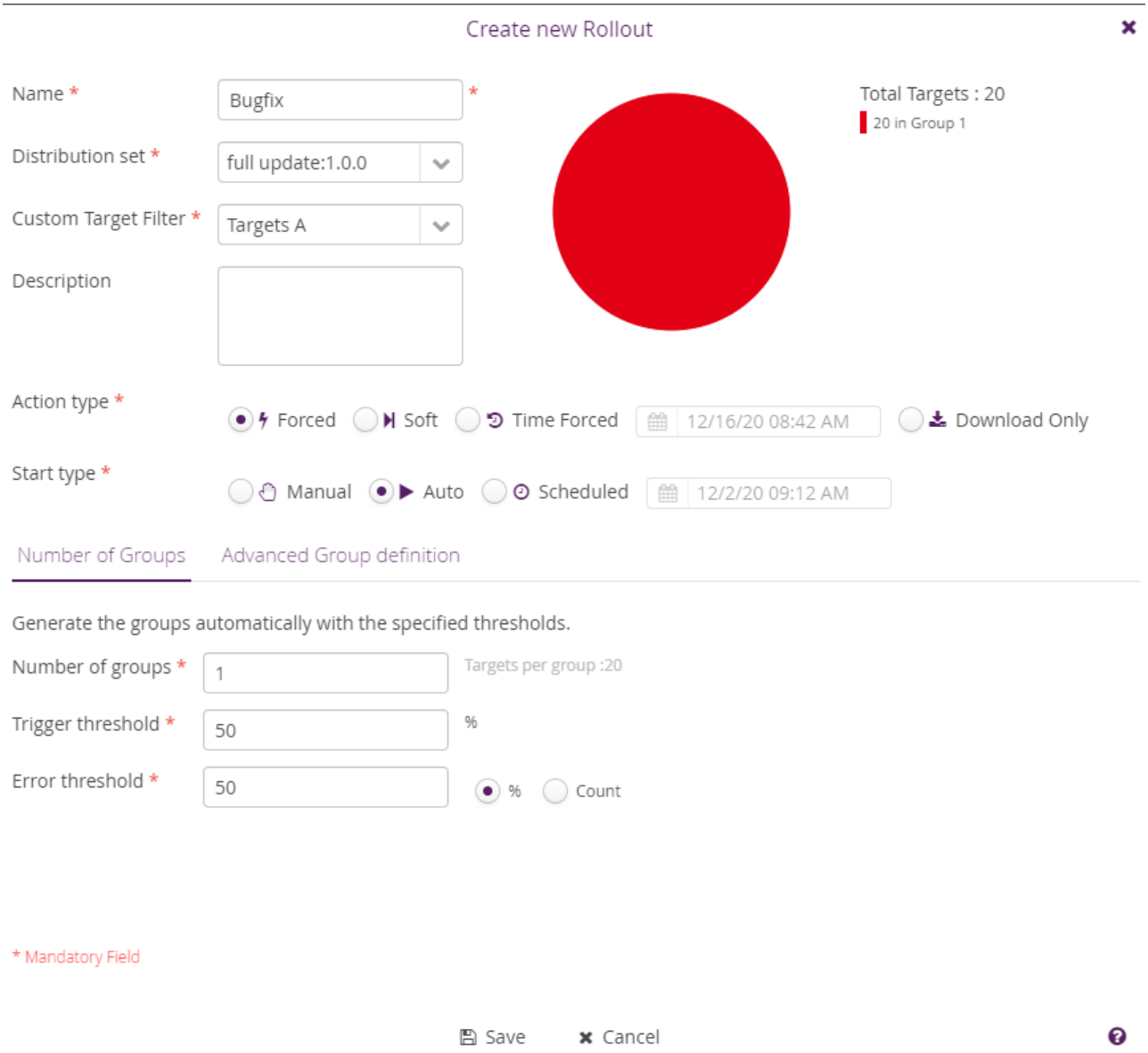


9. Rollout を作成してアップデートを開始する

   左側のメニューから"Rollout"をクリックして、Rollout Management の画面に移ります。



   "+"をクリックして Rollout を作成します。



項目	説明
Name	任意の文字列を設定します。
Distribution Set	先程作成した Distribution を選択します。
Custom Target Filter	先程作成した Target Filter を選択します。
Action Type	アップデート処理をどのように行うかを設定します。 ・Forced/Soft: 通常のアップデート ・Time Forced: 指定した時刻までにアップデートする ・Download only: ダウンロードのみ行う
Start Type	Rollout の実行をどのように始めるかを設定します。 ・Manual: 後で手動で開始する ・Auto: Target からのハートビートで開始する ・Scheduled: 決まった時間から開始する

10. アップデートの状態を確認する

    Rollout Management の画面の Detail Status で、各 Rollout のアップデートの状態を確認できます。

    アップデート中は黄色、アップデートが正常に完了すると緑色になります。

5.6.3.1. hawkBit のアップデート管理を CLI で行う

一つのアップデートを登録するには、hawkBit の Web UI で必要な手順が長いので CLI で行うことで効率よく実行できます。

サーバーの設定の段階では、「mkswu」のユーザーを作成する必要があります。作成していない場合は `setup_container.sh --add-user mkswu` で作成してください。

1. hawkbit_push_update の実行例

```
[ATDE ~/mkswu]$ ls enable_sshd.swu ❶
enable_sshd.swu

[ATDE ~/mkswu]$ hawkbit_push_update --help
Usage: /usr/bin/hawkbit_push_update [options] file.swu

rollout creation:
  --no-rollout: only upload the file without creating a rollout ❷
  --new: create new rollout even if there already is an existing one ❸
  --failed: Apply rollout only to nodes that previously failed update ❹

post action:
  --start: start rollout immediately after creation ❺

[ATDE ~/mkswu]$ hawkbit_push_update --start enable_sshd.swu ❻
Uploaded (or checked) image extra_os.sshd 1 successfully
Created rollout extra_os.sshd 1 successfully
Started extra_os.sshd 1 successfully
```

❶ この例ではあらかじめ作成されている enable_sshd.swu を hawkBit に登録します。

❷ --no-rollout を使う場合に SWU を「distribution」として登録します。デフォルトでは rollout も作成します。テストする際、デバイスがまだ登録されていなければ rollout の段階で失敗します。

❸ 同じ SWU で rollout を二回作成した場合にエラーが出ます。もう一度作成する場合は `--new` を使ってください。

❹ 一度 rollout をスタートして、 Armadillo で失敗した場合には失敗したデバイスだけに対応した rollout を作れます。

❺ 作成した rollout をすぐ実行します。このオプションには追加の権限を許可する必要があります。

❻ スタートまで行う実行例です。実行結果は Web UI で表示されます。

5.6.3.2. SWU で hawkBit を登録する

デバイスが多い場合は、SWU を一度作って armadillo を自己登録させることができます。

サーバーの設定の段階では、「device」のユーザーを作成する必要があります。作成していない場合は `setup_container.sh --add-user device` で作成してください。

1. hawkbit_register.desc で hawkBit の自己登録を行う例

```
[ATDE ~]$ cd mkswu/
```

```
[ATDE ~/mkswu]$ cp /usr/share/mkswu/examples/hawkbit_register.* . ❶

[ATDE ~/mkswu]$ vi hawkbit_register.sh ❷
# Script configuration: edit this if required!
# user given here must have CREATE_TARGET,READ_TARGET_SECURITY_TOKEN permissions
HAWKBIT_USER=device
HAWKBIT_PASSWORD="CS=wC,zJmrQeeKT.3" ❸
HAWKBIT_URL=https://10.1.1.1 ❹
HAWKBIT_TENANT=default
# set custom options for suricatta block or in general in the config
CUSTOM_SWUPDATE_SURICATTA_CFG="" # e.g. "polldelay = 86400;"
CUSTOM_SWUPDATE_CFG=""
# set to non-empty if server certificate is invalid
SSL_NO_CHECK_CERT=
# or set to cafile that must have been updated first
SSL_CAFILE=
# ... or paste here base64 encoded crt content
SSL_CA_BASE64="
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUJlakNDQVNHZ0F3SUJBZ0lVYTMvYXpNSHZ0
bFFnaFZnZDhIZWhMaEwxNm5Bd0NnWUlLb1pJemowRUF3SXcKRXpFUk1BOEdBMVVFQXd3SU1UQXVN
UzR4TGpFd0hoY05Nakl3TWpFNE1EVTFNakV6V2hjTk16SXdNakUyTURVMQpNakV6V2pBVE1SRXdE
d1lEVlFRRERBZ3hNQzR4TGpFdU1UQlpNQk1HQnlxR1NNNDlBZ0VHQ0NxR1NNNDlBd0VICkEwSUFC
RFJGcnJVV3hHNnBHdWVoejRkRzVqYkVWTm5scHUwYXBHT1c3UVBPYUF4cWp1ZzJWYjk2UHNScWJY
Sk8KbEFDVVo2OStaMHk3clBqeDJHYnhDNms0czFHalV6QlJNQjBHQTFVZERnUVdCQlJtZzhxL2FV
OURRc3EvTGE1TgpaWFdkTHROUmNEQWZCZ05WSFNNRUdEQVdnQlJtZzhxL2FVOURRc3EvTGE1TlpY
V2RMdE5SY0RBUEJnTlZIUk1CCkFmOEVCVEFEQVFIL01Bb0dDQ3FHU000OUJBTUNBMGNBTUVRQ0lB
ZTRCQ0xKREpWZnFTQVdRcVBqNTFmMjJvQkYKRmVBbVlGY2VBMU45dE8rN0FpQXVvUEV1VGFxWjhH
UFYyRUg1UWdOMFRKS05SckJDOEtpNkZWcFlkRUowYWc9PQotLS0tLUVORCBDRVJUSUZJQ0FURS0t
LS0tCg== ❺
"
# ... or add your own options if required
CURLOPT=-s

:（省略）

[ATDE ~/mkswu]$ cat hawkbit_register.desc ❻
:（省略）
swdesc_script hawkbit_register.sh --version extra_os.hawkbit 1

[ATDE ~/mkswu]$ mkswu hawkbit_register.desc ❼
hawkbit_register.swu を作成しました。

[ATDE ~/mkswu]$ mkswu initial_setup.desc hawkbit_register.desc ❽
hawkbit_register.desc を組み込みました。
initial_setup.swu を作成しました。
```

❶ hawkbit_register.sh と .desc ファイルをカレントディレクトリにコピーします。

❷ hawkbit_register.sh を編集して、設定を記載します。

❸ hawkBit の設定の時に入力した「device」ユーザーのパスワードを入力します。この例のパスワードは使用しないでください。

❹ hawkBit サーバーの URL を入力します。

❺ TLS を使用の場合に、コンテナ作成の時の証明書を base64 で入力します。

❻ hawkbit_register.desc の中身を確認します。hawkbit_register.sh を実行するだけです。

❼ SWU を作成して、initial_setup がすでにインストール済みの Armadillo にインストールできます。

❽ または、initial_setup.desc と合わせて hawkbit_register を含んだ initial_setup.swu を作成します。

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。

に「図 5.18. eMMC の予備領域使用率を確認する」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/mmcblk2 | grep EXT_CSD_PRE_EOL_INFO
eMMC Pre EOL information [EXT_CSD_PRE_EOL_INFO]: 0x01
```

図 5.18 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「3.3.6. 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
- 物理的に起動できなくする

6. 応用編

本章では、ここまでの内容で紹介しきれなかった、より細かな Armadillo の設定方法や、開発に役立つヒントなどを紹介します。

各トピックを羅列していますので、目次の節タイトルからやりたいことを探して辞書的にご使用ください。

6.1. persist_file について

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

そのため、ファイルを変更した後 Armadillo の電源を切ると変更内容は保持されません。開発中などに rootfs の変更内容を保持するには、変更したファイルに対して persist_file コマンドを使用します。

開発以外の時は安全のため、ソフトウェアアップデートによる更新を実行してください。SWUpdate に関しては 「3.2.3. アップデート機能について」 を参照してください。

rootfs の内容を変更しても、ソフトウェアアップデートを実施した際に変更した内容が保持されない可能性があります。ソフトウェアアップデート実施後も変更内容を保持する手順に関しては 「6.5. swupdate_preserve_files について」 を参照してください。

persist_file コマンドの概要を 「図 6.1. persist_file のヘルプ」 に示します。

```
[armadillo ~]# persist_file -h
Usage: /usr/bin/persist_file [options] file [more files...]

Mode selection:
  (none) single entry copy
  -d, --delete   delete file
  -l, --list     list content of overlay
  -a, --apk      apk mode: pass any argument after that to apk on rootfs
  -R, --revert   revert change: only delete from overlay, making it
                 look like the file was reverted back to original state

Copy options:
  -r, --recurse  recursive copy (note this also removes files!)
  -p, --preserve make the copy persist through baseos upgrade
                 by adding entries to /etc/swupdate_preserve_files
  -P, --preserve-post   same, but copy after upgrade (POST)

Delete options:
  -r, --recurse  recursively delete files

Common options:
  -v, --verbose  verbose mode for all underlying commands

Note this directly manipulates overlayfs lower directories
so might need a reboot to take effect
```

図 6.1 persist_file のヘルプ

1. ファイルの保存・削除手順例

```
[armadillo ~]# echo test > test
[armadillo ~]# persist_file -rv /root
'/root/test' -> '/mnt/root/test' ❶
'/root/.ash_history' -> '/mnt/root/.ash_history'
[armadillo ~]# rm -f test
[armadillo ~]# persist_file -rv /root
removed '/mnt/root/test' ❷
removed '/mnt/root/.ash_history' ❸
'/root/.ash_history' -> '/mnt/root/.ash_history'
```

図 6.2 persist_file 保存・削除手順例

❶ 追加・変更したファイルを rootfs へコピーします。

❷ -r を指定すると、ひとつ前の rm -f コマンドで削除したファイルが rootfs から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 6.3 persist_file ソフトウェアアップデート後も変更を維持する手順例

❶ 何らかのファイルの内容を変更します。

❷ -P オプションを付与して persist_file を実行します。

❸ swupdate_preserve_files に追加されたことを確認します。

3. 変更ファイルの一覧表示例

```
[armadillo ~]# mkdir dir
[armadillo ~]# persist_file -l
directory          /
directory          /root
opaque directory   /root/dir ❶
whiteout           /root/test ❷
regular file       /root/.ash_history
directory          /etc
regular file       /etc/resolv.conf
directory          /var
```

```
symbolic link      /var/lock
:（省略）
```

図 6.4 persist_file 変更ファイルの一覧表示例

❶ rootfs のファイルを見せないディレクトリは opaque directory と表示されます。

❷ 削除したファイルは whiteout と表示されます。

4. パッケージをインストールする時は apk コマンドを使用してメモリ上にインストールできますが、persist_file コマンドで rootfs に直接インストールすることも可能です。

```
[armadillo ~]# persist_file -a add strace
(1/3) Installing fts (1.2.7-r1)
(2/3) Installing libelf (0.185-r0)
(3/3) Installing strace (5.14-r0)
Executing busybox-1.34.1-r3.trigger
OK: 251 MiB in 188 packages
Install succeeded, but might not work in the running system
Please reboot if installed program does not work ❶
[armadillo ~]# strace ls
:（省略）
exit_group(0)                           = ?
+++ exited with 0 +++
```

図 6.5 persist_file でのパッケージインストール手順例

❶ この例では Armadillo を再起動せずにインストールしたコマンドを使用できましたが、Armadillo の再起動が必要となるパッケージもありますので、その場合は Armadillo を再起動してください。

6.2. コンテナ

Armadillo Base OS において、ユーザーアプリケーションは基本的にコンテナ内で実行されます。「3. 開発編」で紹介した開発手順では、基本的に SWUpadate 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-IoT ゲートウェイ G4 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-IoT ゲートウェイ G4 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-IoT ゲートウェイ G4 がある場合でも、全てのボード上で同一の環境を再現させることが可能となります。

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

6.2.2.1. イメージからコンテナを作成する

イメージからコンテナを作成するためには、podman_start コマンドを実行します。podman や docker にすでに詳しいかたは podman run コマンドでも実行できますが、ここでは 「6.2.4. コンテナ起動設定ファイルを作成する」 で紹介するコンテナの自動起動の準備も重ねて podman_start を使います。イメージは Docker Hub [https://hub.docker.com] から自動的に取得されます。ここでは、簡単な例として "ls /" コマンドを実行するコンテナを作成します。

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

図 6.6 コンテナを作成する実行例

❶ コンテナのコンフィグを作成します。このファイルでは、コンテナのイメージやコマンド、デバイスへのアクセス権限を設定します。詳しい設定の説明には 「6.2.4. コンテナ起動設定ファイルを作成する」 を参照ください。

❷ コンテナのイメージを取得します。イメージが Armadillo に置いてない場合は「Error: docker.io/alpine: image not known」の様なエラーで失敗します。

❸ コンテナを起動します。これは Armadillo 起動時に自動的に起動されるコンテナと同じものになります。自動起動が不要な場合には set_autostart no で無効化できます。

❹ podman logs コマンドで出力を確認します。

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

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

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

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

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

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

6.2.2.2. イメージ一覧を表示する

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

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

図 6.7 イメージ一覧の表示実行例

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

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

図 6.8 podman images --help の実行例

6.2.2.3. コンテナー一覧を表示する

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                    ⏎
PORTS     NAMES
```

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

図 6.9 コンテナ一覧の表示実行例

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

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

図 6.10 podman ps --help の実行例

6.2.2.4. コンテナを起動する

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

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

図 6.11 コンテナを起動する実行例

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

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

図 6.12 コンテナを起動する実行例(a オプション付与)

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

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

図 6.13 podman start --help 実行例

6.2.2.5. コンテナを停止する

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

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

図 6.14 コンテナを停止する実行例

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

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

図 6.15 podman stop --help 実行例

6.2.2.6. コンテナの変更を保存する

コンテナに対して変更が行われた状態で、そのままコンテナを停止してしまうと変更が失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 6.16 my_container を保存する例

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

2. 「3.2.4.1. 電源を切っても保持されるディレクトリ(ユーザーデータディレクトリ)」を使用する。

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

保存するデータの性質によって、保存先を選択してください。

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

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

6.2.2.7. コンテナの自動作成やアップデート

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

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

1. イメージを docker.io のイメージから作りなおします

```
[armadillo ~/podman-build]# cat Dockerfile
FROM docker.io/arm64v8/alpine:latest

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

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

[armadillo ~/podman-build]# podman build -t my_image:1 -t my_image:latest .
STEP 1: FROM docker.io/arm64v8/alpine:latest
STEP 2: RUN apk upgrade && apk add usbutils && rm -f /var/cache/apk/*
--> 234bf79175e
STEP 3: COPY my_application /my_application
--> 05ab31bb278
STEP 4: ENTRYPOINT /my_application
STEP 5: COMMIT my_image:latest
--> 590e3ba6d55
Successfully tagged localhost/my_image:1
Successfully tagged localhost/my_image:latest
590e3ba6d55f3e29bdef158d7283e9c4f7515567b2d3f978cfab2510dc02376b

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

図 6.17 podman build の実行例

2. イメージを前のバージョンからアップデートします

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

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

# update application
```

```
COPY my_application /my_application
[armadillo ~/podman-build-update]# podman build -t my_image:2 -t my_image:latest .
STEP 1: FROM localhost/my_image:latest
STEP 2: RUN apk upgrade --no-cache
--> cf1dc0d7296
STEP 3: COPY my_application /my_application
STEP 4: COMMIT my_image:latest
--> 9e9d9366072
Successfully tagged localhost/my_image:2
Successfully tagged localhost/my_image:latest
9e9d9366072751007b2e70544d76c46b95a7a5a02df658ef0fa3f7dcccf8850a

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

図 6.18 podman build でのアップデートの実行例

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

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

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

作成した .tar アーカイブは 「6.4. mkswu の .desc ファイルを編集する」 の swdesc_embed_container と swdesc_usb_container で使えます。

6.2.2.8. コンテナを削除する

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

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

図 6.19 コンテナを削除する実行例

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

1. podman rm --help 実行例

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

6.2.2.9. イメージを削除する

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

```
[armadillo ~]# podman rm my_container
[armadillo ~]# podman images
REPOSITORY                TAG         IMAGE ID      CREATED      SIZE
docker.io/library/alpine  latest      02480aeb44d7  2 weeks ago  5.62 MB
[armadillo ~]# podman rmi 02480aeb44d7
Untagged: docker.io/library/alpine:latest
Deleted: 02480aeb44d78f1a44b8791af7edf7d6e1b18707397a1dfb3ff4f21c5ce4a44f
[armadillo ~]# podman images
REPOSITORY                TAG         IMAGE ID      CREATED      SIZE
```

図 6.20 イメージを削除する実行例

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

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

図 6.21 podman rmi --help 実行例

SWU で転送されたイメージは podman images で Read-Only として表示されますので、podman rmi を実行するとエラーとなります。その場合は abos-ctrl podman-rw rmi をご使用ください。 abos-ctrl podman-rw については 「6.2.2.16. イメージを eMMC に保存する」 を参照してください。

```
[armadillo ~]# podman images
REPOSITORY              TAG       IMAGE ID     CREATED     SIZE      R/O
docker.io/library/alpine  latest     02480aeb44d7  2 weeks ago  5.62
MB    true
[armadillo ~]# podman rmi docker.io/alpine
Error: cannot remove read-only image
"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 6.22 Read-Only のイメージを削除する実行例

6.2.2.10. 実行中のコンテナに接続する

実行中のコンテナに接続し、コンテナ内で指定したコマンドを実行するには podman exec コマンドを実行します。podman exec コマンドでコンテナ内部のシェルを起動すると、コンテナ内部を操作できるよ

うになります。ここでは、sleep infinity コマンドを実行して待ち続けるだけのコンテナを作成し、そのコンテナに対して podman exec コマンドでシェルを起動する例を示します。

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

図 6.23 コンテナ内部のシェルを起動する実行例

podman_start コマンドでコンテナを作成し、その後作成したコンテナ内で sh を実行しています。sh を実行すると、コンテナ内のプロンプトが表示されコンテナ内部を操作できるようになります。上記ではコンテナ内で、ps コマンドを実行しています。コンテナ作成時に実行した sleep と podman exec で実行した sh がプロセスとして存在していることが確認できます。

コンテナ内のシェルから抜ける時は exit コマンドを実行します。

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

図 6.24 コンテナ内部のシェルから抜ける実行例

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

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

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

図 6.25 podman exec --help 実行例

6.2.2.11. コンテナ間で通信をする

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

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

準備として、2 つのコンテナを作成します。

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

図 6.26 コンテナを作成する実行例

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

```
[armadillo ~]# podman inspect --format='{{.NetworkSettings.IPAddress}}' my_container_1
10.88.0.108
[armadillo ~]# podman inspect --format='{{.NetworkSettings.IPAddress}}' my_container_2
10.88.0.109
```

図 6.27 コンテナの IP アドレスを確認する実行例

これらの IP アドレスを使って、一方のコンテナからもう一方のコンテナへ対し 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 6.28 ping コマンドによるコンテナ間の疎通確認実行例

このように、my_container_1(10.88.0.108) から my_container_2(10.88.0.109) への通信が確認できます。

6.2.2.12. pod でコンテナのネットワークネームスペースを共有する

podman_start で pod 機能を使うことができます。

pod を使うことで、複数のコンテナが同じネットワークネームスペースを共有することができます。同じ pod 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 6.29 pod を使うコンテナを自動起動するための設定例

コンテナと同じく、 /etc/atmark/containers/[NAME].conf ファイルを作って、 set_type pod を設定することで pod を作成します。

pod を使う時にコンテナの設定ファイルに set_pod [NAME] の設定を追加します。

ネットワークネームスペースは pod を作成するときに必要なため、 ports, network と ip の設定は pod のコンフィグファイルに入れなければなりません。

必要であれば、他の podman pod create のオプションを add_args で設定することができます。

.conf ファイルで使用できる各種パラメータについては、「6.2.4. コンテナ起動設定ファイルを作成する」を参照してください。

6.2.2.13. network の作成

podman_start で podman の network も作成ことができます。

デフォルトの 10.88.0.0/16 が使えない場合、あるいはコンテナ同士で接続できないようにしたい場合は使ってください。

```
[armadillo ~]# cat /etc/atmark/containers/mynetwork.conf
set_type network
set_subnet 192.168.100.0/24

[armadillo ~]# cat /etc/atmark/containers/nginx.conf
set_image docker.io/library/nginx:alpine
add_ports 80:80
set_ip 192.168.100.10
set_network mynetwork

[armadillo ~]# podman ps
CONTAINER ID  IMAGE                           COMMAND               CREATED      STATUS                  ⏎
```

```
PORTS                NAMES
3292e5e714a2  docker.io/library/nginx:alpine  nginx -g daemon o...  2 hours ago  Up 2 hours ago
0.0.0.0:80->80/tcp  nginx
```

図 6.30 network を使うコンテナを自動起動するための設定例

コンテナと同じく、 /etc/atmark/containers/[NAME].conf ファイルを作って、 set_type network を設定することで network を作成します。

そのネットワークを使う時にコンテナの設定ファイルに set_network [NAME] の設定をいれます。

ネットワークのサブネットは set_subnet [SUBNET] で設定します。この設定は set_type network の後しか使えませんので、set_type はファイルの最初のところに使ってください

他の podman network create のオプションが必要であれば、 add_args で設定することができます。

.conf ファイルで使用できる各種パラメータについては、「6.2.4. コンテナ起動設定ファイルを作成する」を参照してください。

6.2.2.14. コンテナからのコンテナ管理

podman では REST API による管理アクセスも可能です。

自分のコンテナから他のコンテナの管理が必要な場合に、ホストの podman サービスを有効にして、コンテナに /run/podman をボリュームマウントすれば podman --remote で管理できます。

コンテナの設定によって podman の socket へのパスが自動設定されない場合もあります。podman --remote でエラーが発生した場合に CONTAINER_HOST=unix:/path/to/podman.sock で socket へのパスを設定してください。

Armadillo のホスト側の udev rules からコンテナを起動する場合は podman_start 等を直接実行すると udev の子プロセス管理によってコンテナが停止されますので、その場合はサービスを有効にし、podman_start --create <container> コマンドまたは set_autostart create の設定でコンテナを生成した上 podman --remote start <container> で起動してください。

6.2.2.15. リモートリポジトリにコンテナを送信する

1. イメージをリモートリポジトリに送信する：

```
[armadillo ~]$ podman image push <localimage> docker://<registry>/<remoteimage>:<tag>
```

2. set_pull always を設定しないかぎり、SWUpdate でダウンロードの命令を送らないとアップデートを行いません。

   (mkswu については「5.4. Armadillo のソフトウェアをアップデートする」を参考にしてください)

```
[ATDE ~/mkswu]$ cp /usr/share/mkswu/examples/pull_container_nginx.desc .
[ATDE ~/mkswu]$ cp -r /usr/share/mkswu/examples/nginx_start .
```

```
[ATDE ~/mkswu]$ cat pull_container_nginx.desc
swdesc_option version=1

swdesc_pull_container "docker.io/nginx:alpine"
swdesc_files --extra-os nginx_start
[ATDE ~/mkswu]$ mkswu pull_container_nginx.desc
Enter pass phrase for /home/atmark/mkswu/swupdate.key:
pull_container_nginx.swu を作成しました。
```

6.2.2.16. イメージを eMMC に保存する

Armadillo Base OS のデフォルトでは、Podman のデータは tmpfs に保存されます。

起動時にコンテナを起動するにはイメージを eMMC に書き込む必要があります。開発が終わって運用の場合は 「6.2.2.17. イメージを SWUpdate で転送する」 でコンテナのイメージを転送します。この場合は読み取り専用の app パーティションのサブボリュームに展開します。

開発の時に以下の abos-ctrl podman-rw か abos-ctrl podman-storage --disk のコマンドを使って直接にイメージを編集することができます。

ここで紹介する内容はコンテナのイメージの管理の説明です。データベース等のコンテナから書き込みが必要な場合には 「6.2.2.6. コンテナの変更を保存する」 にあるボリュームの説明を参照してください。

- abos-ctrl podman-rw

`abos-ctrl podman-rw` を使えば、read-only になっているイメージを扱う事ができます。

```
[armadillo ~]# podman images
REPOSITORY                TAG         IMAGE ID      CREATED           SIZE        R/O
[armadillo ~]# mount /dev/sda1 /mnt
[armadillo ~]# abos-ctrl podman-rw load -i /mnt/at-debian-image.tar
Getting image source signatures
Copying blob 63c098a71e7b done
Copying blob 837e73dd4d20 done
Copying blob a25086e65f63 done
Copying config b5a30f8581 done
Writing manifest to image destination
Storing signatures
Loaded image(s): localhost/at-debian-image:latest
[armadillo ~]# podman image list
REPOSITORY                TAG         IMAGE ID      CREATED      SIZE        R/O
localhost/at-debian-image  latest      b5a30f8581cc  2 hours ago  233 MB      true
```

図 6.31 abos-ctrl podman-rw の実行例

- abos-ctrl podman-storage

`abos-ctrl podman-storage` はメモリとディスクの切り替えの他に、読み書きストレージから読み取り専用ストレージへのコピーもできます。

```
[armadillo ~]# podman pull docker.io/alpine ❶
Trying to pull docker.io/library/alpine:latest...
Getting image source signatures
Copying blob f97344484467 done
Copying config 3d81c46cd8 done
Writing manifest to image destination
Storing signatures
3d81c46cd8756ddb6db9ec36fa06a6fb71c287fb265232ba516739dc67a5f07d
[armadillo ~]# abos-ctrl podman-storage ❷
List of images configured on development storage:
REPOSITORY                TAG         IMAGE ID      CREATED     SIZE
docker.io/library/alpine  latest      3d81c46cd875  3 days ago  5.56 MB

What should we do? ([C]opy (default), [N]othing, [D]elete)
copy ❸
Create a snapshot of '/mnt/boot_1/containers_storage' in '/mnt/new_storage'
Getting image source signatures
Copying blob 8ec3165d6e61 done
Copying config 4a49b68e7c done
Writing manifest to image destination
Storing signatures
Delete subvolume (no-commit): '/mnt/new_storage'
Merging development images to readonly storage succeeded
Feel free to adjust the result with abos-ctrl podman-rw commands

Now freeing up original data...
Podman is in tmpfs mode ❹
[armadillo ~]# podman image list ❺
REPOSITORY                TAG         IMAGE ID      CREATED     SIZE        R/O
docker.io/library/alpine  latest      3d81c46cd875  3 days ago  5.56 MB     true
```

図 6.32 abos-ctrl podman-storage のイメージコピー例

❶ イメージを書き込み可能ストレージに取得します。

❷ abos-ctrl podman-storage をオプション無しで実行します。

❸ 書き込み可能ストレージにイメージがある場合に対応を聞かれます。今回はコピー（copy)します。

❹ abos-ctrl podman-storage にオプションを指定しなかったので、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 コピーの確認します。イメージが読み取り専用（R/O, Read only）になりました。

podman が壊れやすいので、デフォルトの「abos-ctrl podman-storage --tmpfs」で運用することを推奨しますが、tmpfs の容量が小さくてイメージの操作には向いてません。

開発時には「abos-ctrl podman-storage --disk」の状態で作業を行い、運用時には「abos-ctrl podman-storage --tmpfs」に戻してください。戻る際に「copy」を選択肢する場合は一時的なストレージをそのまま使いつづけますので、すべての変更が残ります。

SWUpdate でアップデートをインストールする際には、/var/lib/containers/storage_readonly ディレクトリの不要になったイメージを自動的に削除します。

自動起動させる予定がなくても、「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. VSCode から実行する

VSCode 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。

VSCode 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 「3.2.3.5. 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。

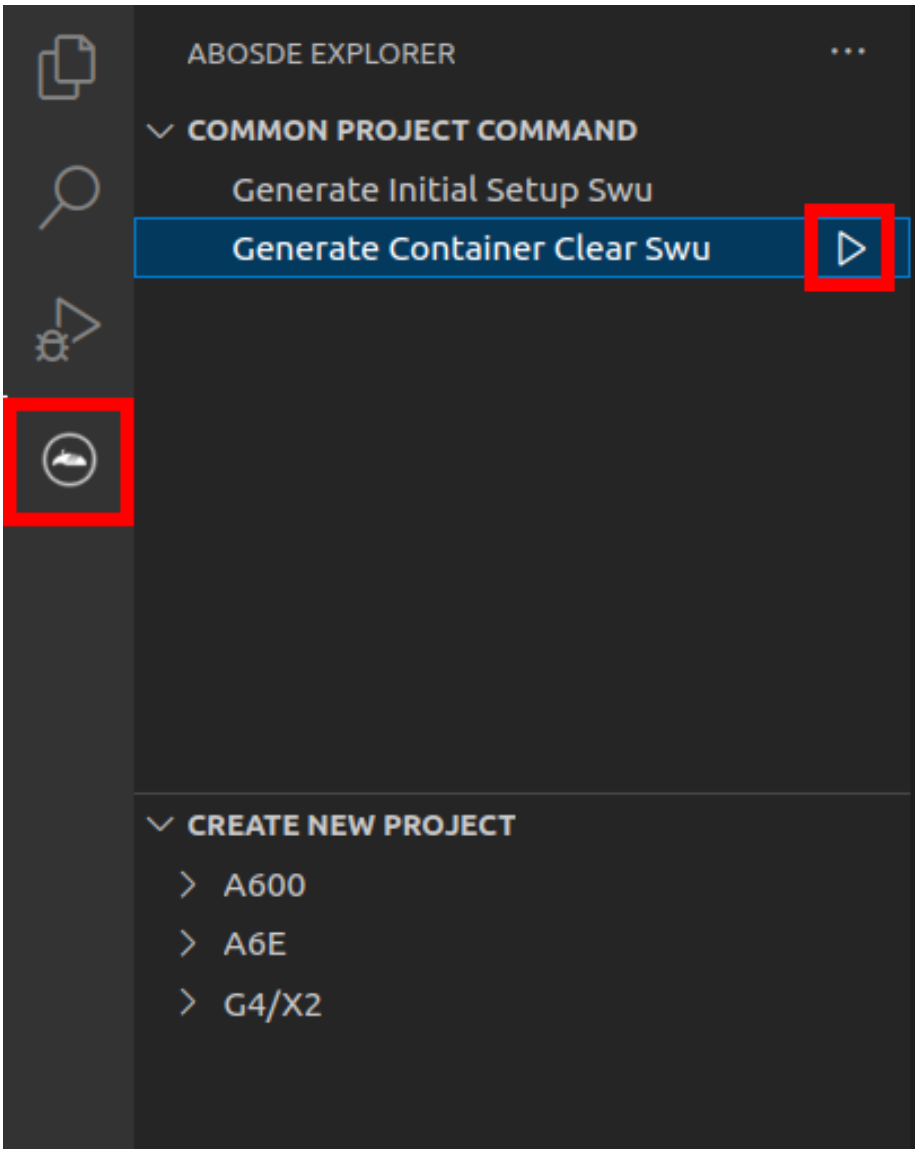


図 6.33 Armadillo 上のコンテナイメージを削除する

6.2.3.2. コマンドラインから実行する

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

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

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

・ /etc/swupdate.watch

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

図 6.34 abos-ctrl container-clear 実行例

6.2.4. コンテナ起動設定ファイルを作成する

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

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

図 6.35 コンテナを自動起動するための設定例

.conf ファイルは以下のパラメータを設定できます。

6.2.4.1. コンテナイメージの選択

set_image [イメージ名]

イメージの名前を設定できます。

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

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

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

6.2.4.2. ポート転送

add_ports [ホストポート]:[コンテナポート]

設定したポートで外部からコンテナへのアクセスが可能となります。

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

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

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

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

6.2.4.3. デバイスファイル作成

add_devices [ホストパス]:[コンテナパス]

コンテナでデバイスを作成して、使用可能となります。

コンテナパスを設定しない場合はホストと同じパスを使います。

複数のデバイスを作成したい場合はスペースで分けて設定してください。

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

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

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

コンテナパスに「:」を含むようなパスは設定できません。

6.2.4.4. ボリュームマウント

add_volumes [ホストパス]:[コンテナパス]:[オプション]

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

ホストパスは以下のどちらかを指定してください。

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

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

  SWUpdate でアップデートするデータに向いています。

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

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

  ログやデータベースに向いています。

- /tmp/<folder>: 複数のコンテナでメモリファイルシステムを共有したい場合に使ってください。

- /opt/firmware: 学習能力に必要なファムウェアライブラリーのパス。

コンテナパスを設定しない場合はホストパスと同じパスを使います。

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

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

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

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

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

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

図 6.36 ボリュームを shared でサブマウントを共有する例

❶ マウントを行うコンテナに shared の設定とマウント権限 (SYS_ADMIN) を与えます。

❷ マウントを使うコンテナに slave だけを設定すれば一方にしか共有されません。

❸ USB デバイスをマウントします。

❹ マウントされたことを確認します。

6.2.4.5. ホットプラグデバイスの追加

add_hotplugs [デバイスタイプ]

コンテナ起動後に挿抜を行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 6.37. /proc/devices の内容例」に示す状態の場合、デバイスタイプを示す文字列としては、各行の先頭の数字を除いた mem や pty などを指定できることがわかります。

```
[armadillo ~]# cat /proc/devices
Character devices:
  1 mem
  2 pty
  3 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 6.37 /proc/devices の内容例

デバイスタイプと実際のデバイスファイルの対応については、 カーネルドキュメント: devices.txt(Github) [https://github.com/torvalds/linux/blob/master/Documentation/admin-guide/devices.txt] を参照してください。

複数のデバイスタイプを指定したい場合はスペースで分けて設定してください。

例: add_hotplugs input video4linux sd

6.2.4.6. 個体識別情報の環境変数の追加

add_armadillo_env

アットマークテクノが設定した個体識別情報をコンテナの環境変数として追加することができます。

例: add_armadillo_env

add_armadillo_env を設定することで追加されるコンテナの環境変数について、「表 6.2. add_armadillo_env で追加される環境変数」に示します。

表 6.2 add_armadillo_env で追加される環境変数

環境変数	環境変数の説明	表示例
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_LAN_MAC2	アットマークテクノが設定した LAN2（eth1）の MAC アドレス	00:11:0C:12:34:57
AT_PRODUCT_NAME	製品名	Armadillo-IoT G4
AT_SERIAL_NUMBER	個体番号	00C900010001

「表 6.2. add_armadillo_env で追加される環境変数」に示した環境変数をコンテナ上で確認する場合、「図 6.38. add_armadillo_env で設定した環境変数の確認方法」に示すコマンドを実行してください。ここでは、個体番号の環境変数を例に示します。

