

Armadillo-X2 製品マニュアル

AX2210-U00D0
AX2210-U00Z
AX2210-C00Z

Version 1.3.0
2023/04/26

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-X2 製品マニュアル

株式会社アットマークテクノ

製作著作 © 2023 Atmark Techno, Inc.

Version 1.3.0
2023/04/26

目次

1. はじめに	18
1.1. 本書で扱うこと扱わないこと	18
1.1.1. 扱うこと	18
1.1.2. 扱わないこと	18
1.2. 本書で必要となる知識と想定する読者	19
1.3. ユーザー限定コンテンツ	19
1.4. 本書および関連ファイルのバージョンについて	19
1.5. 本書の構成	19
1.6. 表記について	20
1.6.1. フォント	20
1.6.2. コマンド入力例	20
1.6.3. アイコン	21
1.7. 謝辞	21
2. 注意事項	22
2.1. 安全に関する注意事項	22
2.2. 取扱い上の注意事項	23
2.3. 製品の保管について	24
2.4. ソフトウェア使用に関する注意事項	24
2.5. 電波障害について	25
2.6. 保証について	25
2.7. 輸出について	25
2.8. 商標について	25
3. 製品概要	27
3.1. 製品の特長	27
3.1.1. Armadillo とは	27
3.1.2. Armadillo-X2 とは	27
3.1.3. Armadillo Base OS とは	30
3.2. 製品ラインアップ	32
3.2.1. Armadillo-X2 開発セット	32
3.2.2. Armadillo-X2 量産ボード	32
3.3. 仕様	33
3.4. ブロック図	33
3.5. ストレージデバイスのパーティション構成	34
4. Armadillo の電源を入れる前に	36
4.1. 準備するもの	36
4.2. 開発/動作確認環境の構築	36
4.2.1. ATDE のセットアップ	37
4.2.2. 取り外し可能デバイスの使用	42
4.2.3. コマンドライン端末(GNOME 端末)の起動	43
4.2.4. シリアル通信ソフトウェア(minicom)の使用	44
4.3. インターフェースレイアウト	47
4.4. 接続方法	48
4.5. ジャンパピンの設定について	51
4.6. vi エディタの使用方法	52
4.6.1. vi の起動	52
4.6.2. 文字の入力	53
4.6.3. カーソルの移動	53
4.6.4. 文字の削除	54
4.6.5. 保存と終了	54
5. 起動と終了	55

5.1. 起動	55
5.2. ログイン	57
5.3. 終了方法	59
6. ユーザー登録	61
6.1. 購入製品登録	61
7. 動作確認方法	62
7.1. ネットワーク	62
7.1.1. 接続可能なネットワーク	62
7.1.2. IP アドレスの確認方法	62
7.1.3. ネットワークの設定方法	63
7.1.4. nmcli の基本的な使い方	63
7.1.5. 有線 LAN	67
7.2. ストレージ	67
7.2.1. ストレージの使用方法	68
7.2.2. ストレージのパーティション変更とフォーマット	69
7.3. LED	70
7.3.1. LED を点灯/消灯する	70
7.3.2. トリガを使用する	71
7.4. ユーザースイッチ	72
7.4.1. イベントを確認する	72
8. 開発の基本的な流れ	74
8.1. アプリケーション開発の流れ	74
8.1.1. Armadillo への接続	74
8.1.2. overlaysfs の扱い	74
8.1.3. Podman のデータを eMMC に保存する	74
8.1.4. ベースとなるコンテナを取得する	75
8.1.5. デバイスのアクセス権を与える	75
8.1.6. アプリケーションを作成する	76
8.1.7. コンテナやデータを保存する	76
8.2. アプリケーションコンテナの運用	76
8.2.1. アプリケーションの自動起動	76
8.2.2. アプリケーションの送信	76
8.2.3. インストール確認：初期化	76
8.2.4. アプリケーションのアップデート	77
8.3. VPU や NPU を使用する	77
8.3.1. ATDE にクロスコンパイル用ライブラリをインストールする	77
8.3.2. Armadillo へ書き込むためのライブラリイメージを作成する	78
8.3.3. Armadillo にライブラリイメージを書き込む	78
8.3.4. ライブラリイメージのバージョンを確認する	79
8.3.5. コンテナ内からライブラリを使用するための準備	79
9. Howto	81
9.1. GUI アプリケーションを開発する	81
9.1.1. Flutter とは	81
9.1.2. Flutter を用いた開発の流れ	81
9.1.3. ATDE 上でのセットアップ	82
9.1.4. Armadillo 上でのセットアップ	88
9.1.5. アプリケーション開発	88
9.1.6. 動作確認	91
9.1.7. リリース版のビルド	93
9.1.8. 製品への書き込み	94
9.2. CUI アプリケーションを開発する	95
9.2.1. CUI アプリケーション開発の流れ	95
9.2.2. ATDE 上でのセットアップ	96

- 9.2.3. Armadillo 上でのセットアップ 101
- 9.2.4. アプリケーション開発 101
- 9.2.5. リリース版のビルド 103
- 9.2.6. 製品への書き込み 104
- 9.3. VSCode を使用して Armadillo のセットアップを行う 104
 - 9.3.1. ATDE 上でのセットアップ 104
 - 9.3.2. initial_setup.swu の作成 107
 - 9.3.3. LAN の設定 107
 - 9.3.4. コンテナイメージインストール用の SWU ファイルの作成 109
- 9.4. アプリケーションをコンテナで実行する 111
 - 9.4.1. Podman - コンテナ仮想化ソフトウェア 111
 - 9.4.2. コンテナを操作する 111
 - 9.4.3. アットマークテクノが提供するイメージを使う 122
 - 9.4.4. 入出力デバイスを扱う 123
 - 9.4.5. 近距離通信を行う 133
 - 9.4.6. ネットワークを扱う 135
 - 9.4.7. サーバを構築する 137
 - 9.4.8. 画面表示を行う 140
 - 9.4.9. パワーマネジメント機能を使う 148
 - 9.4.10. コンテナからの poweroff か reboot 151
 - 9.4.11. 異常検知 151
 - 9.4.12. NPU を扱う 152
- 9.5. コンテナの運用 158
 - 9.5.1. コンテナの自動起動 158
 - 9.5.2. pod の作成 163
 - 9.5.3. network の作成 164
 - 9.5.4. コンテナからのコンテナ管理 164
 - 9.5.5. コンテナの配布 165
- 9.6. マルチメディアデータを扱う 169
 - 9.6.1. GStreamer - マルチメディアフレームワーク 169
 - 9.6.2. GStreamer 実行用コンテナを作成する 169
 - 9.6.3. GStreamer パイプラインの実行例 170
 - 9.6.4. 動画を再生する 171
 - 9.6.5. ストリーミングデータを再生する 172
 - 9.6.6. USB カメラからの映像を表示する 173
 - 9.6.7. USB カメラからの映像を録画する 173
 - 9.6.8. Video Processing Unit(VPU) 174
- 9.7. Armadillo のソフトウェアをビルドする 176
 - 9.7.1. ブートローダーをビルドする 176
 - 9.7.2. Linux カーネルをビルドする 178
 - 9.7.3. Alpine Linux ルートファイルシステムをビルドする 181
- 9.8. SD ブートの活用 184
 - 9.8.1. ブートディスクの作成 184
 - 9.8.2. SD ブートの実行 186
- 9.9. Armadillo のソフトウェアの初期化 187
 - 9.9.1. インストールディスクの作成 188
 - 9.9.2. インストールディスクを使用する 190
- 9.10. Armadillo のソフトウェアをアップデートする 190
 - 9.10.1. SWU イメージとは 190
 - 9.10.2. SWU イメージの作成 191
 - 9.10.3. イメージのインストール 193
 - 9.10.4. swupdate がエラーする場合の対処 195
 - 9.10.5. hawkBit サーバーから複数の Armadillo に配信する 196

9.10.6. mkswu の desc ファイル	210
9.10.7. swupdate_preserve_files について	215
9.10.8. SWU イメージの内容の確認	216
9.10.9. SWUpdate と暗号化について	216
9.11. Armadillo Base OS の操作	216
9.11.1. アップデート	216
9.11.2. overlaysfs と persist_file について	217
9.11.3. ロールバック状態の確認	219
9.11.4. ボタンやキーを扱う	219
9.11.5. Armadillo Base OS 側の起動スクリプト	221
9.11.6. u-boot の環境変数の設定	222
9.11.7. Network Time Protocol (NTP, ネットワーク・タイム・プロトコル)	224
9.12. Device Tree をカスタマイズする	224
9.12.1. at-dtweb のインストール	225
9.12.2. at-dtweb の起動	225
9.12.3. Device Tree をカスタマイズ	227
9.12.4. DT overlay によるカスタマイズ	232
9.13. eMMC のデータリテンション	234
9.13.1. データリテンションの設定	235
9.13.2. 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	237
9.13.3. 実装仕様に関する技術情報	238
9.14. デモアプリケーションを実行する	239
9.14.1. コンテナを作成する	239
9.14.2. デモアプリケーションランチャを起動する	240
9.14.3. mediaplayer	241
9.14.4. video recoder	242
9.14.5. led switch tester	243
9.14.6. rtc tester	243
9.14.7. object detection demo	244
9.14.8. pose estimation demo	245
9.14.9. image segmentation demo	246
9.14.10. super resolution demo	247
9.14.11. hand estimation demo	248
9.14.12. screw detection demo	249
9.15. 動作中の Armadillo の温度を測定する	250
9.15.1. 温度測定的重要性	250
9.15.2. atmark-thermal-profiler をインストールする	250
9.15.3. atmark-thermal-profiler を実行・停止する	251
9.15.4. atmark-thermal-profiler が出力するログファイルを確認する	251
9.15.5. 温度測定結果の分析	252
9.16. 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	253
9.16.1. Kdump を利用する準備	253
9.16.2. Kdump の動作確認	255
9.16.3. vmcore の確認	256
10. 動作ログ	259
10.1. 動作ログについて	259
10.2. 動作ログを取り出す	259
10.3. ログファイルのフォーマット	259
10.4. ログ用パーティションについて	259
11. 製品機能	260
11.1. SD ホスト	260
11.2. Ethernet	261
11.3. USB ホスト	261

11.4. USB ハブ	262
11.5. UART	263
11.6. HDMI	263
11.7. MIPI CSI-2	265
11.8. リアルタイムクロック	266
11.9. ユーザースイッチとイベント信号	267
11.10. LED	268
11.11. I2C	268
11.12. GPIO	269
11.13. 温度センサー	270
11.14. ウォッチドッグタイマー	270
11.15. パワーマネジメント	271
12. ソフトウェア仕様	273
12.1. SWUpdate	273
12.1.1. SWUpdate とは	273
12.1.2. swu パッケージ	273
12.1.3. A/B アップデート(アップデートの 2 面化)	273
12.1.4. ロールバック (リカバリー)	274
12.2. hawkBit	274
12.2.1. hawkBit とは	274
12.2.2. データ構造	274
13. ハードウェア仕様	276
13.1. 電氣的仕様	276
13.1.1. 絶対最大定格	276
13.1.2. 推奨動作条件	276
13.1.3. 電源出力仕様	276
13.1.4. 入出力インターフェースの電氣的仕様	277
13.1.5. 電源回路の構成	277
13.1.6. 電源シーケンス	279
13.1.7. リセット回路の構成	279
13.1.8. リセットシーケンス	280
13.1.9. 外部からの電源制御	280
13.2. インターフェース仕様	281
13.2.1. CON1 (SD インターフェース)	283
13.2.2. CON3 (LAN インターフェース)	284
13.2.3. CON4 (USB インターフェース 1)	284
13.2.4. CON6 (USB コンソールインターフェース)	285
13.2.5. CON8 (HDMI インターフェース)	285
13.2.6. CON10 (MIPI-CSI インターフェース)	286
13.2.7. CON11 (拡張インターフェース)	287
13.2.8. CON13(RTC バックアップインターフェース)	290
13.2.9. CON14、CON15 (電源入力インターフェース)	291
13.2.10. CON16 (3.3V 電源出力インターフェース)	292
13.2.11. CON17 (USB インターフェース 1)	292
13.2.12. JP1 (起動デバイス設定ジャンパ)	293
13.2.13. SW1 (ユーザースイッチ)	293
13.2.14. LED3(ユーザー LED)	293
13.2.15. LED4 (電源 LED)	293
13.3. 形状図	294
13.3.1. 基板形状図	294
13.4. 設計情報	295
13.4.1. 信頼性試験データについて	295
13.4.2. 放射ノイズ	295

13.4.3. ESD/雷サージ	295
13.4.4. 放熱	295
13.4.5. 拡張ボードの設計	296
13.4.6. 回路設計	298
13.5. オプション品	300
13.5.1. Armadillo-X2 オプションケース(金属製)	301
13.5.2. Armadillo-X2、G4 ケースモデル VESA 規格固定用プレート	307

目次

3.1. Armadillo-X2 とは	28
3.2. Flutter を用いた GUI アプリケーション開発例	28
3.3. エッジ AI 処理、機械学習の例	29
3.4. SoC の発熱をアルミケースに直接放熱	29
3.5. Armadillo Base OS とは	30
3.6. コンテナによるアプリケーションの運用	31
3.7. ロールバックの仕組み	31
3.8. Armadillo-X2 の外観	32
3.9. ブロック図	34
4.1. GNOME 端末の起動	43
4.2. GNOME 端末のウィンドウ	44
4.3. minicom の設定の起動	44
4.4. minicom の設定	44
4.5. minicom のシリアルポートの設定	45
4.6. 例. シリアル通信用 USB ケーブル(A-microB)接続時のログ	45
4.7. minicom のシリアルポートのパラメータの設定	46
4.8. minicom シリアルポートの設定値	46
4.9. minicom 起動方法	47
4.10. minicom 終了確認	47
4.11. インターフェースレイアウト	48
4.12. Armadillo-X2 の接続例	49
4.13. COM7 が競合している状態	50
4.14. Bluetooth に割当の COM を変更した状態	51
4.15. JP1 の位置	52
4.16. vi の起動	53
4.17. 入力モードに移行するコマンドの説明	53
4.18. 文字を削除するコマンドの説明	54
7.1. IP アドレスの確認	62
7.2. IP アドレス(eth0)の確認	62
7.3. nmcli のコマンド書式	63
7.4. コネクションの一覧	63
7.5. コネクションの有効化	64
7.6. コネクションの無効化	64
7.7. コネクションの作成	64
7.8. コネクションファイルの永続化	64
7.9. コネクションの削除	65
7.10. コネクションファイル削除時の永続化	65
7.11. 固定 IP アドレス設定	65
7.12. DNS サーバーの指定	65
7.13. DNS サーバーの指定	66
7.14. コネクションの修正の反映	66
7.15. デバイスの一覧	66
7.16. デバイスの接続	66
7.17. デバイスの切断	67
7.18. 有線 LAN の PING 確認	67
7.19. mount コマンド書式	68
7.20. ストレージのマウント	69
7.21. ストレージのアンマウント	69
7.22. fdisk コマンドによるパーティション変更	69
7.23. EXT4 ファイルシステムの構築	70

7.24. LED を点灯させる	71
7.25. LED を消灯させる	71
7.26. LED の状態を表示する	71
7.27. 対応している LED トリガを表示	71
7.28. LED のトリガに heartbeat を指定する	72
7.29. evtest コマンドのインストール	72
7.30. ユーザースイッチ: イベントの確認	72
8.1. ビルドツール実行前の準備	77
8.2. ビルドツールの実行	78
8.3. ライブラリイメージ作成ツールの実行	78
8.4. ライブラリイメージ作成ツールの実行 (VPU が不要の場合)	78
8.5. ライブラリイメージを書き込む	78
8.6. ライブラリパーティションのマウント	79
8.7. ライブラリバージョンの確認	79
8.8. imx-mm へのシンボリックリンクを作成する	80
9.1. Flutter アプリケーションの例	81
9.2. Flutter アプリケーション開発の流れ	82
9.3. ソフトウェアをアップデートする	82
9.4. VSCode を起動する	83
9.5. VSCode に開発用エクステンションをインストールする	83
9.6. GUI アプリケーションのプロジェクトを作成する	84
9.7. Signage アプリケーションのプロジェクトを作成する	85
9.8. プロジェクト名を入力する	85
9.9. 初期設定を行う	85
9.10. VSCode で初期設定を行う	86
9.11. VSCode のターミナル	86
9.12. SSH 用の鍵を生成する	86
9.13. VSCode でコンテナイメージの作成を行う	87
9.14. コンテナイメージの作成完了	88
9.15. my_project へ移動して VSCode を起動する。	88
9.16. ATDE 上でアプリケーションを実行する	89
9.17. 起動したサンプルアプリケーション	90
9.18. dart_periphery パッケージをインストールする例	90
9.19. video_player パッケージをインストールする例	91
9.20. dart_periphery パッケージをアンインストールする例	91
9.21. ssh_config を編集する	91
9.22. Armadillo 上でアプリケーションを実行する	92
9.23. 実行時に表示されるメッセージ	92
9.24. アプリケーションを終了する	93
9.25. ホットリロード機能を使う	93
9.26. リリース版をビルドする	94
9.27. SWU イメージを作成する	95
9.28. CUI アプリケーション開発の流れ	96
9.29. ソフトウェアをアップデートする	96
9.30. VSCode を起動する	96
9.31. VSCode に開発用エクステンションをインストールする	97
9.32. プロジェクトを作成する	98
9.33. プロジェクト名を入力する	98
9.34. 初期設定を行う	98
9.35. VSCode で初期設定を行う	99
9.36. VSCode のターミナル	99
9.37. SSH 用の鍵を生成する	99
9.38. VSCode でコンテナイメージの作成を行う	100

9.39. コンテナイメージの作成完了	101
9.40. my_project へ移動して VSCode を起動する。	101
9.41. ssh_config を編集する	101
9.42. Armadillo 上でアプリケーションを実行する	102
9.43. 実行時に表示されるメッセージ	102
9.44. アプリケーションを終了する	103
9.45. リリース版をビルドする	104
9.46. ソフトウェアをアップデートする	105
9.47. VSCode を起動する	105
9.48. VSCode に開発用エクステンションをインストールする	105
9.49. プロジェクトを作成する	106
9.50. プロジェクト名を入力する	106
9.51. プロジェクトディレクトリへ移動して VSCode を起動する	107
9.52. initial_setup.swu を作成する	107
9.53. 有線 LAN の設定ファイルを編集する	107
9.54. 無線 LAN の設定ファイルを編集する	108
9.55. LAN 設定をインストールする SWU ファイルを作成する	109
9.56. 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	110
9.57. 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	111
9.58. コンテナを作成する実行例	112
9.59. イメージ一覧の表示実行例	113
9.60. podman images --help の実行例	113
9.61. コンテナ一覧の表示実行例	113
9.62. podman ps --help の実行例	113
9.63. コンテナを起動する実行例	114
9.64. コンテナを起動する実行例(a オプション付与)	114
9.65. podman start --help 実行例	114
9.66. コンテナを停止する実行例	115
9.67. podman stop --help 実行例	115
9.68. my_container を保存する例	115
9.69. chattr によって copy-on-write を無効化する例	116
9.70. podman build の実行例	117
9.71. podman build でのアップデートの実行例	117
9.72. コンテナを削除する実行例	118
9.73. \$ podman rm --help 実行例	118
9.74. イメージを削除する実行例	118
9.75. podman rmi --help 実行例	119
9.76. Read-Only のイメージを削除する実行例	119
9.77. abos-ctrl container-clear 実行例	120
9.78. コンテナ内部のシェルを起動する実行例	120
9.79. コンテナ内部のシェルから抜ける実行例	120
9.80. podman exec --help 実行例	121
9.81. コンテナを作成する実行例	121
9.82. コンテナの IP アドレスを確認する実行例	121
9.83. ping コマンドによるコンテナ間の疎通確認実行例	121
9.84. Docker ファイルによるイメージのビルドの実行例	122
9.85. ビルド済みイメージを load する実行例	123
9.86. GPIO を扱うためのコンテナ作成例	123
9.87. コンテナ内からコマンドで GPIO を操作する例	123
9.88. gpiodetect コマンドの実行	124
9.89. gpioinfo コマンドの実行	124
9.90. I2C を扱うためのコンテナ作成例	125
9.91. i2cdetect コマンドによる確認例	125

9.92. SPI を扱うためのコンテナ作成例 125

9.93. spi-config コマンドによる確認例 126

9.94. CAN を扱うためのコンテナ作成例 126

9.95. CAN の設定例 126

9.96. PWM を扱うためのコンテナ作成例 127

9.97. PWM の動作設定例 127

9.98. 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 128

9.99. setserial コマンドによるシリアルインターフェース設定の確認例 128

9.100. USB シリアルデバイスを扱うためのコンテナ作成例 128

9.101. setserial コマンドによる USB シリアルデバイス設定の確認例 128

9.102. USB カメラを扱うためのコンテナ作成例 129

9.103. USB メモリをホスト OS 側でマウントする例 129

9.104. 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 129

9.105. USB メモリに保存されているデータの確認例 130

9.106. USB メモリをマウントするためのコンテナ作成例 130

9.107. コンテナ内から USB メモリをマウントする例 130

9.108. RTC を扱うためのコンテナ作成例 131

9.109. hwclock コマンドによる RTC の時刻表示と設定例 131

9.110. 音声出力を行うためのコンテナ作成例 131

9.111. alsa-utils による音声出力を行う例 132

9.112. 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 132

9.113. evtest コマンドによる確認例 132

9.114. LED を扱うためのコンテナ作成例 133

9.115. LED の点灯/消灯の実行例 133

9.116. Bluetooth デバイスを扱うためのコンテナ作成例 133

9.117. Bluetooth を起動する実行例 134

9.118. bluetoothctl コマンドによるスキャンとペアリングの例 134

9.119. Wi-SUN デバイスを扱うためのコンテナ作成例 135

9.120. EnOcean デバイスを扱うためのコンテナ作成例 135

9.121. コンテナの IP アドレス確認例 135

9.122. ip コマンドを用いたコンテナの IP アドレス確認例 136

9.123. ユーザ定義のネットワーク作成例 136

9.124. IP アドレス固定のコンテナ作成例 136

9.125. コンテナの IP アドレス確認例 137

9.126. コンテナに Apache をインストールする例 137

9.127. コンテナに lighttpd をインストールする例 138

9.128. コンテナに vsftpd をインストールする例 138

9.129. ユーザを追加する例 138

9.130. 設定ファイルの編集例 139

9.131. vsftpd の起動例 139

9.132. コンテナに samba をインストールする例 139

9.133. ユーザを追加する例 139

9.134. samba の起動例 140

9.135. コンテナに sqlite をインストールする例 140

9.136. sqlite の実行例 140

9.137. Wayland を扱うためのコンテナ作成例 140

9.138. 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 141

9.139. weston.ini 142

9.140. weston.ini をボリュームで渡す実行例 143

9.141. X Window System を扱うためのコンテナ起動例 145

9.142. コンテナ内で X Window System を起動する実行例 146

9.143. 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 146

9.144. フレームバッファに直接描画する実行例 147

9.145. タッチパネルを扱うためのコンテナ作成例 147

9.146. VPU を扱うためのコンテナ作成例 147

9.147. weston と GStreamer を扱うためのコンテナ作成例 148

9.148. GStreamer によるデコード実行例 148

9.149. GStreamer によるエンコード実行例 148

9.150. パワーマネジメント機能を使うためのコンテナ作成例 149

9.151. サスペンド状態にする実行例 149

9.152. サスペンド状態にする実行例、rtc で起こす 149

9.153. コンテナから shutdown を行う 151

9.154. ソフトウェアウォッチドッグタイマーを使うためのコンテナ作成例 151

9.155. 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 152

9.156. 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 152

9.157. 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 152

9.158. NPU を扱うためのコンテナ作成例 152

9.159. ONNX Runtime をインストールする例 153

9.160. python から ONNX Runtime を使う例 153

9.161. TensorFlow Lite をインストールする例 154

9.162. python から TensorFlow Lite を使う例 154

9.163. tflite-runtime のバージョンを確認する 155

9.164. tflite-runtime のバージョンを下げる 156

9.165. at-debian-image のバージョンを確認する 157

9.166. Arm NN をインストールする例 157

9.167. python から Arm NN を使う例 157

9.168. コンテナを自動起動するための設定例 158

9.169. ボリュームを shared でサブマウントを共有する例 159

9.170. /proc/devices の内容例 160

9.171. pod を使うコンテナを自動起動するための設定例 163

9.172. network を使うコンテナを自動起動するための設定例 164

9.173. abos-ctrl podman-rw の実行例 166

9.174. abos-ctrl podman-storage のイメージコピー例 166

9.175. GStreamer を実行するためのコンテナ作成例 169

9.176. gstreamer のインストール 170

9.177. weston の起動 170

9.178. pulseaudio の起動 170

9.179. GStreamer の実行例 170

9.180. H.264/AVC 動画の再生(音声あり) 171

9.181. H.264/AVC 動画の再生(音声なし) 171

9.182. VP8 動画の再生(音声あり) 171

9.183. VP8 動画の再生(音声なし) 172

9.184. VP9 動画の再生(音声あり) 172

9.185. VP9 動画の再生(音声なし) 172

9.186. HTTP ストリーミングの再生(音声あり) 172

9.187. HTTP ストリーミングの再生(音声なし) 172

9.188. RTSP ストリーミングの再生(音声あり) 172

9.189. RTSP ストリーミングの再生(音声なし) 173

9.190. USB カメラからの映像表示(音声あり) 173

9.191. USB カメラからの映像表示(音声なし) 173

9.192. USB カメラからの映像を H.264 で録画(音声あり) 174

9.193. USB カメラからの映像を H.264 で録画(音声なし) 174

9.194. 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) 174

9.195. 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) 174

9.196. Linux カーネルを SWU でインストールする方法 180

9.197. Linux カーネルを build_rootfs でインストールする方法 180

9.198. 自動マウントされた microSD カードのアンマウント	185
9.199. hawkBit コンテナの TLS なしの場合 (テスト用) の実行例	196
9.200. hawkBit コンテナの TLS ありの場合の実行例	197
9.201. persist_file のヘルプ	217
9.202. persist_file 保存・削除手順例	217
9.203. 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	218
9.204. persist_file 変更ファイルの一覧表示例	218
9.205. persist_file でのパッケージインストール手順例	219
9.206. /var/at-log/atlog の内容の例	219
9.207. buttdond で SW1 を扱う	220
9.208. local サービスの実行例	221
9.209. uboot_env.d のコンフィグファイルの例	222
9.210. chronyd のコンフィグの変更例	224
9.211. at-dtweb の起動開始	225
9.212. ボード選択画面	226
9.213. Linux カーネルディレクトリ選択画面	226
9.214. at-dtweb 起動画面	226
9.215. UART3(RXD/TXD) のドラッグ	227
9.216. CON11 8/10 ピンへのドロップ	228
9.217. 信号名の確認	228
9.218. プロパティの設定	229
9.219. プロパティの保存	229
9.220. 全ての機能の削除	230
9.221. I2C5(SCL/SDA) の削除	230
9.222. DTS/DTB の生成	231
9.223. dtbo/desc の生成完了	231
9.224. /boot/overlays.txt の変更例	232
9.225. DT overlay を作成する例	233
9.226. データリテンション開始トリガーの方式	236
9.227. データリテンションの開始トリガーの動作例	237
9.228. 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	239
9.229. デモアプリケーションランチャの起動	240
9.230. パッケージのバージョンを確認する	241
9.231. pulseaudio のインストールと起動	242
9.232. pillow のインストールと起動	244
9.233. ビデオデバイスの変更	244
9.234. ビデオデバイスの変更	245
9.235. ビデオデバイスの変更	246
9.236. ビデオデバイスの変更	247
9.237. ビデオデバイスの変更	248
9.238. 各種コンフィグの変更	249
9.239. atmark-thermal-profiler をインストールする	250
9.240. atmark-thermal-profiler を実行する	251
9.241. atmark-thermal-profiler を停止する	251
9.242. ログファイルの内容例	251
9.243. 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)	252
9.244. Armadillo-IoT ゲートウェイ G4 で取得した温度のグラフ	253
10.1. 動作ログのフォーマット	259
12.1. hawkBit が扱うソフトウェアのデータ構造	275
13.1. 電源回路の構成	278
13.2. 電源シーケンス	279
13.3. リセット回路の構成	280
13.4. リセットシーケンス	280

13.5. ONOFF 回路の構成	281
13.6. Armadillo-X2 のインターフェース	282
13.7. CON1 microSD スロット 取り扱い上の注意事項	283
13.8. CON3 LAN LED 配置	284
13.9. CON10 接続可能なフレキシブルフラットケーブルの形状	286
13.10. AC アダプタの極性マーク	291
13.11. 基板形状図	294
13.12. Armadillo-X2 の IC1 とヒートシンク固定穴の位置	296
13.13. Armadillo-X2 の拡張インターフェース	297
13.14. Armadillo-IoT ゲートウェイ X2 の拡張ボード例	298
13.15. スイッチ、LED、リレー接続例	299
13.16. DC/DC コンバータ回路(VDD_5V 入力、3.3V 1.5A 出力)例	299
13.17. 1.8V ↔ 3.3V 双方向レベル変換回路の例	300
13.18. Armadillo-X2 オプションケース(金属製)	301
13.19. オプションケース(金属製)の加工痕例	302
13.20. オプションケース(金属製) 放熱シート貼付	302
13.21. オプションケース(金属製) ケース(下)ねじ止め	303
13.22. オプションケース(金属製) ケース(上)ねじ止め	304
13.23. ケース(上)を閉じる際の注意	304
13.24. ケース(上)形状図	305
13.25. ケース(下)形状図	306
13.26. ケースを固定する際の注意	307
13.27. Armadillo-X2、G4 ケースモデル VESA 規格固定用プレート	307
13.28. ケースに VESA 規格固定用プレートを取り付け	308
13.29. モニターに VESA 規格固定用プレートを取り付け	309
13.30. Armadillo-X2、G4 ケースモデル VESA 規格固定用プレート形状図	310

表目次

1.1. 使用しているフォント	20
1.2. 表示プロンプトと実行環境の関係	20
1.3. コマンド入力例での省略表記	21
2.1. 推奨温湿度環境について	24
3.1. Armadillo-X2 ラインアップ	32
3.2. 仕様	33
3.3. eMMC メモリマップ	34
3.4. eMMC ブートパーティション構成	35
3.5. eMMC GPP 構成	35
4.1. ユーザー名とパスワード	40
4.2. 動作確認に使用する取り外し可能デバイス	42
4.3. シリアル通信設定	44
4.4. インターフェース内容	48
4.5. ジャンパの状態と起動デバイス	52
4.6. 入力モードに移行するコマンド	53
4.7. カーソルの移動コマンド	54
4.8. 文字の削除コマンド	54
4.9. 保存・終了コマンド	54
7.1. ネットワークとネットワークデバイス	62
7.2. 固定 IP アドレス設定例	65
7.3. ストレージデバイス	67
7.4. eMMC の GPP の用途	68
7.5. LED クラスディレクトリと LED の対応	70
7.6. LED トリガの種類	71
7.7. インプットデバイスファイルとイベントコード	72
8.1. ライブラリイメージ書き込み済みの製品	78
9.1. 組み合わせて使うパッケージ	90
9.2. 対応するパワーマネジメント状態	149
9.3. ライブラリと tf-lite-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 ..	155
9.4. 2.6.0-1 未満の TensorFlow Lite 関連 deb パッケージ	155
9.5. add_hotplugs オプションに指定できる主要な文字列	160
9.6. H.264/AVC デコーダー仕様	175
9.7. VP8 デコーダー仕様	175
9.8. VP9 デコーダー仕様	175
9.9. H.264/AVC エンコーダー仕様	175
9.10. build-rootfs のファイル説明	182
9.11. microSD カードのパーティション構成	186
9.12. u-boot の主要な環境変数	223
9.13. データリテンションの挙動	235
9.14. Armadillo のデータリテンションの設定	239
9.15. デモアプリケーション動作のために必要なバージョン	241
9.16. ネジ検出デモのパラメータの詳細	250
9.17. thermal_profile.csv の各列の説明	252
11.1. キーコード	267
11.2. I2C デバイス	268
11.3. 対応するパワーマネジメント状態	271
13.1. 絶対最大定格	276
13.2. 推奨動作条件	276
13.3. 電源出力仕様	276
13.4. 拡張入出力ピンの電氣的仕様(OVDD=VDD_3V3, VDD_1V8)	277

13.5. 拡張入出力ピンの電氣的仕様(OVDD=VEXT_3V3)	277
13.6. 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	281
13.7. Armadillo-X2 インターフェース一覧	282
13.8. CON1 信号配列	283
13.9. CON3 信号配列 (10BASE-T/100BASE-TX)	284
13.10. CON3 信号配列 (1000BASE-T)	284
13.11. CON3 LAN LED の動作	284
13.12. CON4 信号配列	285
13.13. CON6 信号配列	285
13.14. CON7 信号配列	285
13.15. CON10 搭載コネクタとフレキシブルフラットケーブル例	286
13.16. CON10 信号配列	286
13.17. CON11 搭載コネクタと対向コネクタ例	288
13.18. CON11 信号配列	288
13.19. CON13 信号配列	291
13.20. CON15 搭載コネクタと対向コネクタ例	291
13.21. CON15 信号配列	291
13.22. CON15 搭載コネクタ例	292
13.23. CON16 信号配列	292
13.24. CON4 信号配列	293
13.25. ジャンパの状態と起動デバイス	293
13.26. JP1 信号配列	293
13.27. SW1 信号配列	293
13.28. LED3 の状態	293
13.29. LED4 の状態	294
13.30. 各インターフェースへの電流供給例	300
13.31. Armadillo-X2 関連のオプション品	301
13.32. Armadillo-X2 オプションケースセット(金属製)について	301
13.33. Armadillo-X2 オプションケース(金属製)の仕様	301
13.34. Armadillo-X2、G4 ケースモデル VESA 規格固定用プレートについて	307
13.35. Armadillo-X2、G4 ケースモデル VESA 規格固定用プレートの仕様	307

1. はじめに

このたびは Armadillo-X2 をご利用いただき、ありがとうございます。

Armadillo-X2 は、GUI を必要とする機器や映像出力機器などに最適な CPU ボードです。Google が提供するオープンソースの GUI 開発環境「Flutter」に標準対応しており、モバイルアプリフレームワークを活用することで、容易に GUI アプリケーションを開発できます。また、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 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-X2 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によって中断してもアップデート前の状態に復旧します。

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

1.1. 本書で扱うこと扱わないこと

1.1.1. 扱うこと

本書では、Armadillo-X2 の使い方、製品仕様(ソフトウェアおよびハードウェア)、オリジナルの製品を開発するために必要となる情報、その他注意事項について記載しています。Linux あるいは組み込み機器に不慣れな方でも読み進められるよう、コマンドの実行例なども記載しています。

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

1.1.2. 扱わないこと

本書では、一般的な Linux のプログラミング、デバッグ方法やツールの扱い方、各種モジュールの詳細仕様など、一般的な情報や、他に詳しい情報があるものは扱いません。また、(Armadillo-X2 を使用した)最終製品あるいはサービスに固有な情報や知識も含まれていません。

1.2. 本書で必要となる知識と想定する読者

本書は、読者として Armadillo-X2 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-X2 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-X2 は組み込みプラットフォームとして実績のある Armadillo をベースとしているため、標準で有効になっている機能以外にも様々な機能を実現することができます。

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

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

1.3. ユーザー限定コンテンツ

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

- ・ 各種信頼性試験データ・納入仕様書等製造関連情報

限定コンテンツを取得するには、「6. ユーザー登録」を参照してください。

1.4. 本書および関連ファイルのバージョンについて

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

Armadillo サイト - Armadillo-X2 ドキュメントダウンロード

<https://armadillo.atmark-techno.com/armadillo-x2/resources/documents>

Armadillo サイト - Armadillo-X2 ソフトウェアダウンロード

<https://armadillo.atmark-techno.com/armadillo-x2/resources/software>

1.5. 本書の構成

本書には、Armadillo-X2 をベースに、オリジナルの製品を開発するために必要となる情報を記載しています。また、取扱いに注意が必要な事柄についても説明しています。

- ・ はじめにお読みください。
 - 「1. はじめに」、「2. 注意事項」
- ・ Armadillo-X2 の仕様を紹介します。

- 「3. 製品概要」
- ・ 工場出荷状態のソフトウェアの使い方や、動作を確認する方法を紹介します。
- 「4. Armadillo の電源を入れる前に」、「5. 起動と終了」、「7. 動作確認方法」
- ・ 工場出荷状態のソフトウェア仕様について紹介します。
- 「10. 動作ログ」、「12. ソフトウェア仕様」
- ・ システム開発に必要な情報を紹介します。
- 「8. 開発の基本的な流れ」、「9. Howto」、「11. 製品機能」
- ・ 拡張基板の開発や、ハードウェアをカスタマイズする場合に必要な情報を紹介します。
- 「13. ハードウェア仕様」
- ・ ご購入ユーザーに限定して公開している情報の紹介やユーザー登録について紹介します。
- 「6. ユーザー登録」

1.6. 表記について

1.6.1. フォント

本書では以下のような意味でフォントを使いわけています。

表 1.1 使用しているフォント

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

1.6.2. コマンド入力例

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

表 1.2 表示プロンプトと実行環境の関係

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




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

表 1.3 コマンド入力例での省略表記

表記	説明
[VERSION]	ファイルのバージョン番号

1.6.3. アイコン

本書では以下のようにアイコンを使用しています。

	注意事項を記載します。
	役に立つ情報を記載します。
	用語の説明や補足的な説明を記載します。

1.7. 謝辞

Armadillo で使用しているソフトウェアの多くは Free Software / Open Source Software で構成されています。Free Software / Open Source Software は世界中の多くの開発者の成果によってなっています。この場を借りて感謝の意を表します。

2. 注意事項

2.1. 安全に関する注意事項

本製品を安全にご使用いただくために、特に以下の点にご注意ください。



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

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

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

2.2. 取扱い上の注意事項

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

破損しやすい箇所	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可能性があります。ノイズの影響を受けやすい入出力ラインには、保護回路を入れることや、ノイズ源となる装置と共通の電源を使用しない等の対策をとることをお勧めします。
衝撃	落下や衝撃などの強い振動を与えないでください。

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

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

^[3]別途、活線挿抜を禁止している場合を除く

- 使用場所の制限 無線機能を搭載した製品は、テレビ・ラジオに近接する場所で使用すると、受信障害を招く恐れがあります。
- 振動 振動が発生する環境では、Armadillo が動かないよう固定して使用してください。
- 電池の取り扱い 電池の使用推奨期限を過ぎる前に電池の交換をしてください。使用推奨期限を超えて使用すると、電池の性能が十分に発揮できない場合や、電池を漏液させたり、製品を破損させるおそれがあります。

2.3. 製品の保管について



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

表 2.1 推奨温湿度環境について

推奨温湿度環境	5~35°C/70%RH 以下 ^[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。

2.4. ソフトウェア使用に関する注意事項

本製品に含まれるソフトウェアについて

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

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



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

2.5. 電波障害について



この装置は、クラス 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

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。

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

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。

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



3. 製品概要

3.1. 製品の特長

3.1.1. Armadillo とは

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

- ・ Arm プロセッサ搭載・省電力設計

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

- ・ 小型・手のひらサイズ

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

- ・ 標準 OS として Linux をプリインストール

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

- ・ 開発環境

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

3.1.2. Armadillo-X2 とは

Armadillo-X2 は、GUI を必要とする機器や映像出力機器などに最適な小型・高性能 CPU ボードです。Armadillo-X2 には以下の特長があります。

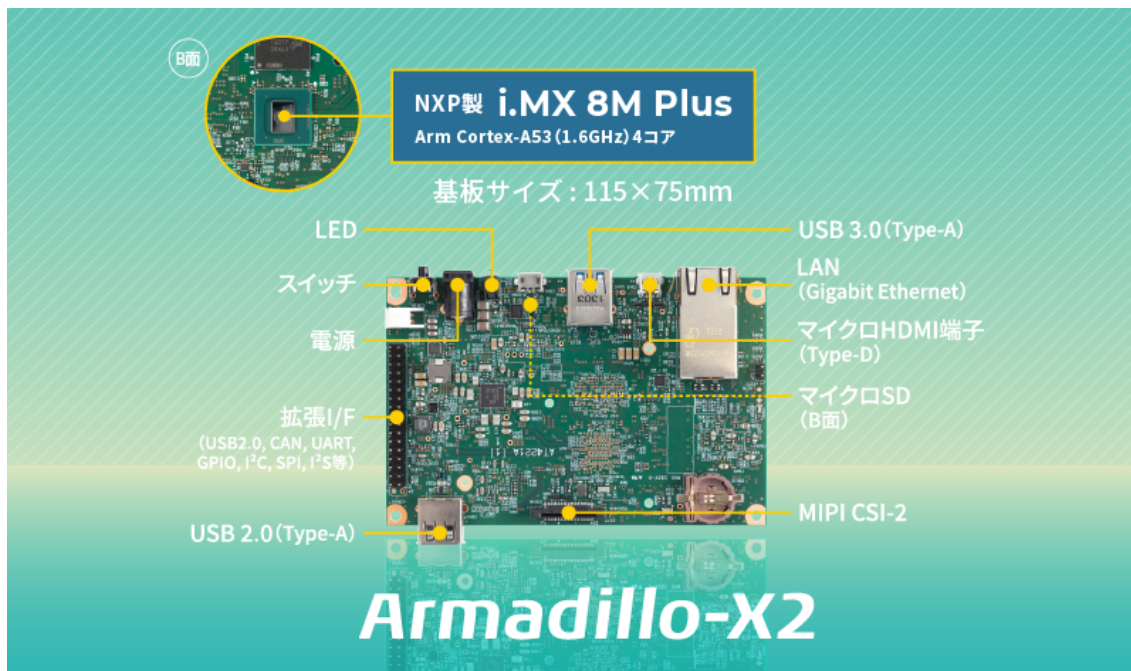


図 3.1 Armadillo-X2 とは

- ・ GUI 開発環境「Flutter」に標準対応

Google が提供するオープンソースの GUI 開発環境「Flutter」に標準対応。このモバイルアプリフレームワークを活用することで、容易に GUI アプリケーションを開発できます。



図 3.2 Flutter を用いた GUI アプリケーション開発例

- ・ エッジ AI 処理を省電力で実現

内蔵する NPU 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。

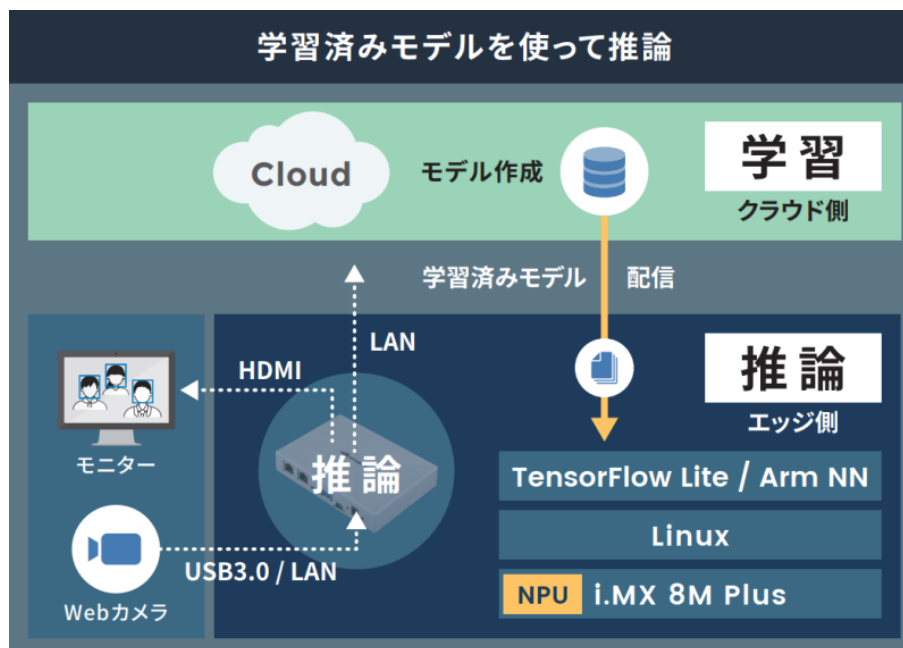


図 3.3 エッジ AI 処理、機械学習の例

- ・ NXP 製 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。

- ・ ファンレス・小型設計

高負荷のかかるエッジ AI 処理でも、動作温度範囲内であれば処理能力が低下しない稼働を期待できます。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。

- ・ オプションで専用アルミケースを選択可能

オプションで専用のアルミケースが用意されています。熱源である SoC をメインボードの裏面にレイアウトし、ヒートシンクなどを使わずアルミ製のケースに直接放熱することで、ボード単体で使用するよりも更に安定した稼働を見込めます。

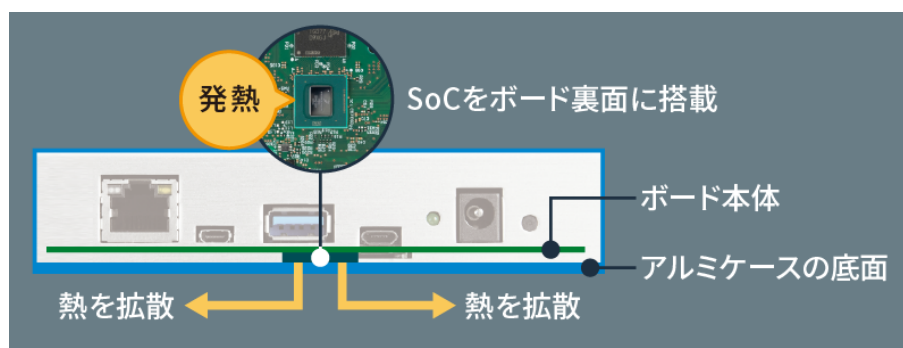


図 3.4 SoC の発熱をアルミケースに直接放熱

- ・ ADLINK 製タッチモニタ「OM Series」に対応

オープンフレーム産業用タッチモニタ「OM Series」は、Armadillo-X2 で動作確認済みのデバイスです。10.1/15.6/21.5 インチからモデルを選択できます。

- ・ 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。

3.1.3. Armadillo Base OS とは

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

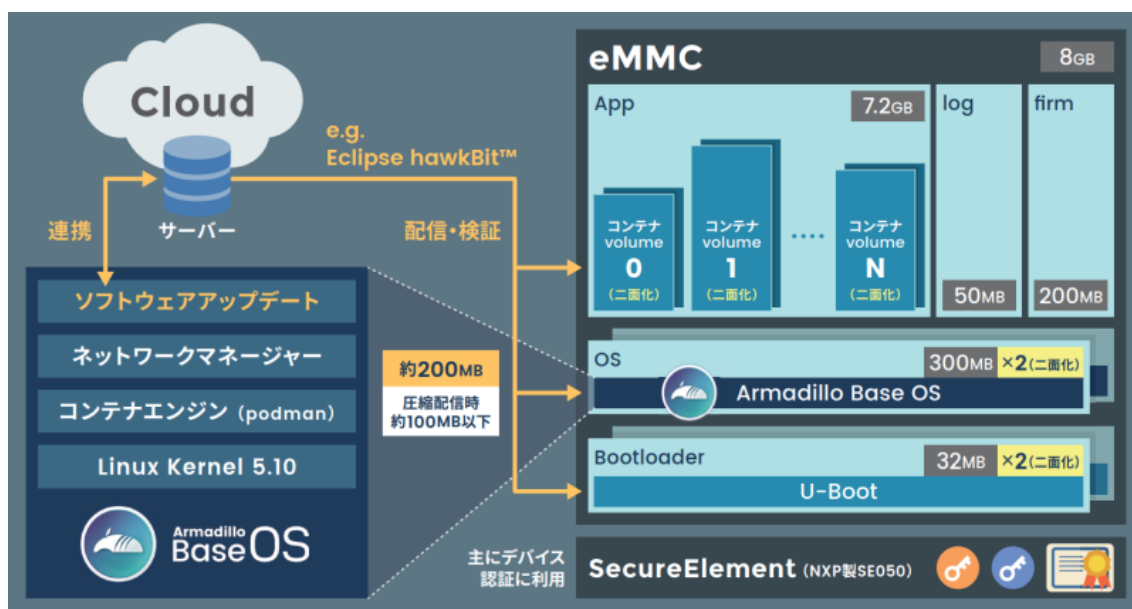


図 3.5 Armadillo Base OS とは

- ・ OS のコンパクト化

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

- ・ コンテナによるアプリケーション運用

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

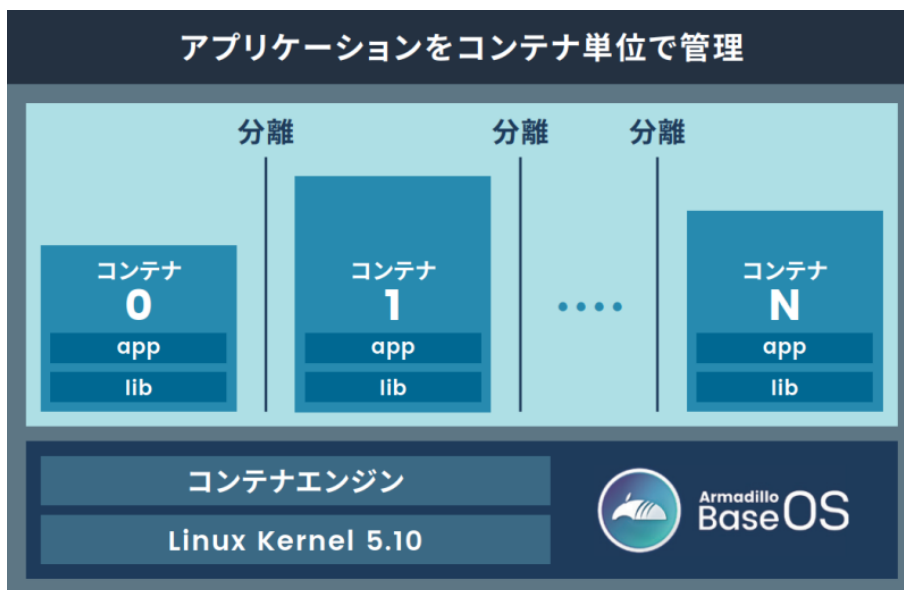


図 3.6 コンテナによるアプリケーションの運用

- ・アップデート機能を標準搭載

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

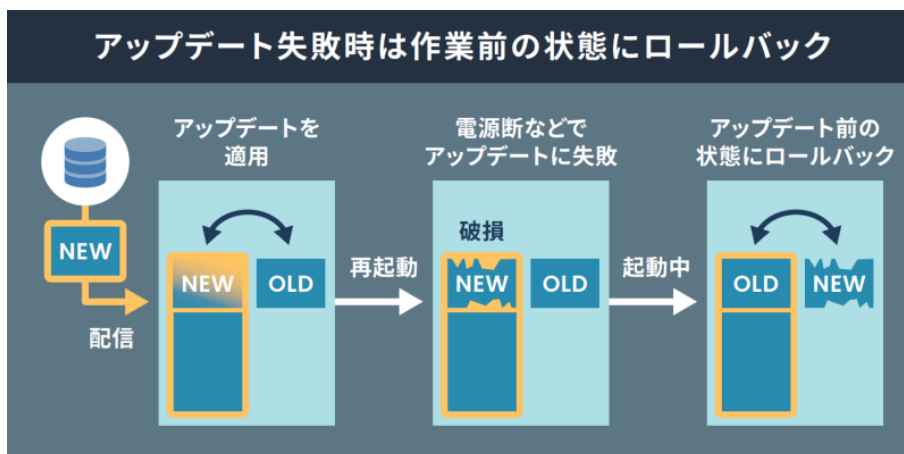


図 3.7 ロールバックの仕組み

- ・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。

3.2. 製品ラインアップ

Armadillo-X2 の製品ラインアップは次のとおりです。

表 3.1 Armadillo-X2 ラインアップ

名称	型番
Armadillo-X2 開発セット(メモリ 2GB)	AX2210-U00D0
Armadillo-X2 量産ボード(メモリ 2GB、ストレージ 10GB)	AX2210-U00Z
Armadillo-X2 量産ボード(メモリ 2GB、ストレージ 10GB、ケース入り)	AX2210-C00Z

3.2.1. Armadillo-X2 開発セット

Armadillo-X2 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。

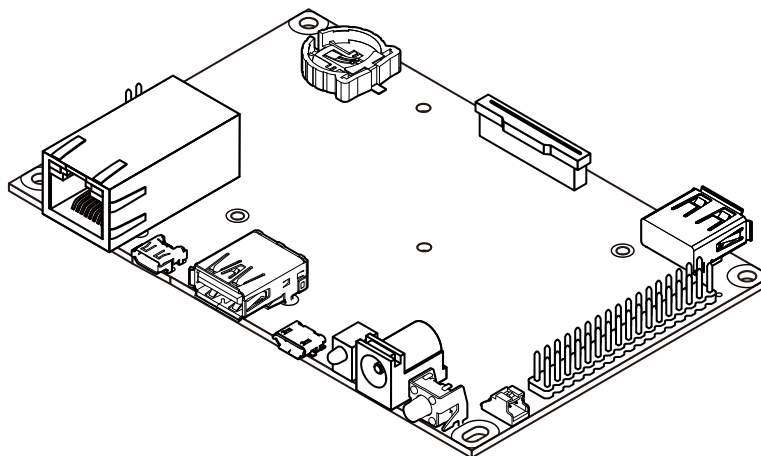


図 3.8 Armadillo-X2 の外観

Armadillo-X2 開発セットのセット内容は以下のとおりです。ケースは付属しませんので、必要な場合は別途ご用意ください。

- ・ Armadillo-X2 (基板単体)
- ・ 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

3.2.2. Armadillo-X2 量産ボード

Armadillo-X2 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。

AX2210-UxxZ が基板単体、AX2210-CxxZ がケースに収められた製品となります。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。

3.3. 仕様

Armadillo-X2 の主な仕様は次のとおりです。

表 3.2 仕様

型番	AX2210-U00D0,AX2210-U00Z,AX2210-C00Z
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 × 1 AUTO-MDIX 対応
モバイル通信	非搭載
USB	USB 3.0 Host × 1 (Type-A)、USB 2.0 Host × 1 (Type-A)、USB 2.0 Host × 1 (ピンヘッダ)
SD	microSD スロット × 1 (UHS-I)
ビデオ	HDMI 出力 × 1 (micro Type-D)
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 × 21、SPI × 2、UART × 2、PDM MIC × 4、I2S × 1、CAN × 2、I2C × 3、PWM × 4
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°C~70°C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]
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)	2.2W(定常状態) ^[e]
動作温度範囲	-20~+70°C(結露なきこと)
外形サイズ(基板)	115×75mm(突起部を除く)
外形サイズ(ケース)	123×82×26mm(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)の信号線において、i.MX 8M Plus のピンマルチプレスの設定で、優先的に機能を割り当てた場合に拡張可能な最大数を記載しています。

^[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。

3.4. ブロック図

Armadillo-X2 のブロック図を「図 3.9. ブロック図」に示します。

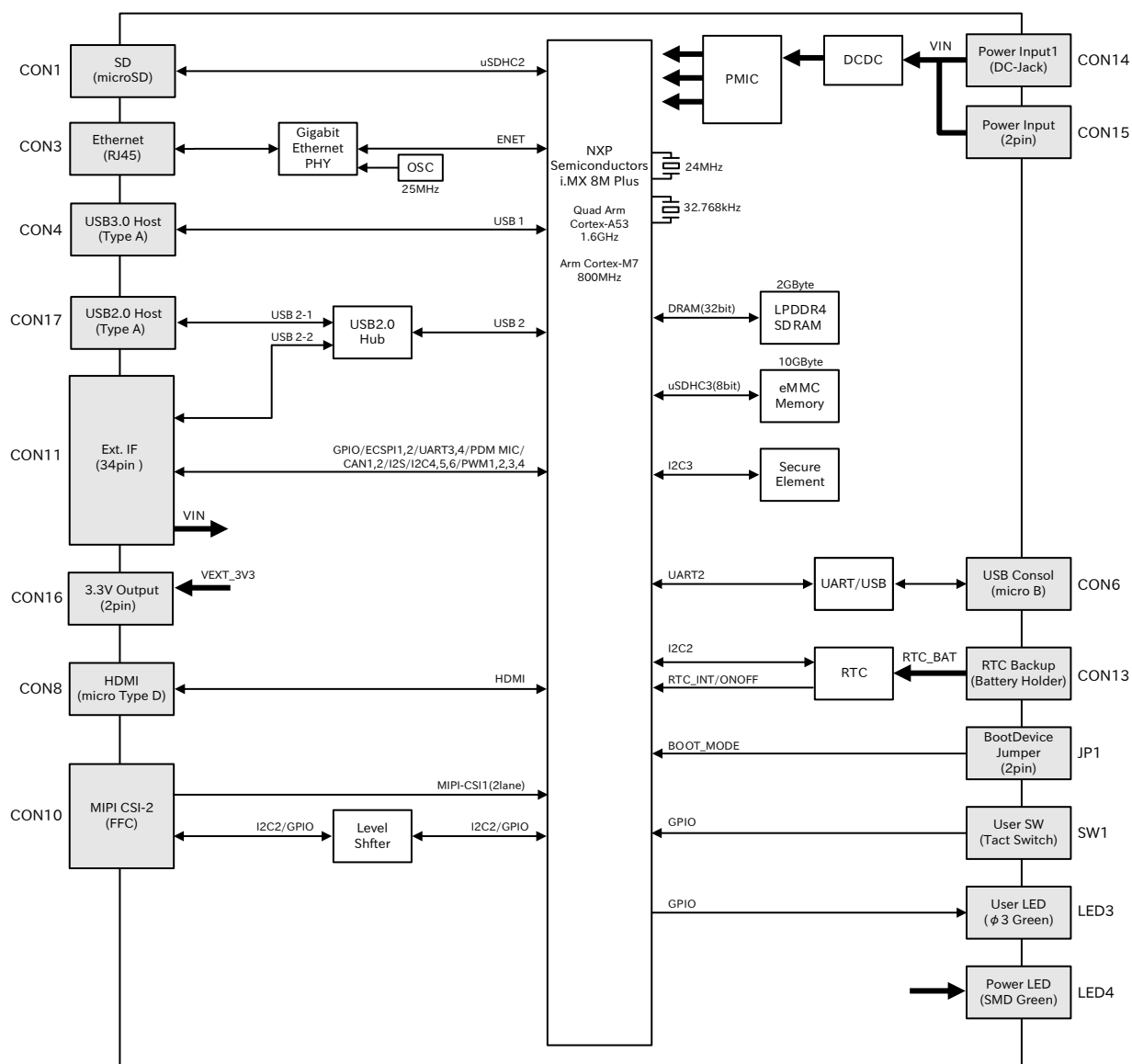


図 3.9 ブロック図

3.5. ストレージデバイスのパーティション構成

Armadillo-X2 の eMMC のパーティション構成を「表 3.3. eMMC メモリマップ」に示します。

表 3.3 eMMC メモリマップ

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

パーティション	サイズ	ラベル	説明
5	8.95GiB	app	アプリケーション用パーティション

Armadillo-X2 の eMMC のブートパーティションの構成を「表 3.4. eMMC ブートパーティション構成」に示します。

表 3.4 eMMC ブートパーティション構成

ディスクデバイス	サイズ	説明
/dev/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-X2 の eMMC の GPP(General Purpose Partition)の構成を「表 3.5. eMMC GPP 構成」に示します。

表 3.5 eMMC GPP 構成

ディスクデバイス	サイズ	説明
/dev/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「10.4. ログ用パーティションについて」を参照ください。

4. Armadillo の電源を入れる前に

4.1. 準備するもの

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

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

4.2. 開発/動作確認環境の構築

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



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

- ・ GPL v2 (General Public License version 2) で提供されている ^[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-X2 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-X2 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。