```
[container ~]# echo $AT_SERIAL_NUMBER
00C900010001
```

図 6.38 add_armadillo_env で設定した環境変数の確認方法

お客様が独自の環境変数をコンテナに追加する場合は「図 5.6. 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. 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。

6.2.4.15. 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"

6.2.4.16. 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

「3.3.5. VSCode を使用して Armadillo のセットアップを行う」を参照して、 Armadillo のセットアッププロジェクトを作成しておいてください。

VSCode 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。

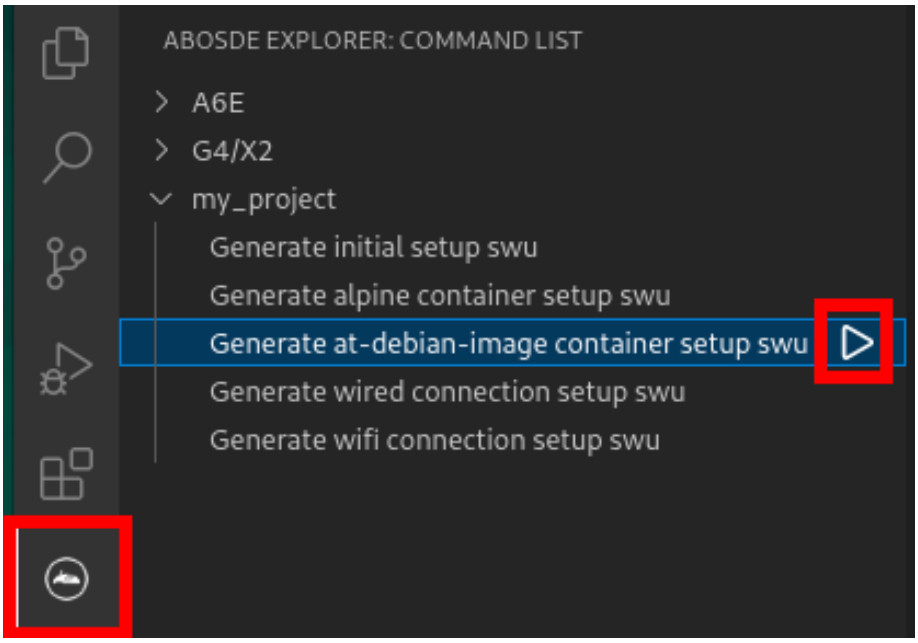


図 6.39 at-debian-image のコンテナイメージをインストールする SWU ファイルを作成する

作成した SWU ファイルは container_setup/at-debian-image/at-debian-image.swu に保存されています。この SWU イメージを 「3.2.3.5. 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。

6.2.5.2. Docker ファイルからイメージをビルドする

Armadillo-IoT ゲートウェイ G4 コンテナ [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/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 6.40 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-IoT ゲートウェイ G4 コンテナ [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/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 6.41 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。「3.3.5. VSCode を使用して Armadillo のセットアップを行う」を参照して、 Armadillo のセットアッププロジェクトを作成しておいてください。

VSCode の左ペインの [my_project] から [Generate alpine container setup swu] を実行してください。

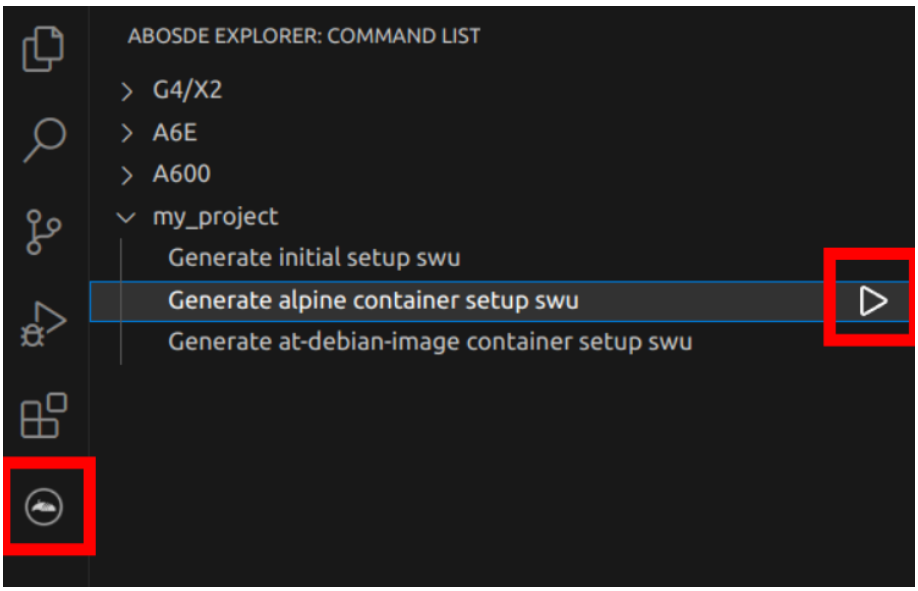


図 6.42 alpine のコンテナイメージをインストールする SWU ファイルを作成する

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 「3.2.3.5. 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。

6.2.7. 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 6.43 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 6.44 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 192.168.1.0/24 にユーザ定義のネットワークを作成する例を示します。

```
[armadillo ~]# vi /etc/atmark/containers/my_network.conf
set_type network
set_subnet 192.168.1.0/24
[armadillo ~]# podman_start my_network
Creating network 'my_network'
my_network
```

図 6.45 ユーザ定義のネットワーク作成例

コンテナを作成する際に、上記で作成したネットワークと設定したい IP アドレスを渡すことで、コンテナの IP アドレスを固定することができます。以下の例では、IP アドレスを 192.168.1.10 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 192.168.1.10
[armadillo ~]# podman_start network_example
Starting 'network_example'
3ea8c9031bf833228908bd73d8929b1d543b189b436c218e0634e0d39409e100
```

図 6.46 IP アドレス固定のコンテナ作成例

コンテナの IP アドレスが、192.168.1.10 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
192.168.1.10
```

図 6.47 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 6.48 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 6.49 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 6.50 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 6.51 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 6.52 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 6.53 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 6.54 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 6.55 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 6.56 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 6.57 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 6.58 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-IoT ゲートウェイ G4 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. Wayland を扱う

コンテナ内から、Wayland のコンポジタである weston 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/wayland_example.conf
set_image localhost/at-debian-image:latest
add_args --env=XDG_RUNTIME_DIR=/tmp ❶
add_devices /dev/dri /dev/galcore /dev/tty7 ❷
add_devices /dev/input ❸
add_volumes /run/udev:/run/udev:ro ❹
add_volumes /opt/firmware:/opt/firmware:ro ❺
add_args --cap-add=SYS_TTY_CONFIG ❻
set_command weston --tty 7 ❼
[armadillo ~]# podman_start wayland_example
Starting 'wayland_example'
654fe87422f85e8835b00761071347bafa632f969645db5fa835c88e2a55e2cc
```

図 6.59 Wayland を扱うためのコンテナ作成例

❶ weston の実行に必要な環境変数を設定します。

❷ 画面描画に必要なデバイスを設定します。

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

❻ tty を操作するための権限を設定します。

❼ weston を起動します。ここで設定する tty は add_devices の tty7 を使います。

次に、以下のように weston を起動します。オプションである --tty 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 logs wayland_example
Date: 2021-11-21 UTC
[23:46:52.823] weston 9.0.0
               https://wayland.freedesktop.org
               Bug reports to: https://gitlab.freedesktop.org/wayland/weston/issues/
               Build: lf-5.10.35-2.0.0-rc2-0-g230e9bc+
[23:46:52.825] Command line: weston --tty=7
[23:46:52.825] OS: Linux, 5.10.52-1-at, #2-Alpine SMP PREEMPT Thu Nov 18 09:10:13 UTC 2021, aarch64
[23:46:52.826] Using config file '/etc/xdg/weston/weston.ini'
[23:46:52.829] Output repaint window is 16 ms maximum.
[23:46:52.831] Loading module '/usr/lib/aarch64-linux-gnu/libweston-9/drm-backend.so'
[23:46:52.897] initializing drm backend
[23:46:52.897] logind: not running in a systemd session
[23:46:52.897] logind: cannot setup systemd-logind helper (-2), using legacy fallback
[23:46:52.902] using /dev/dri/card1
[23:46:52.902] DRM: supports atomic modesetting
[23:46:52.902] DRM: does not support GBM modifiers
```

```
[23:46:52.902] DRM: supports picture aspect ratio
[23:46:52.903] Loading module '/usr/lib/aarch64-linux-gnu/libweston-9/g2d-renderer.so'
[23:46:52.982] event1  - gpio-keys: is tagged by udev as: Keyboard
[23:46:52.983] event1  - gpio-keys: device is a keyboard
[23:46:52.986] event0  - audio-hdmi HDMI Jack: is tagged by udev as: Switch
[23:46:53.027] event0  - not using input device '/dev/input/event0'
[23:46:53.066] libinput: configuring device "gpio-keys".
[23:46:53.067] DRM: head 'LVDS-1' found, connector 39 is connected, EDID make 'unknown', model ⏎
'unknown', serial 'unknown'
[23:46:53.067] DRM: head 'HDMI-A-1' found, connector 40 is disconnected.
[23:46:53.067] Registered plugin API 'weston_drm_output_api_v1' of size 24
[23:46:53.067] Compositor capabilities:
               arbitrary surface rotation: yes
               screen capture uses y-flip: yes
               presentation clock: CLOCK_MONOTONIC, id 1
               presentation clock resolution: 0.000000001 s
[23:46:53.070] Loading module '/usr/lib/aarch64-linux-gnu/weston/desktop-shell.so'
[23:46:53.073] launching '/usr/libexec/weston-keyboard'
[23:46:53.079] Loading module '/usr/lib/aarch64-linux-gnu/libweston-9/xwayland.so'
[23:46:53.210] Registered plugin API 'weston_xwayland_v1' of size 32
[23:46:53.210] Registered plugin API 'weston_xwayland_surface_v1' of size 16
[23:46:53.210] xserver listening on display :0
[23:46:53.211] launching '/usr/libexec/weston-desktop-shell'
[armadillo ~]# podman exec -ti wayland_example bash
[container ~]# weston-terminal
```

図 6.60 コンテナ内で weston を起動したログの出力とアプリケーションの実行例

Armadillo-IoT ゲートウェイ G4 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。

・ weston の設定

アットマークテクノが提供するイメージでは、weston の設定ファイルは /etc/xdg/weston/weston.ini に配置してあります。

```
[container ~]# cat /etc/xdg/weston/weston.ini
[core]
idle-time=0
use-g2d=1
xwayland=true
repaint-window=16

[shell]
panel-position=none

[output]
name=HDMI-A-1
mode=1920x1080 ❶

[output]
name=LVDS-1
mode=off
```

図 6.61 weston.ini

❶ この行で HDMI モニタに出力する画像の解像度指定を行うことができます。初期値は 1920x1080 です。

weston.ini で解像度を指定しない場合や、指定した解像度にモニタが対応していない場合は、モニタが対応している別な解像度に自動的に切り替わります。その場合、意図しない解像度で描画されることがあります。GUI アプリケーションの描画の乱れにつながる場合がありますので、予め使用するモニタに合わせて解像度を指定しておくことをお勧めします。

設定ファイルを更新するにはコンテナイメージを新しく保存することもできますが、ボリュームを使ってこのファイルだけを更新することができます。

```
[armadillo ~]# vi /etc/atmark/containers/wayland_example.conf
...
add_volumes weston_conf:/etc/xdg/weston ❶
[armadillo ~]# mkdir /var/app/rollback/volumes/weston_conf
[armadillo ~]# cp weston.ini /var/app/rollback/volumes/weston_conf/ ❷
[armadillo ~]# podman_start wayland_example ❸
Starting 'wayland_example'
654fe87422f85e8835b00761071347bafa632f969645db5fa835c88e2a55e2cc
7a1e74510b14012110bfc4daf6f56cb0554378f513bc77554016b616e7452d58
[armadillo ~]# persist_file -v /etc/atmark/containers/
wayland_example.conf ❹
'/etc/atmark/containers/wayland_example.conf' -> '/mnt/etc/atmark/
containers/wayland_example.conf'
```

図 6.62 weston.ini をボリュームで渡す実行例

❶ コンフィグファイルにボリュームを追加します。 weston_conf は相対パスなので /var/app/rollback/volumes/weston_conf がマウントされます。ボリュームの選択については 「6.2.2.6. コンテナの変更を保存する」 を参照ください。

❷ コンフィグをコピーします。

❸ コンテナを再起動させます。

❹ 動作確認ができた後にコンフィグファイルを保存します。

・ weston の運用

コンテナの管理として、一つのコンテナで一つのアプリケーションを動かす事を推奨します。

一つのコンテナで weston を起動して、XDG_RUNTIME_DIR を共有することで別のコンテナで weston を使用するアプリケーションを起動させることは以下のコンフィグで可能です。

```
[armadillo ~]# vi /etc/atmark/containers/weston.conf ❶
set_image localhost/at-debian-image:latest
```

```
add_devices /dev/dri /dev/galcore /dev/input /dev/tty7
add_volumes /run/udev:/run/udev:ro /opt/firmware:/opt/firmware:ro
add_volumes /tmp/xdg_home:/run/xdg_home
add_args --env=XDG_RUNTIME_DIR=/run/xdg_home ❷
add_args --cap-add=SYS_TTY_CONFIG
set_command weston --tty=7
[armadillo ~]# vi /etc/atmark/containers/detect_object.conf ❸
set_image localhost/at-debian-image:latest
add_devices /dev/galcore /dev/video3
add_volumes /opt/firmware:/opt/firmware:ro /tmp/xdg_home:/run/xdg_home
set_restart always ❹
add_args --env=XDG_RUNTIME_DIR=/run/xdg_home
set_command /root/start_detect_object.sh
[armadillo ~]# podman_start weston ❺
[armadillo ~]# podman_start detect_object ❻
```

❶ weston の設定ファイルを作成します。

❷ XDG_RUNTIME_DIR を volume で共有して、同じディレクトリを使います。

❸ 例として detect_object という名前のクライアントの設定ファイルを作成します。ここでは任意の名前を設定できます。

❹ アプリケーションによっては、weston が異常終了した時にエラーを出力しない場合があるため、set_restart always にします。

❺ 確認のためコンテナを手動で起動します。

❻

・ユーザを指定して weston を起動する

アットマークテクノが提供するイメージ at-debian-image にはデフォルトで atmark ユーザが存在しています。at-weston-launch コマンドを使うと、 root ユーザではなく atmark ユーザで weston を起動することができます。

```
[armadillo ~]# vi /etc/atmark/containers/weston.conf ❶
set_image localhost/at-debian-image:latest
add_devices /dev/dri /dev/galcore /dev/input /dev/tty7 ❷
add_volumes /run/udev:/run/udev:ro /opt/firmware:/opt/firmware:ro
add_volumes /tmp/xdg_home:/run/xdg_home
add_args --env=XDG_RUNTIME_DIR=/run/xdg_home
add_args --cap-add=SYS_TTY_CONFIG
set_command at-weston-launch --tty /dev/tty7 --user atmark ❸
[armadillo ~]# podman_start weston ❹
```

❶ weston の設定ファイルを作成します。

❷ 使用する tty として /dev/tty7 を追加します。

❸ at-weston-launch コマンドのオプションとして使用する tty とユーザ名を渡します。

❹ 確認のためコンテナを手動で起動します。

--tty と --user を指定しなかった場合は、デフォルトで /dev/tty7 と atmark ユーザが使われます。

・スクリーンショットを保存する

weston を起動する際に、--debug オプションを渡すと weston-screenshooter コマンドでスクリーンショットを保存することができます。

```
[armadillo ~]# vi /etc/atmark/containers/weston.conf ❶
set_image localhost/at-debian-image:latest
add_devices /dev/dri /dev/galcore /dev/input /dev/tty7
add_volumes /run/udev:/run/udev:ro /opt/firmware:/opt/firmware:ro
add_volumes /tmp/xdg_home:/run/xdg_home
add_args --env=XDG_RUNTIME_DIR=/run/xdg_home
add_args --cap-add=SYS_TTY_CONFIG
set_command weston --tty=7 --debug ❷
[armadillo ~]# podman_start weston ❸
[armadillo ~]# podman exec -it weston /bin/bash ❹
[container ~]# weston-screenshooter ❺
[container ~]# ls
wayland-screenshot-[date].png ❻
```

❶ weston の設定ファイルを作成します。

❷ --debug オプションを渡します。

❸ 確認のためコンテナを手動で起動します。

❹ 起動した weston コンテナ内で /bin/bash を起動してログインします。

❺ weston-screenshooter コマンドを実行します。

❻ カレントディレクトリ内に wayland-screenshot-[date].png というファイル名で保存されます。

--debug オプションは開発時にのみ使用してください。正式運用時の使用は非推奨です。

Armadillo-IoT ゲートウェイ G4 にキーボードを接続している場合は、--debug オプションを渡さなくても Windows キー + s を押下することによりスクリーンショットを保存することができます。この場合、スクリーンショットはコンテナ内の /proc/[weston の PID]/cwd 下に保存されます。

6.2.9.2. 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 6.63 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 6.64 コンテナ内で X Window System を起動する実行例

Armadillo-IoT ゲートウェイ G4 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。

6.2.9.3. 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 6.65 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 6.66 フレームバッファに直接描画する実行例

6.2.9.4. 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 6.67 タッチパネルを扱うためのコンテナ作成例

Wayland 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。

6.2.9.5. VPU を扱う

Armadillo-IoT ゲートウェイ G4 で採用している i.MX 8M Plus には、動画のエンコード/デコード処理に特化した演算ユニットである VPU (Video Processing Unit) が搭載されています。VPU を活用することでシステム全体のパフォーマンスを落とすことなく、動画のエンコード/デコード処理を行うことができます。コンテナ内で動作するアプリケーションから VPU を扱うためには、コンテナ作成時にデバイスとして、/dev/mxc_hantro と /dev/mxc_hantro_vc8000e および /dev/ion を渡す必要があります。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/vpu_example.conf
set_image at-debian-image
```

```
set_command sleep infinity
add_devices /dev/mxc_hantro /dev/mxc_hantro_vc8000e
add_devices /dev/ion
[armadillo ~]# podman_start vpu_example
Starting 'vpu_example'
2aeea66c6f54abddc7d4dbdca915b279acd6962ce0c06b36c7173ec36f1c88ee
[armadillo ~]# podman exec -it vpu_example /bin/bash
[container ~]# ls /dev/mxc_hantro /dev/mxc_hantro_vc8000e /dev/ion
/dev/ion  /dev/mxc_hantro  /dev/mxc_hantro_vc8000e
```

図 6.68 VPU を扱うためのコンテナ作成例

weston と GStreamer がインストール済みのイメージと組み合わせて使うことで、VPU を使用して動画のエンコード/デコードを行うことができます。

```
[armadillo ~]# vi /etc/atmark/containers/gst_example.conf
set_image at-debian-image
set_command sleep infinity
add_devices /dev/dri /dev/galcore
add_devices /dev/mxc_hantro /dev/mxc_hantro_vc8000e
add_devices /dev/ion /dev/video3
add_devices /dev/input /dev/tty7
add_volumes /run/udev:/run/udev:ro
add_volumes /opt/firmware:/opt/firmware:ro
add_args --cap-add=SYS_TTY_CONFIG
[armadillo ~]# podman_start gst_example
Starting 'gst_example'
1332389d3c7004b623cee4227545c62aefd17b363c9a0a494d7bb217341c38ae
```

図 6.69 weston と GStreamer を扱うためのコンテナ作成例

このようにして作成したコンテナにログインすると、GStreamer で VPU を使用した動画のエンコード/デコードが行なえます。

```
[armadillo ~]# podman exec -ti gst_example bash
[container ~]# apt install gstreamer1.0-imx libgstreamer-imx gstreamer1.0-plugins-bad ¥
libgstreamer-plugins-bad1.0-0 gstreamer1.0-plugins-base libgstreamer-plugins-base1.0-0 ¥
gstreamer1.0-plugins-good libgstreamer1.0-0 gstreamer1.0-tools gstreamer1.0-imx-tools
[container ~]# weston --tty=7 &
[container ~]# gst-launch-1.0 filesrc location=<ファイル名> ! qtdemux ! h264parse ! vpudec ! queue ! ⏎
waylandsink
```

図 6.70 GStreamer によるデコード実行例

USB カメラも組み合わせると、カメラからの映像をエンコードしてファイルに保存することも可能です。

```
[container ~]# gst-launch-1.0 -e v4l2src device=/dev/video3 ! video/x- ⏎
raw,width=640,height=480,framerate=30/1 ! queue ! vpuenc_h264 ! h264parse ! queue ! qtmux ! ⏎
filesink location=./output.mp4
```

図 6.71 GStreamer によるエンコード実行例

上記を実行することで、USB カメラからの映像が H.264 にエンコードされてファイルに保存されます。この例ではカメラデバイスを /dev/video3 としていますが、環境によって異なりますので適切なものを設定してください。

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 6.72 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
Suspend-to-Idle	freeze	最も短時間で復帰することができる

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

図 6.73 サスペンド状態にする実行例

6.2.10.2. 起床要因を有効化する

サスペンド状態から起床要因として、利用可能なデバイスを以下に示します。

UART2 (console)　起床要因　データ受信

有効化

```
[container ~]# echo enabled > /sys/bus/platform/drivers/
imx-uart/30890000.serial/tty/ttymxc1/power/wakeup
```

USB　起床要因　USB デバイスの挿抜

有効化

```
[container ~]# echo enabled > /sys/bus/platform/devices/
32f10100.usb/power/wakeup
```

RTC(i.MX8MP)

起床要因　アラーム割り込み

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'
8fbef3edda3b7fcea5b1f8cbf960cf469b7e82c4d1ecd35477076e81f
c24e39f
[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] Disabling non-boot CPUs ...
...
```

図 6.74 サスペンド状態にする実行例、rtc で起こす

RV-8803-C7 は、毎分 0 秒にしかアラーム割り込みを発生させることができません。0 時 0 分 30 秒の時に、1 秒後にアラームが鳴るように設定しても、実際にアラーム割り込みが発生するのは 0 時 1 分 0 秒となります。

ユーザースイッチ

起床要因　ユーザースイッチ押下

有効化

```
[armadillo ~]# vi /boot/overlays.txt ❶
fdt_overlays=armadillo_iotg_g4-sw1-wakeup.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_iotg_g4-sw1-wakeup.dtbo
❹
```

```
：（省略）

[armadillo ~]# cat /sys/devices/platform/gpio-keys/power/
wakeup ❺
enabled
```

❶ /boot/overlays.txt ファイルに「armadillo_iotg_g4-sw1-wakeup.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。

❺ Linux からも確認できます。

SMS 受信(LTE モデルのみ)	起床要因	SMS 受信

6.2.10.3. パワーマネジメントの仕様

Armadillo-IoT ゲートウェイ G4 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-IoT ゲートウェイ G4 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

Armadillo-IoT ゲートウェイ G4 が対応するパワーマネジメント状態と、/sys/power/state に書き込む文字列の対応を次に示します。

表 6.4 対応するパワーマネジメント状態

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

起床要因として利用可能なデバイスは次の通りです。

USB コンソールインターフェース (CON6)	起床要因	データ受信
	有効化	`[armadillo ~]# echo enabled > /sys/class/tty/ttymxc1/power/wakeup`

USB インターフェース(CON4)	起床要因	USB デバイスの挿抜
	有効化	`[armadillo ~]# echo enabled > /sys/devices/platform/soc@0/32f10100.usb/power/wakeup` `[armadillo ~]# echo enabled > /sys/bus/usb/devices/usb1/power/wakeup`
RTC(SNVS_HP Real Time Counter)	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
RTC(RV-8803-C7)	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
EC25-J RI (Ring Indicator)	起床要因	SMS 受信(LTE モデルのみ)
	有効化	デフォルトで有効化されています

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 6.75 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クタイマーを使って、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ク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 6.76 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クタイマーは、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クタイマーを 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 6.77 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クタイマーを起動した後、/dev/watchdog0 に任意の文字を書き込むことでソフトウェアウォッチドッグタイマーをリセットすることができます。60 秒間任意の文字の書き込みがない場合は、システムが再起動します。

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

図 6.78 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ク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 6.79 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

6.2.13. NPU を扱う

Armadillo-IoT ゲートウェイ G4 で採用している i.MX 8M Plus には、機械学習に特化した演算処理ユニットである NPU (Neural Processor Unit) が搭載されています。NPU を活用することで、顔認識や物体認識などの推論処理を高速に行うことができます。

コンテナ内で動作するアプリケーションから NPU を扱うためには、 アットマークテクノが提供するコンテナイメージである at-debian-image を使用する必要があります。また、コンテナ作成時にデバイ

スとして、/dev/galcore を渡す必要があります。以下は、/dev/galcore を渡して at-debian-image からコンテナを作成する例です。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/npu_example.conf
set_image at-debian-image
set_command sleep infinity
add_devices /dev/galcore
add_volumes /opt/firmware:/opt/firmware:ro
[armadillo ~]# podman_start npu_example
Starting 'npu_example'
cf27a327d19d8bc37e3722fe6101c7d52fbf984351056e1c0ca4e89ff041cfbb
[armadillo ~]# podman exec -it npu_example sh
[container ~]# ls /dev/galcore
/dev/galcore
```

図 6.80 NPU を扱うためのコンテナ作成例

i.MX 8M Plus に搭載されている NPU は INT8 で量子化された学習済みモデルを高速に推論するように設計されています。INT8 で量子化されていないモデルの場合、正常に推論できない、または推論実行速度の低下が発生する場合があります。

具体的な機械学習アプリケーションの開発方法については、NXP Semiconductors の公式サイト[https://www.nxp.com/design/software/development-software/eiq-ml-development-environment:EIQ]を参照してください。

アットマークテクノからも機械学習に関する開発ガイドを公開していますので、そちらも参照してください。Armadillo Base OS 開発ガイド [https://armadillo.atmark-techno.com/resources/documents/armadillo-iot-g4/manuals]。

6.2.13.1. ONNX Runtime を使う

ONNX Runtime は 学習済みの ONNX モデルを使って推論を行うためのソフトウェアです。[1]Armadillo-IoT ゲートウェイ G4 では、NPU を使って ONNX Runtime を実行することができます。

- ONNX Runtime をインストールする

at-debian-image から作成したコンテナであれば、apt install でインストールすることができます。

```
[armadillo ~]# vi /etc/atmark/containers/onnxruntime_example.conf
set_image at-debian-image
set_command sleep infinity
add_devices /dev/galcore
add_volumes /opt/firmware:/opt/firmware:ro
[armadillo ~]# podman_start onnxruntime_example
Starting 'onnxruntime_example'
3aaefa9bdc4d7423385ee249b022ed4056a40cd66ce52b9d3eb8363379907d00
```

[1]推論実行用のソフトウェアであり、学習は行なえません。

```
[armadillo ~]# podman exec -ti onnxruntime_example bash
[container ~]# apt install onnxruntime onnxruntime-dev onnxruntime-tools python3-onnxruntime
```

図 6.81 ONNX Runtime をインストールする例

・ python から ONNX Runtime を使う

python から ONNX Runtime を使うためには onnxruntime モジュールを import します。また、NPU を使うために InferenceSession オブジェクトを作成する際に、Providers として「VsiNpuExecutionProvider」を指定します。

```
[container ~]# python3
>>> import onnxruntime
:（省略）
>>> sess = onnxruntime.InferenceSession('model.onnx', providers=['VsiNpuExecutionProvider'])
```

図 6.82 python から ONNX Runtime を使う例

以上により、python から ONNX Runtime を使うことができます。

6.2.13.2. TensorFlow Lite を使う

TensorFlow Lite からも NPU を使って高速に推論を行うことができます。

・ TensorFlow Lite をインストールする

at-debian-image から作成したコンテナであれば、apt install でインストールすることができます。

```
[armadillo ~]# vi /etc/atmark/containers/tflite_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/galcore
add_volumes /opt/firmware:/opt/firmware:ro
[armadillo ~]# podman_start tflite_example
Starting 'tflite_example'
1b7cc807b7f2857a406528f295040c817e24572f7c8abf7e6a32a62fc4f3793b
[armadillo ~]# podman exec -ti tflite_example bash
[container ~]# apt install tensorflow-lite tensorflow-lite-dev python3-tflite-runtime ¥
tim-vx tensorflow-lite-vx-delegate
```

図 6.83 TensorFlow Lite をインストールする例

・ python から TensorFlow Lite を使う

python から TensorFlow Lite を使うためには Interpreter モジュールを import します。python から TensorFlow Lite を使う場合は特別な設定をしなくても、自動的に NPU が使われます。

```
[container ~]# python3
>>> from tflite_runtime.interpreter import Interpreter
```

```
:（省略）
>>> interpreter = Interpreter('model.tflite')
```

図 6.84 python から TensorFlow Lite を使う例

以上により、python から TensorFlow Lite を使うことができます。

tflite-runtime パッケージと、ライブラリイメージ(imx_lib)のバージョンの組み合わせによっては、使用する delegate と ライブラリの整合性が取れずに TensorFlow Lite を用いたアプリケーションが正しく動作しない場合があります。

バージョン 2.6.0-1 以降の tflite-runtime パッケージを使用する際には、必ずバージョン 2.2.0 以降のライブラリイメージ(imx_lib)を使用してください。

ライブラリイメージのアップデート方法については「6.10. VPU や NPU を使用する」を参照してください。

それぞれのバージョンと動作の関係を「表 6.5. ライブラリと tflite-runtime のバージョンと NPU を用いたアプリケーションの動作の関係」に示します。

表 6.5 ライブラリと tflite-runtime のバージョンと NPU を用いたアプリケーションの動作の関係

	tflite-runtime のバージョンが 2.6.0-1 以降	tflite-runtime のバージョンが 2.6.0-1 未満
ライブラリのバージョンが確認できる(2.2.0 以降)	VX delegate で動作	NNAPI delegate で動作
ライブラリのバージョンが確認できない(2.2.0 未満)	正しく動作しない場合あり	NNAPI delegate で動作

ライブラリのバージョン確認手順については、「6.10.3. ライブラリイメージのバージョンを確認する」を参照してください。

tflite-runtime パッケージのバージョンは、コンテナ内で以下のコマンドを実行することで確認できます。

```
[container ~]# dpkg -l python3-tflite-runtime
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/
Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name                  Version      Architecture Description
+++-=====================-============-============-
==============================================
ii  python3-tflite-runtime 2.6.0-1      arm64        deep learning
framework for on-device inference
```

図 6.85 tflite-runtime のバージョンを確認する

推奨はしませんが、 tflite-runtime のバージョンを 2.6.0-1 未満に下げたい場合は「表 6.6. 2.6.0-1 未満の TensorFlow Lite 関連 deb パッケージ」に示す deb パッケージを全てコンテナ内にダウンロードして、「図 6.86. tflite-runtime のバージョンを下げる」のコマンドを実行してください。

表 6.6 2.6.0-1 未満の TensorFlow Lite 関連 deb パッケージ

パッケージ名	バージョン	URL
python3-tflite-runtime	2.4.0-1	https://download.atmark-techno.com/debian/pool/main/t/tensorflow-lite/python3-tflite-runtime_2.4.0-1_arm64.deb
python3-tflite-runtime-dbgsym	2.4.0-1	https://download.atmark-techno.com/debian/pool/main/t/tensorflow-lite/python3-tflite-runtime-dbgsym_2.4.0-1_arm64.deb
tensorflow-lite	2.4.0-1	https://download.atmark-techno.com/debian/pool/main/t/tensorflow-lite/tensorflow-lite_2.4.0-1_arm64.deb
tensorflow-lite-dev	2.4.0-1	https://download.atmark-techno.com/debian/pool/main/t/tensorflow-lite/tensorflow-lite-dev_2.4.0-1_arm64.deb

```
[container ~]# ls ./*.deb ❶
python3-tflite-runtime_2.4.0-1_arm64.deb
python3-tflite-runtime-dbgsym_2.4.0-1_arm64.deb
tensorflow-lite_2.4.0-1_arm64.deb
tensorflow-lite-dev_2.4.0-1_arm64.deb
[container ~]# apt purge ¥
tensorflow-lite ¥
tensorflow-lite-dev ¥
tensorflow-lite-vx-delegate ¥
tensorflow-lite-vx-delegate-dev ¥
python3-tflite-runtime ¥
tim-vx ¥
tim-vx-dev ❷
[container ~]# apt install ./*.deb ❸
```

図 6.86 tflite-runtime のバージョンを下げる

❶ カレントディレクトリに「表 6.6. 2.6.0-1 未満の TensorFlow Lite 関連 deb パッケージ」の deb パッケージのみ存在していることを確認します。

❷ 2.6.0-1 以上のインストールされているパッケージを削除します。
❸ ダウンロードした deb パッケージをインストールします。

6.2.13.3. Arm NN を使う

Arm NN はコンテナイメージ at-debian-image のバージョン 1.0.6 以降では、動作非対応となります。現在利用中の at-debian-image のバージョンは以下のコマンドで確認できます。

```
[armadillo ~]# podman inspect --format='{{.Config.Labels.version}}'
localhost/at-debian-image
1.0.6
```

図 6.87 at-debian-image のバージョンを確認する

Arm NN とは TensorFlow Lite および ONNX のモデル形式をサポートしている推論用ソフトウェアです。Arm NN からも NPU を使って高速に推論を行うことができます。

・ Arm NN をインストールする

at-debian-image から作成したコンテナであれば、apt install でインストールすることができます。

```
[armadillo ~]# vi /etc/atmark/containers/armnn_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/galcore
add_volumes /opt/firmware:/opt/firmware:ro
[armadillo ~]# podman_start armnn_example
Starting 'armnn_example'
18892fb9fc82f36a6de1bb9036af6e76ee552cf66e1537d70666319bb35af1e1
[armadillo ~]# podman exec -ti armnn_example bash
[container ~]# apt install libarmnn22 libarmnn-dev python3-pyarmnn armnn-examples
```

図 6.88 Arm NN をインストールする例

・ python から Arm NN を使う

python から TensorFlow Lite を使うためには pyarmnn モジュールを import します。また、NPU を使うために BackendId として「VsiNpu」を指定して、Optimize オブジェクトを作成します。

```
[container ~]# python3
>>> import pyarmnn as ann
: (省略)
>>> options = ann.CreationOptions()
>>> runtime = ann.IRuntime(options)
>>> parser = ann.ITfLiteParser()
>>> network = parser.CreateNetworkFromBinaryFile('model.tflite')
>>> preferred_backends = []
```

```
>>> preferred_backends.append(ann.BackendId('VsiNpu'))
>>> opt_network, _ = ann.Optimize(network, preferred_backends, runtime.GetDeviceSpec(),
ann.OptimizerOptions())
>>> net_id, _ = runtime.LoadNetwork(opt_network)
```

図 6.89 python から Arm NN を使う例

以上により、python から Arm NN を使うことができます。

6.3. 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、「3.2.3.2. 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ていない。

     この場合、/etc/swupdate_preserve_files に載ってるファイルのみをコピーして新しい base OS を展開します。

     この component がないと現在の rootfs のすべてがコピーされます。

  2. `extra_os.`<文字列>: rootfs の変更を行う時に使います。<文字列> には任意の文字列を指定します。

     rootfs を変更を行う時に使います。 `swdesc_*` コマンドに `--extra-os` オプションを追加すると、 component に自動的に `extra_os.` を足します。

  3. <文字列> (コンテナの名前などの任意の文字列) : rootfs の変更がないときに使います。

     この component を使うと rootfs の変更ができませんのでご注意ください。

・ アップデートを行う際にこのバージョンと現在のバージョンを比べてアップデートの判断を行います。

<component> がまだインストールされてなかった時や <version> が上がる時にインストールします。

デフォルトではダウングレードはできませんが、 --install-if=different オプションを追加することで <version> が変わる際にインストール可能になります。

アップデートの一部をインストールすることもありますので、複数の component で管理し、いくつかの古いバージョンに対応するアップデートも作成可能です。

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>] <tar_file>
swdesc_files [--dest <dest>] [--basedir <basedir>] ¥
             <file> [<more files>]
```

swdesc_tar の場合、予め用意されてある tar アーカイブをこのままデバイスで展開します。

--dest <dest> で展開先を選ぶことができます。デフォルトは / （--extra-os を含め、バージョンの component は base_os か extra_os.* の場合）か /var/app/rollback/volumes/ （それ以外の component)。後者の場合は /var/app/volumes と /var/app/rollback/volumes 以外は書けないので必要な場合に --extra-os を使ってください。

swdesc_files の場合、mkswu がアーカイブを作ってくれますが同じ仕組みです。

--basedir <basedir> でアーカイブ内のパスをどこで切るかを決めます。

・ 例えば、swdesc_files --extra-os --basedir /dir /dir/subdir/file ではデバイスに /subdir/file を作成します。

・ デフォルトは <file> から設定されます。ディレクトリであればそのまま basedir として使います。それ以外であれば親ディレクトリを使います。

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. 起動中の Armadillo で任意のコマンドを実行する

- swdesc_command_nochroot, swdesc_script_nochroot, swdesc_exec_nochroot で起動中のシステム上でコマンドを実行します。

  このコマンドは nochroot なしのバージョンと同じ使い方で、現在起動中のシステムに変更や確認が必要な場合にのみ使用してください。

nochroot コマンドは確認を一切しないため、Armadillo が起動できない状態になる可能性もあります。充分にご注意ください。

例が必要な場合は /usr/share/mkswu/examples/firmware_update.desc を参考にしてください。

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 imx-boot 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: インストールされたあるコンテナと /etc/atmark/containers/*.conf をすべて削除します。

  このオプションは簡単な初期化と考えてください。通常の運用では、不要になったイメージは自動的に削除されますのでこのオプションを設定する必要はありません。

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-IoT ゲートウェイ G4 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、「6.25. 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-IoT ゲートウェイ G4 向けのイメージをインストールする方法

Armadillo-IoT ゲートウェイ G4 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-IoT ゲートウェイ G4 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. 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。

6.8.1. 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、「6.8.2. 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。

6.8.2. 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。

6.8.3. 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 6.90. コンテナ管理」の画面に遷移します。

図 6.90 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。

「3.8.10.3. 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。

6.8.4. 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、「3.2.3.2. 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 6.91. SWU インストール」の画面に遷移します。

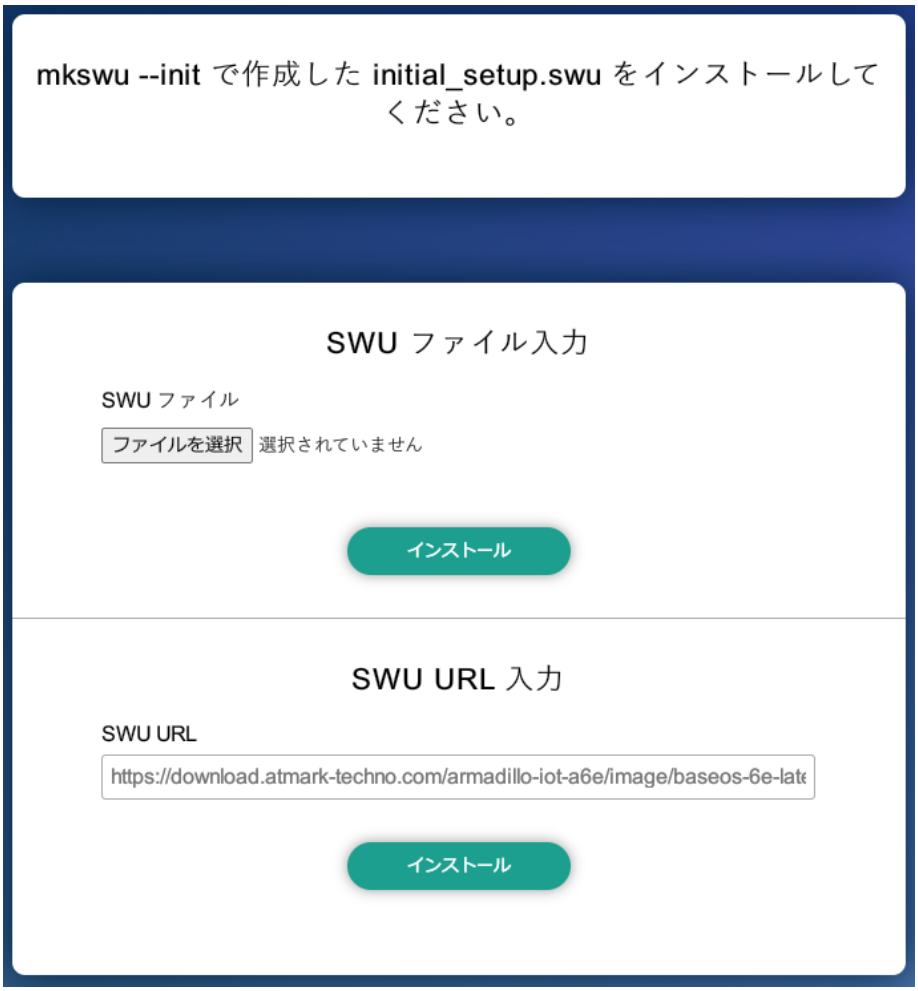


図 6.91 SWU インストール

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。

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 6.92 SWU 管理対象ソフトウェアコンポーネントの一覧表示

6.8.5. 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 6.93. 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 6.93 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 6.94. 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 6.94 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 6.95. 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 6.95 ネットワークタイムサーバーの設定項目

最後に、「図 6.96. 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 6.96 タイムゾーンの設定項目

6.8.6. 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。

6.8.6.1. 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:

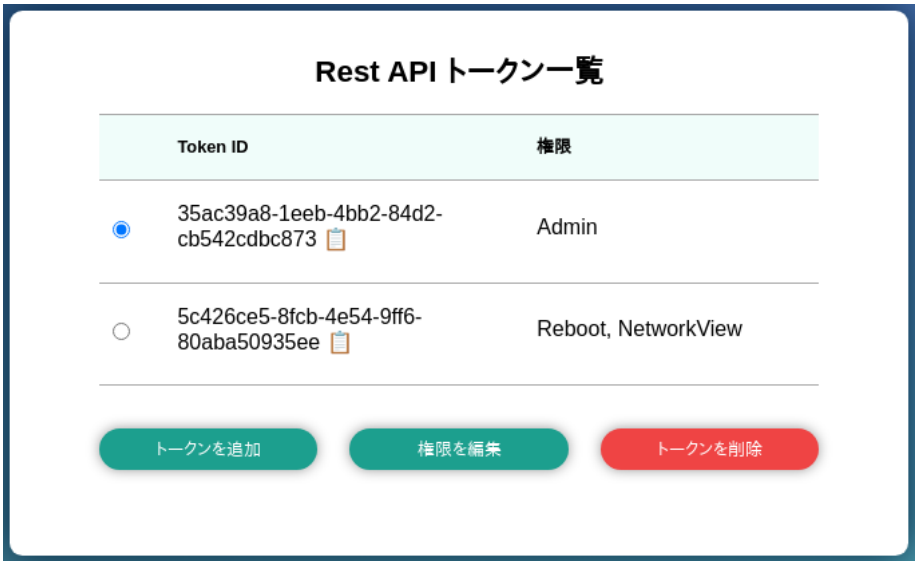


図 6.97 設定管理の Rest API トークン一覧表示

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。

6.8.6.2. 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" '
```