[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。

4.2.1. ATDE のセットアップ

4.2.1.1. VMware のインストール

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



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



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

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

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

4.2.1.2. ATDE のアーカイブを取得

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



本製品に対応している ATDE のバージョンは ATDE9 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。

4.2.1.3. ATDE のアーカイブを展開

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

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

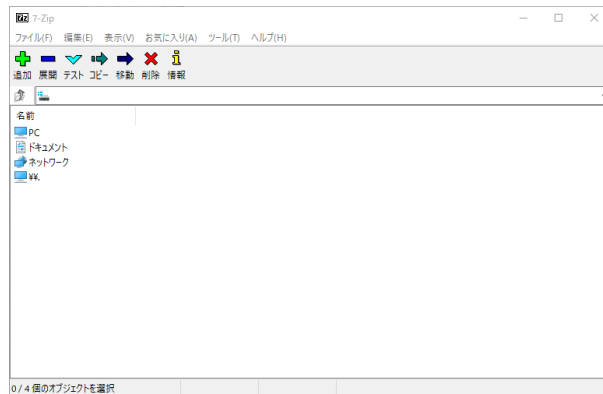
4.2.1.4. Windows で ATDE のアーカイブ展開する

1. 7-Zip のインストール

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

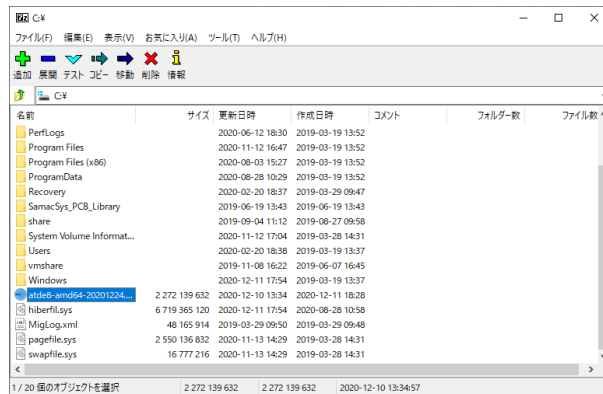
2. 7-Zip の起動

7-Zip を起動します。



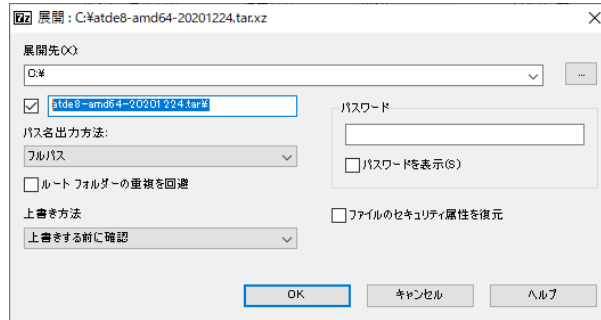
3. xz 圧縮ファイルの選択

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



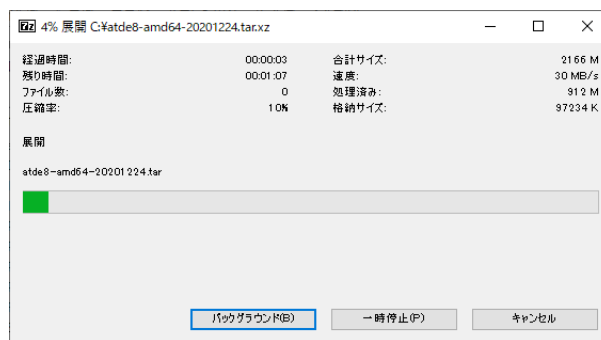
4. xz 圧縮ファイルの展開先の指定

「展開先」を指定して、「OK」をクリックします。



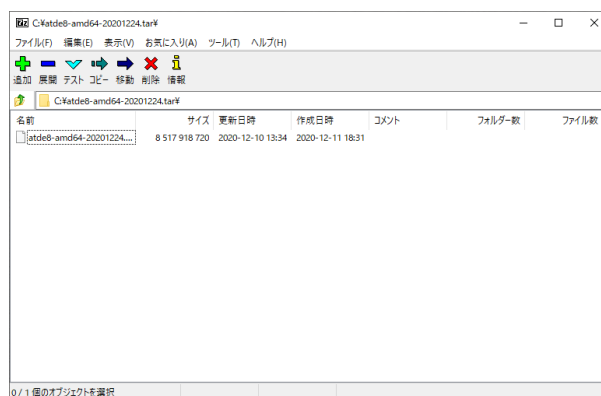
5. xz 圧縮ファイルの展開

展開が始まります。



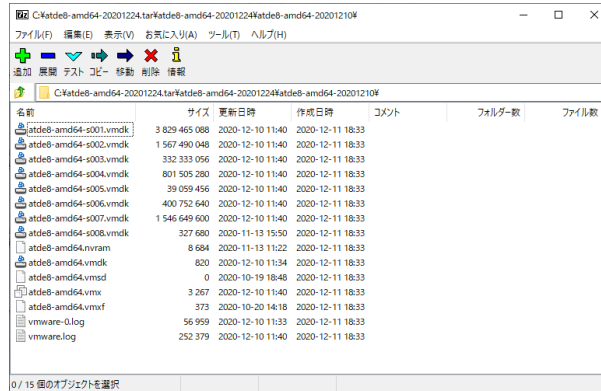
6. tar アーカイブファイルの選択

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



7. 展開の完了確認

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



4.2.1.5. Linux で tar.xz 形式のファイルを展開する

1. tar.xz 圧縮ファイルの展開

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

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

2. 展開の完了確認

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


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

4.2.1.6. ATDE の起動

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

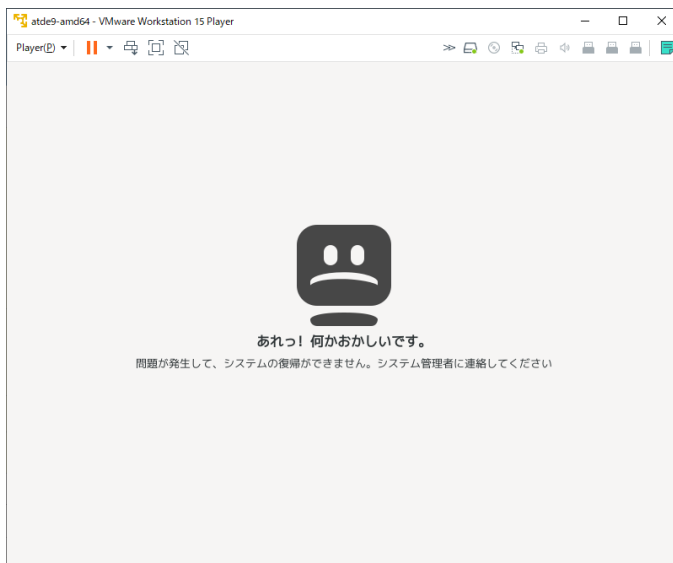
表 4.1 ユーザー名とパスワード

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

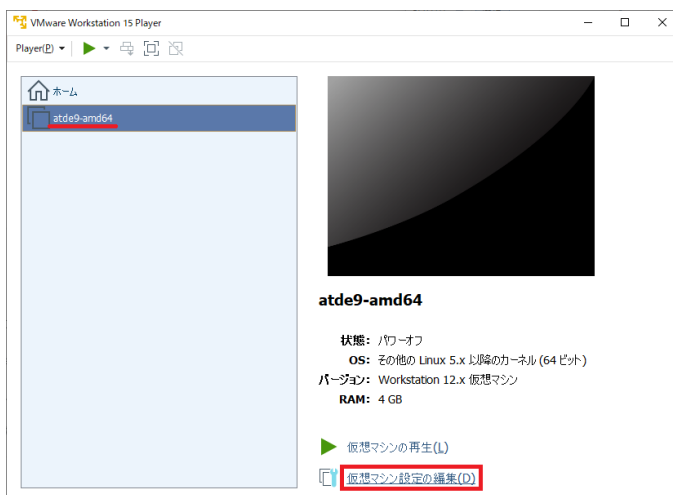


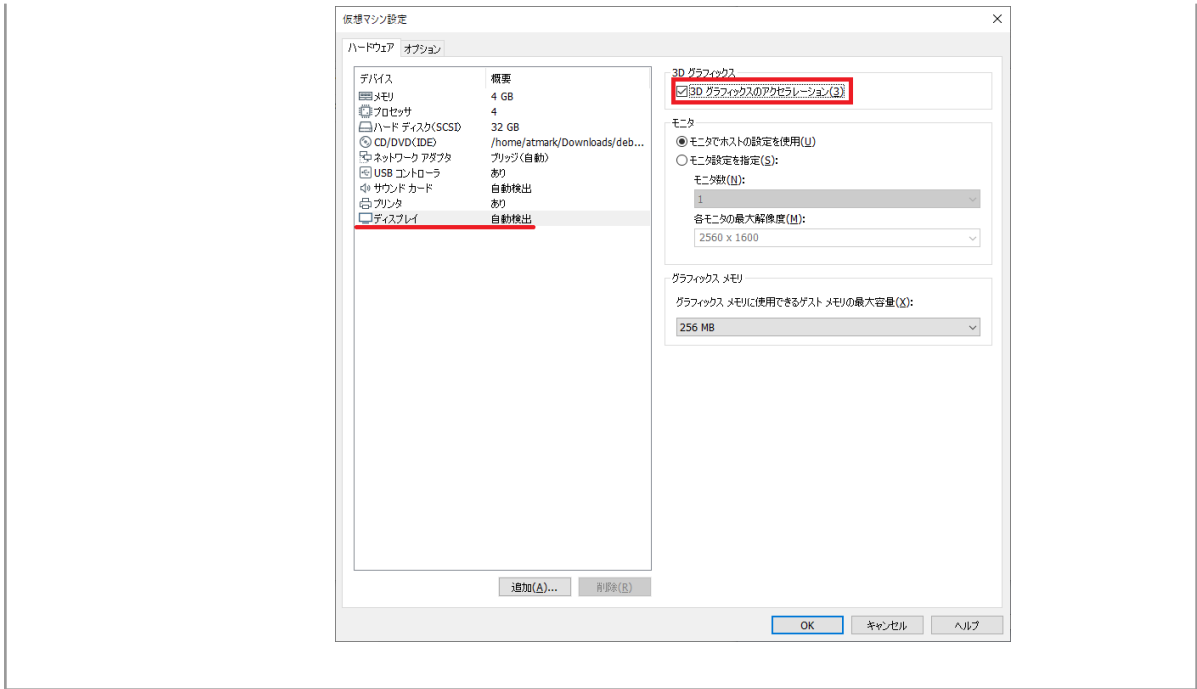
ATDE を起動する環境によっては、GUI ログイン画面が表示されずに以下のようなエラー画面が表示される場合があります。


[2]特権ユーザーで GUI ログインを行うことはできません



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








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

4.2.2. 取り外し可能デバイスの使用

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



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

Armadillo-X2 の動作確認を行うためには、「表 4.2. 動作確認に使用する取り外し可能デバイス」に示すデバイスをゲスト OS に接続する必要があります。

表 4.2 動作確認に使用する取り外し可能デバイス

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

4.2.3. コマンドライン端末(GNOME 端末)の起動

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

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

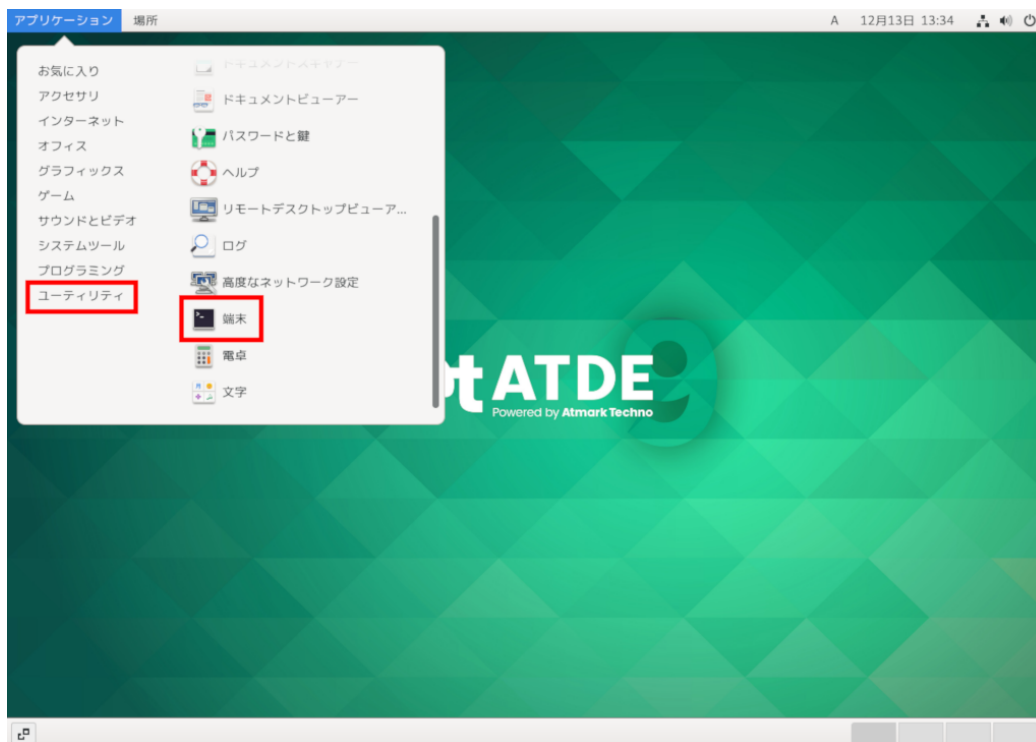


図 4.1 GNOME 端末の起動

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

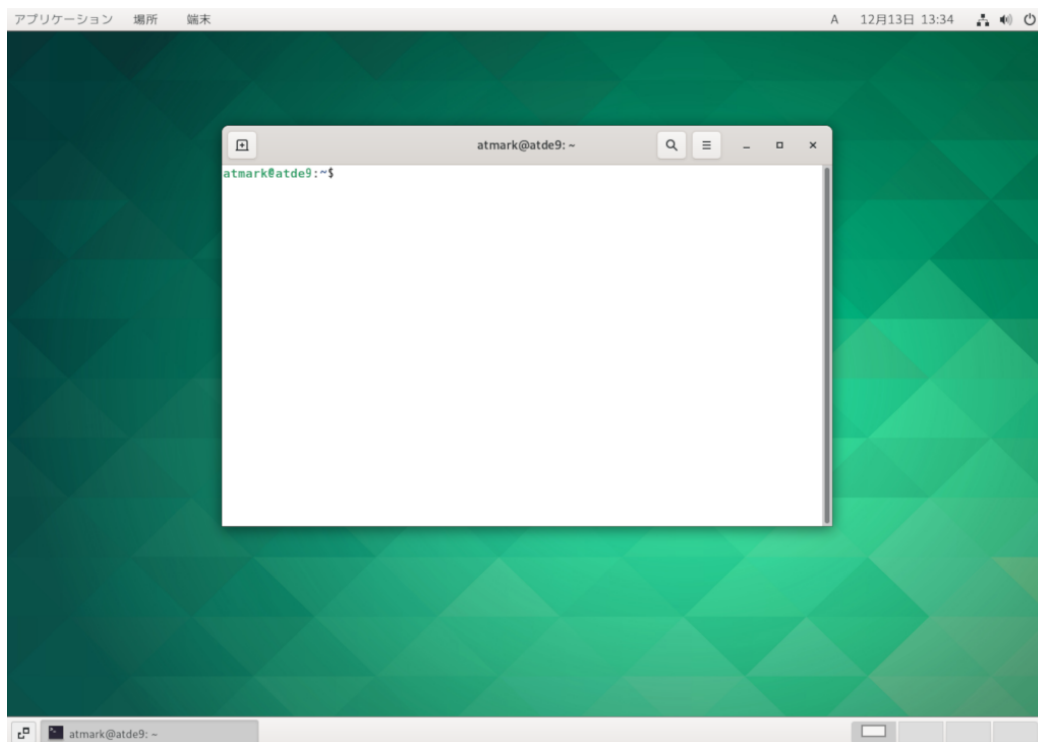


図 4.2 GNOME 端末のウィンドウ

4.2.4. シリアル通信ソフトウェア(minicom)の使用

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

表 4.3 シリアル通信設定

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

1. 「図 4.3. minicom の設定の起動」に示すコマンドを実行し、minicom の設定画面を起動してください。

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

図 4.3 minicom の設定の起動

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

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

```

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

図 4.4 minicom の設定

- 「図 4.5. minicom のシリアルポートの設定」が表示されますので、A キーを押して Serial Device を選択してください。

```

-----
A - Serial Device      : /dev/ttyUSB0
B - Lockfile Location  : /var/lock
C - Callin Program     :
D - Callout Program    :
E - Bps/Par/Bits       : 115200 8N1
F - Hardware Flow Control : No
G - Software Flow Control : No

Change which setting?
    
```

図 4.5 minicom のシリアルポートの設定

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



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 4.6 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 4.7. minicom のシリアルポートのパラメータの設定」が表示されます。

```
+-----[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 4.7 minicom のシリアルポートのパラメータの設定

8. 「図 4.7. minicom のシリアルポートのパラメータの設定」では、転送レート、データ長、ストップビット、パリティの設定を行います。
9. 現在の設定値は「Current」に表示されています。それぞれの値の内容は「図 4.8. minicom シリアルポートの設定値」を参照してください。

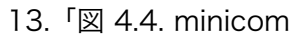
Current: 115200 8 N 1

└───┬───┘
└───┬───┘
└───┬───┘
└───┬───┘

転送レート
データ長
ストップビット
パリティ

図 4.8 minicom シリアルポートの設定値

10. E キーを押して、転送レートを 115200 に設定してください。
11. Q キーを押して、データ長を 8、パリティを None、ストップビットを 1 に設定してください。
12. Enter キーを 2 回押して、「図 4.4. minicom の設定」に戻ってください。

13. 「 4.4. minicom の設定」から、「Save setup as dfl」を選択し、設定を保存してください。

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

minicom を起動させるには、「 4.9. minicom 起動方法」のようにしてください。

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

図 4.9 minicom 起動方法



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



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

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

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

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

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

図 4.10 minicom 終了確認



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

4.3. インターフェースレイアウト

Armadillo-X2 のインターフェースレイアウトです。各インターフェースの配置場所等を確認してください。

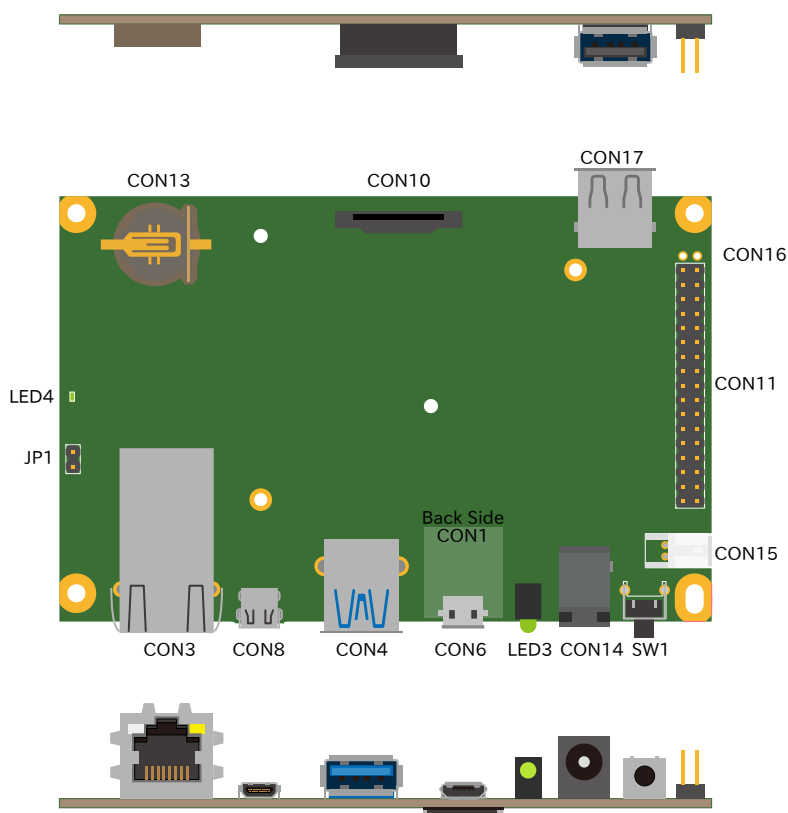


図 4.11 インターフェースレイアウト

表 4.4 インターフェース内容

部品番号	インターフェース名	形状	備考
CON1	SD インターフェース	microSD スロット	-
CON3	LAN インターフェース	RJ-45 コネクタ	-
CON4	USB インターフェース 1	USB 3.0 Type-A コネクタ	-
CON6	USB コンソールインターフェース	USB micro-B コネクタ	-
CON8	HDMI インターフェース	HDMI Type-D コネクタ	-
CON10	MIPI-CSI インターフェース	FFC コネクタ 15 ピン(1mm ピッチ)	挿抜寿命: 30 回 ^[a]
CON11	拡張インターフェース 1	ピンヘッダ 34 ピン(2.54mm ピッチ)	-
CON13	RTC バックアップインターフェース	電池ボックス	対応電池: CR1220 等
CON14	電源入力インターフェース 1	DC ジャック	対応プラグ: 内径 2.1mm 外形 5.5mm
CON15	電源入力インターフェース 2	ピンヘッダ 2 ピン(2mm ピッチ)	-
CON16	3.3V 電源出力インターフェース	ピンヘッダ 2 ピン(2.54mm ピッチ)	-
CON17	USB インターフェース 2	USB 2.0 Type-A コネクタ	-
JP1	起動デバイス設定ジャンパ	ピンヘッダ 2 ピン(2.54mm ピッチ)	-
SW1	ユーザースイッチ	タクトスイッチ	-
LED3	ユーザー LED	LED(緑色、φ3mm)	-
LED4	電源 LED	LED(緑色、面実装)	-

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

4.4. 接続方法

Armadillo-X2 と周辺装置の接続例を「図 4.12. Armadillo-X2 の接続例」に示します。

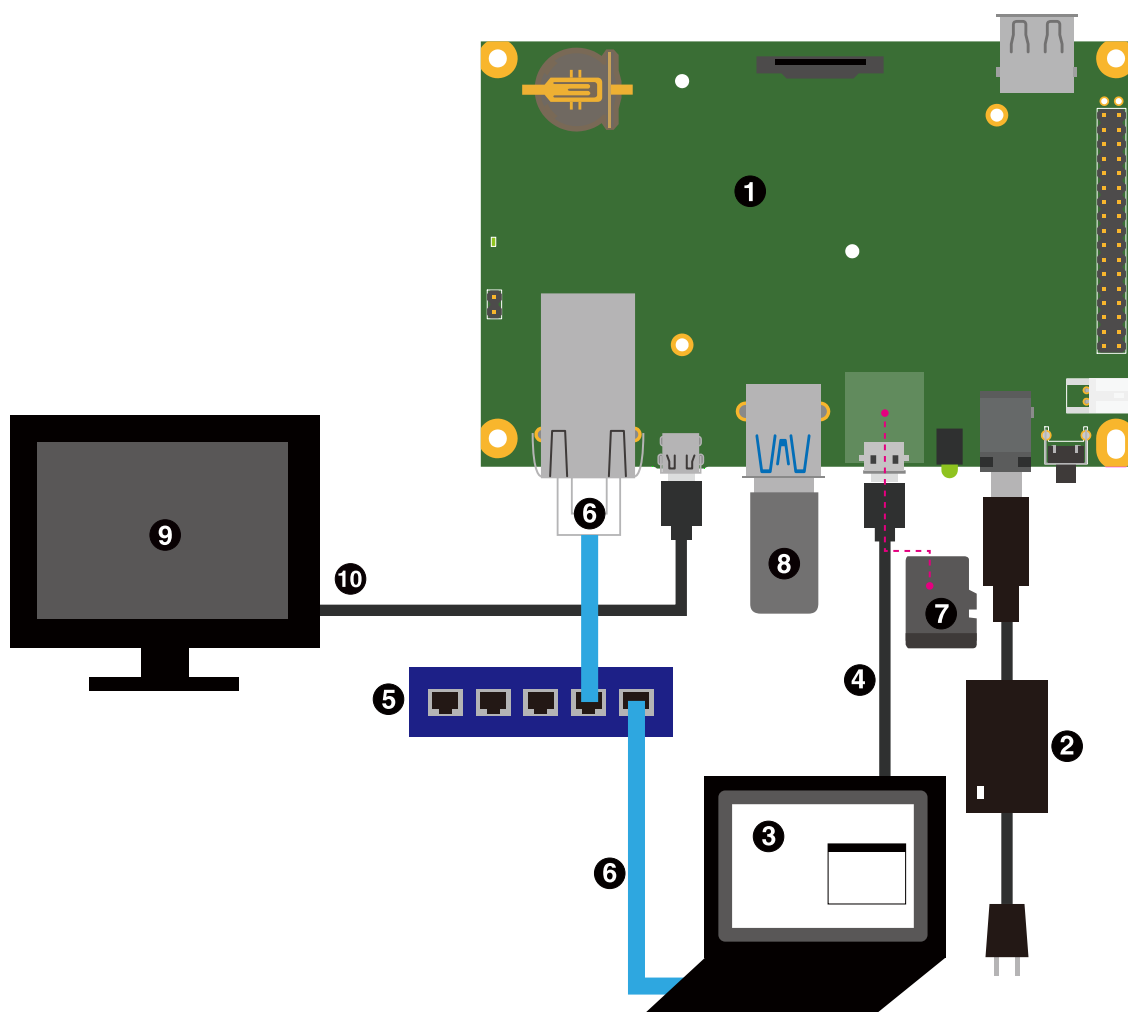


図 4.12 Armadillo-X2 の接続例

- ① Armadillo-X2
- 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)
- ⑤ LAN HUB
- ⑥ Ethernet ケーブル
- ⑦ microSD カード
- ⑧ USB メモリ
- ⑨ ディスプレイ(HDMI 対応)
- ⑩ 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。

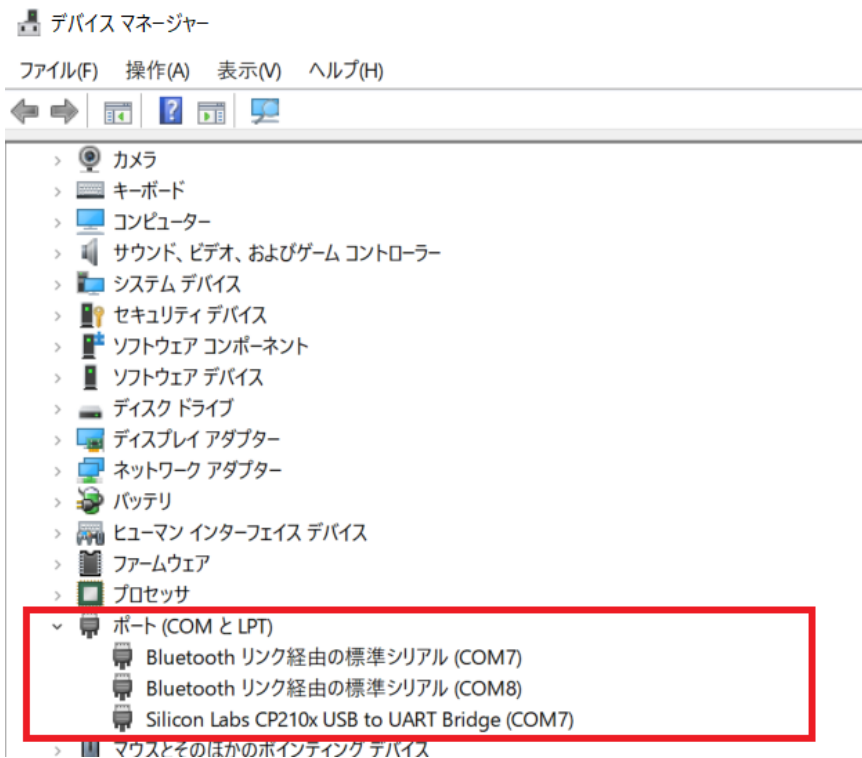


図 4.13 COM7 が競合している状態

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

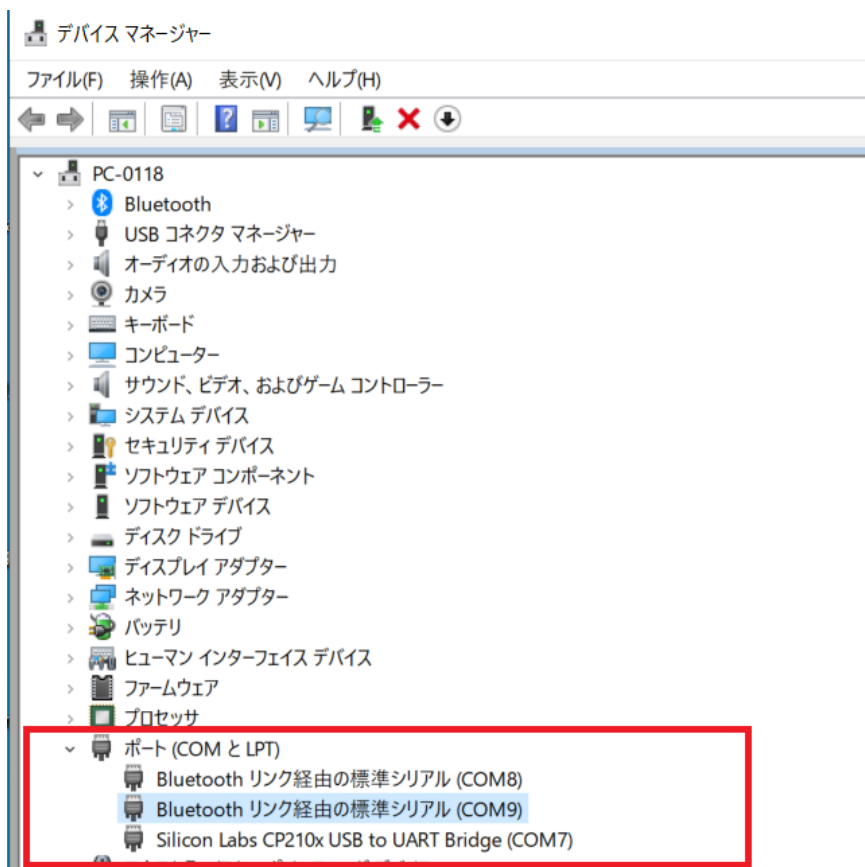


図 4.14 Bluetooth に割当の COM を変更した状態

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

4.5. ジャンパピンの設定について

ジャンパの設定を変更することで、Armadillo-X2 の動作を変更することができます。

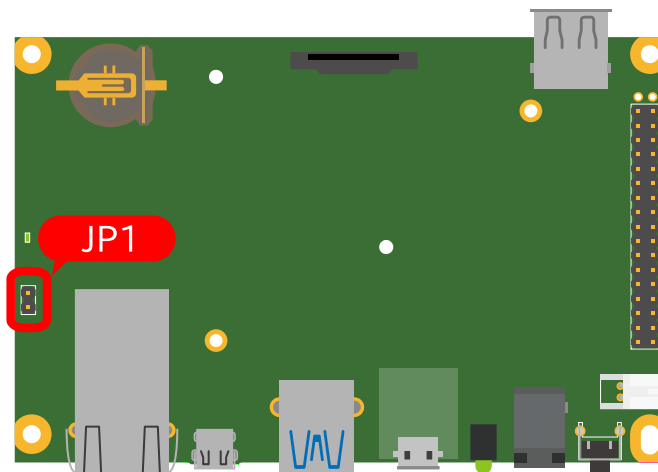


図 4.15 JP1 の位置

表 4.5 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)