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" '
```

6.8.6.3. 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
```

6.8.6.4. 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
```

6.8.6.5. 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
```

6.8.6.6. Rest API : コンテナ操作

- **コンテナ一覧**
  GET "/api/containers"
  必要権限: ContainerView
  パラメータ: 無し
  出力: 各コンテナの id, name, state, command, image 情報

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/containers
{"containers":
[{"id":"02616122dcea5bd75c551b29b2ef54f54e09f59c50ce3282684773bc6bfb86a8","name":"python_app
","state":"running","command":["python3","/vol_app/src/main.py"],"image":"localhost/
python_arm64_app_image:latest"}]}
http code: 200
```

- **コンテナログ取得**
  GET "/api/containers/{container}/logs"
  必要権限: ContainerView
  パラメータ: follow=true (podman logs -f と同様の効果)
  出力: podman logs プロセスの出力 (stdout または stderr)、またはアップデートプロセスの出力ステータス (exit_code または exit_signal)

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/containers/python_app/logs
{"stdout":"Some message¥n"}
{"exit_code":0}

http code: 200
```

  follow=true を付与する例

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/containers/python_app/logs?follow=true
{"stdout":"Some message¥n"}
Ctrl-C で終了
```

- **コンテナ起動**
  POST "/api/containers/{container}/start"
  必要権限: ContainerAdmin
  パラメータ: 無し
  出力: 無し

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

http code: 200
```

- **コンテナ停止**
  POST "/api/containers/{container}/stop"
  必要権限: ContainerAdmin
  パラメータ: 無し
  出力: 無し

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

http code: 200
```

6.8.6.7. Rest API : ネットワーク設定

- **ネットワーク接続一覧**
  GET "/api/connections"
  必要権限: NetworkView
  パラメータ: 無し
  出力: ネットワーク接続一覧と各接続の uuid, name, state, ctype, あれば 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
```

- **ネットワーク接続詳細取得**
  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
```

- **ネットワーク接続の変更**
  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
```

6.8.6.8. 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
```

6.8.6.9. Rest API : 電源制御

- **再起動**
  POST "/api/reboot"
  必要権限: Reboot
  パラメータ: 無し
  出力: 無し

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

  http code: 200
  ```

- **停止**
  POST "/api/poweroff"
  必要権限: Poweroff
  パラメータ: 無し
  出力: 無し

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

  http code: 200
  ```

6.8.7. カスタマイズ

ABOS Web をお客様の最終製品へ組み込む場合に、ロゴ画像や背景色、メニューの文言などをカスタマイズすることができます。詳細は 「3.9. ABOS Web をカスタマイズする」 を参照してください。

6.9. 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 6.98. 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。

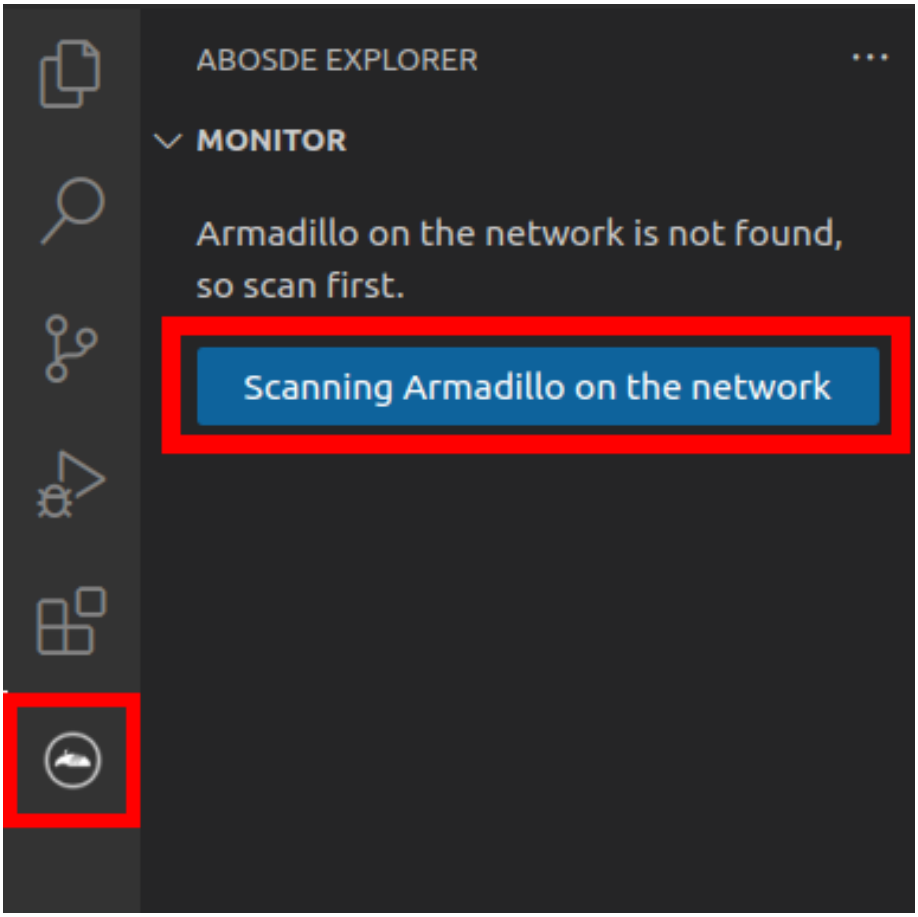


図 6.98 ABOSDE で ローカルネットワーク上の Armadillo をスキャンする

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 6.99. 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。

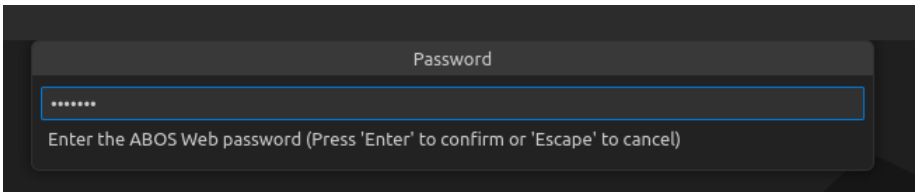


図 6.99 ABOSDE の ABOS Web パスワード入力画面

6.9.1. 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 6.100. 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。

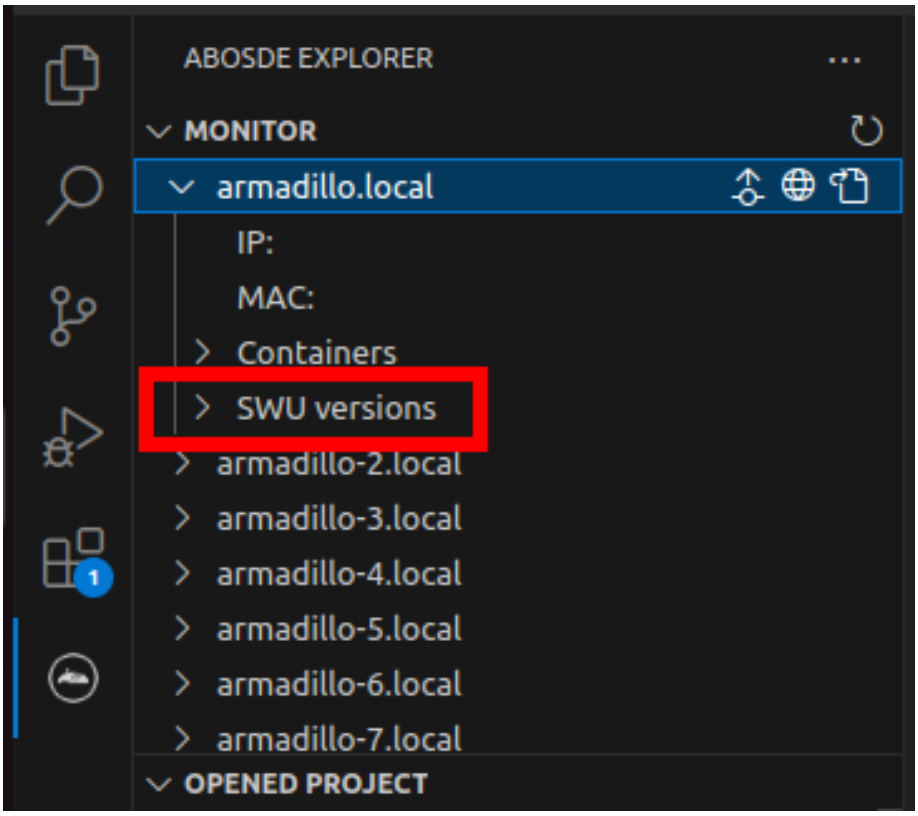


図 6.100 ABOSDE で Armadillo の SWU バージョンを取得

6.9.2. 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 6.101. 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

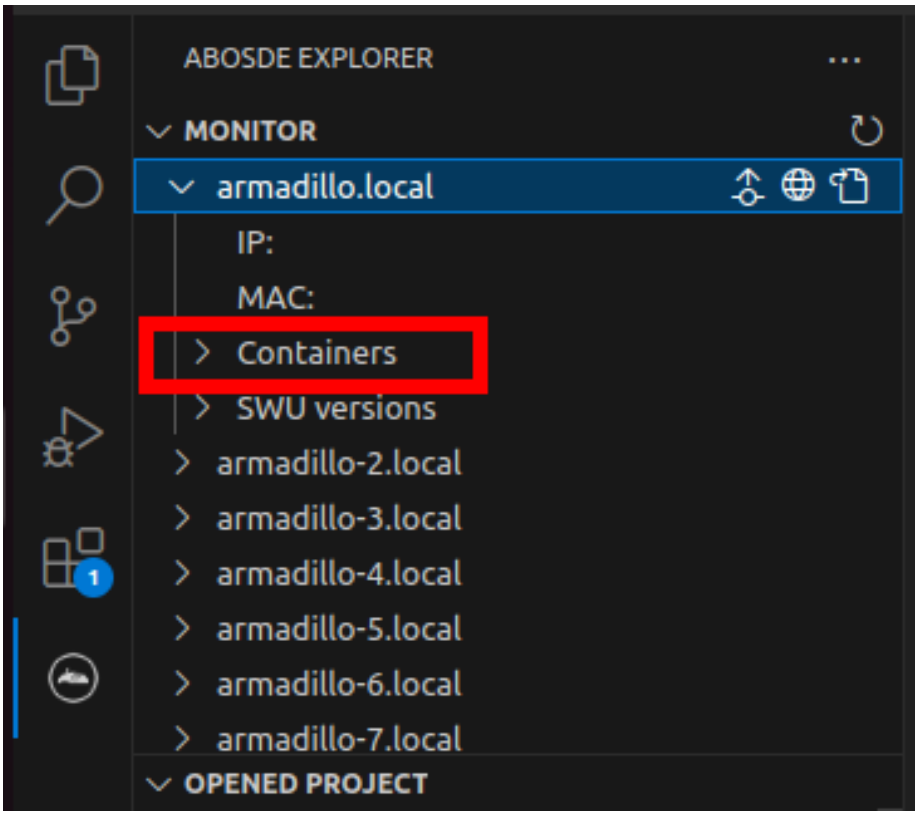


図 6.101 ABOSDE で Armadillo のコンテナ情報を取得

6.9.3. 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 6.102. 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 6.103. 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。

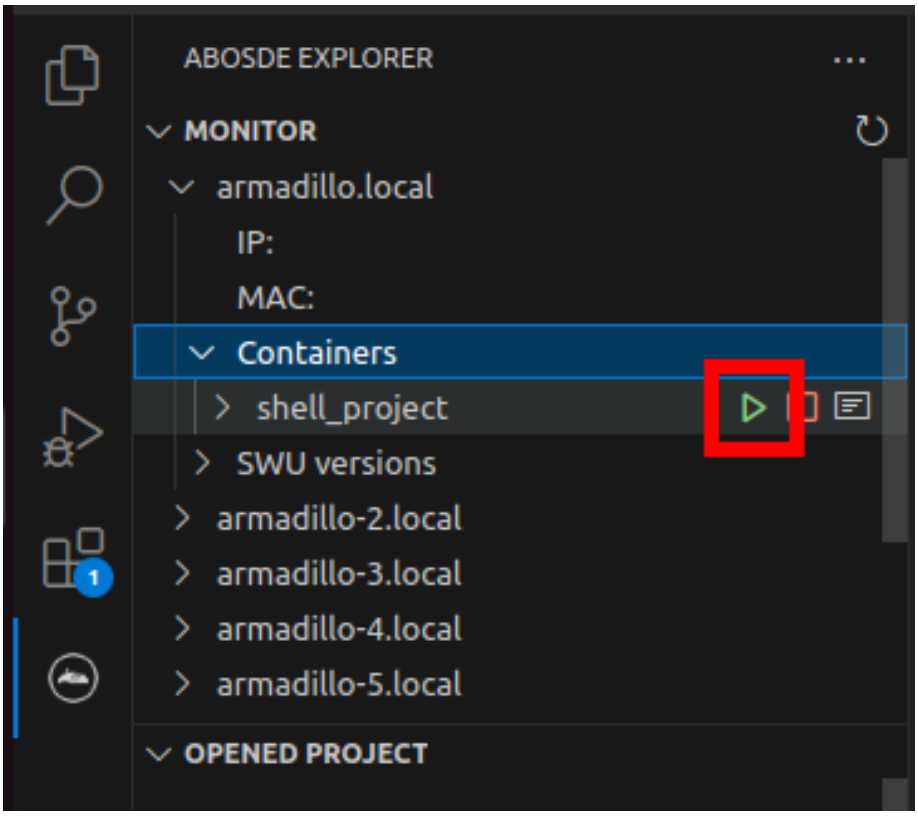


図 6.102 ABOSDE で Armadillo のコンテナを起動

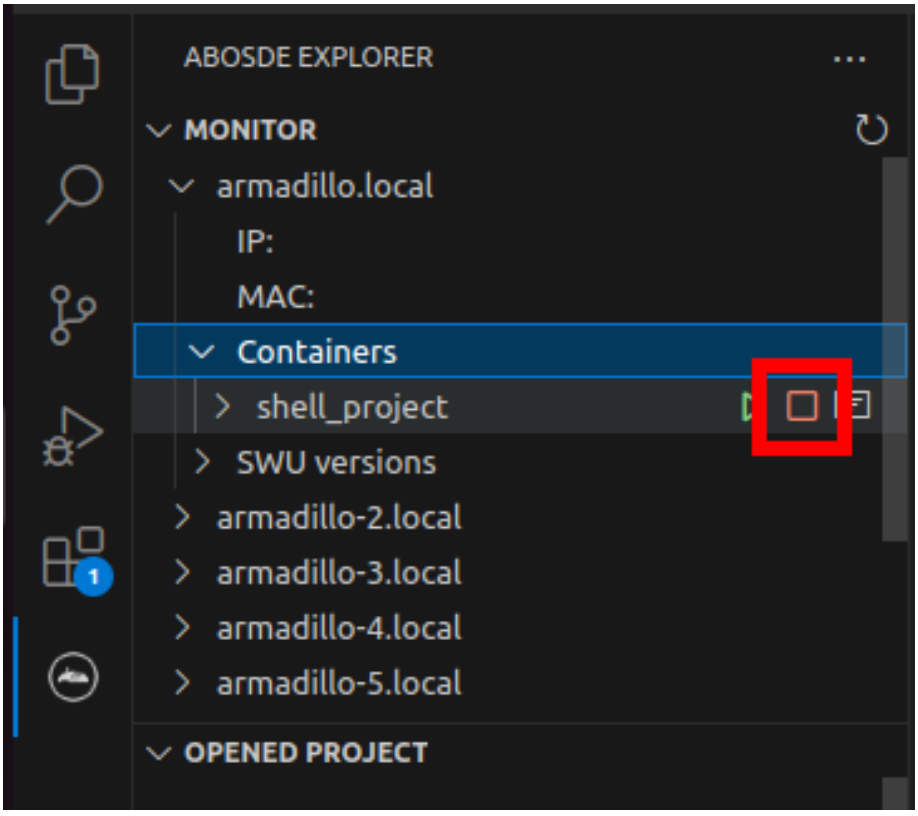


図 6.103 ABOSDE で Armadillo のコンテナを停止

6.9.4. Armadillo のコンテナのログを取得する

「図 6.104. 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 VSCode 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。

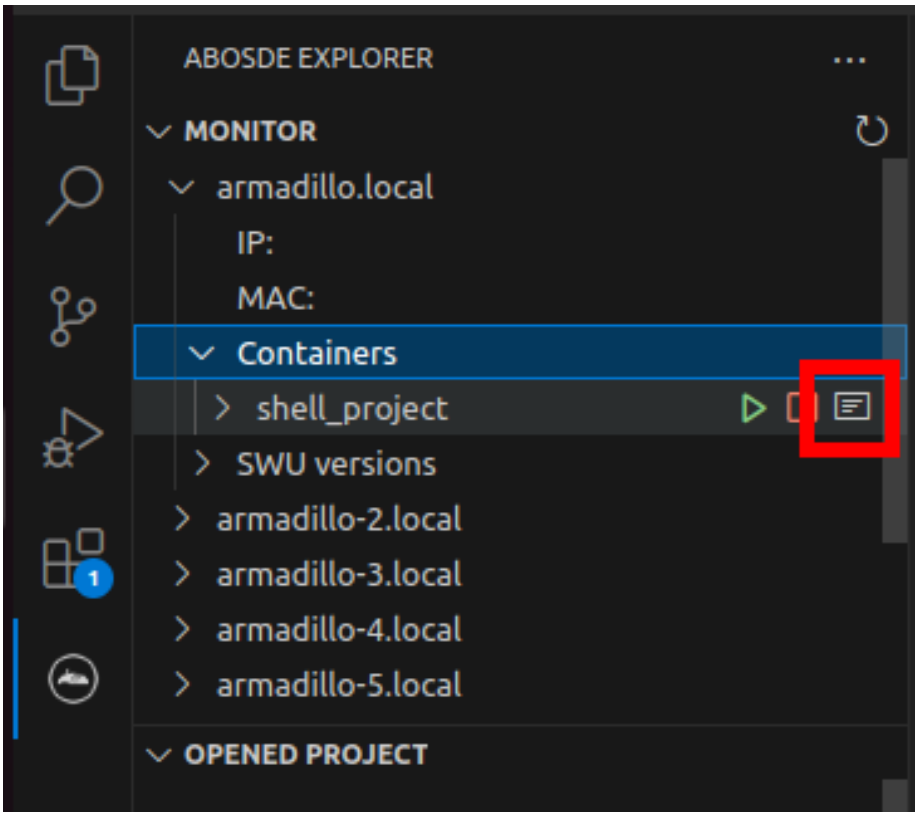


図 6.104 ABOSDE で Armadillo のコンテナのログを取得

6.9.5. 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 6.105. 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 VSCode 画面下部の OUTPUT に表示されます。

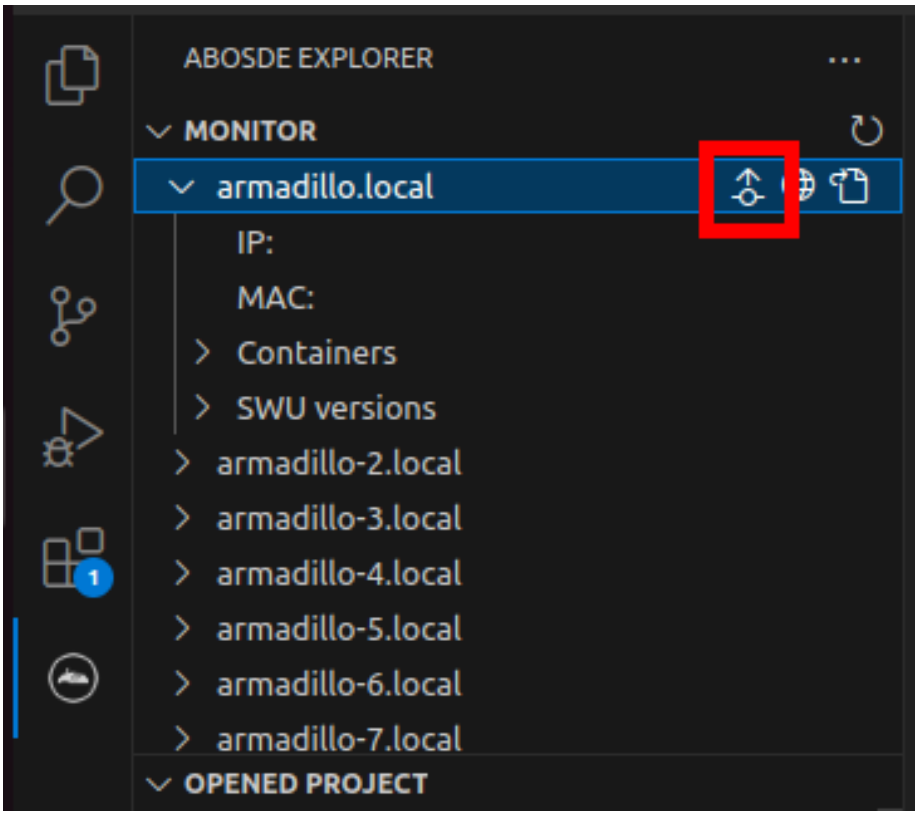


図 6.105 ABOSDE で Armadillo に SWU をインストール

6.10. VPU や NPU を使用する

VPU や NPU などを使うアプリケーションを ATDE 上で開発する場合や、Armadillo Base OS 上のコンテナ内で動作させる場合、ライブラリを ATDE 上でビルドする必要があります。ここではその手順について説明します。

予め「3.3.4.3. クロスコンパイル用ライブラリをインストールする」の手順を実施しておいてください。

6.10.1. Armadillo へ書き込むためのライブラリイメージを作成する

以下に示す製品では、出荷状態でライブラリイメージが Armadillo に書き込まれています。このため、ここで説明する手順はライブラリをアップデートする場合や、「6.24.1. ブートディスクの作成」 または「3.2.5.1. 初期化インストールディスクの作成」 の手順に従ってディスクイメージを作成する場合に必要となります。

表 6.7 ライブラリイメージ書き込み済みの製品

名称	型番
Armadillo-IoT ゲートウェイ G4 LAN モデル開発セット	AGX4500-C00D0
Armadillo-IoT ゲートウェイ G4 LAN モデル量産用	AGX4500-C00Z
Armadillo-IoT ゲートウェイ G4 LAN モデル量産ボード	AGX4500-U00Z
Armadillo-IoT ゲートウェイ G4 LTE モデル開発セット	AGX4520-C02D0
Armadillo-IoT ゲートウェイ G4 LTE モデル量産用	AGX4520-C02Z
Armadillo-IoT ゲートウェイ G4 LTE+WLAN モデル開発セット	AGX4520-C03D0
Armadillo-IoT ゲートウェイ G4 LTE+WLAN モデル量産用	AGX4520-C03Z
Armadillo-IoT ゲートウェイ G4 WLAN モデル量産用	AGX4500-C03Z

名称	型番
Armadillo-IoT ゲートウェイ G4 WLAN モデル量産ボード	AGX4500-U03Z

Armadillo Base OS 上のコンテナ内から利用できるイメージを作成します。

```
[ATDE ~]$ cd at-imxlibpackage
[ATDE ~/at-imxlibpackage]$ make-imxlibimage
```

図 6.106 ライブラリイメージ作成ツールの実行

VPU を使用しない場合は、--without-vpu オプションを付けてください。

```
[ATDE ~]$ cd at-imxlibpackage
[ATDE ~/at-imxlibpackage]$ make-imxlibimage --without-vpu
```

図 6.107 ライブラリイメージ作成ツールの実行 (VPU が不要の場合)

実行が完了すると imx_lib.img というファイルが生成されます。

6.10.2. Armadillo にライブラリイメージを書き込む

Armadillo Base OS 上で、「6.10.1. Armadillo へ書き込むためのライブラリイメージを作成する」で作成した imx_lib.img を eMMC の /dev/mmcblk2p4 パーティションに書き込みます。

次のコマンドは、 imx_lib.img が /tmp にある場合の実行例です。

```
[armadillo ~]$ umount /opt/firmware
[armadillo ~]$ dd if=/tmp/imx_lib.img of=/dev/mmcblk2p4 bs=1M conv=fsync
23+1 records in
23+1 records out
24965120 bytes (25 MB, 24 MiB) copied, 0.357741 s, 69.8 MB/s
```

図 6.108 ライブラリイメージを書き込む

書き込みが完了した後、/opt/firmware にマウントします。

```
[armadillo ~]$ mount /opt/firmware
```

図 6.109 ライブラリパーティションのマウント

6.10.3. ライブラリイメージのバージョンを確認する

Armadillo に書き込んだライブラリイメージのバージョンは、次のコマンドを実行することで確認できます。

```
[armadillo ~]$ cat /opt/firmware/etc/imxlib_version
2.2.0
```

図 6.110 ライブラリバージョンの確認

「図 6.110. ライブラリバージョンの確認」によるバージョン確認方法は、ライブラリイメージのバージョンが 2.2.0 以降の場合のみ可能です。ライブラリイメージのバージョンが 2.2.0 未満の場合、/opt/firmware/etc/imxlib_version ファイルは存在しません。

6.10.4. コンテナ内からライブラリを使用するための準備

コンテナ内からライブラリを使用するためには、コンテナ作成時にライブラリの場所を明示する必要があります。

podman_start のコンテナコンフィグに add_volumes コマンドでファームウェアが書き込まれているディレクトリ (/opt/firmware) を、add_args で podman run の --env オプションにライブラリのパスを指定します。次の例では、コンテナイメージに Debian(bullseye) を利用しています。

1. /opt/firmware を渡すコンテナコンフィグの例

```
[armadillo ~]$ vi /etc/atmark/containers/container_name.conf
add_volumes /opt/firmware:/opt/firmware:ro ❶
add_args --env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu ❷
set_image docker.io/debian:bullseye
set_command sleep infinity
[armadillo ~]# podman_start container_name
Starting 'container_name'
5c2078ff7d54082c1d18b6c4f026c36675328cea61ee6a1ab1b27145df18d72a
```

❶ add_volumes で /opt/firmware を渡します。

❷ --env に LD_LIBRARY_PATH を指定し、コンテナ内のアプリケーションからライブラリをリンクできるようにします。

次に、コンテナにログインし、/opt/firmware/usr/lib/aarch64-linux-gnu/imx-mm へのシンボリックリンクを/usr/lib/aarch64-linux-gnu/ に作成します。

```
[armadillo ~]# podman exec -it container_name /bin/bash
[container ~]# ln -s /opt/firmware/usr/lib/aarch64-linux-gnu/imx-mm /usr/lib/aarch64-linux-gnu
```

図 6.111 imx-mm へのシンボリックリンクを作成する

以上で、コンテナからライブラリを使用できるようになります。

at-debian-image のコンテナを使用する場合には、変数やリンクがすでに作成されていますので add_volumes だけでライブラリを使えます。

6.11. マルチメディアデータを扱う

6.11.1. GStreamer - マルチメディアフレームワーク

6.11.1.1. GStreamer - マルチメディアフレームワークとは

GStreamer は、オープンソースのマルチメディアフレームワークです。小さなコアライブラリに様々な機能をプラグインとして追加できるようになっており、多彩な形式のデータを扱うことができます。GStreamer で扱うことができるデータフォーマットの一例を下記に示します。

- コンテナフォーマット: mp4, avi, mpeg-ps/ts, mkv/webm, ogg
- 動画コーデック: H.264/AVC, VP8, VP9
- 音声コーデック: AAC, MP3, Theora, wav
- 画像フォーマット: JPEG, PNG, BMP
- ストリーミング: http, rtp

GStreamer では、マルチメディアデータをストリームとして扱います。ストリームを流すパイプラインの中に、エレメントと呼ばれる処理単位を格納し、それらを繋ぎ合わせることで、デコードやエンコードなどの処理を行います。

6.11.2. GStreamer 実行用コンテナを作成する

この章における GStreamer の実行例はアットマークテクノが提供する debian イメージから作成したコンテナ内で実行することを想定しています。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/gst_example.conf
set_image at-debian-image
set_command weston --tty=7
add_devices /dev/dri /dev/galcore
add_devices /dev/mxc_hantro /dev/mxc_hantro_vc8000e
add_devices /dev/ion
add_devices /dev/input /dev/tty7
add_volumes /run/udev:/run/udev:ro
add_volumes /opt/firmware:/opt/firmware:ro
add_args --cap-add=SYS_TTY_CONFIG
add_volumes /tmp/xdg_home:/run/xdg_home
add_args --env=XDG_RUNTIME_DIR=/run/xdg_home
# カメラをコンテナ内で video3 として見せます
# パスがカメラによって変わりますので、自分の環境にあわせて
# 設定してください。
add_device /dev/v4l/by-path/platform-xhci-hcd.1.auto-usb-0:1.1:1.0-video-index1 /dev/video3
[armadillo ~]# podman_start gst_example
Starting 'gst_example'
b53c127075cdd157a2118f37121451ec58f2c273b84da197d61b4468a8328a8b
[armadillo ~]# podman exec -ti gst_example bash
[container /]#
```

図 6.112 GStreamer を実行するためのコンテナ作成例

コンテナ内では最初に GStreamer をインストールします。

```
[container /]# apt update
[container /]# apt install gstreamer1.0-imx gstreamer1.0-imx-tools ¥
gstreamer1.0-tools gstreamer1.0-plugins-good gstreamer1.0-plugins-bad
```

図 6.113 gstreamer のインストール

次に、コンテナ内で画面表示を行うためのデスクトップ環境を起動します。ここでは weston を起動します。

```
[container /]# weston --tty=7 &
```

図 6.114 weston の起動

--tty=7 のオプションは画面表示に使用する tty の値を設定してください。

次に、音声を出力するのに必要な pulseaudio を起動します。

```
[conteiner /]# apt install pulseaudio
[container /]# pulseaudio --start --exit-idle-time=-1
```

図 6.115 pulseaudio の起動

以上により、GSreamer をコンテナ内で実行できるようになります。

6.11.3. GStreamer パイプラインの実行例

パイプラインの実行例を以下に示します。

```
[container ~]# gst-launch-1.0 filesrc location=<ファイルパス> ¥
! qtdemux name=demux0 demux0.video_0 ! h264parse ! queue ! vpudec ! queue ¥
! waylandsink window-width=1920 window-height=1080 demux0.audio_0 ! queue ! beepdec ! autoaudiosink
```

図 6.116 GStreamer の実行例

GStreamer のパイプラインは、シェルスクリプトのパイプ構文の構造に似ています。GStreamer の各エレメントとシェルスクリプト内のコマンドを対比することができます。構文的な違いとして、GStreamer のパイプラインは「!」を使って各エレメントを繋ぎますが、シェルスクリプトは「|」を使います。

上記例は、GStreamer のデバッグ/プロトタイピング用のコマンドラインツールである gst-launch-1.0 を使って説明しましたが、GStreamer はライブラリとして提供されているため、GStreamer を使ったマルチメディア機能を自作のアプリケーションプログラムに組み込むことができます。API やアプリケーション開発マニュアルは、gstreamer.freedesktop.org の Documentation ページ(http://gstreamer.freedesktop.org/documentation/)から参照することができます。

Armadillo-IoT ゲートウェイ G4 が採用している SoC である i.MX 8M Plus は、動画のデコード/エンコードを行うための Video Processing Unit(VPU) と呼ばれる専用プロセッサを搭載しています。

Armadillo-IoT ゲートウェイ G4 には、この VPU を使用するための GStreamer エレメントがインストールされており、以下の動画コーデックではメイン CPU のパフォーマンスを落とすことなく動画のデコード/エンコードが行なえます。

- デコード可能なコーデック
  - H.264/AVC
  - VP8
  - VP9
- エンコード可能なコーデック
  - H.264/AVC

以降の章では、これらのコーデックに対する GStreamer の実行例を紹介します。

上記で挙げたコーデック以外のものであってもデコード/エンコードは可能ですが、その場合は CPU を使ったソフトウェア処理となってしまうため、システム全体のパフォーマンスは低下します。

6.11.4. 動画を再生する

GStreamer を使用して動画を再生するための実行例を、音声を含んでいる動画と含んでいない動画の 2 通りについて示します。VPU でハードウェアデコードを行う GStreamer エレメントとして vpudec を使うことができます。

6.11.4.1. H.264/AVC 動画を再生する

```
[container ~]# gst-launch-1.0 filesrc location=<ファイルパス> ¥
! qtdemux name=demux0 demux0.video_0 ! h264parse ! queue ! vpudec ! queue ¥
! waylandsink window-width=1920 window-height=1080 demux0.audio_0 ! queue ! beepdec ! autoaudiosink
```

図 6.117 H.264/AVC 動画の再生(音声あり)

```
[container ~]# gst-launch-1.0 filesrc location=<ファイルパス> ¥
! qtdemux ! h264parse ! vpudec ! queue ! waylandsink window-width=1920 window-height=1080
```

図 6.118 H.264/AVC 動画の再生(音声なし)

6.11.4.2. VP8 動画を再生する

```
[container ~]# gst-launch-1.0 filesrc location=<ファイルパス> ¥
! matroskademux name=demux0 demux0.video_0 ! queue ! vpudec ! queue ¥
! waylandsink window-width=1920 window-height=1080 demux0.audio_0 ! queue ! beepdec ! autoaudiosink
```

図 6.119 VP8 動画の再生(音声あり)

```
[container ~]# gst-launch-1.0 filesrc location=<ファイルパス> ¥
! matroskademux ! vpudec ! queue ! waylandsink window-width=1920 window-height=1080
```

図 6.120 VP8 動画の再生(音声なし)

6.11.4.3. VP9 動画を再生する

```
[container ~]# gst-launch-1.0 filesrc location=<ファイルパス> ¥
! matroskademux name=demux0 demux0.video_0 ! queue ! vpudec ! queue ¥
! waylandsink window-width=1920 window-height=1080 demux0.audio_0 ! queue ! beepdec ! autoaudiosink
```

図 6.121 VP9 動画の再生(音声あり)

```
[container ~]# gst-launch-1.0 filesrc location=<ファイルパス> ¥
! matroskademux ! vpudec ! queue ! waylandsink window-width=1920 window-height=1080
```

図 6.122 VP9 動画の再生(音声なし)

6.11.5. ストリーミングデータを再生する

GStreamer を使用してネットワーク上にある動画ファイルを HTTP 及び RTSP でストリーミング再生する実行例を示します。VPU でハードウェアデコードを行う GStreamer エレメントとして vpudec を使うことができます。

6.11.5.1. HTTP ストリーミング

```
[container ~]# gst-launch-1.0 souphttpsrc location=<動画ファイルの URI> ¥
! qtdemux name=demux demux. ! queue ! vpudec ! queue ¥
! waylandsink demux. ! queue ! beepdec ! autoaudiosink
```

図 6.123 HTTP ストリーミングの再生(音声あり)

```
[container ~]# gst-launch-1.0 souphttpsrc location=<動画ファイルの URI> ¥
! qtdemux ! queue ! vpudec ! queue ! waylandsink
```

図 6.124 HTTP ストリーミングの再生(音声なし)

6.11.5.2. RTSP ストリーミング

```
[container ~]# gst-launch-1.0 rtspsrc location=<動画ファイルの URI> name=source ¥
! queue ! rtph264depay ! vpudec ! queue ! waylandsink source. ! queue ¥
! rtpmp4gdepay ! aacparse ! beepdec ! autoaudiosink
```

図 6.125 RTSP ストリーミングの再生(音声あり)

```
[container ~]# gst-launch-1.0 rtspsrc location=<動画ファイルの URI> ¥
! queue ! rtph264depay ! vpudec ! queue ! waylandsink
```

図 6.126 RTSP ストリーミングの再生(音声なし)

6.11.6. USB カメラからの映像を表示する

GStreamer の v4l2src エレメントを使うことで、V4L2(Video for Linux 2) デバイスとして実装されているカメラデバイスから映像を取得できます。どのデバイスから映像を取得するかは、v4l2src エレメントの device プロパティにデバイスファイル名を指定することで変更できます。UVC 対応 USB カメラなども同様に v4l2src で扱うことができるので、ここでは USB カメラからの映像を表示する実行例を示します。