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。

4.6. vi エディタの使用方法

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

vi エディタは、ATDE にインストールされてる gedit や emacs などのテキストエディタとは異なり、モードを持っていることが大きな特徴です。vi のモードには、コマンドモードと入力モードがあります。コマンドモードの時に入力した文字はすべてコマンドとして扱われます。入力モードでは文字の入力ができます。

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

4.6.1. vi の起動

vi を起動するには、以下のコマンドを入力します。

```
[ATDE ~]# vi [file]
```

図 4.16 vi の起動

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


4.6.2. 文字の入力

文字を入力するにはコマンドモードから入力モードへ移行する必要があります。コマンドモードから入力モードに移行するには、「表 4.6. 入力モードに移行するコマンド」に示すコマンドを入力します。入力モードへ移行後は、キーを入力すればそのまま文字が入力されます。

表 4.6 入力モードに移行するコマンド

コマンド	動作
i	カーソルのある場所から文字入力を開始
a	カーソルの後ろから文字入力を開始

入力モードからコマンドモードに戻りたい場合は、ESC キーを入力することで戻ることができます。現在のモードが分からなくなった場合は、ESC キーを入力し、一旦コマンドモードへ戻ることにより混乱を防げます。



日本語変換機能を OFF に

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

「i」、「a」それぞれのコマンドを入力した場合の文字入力の開始位置を「図 4.17. 入力モードに移行するコマンドの説明」に示します。

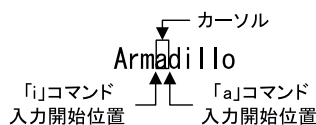



図 4.17 入力モードに移行するコマンドの説明



vi での文字削除

コンソールの環境によっては BS(Backspace)キーで文字が削除できず、「^H」文字が入力される場合があります。その場合は、「4.6.4. 文字の削除」で説明するコマンドを使用し、文字を削除してください。

4.6.3. カーソルの移動

方向キーでカーソルの移動ができますが、コマンドモードで「表 4.7. カーソルの移動コマンド」に示すコマンドを入力することでもカーソルを移動することができます。

表 4.7 カーソルの移動コマンド

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

4.6.4. 文字の削除

文字を削除する場合は、コマンドモードで「表 4.8. 文字の削除コマンド」に示すコマンドを入力します。

表 4.8 文字の削除コマンド

コマンド	動作
x	カーソル上の文字を削除
dd	現在行を削除

「x」コマンド、「dd」コマンドを入力した場合に削除される文字を「図 4.18. 文字を削除するコマンドの説明」に示します。

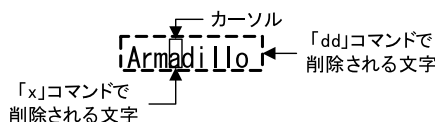


図 4.18 文字を削除するコマンドの説明

4.6.5. 保存と終了

ファイルの保存、終了を行うコマンドを「表 4.9. 保存・終了コマンド」に示します。

表 4.9 保存・終了コマンド

コマンド	動作
:q!	変更を保存せずに終了
:w[file]	ファイルに+file+に指定して保存
:wq	ファイルを上書き保存して終了

保存と終了を行うコマンドは「:」（コロン）からはじまるコマンドを使用します。": "キーを入力すると画面下部にカーソルが移り入力したコマンドが表示されます。コマンドを入力した後 Enter キーを押すことで、コマンドが実行されます。

5. 起動と終了

5.1. 起動

電源入力インターフェースに電源を接続すると Armadillo-X2 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-X2 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「13.2.12. JP1 (起動デバイス設定ジャンパ)」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-at11 (Jan 19 2023 - 10:53:24 +0000)
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):2020.04-at10-0-ge26bfd065
NOTICE: BL31: Built : 10:54:50, Jan 19 2023

U-Boot 2020.04-at11 (Jan 19 2023 - 10:53:24 +0000)

CPU:   i.MX8MP[8] rev1.1 1600 MHz (running at 1200 MHz)
CPU:   Industrial temperature grade (-40C to 105C) at 44C
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 e26bfd0
- U-Boot 2020.04-at11

first boot since power on
switch to partitions #0, OK
mmc2(part 0) is current device
flash target is MMC:2
Net:

Warning: ethernet@30be0000 using MAC address from ROM
eth0: ethernet@30be0000 [PRIME]
```

```

Fastboot: Normal
Normal Boot
Hit any key to stop autoboot: 0
switch to partitions #0, OK
mmc2(part 0) is current device
28962824 bytes read in 628 ms (44 MiB/s)
Booting from mmc ...
78223 bytes read in 3 ms (24.9 MiB/s)
Loading fdt boot/armadillo.dtb
## Flattened Device Tree blob at 45000000
   Booting using the fdt blob at 0x45000000
   Loading Device Tree to 0000000052bbe000, end 0000000052bf4fff ... OK

Starting kernel ...

[   0.521327] fxl6408 2-0043: FXL6408 probe returned DID: 0xfa
[   0.835647] mdio_bus 30be0000.ethernet-1: MDIO device at address 3 is missing.

OpenRC 0.45.2 is starting up Linux 5.10.161-0-at (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 ]
fsck                | * Checking local filesystems ... [ ok ]
[ ok ]
root                | * Remounting filesystems ... [ ok ]
localmount          | * Mounting local filesystems ... [ ok ]
overlays            | * Preparing overlays over / ... [ ok ]
udev-trigger        | * Generating a rule to create a /dev/root symlink ...sysctl
| * Configuring kernel parameters ... [ ok ]
hostname            | * Setting hostname ...udev-trigger          | * Populating /dev with
existing devices through uevents ... [ ok ]
[ ok ]
[ ok ]
bootmisc            | * Migrating /var/lock to /run/lock ... [ ok ]
bootmisc            | * Creating user login records ... [ ok ]
bootmisc            | * Wiping /var/tmp directory ... [ ok ]
syslog              | * Starting busybox syslog ...dbus
creating directory  | * /run/dbus:
dbus                | * /run/dbus: correcting owner
micron-emmc-reten  | * Starting micron-emmc-reten
dbus                | * Starting System Message Bus ... [ ok ]
[ ok ]
klogd               | * Starting busybox klogd ... [ ok ]
networkmanager     | * Starting networkmanager ... [ ok ]

```



```

dnsmasq          | * /var/lib/misc/dnsmasq.leases: creating file
dnsmasq          | * /var/lib/misc/dnsmasq.leases: correcting owner
dnsmasq          | * Starting dnsmasq ... [ ok ]
reset_bootcount  | * Resetting bootcount in bootloader env ...buttond | *
Starting button watching daemon ... [ ok ]
Environment OK, copy 1
reset_bootcount  | [ ok ]
zramswap         | [ ok ]
podman-atmark    | * Starting configured podman containers ...chronyd | *
Starting chronyd ...zramswap          | * Creating zram swap device ... [ ok ]
[ ok ]
[ ok ]
local            | * Starting local ... [ ok ]

Welcome to Alpine Linux 3.17
Kernel 5.10.161-0-at on an aarch64 (/dev/ttyxc1)

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 e26bfd0
- U-Boot 2020.04-at11

reset cause: normal reboot
switch to partitions #0, OK
mmc2(part 0) is current device
flash target is MMC:2
Net:
Warning: ethernet@30be0000 using MAC address from ROM
eth0: ethernet@30be0000 [PRIME]
Fastboot: Normal
Normal Boot
Hit any key to stop autoboot: 0
u-boot=>

```

5.2. ログイン

起動が完了するとログインプロンプトが表示されます。「root」か一般ユーザーの「atmark」でログインすることができます。

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

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

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

1. root でログイン

初期パスワードを変更します。

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

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

2. atmark でログイン

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

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

Welcome to Alpine Linux 3.17
Kernel 5.10.161-0-at on an aarch64 (/dev/ttyxc1)

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。「9.10.2. SWU イメージの作成」を使用した場合には不要です
- ❷ パスワードファイルを永続化します。
- ❸ 設定したパスワードでログインすることができます。



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

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

persist_file コマンドに関する詳細は「9.11.2. overlayfs と persist_file について」を参照してください。

5.3. 終了方法

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

```

armadillo:~# poweroff
armadillo:~# zramswap | * Deactivating zram swap device ...local
| * Stopping local ... [ ok ]
dnsmasq | * Stopping dnsmasq ... [ ok ]rngd | * Stopping rngd ...
podman-atmark | * Stopping all podman containers ...buttond | * Stopping
button watching daemon ... [ ok ]
klogd | * Stopping busybox klogd ...chronyd | * Stopping
chronyd ... [ ok ]
[ ok ]
[ ok ]
[ ok ]
networkmanager | * Stopping networkmanager ...syslog | * Stopping busybox
syslog ... [ ok ]
udev | * Stopping udev ... [ ok ]
[ ok ]
dbus | * Stopping System Message Bus ...nm-dispatcher: Caught signal 15, shutting
down...
[ ok ]
[ ok ]
localmount | * Unmounting loop devices
localmount | * Unmounting filesystems
localmount | * Unmounting /opt/firmware ... [ ok ]
localmount | * Unmounting /var/at-log ... [ ok ]
localmount | * Unmounting /var/tmp ... [ ok ]
localmount | * Unmounting /var/app/volumes ... [ ok ]
localmount | * Unmounting /var/app/rollback/volumes ... [ ok ]
localmount | * Unmounting /var/lib/containers/storage_readonly ... [ ok ]
localmount | * Unmounting /var/log ... [ ok ]
localmount | * Unmounting /tmp ... [ ok ]
killprocs | * Terminating remaining processes ...mount-ro | *
Remounting remaining filesystems read-only ... * Remounting / read only ... [ ok ]
mount-ro | [ ok ]
indicator_signals | * Signaling external devices we are shutting down ... [ ok ]
The system is going down NOW!
Sent SIGTERM to all processes
Sent SIGKILL to all processes
Requesting system poweroff
[ 88.401459] imx2-wdt 30280000.watchdog: Device shutdown: Expect reboot!
[ 88.408763] reboot: Power down

```

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

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



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



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。

6. ユーザー登録

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

ユーザー登録すると次のようなサービスを受けることができます。

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

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

アットマークテクノ Armadillo サイト

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

6.1. 購入製品登録

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

購入製品登録の詳しい手順は以下の URL をご参照ください。

Armadillo-X2 購入製品登録

<https://armadillo.atmark-techno.com/armadillo-x2/register>

7. 動作確認方法

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

ハードウェアの動作確認以外が目的のコマンドや手順については「9. Howto」を参照してください。

7.1. ネットワーク

ここでは、ネットワークの設定方法について説明します。

7.1.1. 接続可能なネットワーク

Armadillo-X2 は、1 つの Ethernet ポートが搭載されています。Linux からは、eth0 に見えます。

表 7.1 ネットワークとネットワークデバイス

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

7.1.2. IP アドレスの確認方法

Armadillo-X2 の IP アドレスを確認するには、ip addr コマンドを使用します。

```
[armadillo ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
    link/ether 00:11:0c:00:0b:79 brd ff:ff:ff:ff:ff:ff
    inet 172.16.1.84/16 brd 172.16.255.255 scope global dynamic noprefixroute eth0
        valid_lft 28786sec preferred_lft 28786sec
    inet6 fe80::e9c0:7b3c:c0c9:3c4/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

図 7.1 IP アドレスの確認

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

```
[armadillo ~]# ip addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
    link/ether 00:11:0c:00:0b:79 brd ff:ff:ff:ff:ff:ff
    inet 172.16.1.84/16 brd 172.16.255.255 scope global dynamic noprefixroute eth0
        valid_lft 28656sec preferred_lft 28656sec
```

```
inet6 fe80::e9c0:7b3c:c0c9:3c4/64 scope link noprefixroute
    valid_lft forever preferred_lft forever
```

図 7.2 IP アドレス(eth0)の確認

7.1.3. ネットワークの設定方法

Armadillo-X2 では、通常の Linux システムと同様、ネットワークインターフェースの設定は NetworkManager を使用します。NetworkManager はすべてのネットワーク設定をコネクションとして管理します。コネクションには「どのようにネットワークへ接続するか」、「どのようにネットワークを作成するか」を記述し、/etc/NetworkManager/system-connections/ に保存します。また、1つのデバイスに対して複数のコネクションを保存することは可能ですが、1つのデバイスに対して有効化にできるコネクションは1つだけです。

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

7.1.3.1. nmcli について

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

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

図 7.3 nmcli のコマンド書式

7.1.4. nmcli の基本的な使い方

ここでは nmcli の、基本的な使い方を説明します。

7.1.4.1. コネクションの一覧

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

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

図 7.4 コネクションの一覧

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

7.1.4.2. コネクションの有効化・無効化

コネクションを有効化するには、次のようにコマンドを実行します。

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

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

図 7.5 コネクションの有効化

コネクションを無効化するには、次のようにコマンドを実行します。

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

図 7.6 コネクションの無効化

7.1.4.3. コネクションの作成

コネクションを作成するには、次のようにコマンドを実行します。

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

図 7.7 コネクションの作成

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

Armadillo-X2 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「9.11.2. overlays と persist_file について」を参照してください。

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

図 7.8 コネクションファイルの永続化



別の Armadillo-X2 からコネクションファイルをコピーした場合は、コネクションファイルのパーミッションを 600 に設定してください。600 に設定後、nmcli c reload コマンドでコネクションファイルを再読み込みします。

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

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

マンド実行手順は不要です。swu イメージに関しては「9.10. Armadillo のソフトウェアをアップデートする」を参考にしてください。

7.1.4.4. コネクションの削除

コネクションを削除するには、次のようにコマンドを実行します。

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

図 7.9 コネクションの削除

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

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

図 7.10 コネクションファイル削除時の永続化

7.1.4.5. 固定 IP アドレスに設定する

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

表 7.2 固定 IP アドレス設定例

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

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

図 7.11 固定 IP アドレス設定

7.1.4.6. DNS サーバーを指定する

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

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

図 7.12 DNS サーバーの指定

7.1.4.7. DHCP に設定する

DHCP に設定する例を、「図 7.13. DNS サーバーの指定」に示します。

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

図 7.13 DNS サーバーの指定



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

7.1.4.8. コネクションの修正を反映する

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

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

図 7.14 コネクションの修正の反映

7.1.4.9. デバイスの一覧

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

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

図 7.15 デバイスの一覧

7.1.4.10. デバイスの接続

デバイスを接続するには、次のようにコマンドを実行します。

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

図 7.16 デバイスの接続



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

7.1.4.11. デバイスの切断

デバイスを切断するには、次のようにコマンドを実行します。

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

図 7.17 デバイスの切断

7.1.5. 有線 LAN

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

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

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

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

図 7.18 有線 LAN の PING 確認

❶ -I オプションでインターフェースを指定できます。



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

7.2. ストレージ

Armadillo-X2 でストレージとして使用可能なデバイスを次に示します。

表 7.3 ストレージデバイス

デバイス種類	ディスクデバイス	先頭パーティション	インターフェース
オンボード eMMC	/dev/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, CON17)

^[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-X2 では、8 MiB の GPP を 4 つ作成しています。各領域の用途を「表 7.4. eMMC の GPP の用途」に示します。

表 7.4 eMMC の GPP の用途

ディスクデバイス	用途
/dev/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「10.4. ログ用パーティションについて」を参照ください。

7.2.1. ストレージの使用方法

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

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

mount コマンドの典型的なフォーマットは、次の通りです。

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

図 7.19 mount コマンド書式

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



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

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

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

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 7.20 ストレージのマウント

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

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

図 7.21 ストレージのアンマウント

7.2.2. ストレージのパーティション変更とフォーマット

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

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

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

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

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

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

```

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

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

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

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

```

図 7.22 fdisk コマンドによるパーティション変更

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

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

図 7.23 EXT4 ファイルシステムの構築

7.3. LED

Armadillo-X2 の LED は GPIO で接続されているため、ソフトウェアで制御することができます。

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

表 7.5 LED クラスディレクトリと LED の対応

LED クラスディレクトリ	インターフェース	デフォルトトリガ
/sys/class/leds/led1/	ユーザー LED 緑	none

7.3.1. LED を点灯/消灯する

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

brightness に 0 以外の値を書き込むと LED が点灯します。

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

図 7.24 LED を点灯させる



Armadillo-X2 の LED には輝度制御の機能がないため、0(消灯)、1~255(点灯)の2つの状態のみ指定することができます。

brightness に 0 を書き込むと LED が消灯します。

```
[armadillo ~]# echo 0 > /sys/class/leds/led1/brightness
```

図 7.25 LED を消灯させる

brightness を読み出すと LED の状態が取得できます。

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

図 7.26 LED の状態を表示する

7.3.2. トリガを使用する

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

表 7.6 LED トリガの種類

設定	説明
none	トリガを設定しません
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。

```
[armadillo ~]# 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-shiftrllock kbd-shiftrlock kbd-ctrllock kbd-ctrlrlock rfkill0 rfkill1 di
sk-activity disk-read disk-write ide-disk heartbeat cpu cpu0 cpu1 cpu2 cpu3 mmc2
default-on panic mmc1
```

図 7.27 対応している LED トリガを表示

以下のコマンドを実行すると、心拍のように点灯/消灯を行います。

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

図 7.28 LED のトリガに heartbeat を指定する

7.4. ユーザースイッチ

Armadillo-X2 のユーザースイッチのデバイスドライバは、インプットデバイスとして実装されています。インプットデバイスのデバイスファイルからボタンプッシュ/リリースイベントを取得することができます。

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

表 7.7 インプットデバイスファイルとイベントコード

ユーザースイッチ	インプットデバイスファイル	イベントコード
SW1	/dev/input/by-path/platform-gpio-keys-event	148 (KEY_PROG1)



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

7.4.1. イベントを確認する

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

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

図 7.29 evtest コマンドのインストール

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



```
Event: time 1638343703.991022, type 1 (EV_KEY), code 148 (KEY_PROG1), value 0 ②  
Event: time 1638343703.991022, ----- SYN_REPORT -----
```

図 7.30 ユーザスイッチ: イベントの確認

- ① SW1 のボタン プッシュ イベントを検出したときの表示
- ② SW1 のボタン リリース イベントを検出したときの表示

8. 開発の基本的な流れ

8.1. アプリケーション開発の流れ

8.1.1. Armadillo への接続

8.1.1.1. シリアルコンソール

Armadillo-X2 では CON6 (USB コンソールインターフェース)をシリアルコンソールとして使用できます。シリアル通信設定等については、「4.2.4. シリアル通信ソフトウェア(minicom)の使用」を参照してください。

ログイン方法については、「5.2. ログイン」を参照してください。

8.1.1.2. ssh

Armadillo-X2 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。

8.1.2. overlayfs の扱い

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

その為、ファイルを変更した後 Armadillo の電源を切ると変更内容は保持されません。開発中などに rootfs の変更内容を保持するには、変更したファイルに対して `persist_file` コマンドを使用します。`persist_file` コマンドの詳細は「9.11.2. overlayfs と `persist_file` について」を参照してください。

開発以外の時は安全のため、ソフトウェアアップデートによる更新を実行してください。アップデート手順に関しては「9.10. Armadillo のソフトウェアをアップデートする」を参照してください。

rootfs の内容を変更しても、ソフトウェアアップデートを実施した際に変更した内容が保持されない可能性があります。ソフトウェアアップデート実施後も変更内容を保持する手順に関しては「9.10.7. `swupdate_preserve_files` について」を参照してください。

8.1.3. Podman のデータを eMMC に保存する

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

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

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

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

```
[armadillo ~]# abos-ctrl podman-storage --disk
Creating configuration for persistent container storage
Create subvolume '/mnt/containers_storage'
[ 2145.288677] EXT4-fs (mmcblk2p1): re-mounted. Opts: (null)
[armadillo ~]# abos-ctrl podman-storage --status
Currently in disk mode, run with --tmpfs to switch
```



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

コンテナのデータをボリュームに入れたら消えません。詳しくは「9.4.2.6. コンテナの変更を保存する」を参照してください。

8.1.4. ベースとなるコンテナを取得する

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

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

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

8.1.5. デバイスのアクセス権を与える

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

podman_start のコンテナコンフィグに add_devices コマンドでデバイスファイルを指定します。詳細は「9.5.1. コンテナの自動起動」を参照ください。



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

8.1.6. アプリケーションを作成する

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

8.1.7. コンテナやデータを保存する

ログやデータベース、自分のアプリケーションのデータを保存する場合にボリュームを使ってください。実行中のコンテナイメージを保存するには `podman commit` コマンドで保存してください。詳しい手順は「9.4.2.6. コンテナの変更を保存する」を参考にしてください。

8.2. アプリケーションコンテナの運用

「9.5. コンテナの運用」を参考にしてください。

8.2.1. アプリケーションの自動起動

`podman_start` 用の設定ファイル (`/etc/atmark/containers/*.conf`) を作成します。その後、`podman_start -a` コマンドを実行するか、`armadillo` を再起動してコンテナが自動起動することを確認してください。コンテナの自動起動に関する詳しい説明は「9.5.1. コンテナの自動起動」を参考してください。

8.2.2. アプリケーションの送信

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

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

手順の詳しい説明やオプションは「9.10. Armadillo のソフトウェアをアップデートする」を参考にしてください。

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

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

ここで作成した `initial_setup_container.swu` ファイルを USB メモリに配置して、Armadillo-X2 に刺すとインストールされます。

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

8.2.3. インストール確認：初期化

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

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

8.2.4. アプリケーションのアップデート

アップデートを行う方法は以下の二通りです：

1. podman run コマンドで、差分アップデートを行う

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

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

出来上がった myimage2_update.tar は普通のコンテナと同じように扱うことができます。myimage:1 が存在しない場合はエラーとなります。

詳しいコンテナアップデートの手順は「9.4.2.7. コンテナの自動作成やアップデート」を参考にしてください。

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

2. コンテナを新しく構築する

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

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

自動的にイメージを作る方法は「9.4.2.7. コンテナの自動作成やアップデート」を参考にしてください。

8.3. 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。

8.3.1. ATDE にクロスコンパイル用ライブラリをインストールする

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 8.1 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 8.2 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。

8.3.2. 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、「9.8.1. ブートディスクの作成」または「9.9.1.1. 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 8.1 ライブラリイメージ書き込み済みの製品

名称	型番
Armadillo-X2 開発セット(メモリ 2GB)	AX2210-U00D0
Armadillo-X2 量産ボード(メモリ 2GB、ストレージ 10GB)	AX2210-U00Z
Armadillo-X2 量産ボード(メモリ 2GB、ストレージ 10GB、ケース入り)	AX2210-C00Z

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 8.3 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 8.4 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。

8.3.3. Armadillo にライブラリイメージを書き込む

Armadillo Base OS 上で、「8.3.2. 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 8.5 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 8.6 ライブラリパーティションのマウント

8.3.4. 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 8.7 ライブラリバージョンの確認



「図 8.7. 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。

8.3.5. 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。

- ② `--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 8.8 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。

9. Howto

9.1. GUI アプリケーションを開発する

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。

9.1.1. 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 9.1 Flutter アプリケーションの例

9.1.2. 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。

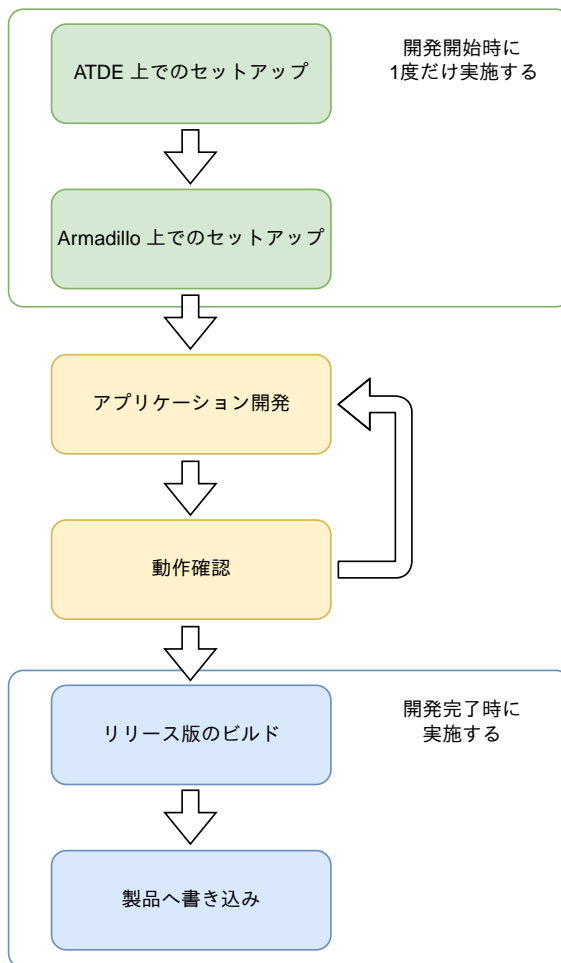


図 9.2 Flutter アプリケーション開発の流れ

9.1.3. 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「4.2.1. ATDE のセットアップ」を参照して ATDE のセットアップを完了してください。

9.1.3.1. 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、Flutter 開発環境と 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。

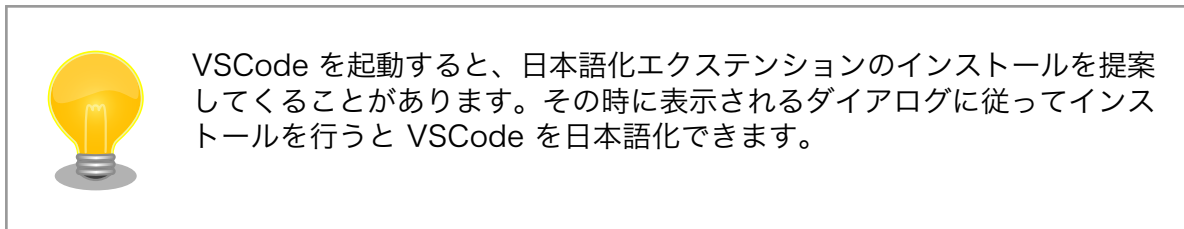
```
[ATDE ~]$ sudo apt update
[ATDE ~]$ sudo apt upgrade
```

図 9.3 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 9.4 VSCode を起動する



9.1.3.2. 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。

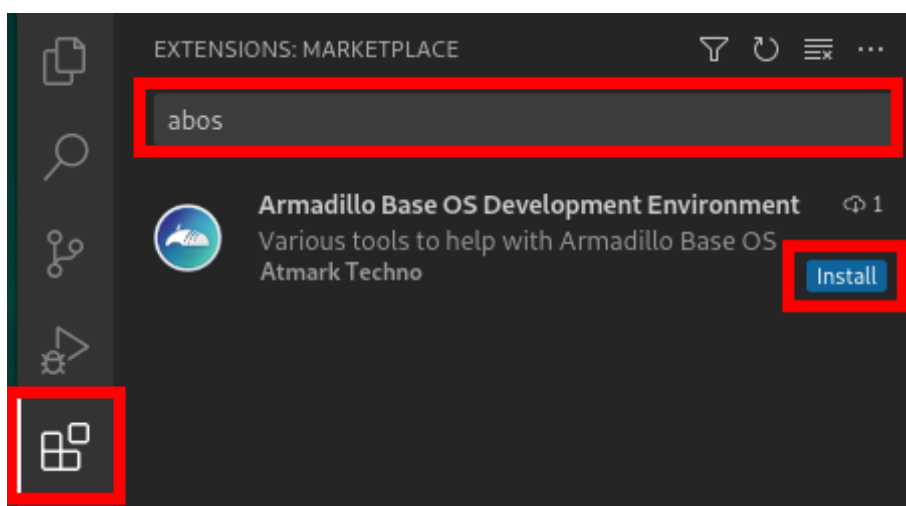


図 9.5 VSCode に開発用エクステンションをインストールする

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。

9.1.3.3. クロスコンパイル用ライブラリをインストール

「8.3.1. ATDE にクロスコンパイル用ライブラリをインストールする」を参照して、クロスコンパイル用のライブラリをインストールしてください。

9.1.3.4. 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。

- ・ GUI アプリケーション
- ・ Signage アプリケーション



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 GUI アプリケーションを作成する場合は [GUI New Project] を、Signage アプリケーションを作成する場合は [Signage New Project] 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。

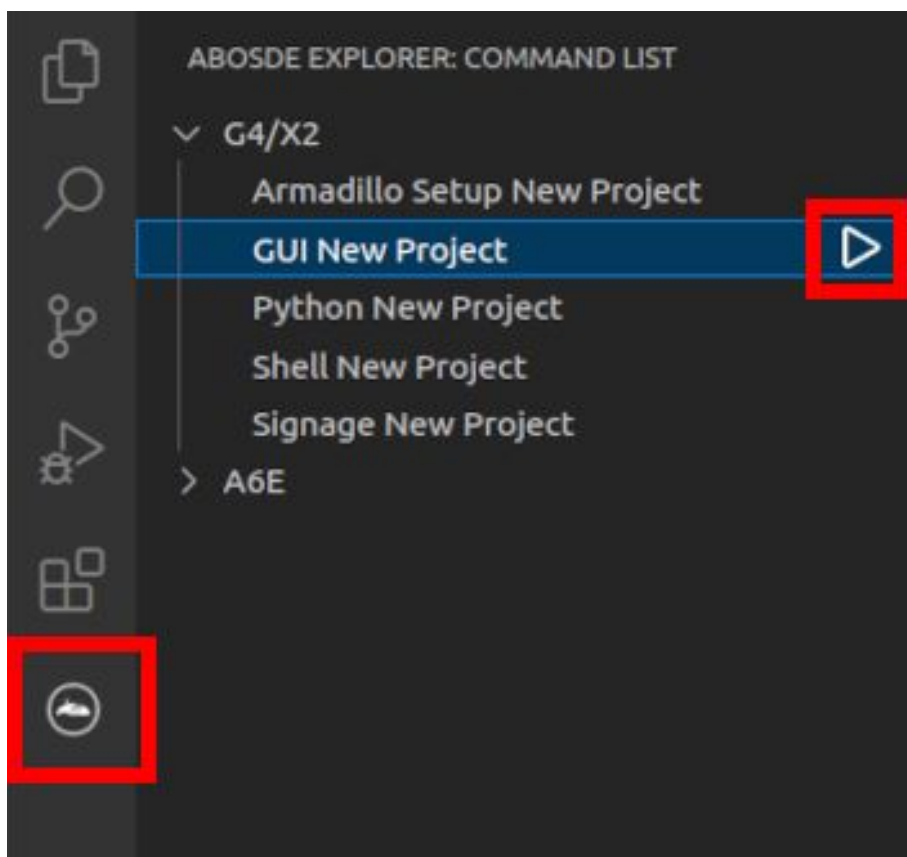


図 9.6 GUI アプリケーションのプロジェクトを作成する

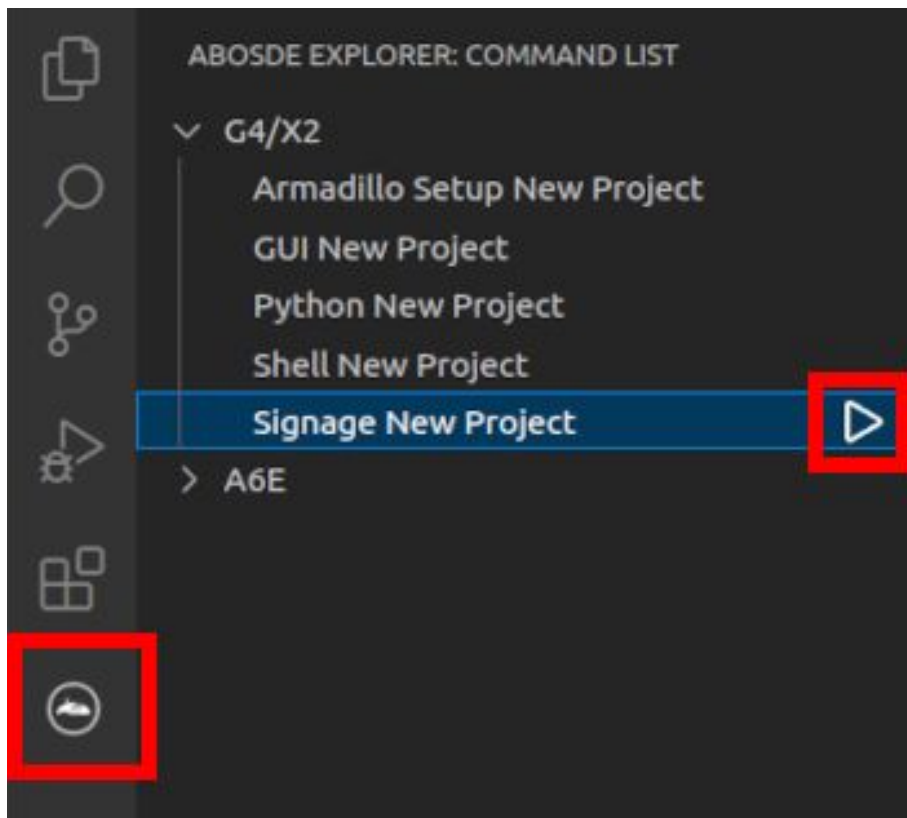


図 9.7 Signage アプリケーションのプロジェクトを作成する

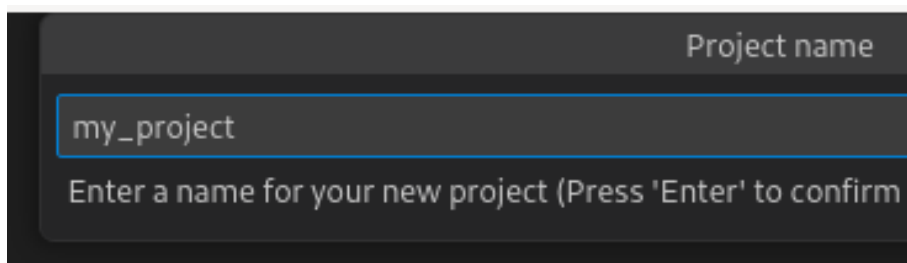


図 9.8 プロジェクト名を入力する

9.1.3.5. 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 9.9 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。

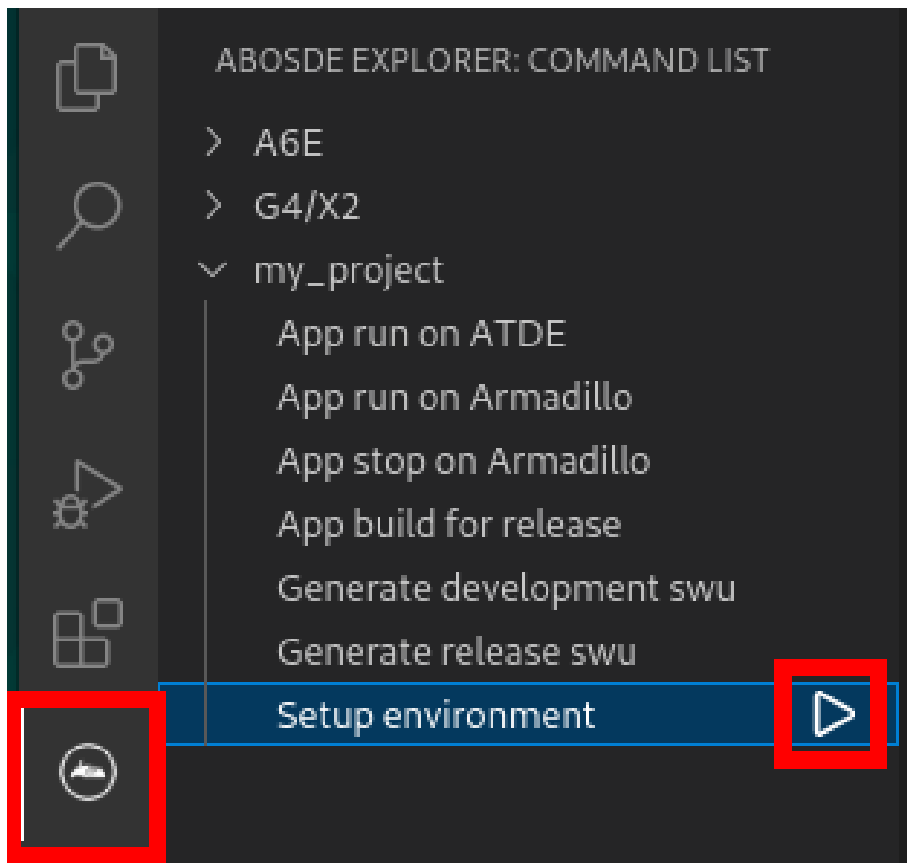


図 9.10 VSCode で初期設定を行う

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。

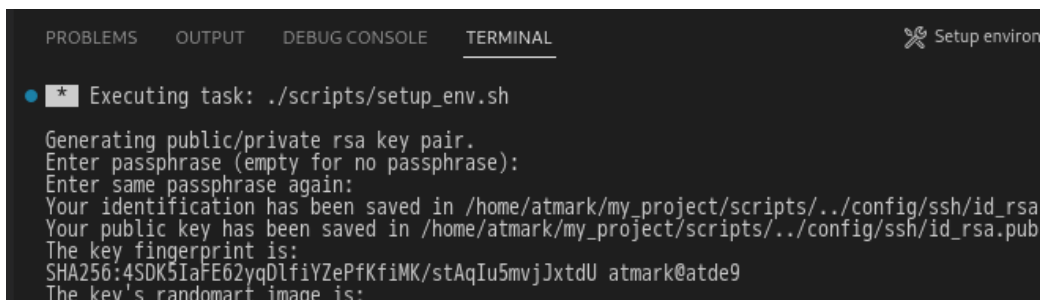


図 9.11 VSCode のターミナル

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. ②

図 9.12 SSH 用の鍵を生成する

- ① パスフレーズを設定します。設定しない場合は何も入力せず Enter を押します。
- ② 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。

9.1.3.6. 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 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。

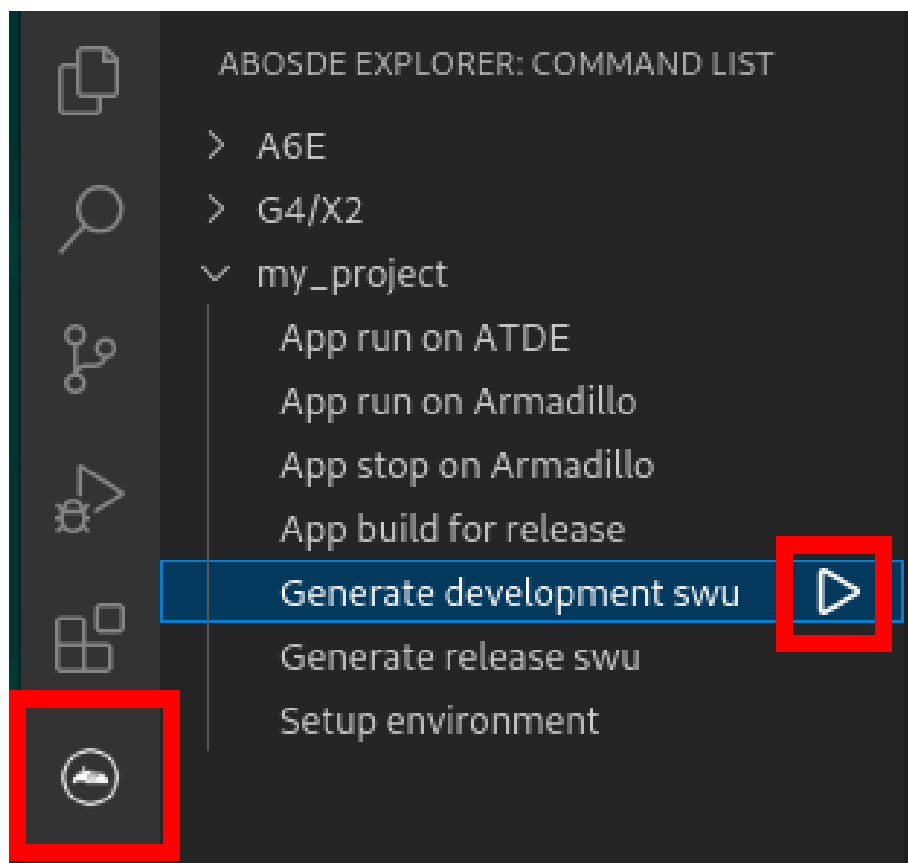


図 9.13 VSCode でコンテナイメージの作成を行う

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。

```
gui_app_dev_image.swu を作成しました。  
* Terminal will be reused by tasks, press any key to close it.
```

図 9.14 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/container/dev` ディレクトリ下に `gui_app_dev_image.swu` というファイル名で保存されています。

9.1.4. Armadillo 上でのセットアップ

9.1.4.1. ディスプレイの接続

「4.4. 接続方法」を参照して Armadillo にディスプレイを接続してください。

9.1.4.2. 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

「9.1.3.6. 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 `gui_app_dev_image.swu` を「9.10.3. イメージのインストール」を参照して Armadillo へインストールしてください。

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。

9.1.5. アプリケーション開発

9.1.5.1. 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 9.15 my_project へ移動して VSCode を起動する。

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

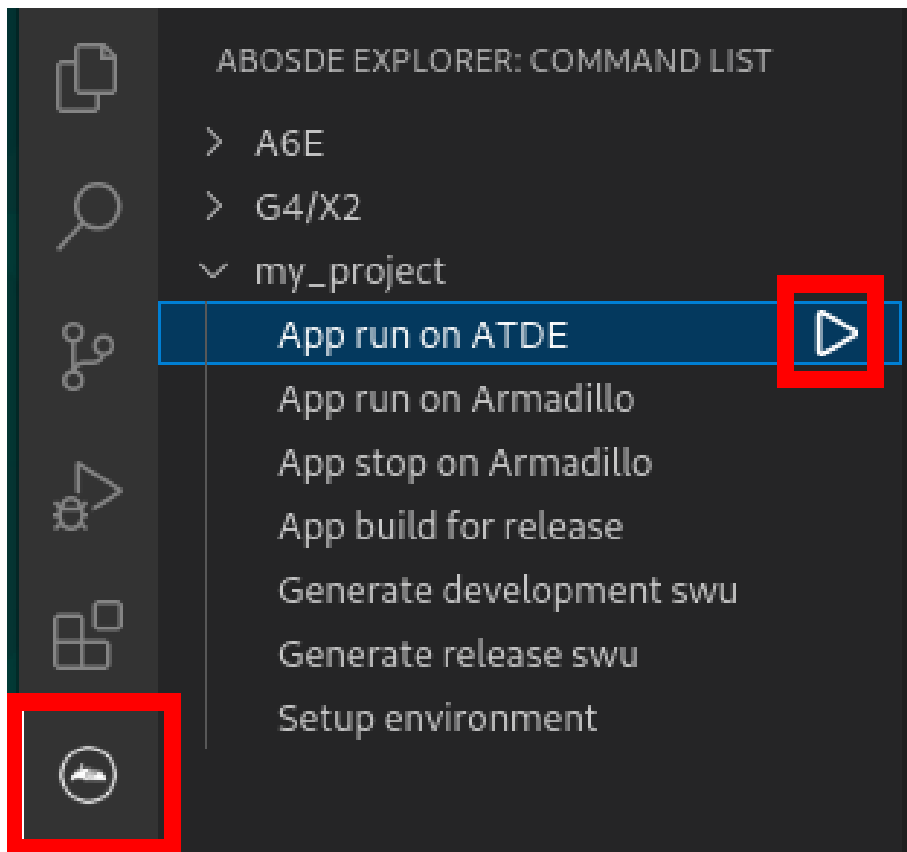



図 9.16 ATDE 上でアプリケーションを実行する

 flutter-elinix をインス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。

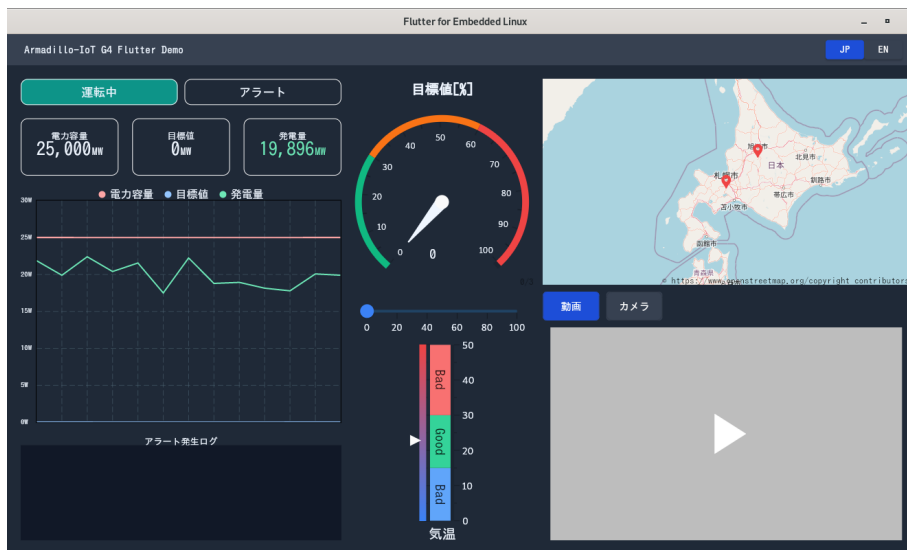


図 9.17 起動したサンプルアプリケーション

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。

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。

9.1.5.2. 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]$ flutter-elixir pub add dart_periphery
```

図 9.18 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-elixir-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 9.1 組み合わせて使うパッケージ

パッケージ名	/opt/flutter-elixir-package 内のパッケージ名
video_player	video_player_elixir
camera	camera_elixir
path_provider	path_provider_elixir
shared_preferences	shared_preferences_elixir
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]$ flutter-elixir pub add video_player
[ATDE ~/my_project]$ flutter-elixir pub add video_player_elixir ¥
--path /opt/flutter-elixir-plugins/packages/video_player
```

図 9.19 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]$ flutter-elixir pub remove dart_periphery
```

図 9.20 dart_periphery パッケージをアンインストールする例

9.1.6. 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。

9.1.6.1. ssh_config の準備

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

```
[ATDE ~/my_project]$ code config/ssh_config
Host Armadillo
  Hostname 0.0.0.0 ❶
  User root
  Port 2222
  IdentityFile ../config/ssh/id_rsa
```

図 9.21 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。

9.1.6.2. 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。

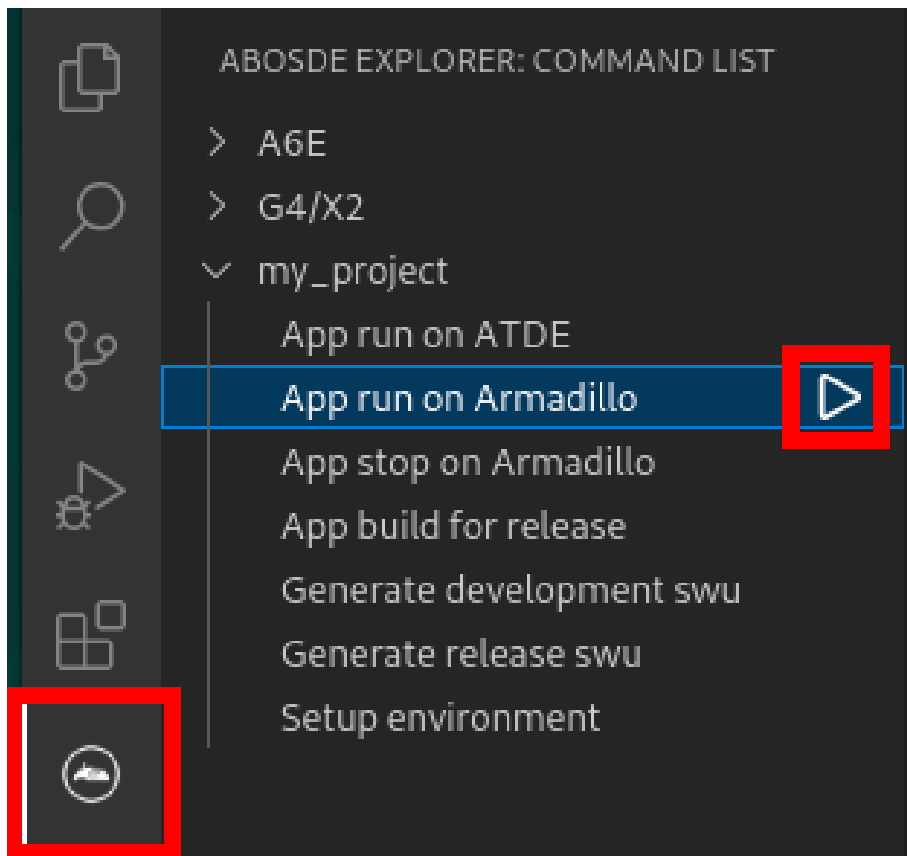


図 9.22 Armadillo 上でアプリケーションを実行する

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 9.23 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。

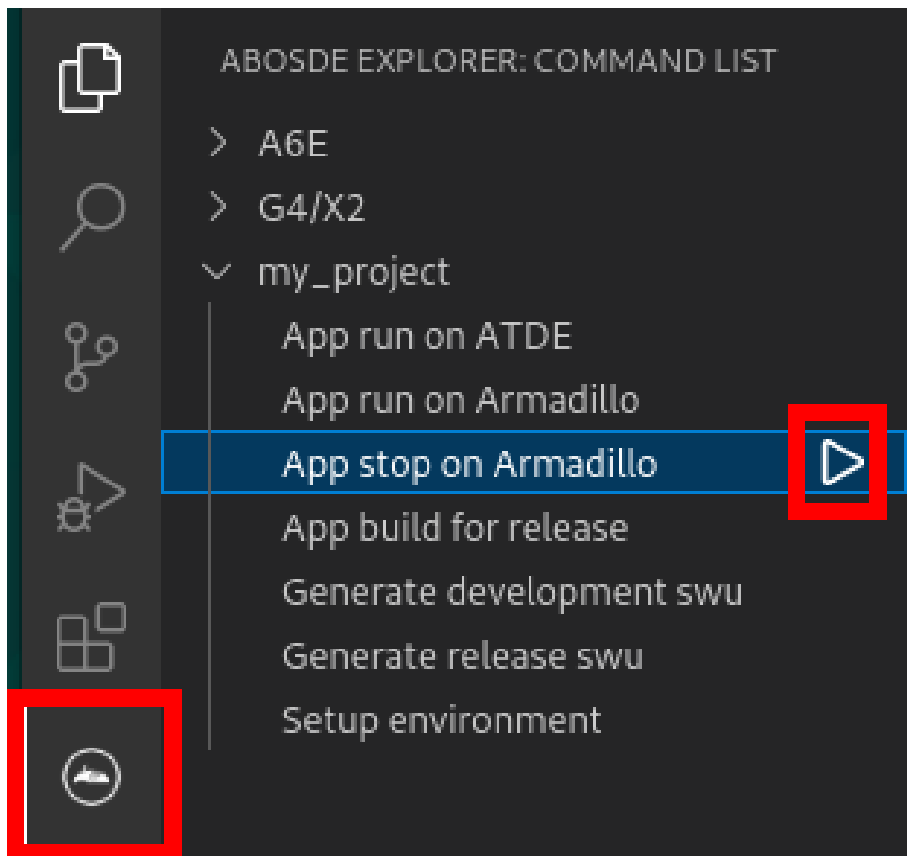


図 9.24 アプリケーションを終了する

9.1.6.3. 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。

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 9.25 ホットリロード機能を使う

9.1.7. 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 [App build for release] を実行すると、リリース版のアプリケーションがビルドされます。

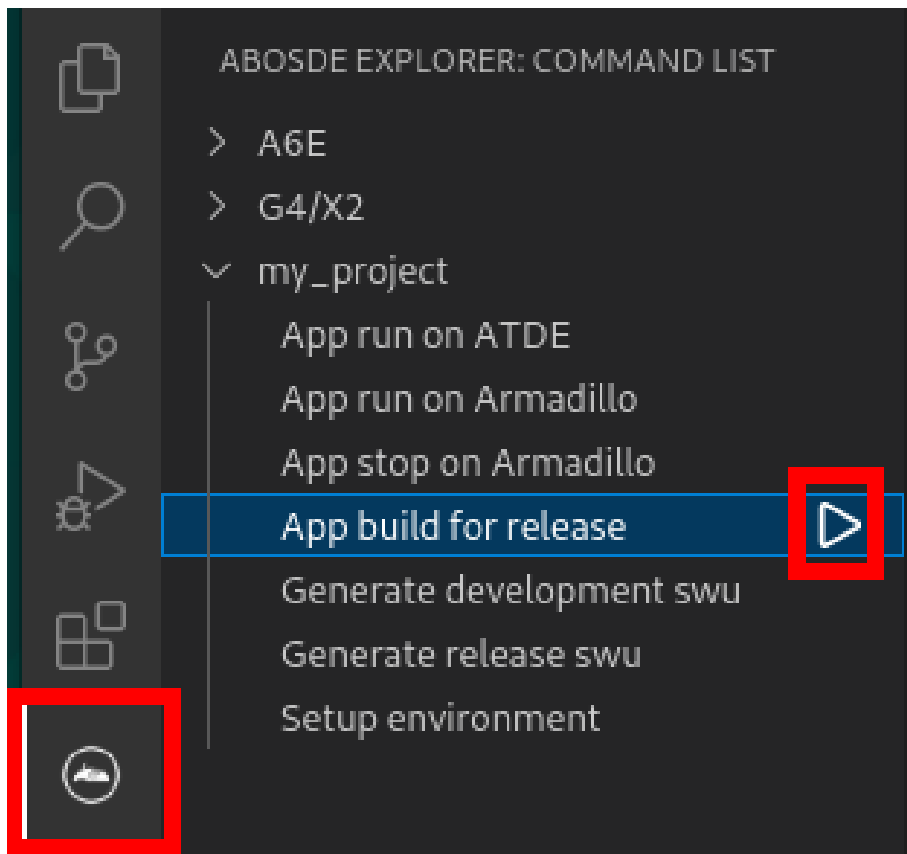


図 9.26 リリース版をビルドする

実行ファイルを含むビルド生成物は `app/build/elixir/arm64/release/bundle/` ディレクトリ下に保存されています。

9.1.8. 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「9.10.2. SWU イメージの作成」を参照して SWU の初期設定を行ってください。

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。

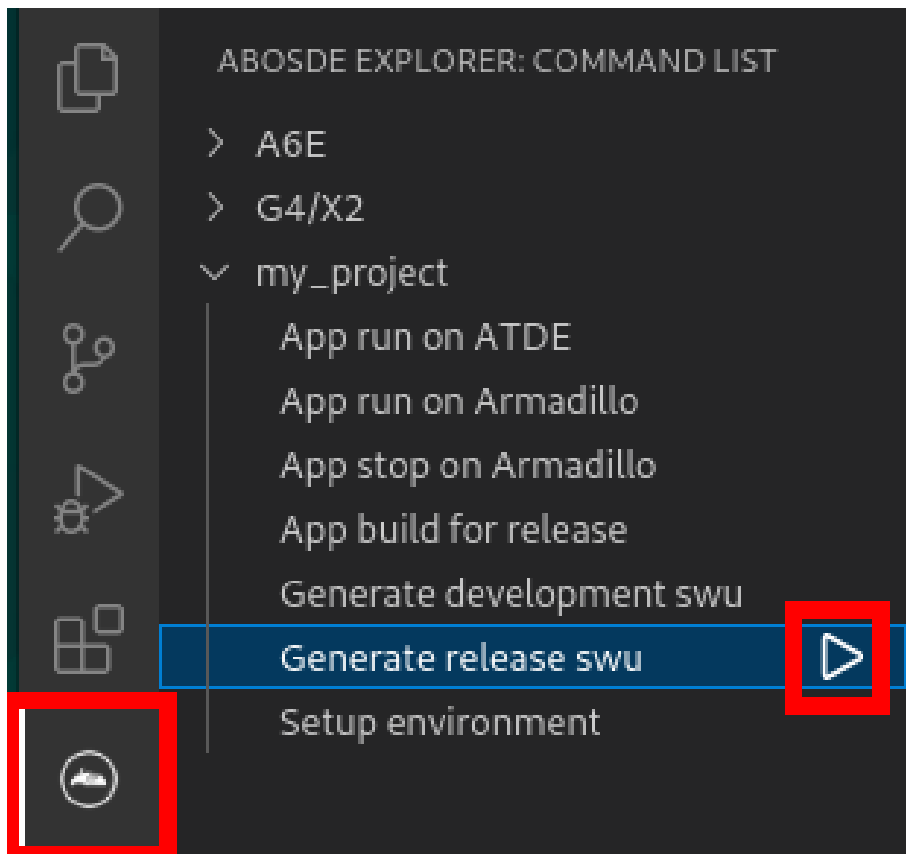


図 9.27 SWU イメージを作成する

作成した SWU イメージは `my_project/container/release` ディレクトリ下に `gui_app_rel_image.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「9.10.3. イメージのインストール」を参照して Armadillo へインストールすると、Armadillo 起動時にアプリケーションも自動起動します。

9.2. CUI アプリケーションを開発する

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

9.2.1. 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。

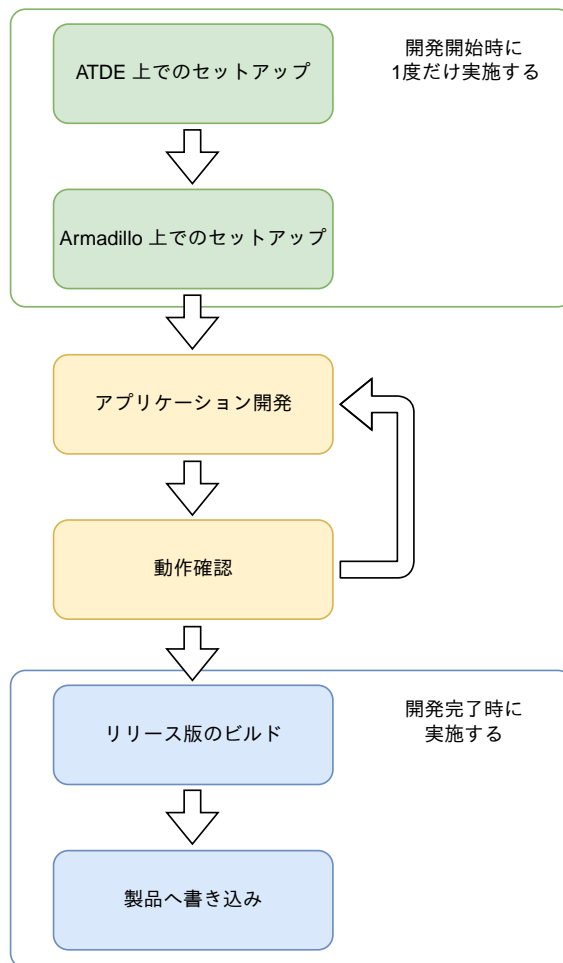


図 9.28 CUI アプリケーション開発の流れ

9.2.2. ATDE 上でのセットアップ

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

9.2.2.1. ソフトウェアのアップデート

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

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

図 9.29 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 9.30 VSCode を起動する



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

9.2.2.2. VSCoDe に開発用エクステンションをインストールする

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

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

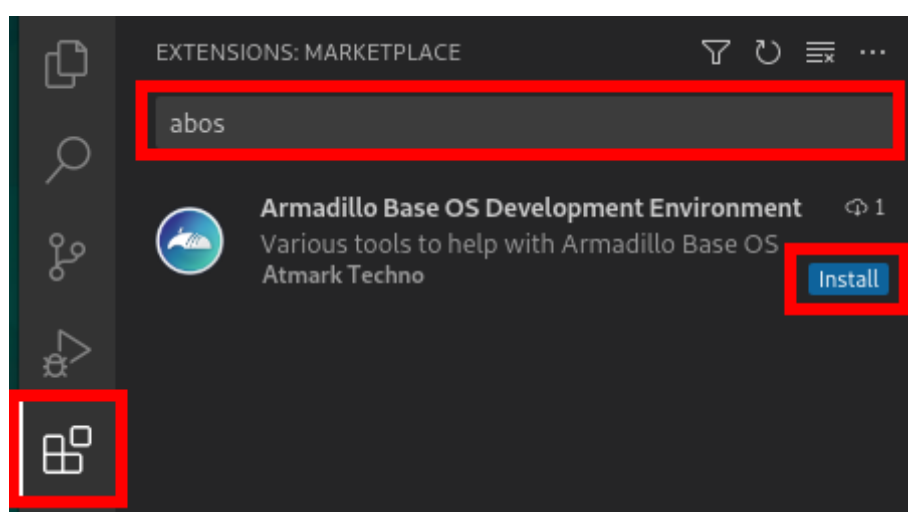


図 9.31 VSCoDe に開発用エクステンションをインストールする

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。

9.2.2.3. 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。

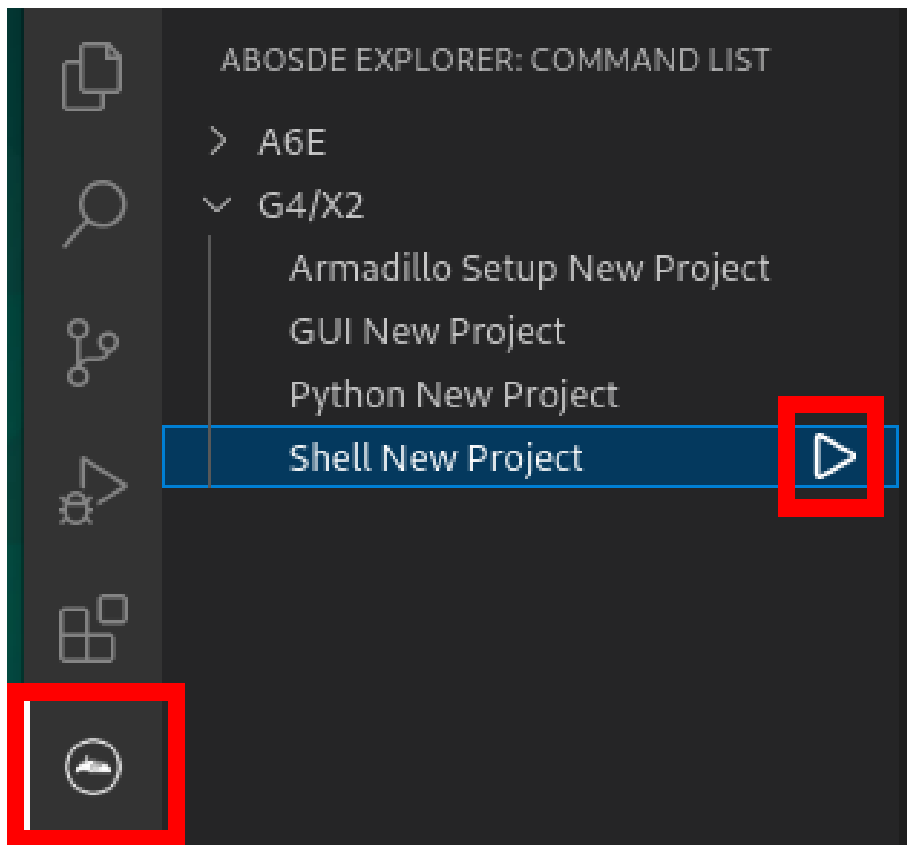


図 9.32 プロジェクトを作成する

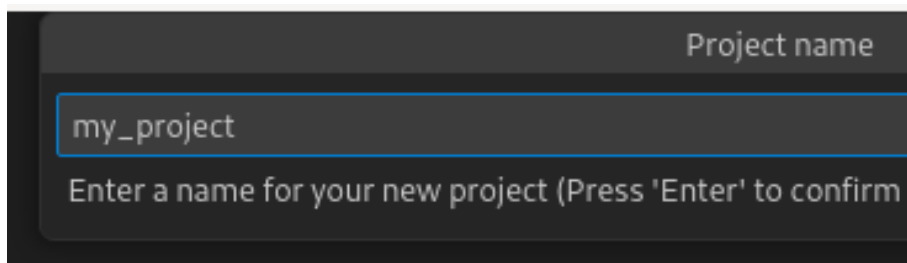


図 9.33 プロジェクト名を入力する

9.2.2.4. 初期設定

初期設定では主に Armadillo と SSH で接続するための秘密鍵と公開鍵の生成を行います。
作成したプロジェクトディレクトリへ移動して VSCode を起動してください。

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

図 9.34 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。

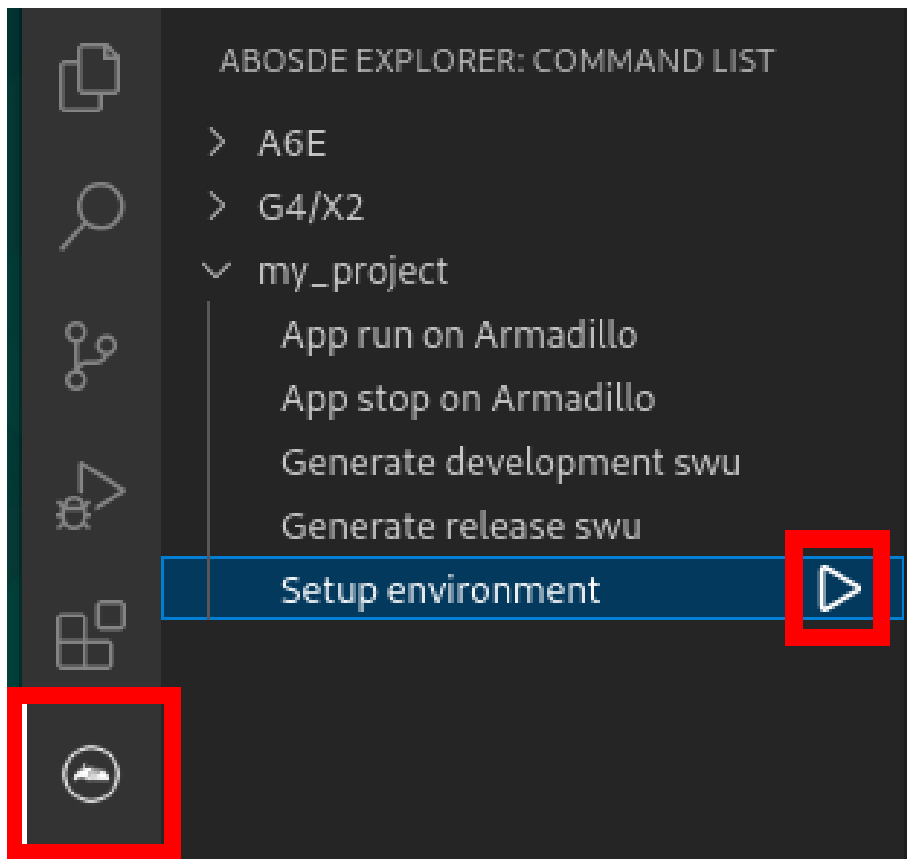


図 9.35 VSCode で初期設定を行う

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。

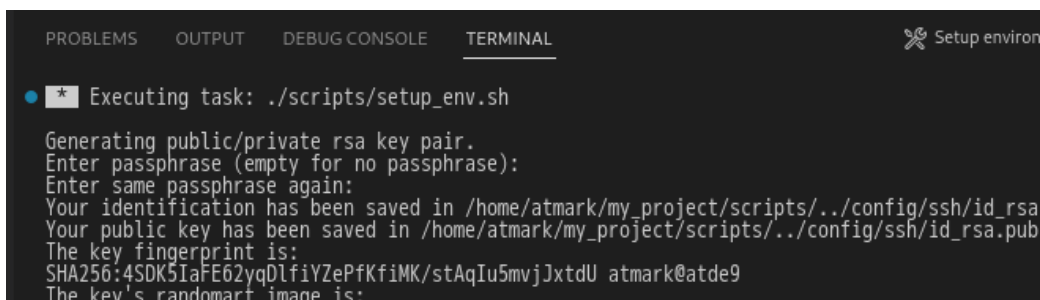


図 9.36 VSCode のターミナル

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: ❷
Your identification has been saved in /home/atmark/my_project/scripts/./config/ssh/id_rsa
Your public key has been saved in /home/atmark/my_project/scripts/./config/ssh/id_rsa.pub
:(省略)
```