加えて、カメラの他にマイクも接続していて、同時にマイクからの音声も出力する場合の例も示しています。実行例中のデバイスファイル /dev/video1 の部分や、縦横サイズである width や height の値は実行する環境によって異なる可能性がありますので、適宜変更してください。また、/dev/v4l/by-id ディレクトリの下に、接続しているカメラ名の付いた /dev/videoN へのシンボリックリンクがありますので、デバイスとしてそれを指定することも可能です。

```
[container ~]# gst-launch-1.0 v4l2src device=/dev/video1 ¥
! video/x-raw,width=640,height=480,framerate=30/1 ¥
! waylandsink window-width=640 window-height=480 pulsesrc ¥
! audio/x-raw,rate=44100,channels=2 ! autoaudiosink
```

図 6.127 USB カメラからの映像表示(音声あり)

```
[container ~]# gst-launch-1.0 v4l2src device=/dev/video1 ¥
! video/x-raw,width=640,height=480,framerate=30/1 ¥
! waylandsink window-width=640 window-height=480
```

図 6.128 USB カメラからの映像表示(音声なし)

6.11.7. USB カメラからの映像を録画する

GStreamer の v4l2src エレメントを使うことで、V4L2(Video for Linux 2) デバイスとして実装されているカメラデバイスから映像を取得できます。どのデバイスから映像を取得するかは、v4l2src エレメントの device プロパティにデバイスファイル名を指定することで変更できます。UVC 対応 USB カメラなども同様に v4l2src で扱うことができるので、ここでは USB カメラからの映像をファイルへ保存する実行例と、映像を表示しながら同時にファイルへ保存する実行例を示します。

加えて、カメラの他にマイクも接続していて、映像の保存と同時にマイクからの音声も MP3 へエンコードして保存する場合の例も示しています。実行例中のデバイスファイル /dev/video1 の部分や、縦横サイズである width や height の値は実行する環境によって異なる可能性がありますので、適宜変更してください。また、/dev/v4l/by-id ディレクトリの下に、接続しているカメラ名の付いた /dev/videoN へのシンボリックリンクがありますので、デバイスとしてそれを指定することも可能です。

パイプライン停止時に EOS イベントを発行するように、gst-launch-1.0 コマンドに-e オプションを付けています。エンコードを終了するには、Ctrl-C で gst-launch-1.0 コマンドを停止してください。

6.11.7.1. H.264/AVC で録画する

VPU でハードウェアエンコードを行う GStreamer エレメントとして vpuenc_h264 を使うことができます。

```
[container ~]# gst-launch-1.0 -e v4l2src device=/dev/video1 ¥
! video/x-raw,width=640,height=480,framerate=30/1 ¥
! queue ! vpuenc_h264 ! h264parse ! queue ! mux. pulsesrc ¥
! audio/x-raw,rate=44100,channels=2 ! lamemp3enc ! queue ¥
! mux. qtmux name=mux ! filesink location=./output.mp4
```

図 6.129 USB カメラからの映像を H.264 で録画(音声あり)

```
[container ~]# gst-launch-1.0 -e v4l2src device=/dev/video1 ¥
! video/x-raw,width=640,height=480,framerate=30/1 ¥
! queue ! vpuenc_h264 ! h264parse ! queue ¥
! filesink location=./output.mp4
```

図 6.130 USB カメラからの映像を H.264 で録画(音声なし)

・ 表示と録画を同時に行う

```
[container ~]# gst-launch-1.0 -e v4l2src device=/dev/video1 ¥
! video/x-raw,width=640,height=480,framerate=30/1 ¥
! tee name=t1 ! queue ! vpuenc_h264 ! h264parse ! queue ! mux. pulsesrc ¥
! tee name=t2 ! audio/x-raw,rate=44100,channels=2 ! lamemp3enc ! queue ¥
! mux. qtmux name=mux ! filesink location=./output.mp4 t1. ! queue ¥
! waylandsink window-width=640 window-height=480 t2. ! queue ! autoaudiosink
```

図 6.131 USB カメラからの映像を表示しながら H.264 で録画(音声あり)

```
[container ~]# gst-launch-1.0 -e v4l2src device=/dev/video1 ¥
! video/x-raw,width=640,height=480,framerate=30/1 ¥
! tee name=t1 ! queue ! vpuenc_h264 ! h264parse ! queue ¥
! qtmux ! filesink location=./output.mp4 t1. ! queue ¥
! waylandsink window-width=640 window-height=480
```

図 6.132 USB カメラからの映像を表示しながら H.264 で録画(音声なし)

6.11.8. Video Processing Unit(VPU)

6.11.8.1. Video Processing Unit とは

Video Processing Unit(以下、VPU) とは i.MX 8M Plus に搭載されている、動画のエンコード/デコード処理専用のプロセッサです。動画のエンコード/デコード処理は、システムに負荷をかけることが多く、メイン CPU で処理を行うとシステム全体のパフォーマンスが低下します。VPU を利用することでシステム全体のパフォーマンスを落とすことなく、動画のエンコード/デコード処理を行うことができます。

VPU が対応しているフォーマットは以下の通りです。

- デコーダーが対応しているフォーマット
  - H.264/AVC
  - VP8
  - VP9
- エンコーダが対応しているフォーマット
  - H.264/AVC

6.11.8.2. VPU の仕様

- H.264/AVC デコーダー

表 6.8 H.264/AVC デコーダー仕様

Profile	High、Main、Baseline
Min resolution	48x48
Max resolution	1920x1080
Frame rate	60 fps
Bitrate	60 Mbps

- VP8 デコーダー

表 6.9 VP8 デコーダー仕様

Profile	-
Min resolution	48x48
Max resolution	1920x1080
Frame rate	60 fps
Bitrate	60 Mbps

- VP9 デコーダー

表 6.10 VP9 デコーダー仕様

Profile	Profile 0, 2
Min resolution	72x72
Max resolution	1920x1080
Frame rate	60 fps
Bitrate	100 Mbps

- H.264/AVC エンコーダー

表 6.11 H.264/AVC エンコーダー仕様

Profiles	Baseline、Main、High、High 10
Maximum Luma pixel sample rate	1920x1080 @ 60 fps
Slices	I, P and B slices
Frame Types	Progressive
Entropy encoding	CABAC、CAVLC
Error resilience	Slices

Maximum MV range	Horizontal (P slice) in pixels: +/-139 Horizontal (B slice) in pixels: +/-75 Vertical (P or B slice) in pixels: ・ Config1: +/-13 (planned) ・ Config2: +/-21 ・ Config3: +/-29 (planned) ・ Config4: +/-45 (planned) ・ Config5: +/-61 (planned) (= Search Window Size -3 pixels)
MV accuracy	1/4 pixel
Supported block sizes	Macroblock and sub-macroblock partitions: ・ Intra PU: 16x16 / 8x8 / 4x4 ・ Inter PU: 16x16 / 8x16 / 16x8 ・ TU: 4x4 and 8x8 transforms
Intra-prediction modes	16x16: 4 modes 8x8: 9 modes 4x4: 9 modes
Maximum number of reference frames	2
Encoding picture type	Only progressive frame
IPCM encoding	Supported
Temporal scalable video coding	Up to 5 layers including the base layer
IPCM	IPCM rectangle mode
ROI / ROI_map	Absolute QP and qpoffset mode (-32 ～ 31) User controllable CU coded as IPCM CU or skip CU

6.12. デモアプリケーションを実行する

この章では、アットマークテクノが提供するデモアプリケーションについて説明します。デモアプリケーションは GUI アプリケーションであるため、ディスプレイを接続している必要があります。デモアプリケーションを実行するためのコンテナイメージとして、アットマークテクノが提供するコンテナイメージを想定しています。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。

また、パッケージのインストールにはデフォルトの tmpfs の容量が少ないので、あらかじめ abos-ctrl podman-storage --disk で podman のストレージを eMMC に切り替えてください。開発が終わったら必ず tmpfs に戻ってください。

6.12.1. コンテナを作成する

デモアプリケーションを実行するためのコンテナを以下のように作成します。

デモアプリケーションは GUI アプリケーションであるため、まずデスクトップ環境を起動する必要があります。ここでは weston を起動します。

```
[armadillo ~]# vi /etc/atmark/containers/demo_app.conf
set_image localhost/at-debian-image:latest
set_command weston --tty=7
add_args -ti --privileged
add_args --env=VIV_VX_ENABLE_CACHE_GRAPH_BINARY=1
add_args --env=VIV_VX_CACHE_BINARY_GRAPH_DIR=/var/cache/armadillo-demo-experience
```

```
add_volumes /sys /dev /run/udev /opt/firmware
add_volumes cache:/var/cache/armadillo-demo-experience
[armadillo ~]# podman_start demo_app
Starting 'demo_app'
b984a8cdf88539c32241618baf45651c92ab9318d82e1a7fbf2a4c0cba315efd
[armadillo ~]# podman exec -it demo_app bash
[container /]#
```

図 6.133 デモアプリケーションを実行するためのコンテナ作成例

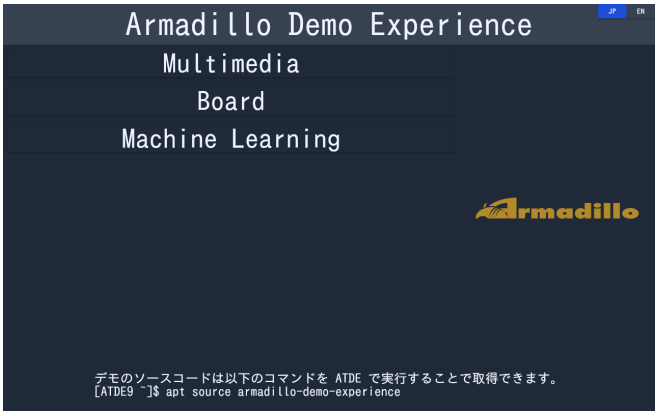
6.12.2. デモアプリケーションランチャを起動する

デモアプリケーションランチャを起動します。個々のデモアプリケーションはこのデモアプリケーションランチャから起動できます。このデモアプリケーションランチャは GUI フレームワークとして Flutter を使用しています。デモアプリケーションランチャのソースコードは、apt source で取得することができます。

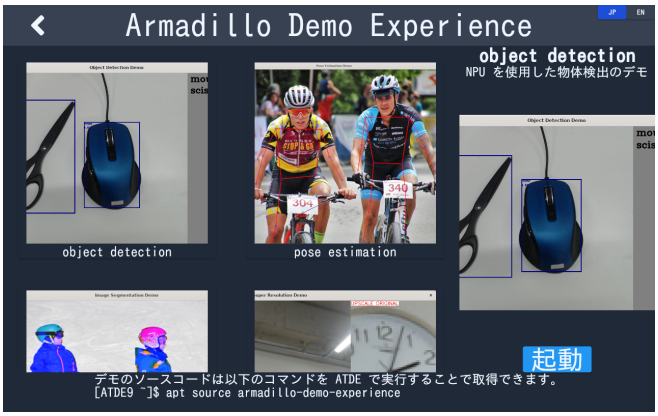
```
[container /]# apt install armadillo-demo-experience
[container /]# demoexperience
```

図 6.134 デモアプリケーションランチャの起動

以下のようなアプケーションが起動します。



左側のカテゴリから起動したいデモアプリケーションを選びます。



選んだアプリケーションは、右下の「起動」ボタンで起動することができます。

デモアプリケーションには TensorFlow Lite と NPU を使用するものが含まれています。TensorFlow Lite と NPU を扱うライブラリのバージョンによっては、デモアプリケーションが正しく動作しない場合があります。以下のバージョンになっていることを確認してください。

表 6.12 デモアプリケーション動作のために必要なバージョン

パッケージ名	必要バージョン
tensorflow-lite	2.8.0 以上
python3-tflite-runtime	2.8.0 以上
tensorflow-lite-vx-delegate	2.8.0 以上
tim-vx	1.1.39 以上

各パッケージのバージョンは、コンテナ内で以下のコマンドを実行することで確認できます。以下は、tensorflow-lite のバージョンを確認する例です。

```
[container ~]# dpkg -l tensorflow-lite
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/
Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name           Version      Architecture Description
+++-==============-============-============-
=================================================
ii  tensorflow-lite 2.8.0-1      arm64        deep learning framework
for on-device inference
```

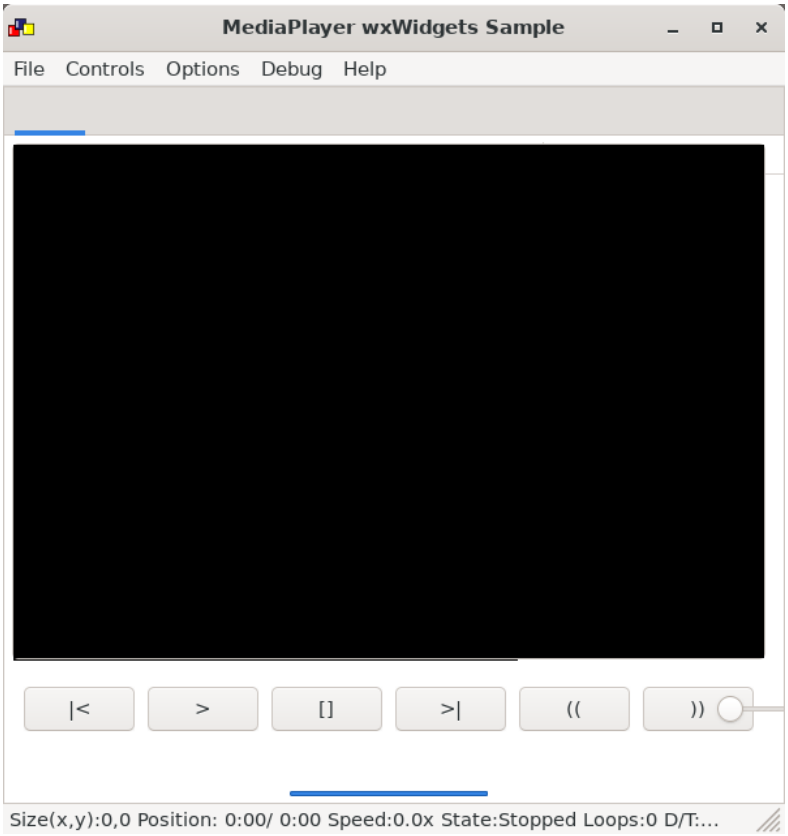
図 6.135 パッケージのバージョンを確認する

バージョンの条件を満たしていない場合は、 `apt update && apt upgrade` を実行してアップデートを行ってください。

6.12.3. mediaplayer

mediaplayer は動画を再生するアプリケーションです。H.264, VP8, VP9 でエンコードされた動画ファイルであれば、動画のデコードに VPU が使われます。File メニューから、再生したい動画ファイル

を選択することができます。このアプリケーションは、GUI フレームワークとして wxWidgets を使用しています。



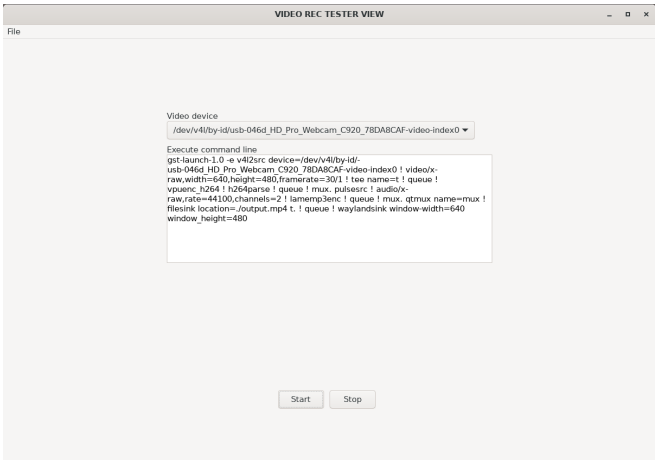
音声も出力したい場合は、pulseaudio をインストールして起動する必要があります。

```
[container /]# apt install pulseaudio
[container /]# pulseaudio --start --exit-idle-time=-1
```

図 6.136 pulseaudio のインストールと起動

6.12.4. video recoder

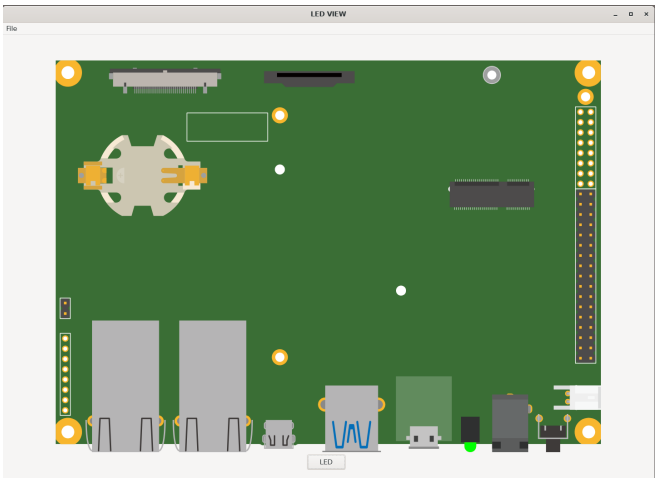
video recoder は gstreamer を使用してカメラからの映像を録画することができます。そのため、このアプリケーションを使用するためには、Armadillo 本体にカメラを接続する必要があります。カメラが接続されていると Video device の項目でカメラを選択できるようになります。カメラを選択し、Start ボタンを押すと別ウィンドウが表示され録画が開始されます。アプリケーション上のテキストボックスには、Start ボタンを押したときに起動する gstreamer のコマンドを表示しています。テキストボックスの内容はキーボードで編集可能です。このアプリケーションは、GUI フレームワークとして wxWidgets を使用しています。



マイク付きのカメラなどで同時に音声も録音したい場合は、「6.12.3. mediaplayer」 を参照して pulseaudio を起動してください。

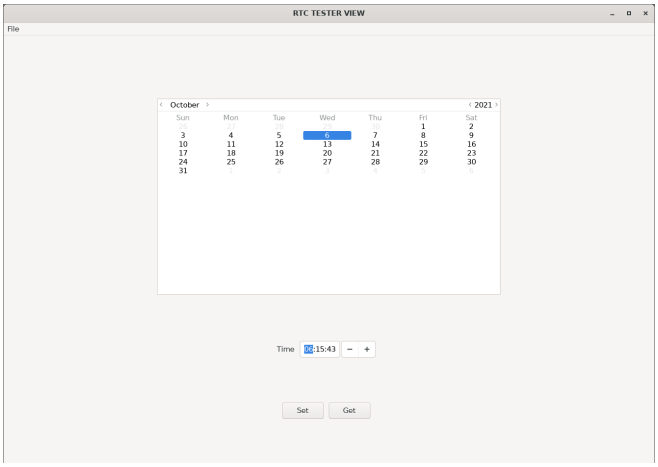
6.12.5. led switch tester

led switch tester は Armadillo 本体上の LED と SW1 を扱うアプリケーションです。LED ボタンを押すことで Armadillo 本体上の LED の 点灯・消灯を確認することができます。Armadillo 本体上の SW1 を押すとアプリケーションの SW1 部分の表示が変化することを確認できます。このアプリケーションは、GUI フレームワークとして wxWidgets を使用しています。



6.12.6. rtc tester

rtc tester は Armadillo 本体上の RTC に対して日時の設定および取得が行えるアプリケーションです。カレンダー上から日付を選び、Time に設定したい時刻を入力した後、Set ボタンを押すと RTC にその日時が設定されます。Get ボタンを押すと、現在の日時を RTC から読み込みアプリケーション上に反映されます。このアプリケーションは、GUI フレームワークとして wxWidgets を使用しています。



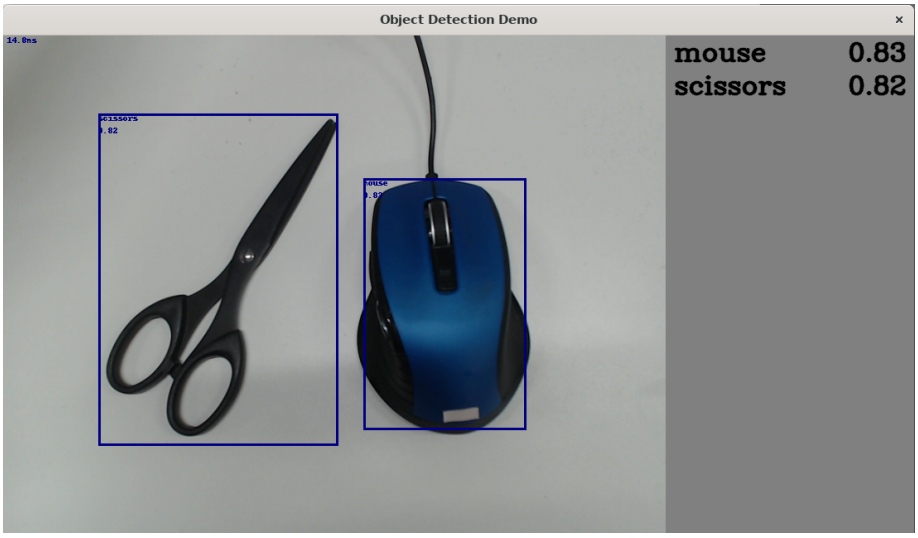
6.12.7. object detection demo

object detection demo はカメラからの映像に対して物体認識を行うアプリケーションです。NPU を使用しているため高速に物体認識を行えます。画面の左側には認識した物体を囲む四角形が表示され、右側には認識した物体のラベルとスコアが表示されます。このアプリケーションは機械学習のライブラリとして TensorFlow Lite を使用しています。

起動する前に、必要な Python ライブラリをインストールする必要があります。

```
[container /]# pip3 install pillow
```

図 6.137 pillow のインストールと起動



このアプリケーションはカメラデバイスとしてデフォルトで /dev/video2 を使用します。お使いの環境によって別のカメラデバイスに設定したい場合は、以下のファイルを変更してください。

```
[container /]# vi /usr/share/armadillo-demo-experience/resources/demos.json
:
:（省略）
:
```

```
{"Machine Learning":[{
        "Tensorflow Lite":[{
            "name": "object detection demo",
            "executable": "python3 /usr/share/armadillo-demo-experience/AI-demo/object_detection/
detect_usbcamera.py --model /usr/share/armadillo-demo-experience/AI-demo/object_detection/
detect.tflite --labels /usr/share/armadillo-demo-experience/AI-demo/object_detection/
coco_labels.txt --camera_id 2", ❶
            "compatible": "armadillo-x2",
          "description": "This is a simple object detection aplication that used NPU and TensorFlow
Lite on the Armadillo-X2 board."
        }]
    }]
```

図 6.138 ビデオデバイスの変更

❶ --camera_id の値を環境に合わせて変更します。

6.12.8. pose estimation demo

pose estimation demo はカメラに映った人物の姿勢を推定して表示するアプリケーションです。NPU を使用しているため高速に姿勢推定を行えます。推定した姿勢は人物の上に重ねて表示されます。このアプリケーションは機械学習のライブラリとして TensorFlow Lite を使用しています。

このアプリケーションは起動してから画面に映像が表示されるまで初回のみ約 1 分ほどかかります。2 回目以降の起動では 5 秒程度で映像が表示されます。また、カメラデバイスとしてデフォルトで /dev/video2 を使用します。お使いの環境によって別のカメラデバイスに設定したい場合は、以下のファイルを変更してください。

```
[container /]# vi /usr/share/armadillo-demo-experience/resources/demos.json
:
:（省略）
:
{"Machine Learning":[{
:
:（省略）
```

```
:
        {
            "name": "pose estimation",
            "executable": "python3 /usr/share/armadillo-demo-experience/AI-demo/pose_estimation/
pose_estimation.py --model /usr/share/armadillo-demo-experience/AI-demo/pose_estimation/
posenet.tflite --camera_id 2", ❶
            "source": "",
            "screenshot": "pose_estimation_demo.png",
            "compatible": "armadillo-x2",
          "description": "This is a simple pose estimation aplication that uses NPU on the Armadillo-
X2 board."
        }]
    }]
```

図 6.139 ビデオデバイスの変更

❶ --camera_id の値を環境に合わせて変更します。

6.12.9. image segmentation demo

image segmentation demo はカメラに映った人物の「人物として認識された領域(セグメント)」を推定して表示するアプリケーションです。NPU を使用しているため高速に領域推定を行えます。推定した領域は人物の上に青の透過色で重ねて表示されます。このアプリケーションは機械学習のライブラリとして TensorFlow Lite を使用しています。

このアプリケーションはカメラデバイスとしてデフォルトで /dev/video2 を使用します。お使いの環境によって別のカメラデバイスに設定したい場合は、以下のファイルを変更してください。

```
[container /]# vi /usr/share/armadillo-demo-experience/resources/demos.json
:
: (省略)
:
{"Machine Learning":[{
:
: (省略)
:
        {
            "name": "image segmentation",
           "executable": "python3 /usr/share/armadillo-demo-experience/AI-demo/image_segmentation/
```

```
image_segmentation.py --model /usr/share/armadillo-demo-experience/AI-demo/image_segmentation/
human_segmentation.tflite --camera_id 2", ❶
            "source": "",
            "screenshot": "image_segmentation_demo.png",
            "compatible": "armadillo-x2",
            "description": "This is a simple image segmentation aplication that uses NPU on the
Armadillo-X2 board."
        }]
    }]
```

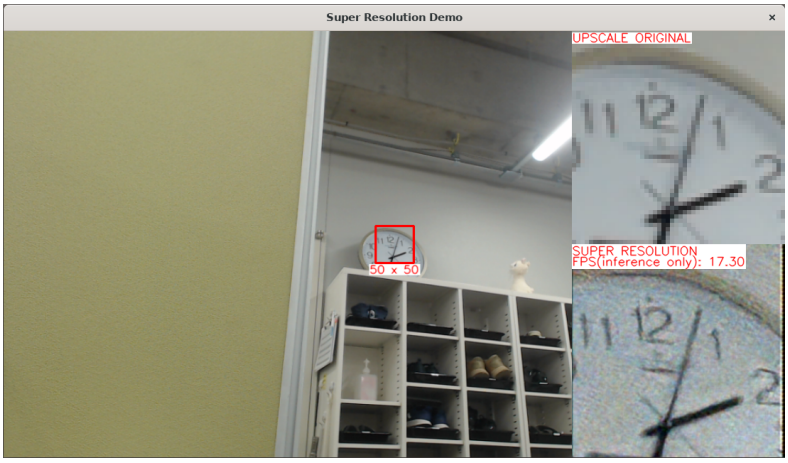
図 6.140 ビデオデバイスの変更

❶ --camera_id の値を環境に合わせて変更します。

6.12.10. super resolution demo

super resolution demo はカメラの映像の中央部 50 x 50 ピクセルの領域を 200 x 200 ピクセルに解像度を上げて表示する (超解像) アプリケーションです。NPU を使用しているため高速に超解像を行えます。このアプリケーションは機械学習のライブラリとして TensorFlow Lite を使用しています。

画面右上は最近傍補間で 200 x 200 ピクセルに拡大した映像、画面右下が超解像で 200 x 200 ピクセルにした映像です。



このアプリケーションは起動してから画面に映像が表示されるまで初回のみ約 1 分ほどかかります。2 回目以降の起動では 5 秒程度で映像が表示されます。また、カメラデバイスとしてデフォルトで /dev/video2 を使用します。お使いの環境によって別のカメラデバイスに設定したい場合は、以下のファイルを変更してください。

```
[container /]# vi /usr/share/armadillo-demo-experience/resources/demos.json
:
: (省略)
:
{"Machine Learning":[{
:
: (省略)
:
        {
            "name": "super resolution",
            "executable": "python3 /usr/share/armadillo-demo-experience/AI-demo/super_resolution/
```

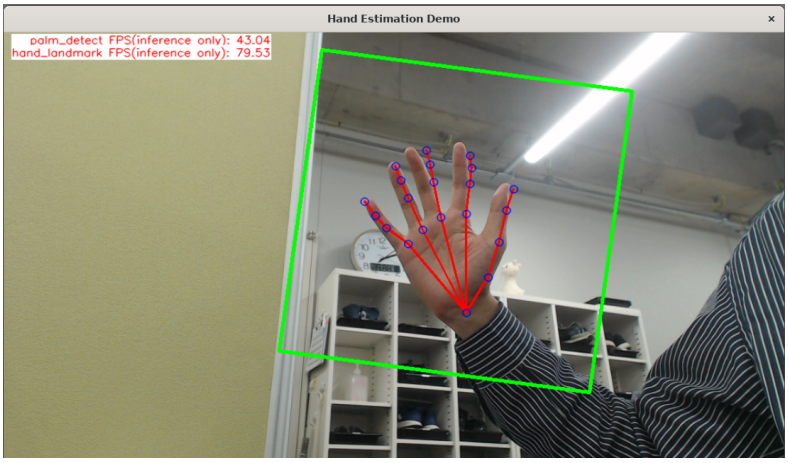
```
super_resolution.py --model /usr/share/armadillo-demo-experience/AI-demo/super_resolution/
super_resolution.tflite --camera_id 2", ❶
            "source": "",
            "screenshot": "super_resolution_demo.png",
            "compatible": "armadillo-x2",
            "description": "This is a super resolution application that uses NPU on the Armadillo-
X2 board."
        }]
    }]
```

図 6.141 ビデオデバイスの変更

❶ --camera_id の値を環境に合わせて変更します。

6.12.11. hand estimation demo

hand estimation demo はカメラに映った人物の手指を検出してその領域と手の骨格を同時に表示するアプリケーションです。NPU を使用しているため高速に手指検出を行えます。検出した手指の領域と骨格は手指の上に重ねて表示されます。このアプリケーションは機械学習のライブラリとして TensorFlow Lite を使用しています。



このアプリケーションは起動してから画面に映像が表示されるまで初回のみ約 30 秒ほどかかります。2 回目以降の起動では 5 秒程度で映像が表示されます。また、カメラデバイスとしてデフォルトで /dev/video2 を使用します。お使いの環境によって別のカメラデバイスに設定したい場合は、以下のファイルを変更してください。

```
[container /]# vi /usr/share/armadillo-demo-experience/resources/demos.json
:
: (省略)
:
{"Machine Learning":[{
:
: (省略)
:
        {
            "name": "hand estimation",
            "executable": "python3 /usr/share/armadillo-demo-experience/AI-demo/hand_estimation/
hand_estimation.py --landmark_model /usr/share/armadillo-demo-experience/AI-demo/hand_estimation/
hand.tflite --detection_model /usr/share/armadillo-demo-experience/AI-demo/hand_estimation/
```

```
palm.tflite --camera_id 2", ❶
            "source": "",
            "screenshot": "hand_estimation_demo.png",
            "compatible": "armadillo-x2",
            "description": "This is a simple hand estimation application that uses NPU on the
Armadillo-X2 board."
        }]
    }]
```

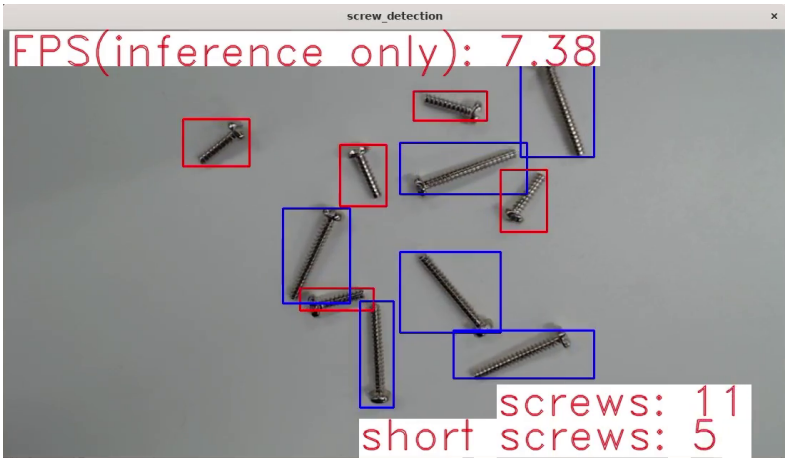
図 6.142 ビデオデバイスの変更

❶ --camera_id の値を環境に合わせて変更します。

6.12.12. screw detection demo

screw detection demo はカメラに映ったネジを検出してその領域を表示するアプリケーションです。また、領域の大きさからネジの長さを測り、しきい値以下であれば赤枠、それ以外は青枠で領域を囲います。各種パラメータはコマンドライン引数で指定可能です。

NPU を使用しているため高速にネジの検出を行えます。検出したネジの領域はネジの上に重ねて表示されます。このアプリケーションは機械学習のライブラリとして TensorFlow Lite を使用しています。



このアプリケーションは起動してから画面に映像が表示されるまで初回のみ約 30 秒ほどかかります。2 回目以降の起動では 5 秒程度で映像が表示されます。また、カメラデバイスとしてデフォルトで /dev/video2 を使用します。お使いの環境によって別のカメラデバイスに設定したい場合は、以下のファイルを変更してください。

```
[container /]# vi /usr/share/armadillo-demo-experience/resources/demos.json
:
: (省略)
:
{"Machine Learning":[{
:
: (省略)
:
        {
            "name": "screw detection",
            "executable": "python3 /usr/share/armadillo-demo-experience/AI-demo/screw_detection/
screw_detection.py --camera_id 2 --conf_thres 0.5 --iou_thres 0.45 --len_thres 130", ❶
```

```
            "source": "",
            "screenshot": "screw_detection_demo.png",
            "compatible": "armadillo-x2",
            "description": "This is a simple screw detection application that uses NPU on the
Armadillo-X2 board."
        }]
    }]
```

図 6.143 各種コンフィグの変更

❶ この行の各パラメータを変更することで、アプリケーションの挙動が変化します。各パラメータの詳細は「表 6.13. ネジ検出デモのパラメータの詳細」を参照してください。

表 6.13 ネジ検出デモのパラメータの詳細

パラメータ名	意味
camera_id	使用するカメラを指定します。/dev/videoN の N に相当します。
conf_thres	AI がネジと認識した確度のしきい値です。この値以下の確度の物体はネジとみなされません。
iou_thres	ネジ検出の後処理に使用するパラメータです。
len_thres	長いネジと短いネジの境界値を設定できます。
visualize_score	これを指定すると、AI がネジと認識した確度を描画します。
visualize_length	これを指定すると、ネジの長さを描画します。

6.13. SMS を利用する

Armadillo-IoT ゲートウェイ G4 は、3G/LTE モジュール を使用した SMS の送受信を行うことができます。SMS の送信、受信した SMS の確認および削除などの操作は ModemManager の mmcli コマンドで行うことができます。

本章では mmcli コマンドでの SMS の使用方法について説明します。

6.13.1. 初期設定

SMS が利用可能な SIM を挿入して Armadillo-IoT ゲートウェイ G4 の電源を入れると、ModemManager が必要な初期設定を行い、 SMS が利用可能になります。

SMS の受信は自動的に行われます。

「図 6.144. 言語設定」に示すコマンドを実行し、言語設定を行います。

```
[armadillo ~]# export LANG="ja_JP.UTF-8"
```

図 6.144 言語設定

6.13.2. SMS を送信する

SMS を作成するには、「図 6.145. SMS の作成」に示すコマンドを実行します。

```
[armadillo ~]# mmcli -m 0 --messaging-create-sms="number=[送信先電話番号],text='[SMS 本文]'"
```

図 6.145 SMS の作成

SMS の作成に成功すると、以下のように SMS 番号が表示されます。SMS 番号は送信時に使用します。

```
Successfully created new SMS: /org/freedesktop/ModemManager1/SMS/[SMS 番号]
```

図 6.146 SMS 番号の確認

「図 6.147. SMS の送信」に示すコマンドを実行し、SMS 送信を行います。 [SMS 番号] には、 SMS の作成時に表示された番号を指定します。

```
[armadillo ~]# mmcli -s [SMS 番号] --send
```

図 6.147 SMS の送信

6.13.3. SMS を受信する

SMS を送信可能な端末から Armadillo-IoT ゲートウェイ G4 に SMS を送信すると、 Armadillo-IoT ゲートウェイ G4 は自動的に SMS を受信します。

また、 EC25-J の内蔵ストレージに 255 件 SMS を保存した状態で Armadillo-IoT ゲートウェイ G4 に SMS を送信した場合は、Armadillo-IoT ゲートウェイ G4 は受信を行いません。

受信を行うには、 EC25-J の内蔵ストレージに保存している SMS を削除するか、他のストレージに移動する必要があります。

6.13.4. SMS 一覧を表示する

「図 6.148. SMS の一覧表示」のコマンドを実行することで、 SMS 一覧を表示できます。

末尾が "(sent)" となっているものが送信した SMS で "(received)" となっているものが受信した SMS です。

```
[armadillo ~]# mmcli -m 0 --messaging-list-sms
Found 7 SMS messages:
        /org/freedesktop/ModemManager1/SMS/0 (received)
        /org/freedesktop/ModemManager1/SMS/1 (received)
        /org/freedesktop/ModemManager1/SMS/2 (received)
        /org/freedesktop/ModemManager1/SMS/3 (received)
        /org/freedesktop/ModemManager1/SMS/4 (sent)
        /org/freedesktop/ModemManager1/SMS/5 (received)
        /org/freedesktop/ModemManager1/SMS/6 (sent)
```

図 6.148 SMS の一覧表示

6.13.5. SMS の内容を表示する

SMS の内容を表示するには、「図 6.149. SMS の内容を表示」に示すコマンドを実行します。

```
[armadillo ~]# mmcli -s [SMS 番号]
  ----------------------------------
  Content    |              number: XXXXXXXXXXX
```

```
            |                text: hello world
  ----------------------------------
  Properties |            PDU type: deliver
             |               state: received
             |             storage: me
             |                smsc: +XXXXXXXXXXX
             |           timestamp: XXXXXXXXXXX+XX
```

図 6.149 SMS の内容を表示

受信した SMS は自動的に 3G/LTE モジュールの内蔵ストレージに保存されます。Armadillo-IoT ゲートウェイ G4 に搭載されている、 EC25-J は、最大 255 件まで SMS を保存することが可能です。

SMS の内容を表示した際の「storage: **me**」は、 3G/LTE モ ジュールの内蔵ストレージに SMS が保存されていることを意味しています。

「storage: **sm**」と表示された場合、 SIM カードのストレージに SMS が保存されています。 SIM カードのストレージに保存できる SMS の件数は SIM カードによって異なります。

ストレージに保存されている SMS は、Armadillo-IoT ゲートウェイ G4 の電源を切断してもデータが保持されます。

6.13.6. SMS を削除する

SMS を削除するには、「図 6.150. SMS の削除」に示すコマンドを実行します。

```
[armadillo ~]# mmcli -m 0 --messaging-delete-sms=[SMS 番号]
```

図 6.150 SMS の削除

6.13.7. SMS を他のストレージに移動する

SIM カードのストレージに SMS を移動するには、「図 6.151. SIM カードのストレージに SMS を移動」に示すコマンドを実行します。

```
[armadillo ~]# mmcli -s [SMS 番号] --store-in-storage="sm"
```

図 6.151 SIM カードのストレージに SMS を移動

3G/LTE モジュールの内蔵ストレージに SMS を移動するには、「図 6.152. 3G/LTE モジュールの内蔵ストレージに SMS を移動」に示すコマンドを実行します。

```
[armadillo ~]# mmcli -s [SMS 番号] --store-in-storage="me"
```

図 6.152 3G/LTE モジュールの内蔵ストレージに SMS を移動

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

Armadillo-IoT ゲートウェイ G4 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。

6.15. 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-IoT ゲートウェイ G4 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。