* Terminal will be reused by tasks, press any key to close it. ③

図 9.37 SSH 用の鍵を生成する

- ① パスフレーズを設定します。設定しない場合は何も入力せず Enter を押します。
- ② 1 でパスフレーズを設定した場合は、確認のため再度入力してください。
- ③ ここで何か任意のキーを押すとターミナルが閉じます。

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

9.2.2.5. アプリケーション実行用コンテナイメージの作成

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

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

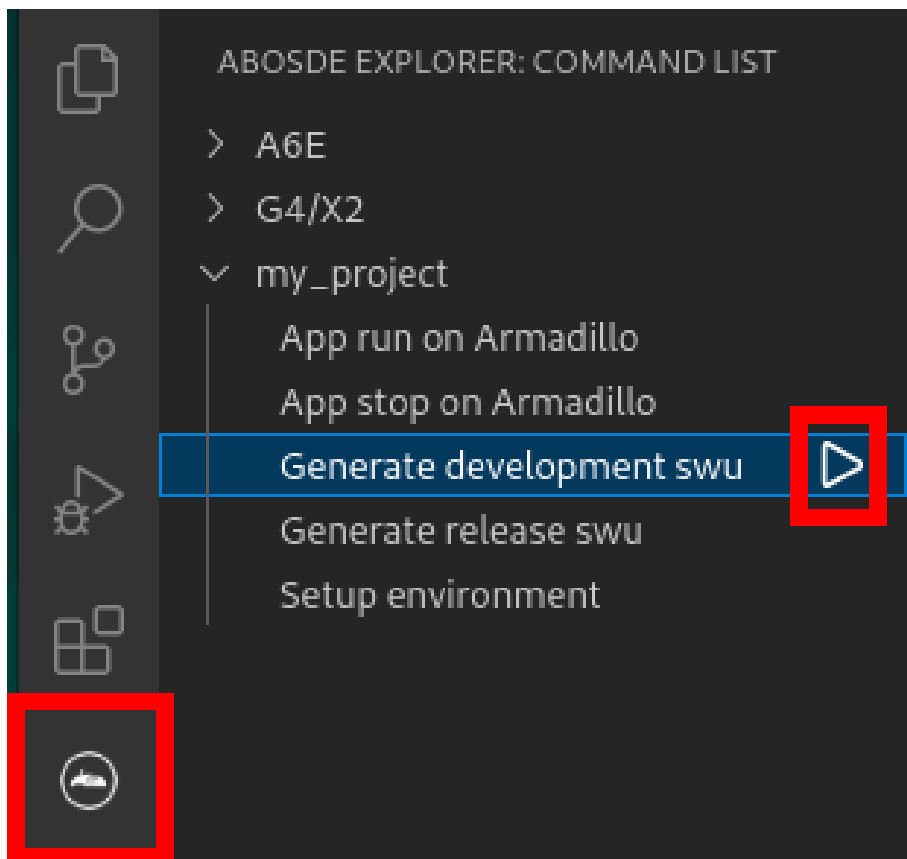


図 9.38 VSCode でコンテナイメージの作成を行う

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。

```
shell_app_dev_image.swu を作成しました。  
* Terminal will be reused by tasks, press any key to close it.
```

図 9.39 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/container_arm64/dev` ディレクトリ下に `shell_app_dev_image.swu` というファイル名で保存されています。Python の場合は `python_app_dev_image.swu` というファイル名になります。

9.2.3. Armadillo 上でのセットアップ

9.2.3.1. アプリケーション実行用コンテナイメージのインストール

「9.2.2.5. アプリケーション実行用コンテナイメージの作成」で作成した `shell_app_dev_image.swu` または `python_app_dev_image.swu` を「9.10.3. イメージのインストール」を参照して Armadillo ヘインストールしてください。

インストール後に自動で Armadillo が再起動します。

9.2.4. アプリケーション開発

9.2.4.1. サンプルアプリケーションの動作確認

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

図 9.40 my_project へ移動して VSCode を起動する。

サンプルアプリケーションのソースコードは `app_arm64/src` ディレクトリ下に入っています。開発を行う場合はソースコードのファイル名を `main.sh` としてください。Python の場合は `main.py` としてください。

このサンプルアプリケーションは、CPU と SOC の温度を `/root/shell_app/log/temp.txt` に出力し、LED3 を 5 回点滅させます。

9.2.4.2. ssh_config の準備

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

```
[ATDE ~/my_project]$ code config/ssh_config  
Host Armadillo  
  Hostname 0.0.0.0 ❶  
  User root  
  Port 2222  
  IdentityFile ../config/ssh/id_rsa
```

図 9.41 ssh_config を編集する

- 1 Armadillo の IP アドレスに置き換えてください。

9.2.4.3. 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。

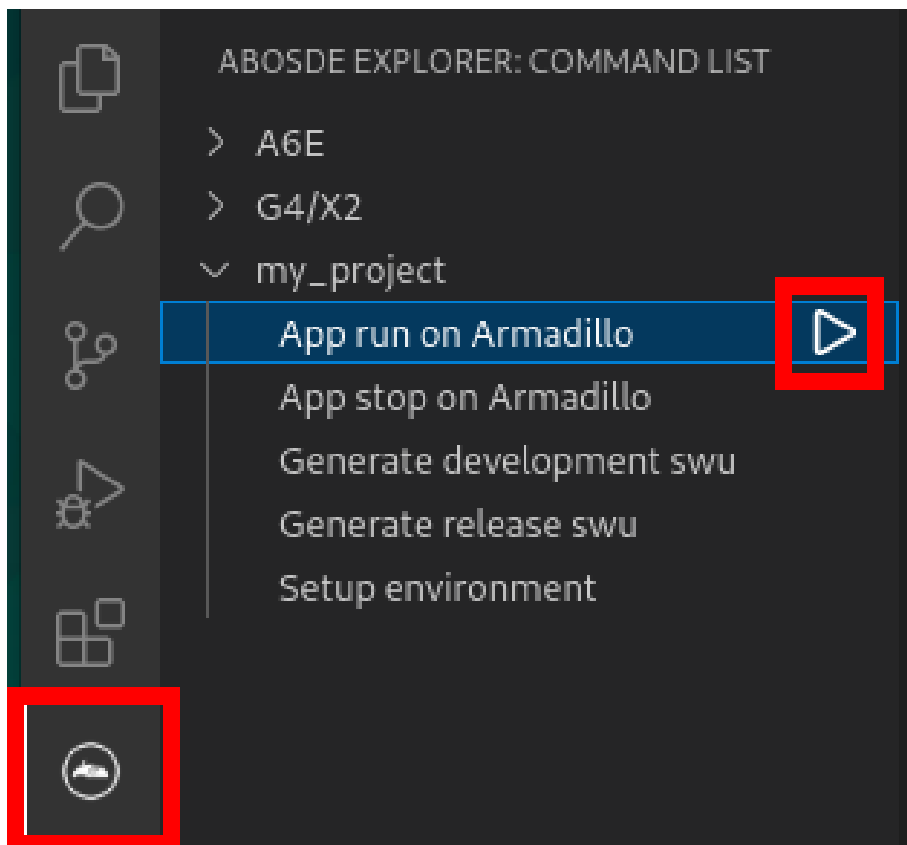


図 9.42 Armadillo 上でアプリケーションを実行する

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 9.43 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] を実行して下さい。

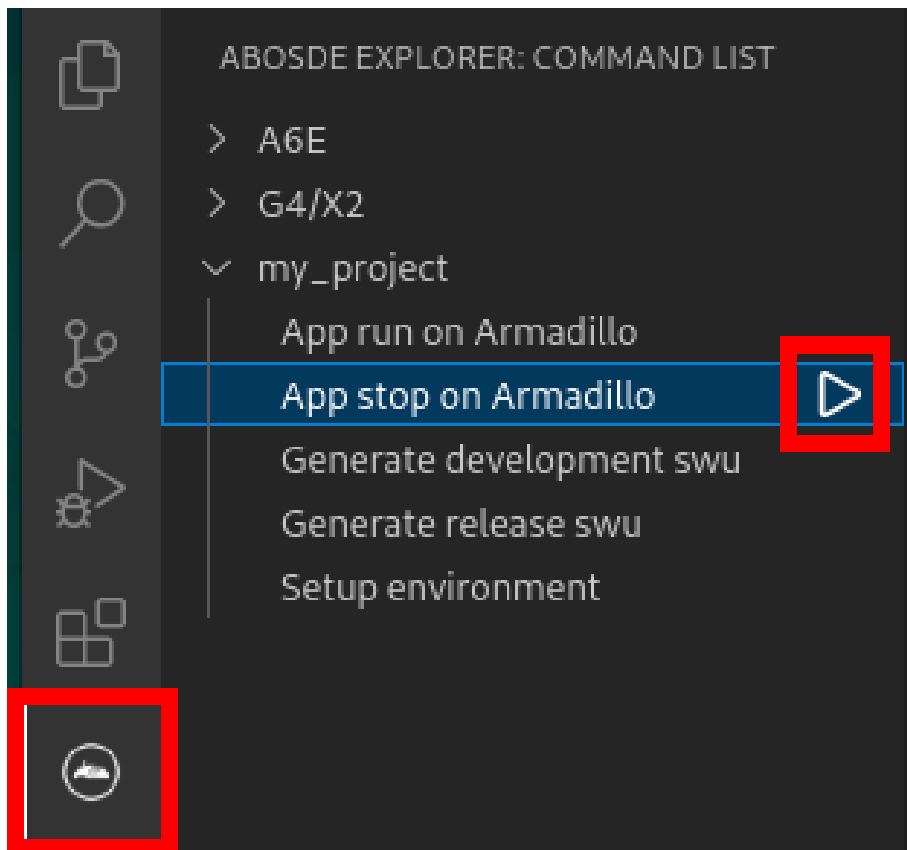


図 9.44 アプリケーションを終了する

9.2.5. リリース版のビルド

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

VSCode の左ペインの [my_project] から [Generate release swu] を実行すると、リリース版のアプリケーションを含んだ SWU イメージが作成されます。事前に「9.10.2. SWU イメージの作成」を参照して SWU の初期設定を行ってください。

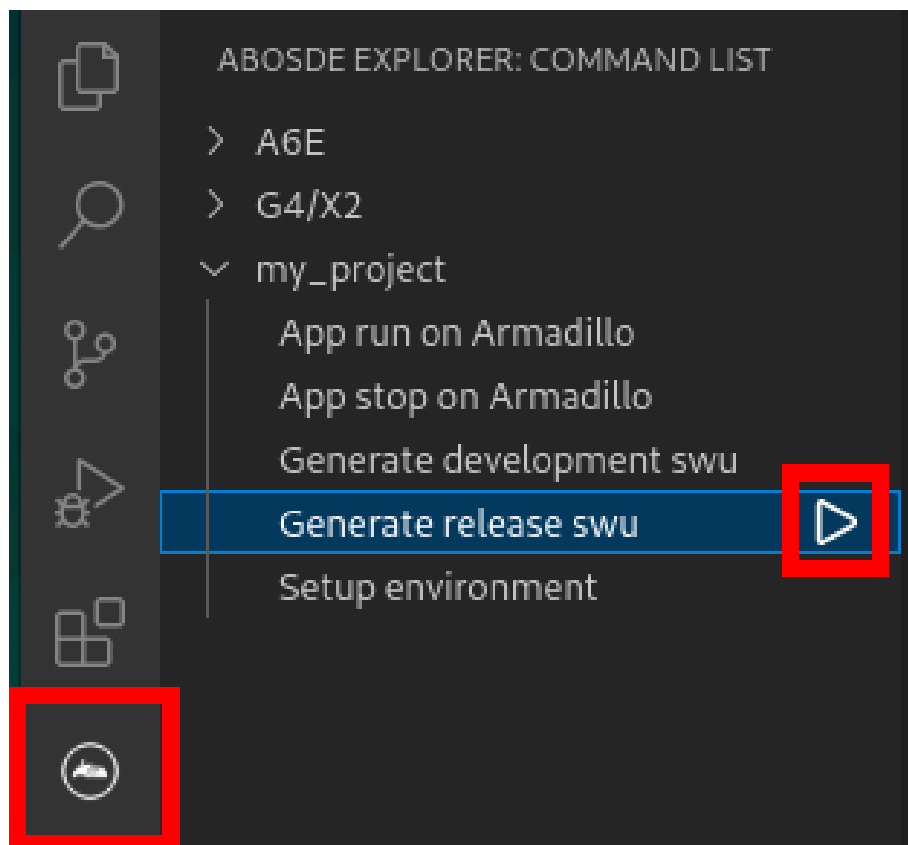


図 9.45 リリース版をビルドする

9.2.6. 製品への書き込み

作成した SWU イメージは `my_project/container_arm64/release` ディレクトリ下に `shell_app_release_image.swu` または `python_app_release_image.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「9.10.3. イメージのインストール」を参照して Armadillo へインストールすると、Armadillo 起動時にアプリケーションも自動起動します。

9.3. 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。VSCode を使用して以下のセットアップを行うことができます。

- ・ `initial_setup.swu` の作成
- ・ 有線・無線 LAN を設定するための SWU ファイルの作成
- ・ `at-debian-image` と `alpine` のコンテナイメージをインストールするための SWU ファイルの作成

9.3.1. ATDE 上でのセットアップ

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

9.3.1.1. 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 9.46 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 9.47 VSCode を起動する



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

9.3.1.2. 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。

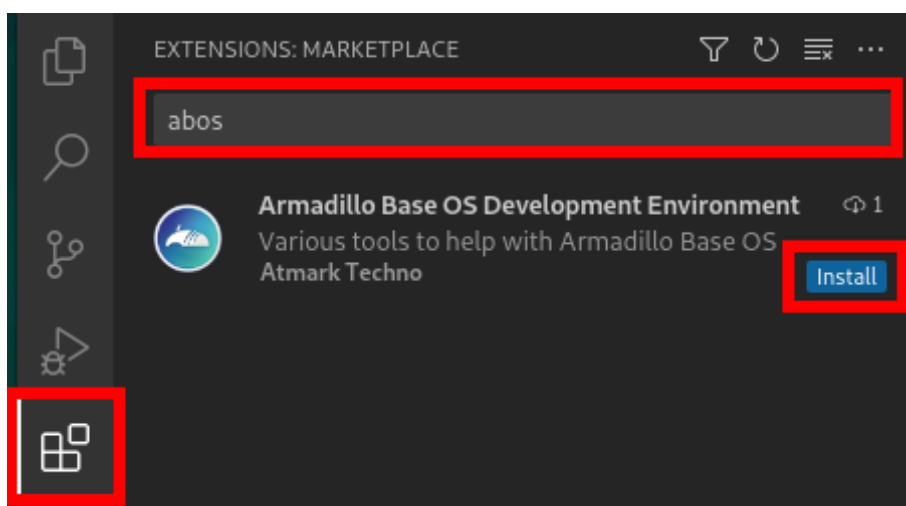


図 9.48 VSCode に開発用エクステンションをインストールする

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。

9.3.1.3. プロジェクトの作成

VSCoDe の左ペインの [G4/X2] から [Armadillo Setup 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。

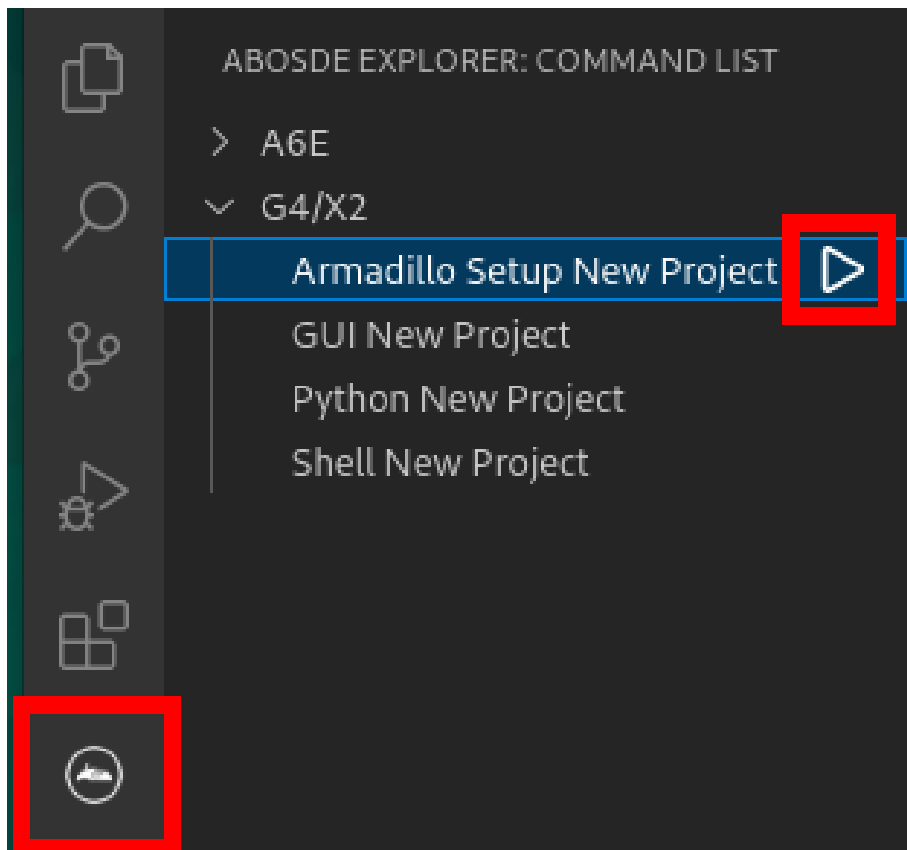


図 9.49 プロジェクトを作成する

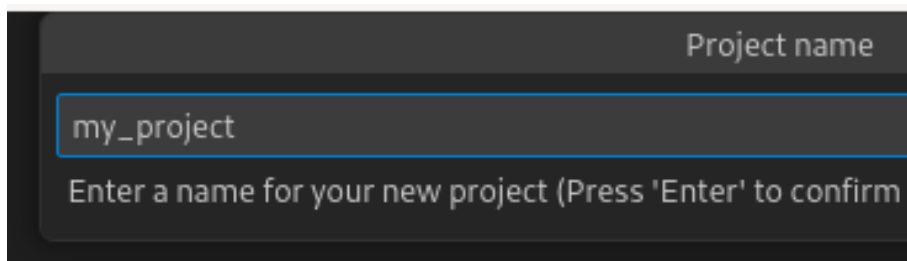


図 9.50 プロジェクト名を入力する

プロジェクト作成後、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 9.51 プロジェクトディレクトリへ移動して VSCode を起動する

9.3.2. initial_setup.swu の作成

initial_setup.swu ファイルの作成を行います。VSCode の左ペインの [my_project] から [Generate initial setup swu] を実行すると、initial_setup.swu が作成されます。これまでに mkswu --init コマンドを実行したことがない場合、VSCode のターミナル上で各種設定の入力を求められます。入力する設定の内容は「9.10.2. SWU イメージの作成」を参照してください。

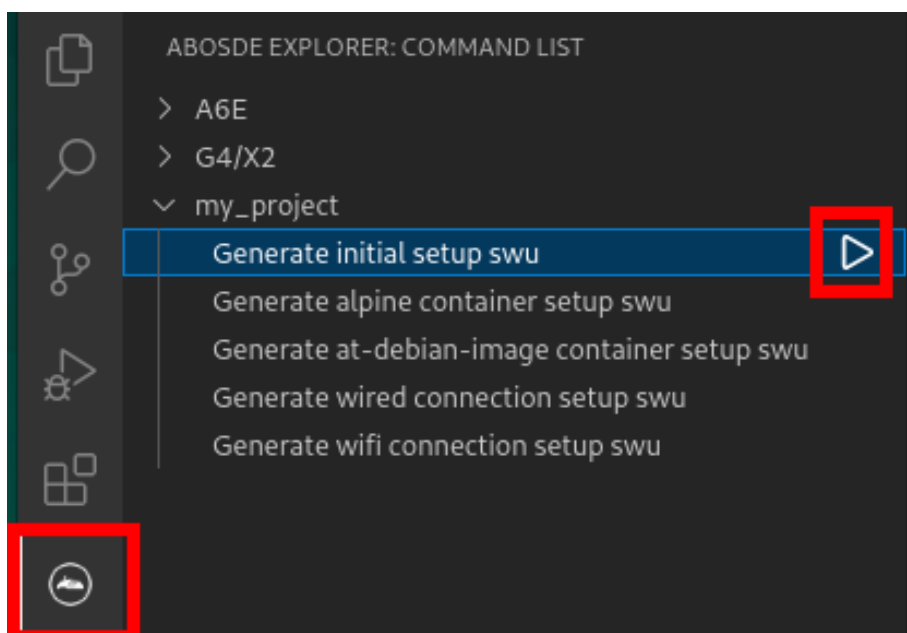


図 9.52 initial_setup.swu を作成する

ファイルは my_project/initial_setup/initial_setup.swu に保存されています。この SWU イメージを「9.10.3. イメージのインストール」を参照して Armadillo ヘインストールしてください。

9.3.3. LAN の設定

ここでは、有線・無線 LAN の設定ファイルの編集を VSCode 上で行い、編集後の設定ファイルを SWU ファイルで Armadillo ヘインストールするための手順を説明します。

9.3.3.1. 有線 LAN の設定

有線 LAN の設定を行うには、my_project/network_setup/conf/wired_connection.nmconnection を編集します。

```
[connection]  
id=wired_connection  
type=ethernet  
interface-name=eth0
```

```
[ethernet]

[ipv4]
method=auto ❶
# for static IP
#method=manual
#address1=<IP address>/<subnet mask>,<gateway address>
#e.g address1=192.0.2.0/24,192.0.2.1
# for DNS
#dns=<IP address>;
#e.g dns=192.0.2.1;

[ipv6]
addr-gen-mode=default
method=auto

[proxy]
```

図 9.53 有線 LAN の設定ファイルを編集する

- ❶ 固定 IP アドレスにする場合は manual に変更してください。設定例はコメントアウトしてある箇所を参考にしてください。

9.3.3.2. 無線 LAN の設定

無線 LAN の設定を行うには、`my_project/network_setup/conf/wifi_connection.nmconnection` を編集します。

```
[connection]
id=wifi_connection
type=wifi
interface-name=mlan0

[wifi]
mode=infrastructure
# SSID of AP
ssid= ❶

[wifi-security]
auth-alg=open
key-mgmt=wpa-psk
# Passphrase to connect to AP
psk= ❷

[ipv4]
method=auto ❸
# If you use static IP
#method=manual
#address1=<IP address>/<subnet mask>,<gateway address>
#e.g address1=192.0.2.0/24,192.0.2.1
# DNS
#dns=<IP address>;
#e.g dns=192.0.2.1;

[ipv6]
```

```
addr-gen-mode=default  
method=auto  
  
[proxy]
```

図 9.54 無線 LAN の設定ファイルを編集する

- ① 接続する AP の SSID を設定してください。
- ② AP に接続するためのパスワードを設定してください。
- ③ 固定 IP アドレスにする場合は manual に変更してください。設定例はコメントアウトしてある箇所を参考にしてください。

9.3.3.3. インストール用の SWU ファイルを作成する

LAN の設定を Armadillo にインストールするための SWU ファイルを作成します。VSCode の左ペインの [my_project] から、有線の場合は [Generate wired connection setup swu] を、無線の場合は [Generate wifi connection setup swu] を実行してください。

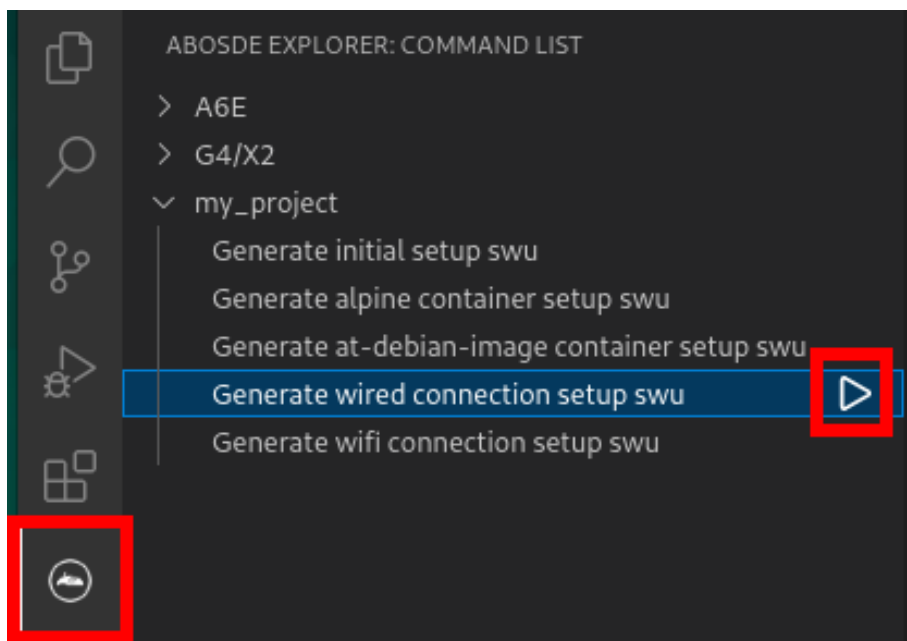


図 9.55 LAN 設定をインストールする SWU ファイルを作成する

それぞれ、my_project/network_setup/wired_connection.swu または my_project/network_setup/wifi_connection.swu に保存されています。この SWU イメージを「9.10.3. イメージのインストール」を参照して Armadillo へインストールしてください。

9.3.4. コンテナイメージインストール用の SWU ファイルの作成

ここでは、Armadillo へコンテナイメージをインストールするための SWU ファイルの作成手順について説明します。インストールするコンテナイメージは、alpine と 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 の 2 種類です。

9.3.4.1. alpine のコンテナイメージをインストールする

VSCoide の左ペインの [my_project] から [Generate alpine container setup swu] を実行してください。

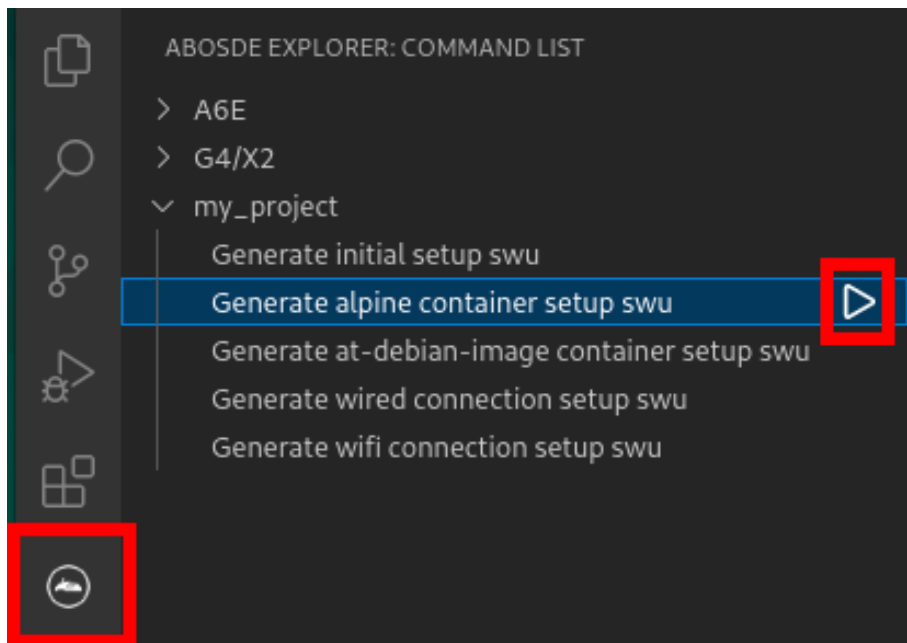


図 9.56 alpine のコンテナイメージをインストールする SWU ファイルを作成する

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「9.10.3. イメージのインストール」を参照して Armadillo へインストールしてください。

9.3.4.2. at-debian-image のコンテナイメージをインストールする

VSCoide 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。

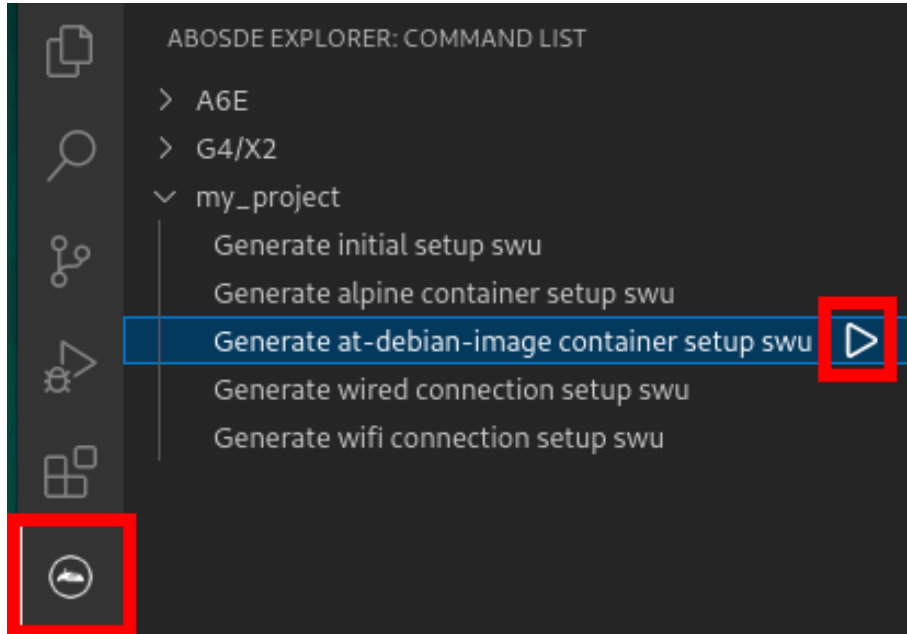


図 9.57 at-debian-image のコンテナイメージをインストールする SWU ファイルを作成する

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「9.10.3. イメージのインストール」を参照して Armadillo へインストールしてください。

9.4. アプリケーションをコンテナで実行する

9.4.1. Podman - コンテナ仮想化ソフトウェア

9.4.1.1. Podman - コンテナ仮想化ソフトウェアとは

コンテナとはホスト OS 上に展開される仮想的なユーザ空間のことです。コンテナを使用することで複数の Armadillo-X2 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。

9.4.2. コンテナを操作する

この章では、コンテナ仮想化ソフトウェアの 1 つである Podman の基本的な使い方について説明します。Armadillo-X2 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-X2 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。

9.4.2.1. 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` にすでに詳しいかは `podman run` コマンドでも実行できますが、ここでは「9.5. コンテナの運用」で

紹介するコンテナの自動起動の準備も重ねて `podman_start` を使います。イメージは Docker Hub [https://hub.docker.com] から自動的に取得されます。ここでは、簡単な例として `ls /` コマンドを実行するコンテナを作成します。

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

図 9.58 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「9.5.1. コンテナの自動起動」を参照ください。
- ❷ コンテナのイメージを取得します。イメージが Armadillo に置いてない場合は「Error: docker.io/alpine: image not known」の様なエラーで失敗します。
- ❸ コンテナを起動します。これは Armadillo 起動時に自動的に起動されるコンテナと同じものになります。自動起動が不要な場合には `set_autostart no` で無効化できます。
- ❹ `podman logs` コマンドで出力を確認します。

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



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

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

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

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



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

9.4.2.2. 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 9.59 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 9.60 `podman images --help` の実行例

9.4.2.3. コンテナ一覧を表示する

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

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

図 9.61 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 9.62 `podman ps --help` の実行例

9.4.2.4. 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 9.63 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 9.64 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 9.65 `podman start --help` 実行例

9.4.2.5. 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 9.66 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 9.67 podman stop --help 実行例

9.4.2.6. コンテナの変更を保存する

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

変更を保存するには二つの方法があります。

1. podman commit コマンドで保存する。

```
[armadillo ~]# podman commit my_container image_name:latest
Getting image source signatures
Copying blob f4ff586c6680 skipped: already exists
Copying blob 3ae0874b0177 skipped: already exists
Copying blob ea59ffe27343 done
Copying config 9ca3c55246 done
Writing manifest to image destination
Storing signatures
9ca3c55246eaac267a71731bad6bfe4b0124afcd2b80c4f730c46aae17a88f3
```

図 9.68 my_container を保存する例

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

2. ボリュームを使用する。

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

保存するデータの性質によって、保存先を選択してください。

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



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 9.69 `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。

9.4.2.7. 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 9.70 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
$ podman build -t my_image:2 -t my_image:latest .
STEP 1: FROM localhost/my_image:latest
STEP 2: RUN apk upgrade --no-cache
--> cf1dc0d7296
STEP 3: COPY my_application /my_application
STEP 4: COMMIT my_image:latest
--> 9e9d9366072
Successfully tagged localhost/my_image:2
Successfully tagged localhost/my_image:latest
9e9d9366072751007b2e70544d76c46b95a7a5a02df658ef0fa3f7dccc8850a

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

図 9.71 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「9.10.6. mkswu の desc ファイル」の swdesc_embed_container と swdesc_usb_container で使えます。

9.4.2.8. 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 9.72 コンテナを削除する実行例

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

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

図 9.73 \$ podman rm --help 実行例

9.4.2.9. 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 9.74 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 9.75 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「9.5.5.2. イメージを eMMC に保存する方法」を参照してください。

```
[armadillo ~]# podman images
REPOSITORY          TAG          IMAGE ID        CREATED        SIZE        R/0
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 9.76 Read-Only のイメージを削除する実行例

9.4.2.10. コンテナとコンテナに関連するデータを削除する

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



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

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

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

- /etc/swupdate.watch

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

図 9.77 abos-ctrl container-clear 実行例

9.4.2.11. 実行中のコンテナに接続する

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

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

図 9.78 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 9.79 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 9.80 podman exec --help 実行例

9.4.2.12. コンテナ間で通信をする

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

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

準備として、2 つのコンテナを作成します。

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

図 9.81 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 9.82 コンテナの IP アドレスを確認する実行例

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

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

```

64 bytes from 10.88.0.109: seq=1 ttl=42 time=0.210 ms

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

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

```

図 9.83 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。

9.4.2.13. 開発時に有用な—privileged オプション

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

9.4.3. アットマークテクノが提供するイメージを使う

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

9.4.3.1. Docker ファイルからイメージをビルドする

Armadillo-X2 コンテナ [<https://armadillo.atmark-techno.com/resources/software/armadillo-x2/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 9.84 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。

9.4.3.2. ビルド済みのイメージを使用する

Armadillo-X2 コンテナ [https://armadillo.atmark-techno.com/resources/software/armadillo-x2/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 9.85 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。

9.4.4. 入出力デバイスを扱う

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

9.4.4.1. GPIO を扱う

コンテナ内で動作するアプリケーションから GPIO を扱うためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/gpiochipN を渡す必要があります。以下は、/dev/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
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/gpiochip2
[armadillo ~]# podman_start gpio_example
Starting 'gpio_example'
956a0fecc48d5ea1210069910f7bb48b9e90b2dadbd12895064d9776dae0360b5
```

図 9.86 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_I021 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 9.87 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 9.88 `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 9.89 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。

9.4.4.2. I2C を扱う

コンテナ内で動作するアプリケーションから I2C を扱うためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/i2c-N を渡す必要があります。以下は、/dev/i2c-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 9.90 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 9.91 i2cdetect コマンドによる確認例

9.4.4.3. SPI を扱う

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

```

[armadillo ~]# vi /etc/atmark/containers/spi_example.conf
set_image docker.io/alpine
    
```

```
set_command sleep infinity
add_devices /dev/spidev1.0
[armadillo ~]# podman_start spi_example
Starting 'spi_example'
45302bc9f95eef0e25c5d98acf198d96fc5bec1f83e791018cbe4221cc1f4523
```

図 9.92 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 9.93 spi-config コマンドによる確認例

9.4.4.4. CAN を扱う

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

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

図 9.94 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 9.95 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。

9.4.4.5. PWM を扱う

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

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

図 9.96 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 9.97 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。

9.4.4.6. シリアルインターフェースを扱う

コンテナ内で動作するアプリケーションから RS-232C や RS-485 などのシリアル通信を行うためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル `/dev/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 9.98 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 9.99 `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

9.4.4.7. USB を扱う

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

- ・ USB シリアルデバイスを扱う

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

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

図 9.100 USB シリアルデバイスを扱うためのコンテナ作成例

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

```
[armadillo ~]# podman exec -it usb_example sh
[container ~]# setserial -a /dev/ttyUSB0
```



```
/dev/ttyUSB0, Line 0, UART: unknown, Port: 0x0000, IRQ: 0
  Baud_base: 24000000, close_delay: 0, divisor: 0
  closing_wait: infinite
  Flags: spd_normal
```

図 9.101 setserial コマンドによる USB シリアルデバイス設定の確認例

- ・ USB カメラを扱う

USB カメラをコンテナ内から扱う場合には、Podman のイメージからコンテナを作成する際にホスト OS 側の `/dev/videoN` を渡す必要があります。以下は、`/dev/video3` を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# vi /etc/atmark/containers/usbcam_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/video3
[armadillo ~]# podman_start usbcam_example
Starting 'usbcam_example'
ffe06090b45826cc0b1c7710e9e850ba9521d36b70de4288d0dfe1fe91a35632
[armadillo ~]# podman exec -it usbcam_example sh
[container ~]# ls /dev/video3
/dev/video3
```

図 9.102 USB カメラを扱うためのコンテナ作成例

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

- ・ USB メモリを扱う

ここでは、USB メモリを扱う方法について 2 つの例を示します。

- ・ ホスト OS 側でマウントした USB メモリをコンテナから扱う

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

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

図 9.103 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 9.104 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 9.105 USB メモリに保存されているデータの確認例

- ・ USB メモリをコンテナ内からマウントする

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

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

図 9.106 USB メモリをマウントするためのコンテナ作成例

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

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

図 9.107 コンテナ内から USB メモリをマウントする例

9.4.4.8. RTC を扱う

コンテナ内から RTC を扱うためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/rtcN を渡すと同時に、RTC への時刻の設定を行うための権限も渡す必要

があります。以下は、/dev/rtc0 を渡して alpine イメージからコンテナを作成する例です。権限として SYS_TIME も渡しています。

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

図 9.108 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 9.109 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。

9.4.4.9. 音声出力を行う

Armadillo-X2 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 9.110 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 9.111 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。

9.4.4.10. ユーザースイッチのイベントを取得する

Armadillo-X2 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 9.112 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 ②
Event: time 1612849229.894444, ----- SYN_REPORT -----
```

図 9.113 evtest コマンドによる確認例

- ① SW1 のボタン プッシュ イベントを検出したときの表示
- ② SW1 のボタン リリース イベントを検出したときの表示

9.4.4.11. LED を扱う

Armadillo-X2 には LED が実装されています。これらの LED を扱うためには、Podman のイメージからコンテナを作成する際にホスト OS 側の /sys ディレクトリを渡す必要があります。以下は、/sys を渡して alpine イメージからコンテナを作成する例です。ここで渡された /sys ディレクトリはコンテナ内の /sys にマウントされます。

```
[armadillo ~]# vi /etc/atmark/containers/led_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_volumes /sys
[armadillo ~]# podman_start led_example
Starting 'led_example'
c770f76d7714f4ccebe1229be2240382bde236c2c51bb6b866bc0098c2cb987a
```

図 9.114 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 9.115 LED の点灯/消灯の実行例

9.4.5. 近距離通信を行う

この章では、コンテナ内から近距離通信デバイスを扱う方法について示します。

9.4.5.1. Bluetooth デバイスを扱う

コンテナ内から Bluetooth デバイスを扱うためには、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/ttymxN を渡すと同時にネットワークとして host を、権限として NET_ADMIN を渡す必要があります。/dev/ttymxN は Bluetooth 通信で使用するよう設定したシリアルデバイスを指定してください。以下は、/dev/ttymx0 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/bt_example.conf
set_image docker.io/alpine
set_command sleep infinity
set_network host
```

```
add_devices /dev/ttyxc0
add_args --cap-add=NET_ADMIN
[armadillo ~]# podman_start bt_example
Starting 'bt_example'
45fe1eb6b25529f0c84cd4b97ca1aef8451785fc9a87a67d54873c1ed45b70a4
```

図 9.116 Bluetooth デバイスを扱うためのコンテナ作成例

コンテナ内で必要なソフトウェアをインストールして、Bluetooth を起動します。btattach コマンドの引数にはコンテナ作成時に渡した ttyxc0 を設定してください。

```
[armadillo ~]# podman exec -it bt_example sh
[container ~]# apk upgrade
[container ~]# apk add bluez dbus
[container ~]# /usr/bin/dbus-daemon --system
[container ~]# /usr/lib/bluetooth/bluetoothd &
[container ~]# btattach -B /dev/ttyxc0 -S 115200 &
```

図 9.117 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 9.118 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。

9.4.5.2. Wi-SUN デバイスを扱う

ここでは、Wi-SUN デバイスが UART で接続されている場合の例を示します。この場合、コンテナ内で動作するアプリケーションから Wi-SUN デバイスで通信を行うためには、Podman のイメージからコ

ンテナを作成する際にホスト OS 側のデバイスファイル /dev/ttyxcN のうち、Wi-SUN と対応するものを渡す必要があります。以下は、/dev/ttyxc0 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/ttyxc0
[armadillo ~]# podman_start wisun_example
Starting 'wisun_example'
ef9a5a2f7eee4236cb28c1bf5090a6f0db9d6dfe7f3a34573867e0833dd3122
[armadillo ~]# podman exec -it wisun_example sh
[container ~]# ls /dev/ttyxc0
/dev/ttyxc0
```

図 9.119 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/ttyxc0 を使って Wi-SUN データの送受信ができるようになります。

9.4.5.3. EnOcean デバイスを扱う

ここでは、EnOcean デバイスが UART で接続されている場合の例を示します。この場合、コンテナ内で動作するアプリケーションから EnOcean デバイスで通信を行うためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/ttyxcN のうち、EnOcean と対応するものを渡す必要があります。以下は、/dev/ttyxc0 を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# vi /etc/atmark/containers/enoccean_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_deivces /dev/ttyxc0
[armadillo ~]# podman_start enoccean_example
Starting 'enoccean_example'
a808b491a100f9078d8c72a7b36966d9182614f3657fe054fb8d7f87b0d4b31c
[armadillo ~]# podman exec -it enoccean_example sh
[container ~]# ls /dev/ttyxc0
/dev/ttyxc0
```

図 9.120 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/ttyxc0 を使って EnOcean データの送受信ができるようになります。

9.4.6. ネットワークを扱う

この章では、コンテナ内のネットワークを扱う方法について示します。

9.4.6.1. コンテナの IP アドレスを確認する

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

```
[armadillo ~]# vi /etc/atmark/containers/net_example.conf
set_image docker.io/alpine
```

```

set_command sleep infinity
[armadillo ~]# podman start net_example
Starting 'net_example'
48ae479af65445674323567c17c5418dd4624292351e061bd2bd8a0add4cf150
[armadillo ~]# podman inspect --format '{{ .NetworkSettings.IPAddress }}' net_example
10.88.0.17

```

図 9.121 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@if18: <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 9.122 ip コマンドを用いたコンテナの IP アドレス確認例

9.4.6.2. コンテナに固定 IP アドレスを設定する



podman はデフォルトで 10.88.0.0/16 を使います。

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

コンテナに固定 IP アドレスを設定するためには、最初にユーザ定義のネットワークを作成する必要があります。以下に 192.168.1.0/24 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 9.123 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 9.124 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 9.125 コンテナの IP アドレス確認例

9.4.7. サーバを構築する

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

9.4.7.1. HTTP サーバを構築する

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

- ・ Apache を使用する

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

```
[armadillo ~]# vi /etc/atmark/containers/apache_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_ports 8080:80
[armadillo ~]# podman_start apache_example
Starting 'apache_example'
ea0a1ed9c2fe170a6db02e480300467510f4e844900efb35c7a24cc1a8653af2
[armadillo ~]# podman exec -it apache_example sh
[container ~]# apk upgrade && apk add apache2
[container ~]# httpd
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using
10.88.0.2. Set the 'ServerName' directive globally to suppress this message
```

図 9.126 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 9.127 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。

9.4.7.2. FTP サーバを構築する

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

```
[armadillo ~]# vi /etc/atmark/containers/ftp_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_ports 21:21 21100-21110:21100-21110
add_args --env=PASV_ADDRESS=<ホストの IP アドレス>
[armadillo ~]# podman_start ftp_example
Starting 'ftp_example'
efcf1ba752c2db9ae1a33ac11af3be71d95ac7b737ce9734730ebca602e57796
[armadillo ~]# podman exec -it ftp_example sh
[container ~]# apk upgrade && apk add vsftpd
```

図 9.128 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 9.129 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 9.130 設定ファイルの編集例

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

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

図 9.131 vsftpd の起動例

9.4.7.3. Samba サーバを構築する

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

```
[armadillo ~]# vi /etc/atmark/containers/smb_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_ports 139:139 445:445
[armadillo ~]# podman_start smb_example
Starting 'smb_example'
6d81c01fe27b5a92ee6ea69de2f9a8dbb569d420c2f5f630ece1966c81824a1f
[armadillo ~]# podman exec -it smb_example sh
[container ~]# apk upgrade && apk add samba
```

図 9.132 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 9.133 ユーザを追加する例

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

```
[container ~]# smb
```

図 9.134 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。

9.4.7.4. SQL サーバを構築する

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

```
[armadillo ~]# vi /etc/atmark/containers/sqlite_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_volumes /var/app/volumes/sqlite_db:/db
[armadillo ~]# podman_start sqlite_example
Starting 'sqlite_example'
114c5f1dbb7e81293dcb8f8e0c600b861626375b14cfe4023761acaa84fdcad1
[armadillo ~]# podman exec -it sqlite_example sh
[container ~]# apk upgrade && apk add sqlite
```

図 9.135 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 9.136 sqlite の実行例

9.4.8. 画面表示を行う

この章では、コンテナ内で動作するアプリケーションから Armadillo-X2 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。

9.4.8.1. 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「9.4.3. アットマークテクノが提供するイメージを使う」を参照してください。

```
[armadillo ~]# vi /etc/atmark/containers/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 ④
add_volumes /opt/firmware:/opt/firmware:ro ⑤
add_args --cap-add=SYS_TTY_CONFIG ⑥
set_command weston --tty 7 ⑦
[armadillo ~]# podman start wayland_example
Starting 'wayland_example'
654fe87422f85e8835b00761071347bafa632f969645db5fa835c88e2a55e2cc

```

図 9.137 Wayland を扱うためのコンテナ作成例

- ① weston の実行に必要な環境変数を設定します。
- ② 画面描画に必要なデバイスを設定します。
- ③ キーボードやマウスなどを使用可能にするためのデバイスを設定します。
- ④ ホスト OS 側の /run/udev をコンテナ内からマウントするように設定します。
- ⑤ 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。
- ⑥ tty を操作するための権限を設定します。
- ⑦ 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 9.138 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-X2 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 9.139 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 9.140 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「9.4.2.6. 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 ⑤
[armadillo ~]# podman_start detect_object ⑥
```

- ① weston の設定ファイルを作成します。
- ② XDG_RUNTIME_DIR を volume で共有して、同じディレクトリを使います。
- ③ 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。
- ④ 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。
- ⑤ 確認のためコンテナを手動で起動します。
- ⑥

- ・ 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 ①
set_image localhost/at-debian-image:latest
add_devices /dev/dri /dev/galcore /dev/input /dev/tty7 ②
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 ③
[armadillo ~]# podman_start weston ④
```

- ① weston の設定ファイルを作成します。
- ② 使用する tty として /dev/tty7 を追加します。
- ③ 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。
- ④ 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 ①
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-X2 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。

9.4.8.2. 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「9.4.3. アットマークテクノが提供するイメージを使う」を参照してください。

```

[armadillo ~]# vi /etc/atmark/containers/x_example.conf
set_image at-debian-image
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 9.141 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=ttyMX1,115200 root=/dev/mmcblk2p1rootwait 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 9.142 コンテナ内で X Window System を起動する実行例

Armadillo-X2 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。

9.4.8.3. 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 9.143 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 9.144 フレームバッファに直接描画する実行例

9.4.8.4. 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 9.145 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。

9.4.8.5. VPU を扱う

Armadillo-X2 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「9.4.3. アットマークテクノが提供するイメージを使う」を参照してください。

```
[armadillo ~]# vi /etc/atmark/containers/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 9.146 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 9.147 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 9.148 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 9.149 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。

9.4.9. パワーマネジメント機能を使う

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

9.4.9.1. サスペンド状態にする

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

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

図 9.150 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 9.2 対応するパワーマネジメント状態

パワーマネジメント状態	文字列	説明
Suspend-to-RAM	mem	最も消費電力を抑えることができる
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 9.151 サスペンド状態にする実行例

9.4.9.2. 起床要因を有効化する

サスペンド状態から起床要因として、利用可能なデバイスを以下に示します。

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/ttymx1/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/32f10108.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'
```

```
8fbef3edda3b7fcea5b1f8cbf960cf469b7e82c4d1ecd35477076e81fc24
e39f
[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 9.152 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「9.12.4. DT overlay によるカスタマイズ」を参照してください。
- ❷ /boot/overlays.txt を保存し、アップデートの場合でも保存します。
- ❸ overlay の実行のために再起動します。

- ④ 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。
- ⑤ Linux から確認できます。



Armadillo-X2 は Armadillo-IoT ゲートウェイ G4 と同じ DTB で動作します。そのため、「9.12. Device Tree をカスタマイズする」や「9.12.4. DT overlay によるカスタマイズ」でも `armadillo_iotg_g4-` から始まる `dtb` を利用します。

この動作は、RAM 上にロードされた DTB をブートローダーが修正することで実現されています。

9.4.10. コンテナからの poweroff か reboot

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

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

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

図 9.153 コンテナから shutdown を行う

9.4.11. 異常検知

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

9.4.11.1. ソフトウェアウォッチドッグタイマーを扱う

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

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

```
Starting 'watchdog_example'  
a5d329cca49d60423ce4155d72a119b8049a03dbd1d0277817a253e96dce7bc7
```

図 9.154 ソフトウェアウォッチドッグタイマーを使うためのコンテナ作成例

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

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

図 9.155 コンテナ内からソフトウェアウォッチドッグタイマーを起動する実行例

ソフトウェアウォッチドッグタイマーを起動した後、`/dev/watchdog0` に任意の文字を書き込むことでソフトウェアウォッチドッグタイマーをリセットすることができます。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 9.156 ソフトウェアウォッチドッグタイマーをリセットする実行例

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

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

図 9.157 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

9.4.12. NPU を扱う

Armadillo-X2 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「9.4.3. アットマークテクノが提供するイメージを使う」を参照してください。