6.15.1. 接続可能なネットワーク

Armadillo-IoT ゲートウェイ G4 は、2 つの Ethernet ポートが搭載されています。Linux からは、それぞれ eth0 、 eth1 に見えます。

表 6.14 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
Ethernet	eth1	DHCP
無線 LAN	mlan0	クライアントモード
3G/LTE	ttyCommModem	SIM / 料金プランに依存します

eth1(LAN インターフェース 2)は 10Mbps(10BASE-T)に非対応です。10Mbps で通信を行う場合は、LAN インターフェース 0(LAN インターフェース 1)をご利用ください。

6.15.2. IP アドレスの確認方法

Armadillo-IoT ゲートウェイ G4 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 4a:1e:d4:a4:ab:ee brd ff:ff:ff:ff:ff:ff
    inet 172.16.1.20/16 brd 172.16.255.255 scope global dynamic noprefixroute eth0
       valid_lft 14810sec preferred_lft 14810sec
```

```
    inet6 fe80::14d8:e8aa:fada:1ed5/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: eth1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN qlen 1000
    link/ether 4a:71:b8:5c:66:4e brd ff:ff:ff:ff:ff:ff
```

図 6.153 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 4a:1e:d4:a4:ab:ee brd ff:ff:ff:ff:ff:ff
    inet 172.16.1.20/16 brd 172.16.255.255 scope global dynamic noprefixroute eth0
       valid_lft 14517sec preferred_lft 14517sec
    inet6 fe80::14d8:e8aa:fada:1ed5/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

図 6.154 IP アドレス(eth0)の確認

6.15.3. ネットワークの設定方法

Armadillo-IoT ゲートウェイ G4 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。

6.15.3.1. 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 6.155. 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 6.155 nmcli のコマンド書式

6.15.4. 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。

6.15.4.1. 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。[2]

```
[armadillo ~]# nmcli connection
NAME                UUID                                  TYPE      DEVICE
Wired connection 1  a6f99120-b4ed-3823-a6f0-0491d4b6101e  ethernet  eth0
Wired connection 2  fafc0e33-bb47-3c39-b9a1-395e4ce6cb6e  ethernet  --
```

図 6.156 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。

6.15.4.2. 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 6.157 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 6.158 コネクションの無効化

6.15.4.3. コネクションの作成

コネクションを作成するには、次のようにコマンドを実行します。

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

図 6.159 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-IoT ゲートウェイ G4 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 6.160 コネクションファイルの永続化

[2] 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。

別の Armadillo-IoT ゲートウェイ G4 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 「3.2.3. アップデート機能について」 を参考にしてください。

6.15.4.4. 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 6.161 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 6.162 コネクションファイル削除時の永続化

6.15.4.5. 固定 IP アドレスに設定する

「表 6.15. 固定 IP アドレス設定例」の内容に設定する例を、「図 6.163. 固定 IP アドレス設定」に示します。

表 6.15 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 6.163 固定 IP アドレス設定

6.15.4.6. DNS サーバーを指定する

DNS サーバーを指定する例を、「図 6.164. 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 6.164 DNS サーバーの指定

6.15.4.7. DHCP に設定する

DHCP に設定する例を、「図 6.165. DHCP の設定」に示します。

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

図 6.165 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。

6.15.4.8. 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 6.166 コネクションの修正の反映

6.15.4.9. 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
```

```
eth1    ethernet  unavailable  --
lo      loopback  unmanaged    --
```

図 6.167 デバイスの一覧

6.15.4.10. 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 6.168 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。

6.15.4.11. 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 6.169 デバイスの切断

6.15.5. 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 6.170 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。eth1 を確認する場合は -I eth1 としてください。

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。

6.15.6. 無線 LAN

Armadillo-IoT ゲートウェイ G4 WLAN モデルに搭載されている WLAN + BT コンボモジュールで正常に通信が可能か確認します。

例として、WPA2-PSK(AES) のアクセスポイントに接続します。ここでの説明では、アクセスポイントの ESSID を [essid]、パスフレーズを [passphrase] と表記します。

6.15.6.1. 無線 LAN アクセスポイントに接続する

1. 無線 LAN アクセスポイントに接続する

```
[armadillo ~]# nmcli device wifi connect [essid] password [passphrase]
```

Armadillo-IoT ゲートウェイ G4 を再起動したときに 無線 LAN のコネクションファイルが消えてしまわないように、「図 6.160. コネクションファイルの永続化」 を参照してファイルの永続化を行ってください。

6.15.6.2. 無線 LAN の通信を確認する

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 mlan0 -c 3 192.0.2.20
PING 192.0.2.20 (192.0.2.20: 56 data bytes
64 bytes from 192.0.2.20: seq=0 ttl=64 time=51.951 ms
64 bytes from 192.0.2.20: seq=1 ttl=64 time=19.347 ms
64 bytes from 192.0.2.20: seq=2 ttl=64 time=13.861 ms

--- 192.0.2.20 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 13.861/28.386/51.951 ms
```

図 6.171 無線 LAN の PING 確認

有線 LAN と 無線 LAN の両方が有効になっている場合、常に有線 LAN が優先されます。確実に無線 LAN の接続確認を行う場合は、無線 LAN 以外のインターフェースを無効化するか、以下の手順で無線 LAN の優先度を他のインターフェースより高くしてください。

ネットワークインターフェースの優先度を確認します。

```
[armadillo ~]# ip route
default via 172.16.0.1 dev eth0  src 172.16.1.43  metric 100
default via 172.16.0.1 dev mlan0  src 172.16.1.52  metric 600
172.16.0.0/16 dev eth0 scope link  src 172.16.1.43  metric 100
172.16.0.0/16 dev mlan0 scope link  src 172.16.1.52  metric 600
```

図 6.172 ネットワークインターフェースの優先度確認

Metric の値が低ければ優先度は高くなるので、無線 LAN の Metric の値 600 を有線 LAN の 100 よりも低く設定します。以下の例では、 50 に設定します。

ここの例で使用している無線 LAN コネクションの [ID] に関しては、 「6.15.4.1. コネクションの一覧」の手順で確認してください。

```
[armadillo ~]# nmcli connection modify [ID] ipv4.route-metric 50
[armadillo ~]# nmcli connection up [ID]
[armadillo ~]# ip route
default via 172.16.0.1 dev mlan0  src 172.16.1.52  metric 50 ❶
default via 172.16.0.1 dev eth0  src 172.16.1.43  metric 100
172.16.0.0/16 dev mlan0 scope link  src 172.16.1.52  metric 50
172.16.0.0/16 dev eth0 scope link  src 172.16.1.43  metric 100
```

図 6.173 ネットワークインターフェースの優先度設定

❶ Metric が 50 になっていることが確認できます。

これにより、有線 LAN と無線 LAN が両方有効な場合でも無線 LAN が優先して使われるようになります。

6.15.7. 3G/LTE

本章では、Armadillo-IoT ゲートウェイ G4 に搭載されている 3G/LTE モジュールの使用方法について説明します。

Quectel 製 3G/LTE 通信モジュール EC25-J はドコモ/KDDI/ソフトバンクそれぞれの相互接続性試験を完了しています。

6.15.7.1. 3G/LTE データ通信設定を行う前に

3G/LTE データ通信を利用するには、通信事業者との契約が必要です。契約時に通信事業者から貸与された nanoSIM(UIM カード)と APN 情報を準備します。

Quectel 製 EC25-J 搭載モデルでの動作検証済み nanoSIM (料金プラン) に関しては、 Armadillo サイトの「Armadillo-IoT ゲートウェイ 動作確認済み SIM 一覧」を確認ください。

Armadillo-IoT ゲートウェイ 動作確認済み SIM 一覧 [https://armadillo.atmark-techno.com/howto/armadillo-iot-tested-sim]

Armadillo-IoT ゲートウェイ G4 の電源が切断されていることを確認してから nanoSIM(UIM カード)を取り付けてください。

本製品は、nanoSIM スロットを搭載しています。

標準/microSIM サイズの SIM カードを nanoSIM サイズにカットしたもの、サイズの異なるものを使用すると、nanoSIM スロットが故障する原因となります。これらを使用し本製品が故障した場合は、保証期間内であっても保証適用外となります。

nanoSIM(UIM カード)の切り欠きを挿入方向に向け、刻印面を上にして挿入してください。挿入位置などは、「図 3.17. 3G/LTE 用アンテナおよび nanoSIM カード接続例」 を参照してください。

APN の設定を行うには、次に示す情報が必要です。

- APN (最大 39 文字)
- ユーザー名 (最大 127 文字)
- パスワード (最大 127 文字)
- 認証方式 (PAP または CHAP)
- PDP Type (IP のみをサポート)

6.15.7.2. 3G/LTE のコネクションを作成する

「表 6.16. APN 情報設定例」の内容に設定する例を「図 6.174. 3G/LTE のコネクションの作成」に示します。

表 6.16 APN 情報設定例

項目	設定
APN	[apn]
ユーザー名	[user]
パスワード	[password]
ネットワークデバイス	[wwan]

ネットワークデバイス [wwan] は、「表 6.17. 通信モジュールのネットワークデバイス」 を参照ください。

表 6.17 通信モジュールのネットワークデバイス

通信モジュール	ネットワークデバイス
Quectel 製 EC25-J	ttyCommModem

```
[armadillo ~]# nmcli connection add type gsm ifname [wwan] apn [apn] user [user] password [password]
Connection 'gsm-[wwan]' (xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx) successfully added.
```

図 6.174 3G/LTE のコネクションの作成

コネクション設定を永続化するには、以下のコマンドを入力してください。設定を永続化すると、Armadillo 起動時に自動的にデータ接続を行うようになります。

同一インタフェースへの設定が複数存在する場合、 gsm-[wwan]-1.nmconnection など後ろに数値が付与されますので、「図 6.174. 3G/LTE のコネクションの作成」 入力時のメッセージで生成されたファイル名を確認した上で永続化を実施ください。

```
[armadillo ~]# persist_file /etc/NetworkManager/system-connections/gsm-[wwan].nmconnection
```

図 6.175 3G/LTE のコネクションの設定の永続化

Armadillo Base OS 3.17.3-at.6 以降、 KDDI 網を使用したデータ通信専用 SIM をご利用で LTE 接続ができない場合、 LTE モジュールの IMS を無効に設定することで接続できる可能性があります。

「図 6.176. LTE モジュールの IMS 無効化コマンド」に示すコマンドを一度入力してください。一度入力すれば LTE モジュールに設定が反映されますので、 Armadillo を起動する度に設定する必要はありません。

```
[armadillo ~]# ec25-ims-disable
```

図 6.176 LTE モジュールの IMS 無効化コマンド

6.15.7.3. MCC/MNC を指定した 3G/LTE のコネクションを作成する

マルチキャリア SIM などを使用する際、MCC (Mobile Country Code) と MNC (Mobile Network Code) を指定してコネクションを作成すると 3G/LTE ネットワークに接続できることがあります。指定する場合は 「図 6.177. MCC/MNC を指定した 3G/LTE コネクションの作成」 に示すコマンドを実行してください。

[mccmnc] には 44010 などの数字を入力してください。実際に設定する値に関しては、ご契約の通信事業者へお問い合わせください。

```
[armadillo ~]# nmcli connection add type gsm ifname [wwan] apn [apn] user [user] password [password] ⏎
gsm.network-id [mccmnc]
```

図 6.177 MCC/MNC を指定した 3G/LTE コネクションの作成

6.15.7.4. PAP 認証を有効にした 3G/LTE のコネクションを作成する

3G/LTE のコネクションの認証方式は、デフォルトで CHAP に設定されています。PAP 認証を有効にしたコネクションを作成する場合は「図 6.178. PAP 認証を有効にした 3G/LTE コネクションの作成」に示すコマンドを実行してください。

```
[armadillo ~]# nmcli connection add type gsm ifname [wwan] apn [apn] user [user] password [password]
ppp.refuse-eap true ppp.refuse-chap true ppp.refuse-mschap true ppp.refuse-mschapv2 true
ppp.refuse-pap false
```

図 6.178 PAP 認証を有効にした 3G/LTE コネクションの作成

すでに 3G/LTE コネクションを作成済みの場合はコネクション設定を削除し、「図 6.179. 3G/LTE モジュール初期化手順」に示すモジュール初期化を実施した後に、「図 6.178. PAP 認証を有効にした 3G/LTE コネクションの作成」を実施してください。

6.15.7.5. 3G/LTE モジュールの初期化

SIM カードを入れ替えた際、または 「6.15.7.4. PAP 認証を有効にした 3G/LTE のコネクションを作成する」 を実施する際には、「図 6.179. 3G/LTE モジュール初期化手順」に示すモジュール初期化を実施してください。

```
[armadillo ~]# rc-service connection-recover stop ❶
connection-recover| * Stopping connection-recover ... [ ok ]
[armadillo ~]# rc-service modemmanager stop ❷
modemmanager      | * WARNING: you are stopping a boot service
modemmanager      | * Stopping modemmanager ... [ ok ]
[armadillo ~]# ec25-clean-reset ❸
OK
OK
usb 3-1.2: USB disconnect, device number 3
option1 ttyUSB0: GSM modem (1-port) converter now disconnected from ttyUSB0
option 3-1.2:1.0: device disconnected
option1 ttyUSB1: GSM modem (1-port) converter now disconnected from ttyUSB1
option 3-1.2:1.1: device disconnected
option1 ttyUSB2: GSM modem (1-port) converter now disconnected from ttyUSB2
option 3-1.2:1.2: device disconnected
option1 ttyUSB3: GSM modem (1-port) converter now disconnected from ttyUSB3
option 3-1.2:1.3: device disconnected
usb 3-1.2: new high-speed USB device number 4 using xhci-hcd
option 3-1.2:1.0: GSM modem (1-port) converter detected
usb 3-1.2: GSM modem (1-port) converter now attached to ttyUSB0
option 3-1.2:1.1: GSM modem (1-port) converter detected
usb 3-1.2: GSM modem (1-port) converter now attached to ttyUSB1
option 3-1.2:1.2: GSM modem (1-port) converter detected
usb 3-1.2: GSM modem (1-port) converter now attached to ttyUSB2
option 3-1.2:1.3: GSM modem (1-port) converter detected
usb 3-1.2: GSM modem (1-port) converter now attached to ttyUSB3
```

```
[armadillo ~]# rc-service connection-recover start ❹
connection-recover| * Starting connection-recover ... [ ok ]
```

図 6.179 3G/LTE モジュール初期化手順

❶ 再接続サービスを停止します。

❷ ModemManager を停止します。

❸ モジュール初期化処理を実行します。

❹ コンソールに上記ログが表示され、モジュールの再起動を確認後に、再接続サービスを起動します。

停止した ModemManager は 3G/LTE モジュール認識時自動的に再起動しますので、手動での起動は不要です。

6.15.7.6. 3G/LTE コネクションを確立する

3G/LTE コネクションの作成直後や設定変更後に再起動をせずにコネクションを確立するには、「図 6.180. 3G/LTE のコネクション確立」に示すコマンドを実行します。

```
[armadillo ~]# nmcli connection up gsm-[wwan]
Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkManager/
ActiveConnection/x)
```

図 6.180 3G/LTE のコネクション確立

6.15.7.7. 3G/LTE の接続を確認する

ご利用になるサービスとの通信を検証する、ICMP に対応しているアドレス (8.8.8.8 など) と PING 通信を行うなどの方法で 3G/LTE の接続を確認してください。

```
[armadillo ~]# ping -c 3 8.8.8.8 -I ppp0
```

図 6.181 3G/LTE の PING 確認

6.15.7.8. 3G/LTE コネクションを切断する

3G/LTE コネクションを切断するには、「図 6.182. 3G/LTE コネクションを切断する」に示すコマンドを実行します。3G/LTE コネクションを切断する前に、3G/LTE 再接続サービスを停止しないと再接続処理が実行される為、事前に停止します。

```
[armadillo ~]# rc-service connection-recover stop ❶
connection-recover| * Stopping connection-recover ... [ ok ]
[armadillo ~]# nmcli connection down gsm-[wwan] ❷
```

図 6.182 3G/LTE コネクションを切断する

❶ 3G/LTE 再接続サービスを停止します。

❷ 3G/LTE コネクションを切断します。

6.15.7.9. 3G/LTE 再接続サービス

3G/LTE 再接続サービスは、3G/LTE のデータ接続の状態を定期的に監視し、切断を検出した場合に再接続を行うサービスです。

SIM カードが挿入されており、NetworkManager に有効な 3G/LTE コネクションの設定がされているとき、初期設定では 120 秒に一度コネクションの状態を監視します。オプションで SIM カードの認識ができないときに Armadillo の再起動を実施することも可能です。

コネクションが無効になっている場合、切断状態と判定しコネクションを有効にします。

コネクションが有効になっている場合、特定の宛先に PING を実行します。PING がエラーになったとき切断状態と判定し、コネクションの無効化・有効化を行うことで再接続を実施します。

コネクションの無効化・有効化による再接続を実施しても PING がエラーになる場合、電波のオン・オフまたは 3G/LTE モジュールの電源をオン・オフを実施して再接続を実施します。どちらを実施するかは設定ファイルの WWAN_FORCE_RESTART_COUNT に依存します。

WWAN_FORCE_RESTART_COUNT が初期値の 10 である場合、1 から 9 回目は電波のオン・オフを実施し、10 回目は 3G/LTE モジュールの電源オン・オフを実施します。それ以降も NG が続く場合、同じく 10 回に一度 3G/LTE モジュールの電源オン・オフを実施します。

工場出荷状態で本サービスは有効化されており、システム起動時にサービスが自動的に開始されます。PING を実行する宛先は、初期設定では "8.8.8.8" です。

atmark-wwan-utils のバージョンが 1.5.0-r0 以降の場合は、設定ファイルは /etc/atmark/connection-recover.conf を使用します。

設定ファイルの記載例として、サンプルファイル(/etc/atmark/connection-recover.conf.example)がありますので、こちらをリネームまたはコピーしてご利用ください。

atmark-wwan-utils 1.5.0-r0 以降、旧設定ファイル /etc/atmark/connection-recover/gsm-ttyUSB2_connection-recover.conf が存在する場合、/etc/atmark/connection-recover.conf よりも優先して設定ファイルとして使用します。

旧設定ファイルが不要である場合は、「図 6.183. 再接続サービス 旧設定ファイルの削除」に示すとおりに削除してご利用ください。

```
[armadillo ~]# persist_file -d /etc/atmark/connection-recover/gsm-
ttyUSB2_connection-recover.conf
```

図 6.183 再接続サービス 旧設定ファイルの削除

設定ファイルの概要を「表 6.18. 再接続サービス設定パラメーター」に示します。必要に応じて設定値を変更してください。

設定ファイルが存在しない場合は初期値で動作します。

表 6.18 再接続サービス設定パラメーター

パラメーター名	初期値	意味	変更
PRODUCT_NAME	-	製品名	不可
CHECK_INTERVAL_SEC	120	監視周期(秒)	可
PING_DEST_IP	8.8.8.8	コネクション状態確認時 PING 送付先	可
DEVICE	-	ネットワークデバイス名	不可
TYPE	-	ネットワークタイプ	不可
NETWORK_IF	-	ネットワーク I/F 名	不可
FORCE_REBOOT	FALSE	TRUE に設定すると PING 導通チェック NG 時 Armadillo を再起動します。	可
REBOOT_IF_SIM_NOT_FOUND	FALSE	TRUE に設定すると SIM を検出できない時に Armadillo を再起動します。	可
WWAN_FORCE_RESTART_COUNT	10	PING 導通確認を設定した回数連続で失敗した場合モデムの再起動を実行します。設定した回数に満たない場合 、電波のオフ・オン実施のみで 3G/LTE 再接続を試みます。	可

設定ファイル変更後、変更内容を永続化するには「図 6.184. 3G/LTE 再接続サービスの設定値を永続化する」に示すコマンドを実行してください。

```
[armadillo ~]# persist_file /etc/atmark/connection-recover.conf
```

図 6.184 3G/LTE 再接続サービスの設定値を永続化する

3G/LTE 再接続サービスの状態を確認するには、「図 6.185. 3G/LTE 再接続サービスの状態を確認する」に示すコマンドを実行してください。

```
[armadillo ~]# rc-status | grep connection-recover
 connection-recover                                [  started 00:43:02 (0) ]
```

図 6.185 3G/LTE 再接続サービスの状態を確認する

3G/LTE 再接続サービスを停止するには、「図 6.186. 3G/LTE 再接続サービスを停止する」に示すコマンドを実行してください。

```
[armadillo ~]# rc-service connection-recover stop
connection-recover| * Stopping connection-recover ... [ ok ]
```

図 6.186 3G/LTE 再接続サービスを停止する

3G/LTE 再接続サービスを開始するには、「図 6.187. 3G/LTE 再接続サービスを開始する」に示すコマンドを実行してください。

```
[armadillo ~]# rc-service connection-recover start
connection-recover| * Starting connection-recover ... [ ok ]
```

図 6.187 3G/LTE 再接続サービスを開始する

独自に接続状態を確認するサービスを実装されるなどの理由で標準の 3G/LTE 再接続サービスが不要な場合、「図 6.188. 3G/LTE 再接続サービスを無効にする」に示す手順で再接続サービスを永続的に無効にできます。

```
[armadillo ~]# rc-service connection-recover stop ❶
connection-recover| * Stopping connection-recover ... [ ok ]
[armadillo ~]# rc-update del connection-recover default ❷
service connection-recover removed from runlevel default
[armadillo ~]# persist_file -rv /etc/runlevels/default/connection-recover ❸
```

図 6.188 3G/LTE 再接続サービスを無効にする

❶ 再接続サービスを停止します。

❷ 再接続サービスを無効にします。

❸ サービスの設定ファイルを削除を永続化します。

3G/LTE 再接続サービスを無効化した後、再度有効にする場合、「図 6.189. 3G/LTE 再接続サービスを有効にする」に示す手順を実行してください。

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

図 6.189 3G/LTE 再接続サービスを有効にする

❶ 再接続サービスを有効にします。

❷ 再接続サービスを開始します。

❸ サービスの設定ファイルを永続化します。

6.15.7.10. ModemManager - mmcli について

ここでは ModemManager と mmcli について説明します。

Armadillo-IoT ゲートウェイ G4 にはネットワークを管理する NetworkManager とは別に、モデムを管理する ModemManager がインストールされています。ModemManager はモバイルブロードバンドデバイス(3G/LTE モジュールなど)の操作および、接続状況の管理などを行います。

ModemManager のコマンドラインツールである **mmcli** を使用することで、3G/LTE 通信の電波強度や SIM カードの情報(電話番号や IMEI など)を取得することが可能です。mmcli の詳しい使いかたについては **man mmcli** を参照してください。

6.15.7.11. mmcli - 認識されているモデムの一覧を取得する

認識されているモデムの一覧を取得するには、「図 6.190. 認識されているモデムの一覧を取得する」に示すコマンドを実行します。

```
[armadillo:~]# mmcli -L
    /org/freedesktop/ModemManager1/Modem/0 [Quectel] EC25
```

図 6.190 認識されているモデムの一覧を取得する

6.15.7.12. mmcli - モデムの情報を取得する

モデムの情報を取得するには、「図 6.191. モデムの情報を取得する」に示すコマンドを実行します。

```
armadillo:~# mmcli -m 0
  --------------------------------
  General  |                 path: /org/freedesktop/ModemManager[number1]/Modem/[number2]
           |           device id: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  --------------------------------
  Hardware |         manufacturer: Quectel
           |                model: EC25
           |    firmware revision: XXXXXXXXXXXXXXXXXXX
           |            supported: gsm-umts, lte
           |              current: gsm-umts, lte
           |         equipment id: XXXXXXXXXXXXXXX
: (省略)
```

図 6.191 モデムの情報を取得する

モデムの情報を取得するには、SIM カードが取り付けられている必要があります。正しく SIM カードが取り付けられていることを確認してください。

6.15.7.13. mmcli - SIM の情報を取得する

SIM の情報を取得するには、「図 6.192. SIM の情報を取得する」に示すコマンドを実行します。

```
[armadillo ~]# mmcli -m 0
: (省略)
SIM      |     primary sim path: /org/freedesktop/ModemManager1/SIM/[number] # [number] を次のコマ ⏎
ンドで使用
: (省略)

[armadillo ~]# mmcli -i [number]
  --------------------------------
  General    |              path: /org/freedesktop/ModemManager1/SIM/0
  --------------------------------
  Properties |            active: yes
             |              imsi: XXXXXXXXXXXXXXX
             |             iccid: XXXXXXXXXXXXXXXXXXX
             |       operator id: XXXXX
```

```
              |     operator name: XXXXXXXXXX
              | emergency numbers: XXXX
```

図 6.192 SIM の情報を取得する

6.15.7.14. mmcli - 回線情報を取得する

回線情報を取得するには、「図 6.193. 回線情報を取得する」に示すコマンドを実行します。

```
[armadillo ~]# mmcli -m 0
:（省略）
  Bearer   |               paths: /org/freedesktop/ModemManager1/Bearer/[number] # [number] を次の ⏎
コマンドで使用
:（省略）

[armadillo ~]# mmcli -b [number]
  ------------------------------------
  General             |           path: /org/freedesktop/ModemManager1/Bearer/1
                      |           type: default
  ------------------------------------
  Status              |      connected: yes
                      |      suspended: XX
                      |    multiplexed: XX
                      |      interface: [wwan]
                      |     ip timeout: XX
                      |     profile id: X
  ------------------------------------
  Properties          |            apn: XXXXXXXXXX
                      |        roaming: allowed
                      |        ip type: ipv4v6
                      |           user: XXXXX
                      |       password: XXXXX
  ------------------------------------
  IPv4 configuration  |         method: ppp
                      |         prefix: X
  ------------------------------------
  IPv6 configuration  |         method: ppp
                      |         prefix: X
  ------------------------------------
  Statistics          |       duration: XXX
                      |       attempts: X
                      | total-duration: XXX
```

図 6.193 回線情報を取得する

6.16. コマンドラインから無線 LAN でネットワークを構成する

この章では、Armadillo-IoT ゲートウェイ G4 WLAN モデルに搭載されている WLAN + BT コンボモジュールを使って様々なネットーワークを構成する例を紹介します。

なお、 ABOS Web を利用することで簡単に同様の設定を行うことが可能ですので、基本的には ABOS Web の使用を推奨します。詳細は、「3.8. ネットワーク設定」を参照してください。

6.16.1. 無線 LAN アクセスポイント (AP) に接続する

Armadillo-IoT ゲートウェイ G4 WLAN モデルを子機として、無線 LAN AP に接続する方法を説明します。

以下は、WPA2-PSK(AES) の AP に接続する例です。ここでは、AP の ESSID を [essid]、パスフレーズを [passphrase] と表記します。

```
[armadillo ~]# nmcli device wifi connect [essid] password [passphrase]
[armadillo ~]# persist_file /etc/NetworkManager/system-connections/[essid].nmconnection ❶
```

図 6.194 無線 LAN AP に接続する

❶ 設定ファイルを永続化することで、Armadillo 起動時に自動的に AP に接続するようになります。

・ nmtui を使用して設定する

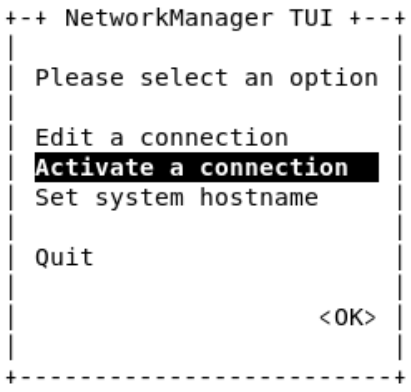
nmtui というツールを使用するとテキストユーザーインターフェースで設定を行うことができます。

```
[armadillo ~]# nmtui
```

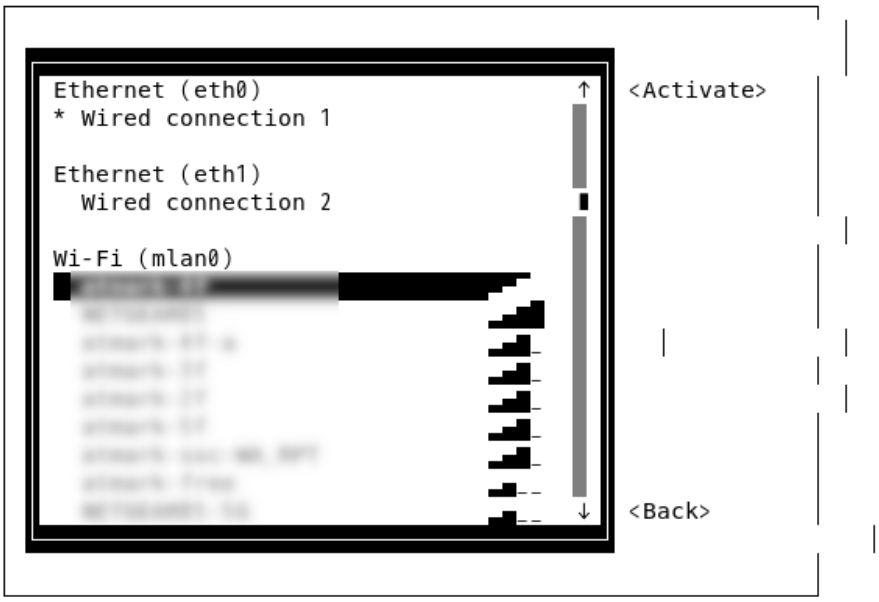
図 6.195 nmtui を起動する

表示された画面上で設定が行なえます。

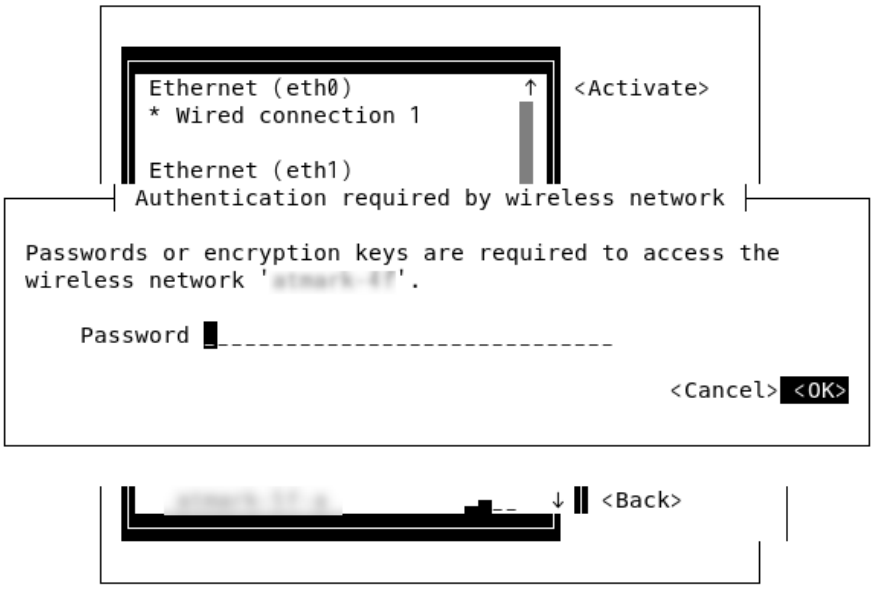
Activate a connection を選択します。



Wi-Fi (mlan0) の下に表示される AP 一覧の中から、接続したいものを選択します。



表示された画面で、接続のためのパスフレーズを入力します。



入力後に OK を押すと接続されます。

6.16.2. 無線 LAN アクセスポイント (AP) として設定する

Armadillo-IoT ゲートウェイ G4 WLAN モデルを無線 LAN AP として設定する方法を説明します。AP を設定するには hostapd というソフトウェアと、DNS/DHCP サーバである dnsmasq というソフトウェアを使用します。

hostapd と dnsmasq は Armadillo Base OS にデフォルトでインストール済みとなっているため、インストール作業は不要です。インストールされていない場合は、 Armadillo Base OS を最新バージョンに更新してください。

アクセスポイントモード (AP) と ステーションモード (STA) の同時利用はできません。Armadillo-IoT ゲートウェイ G4 WLAN モデルを子機として無線 LAN AP に接続しながら、hostapd を起動するような使い方は避けてください。

6.16.2.1. hostapd を使用して設定する

・ bridge インターフェースを追加する

NetworkManager で bridge インターフェース (br0) を追加します。同時に IP アドレスも設定します。ここでは 192.168.1.1 を設定しています。

```
[armadillo ~]# nmcli con add type bridge ifname br0
[armadillo ~]# nmcli con mod bridge-br0 ipv4.method manual ipv4.address "192.168.1.1/24"
[armadillo ~]# nmcli con up bridge-br0
[armadillo ~]# persist_file /etc/NetworkManager/system-connections/bridge-br0.nmconnection ❶
```

図 6.196 bridge インターフェースを作成する

❶ 設定ファイルを永続化します。

また、NetworkManager のデフォルト状態では定期的に mlan0 のスキャンを行っています。スキャン中は AP の性能が落ちてしまうため mlan0 を NetworkManager の管理から外します。

```
[armadillo ~]# vi /etc/NetworkManager/conf.d/90_disable_mlan0.conf
[device_mlan0]
match-device=interface-name:mlan0
managed=0

[armadillo ~]# persist_file /etc/NetworkManager/conf.d/90_disable_mlan0.conf
[armadillo ~]# nmcli d set mlan0 managed no ❶
```

図 6.197 mlan0 インターフェースを NetworkManager の管理から外す

❶ nmcli で NetworkManager をリスタートせずに設定します。

hostapd が使用するインターフェース (uap0) は /etc/NetworkManager/conf.d/00_disable_ap.conf ファイルによってデフォルトで NetworkManager の管理から外しております。

・ hostapd を設定する

hostapd の設定ファイルの雛形として用意してある /etc/hostapd/hostapd.conf.example をコピーして使用します。

```
[armadillo ~]# cp /etc/hostapd/hostapd.conf.example /etc/hostapd/hostapd.conf
[armadillo ~]# vi /etc/hostapd/hostapd.conf
```

```
hw_mode=a ❶
channel=44 ❷
ssid=myap ❸
wpa_passphrase=myap_pass ❹
interface=uap0
bridge=br0
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
driver=nl80211
country_code=JP
ctrl_interface=/var/run/hostapd
ctrl_interface_group=0
disassoc_low_ack=1
preamble=1
wmm_enabled=1
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
ieee80211ac=1
ieee80211ax=1
ieee80211n=1
ieee80211d=1
ieee80211h=1
logger_syslog=-1
logger_syslog_level=2
logger_stdout=-1
logger_stdout_level=2

[armadillo ~]# persist_file /etc/hostapd/hostapd.conf ❺
[armadillo ~]# rc-service hostapd start ❻
[armadillo ~]# rc-update add hostapd ❼
[armadillo ~]# persist_file /etc/runlevels/default/hostapd ❽
```

図 6.198 hostapd.conf を編集する

❶ 5GHz であれば a を、2.4GHz であれば g を設定します。

❷ 使用するチャンネルを設定します。

❸ 子機から接続するための任意の SSID を設定します。この例では myap を設定しています。

❹ 子機から接続するための任意のパスフレーズを設定します。この例では myap_pass を設定しています。

❺ 設定ファイルを永続化します。

❻ hostapd を起動します。

❼ Armadillo 起動時に hostapd が自動的に起動されるようにします。

❽ hostapd 自動起動の設定を永続化します。

・ dnsmasq を設定する

dnsmasq の設定ファイルを以下の内容で作成し /etc/dnsmasq.d/ 下に配置します。ファイル名は任意ですが、拡張子は .conf としてください。ここでは dhcp.conf としています。

```
[armadillo ~]# vi /etc/dnsmasq.d/dhcp.conf
interface=br0
bind-dynamic
dhcp-range=192.168.1.10, 192.168.1.254, 24h ❶

[armadillo ~]# persist_file /etc/dnsmasq.d/dhcp.conf ❷
[armadillo ~]# rc-service dnsmasq restart ❸
```

図 6.199 dnsmasq の設定ファイルを編集する

❶ 子機に割り当てる IP アドレスの範囲とリース期間を設定します。

❷ 設定ファイルを永続化します。

❸ dnsmasq を再起動します。

hostapd と dnsmasq の起動完了後、子機から hostapd.conf で設定した SSID とパスフレーズで接続できます。

6.16.3. ルータとして設定する

Armadillo-IoT ゲートウェイ G4 WLAN モデルをルータとして設定して外部ネットワークに接続する方法を説明します。ここでは外部ネットワークの例として一般的なインターネットを設定しています。

6.16.3.1. 無線 LAN 側に接続した機器から Ethernet 経由でインターネットに接続する

Armadillo-IoT ゲートウェイ G4 WLAN モデルを無線 LAN AP として設定し、AP に接続した子機から、 Ethernet(eth0/eth1) を経由してインターネットに接続する方法を説明します。

最初に、「6.16.2. 無線 LAN アクセスポイント (AP) として設定する」 を参照して AP の設定を完了させてください。

・ ip_forward を有効にする

ルータとして機能させるために、 ip_forward を有効にします。sysctl の設定ファイルを以下の内容で作成し /etc/sysctl.d/ 下に配置します。ファイル名は任意ですが、拡張子は .conf としてください。ここでは router.conf としています。

```
[armadillo ~]# vi /etc/sysctl.d/router.conf
net.ipv4.ip_forward = 1 ❶

[armadillo ~]# persist_file /etc/sysctl.d/router.conf ❷
[armadillo ~]# rc-service sysctl restart ❸
```

図 6.200 ip_forward を有効にする

❶ 1 (有効) に設定します。

❷ 設定ファイルを永続化します。

❸ sysctl を再起動して設定を反映させます。

・ iptables コマンドで NAT を設定する

```
[armadillo ~]# iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -o eth0 -j MASQUERADE ❶
[armadillo ~]# /etc/init.d/iptables save ❷
[armadillo ~]# rc-update add iptables ❸
[armadillo ~]# persist_file /etc/iptables/rules-save /etc/runlevels/default/iptables ❹
```

図 6.201 NAT を設定する

❶ ここで設定する IP アドレスのネットワーク部は AP に設定したものと合わせてください。

❷ iptables の設定を保存します。

❸ サービスを有効にします

❹ 保存した設定ファイルを永続化します。

設定完了後、AP を起動して子機を接続すると子機からインターネットに接続することができます。経由する Ethernet インターフェースを eth1 としたい場合は、 iptables の設定時に -o eth1 としてください。

6.16.3.2. Ethernet 側に接続した機器から無線 LAN 経由でインターネットに接続する

Armadillo-IoT ゲートウェイ G4 WLAN モデルの Ethernet(eth0/eth1) に接続した機器から、無線 LAN を経由してインターネットに接続する方法を説明します。

・ ip_forward を有効にする

「図 6.200. ip_forward を有効にする」 と同様の手順で設定します。

・ bridge インターフェースを追加して eth0 と eth1 を割り当てる

```
[armadillo ~]# nmcli con down "Wired connection 1" ❶
[armadillo ~]# nmcli con down "Wired connection 2"
[armadillo ~]# nmcli con add type bridge ifname br0 ❷
[armadillo ~]# nmcli con mod bridge-br0 ipv4.method manual ipv4.address "192.168.1.1/24" bridge.stp ⏎
off ❸
[armadillo ~]# nmcli con add type ethernet slave-type bridge ifname eth0 master bridge-br0 ❹
[armadillo ~]# nmcli con add type ethernet slave-type bridge ifname eth1 master bridge-br0 ❺
[armadillo ~]# nmcli con up bridge-br0
[armadillo ~]# nmcli con up bridge-slave-eth0
[armadillo ~]# nmcli con up bridge-slave-eth1
[armadillo ~]# persist_file /etc/NetworkManager/system-connections/bridge-br0.nmconnection ❻
[armadillo ~]# persist_file /etc/NetworkManager/system-connections/bridge-slave-eth0.nmconnection
[armadillo ~]# persist_file /etc/NetworkManager/system-connections/bridge-slave-eth1.nmconnection
```

図 6.202 bridge に eth0 と eth1 を割り当てる

❶ デフォルトで存在しているコネクションを down します。

❷ bridge インターフェースを作成します。

❸ 作成した bridge-br0 に任意の IP アドレスを設定し STP を無効にします。

❹ eth0 を bridge-br0 に割り当てます。

❺ eth1 を bridge-br0 に割り当てます。

❻ それぞれの設定ファイルを永続化します。

・dnsmasq を設定する

dnsmasq の設定ファイルを以下の内容で作成し /etc/dnsmasq.d/ 下に配置します。ファイル名は任意ですが、拡張子は .conf としてください。ここでは dhcp.conf としています。

```
[armadillo ~]# vi /etc/dnsmasq.d/dhcp.conf
interface=br0
bind-dynamic
dhcp-range=192.168.1.10, 192.168.1.254, 24h ❶

[armadillo ~]# persist_file /etc/dnsmasq.d/dhcp.conf
[armadillo ~]# rc-service dnsmasq restart ❷
```

図 6.203 dnsmasq の設定ファイルを編集する

❶ 接続した機器に割り当てる IP アドレスの範囲とリース期間を設定します。

❷ dnsmasq を再起動します。

・iptables コマンドで NAT を設定する

```
[armadillo ~]# iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -o mlan0 -j MASQUERADE ❶
[armadillo ~]# /etc/init.d/iptables save ❷
[armadillo ~]# rc-update add iptables ❸
[armadillo ~]# persist_file /etc/iptables/rules-save /etc/runlevels/default/iptables ❹
```

図 6.204 NAT を設定する

❶ ここで設定する IP アドレスのネットワーク部は bridge-br0 に設定したものと合わせてください。

❷ iptables の設定を保存します。

❸ サービスを有効にします

❹ 保存した設定ファイルを永続化します。

・無線 LAN AP に接続する

「6.16.1. 無線 LAN アクセスポイント (AP) に接続する」 と同様の手順で、無線 LAN アクセスポイントに接続します。

設定完了後、eth0/eth1 ポートに機器を接続すると無線 LAN 経由でインターネットに接続することができます。

6.17. 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。

6.17.1. 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 6.205 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 6.206 ストレージのマウント

6.17.2. 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 6.207 ストレージのアンマウント

6.17.3. 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 6.208. 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 6.208 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 6.209 EXT4 ファイルシステムの構築

6.18. 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。

デフォルトでは 「表 3.7. CON12 信号配列」 にある PWR_OFF と REBOOT を 3 秒以上押している(プルダウンしている)場合に poweroff か reboot を実行します。

/etc/atmark/buttond.conf に BUTTOND_ARGS を上書きすればその対応を無効にすることもできますし、別のキー（SW1 など）の対応も追加できます:

- --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。

6.18.1. 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 6.210 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。

6.18.2. 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 6.211 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 6.212 buttond で USB キーボードを扱う

6.18.3. 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 6.213 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。

6.19. 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。

6.19.1. 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。

6.19.2. 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 6.214 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。

6.19.3. atmark-thermal-profiler を実行・停止する

「図 6.215. 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 6.215 atmark-thermal-profiler を実行する

「図 6.216. 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 6.216 atmark-thermal-profiler を停止する

6.19.4. 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 6.217 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 6.19. thermal_profile.csv の各列の説明」に記載します。

表 6.19 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。

6.19.5. 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。

6.19.5.1. 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 6.218 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。

6.19.5.2. 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 6.219. 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。

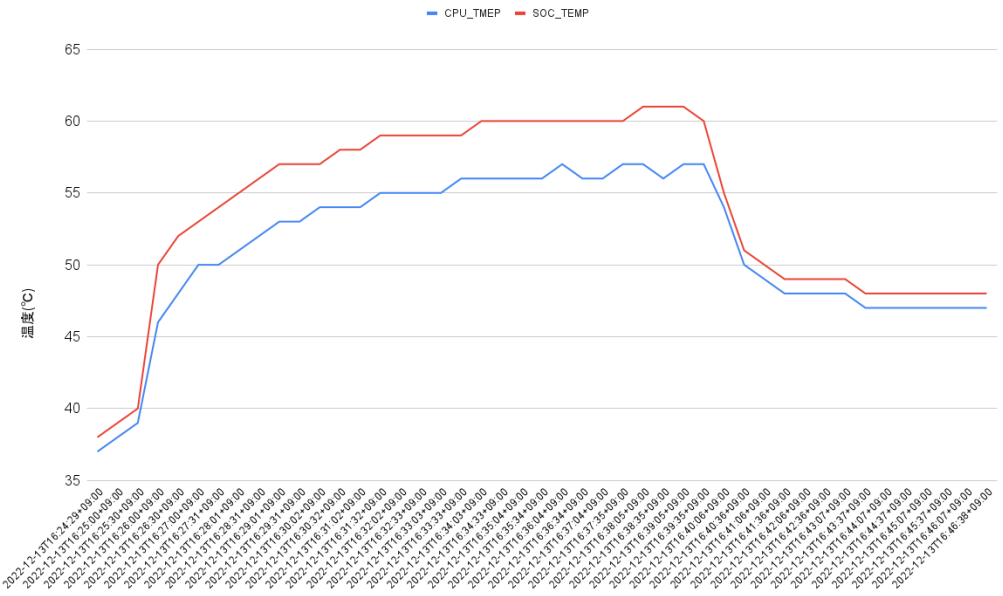


図 6.219 Armadillo-IoT ゲートウェイ G4 で取得した温度のグラフ

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、「6.19.5.1. 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。

6.19.5.3. 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 6.19. 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。

6.19.6. 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。

6.19.7. 温度センサーの仕様

Armadillo-IoT ゲートウェイ G4 の温度センサーは、i.MX 8M Plus の TMU(Thermal Monitoring Unit)を利用しています。CPU(Arm Cortex-A53)周辺温度と、SoC(ANAMIX 内部)温度を測定することができます。

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` (CPU) ・ `/sys/class/thermal/thermal_zone1` (SoC)

6.20. 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。

6.21. 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壊れて起動できなくなった場合に自動的に前のバージョンで再起動します。

自分で確認する必要がある場合に `abos-ctrl status` でロールバックされてるかどうかの確認ができます。

必要な場合（例えば、自分のアプリケーションがアップデート直後に問題があった場合)、 `abos-ctrl rollback` で手動のロールバックも可能です。ロールバックにエラーがなければ、再起動してロールバックを完了します。

なお、/var/at-log/atlog に切り替えの際に必ずログを書きますので、調査の時に使ってください。

```
[armadillo ~]# cat /var/at-log/atlog
Mar 17 14:51:35 armadillo NOTICE swupdate: Installed update to /dev/mmcblk2p2: ¥
extra_os.sshd: unset -> 1, extra_os.initial_setup: unset -> 1
Mar 17 16:48:52 armadillo NOTICE swupdate: Installed update to /dev/mmcblk2p1: ¥
boot: 2020.04-at5 -> 2020.04-at6, base_os: 3.15.0-at.3 -> 3.15.0-at.4
```

```
Mar 17 17:42:15 armadillo NOTICE swupdate: Installed update to /dev/mmcblk2p2: ¥
other_boot: 2020.04-at5 -> 2020.04-at6, container: unset -> 1, extra_os.container: unset -> 1
```

図 6.220 /var/at-log/atlog の内容の例

6.22. 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 6.221 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。

6.23. 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 6.222 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。

「6.25.1. 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 6.20 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。	ttymxc1,115200

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 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、「3.2.3.4. ロールバック（リカバリー）」を参照してください。	1
bootlimit	"bootcount"のロールバックを行うしきい値を指定します。	3
bootdelay	保守モードに遷移するためのキー入力を待つ時間を指定します(単位:秒)。次の値は特別な意味を持ちます。 ・ -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。	2
image	Linux カーネルイメージファイルのパスです。"mmcdev"で指定されたデバイスの、"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/Image
fdt_file	DTB ファイルのパスです。"mmcdev"で指定されたデバイスの、"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。 ・ 1: microSD/microSDHC/microSDXC カード ・ 2: eMMC "mmcautodetect"に yes が指定されている場合は、u-boot の起動時に上書きされます。	2
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/mmcblk2p1 rootwait ro
optargs	Linux カーネル起動時パラメータを指定します。"quiet"を削除すると、コンソールに起動ログが出力されるようになりますが、起動時間が長くなります。nokaslr を削除すると、KASLR(Kernel Adress Space Layout Randomization)が有効となり、Linux カーネルの仮想アドレス空間がランダム化されます。	quiet nokaslr
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。	0x40480000
fdt_addr	DTB が RAM にロードされる物理アドレスを指定します。	0x45000000
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。	0x45020000

6.24. 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。

6.24.1. ブートディスクの作成

1. ブートディスクイメージのビルドします

   「6.25.3. 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 「6.25.1. ブートローダーをビルドする」 でビルドした `imx-boot_armadillo_x2` を利用します。

   VPU や NPU も使用する場合は、 「6.10. VPU や NPU を使用する」 で用意した imx_lib.img も組み込めます。

```
[PC ~/build-rootfs-[VERSION]]$ sudo ./build_image.sh ¥
        --boot ~/imx-boot-[VERSION]/imx-boot_armadillo_x2 ¥
        --firmware ~/at-imxlibpackage/imx_lib.img
:（省略）
[PC ~/build-rootfs-[VERSION]]$ ls baseos-x2*img
baseos-x2-[VERSION].img
```

2. ATDE に microSD カードを接続します。詳しくは「3.3.2.7. 取り外し可能デバイスの使用」を参考にしてください。

3. microSD カードのデバイス名を確認します

```
[ATDE ~]$ ls /dev/sd?
/dev/sda  /dev/sdb
[ATDE ~]$ sudo fdisk -l /dev/sdb
Disk /dev/sdb: 7.22 GiB, 7751073792 bytes, 15138816 sectors
Disk model: SD/MMC
:（省略）
```

4. microSD カードがマウントされている場合、アンマウントします。

```
[ATDE ~]$ mount
:（省略）
/dev/sdb1 on /media/52E6-5897 type ext2
(rw,nosuid,nodev,relatime,uid=1000,gid=1000,fmask=0022,dmask=0077,codepage=cp437,iocharset
=utf8,shortname=mixed,showexec,utf8,flush,errors=remount-ro,uhelper=udisks)
[ATDE ~]$ sudo umount /dev/sdb1
```

図 6.223 自動マウントされた microSD カードのアンマウント

5. ブートディスクイメージの書き込み

```
[PC ~]$ sudo dd if=~/build-rootfs-[VERSION]/baseos-x2-[VERSION].img ¥
                of=/dev/sdb bs=1M oflag=direct status=progress
```

microSD カードの性能にもよりますが、書き込みには 5 分程度かかります。

microSD カードのパーティション構成は次のようになっています。

表 6.21 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。

```
[PC ~]$ sudo gdisk -l /dev/mmcblk1
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        15319006   6.5 GiB     8300  app
```

6.24.2. SD ブートの実行

「6.24.1. 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-IoT ゲートウェイ G4 に電源を投入する前に、ブートディスクを CON1(SD インターフェース)に挿入します。また、JP1 ジャンパーをショート(SD ブートに設定)します。

2. 電源を投入します。

```
U-Boot SPL 2020.04-at7 (May 21 2022 - 11:21:55 +0900)
rv8803 rtc woken by interrupt
DDRINFO: start DRAM init
DDRINFO: DRAM rate 4000MTS
DDRINFO:ddrphy calibration done
DDRINFO: ddrmix config done
Normal Boot
Trying to boot from BOOTROM
image offset 0x8000, pagesize 0x200, ivt offset 0x0
NOTICE:  BL31: v2.4(release):lf-5.10.y-1.0.0-0-gba76d337e956
NOTICE:  BL31: Built : 11:08:22, Apr  6 2022


U-Boot 2020.04-at7 (May 21 2022 - 11:21:55 +0900)

CPU:   i.MX8MP[8] rev1.1 1600 MHz (running at 1200 MHz)
CPU:   Industrial temperature grade (-40C to 105C) at 40C
Model: Atmark-Techno Armadillo X2 Series
DRAM:    Hold key pressed for tests: t (fast) / T (slow)
2 GiB
WDT:   Started with servicing (10s timeout)
MMC:   FSL_SDHC: 1, FSL_SDHC: 2
Loading Environment from MMC... OK
In:    serial
Out:   serial
Err:   serial

 BuildInfo:
  - ATF
  - U-Boot 2020.04-at7

first boot since power on
switch to partitions #0, OK
mmc1 is current device
flash target is MMC:1
Net:   eth0: ethernet@30be0000 [PRIME], eth1: ethernet@30bf0000
Fastboot: Normal
Saving Environment to MMC... Writing to MMC(1)... OK
Normal Boot
Hit any key to stop autoboot:  0
u-boot=>
```

3. ブートディスク上の Linux カーネルを起動します。

```
u-boot=> boot
```

6.25. Armadillo のソフトウェアをビルドする

ここでは、Armadillo-IoT ゲートウェイ G4 で使用するソフトウェアのビルド方法を説明します。

6.25.1. ブートローダーをビルドする

ここでは、Armadillo-IoT ゲートウェイ G4 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. ブートローダーのビルドに必要なパッケージのインストール

   次のコマンドを実行します。

```
[ATDE ~]$ sudo apt install build-essential git wget gcc-aarch64-linux-gnu libgcc-*-dev-
arm64-cross bison flex zlib1g-dev bash python3-pycryptodome python3-pyelftools device-tree-
compiler
```

2. ソースコードの取得

   Armadillo-IoT ゲートウェイ G4 ブートローダー [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/boot-loader] から「ブートローダー ソース」ファイル (imx-boot-[VERSION].tar.gz) を次のようにダウンロードします。

```
[ATDE ~]$ wget https://download.atmark-techno.com/{url-product-dir}/bootloader/imx-boot-
[VERSION].tar.gz
[ATDE ~]$ tar xf imx-boot-[VERSION].tar.gz
[ATDE ~]$ cd imx-boot-[VERSION]
```

3. ビルド

   次のコマンドを実行します。

```
[ATDE ~/imx-boot-[VERSION]]$ make imx-boot_armadillo_x2
:
:（省略）
:
Second Loader IMAGE:
 sld_header_off         0x58000
 sld_csf_off            0x59020
 sld hab block:         0x401fcdc0 0x58000 0x1020
make[1]: ディレクトリ '/home/atmark/imx-boot-[VERSION]/imx-mkimage' から出ます
cp imx-mkimage/iMX8M/flash.bin imx-boot_armadillo_x2
```

   初めてのビルドの場合、i.MX 8M Plus に必要なファームウェアの EULA への同意を求められます。内容を確認の上、同意してご利用ください。[3]

```
Welcome to NXP firmware-imx-8.11.bin

You need to read and accept the EULA before you can continue.
```

[3]スペースキーでページを送ると、最終ページに同意するかどうかの入力プロンプトが表示されます。

```
LA_OPT_NXP_Software_License v19 February 2021
:
:（省略）
:
Do you accept the EULA you just read? (y/N)
```

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 ~/imx-boot-[VERSION]]$ echo 'swdesc_boot imx-boot_armadillo_x2' > boot.desc
[ATDE ~/imx-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 「3.2.3.5. SWU イメージのインストール」を参照ください。

・「6.24.1. 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 imx-boot_armadillo_x2 を使ってください。

6.25.2. Linux カーネルをビルドする

ここでは、Armadillo-IoT ゲートウェイ G4 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-IoT ゲートウェイ G4 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。「6.25.3. 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. Linux カーネルのビルドに必要なパッケージのインストール

次のコマンドを実行します。

```
[ATDE ~]$ sudo apt update
[ATDE ~]$ sudo apt install crossbuild-essential-arm64 bison flex python3-pycryptodome
```

```
python3-pyelftools zlib1g-dev libssl-dev bc firmware-misc-nonfree wireless-regdb atmark-
firmware
```

2. ソースコードの取得

   Armadillo-IoT ゲートウェイ G4 Linux カーネル [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/linux-kernel] から「Linux カーネル」ファイル (linux-at-x2-[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-x2-[VERSION].tar
[ATDE ~]$ tar xf linux-at-x2-[VERSION]/linux-[VERSION].tar.gz
[ATDE ~]$ cd linux-[VERSION]
```

3. 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。

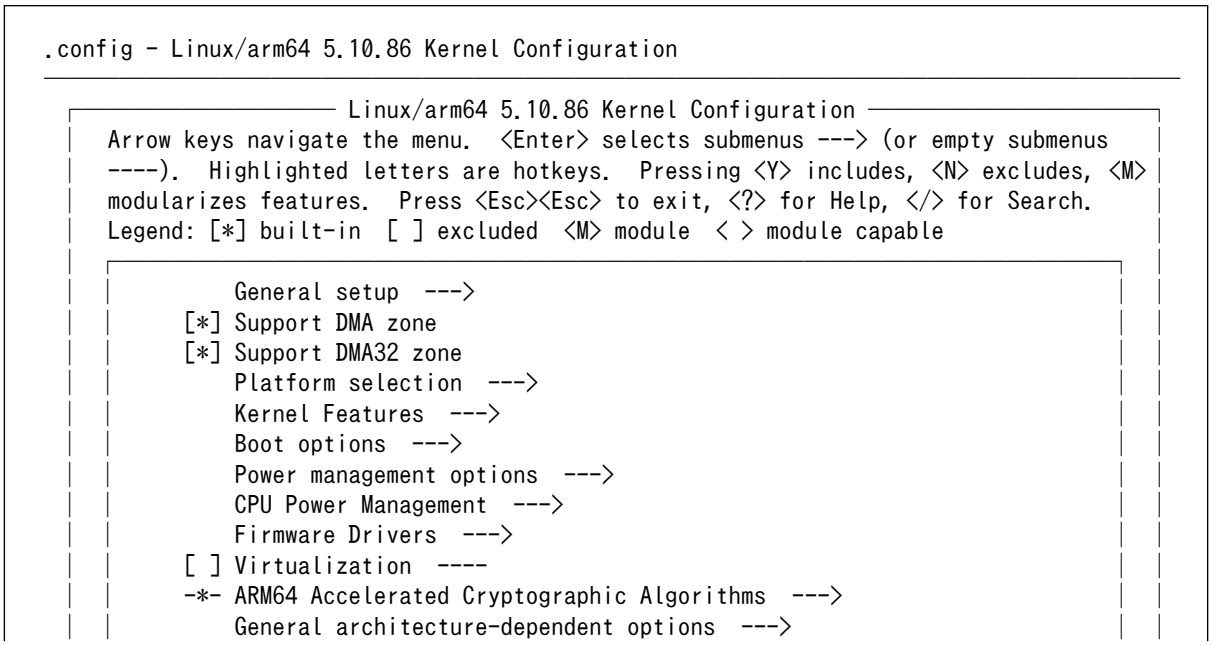
```
[ATDE ~/linux-[VERSION]]$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- x2_defconfig
```

4. カーネルコンフィギュレーションの変更

   次のコマンドを実行します。カーネルコンフィギュレーションの変更を行わない場合はこの手順は不要です。

```
[ATDE ~]$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- 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。



```
    [*] 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。

5. ビルド

次のコマンドを実行します。

```
[ATDE ~/linux-[VERSION]]$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- -j5
```

6. 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/arm64/boot/Image' -> '/home/atmark/mkswu/kernel/Image'
'arch/arm64/boot/dts/freescale/armadillo_iotg_g4.dtb' -> '/home/atmark/mkswu/kernel/
armadillo_iotg_g4.dtb'
: (省略)
  INSTALL arch/arm64/crypto/poly1305-neon.ko
  INSTALL drivers/block/loop.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 6.224 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 「3.2.3.5. 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 6.225. 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/d' "$BROOTFS/ax2/packages" ❷
[ATDE ~/linux-[VERSION]]$ cp -v arch/arm64/boot/Image "$BROOTFS/ax2/resources/boot/"
'arch/arm64/boot/Image' -> '/home/atmark/build-rootfs-v3.17-at.3/ax2/resources/boot/
Image'
[ATDE ~/linux-[VERSION]]$ cp -v arch/arm64/boot/dts/freescale/armadillo_*.{dtb,dtbo}
"$BROOTFS/ax2/resources/boot/"
'arch/arm64/boot/dts/freescale/armadillo_iotg_g4.dtb' -> '/home/atmark/build-rootfs-
v3.17-at.3/ax2/resources/boot/armadillo_iotg_g4.dtb'
'arch/arm64/boot/dts/freescale/armadillo_iotg_g4-at-dtweb.dtbo' -> '/home/atmark/build-
rootfs-v3.17-at.3/ax2/resources/boot/armadillo_iotg_g4-at-dtweb.dtbo'
:（省略）
[ATDE ~/linux-[VERSION]]$ rm -rfv "$BROOTFS/ax2/resources/lib/modules" ❸
[ATDE ~/linux-[VERSION]]$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu-
INSTALL_MOD_PATH="$BROOTFS/ax2/resources" -j5 modules_install
  INSTALL arch/arm64/crypto/poly1305-neon.ko
  INSTALL drivers/block/loop.ko
:（省略）
  DEPMOD  [VERSION]
```

図 6.225 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-x2@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。

6.25.3. 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 で動作している Linux 上で Armadillo-IoT ゲートウェイ G4 用の aarch64 アーキテクチャに対応した 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-IoT ゲートウェイ G4 開発用ツール [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/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://download.atmark-techno.com/{url-product-dir}/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

   ax2 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 ax2 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 ax2 内のサブディレクトリにある同名ファイルは、ax2 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 6.22 build-rootfs のファイル説明

ファイル	説明
ax2/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。

ファイル	説明
ax2/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。
ax2/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。
ax2/image_firstboot/*	配置したファイルやディレクトリは、「6.24.1. ブートディスクの作成」や「3.2.5.1. 初期化インストールディスクの作成」の手順のようにブートディスクイメージを作成する際、そのままルートファイルシステム直下にコピーされます。
ax2/image_installer/*	配置したファイルやディレクトリは、「3.2.5.1. 初期化インストールディスクの作成」の手順のようにインストールディスクイメージを作成する際、そのままインストーラ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。
ax2/image_common/*	配置したファイルやディレクトリは、ブートディスクイメージおよびインストールディスクイメージを作成する際、ルートファイルシステム、インストーラにそれぞれコピーされます。

利用可能なパッケージは以下のページで検索することができます。

Alpine Linux Packages https://pkgs.alpinelinux.org/packages

Alpine Linux ルートファイルシステムを起動している Armadillo でも検索することができます。

```
[armadillo ~]# apk update
[armadillo ~]# apk search ruby
ruby-test-unit-rr-1.0.5-r0
ruby-rmagick-5.1.0-r0
ruby-public_suffix-5.0.0-r0
:
:（省略）
:
ruby-mustache-1.1.1-r5
ruby-nokogiri-1.13.10-r0
```

4. ビルド

次のコマンドを実行します。

パッケージをインターネット上から取得するため回線速度に依存しますが、ビルドには数分かかります。

```
[ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_rootfs.sh
use default(board=ax2)
use default(arch=aarch64)
use default(outdir=/home/xxxx/at-optee-build/build-rootfs)
use default(output=baseos-x2-ATVERSION.tar.zst)
'repositories' -> '/etc/apk/repositories'
:
:（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-IoT ゲートウェイ G4 開発用ツール [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/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-x2-[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 ~/
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 「6.25.4. 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-x2-[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 「3.2.3.5. SWU イメージのインストール」 を参照ください。

- 「6.24.1. 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-x2-[VERSION].tar.zst が自動的に利用されます。

6.25.4. ビルドしたルートファイルシステムの SBOM を作成する

ここでは例として、「6.25.3. Alpine Linux ルートファイルシステムをビルドする」 で作成した OS_update.swu の 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-x2-[VERSION].package_list.txt から、パッケージの情報を記載することができます。

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パッケージ情報から SBOM を作成するには以下のコマンドを実行します。

```
[ATDE ~/build-rootfs-[VERSION]]$ ./make_sbom.sh -i OS_update.swu -c <コンフィグファイル> -p baseos-x2-[VERSION].package_list.txt
INFO:root:created OS_update.swu.spdx.json
```

このツールで作成される SBOM は 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。

当ツールで読み取ることが可能なライセンスは SPDX License List (https://spdx.org/licenses/) に含まれており、SPDX license expressions (https://spdx.github.io/spdx-spec/v2.2.2/SPDX-license-expressions/#d4-composite-license-expressions) に従っている必要があります。ライセンスが読み取れなかった場合は make_sbom.sh 実行時に以下のログが表示され、.spdx.json では NOASSERTION と記載されます。

```
WARNING:root:Failed to parse <パッケージ名> license: <ライセンス名>
```

アットマークテクノが提供している SBOM はソフトウェアダウンロード [https://armadillo.atmark-techno.com/armadillo-iot-g4/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。

6.26. eMMC のデータリテンション

eMMC は主に NAND Flash メモリから構成されるデバイスです。NAND Flash メモリには書き込みしてから 1 年から 3 年程度の長期間データが読み出されないと電荷が抜けてしまう可能性があります。その際、電荷が抜けて正しくデータが読めない場合は、eMMC 内部で ECC (Error Correcting Code) を利用してデータを訂正します。しかし、訂正ができないほどにデータが化けてしまう場合もあります。そのため、一度書いてから長期間利用しない、高温の環境で利用するなどのケースでは、データ保持期間内に電荷の補充が必要になります。電荷の補充にはデータの読み出し処理を実行し、このデータの読み出し処理をデータリテンションと呼びます。

Armadillo-IoT ゲートウェイ G4 に搭載の eMMC には長期間データが読み出されない状態であっても、データリテンションを自動的に行う機能を搭載しています。

詳しい仕様については 「6.26.3. 実装仕様に関する技術情報」 を参照してください。

6.26.1. データリテンションの設定

データリテンションは /etc/conf.d/micron_emmc_reten というファイルに書かれた設定、use_system_time によって以下の２通りの挙動を示します。

表 6.23 データリテンションの挙動

/etc/conf.d/micron_emmc_reten	initiating condition
use_system_time=yes	Linux 起動した時に前回のリテンションから 1 日以上経過していたら開始する
use_system_time=no (default)	Linux 起動した時に毎回開始する

これで設定は完了しました。

以下は挙動ごとのシステム概略図です。

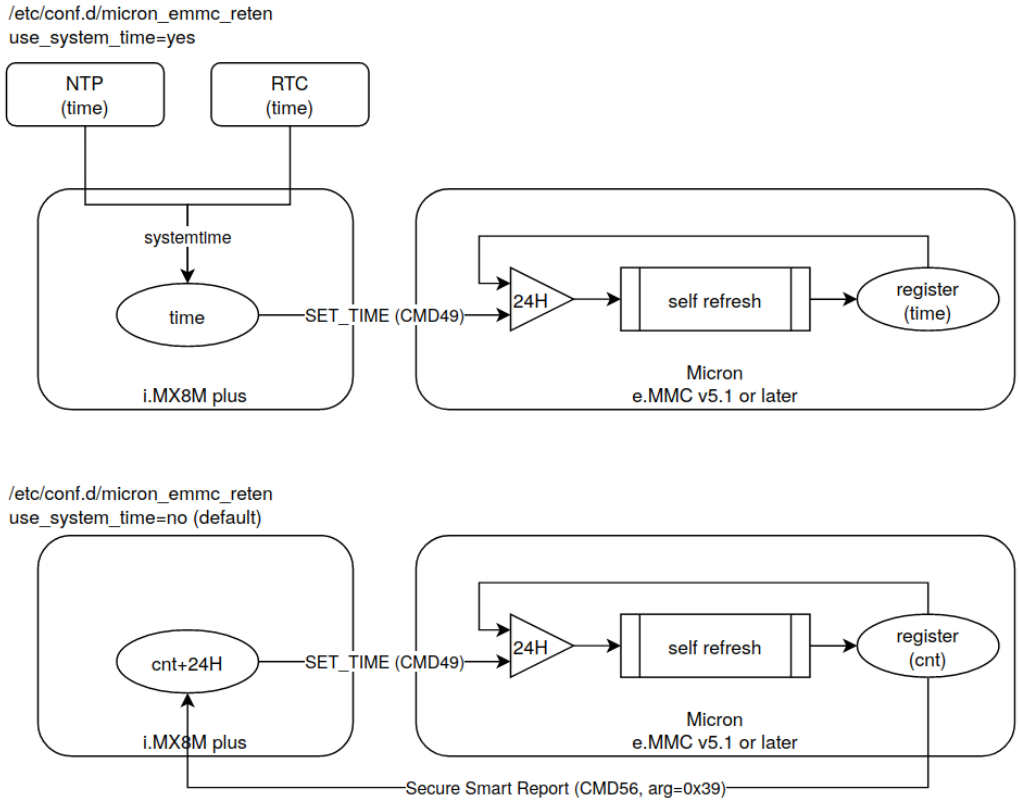


図 6.226 データリテンション開始トリガーの方式

use_system_time を有効にした場合のデータリテンションの動作例を以下に示します。

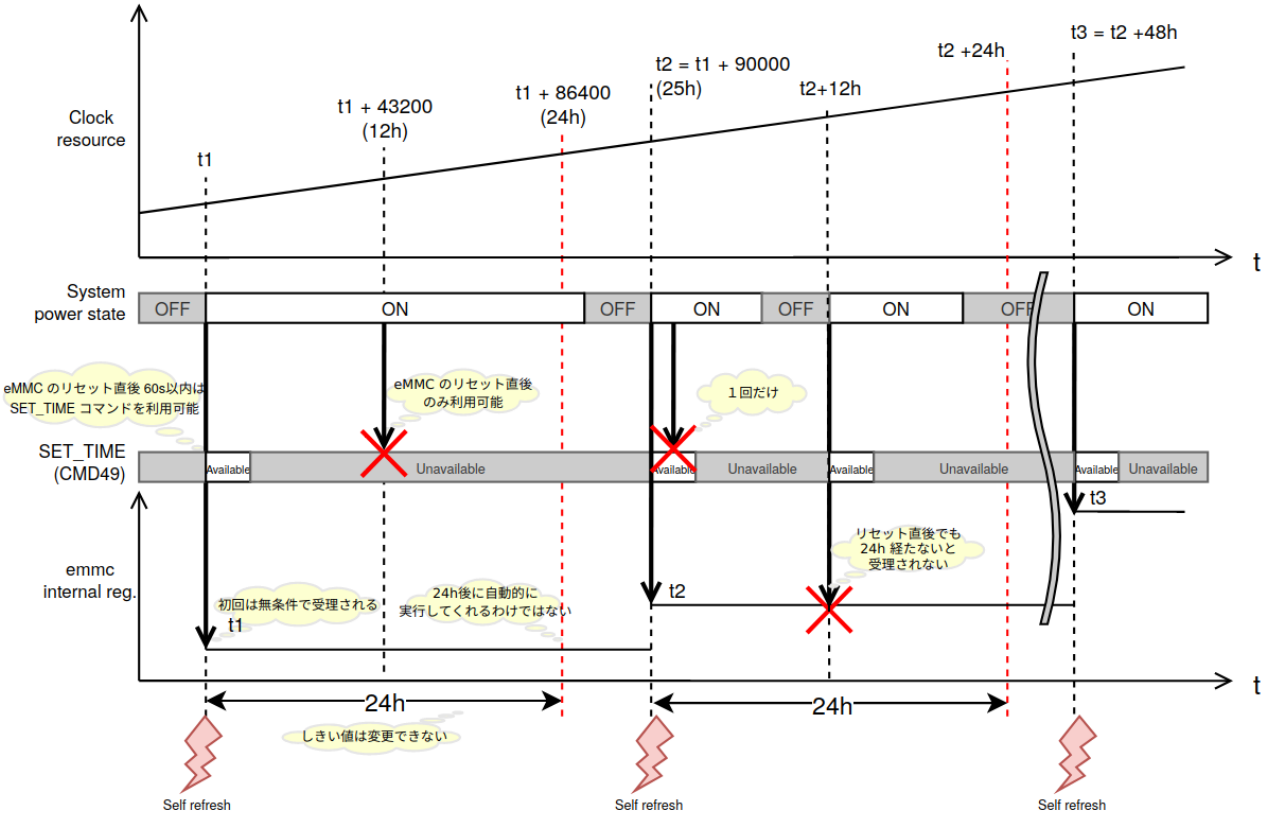


図 6.227 データリテンションの開始トリガーの動作例

6.26.2. より詳しくデータリテンションの統計情報を確認するには

Micron Technology が提供する emmcparm というツールを使うことで、データリテンションの統計情報を確認することができます。統計情報として eMMC 内部に保存されているのは実行回数、最終実行完了時のカウンター値、現在のデータリテンション処理の進捗があります。次の手順で、emmcparm を使って eMMC の情報を確認することができます。このツールではデータリテンション処理のことを「セルフリフレッシュ」と呼びます。

1. emmcparm をダウンロードする

   以下の検索結果から最新の emmcparm をダウンロードする。ユーザー登録が必要になります。

   emmcparmhttps://jp.micron.com/search-results?searchRequest=%7B%22term%22%3A%22emmcparm%20%22%7D

   マニュアル作成時点では 5.0.0 を利用しました

2. パッケージを展開する

```
[armadillo ~]# unzip emmc_emmcparm_c_code_derived_from_TN¥ FC¥ 25_v5.0.0 _binary.zip
```

3. SSR を取得する

```
[armadillo ~]# emmcparm/bin/emmcparm_arm_64bit -r /dev/mmcblk2
```

```
:（省略）
=========================================================================
|                          Secure Smart Report                          |
=========================================================================
  Self Refresh progress of scan[215-212]:              0x00000000 (0) ❶
  Power Loss Counter[195-192]:                         0x00000005 (5)
  Current total ON time[131-128]:                      0x00001b28 (6952)
  Number of Blocks in Refresh Queue[99-96]:            0x00000000 (0)
  Self Refresh Completion date [95-88]:                0xfffffffffd8148931 ❷
                                                       (-669742799)
  Self Refresh Loop Count[81-80]:                      0x00000002 (2)  ❸
  Written Data 100MB Size Count (from NAND)[79-76]:    0x0004889d (297117)
  Cumulative Initialization Count (from NAND)[75-72]:  0x00005300 (21248)
  Written Data 100MB Size Count (from RAM)[71-68]:     0x0004889d (297117)
  Refresh Count[55-52]:                                0x00000004 (4)
:（省略）
```

❶ 現在のセルフリフレッシュ処理の進捗。0 ということは実行中ではない

❷ 最後に行ったセルフリフレッシュのカウンター値

❸ セルフリフレッシュを行った回数

6.26.3. 実装仕様に関する技術情報

ここではデータリテンションを自動的におこなう機能の仕様について詳細に説明します。Armadillo で採用している eMMC には、データリテンションを自動的に実行することができる「セルフリフレッシュ」と呼ばれる機能が搭載されます。実行トリガーは 2 種類のうちどちらかを選択できます。OTP のため一度設定すると変更できません。この設定は出荷時に「eMMC 内部レジスタ値とコマンドに入力された値を比較して 1 日以上経過していると実行する」を設定しています。

1. リセット後に毎回実行する
2. eMMC 内部レジスタ値とコマンドに入力された値を比較して 1 日以上経過していると実行する

2 の設定の場合、セルリフレッシュ機能が実行されるまでの流れは以下のとおりです。

1. ホストによって eMMC がハードウェアもしくはソフトウェアリセットされる
2. 一定時間 (delay 1) 以内に、ホストから SET_TIME (CMD49)と呼ばれるコマンドが eMMC に発行される
3. eMMC コントローラは、バスの稼動状態を監視する
4. eMMC コントローラは、アイドルになってから一定時間 (delay 2) 経過した後にセルフリフレッシュを実行する

・ ECC エラーなどのエラーがしきい値 (2) を越えたセルに対してのみセルフリフレッシュを実行する

Armadillo でのセルフリフレッシュ機能搭載 eMMC への設定は以下のとおりです。

表 6.24 Armadillo のデータリテンションの設定

setting	value	description
RTC	ON	eMMC 内部レジスタの値と SET_TIME の値を比較してセルフリフレッシュを実行する
Delay 1	60s	リセット後の SET_TIME 有効期間
Delay 2	100ms	アイドル確認後のセルフリフレッシュ実行までの遅れ時間

詳しい情報は以下を参照してください。

Refresh Features for Micron e.MMC Automotive 5.1 Devices https://jp.micron.com/search-results?searchRequest=%7B%22term%22%3A%22TN-FC-60%22%7D

マイクロンのサイトの会員登録が必要になります。

6.27. Linux カーネルがクラッシュしたときにメモリの状態を保存する

Armadillo は Linux カーネルがクラッシュすると、ウォッチドッグタイマーによってシステムリセットが発生し、再起動します。

このとき、再起動によってメモリの内容が失われてしまうため、デバッグが困難になる場合があります。

ここでは Kdump を利用して Linux カーネルがクラッシュしたときにメモリの状態(vmcore)を保存し、vmcore を解析する方法を紹介します。

6.27.1. Kdump を利用する準備

ここでは、Kdump の実行環境を構築する手順を紹介します。

1. Linux カーネルの準備

   Armadillo の Linux カーネルをデバッグ用に変更します。以下で紹介する二つの方法のどちらかを選択してください。

   1. ビルド済みの apk パッケージを利用する場合

      以下のコマンドを実行します。

```
[armadillo ~]# persist_file -a del linux-at-x2
[armadillo ~]# persist_file -a add linux-at-x2-debug
```

2. Linux カーネルをビルドする場合

   以下のようにカーネルコンフィギュレーションを変更してください。

```
Kernel hacking  --->
  Compile-time checks and compiler options  --->
    [*] Compile the kernel with debug info              <DEBUG_INFO> ❶
    [ ]   Reduce debugging information          <DEBUG_INFO_REDUCED> ❷
```

   ❶ チェックを入れます

   ❷ チェックを外します

   「6.25.2. Linux カーネルをビルドする」を参照して、ビルドおよびインストールしてください。

2. パッケージのインストール

   kdump-tools をインストールします。

```
[armadillo ~]# persist_file -a add kdump-tools
```

3. 設定ファイルの編集

   Kdump の設定ファイルを編集します。

```
[armadillo ~]# vi /etc/conf.d/kdump-tools
# kdump-tools configuration
:(省略)
KDUMP_KERNEL=/boot/Image ❶
#KDUMP_INITRD=/var/lib/kdump/initrd.img ❷
:(省略)
KDUMP_COREDIR="/var/app/volumes/kdump" ❸
:(省略)
[armadillo ~]# persist_file /etc/conf.d/kdump-tools ❹
```

   ❶ Linux カーネルイメージのパスを指定します。

   ❷ initrd は利用しないのでコメントアウトします。

   ❸ vmcore を保存するディレクトリを指定します。少なくとも 30MByte 以上の空き容量が必要です。

   ❹ ファイルを永続化します。

4. kdump-tools サービスの有効化

   起動時に、自動的に kdump-tools サービスを有効化するようにします。

```
[armadillo ~]# rc-update add kdump-tools ❶
[armadillo ~]# persist_file /etc/runlevels/default/kdump-tools ❷
```

❶ kdump-tools サービスの自動起動を有効にします。

❷ ファイルを永続化します。

5. Linux カーネル起動時パラメータの指定

   Kdump で利用するメモリサイズを、Linux カーネル起動時パラメータの crashkernel に指定します。「6.23. u-boot の環境変数の設定」を参照し、環境変数 optargs を設定してください。

   以下の例では、Kdump で利用するメモリサイズを 128MByte に指定しています。

```
[armadillo ~]# vi /boot/uboot_env.d/kdump ❶
optargs=quiet nokaslr crashkernel=128M ❷
[armadillo ~]# persist_file -v /boot/uboot_env.d/kdump ❸
'/boot/uboot_env.d/kdump' -> '/mnt/boot/uboot_env.d/kdump'
[armadillo ~]# fw_setenv -s /boot/uboot_env.d/kdump ❹
Environment OK, copy 0
[armadillo ~]# fw_printenv | grep optargs= ❺
optargs=quiet nokaslr crashkernel=128M
[armadillo ~]# reboot ❻
```

   ❶ コンフィグファイルを生成します。

   ❷ デフォルト値である"quiet nokaslr"の後ろに追加しています。デフォルト値が不要であれば、削除しても問題ありません。

   ❸ ファイルを永続化します。

   ❹ 変数を書き込みます。

   ❺ 書き込んだ変数を確認します。

   ❻ 再起動して、設定を反映させます。

   以上で、Kdump を利用する準備は完了です。Linux カーネルがクラッシュした場合に、Kdump によって vmcore が保存されるようになりました。

6.27.2. Kdump の動作確認

ここでは、故意に Linux カーネルをクラッシュさせ、Kdump の動作確認を行う手順を紹介します。

```
[armadillo ~]# echo 1 > /proc/sys/kernel/sysrq ❶
[armadillo ~]# echo c > /proc/sysrq-trigger ❷
[   19.295633] sysrq: Trigger a crash
[   19.299079] Kernel panic - not syncing: sysrq triggered crash
: (省略)
[   19.386503] Starting crashdump kernel...
[   19.390426] Bye!
[    0.000000] Booting Linux on physical CPU 0x0000000003 [0x410fd034] ❸
: (省略)
kdump-tools              |makedumpfile Completed.
kdump-tools              |kdump-config: saved vmcore in /var/app/volumes/kdump/202303101530
kdump-tools              |Fri, 10 Mar 2023 15:30:39 +0900
[   20.189148] imx2-wdt 30280000.watchdog: Device shutdown: Expect reboot!
[   20.201853] reboot: Restarting system ❹
```

```
：（省略）
[armadillo ~]# ls /var/app/volumes/kdump/202303101530 ❺
dmesg.202303101530  dump.202303101530 ❻
```

❶ SysRq キーを有効化します。

❷ SysRq キーの"c"コマンドを実行して Linux カーネルをクラッシュさせます。

❸ Kdump に指定した Linux カーネルがブートローダーを経由せずに起動します。

❹ Kdump が vmcore を保存した後、自動的に再起動します。

❺ 作成されたファイルを確認します。

❻ dmesg.[DATE]は Linux カーネルのログです。dump.[DATE]は vmcore です。

Armadillo の再起動が完了後、Kdump のログに表示されたディレクトリ(/var/app/volumes/kdump/[DATE]/)から、Linux カーネルがクラッシュした状態での vmcore や dmesg を確認することができます。

vmcore が保存されるディレクトリおよび、そのディレクトリ内に作成されるファイル名に付与される日時は次のコマンドで作られています。

```
date +"%Y%m%d%H%M"
```

6.27.3. vmcore の確認

ここでは、vmcore の内容を確認する手順を紹介します。

vmcore の内容を確認するには、次の 3 つが必要です。

- vmcore
- vmlinux
- crash コマンド

vmcore は、「6.27.2. Kdump の動作確認」で作成した /var/app/volumes/kdump/[DATE]/dump.[DATE] です。vmlinux および crash コマンドの準備については、以下の手順を参照してください。

1. vmlinux の準備

   現在動作している Linux カーネルと一緒にビルドされた vmlinux を取得します。

   「6.27.1. Kdump を利用する準備」でどちらの Linux カーネルを選択したかによって手順が異なります。以下で紹介する二つの方法のどちらかを選択してください。

   1. ビルド済みの apk パッケージに含まれている Linux カーネルが動作している場合

      以下のコマンドを実行して vmlinux を取得します。vmlinux は、ホストとコンテナ間で共有する/var/app/volumes/kdump/に配置します。

      ```
      [armadillo ~]# cd /var/app/volumes/kdump/
      [armadillo /var/app/volumes/kdump]# apk fetch linux-at-x2-dbg
      ```

```
[armadillo /var/app/volumes/kdump]# mv linux-at-x2-dbg-[VERSION].apk linux-at-x2-
dbg.tar.gz
[armadillo /var/app/volumes/kdump]# tar xf linux-at-x2-dbg.tar.gz
[armadillo /var/app/volumes/kdump]# ln -s usr/lib/debug/lib/modules/[VERSION]/vmlinux .
```

2. ビルドした Linux カーネルが動作している場合

   ビルドした Linux カーネルディレクトリ直下に vmlinux が作成されています。

```
[armadillo ~]# ls linux-[VERSION]/vmlinux
linux-[VERSION]/vmlinux
```

   vmlinux を/var/app/volumes/kdump/にコピーしてください。

2. crash コマンドの準備

   crash コマンドを利用する為に、「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 debian コンテナを作成してください。

crash コマンドが利用できるディストリビューションであれば、debian 以外を利用しても構いません。

   以下のコマンドを実行して crash をインストールします。

```
[armadillo ~]# vi /etc/atmark/containers/kdump.conf
set_image localhost/at-debian-image:latest
set_command sleep infinity
add_volumes /var/app/volumes/kdump:/mnt:ro ❶
[armadillo ~]# podman_start kdump
Starting 'kdump'
8e7ad42534e3fb968dbf597d679246346ae4f766ac33ab0265008f30a7bf7d11
[armadillo ~]# podman exec -it kdump bash
[container /]# apt install crash ❷
```

   ❶ ホスト OS 側の /var/app/volumes/kdump をコンテナ内の /mnt にマウントするように設定します。

   ❷ crash コマンドを含む crash パッケージをインストールします。

3. vmcore の確認

   以下のコマンドを実行して crash を起動します。起動に成功すると crash のプロンプトが表示され、不具合の解析を行うことができるようになります。

```
[container /]# crash /mnt/vmlinux /mnt/[DATE]/dump.[DATE]
:(省略)
crash>
```

crash のコマンド一覧は、help コマンドで確認できます。

```
crash> help
```

help の引数にコマンドを与えると、そのコマンドの詳細を確認できます。例として bt コマンドの詳細は以下のように確認できます。

```
crash> help bt
```

6.28. 動作ログ

6.28.1. 動作ログについて

Armadillo-IoT ゲートウェイ G4 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。

通常のログは /var/log/messages に出力されます。

/var/log/messages はファイルサイズが 4MB になるとローテートされ /var/log/messages.0 に移動されます。

/var/log/messages.0 が存在する状態で、更に /var/log/messages のファイルサイズが 4MB になった場合は、 /var/log/messages の内容が /var/log/messages.0 に上書きされます。 /var/log/messages.1 は生成されません。

6.28.2. 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 3MB 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 3MB 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。

6.28.3. 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 6.228 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 による再起動が合った場合にその日付

6.28.4. 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/mmcblk2gp1 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/mmcblk2gp1 のデータを /dev/mmcblk2gp2 にコピーし、/dev/mmcblk2gp1 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。

6.29. 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。

6.29.1. 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 6.229 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。

6.29.2. 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 6.25. 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 6.25 入力モードに移行するコマンド

コマンド	動作
i	カーソルのある場所から文字入力を開始
a	カーソルの後ろから文字入力を開始

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

日本語変換機能を OFF に

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

「i」、「a」それぞれのコマンドを入力した場合の文字入力の開始位置を「図 6.230. 入力モードに移行するコマンドの説明」に示します。

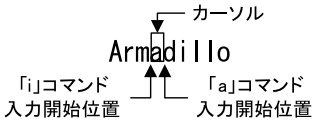


図 6.230 入力モードに移行するコマンドの説明

vi での文字削除

コンソールの環境によっては BS(Backspace)キーで文字が削除できず、「^H」文字が入力される場合があります。その場合は、「6.29.4. 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。

6.29.3. 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 6.26. 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 6.26 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

6.29.4. 文字の削除

文字を削除する場合は、コマンドモードで「表 6.27. 文字の削除コマンド」に示すコマンドを入力します。

表 6.27 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 6.231. 文字を削除するコマンドの説明」に示します。

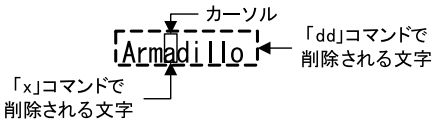


図 6.231 文字を削除するコマンドの説明

6.29.5. 保存と終了

ファイルの保存、終了を行うコマンドを「表 6.28. 保存・終了コマンド」に示します。

表 6.28 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。

6.30. オプション品

本章では、Armadillo-IoT ゲートウェイ G4 のオプション品について説明します。

表 6.29 Armadillo-IoT ゲートウェイ G4 関連のオプション品

名称	型番	備考
オプションケース(金属製)	-	開発セットに付属
CPU 放熱シート 15x15x4mm 10 個セット	OP-THS-151504-00	
3G/LTE 用外付けアンテナ、アンテナケーブル	-	LTE モデル開発セット, LTE+WLAN モデル開発セットに付属
AC アダプタ（12V/3.0A φ2.1mm）標準品	OP-AC12V6-00	開発セットに付属
無線 LAN/BT 用外付けアンテナ、アンテナケーブル	-	LTE+WLAN モデル開発セットに付属
Armadillo-X2、G4 ケースモデル VESA 規格固定用プレート	OP-CASEX2-VESA-00	

6.30.1. オプションケース(金属製)

6.30.1.1. 概要

Armadillo-IoT ゲートウェイ G4 用のアルミ製ケースです。基板を収めた状態で、DC ジャック、LAN ×2、USB、HDMI、USB コンソール、スイッチ、LED にアクセスすることが可能となっています。

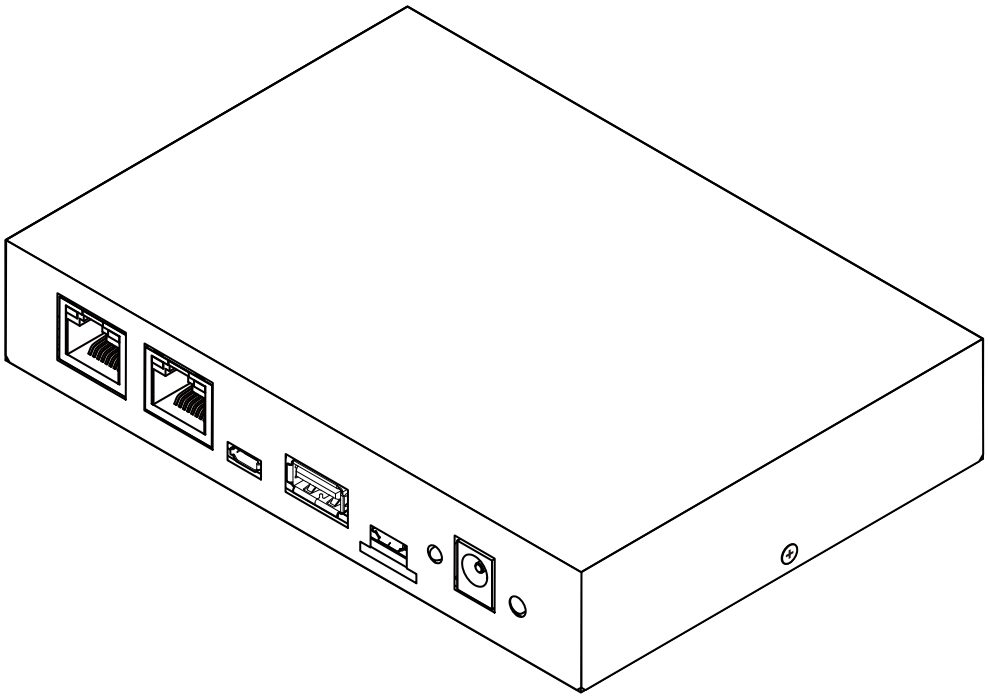


図 6.232 オプションケース(金属製)

コネクタ開口部等に存在する継ぎ目状の加工痕は正常な状態ですのでご了承ください。

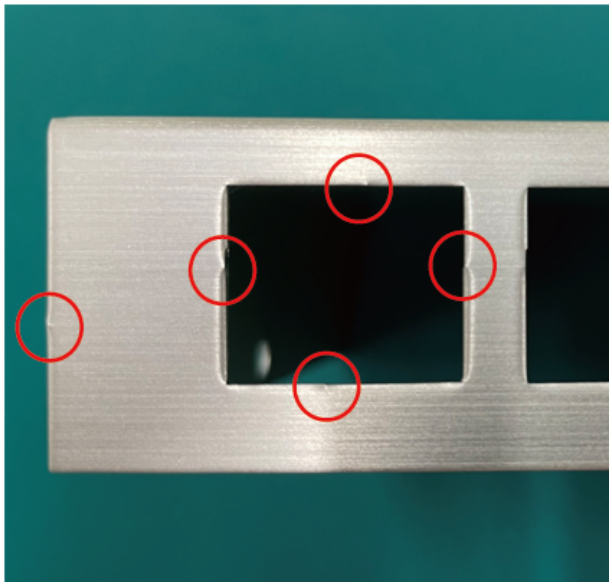


図 6.233 オプションケース(金属製)の加工痕例

6.30.1.2. 組み立て

組み立て手順に関しては、「4.7.2. オプションケース(金属製)への組み付け」を参照してください。

6.30.1.3. 形状図

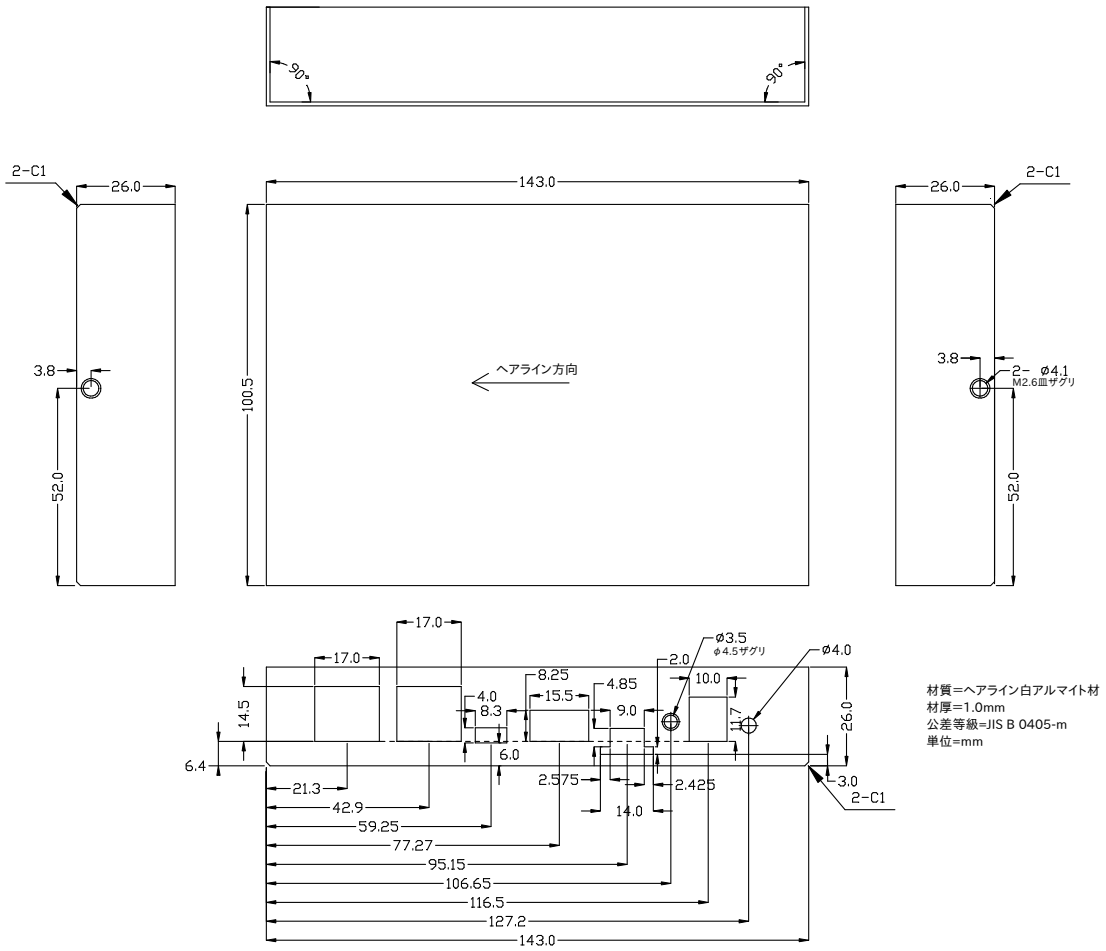


図 6.234 LAN モデル ケース(上)形状図

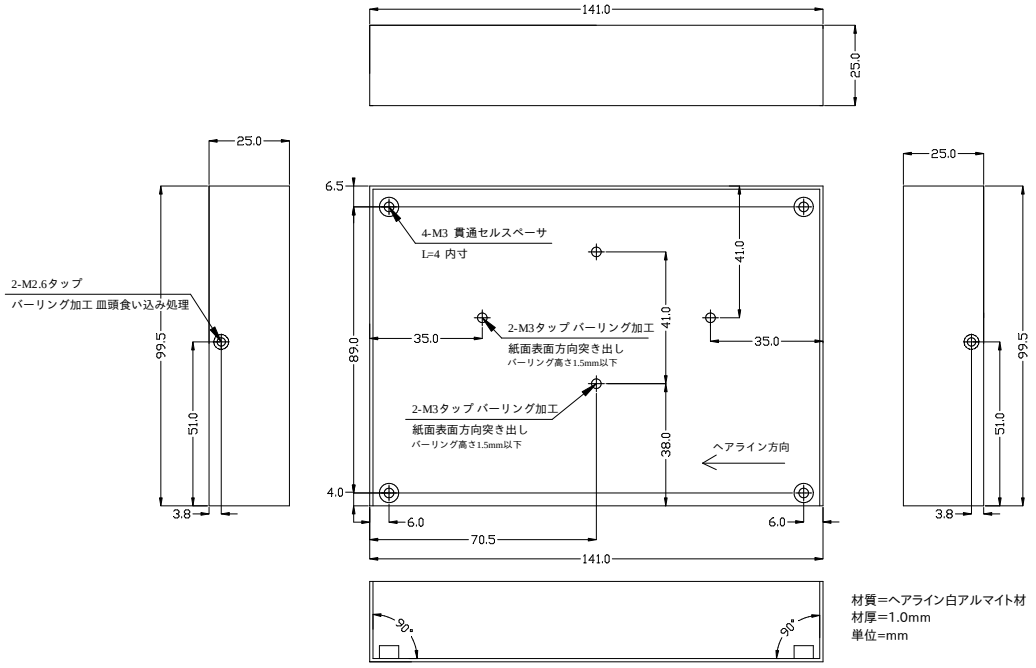


図 6.235 LAN モデル ケース(下)形状図

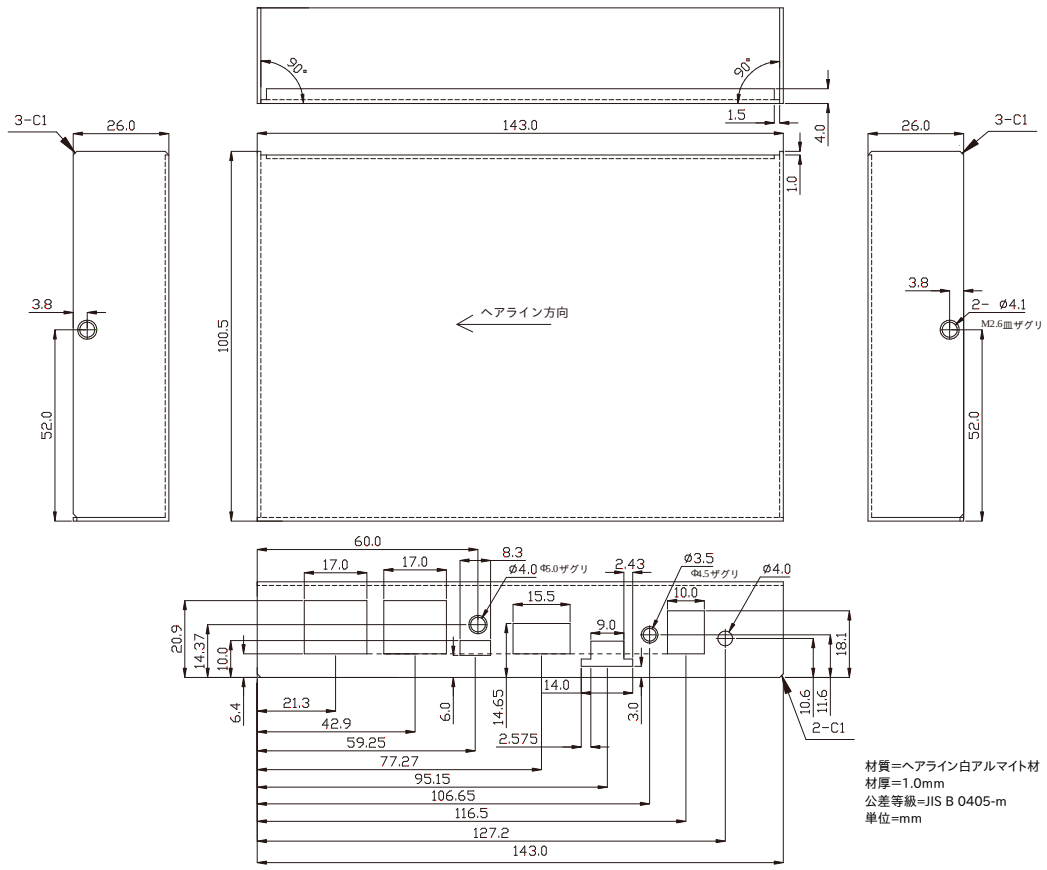


図 6.236 LTE モデル ケース(上)形状図

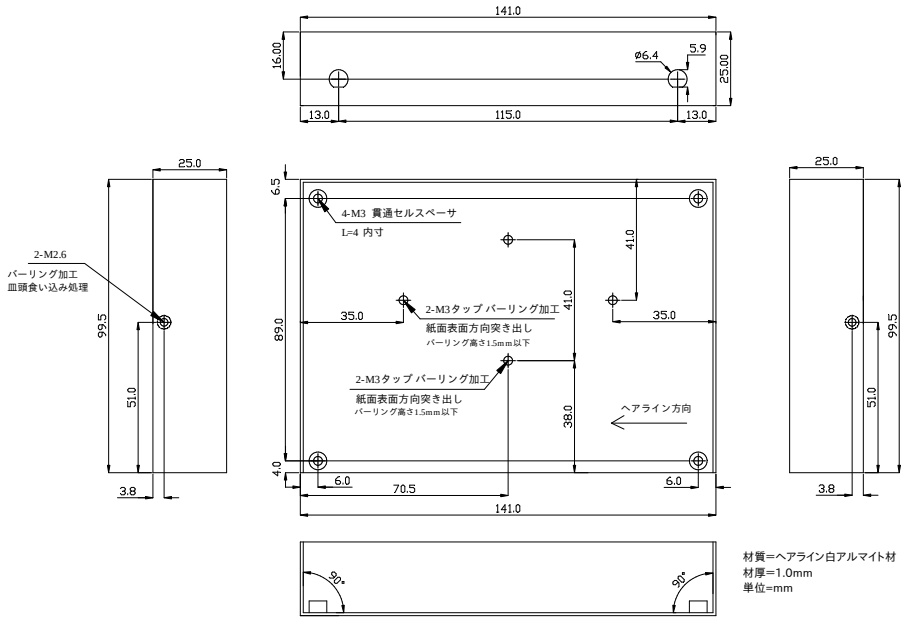


図 6.237 LTE モデル ケース(下)形状図

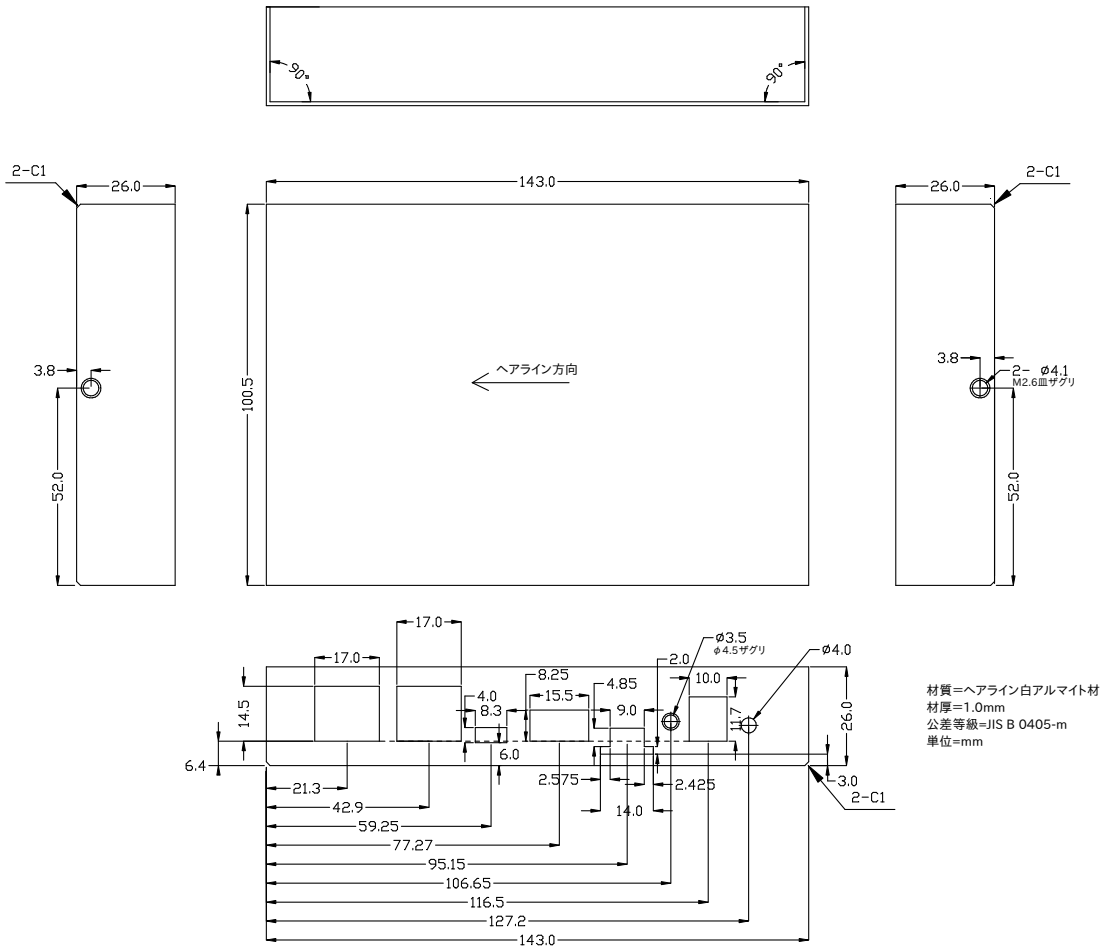


図 6.238 WLAN モデル ケース(上)形状図

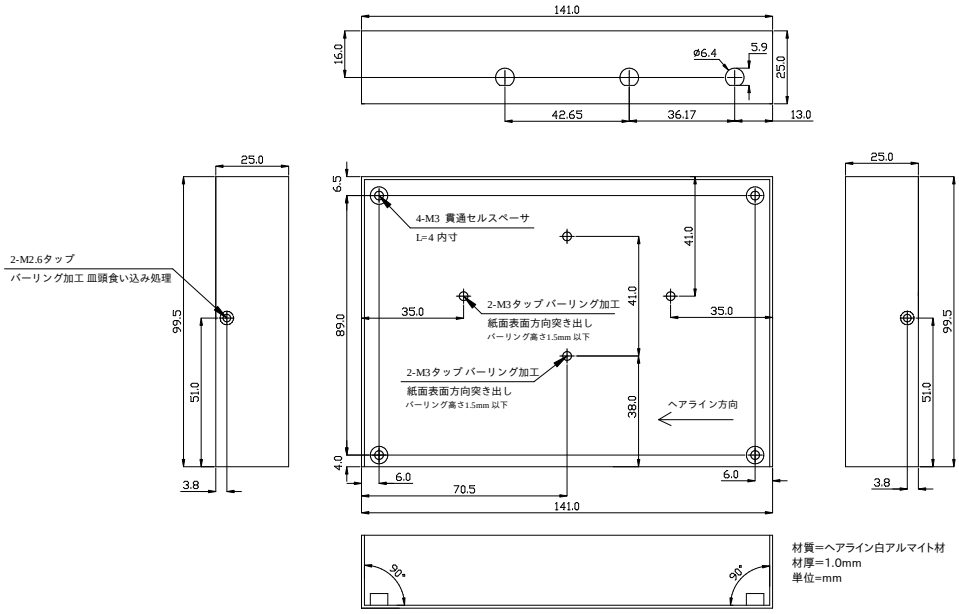


図 6.239 WLAN モデル ケース(下)形状図

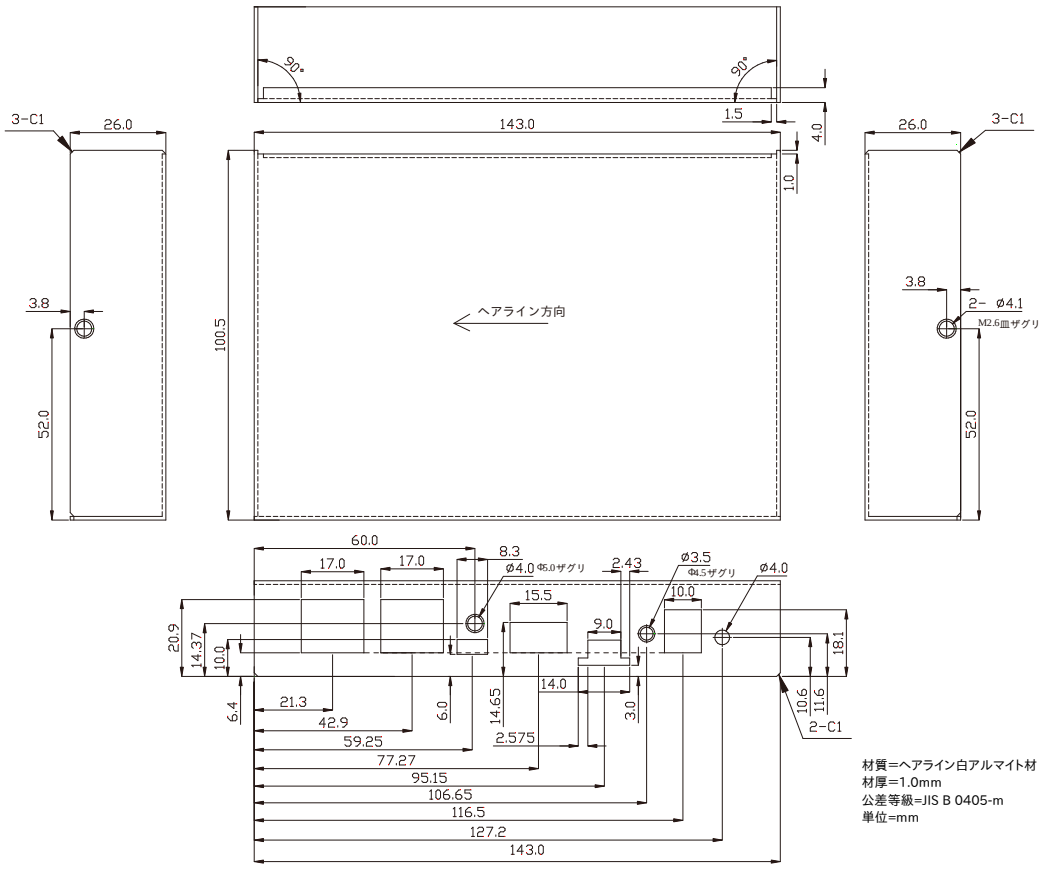


図 6.240 LTE+WLAN モデル ケース(上)形状図

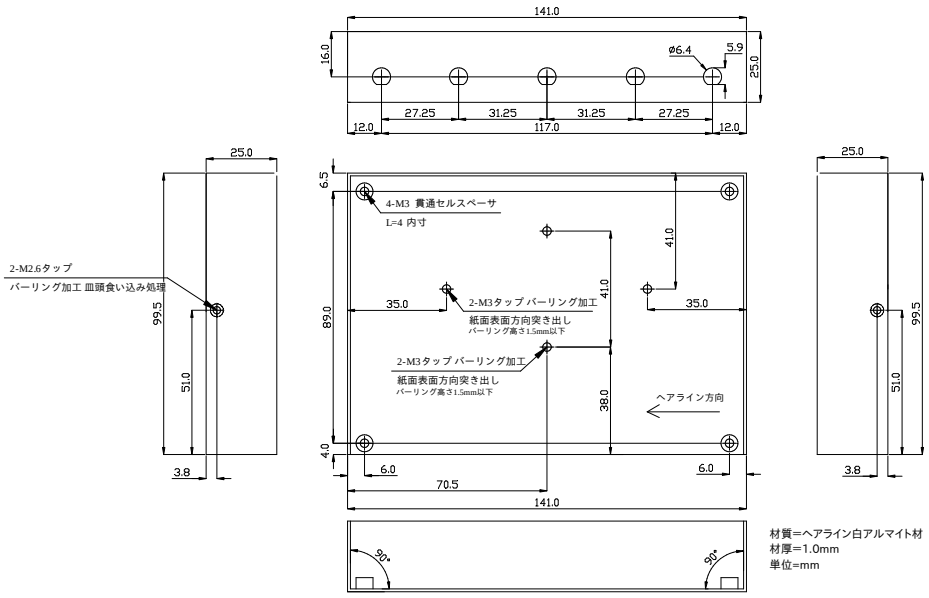


図 6.241 LTE+WLAN モデル ケース(下)形状図

DXF 形式の形状図を「アットマークテクノ Armadillo サイト」から「購入者向けの限定公開データ」としてダウンロード可能です。

6.30.2. 3G/LTE 用外付けアンテナ、アンテナケーブル

6.30.2.1. 概要

3G/LTE 用の外付けアンテナおよびアンテナケーブルです。

6.30.2.2. 形状図

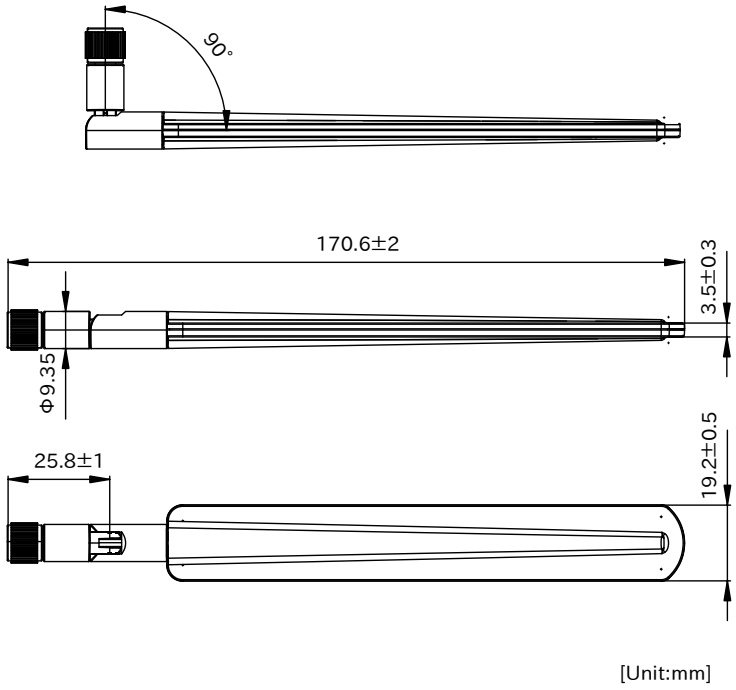


図 6.242 3G/LTE 用外付けアンテナ形状図

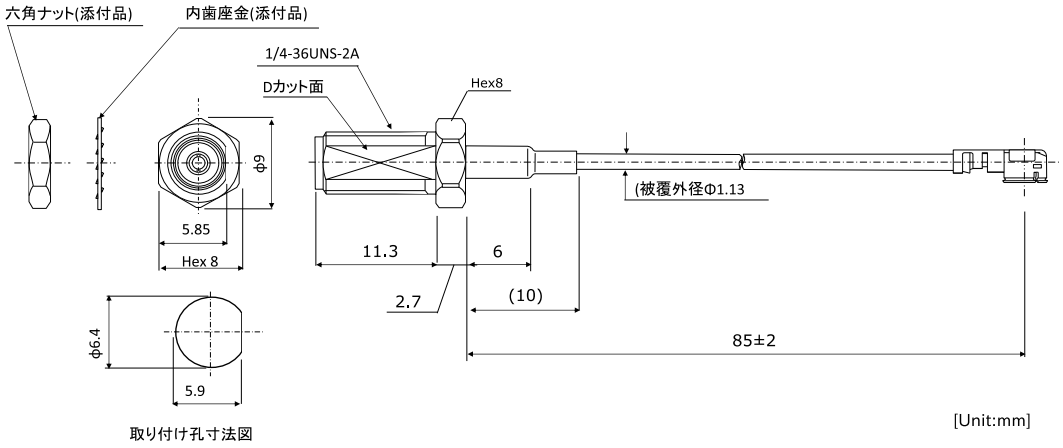


図 6.243 3G/LTE 用アンテナケーブル形状図

6.30.3. 無線 LAN/BT 用外付けアンテナ、アンテナケーブル

6.30.3.1. 概要

無線 LAN/BT 用の外付けアンテナおよびアンテナケーブルです。

6.30.3.2. 形状図

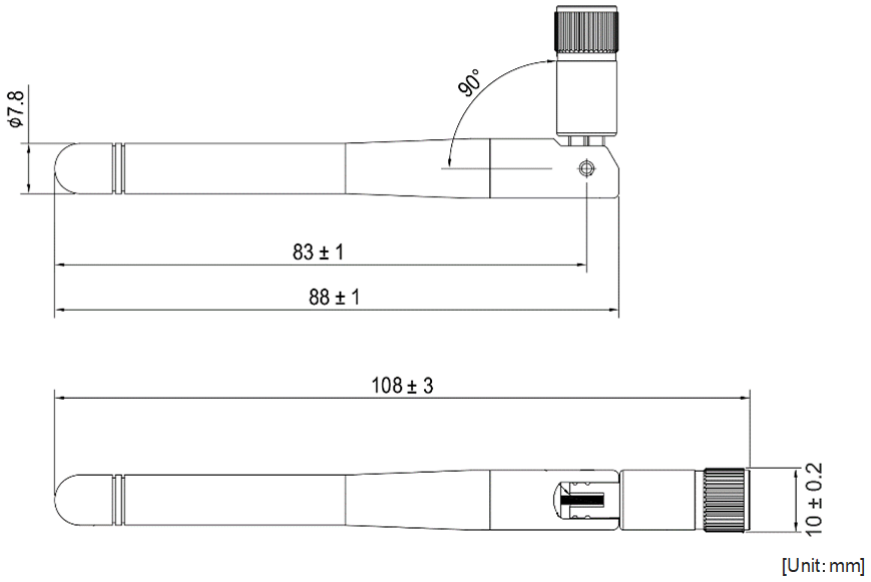


図 6.244 無線 LAN/BT 用外付けアンテナ形状図

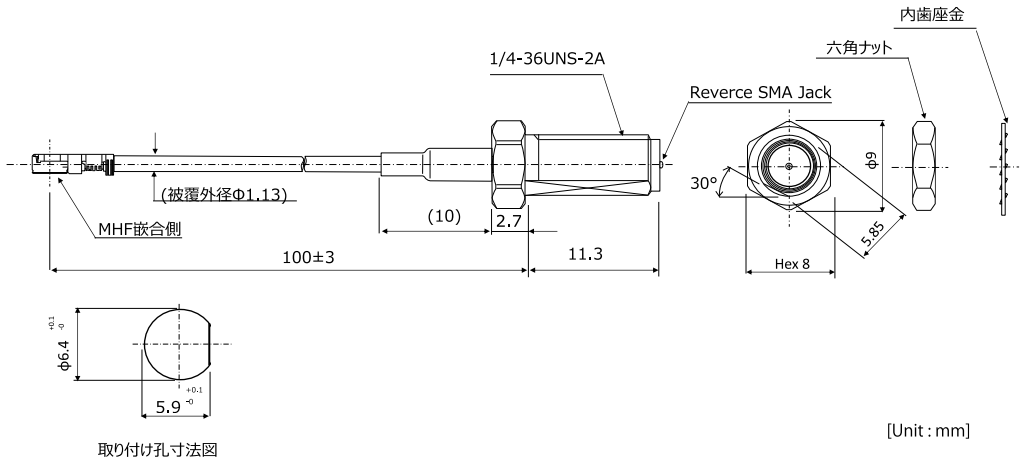


図 6.245 無線 LAN/BT 用アンテナケーブル形状図

6.30.4. Armadillo-X2、G4 ケースモデル VESA 規格固定用プレート

6.30.4.1. 概要

Armadillo-X2、G4 ケースモデル VESA 規格固定用プレートは VESA 規格(100 × 100mm)に対応したテレビやモニターなどに Armadillo-X2 および Armadillo-IoT ゲートウェイ G4 のケースモデルを取り付けるための製品です。

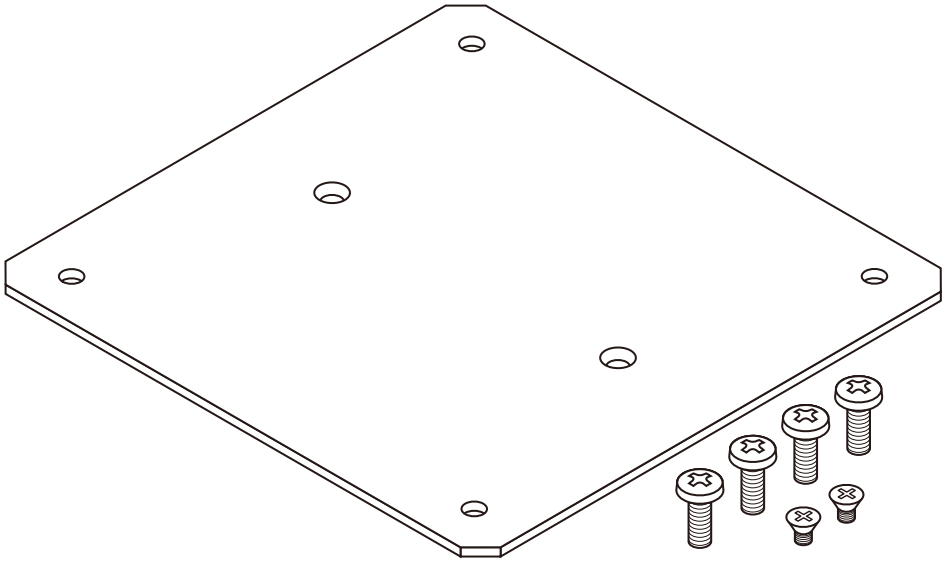


図 6.246 Armadillo-X2、G4 ケースモデル VESA 規格固定用プレート

表 6.30 Armadillo-X2、G4 ケースモデル VESA 規格固定用プレートについて

製品名	Armadillo-X2、G4 ケースモデル VESA 規格固定用プレート
型番	OP-CASEX2-VESA-00
セット内容	VESA 規格固定用プレート、ケース固定用ねじ×2、プレート固定用ねじ×4

表 6.31 Armadillo-X2、G4 ケースモデル VESA 規格固定用プレートの仕様

色	黒
材質	鉄
寸法	120 × 123 mm
板厚	2 mm

6.30.4.2. 組み立て

オプションケース(金属製)の底面には、ケース固定用の M3 のネジ穴が 2 箇所あります。この穴を利用して、VESA 規格固定用プレートを取り付けます。

VESA 規格固定用プレートの中央側にある 2 箇所の穴が、ケース取り付け用の穴です。ケース底面の穴と VESA 規格固定用プレートの穴位置を合わせて、皿もみ加工されている面から、ねじ頭がすっぽり収まるまで、ねじを締めてください。推奨のねじ締めトルクは 31.5cN·m です。

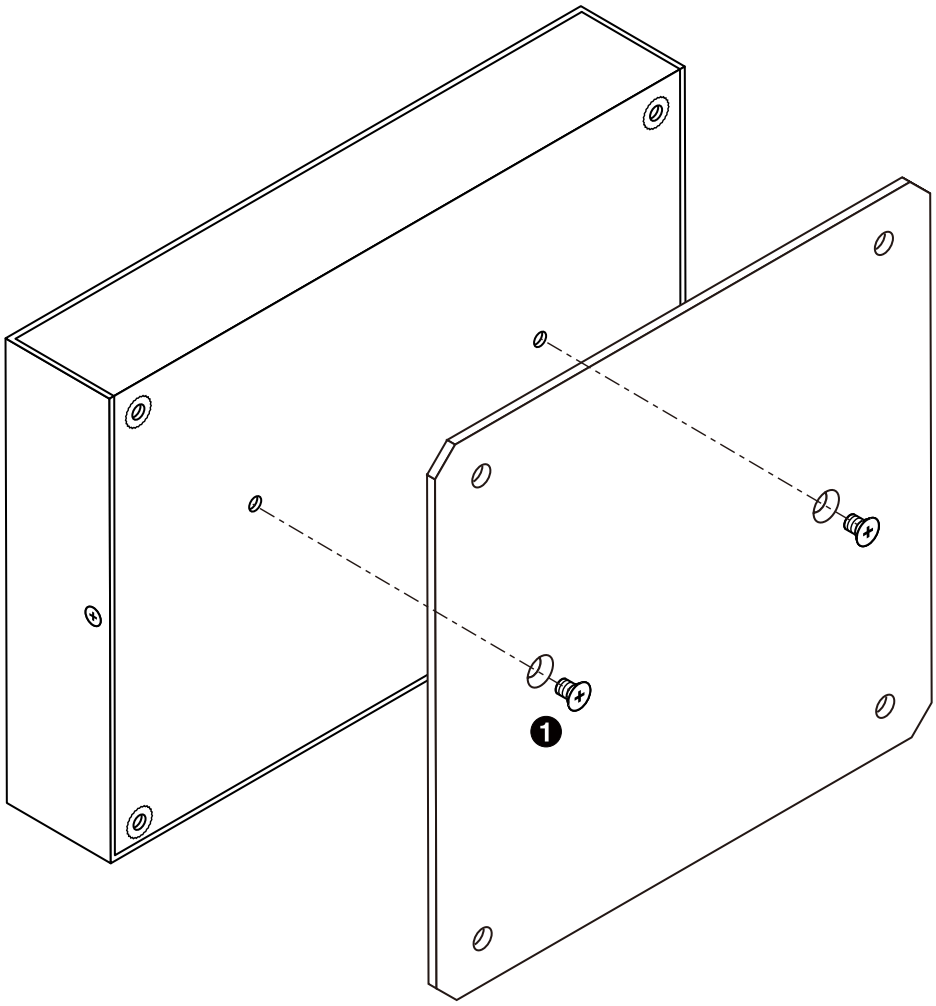


図 6.247 ケースに VESA 規格固定用プレートを取り付け

❶ 皿小ねじ(M3、L=4mm) × 2

> 故障の原因となりますので、付属ねじ以外をご使用の場合、ケース内部の基板や部品にねじが接触していないか、十分にご確認ください。

VESA 規格固定用プレートの 4 隅の穴が、VESA 規格(100 x 100mm)に対応した穴です。ケースと重なるため、固定に使用できる穴は 2 箇所となります。

ケース取り付け済みの VESA 規格固定用プレートを VESA 規格(100 x 100mm)に対応したテレビやモニターなどに取り付けます。

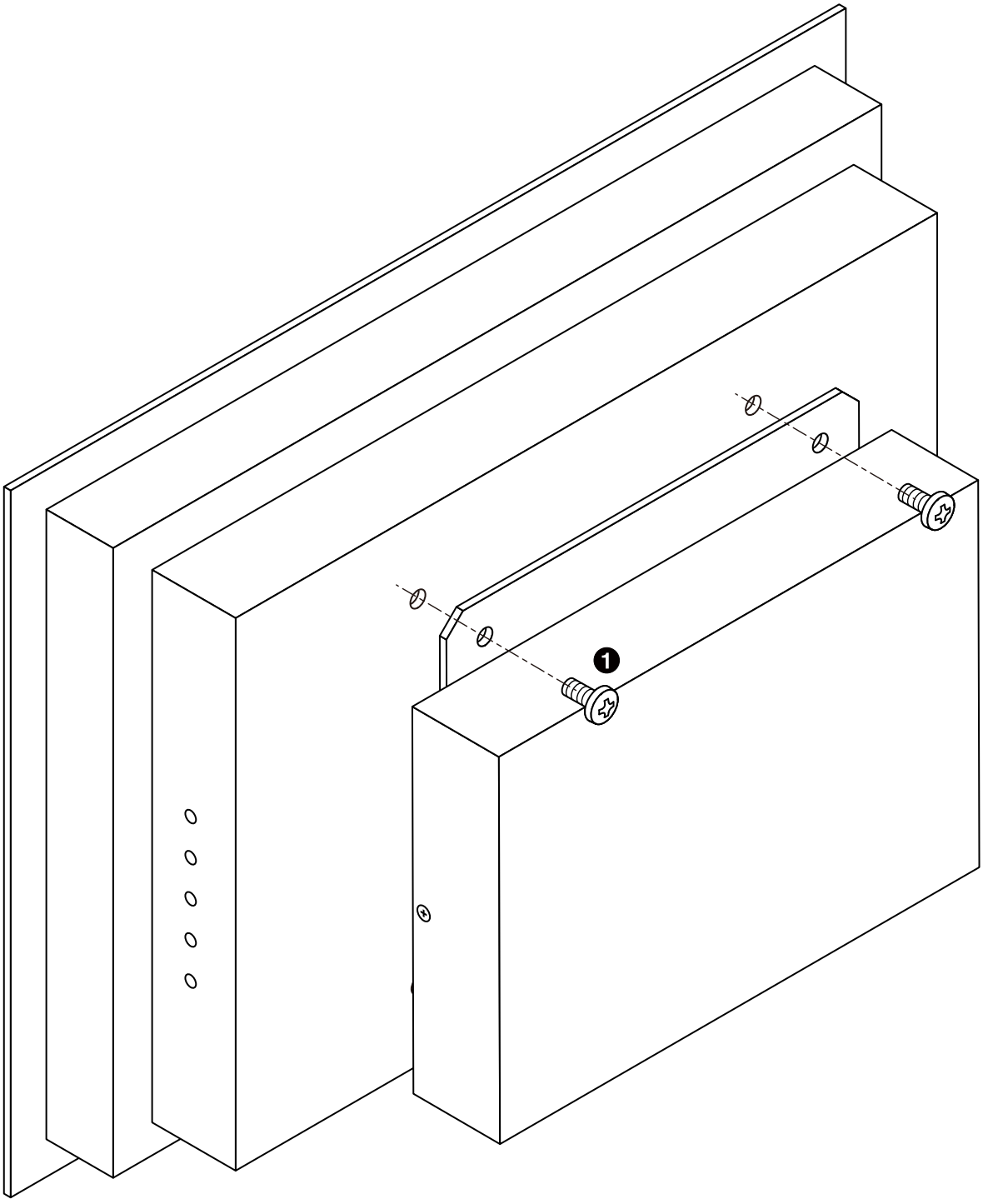


図 6.248 モニターに VESA 規格固定用プレートを取り付け

❶ バインド小ねじ(M4、L=10mm) × 2

6.30.4.3. 形状図

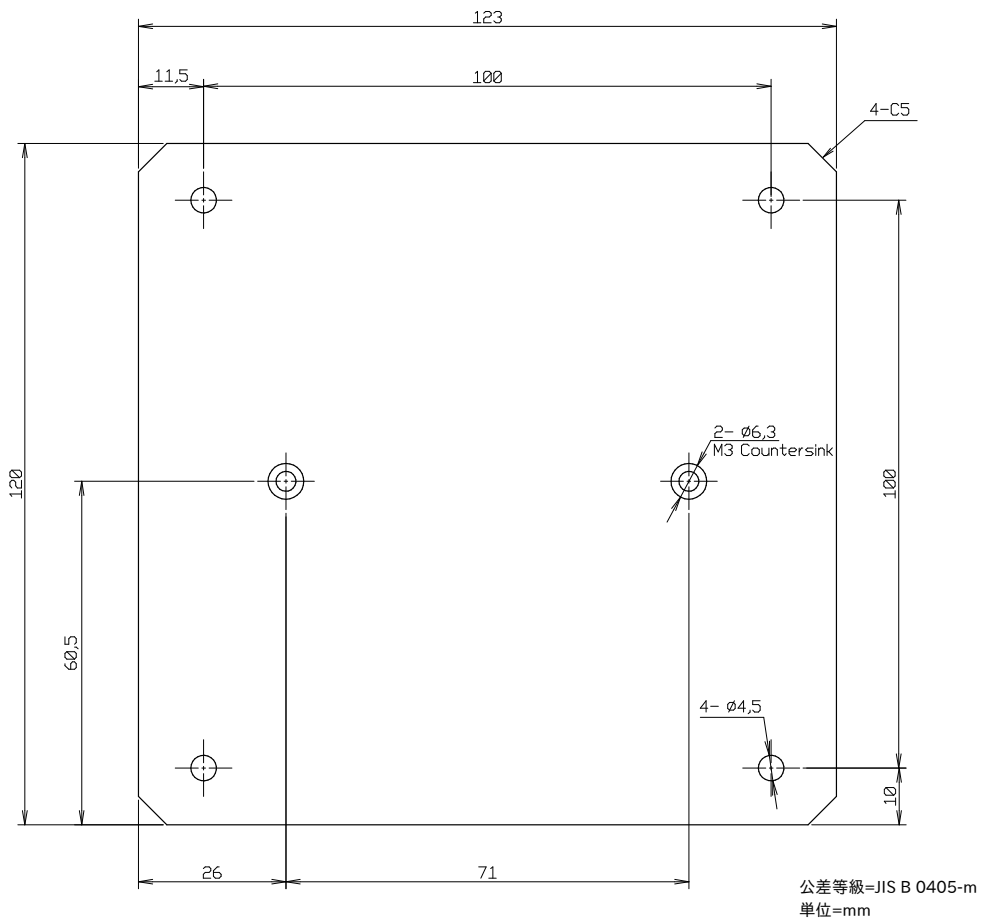


図 6.249 Armadillo-X2、G4 ケースモデル VESA 規格固定用プレート形状図

DXF 形式の形状図を「アットマークテクノ Armadillo サイト」から「購入者向けの限定公開データ」としてダウンロード可能です。

Armadillo-IoT ゲートウェイ G4 製品マニュアル

改訂履歴

バージョン	年月日	改訂内容
1.0.0	2021/12/10	・初版発行
1.0.1	2021/12/10	・ソフトウェアのダウンロードリンクを修正
1.1.0	2021/12/22	・「図 3.4. GNOME 端末の起動」 の画像を修正 ・「図 3.5. GNOME 端末のウィンドウ」 の画像を修正 ・「図 3.39. at-dtweb の起動開始」 の画像を修正 ・「3.3.3.5. 終了方法」 に電源再投入に関する注意を追記 ・「6.1. persist_file について」 を追加 ・「6.2.2.1. イメージからコンテナを作成する」 に詳細な説明を追加 ・「6.2.2.15. リモートリポジトリにコンテナを送信する」 の説明文を修正 ・「インストールの確認：初期化」 の説明文を修正 ・「アプリケーションのアップデート」 の説明文を修正 ・「6.10. VPU や NPU を使用する」のビルドツールのインストール手順を修正 ・「6.2.2.7. コンテナの自動作成やアップデート」 の説明文を修正 ・「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」 を追加 ・「6.2.2.13. network の作成」 を追加 ・「6.5. swupdate_preserve_files について」 を追加 ・「3.4.8.9. 外部からの電源制御」 に電源再投入に関する注意を追記 ・「3.6.21. 電源を入力する」 に電源再投入に関する注意を追記 ・誤記および分かりにくい表記の修正
1.2.0	2022/01/27	・「6.2.9.1. Wayland を扱う」 に weston.ini の設定例を追加 ・「6.2.13. NPU を扱う」 に PID に関する注意事項を追加 ・「6.2.4. コンテナ起動設定ファイルを作成する」 の readonly パラメータの説明を修正 ・「6.25.2. Linux カーネルをビルドする」 にカーネルコンフィギュレーションの変更方法を追加 ・「3.2.5.1. 初期化インストールディスクの作成」 にカスタマイズしたインストールディスクイメージの作成方法を追加 ・「5.4.1. SWU イメージの作成」 の swupdate-mkimage の使用例を mkswu の使用例に修正 ・「3.5. Device Tree をカスタマイズする」 に 「3.5.4.2. 信号名の確認」 を追加 ・「3.6.9. HDMI を使用する」 に HDMI の映像出力の信号を停止する方法を追加 ・「6.19.7. 温度センサーの仕様」 のサーマルシャットダウンの温度を修正 ・「3.6.20. RTC を使用する」に電池を装着する際の注意事項を追加 ・「3.4. ハードウェアの設計」 に 「3.4.1. 信頼性試験データについて」 と 「3.4.4. 放熱」 を追加 ・「6.30.1. オプションケース(金属製)」 にオプションケースへのネジの締付けトルクを追加 ・誤記および分かりにくい表記の修正
1.3.0	2022/02/24	・「3.6.13. GPIO を制御する」 に GPIO の制御に関する説明を追加 ・「3.6.3. USB デバイスを使用する」 に USB ホットプラグに関する説明を追加 ・「6.2.9.1. Wayland を扱う」の weston.conf ファイルの設定例を修正 ・「6.2.9.1. Wayland を扱う」 に at-weston-launch コマンドに関する説明を追加

Armadillo-IoT ゲートウェイ G4 製品マニュアル

		・「6.2.13. NPU を扱う」 の PID に関する注意事項を削除 ・「6.2. コンテナ」 の podman_start の設定に関する説明を修正 ・「5.6.3. hawkBit サーバーから複数の Armadillo に配信する」 に hawkBit サーバー運用に関する手順を追加 ・「3.5. Device Tree をカスタマイズする」 に DTS overlays に関する説明を追加 ・「3.6.13. GPIO を制御する」 を追加 ・HTML 版の検索機能を修正 ・誤記および分かりにくい表記の修正
1.4.0	2022/03/30	・「表 2.2. 仕様」 の eMMC の容量に関する記載を修正 ・「表 2.2. 仕様」 の SD に関する記載の一部を削除 ・「表 2.2. 仕様」 の消費電力(参考値)の値を修正 ・「6.2.9.1. Wayland を扱う」 にスクリーンショットの保存に関する説明を追加 ・「6.2.13. NPU を扱う」 に 「6.2.13.1. ONNX Runtime を使う」 を追加 ・「6.2.13. NPU を扱う」に「6.2.13.2. TensorFlow Lite を使う」 を追加 ・「6.2.13. NPU を扱う」 に 「6.2.13.3. Arm NN を使う」 を追加 ・「6.2.4. コンテナ起動設定ファイルを作成する」 のボリュームマウントの説明に shared と slave オプションに関する説明を追加 ・「6.20. Armadillo Base OS をアップデートする」 を追加 ・「3.5.5. DT overlay によるカスタマイズ」 に 「3.5.5.2. カスタマイズした DT overlay の作成」 を追加 ・「6.12. デモアプリケーションを実行する」 に 「6.12.8. pose estimation demo」 を追加 ・「6.12. デモアプリケーションを実行する」 に 「6.12.9. image segmentation demo」 を追加 ・「3.6.23. ユーザースイッチを使用する」 の説明文を修正 ・誤記および分かりにくい表記の修正
1.5.0	2022/04/08	・表紙に対応型番 「AGX4520-C02D0」「AGX4520-C02Z」を追加 ・「1.2.9. 無線モジュールの安全規制について」 追加 ・「2.2. 製品ラインアップ」 に 「AGX4520-C02D0」「AGX4520-C02Z」を追加 ・「2.2.1. Armadillo-IoT ゲートウェイ G4 開発セット」 に Armadillo-IoT ゲートウェイ G4 LTE モデル開発セットのセット内容を追加 ・「2.3. 仕様」 に 「AGX4520-C02D0」「AGX4520-C02Z」を追加 ・「3.3.1. 準備するもの」に「nanoSIM(UIM カード)と APN 情報」を追加 ・「2.4. インターフェースレイアウト」 に LTE モデルのインターフェースレイアウトを追加 ・「3.3.3.1. Armadillo と開発用 PC を接続」 に Windows PC での USB コンソールインターフェースに関する注記を追加 ・「3.3.3.1. Armadillo と開発用 PC を接続」に 3G/LTE 用アンテナおよび nanoSIM カード接続例追加 ・「6.15.7. 3G/LTE」 追加 ・「6.15.4.3. コネクションの作成」 に設定ファイルのコピー方法に関する注意点を追加 ・「表 6.7. ライブラリイメージ書き込み済みの製品」 に「AGX4520-C02D0」「AGX4520-C02Z」を追加 ・「6.13. SMS を利用する」 追加

Armadillo-IoT ゲートウェイ G4 製品マニュアル

		・「3.6.25. LTE を使用する」 追加 ・「3.4.8. 電気的仕様」 に LTE モデルの値を追加 ・「図 3.33. 電源シーケンス」 を更新 ・「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」 に LTE の内容を追加 ・「表 3.33. CON10 信号配列」 の 28,29 ピンの電圧グループを VDD_3V3 から VEXT_3V3 に変更 ・「6.30.1.3. 形状図」 LTE モデルの形状図追加 ・「6.30.2. 3G/LTE 用外付けアンテナ、アンテナケーブル」 追加 ・誤記修正
1.6.0	2022/04/27	・「1.2.5. 電波障害について」 の文章を修正 ・「6.10.3. ライブラリイメージのバージョンを確認する」 追加 ・「6.2.2.9. イメージを削除する」 に Read-Only のイメージの削除に関する補足説明を追加 ・「6.2.9.1. Wayland を扱う」 に解像度に関する補足説明を追加 ・「6.2.13.2. TensorFlow Lite を使う」 にライブラリのバージョンに関する注意点を追加 ・「6.2.2.16. イメージを eMMC に保存する」 追加 ・「3.5.3. at-dtweb の起動」 にバージョンに関する注意点を追加 ・「3.5.4.5. Device Tree のファイルの生成」にバージョン別の説明を追加 ・「図 3.91. AC アダプタの極性マーク」 を変更 ・誤記および分かりにくい表記の修正
1.7.0	2022/05/27	・「6.2.4. コンテナ起動設定ファイルを作成する」 に --rootfs オプションに関する説明を追加 ・「4.4.4. 開発したシステムをインストールディスクにする」 追加 ・「6.6. SWU イメージの内容の確認」 追加 ・「6.13.5. SMS の内容を表示する」に記載されている SMS の最大保存件数を 255 に修正 ・誤記および分かりにくい表記の修正
1.8.0	2022/06/28	・「1.2.2. 取扱い上の注意事項」 に「設置時の注意事項」、「発熱に関する注意事項」、「電気通信事業法に関する注意事項について」を追加 ・「3.3.3.4. ログイン」 に「atmark」アカウントのロックに関する説明を追加 ・セキュリティガイドの分冊化に伴い 「11. セキュリティ」 を削除 ・誤記および分かりにくい表記の修正
1.9.0	2022/07/27	・at-debian-image からコンテナを作成する例において、 device として渡す tty の番号を 1 から 7 に修正 ・「6.2.2.6. コンテナの変更を保存する」 に podman run コマンドの --volume オプションに関する説明を追加 ・「6.2.2.6. コンテナの変更を保存する」 に Copy-on-Write (CoW) に関する注意を追加 ・「6.2.2.16. イメージを eMMC に保存する」 に abos-ctrl podman-rw コマンドと abos-ctrl podman-storage コマンドに関する説明を追加 ・「6.12. デモアプリケーションを実行する」 に 「6.12.10. super resolution demo」 を追加 ・「6.12. デモアプリケーションを実行する」 に 「6.12.11. hand estimation demo」 を追加 ・誤記および分かりにくい表記の修正
1.10.0	2022/08/29	・「1.2.3. 製品の保管について」 追加 ・「6.23. u-boot の環境変数の設定」 追加

		・「3.2.3.4. ロールバック（リカバリー)」 にロールバックに関する説明を追加 ・誤記および分かりにくい表記の修正
1.10.1	2022/09/28	・「1.2. 注意事項」 に電池の取り扱いに関する注意を追加 ・「6.2.13.3. Arm NN を使う」 に動作非対応に関する注意を追加
1.11.0	2022/10/26	・本文中で例示している podman run コマンドの使用例を podman_start コマンドを使用したものに変更 ・「3.3.3.5. 終了方法」 に poweroff コマンドに関する説明を追加 ・「6.2.4. コンテナ起動設定ファイルを作成する」 に set_init オプションに関する説明を追加 ・「6.2.2.16. イメージを eMMC に保存する」 の abos-ctrl podman-storage コマンドに関する説明文を更新 ・誤記および分かりにくい表記の修正
1.12.0	2022/11/28	・表紙に対応型番「AGX4520-C03D0」「AGX4520-C03Z」「AGX4500-C03Z」「AGX4500-U03Z」を追加 ・「1.2.2. 取扱い上の注意事項」 に電波に関する注意事項を追加 ・「表 1.6. WLAN+BT コンボモジュール:AW-XM458 適合証明情報」 と 「図 1.3. WLAN+BT コンボモジュール:AW-XM458 認証マーク」 を追加 ・「2.2. 製品ラインアップ」 に「AGX4520-C03D0」「AGX4520-C03Z」「AGX4500-C03Z」「AGX4500-U03Z」を追加 ・「2.2.1. Armadillo-IoT ゲートウェイ G4 開発セット」 に Armadillo-IoT ゲートウェイ G4 LTE+WLAN モデル開発セットのセット内容を追加 ・「2.3. 仕様」 に 「AGX4520-C03D0」「AGX4520-C03Z」「AGX4500-C03Z」「AGX4500-U03Z」を追加 ・「3.3.3.5. 終了方法」 に AC アダプタを抜く際の注意事項を追加 ・「6.15. コマンドラインからネットワーク設定をする」に「6.15.6. 無線 LAN」 を追加 ・「6.15.7. 3G/LTE」 に 「6.15.7.3. MCC/MNC を指定した 3G/LTE のコネクションを作成する」 を追加 ・「6.15. コマンドラインからネットワーク設定をする」 に「BT」を追加 ・「表 6.7. ライブラリイメージ書き込み済みの製品」に「AGX4520-C03D0」「AGX4520-C03Z」「AGX4500-C03Z」「AGX4500-U03Z」を追加 ・「6.12. デモアプリケーションを実行する」 に 「6.12.12. screw detection demo」 を追加 ・「6.12. デモアプリケーションを実行する」 の機械学習デモの起動時間に関する説明を修正 ・「6.16. コマンドラインから無線 LAN でネットワークを構成する」を追加 ・「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」 に「3.6.5. WLAN を使用する」 を追加 ・「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」 に「3.6.6. BT デバイスを使用する」 を追加 ・「3.6.5. WLAN を使用する」 に LTE+WLAN モデル、WLAN モデルに関する説明を追加 ・「表 6.29. Armadillo-IoT ゲートウェイ G4 関連のオプション品」に 「無線 LAN/BT 用外付けアンテナ」、「アンテナケーブル」を追加 ・「6.30. オプション品」 に「図 4.19. オプションケース(金属製) 放熱シート貼付(WLAN+BT コンボモジュール搭載モデル)」 を追加

		・「6.30. オプション品」 に 「6.30.3. 無線 LAN/BT 用外付けアンテナ、アンテナケーブル」 を追加 ・誤記および分かりにくい表記の修正
1.13.0	2022/12/26	・「6.19. 動作中の Armadillo の温度を測定する」 を追加 ・「図 3.55. Armadillo-IoT ゲートウェイ G4 のインターフェース」 の USB インターフェースの型番とメーカー名を修正 ・「3.4.9.1. 基板形状図」 を修正 ・誤記および分かりにくい表記の修正
1.14.0	2023/02/02	・「3.13. GUI アプリケーションの開発」 を追加 ・「6.2.10.2. 起床要因を有効化する」 に RV-8803-C7 のアラーム割り込みに関する注記を追加 ・「6.18. ボタンやキーを扱う」 の buttond に関する記述を修正 ・「3.6.1. SD カードを使用する」 の SD カードのクロック速度を修正 ・「3.6.20. RTC を使用する」 に RV-8803-C7 のアラーム割り込みに関する注記を追加 ・「図 6.244. 無線 LAN/BT 用外付けアンテナ形状図」 を修正 ・誤記および分かりにくい表記の修正
1.15.0	2023/02/10	・「6.30.4. Armadillo-X2、G4 ケースモデル VESA 規格固定用プレート」 を追加 ・「表 6.29. Armadillo-IoT ゲートウェイ G4 関連のオプション品」 に CPU 放熱シートについての内容追加 ・表記ゆれ修正
1.15.1	2023/02/27	・「6.15.7.7. 3G/LTE の接続を確認する」に記載の LTE の接続確認方法を修正 ・「3.13. GUI アプリケーションの開発」 に記載のダウンロードするサンプルアプリケーションのファイル名を修正 ・「6.25.2. Linux カーネルをビルドする」 に記載のビルドに必要なパッケージに atmark-firmware を追加 ・「図 6.30. network を使うコンテナを自動起動するための設定例」 の set_ports になっていた箇所を add_ports に修正
1.16.0	2023/03/28	・「6.15.2. IP アドレスの確認方法」 を追加 ・「6.15.7.8. 3G/LTE コネクションを切断する」から 3G/LTE のコネクション設定を編集する場合の注意事項を削除 ・「3.13. GUI アプリケーションの開発」 の VSCode を使用した開発方法に関する説明を VSCode エクステンションを使用した開発方法に変更 ・「3.14. CUI アプリケーションの開発」 を追加 ・「3.3.5. VSCode を使用して Armadillo のセットアップを行う」を追加 ・「6.2.4. コンテナ起動設定ファイルを作成する」にコンテナ起動後にホットプラグデバイスを認識させる方法を追加 ・「6.25.1. ブートローダーをビルドする」 に imx-boot のインストール方法を追加 ・「6.25.2. Linux カーネルをビルドする」 にあるカーネルのインストール手順を変更 ・「6.25.3. Alpine Linux ルートファイルシステムをビルドする」にルートファイルシステムのカスタマイズ方法とインストール方法を追加 ・「表 6.20. u-boot の主要な環境変数」 を追加 ・「6.27. Linux カーネルがクラッシュしたときにメモリの状態を保存する」 を追加 ・誤記および分かりにくい表記の修正

Armadillo-IoT ゲートウェイ G4 製品マニュアル

1.17.0	2023/04/26	・「表 2.2. 仕様」 の「カレンダー時計」に平均月差に関する説明を追加 ・「3.13. GUI アプリケーションの開発」 にサイネージアプリケーションに関する説明を追加 ・「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」 を追加 ・「6.25.2. Linux カーネルをビルドする」 にカーネルを SWU ファイルでアップデートする場合の preserve_files に関する注意を追加 ・「6.25.3. Alpine Linux ルートファイルシステムをビルドする」 に --cache オプションに関する説明を追加 ・「6.3. swupdate がエラーする場合の対処」 を追加 ・「6.4.11.3. 例: swupdate_preserve_files で Linux カーネル以外の Armadillo-IoT ゲートウェイ G4 向けのイメージをインストールする方法」 に --del オプションに関する説明を追加 ・「6.16.2.1. hostapd を使用して設定する」 に mlan0 を NetworkManager の管理から外すための手順を追加 ・誤記および分かりにくい表記の修正
1.18.0	2023/05/29	・「6.15.7.2. 3G/LTE のコネクションを作成する」 KDDI データ専用 SIM が LTE 接続できない場合の対処方法を追加 ・「6.15.7.9. 3G/LTE 再接続サービス」 に設定ファイルの参考例を使用した設定方法を追加 ・「3.13. GUI アプリケーションの開発」 に Factory Signage アプリケーションを追加 ・「3.14. CUI アプリケーションの開発」 の開発手順に関する説明を修正 ・「3.6.6. BT デバイスを使用する」 から不要な手順を削除 ・「6.18. ボタンやキーを扱う」 の buttond に関するオプションの説明を修正 ・「6.18.3. Armadillo 起動時にのみボタンに反応する方法」 を追加 ・誤記および分かりにくい表記の修正
1.19.0	2023/06/29	・ABOS Web についての記述(「6.8. Web UI から Armadillo をセットアップする (ABOS Web)」、「3.8. ネットワーク設定」)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」 に記載している用意する microSD カードの容量を 10GB 以上に修正 ・「6.30.1.3. 形状図」 を修正 ・誤記および分かりにくい表記の修正
1.20.0	2023/07/26	・「3.8. ネットワーク設定」 に 「3.8.10.3. VPN 設定」 を追加 ・「6.8. Web UI から Armadillo をセットアップする (ABOS Web)」 に 「6.8.3. コンテナ管理」 を追加 ・「6.8. Web UI から Armadillo をセットアップする (ABOS Web)」 に 「6.8.4. SWU インストール」 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
1.21.0	2023/8/29	・文章全体の構成を変更 ・「表 2.2. 仕様」 内の「拡張インターフェース」で「非対応」となっていた箇所を CON12 のインターフェースに対応するように修正 ・「3.6.14. I2C デバイスを使用する」 の最大転送レートを 384kbps に修正 ・「3.8.2. ABOS Web へのアクセス」 に VSCode から ABOS WEB を開く方法を追加

		・「3.8.7.1. WLAN 設定（クライアントとしての設定)」 に複数のアクセスポイントを保存する方法を追加 ・「3.13.3.1. プロジェクトの作成」 にサンプルアプリの起動画面例を追加 ・「3.13.8.1. アプリケーションのビルドモード」 を追加 ・「3.13.8.2. サンプルアプリケーションのビルド」 に Debug モードの場合と Release モードの場合の実行方法を追加 ・「3.13.9.1. 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 ・「3.13.9.2. アプリケーションの実行」 に Debug モードの場合と Release モードの場合の実行方法を追加 ・「3.14.7.2. ssh 接続に使用する IP アドレスの設定」 に VSCode から IP アドレスを設定する方法を追加 ・誤記修正
2.0.0	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)
2.0.1	2023/09/27	・「表 2.2. 仕様」 内の拡張インターフェース仕様を修正 ・「表 2.3. 仕様(AGX4500-C03*, AGX4520-C03*)」 内の拡張インターフェース仕様を修正 ・「表 3.1. 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)」 内の説明文を修正 ・「3.3.4.3. クロスコンパイル用ライブラリをインストールする」 に手順に関する補足説明を追記 ・「図 3.70. Bluetooth を起動する実行例」 内の手順を修正 ・誤記修正
2.1.0	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 ・「3.3.5.1. initial_setup.swu の作成」 の説明文を修正 ・「3.6.3. USB デバイスを使用する」 に記載しているコンテナの作成例を、add_hotplugs を使用した例に修正 ・「3.13. GUI アプリケーションの開発」 内にある 「3.13.3.2. ディレクトリ構成」 の説明文を修正 ・「3.14. CUI アプリケーションの開発」 内にある 「3.14.3.2. 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 ・「3.10. Network Time Protocol (NTP, ネットワーク・タイム・プロトコル) の設定」に記載している chrony の設定ファイルに関する説明を修正 ・誤記および分かりにくい表記の修正
2.2.0	2023/11/28	・「図 4.14. 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 ・「6.2.4.14. 自動起動の無効化」 の説明文を修正 ・「6.25.4. 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
2.3.0	2023/12/26	・「3.2.5. 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 ・「5.2.8. 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 ・「6.8.6. アプリケーション向けのインターフェース (Rest API)」を追加 ・「6.25.3. 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 ・誤記および分かりにくい表記の修正
2.4.0	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 ・「3.14.3.2. ディレクトリ構成」 に requirements.txt に関する説明を追記 ・「3.15. C 言語によるアプリケーションの開発」 を追加 ・「6.2.10.2. 起床要因を有効化する」 内にある USB デバイスのアドレスを修正 ・「6.8.6.1. 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
2.5.0	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. VSCode を使用して生成する」を追記
2.6.0	2024/02/27	・Armadillo Twin に関する情報を追加 ・「3.2.3.5. 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 ・「3.6.12.1. 動作確認済み MIPI CSI-2 カメラデバイス」 を追加 ・「3.12. ABOSDE によるアプリケーションの開発」 を追加 ・「3.14. CUI アプリケーションの開発」 に 「3.14.4. コンテナのディストリビューション」 を追加 ・「3.15. C 言語によるアプリケーションの開発」 に 「3.15.4. コンテナのディストリビューション」 を追加 ・「5.3. ABOSDE で開発したアプリケーションをアップデートする」 を追加 ・「6.9. ABOSDE から ABOS Web の機能を使用する」 を追加 ・「6.15.7.1. 3G/LTE データ通信設定を行う前に」 にある APN の説明に設定可能最大長を追記 ・誤記および分かりにくい表記の修正
2.7.0	2024/03/26	・「3.8.3. ABOS Web のパスワード登録」 にパスワード変更に関する補足説明を追加 ・「3.9. ABOS Web をカスタマイズする」 を追加 ・「6.8.5. 時刻設定」 を追加 ・「6.8.6.8. Rest API : 時刻の設定」 を追加
2.8.0	2024/04/04	・「3.13. GUI アプリケーションの開発」 に「3.13.6. コンテナ内のファイル一覧表示」 を追加 ・「3.14. CUI アプリケーションの開発」 に「3.14.6. コンテナ内のファイル一覧表示」 を追加 ・「3.15. C 言語による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

Armadillo-IoT ゲートウェイ G4 製品マニュアル
Version 2.8.0
2024/04/04