```
[armadillo ~]# vi /etc/atmark/containers/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 9.158 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]。

9.4.12.1. 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-X2 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
[armadillo ~]# podman exec -ti onnxruntime_example bash
[container ~]# apt install onnxruntime onnxruntime-dev onnxruntime-tools python3-onnxruntime
```

図 9.159 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
```

^[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。

```
: (省略)
>>> sess = onnxruntime.InferenceSession('model.onnx', providers=['VsiNpuExecutionProvider'])
```

図 9.160 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。

9.4.12.2. TensorFlow Lite を使う

TensorFlow Lite から 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 9.161 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 9.162 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「8.3. 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 9.3. 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 9.3 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、「8.3.4. 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 9.163 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 9.4. 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 9.164. 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 9.4 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


パッケージ名	バージョン	URL
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 9.164 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 9.4. 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。

9.4.12.3. 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 9.165 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 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 9.166 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 9.167 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。

9.5. コンテナの運用

9.5.1. コンテナの自動起動

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

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

図 9.168 コンテナを自動起動するための設定例

`.conf` ファイルは以下のパラメータを設定できます。

- ・ コンテナイメージの選択：**set_image** [イメージ名]

イメージの名前を設定できます。

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

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

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

- ・ ポート転送：**add_ports** [ホストポート]:[コンテナポート]

設定したポートで外部からコンテナへのアクセスが可能となります。

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

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

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



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

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

コンテナでデバイスを作成して、使用可能となります。

コンテナパスを設定しない場合はホストと同じパスを使います。

複数のデバイスを作成したい場合はスペースで分けて設定してください。

例: `add_devices /dev/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。

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

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

ホストパスは以下のどちらかを指定してください。

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

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

SWUpdate でアップデートするデータに向いています。

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

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

ログやデータベースに向いています。

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


コンテナパスを設定しない場合はホストパスと同じパスを使います。

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

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

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

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



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

```

[armadillo ~]# cat /etc/atmark/containers/mounter.conf
set_image docker.io/alpine
add_args -ti
add_volumes /tmp/mnt:/mnt:shared ❶
add_args --cap-add SYS_ADMIN
                    
```

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

図 9.169 ポリ्यूームを shared でサブマウントを共有する例

- ❶ マウントを行うコンテナに shared の設定とマウント権限 (SYS_ADMIN) を与えます。
- ❷ マウントを使うコンテナに slave だけを設定すれば一方にしか共有されません。
- ❸ USB デバイスをマウントします。
- ❹ マウントされたことを確認します。

・ ホットプラグデバイスの追加： **add_hotplugs** [デバイスタイプ]

コンテナ起動後に挿抜を行っても認識される(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 9.5. 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 9.5 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 9.5. add_hotplugs オプションに指定できる主要な文字列」に示した文字列以外にも、/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 9.170. /proc/devices の内容例」に示す状態の場合、デバイスタイプを示す文字列としては、各行の先頭の数字を除いた mem や pty などを指定できることがわかります。

```
[armadillo ~]# cat /proc/devices
Character devices:
 1 mem
 2 pty
 3 tty
 4 /dev/vc/0
```



```

4 tty
4 ttyS
5 /dev/tty
5 /dev/console
5 /dev/ptmx
7 vcs
10 misc
13 input
29 fb
81 video4linux
: (省略)

```

図 9.170 /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`

- ・ pod の選択 : `set_pod` [ポッド名]

「9.5.2. pod の作成」で作成した pod の名前を入れてコンテナを pod 内で起動します。

例: `set_pod mypod`

- ・ ネットワークの選択 : `set_network` [ネットワーク名]

この設定に「9.5.3. network の作成」で作成したネットワーク以外に `none` と `host` の特殊な設定も選べます。

`none` の場合、コンテナに `localhost` しかない名前空間に入ります。

`host` の場合は OS の名前空間をそのまま使います。

例: `set_network mynetwork`

- ・ IP アドレスの設定 : `set_ip` [アドレス]

コンテナの IP アドレスを設定することができます。

例: `set_ip 10.88.0.100`



コンテナ間の接続が目的であれば、pod を使って `localhost` か pod の名前でアクセスすることができます。

- ・ 読み取り専用設定 : `set_readonly yes`

コンテナ内からのファイルシステムへの書き込み許可を設定します。

デフォルトで書き込み可能となっています。

コンテナ内からのファイルシステムへの書き込みを禁止することで、tmpfs として使うメモリの消費を明示的に抑えることができますが、アプリケーションによっては読み込み専用のファイルシステムでは動作しない可能性もあります。

- ・ イメージの自動ダウンロード設定： **set_pull** [設定]

この設定を missing にすると、イメージが見つからない場合にイメージを自動的にダウンロードします。

always にすると、イメージがすでにダウンロード済みでも起動前に必ず更新の確認を取ります。

デフォルトでは never で、イメージが見つからない場合にエラーを表示します。

例：set_pull missing か set_pull always

- ・ コンテナのリスタート設定： **set_restart** [設定]

コンテナが停止した時にリスタートさせます。

podman kill か podman stop で停止する場合、この設定と関係なくリスタートしません。

デフォルトで on-failure になっています。

例：set_restart always か set_restart no

- ・ 信号を受信するサービスの無効化: **set_init no**

コンテナのメインプロセスが PID 1 で起動していますが、その場合のデフォルトの信号の扱いが変わります: SIGTERM などのデフォルトハンドラが無効です。

そのため、init 以外のコマンドを set_command で設定する場合は podman-init のプロセスを PID 1 として立ち上げて、設定したコマンドをその子プロセスとして起動します。

例：set_init no

- ・ 自動起動の無効化： **set_autostart no**

手動かまたは別の手段で操作するコンテナがある場合、Armadillo の起動時に自動起動しないようにします。

その場合、podman_start <name> で起動させることができます。



コンフィグに記載していないイメージはアップデートの際に削除されますので、そういったイメージに対して設定してください。

- ・ 実行コマンドの設定： **set_command** [コマンド]

コンテナを起動するときのコマンド。設定されなかった場合、コンテナイメージのデフォルトを使います。

例: `set_command /bin/sh -c "echo bad example"`

- ・ `podman run` に引数を渡す設定: **add_args [引数]**

ここまでで説明した設定項目以外の設定を行いたい場合は、この設定で `podman run` に直接引数を渡すことができます。

例: `add_args --cap-add=SYS_TTY_CONFIG --env=XDG_RUNTIME_DIR=/run/xdg_home`

9.5.2. pod の作成

`podman_start` で `pod` 機能を使うことができます。

`pod` を使うことで、複数のコンテナが同じネットワークネームスペースを共有することができます。同じ `pod` 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
set_infra_imager k8s.gcr.io/pause:3.5

[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  k8s.gcr.io/pause:3.5                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 9.171 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。

ネットワーク設定の他に、`infra_image` のオプションで `pod` のイメージも固める事ができます。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

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。



pod を使う時に `podman` が特殊な「infra container」も起動します (例の場合、`k8s.gcr.io/pause:3.5` を起動させました)

コンフィグレーションに pod を入れるアップデートの際に自動的に podman pull でイメージをダウンロードしますが、インターネットを使わせたくないアップデートがあれば `swdesc_embed_container` か `swdesc_usb_container` で入れてください。その場合、`infra_image` の設定も使ってください。

9.5.3. network の作成

podman_start で podman の network も作成ことができます。

デフォルトの 10.88.0.0/16 が使えない場合、あるいはコンテナ同士で接続できないようにしたい場合は使ってください。

```
[armadillo ~]# cat /etc/atmark/containers/mynetwork.conf
set_type network
set_subnet 192.168.100.0/24

[armadillo ~]# cat /etc/atmark/containers/nginx.conf
set_image docker.io/library/nginx:alpine
add_ports 80:80
set_ip 192.168.100.10
set_network mynetwork

[armadillo ~]# podman ps
CONTAINER ID   IMAGE                                COMMAND                                CREATED        STATUS
PORTS          NAMES
3292e5e714a2   docker.io/library/nginx:alpine      nginx -g daemon o...  2 hours ago   Up 2 hours ago
0.0.0.0:80->80/tcp   nginx
```

図 9.172 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。

9.5.4. コンテナからのコンテナ管理

podman では REST API による管理アクセスも可能です。

自分のコンテナから他のコンテナの管理が必要な場合に、ホストの podman サービスを有効にして、コンテナに `/run/podman` をボリュームマウントすれば `podman --remote` で管理できます。

podman_start をインストールすればそちらも `--remote` で使えます。

このオプションは Armadillo のホスト側の udev rules からコンテナを扱う時にも必要です。

9.5.5. コンテナの配布



コンテナの作成は「9.4. アプリケーションをコンテナで実行する」を参考にしてください。

コンテナのイメージを配布する方法は大きく分けて二つあります：

1. インターネット上のリポジトリ（dockerhub 等）で登録してそこから配布する
2. SWUpdate のアップデートイメージを配布する



Podman のイメージをインストールする時に、一時データを大量に保存する必要があります。

swu イメージ内で組み込む時は 3 倍、pull や USB ドライブで分けてインストールすると転送するデータ量の 2 倍の空き容量が app パーティションに必要です。

アップデート時にアップデート前のコンテナが使われているのでご注意ください。

9.5.5.1. リモートリポジトリにコンテナを送信する方法

1. イメージをリモートリポジトリに送信する：

```
[armadillo ~]$ podman image push <localimage> docker://<registry>/<remoteimage>:<tag>
```

2. set_pull always を設定しないかぎり、SWUpdate でダウンロードの命令を送らないとアップデートを行いません。

(mkswu については「9.10. 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。
```

9.5.5.2. イメージを eMMC に保存する方法

Armadillo Base OS のデフォルトでは、Podman のデータは tmpfs に保存されます。

起動時にコンテナを起動するにはイメージを eMMC に書き込む必要があります。開発が終わって運用の場合は「9.5.5.3. イメージを SWUpdate で転送する方法」でコンテナのイメージを転送します。この場合は読み取り専用の app パーティションのサブボリュームに展開します。

開発の時に以下の `abos-ctrl podman-rw` か `abos-ctrl podman-storage --disk` のコマンドを使って直接にイメージを編集することができます。



ここで紹介する内容はコンテナのイメージの管理の説明です。データベース等のコンテナから書き込みが必要な場合には「9.4.2.6. 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 9.173 `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 9.174 abos-ctrl podman-storage のイメージコピー例

- ❶ イメージを書き込み可能ストレージに取得します。
- ❷ abos-ctrl podman-storage をオプション無しで実行します。
- ❸ 書き込み可能ストレージにイメージがある場合に対応を聞かれます。今回はコピー (copy) します。
- ❹ abos-ctrl podman-storage にオプションを指定しなかったため、ストレージが tmpfs のままになります。すでに --disk で切り替えた場合にディスクのままでも可能です。
- ❺ コピーの確認します。イメージが読み取り専用 (R/O, Read only) になりました。



podman が壊れやすいので、デフォルトの「abos-ctrl podman-storage --tmpfs」で運用することを推奨しますが、tmpfs の容量が小さくてイメージの操作には向いてません。

開発時には「abos-ctrl podman-storage --disk」の状態で作業を行い、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、「9.5.1. コンテナの自動起動」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。

9.5.5.3. イメージを SWUpdate で転送する方法

1. イメージをファイルに保存する：

```
[armadillo ~]$ podman image save -o <myimage>.tar <localimage>
```

2. ファイルを SWUpdate のイメージに入れる。

二つのやり方があります：

- a. swu イメージ内に組み込む

```
[ATDE ~/mkswu]$ cp /usr/share/mkswu/examples/embed_container_nginx.desc .
[ATDE ~/mkswu]$ cp -r /usr/share/mkswu/examples/nginx_start .
[ATDE ~/mkswu]$ cat embed_container_nginx.desc
swdesc_option version=1

swdesc_embed_container "nginx_alpine.tar"
swdesc_files --extra-os nginx_start
[ATDE ~/mkswu]$ podman pull --arch 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。
```

9.6. マルチメディアデータを扱う

9.6.1. GStreamer - マルチメディアフレームワーク

9.6.1.1. 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。

9.6.2. 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「9.4.3. アットマークテクノが提供するイメージを使う」を参照してください。

```
[armadillo ~]# vi /etc/atmark/containers/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 9.175 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 9.176 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 9.177 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。

```
[container /]# apt install pulseaudio
[container /]# pulseaudio --start --exit-idle-time=-1
```

図 9.178 pulseaudio の起動

以上により、GStreamer をコンテナ内で実行できるようになります。

9.6.3. 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 9.179 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](http://gstreamer.freedesktop.org/documentation/) の Documentation ページ (<http://gstreamer.freedesktop.org/documentation/>)から参照することができます。

Armadillo-X2 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-X2 には、この VPU を使用するための GStreamer エlement 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。

9.6.4. 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 Element として `vpudec` を使うことができます。

9.6.4.1. 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 9.180 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 9.181 H.264/AVC 動画の再生(音声なし)

9.6.4.2. 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 9.182 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 9.183 VP8 動画の再生(音声なし)

9.6.4.3. 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 9.184 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 9.185 VP9 動画の再生(音声なし)

9.6.5. 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lement として vpudec を使うことができます。

9.6.5.1. 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 9.186 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 9.187 HTTP ストリーミングの再生(音声なし)

9.6.5.2. 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 9.188 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 9.189 RTSP ストリーミングの再生(音声なし)

9.6.6. 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 9.190 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 9.191 USB カメラからの映像表示(音声なし)

9.6.7. 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。

9.6.7.1. 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lement 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 9.192 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 9.193 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 9.194 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 9.195 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)

9.6.8. Video Processing Unit(VPU)

9.6.8.1. 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

9.6.8.2. VPU の仕様

- ・ H.264/AVC デコーダー

表 9.6 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 9.7 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 9.8 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 9.9 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: <ul style="list-style-type: none"> · 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: <ul style="list-style-type: none"> · 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 qoffset mode (-32 ~ 31) User controllable CU coded as IPCM CU or skip CU

9.7. Armadillo のソフトウェアをビルドする

ここでは、Armadillo-X2 で使用するソフトウェアのビルド方法を説明します。

9.7.1. ブートローダーをビルドする

ここでは、Armadillo-X2 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-X2 ブートローダー [<https://armadillo.atmark-techno.com/resources/software/armadillo-x2/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/armadillo-x2/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。^[2]

```
Welcome to NXP firmware-imx-8.11.bin

You need to read and accept the EULA before you can continue.

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

ビルドしたブートルoader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「9.10.3. イメージのインストール」を参照ください。

- ・ 「9.8.1. 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。

^[2]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。

9.7.2. Linux カーネルをビルドする

ここでは、Armadillo-X2 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-X2 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。「9.7.3. 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-X2 Linux カーネル [<https://armadillo.atmark-techno.com/resources/software/armadillo-x2/linux-kernel>] から「Linux カーネル」ファイル (linux-at-[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-[VERSION].tar
[ATDE ~]$ tar xf linux-at-[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。

```
.config - Linux/arm64 5.10.86 Kernel Configuration

----- Linux/arm64 5.10.86 Kernel Configuration -----
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus
----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M>
modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search.
Legend: [*] built-in [ ] excluded <M> module < > module capable

  General setup --->
  [*] Support DMA zone
  [*] Support DMA32 zone
  Platform selection --->
  Kernel Features --->
  Boot options --->
  Power management options --->
  CPU Power Management --->
  Firmware Drivers --->
  [ ] Virtualization ----
  -* ARM64 Accelerated Cryptographic Algorithms --->
  General architecture-dependent options --->
  [*] Enable loadable module support --->
  [*] Enable the block layer --->
  IO Schedulers --->
  Executable file formats --->
  Memory Management options --->
  [*] Networking support --->
  Device Drivers --->
  File systems --->
  Security options --->
  -* Cryptographic API --->
  Library routines --->
  Kernel hacking --->

  <Select>  < Exit >  < Help >  < Save >  < Load >
```



Linux Kernel Configuration メニューで"/"キーを押下すると、カーネルコンフィギュレーションの検索を行うことができます。カーネルコンフィギュレーションのシンボル名(の一部)を入力して"Ok"を選択すると、部分一致するシンボル名を持つカーネルコンフィギュレーションの情報が一覧されます。

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 9.196 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 「9.10.3. イメージのインストール」 を参照ください。



この kernel.swu をインストールする際は /etc/swupdate/preserve_files の更新例 の様に /boot と /lib/modules を維持するように追加します。カーネルをインストールした後に Armadillo Base OS を更新しても、この kernel.swu のカーネルが維持されます。

標準のカーネルに戻りたいか、以下の 「図 9.197. 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}
"$BROOTFS/ax2/resources/boot/"
'arch/arm64/boot/dts/freescale/armadillo_iotg_g4.dtb' -> '/home/atmark/build-rootfs-
v3.17-at.3/ax2/resources/boot/armadillo_iotg_g4.dtb'
'arch/arm64/boot/dts/freescale/armadillo_iotg_g4-at-dtweb.dtbo' -> '/home/atmark/buil-
d-rootfs-v3.17-at.3/ax2/resources/boot/armadillo_iotg_g4-at-dtweb.dtbo'
: (省略)
[ATDE ~/linux-[VERSION]]$ 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 9.197 Linux カーネルを build_rootfs でインストールする方法

- ❶ build_rootfs のディレクトリ名を設定します。これによって、長いディレクトリ名を何度も入力する必要がなくなります。
- ❷ アットマークテクノが提供するカーネルをインストールしない様に、 linux-at-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。



Armadillo-X2 は Armadillo-IoT ゲートウェイ G4 と同じ DTB で動作します。そのため、「9.12. Device Tree をカスタマイズする」や「9.12.4. DT overlay によるカスタマイズ」でも armadillo_iotg_g4- から始まる dtb を利用します。

この動作は、RAM 上にロードされた DTB をブートローダーが修正することで実現されています。

9.7.3. 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-X2 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-X2 開発用ツール [<https://armadillo.atmark-techno.com/resources/software/armadillo-x2/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/armadillo-x2/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 9.10 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。
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、「9.8.1. ブートディスクの作成」や「9.9.1.1. 初期化インストールディスクの作成」の手順のようにブートディスクイメージを作成する際、そのままルートファイルシステム直下にコピーされます。
ax2/image_installer/*	配置したファイルやディレクトリは、「9.9.1.1. 初期化インストールディスクの作成」の手順のようにインストールディスクイメージを作成する際、そのままインストーラにコピーされます。ルートファイルシステムに影響はありません。
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
```

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 「9.10.3. イメージのインストール」を参照ください。

- ・ 「9.8.1. 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。

9.8. 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。

9.8.1. 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

「9.7.3. 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 「9.7.1. 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、「8.3. VPU や NPU を使用する」で用意した `imx_lib.img` も組み込みます。

```
[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「4.2.2. 取り外し可能デバイスの使用」を参考にしてください。
3. microSD カードのデバイス名を確認します

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

4. microSD カードがマウントされている場合、アンマウントします。

```
[ATDE ~]$ mount
: (省略)
/dev/sdb1 on /media/52E6-5897 type ext2
(rw,nosuid,nodev,relatime,uid=1000,gid=1000,fmask=0022,dmask=0077,codepage=cp437,ioccharset=utf8,shortname=mixed,showexec=utf8,flush,errors=remount-ro,uhelper=udisks)
[ATDE ~]$ sudo umount /dev/sdb1
```



図 9.198 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 9.11 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
```

9.8.2. SD ブートの実行

「9.8.1. 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-X2 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-at11 (Jan 19 2023 - 10:53:24 +0000)
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):2020.04-at10-0-ge26bfd065
NOTICE: BL31: Built : 10:54:50, Jan 19 2023

U-Boot 2020.04-at11 (Jan 19 2023 - 10:53:24 +0000)

CPU:   i.MX8MP[8] rev1.1 1600 MHz (running at 1200 MHz)
CPU:   Industrial temperature grade (-40C to 105C) at 47C
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... *** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial

BuildInfo:
- ATF e26bfd0
- U-Boot 2020.04-at11

reset cause: normal reboot
switch to partitions #0, OK
mmc1 is current device
flash target is MMC:1
Net:
Warning: ethernet@30be0000 using MAC address from ROM
eth0: ethernet@30be0000 [PRIME]
Fastboot: Normal
Saving Environment to MMC... Writing to redundant 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
```

9.9. Armadillo のソフトウェアの初期化

microSD カードを使用し、Armadillo Base OS の初期化を行えます。



初期化を行っても、ファームウェアパーティション(mmcblk2p4)は変更されません。故障が疑われる場合など、ファームウェアも初期化したい場

合、初期化してから「8.3. VPU や NPU を使用する」を参考にしてもう一度書き込みしてください。

9.9.1. インストールディスクの作成

インストールディスクは二つの種類があります：

- ・ 初期化インストールディスク。Armadillo-X2 インストールディスクイメージ [https://armadillo.atmark-techno.com/resources/software/armadillo-x2/disc-image] にある標準のイメージです。
- ・ 開発が完了した Armadillo-X2 をクローンするためのインストールディスク。

9.9.1.1. 初期化インストールディスクの作成

1. 512 MB 以上の microSD カードを用意してください。
2. 標準のインストールディスクイメージを使用する場合は、Armadillo-X2 インストールディスクイメージ [https://armadillo.atmark-techno.com/resources/software/armadillo-x2/disc-image] から「Armadillo Base OS」をダウンロードしてください。

「9.7. 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「4.2.2. 取り外し可能デバイスの使用」を参考にしてください。
4. microSD カードのデバイス名を確認します

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

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,mask=0022,dmask=0077,codepage=cp437,ioccharset
=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。

9.9.1.2. 開発が完了した Armadillo をクローンするインストールディスクの作成

- microSD カードを用意してください。Armadillo-X2 にインストールされてるソフトウェアをコピーしますので、場合によって 8GB 以上のカードが必要です。
- 初期化インストールディスクをベースとしますので、「9.9.1.1. 初期化インストールディスクの作成」でビルドした SD カードを使用できますが、用意されていない場合は次のステップで自動的にダウンロードされます。
- abos-ctrl make-installer を実行してください

```

[armadillo ~]# abos-ctrl make-installer
It looks like your SD card does not contain an installer image
Download base SD card image from https://armadillo.atmark-techno.com (~200MB) ? [y/N]
WARNING: it will overwrite your sd card!!
y
Downloading installer image
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
   Dload  Upload   Total   Spent    Left     Speed
100 167M  100 167M    0     0  104M      0  0:00:01  0:00:01  --:--:-- 104M
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
   Dload  Upload   Total   Spent    Left     Speed
100   70  100   70    0     0   1441      0  --:--:--  --:--:--  --:--:-- 1458
Writing baseos-x2-installer-3.15.4-at.6.img to SD card (442M)
439353344 bytes (439 MB, 419 MiB) copied, 134 s, 3.3 MB/s
421+0 records in
421+0 records out
441450496 bytes (441 MB, 421 MiB) copied, 134.685 s, 3.3 MB/s
Verifying written image is correct
436207616 bytes (436 MB, 416 MiB) copied, 46 s, 9.5 MB/s
421+0 records in
421+0 records out
441450496 bytes (441 MB, 421 MiB) copied, 46.8462 s, 9.4 MB/s
Checking and growing installer main partition
GPT data structures destroyed! You may now partition the disk using fdisk or
other utilities.
Setting name!
partNum is 0
The operation has completed successfully.

```

```
e2fsck 1.46.4 (18-Aug-2021)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
rootfs_0: 2822/102400 files (0.5% non-contiguous), 352391/409600 blocks
(1/1) Installing e2fsprogs-extra (1.46.4-r0)
Executing busybox-1.34.1-r5.trigger
OK: 202 MiB in 197 packages
resize2fs 1.46.4 (18-Aug-2021)
Resizing the filesystem on /dev/mmcblk1p1 to 15547884 (1k) blocks.
The filesystem on /dev/mmcblk1p1 is now 15547884 (1k) blocks long.

Currently booted on /dev/mmcblk2p1
Copying boot image
Copying rootfs
301989888 bytes (302 MB, 288 MiB) copied, 10 s, 30.1 MB/s
300+0 records in
300+0 records out
314572800 bytes (315 MB, 300 MiB) copied, 10.3915 s, 30.3 MB/s
Copying /opt/firmware filesystem
Copying appfs
At subvol app/snapshots/volumes
At subvol app/snapshots/boot_volumes
At subvol app/snapshots/boot_containers_storage
Cleaning up and syncing changes to disk...
Installer updated successfully!
```

9.9.2. 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。

9.10. Armadillo のソフトウェアをアップデートする

Armadillo-X2 では、開発・製造・運用それぞれに適した複数のソフトウェアアップデート方法を用意しています。本章では、それぞれのソフトウェアアップデート方法について説明します。

ソフトウェアアップデートを実現するソフトウェアの概要や仕様、用語については「12. ソフトウェア仕様」を参照してください。

9.10.1. SWU イメージとは

Armadillo Base OS ではソフトウェアアップデートのために OS やコンテナ等を格納するために SWU というイメージ形式を使います。

SWU イメージは `swupdate` (<https://sbabic.github.io/swupdate/swupdate.html>) によって Armadillo Base OS 上で検証とインストールが実行されます。SWU イメージを Armadillo に転送する

ための方法は、用途や状況に合わせて様々な方法を用意しています。例えば、USB メモリから読み取る、ウェブサーバーからダウンロードする、hawkBit という Web アプリケーションを使うなどです。

9.10.2. SWU イメージの作成

SWU イメージの作成には、mkswu というツールを使います。

mkswu に含まれる mkswu を実行すると、アップデート対象やバージョン等の情報を記載した .desc ファイルに含まれる命令を順次実行してイメージを作り上げます。

詳しくは「9.10.6. mkswu の desc ファイル」を参考にしてください。

1. mkswu の取得

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

インストール済みの場合は、以下のコマンドを実行し最新版への更新を行ってください。

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



git のバージョンからアップデートする場合、mkswu --import で以前使っていたコンフィグをロードしてください。

```
[ATDE ~/swupdate-mkimage]$ mkswu --import
コンフィグファイルを更新しました: /home/atmark/swupdate-mkimage/
mkswu.conf
/home/atmark/swupdate-mkimage/mkswu.conf のコンフィグファイルとそ
の鍵を
/home/atmark/mkswu にコピーします。
mkdir: ディレクトリ '/home/atmark/mkswu' を作成しました
'/home/atmark/swupdate-mkimage/swupdate.key' -> '/home/atmark/
mkswu/swupdate.key'
'/home/atmark/swupdate-mkimage/swupdate.pem' -> '/home/atmark/
mkswu/swupdate.pem'
/home/atmark/swupdate-mkimage/mkswu.conf のコンフィグファイルを
/home/atmark/mkswu/mkswu.conf にコピーしました。
mkswu でイメージ作成を試してから前のディレクトリを消してください。
```

2. 最初に行う設定

mkswu --init を実行して鍵や最初の書き込み用のイメージを生成します。作成する鍵は、swu パッケージを署名するために使用します。

過去に本手順を行っている場合、再度初回アップデート作業を行う必要はありません。再度アップデートを行う際には、Armadillo に配置した公開鍵に対応する秘密鍵でアップデートを行いますので、「9.10.6. mkswu の desc ファイル」を参考にしてください。

```
[ATDE ~]$ mkswu --init
mkdir: ディレクトリ '/home/atmark/mkswu' を作成しました
```

```

コンフィグファイルを更新しました：/home/atmark/mkswu/mkswu.conf
証明書の Common name を入力してください： [COMMON_NAME] ❶
証明書の鍵のパスワードを入力ください（4-1024 文字） ❷
証明書の鍵のパスワード（確認）：
Generating an EC private key
writing new private key to '/home/atmark/mkswu/swupdate.key'
-----
アップデートイメージを暗号化しますか？（N/y） ❸
アットマークテクノが作成したイメージをインストール可能にしますか？（Y/n） ❹
root パスワード： ❺
root パスワード（確認）：
atmark ユーザのパスワード（空の場合はアカウントをロックします）： ❻
atmark ユーザのパスワード（確認）：
BaseOS イメージの armadillo.atmark-techno.com サーバーからの自動アップデートを行いますか？
(y/N) ❼
/home/atmark/mkswu/initial_setup.swu を作成しました。

"/home/atmark/mkswu/initial_setup.swu" をそのまま使うことができますが、
モジュールを追加してイメージを再構築する場合は次のコマンドで作成してください：
mkswu "/home/atmark/mkswu/initial_setup.desc" [他の.desc ファイル]


インストール後は、このディレクトリを削除しないように注意してください。
鍵を失うと新たなアップデートはデバイスの /etc/swupdate.pem
を修正しないとインストールできなくなります。

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

- ❶ COMMON_NAME には証明鍵の「common name」として会社や製品が分かるような任意の名称を入力してください。
- ❷ 証明鍵を保護するパスフレーズを 2 回入力します。
- ❸ swu イメージ自体を暗号化する場合に「y」を入力します。詳細は「9.10.9. SWUpdate と暗号化について」を参考にしてください。
- ❹ アットマークテクノのアップデートをインストールしない場合は「n」を入力します。
- ❺ root のパスワードを 2 回入力します。
- ❻ atmark ユーザーのパスワードを 2 回入力します。何も入力しない場合はユーザーをロックします。
- ❼ 自動アップデートを無効のままに進みます。ここで「y」を入れると、定期的にアットマークテクノのサーバーからアップデートの有無を確認し、自動的にインストールします。
- ❽ 作成したファイルを確認します。「swupdate.aes-key」は暗号化の場合にのみ作成されます。

このイメージは初回インストール用の署名鍵を使って、作成した鍵とユーザーのパスワードを設定します。

インストール後にコンフィグの mkswu.conf と鍵の swupdate.* をなくさないようにしてください。



このイメージに他の変更も入れれます。他の /usr/share/mkswu/examples/ ディレクトリにある.desc ファイルや「9.10.6. mkswu

の desc ファイル」を参考にして、以下の例のように同じ swu にいくつかの desc を組み込みます。

例えば、openssh を有効にします。

```
[ATDE ~/mkswu]$ cp -rv /usr/share/mkswu/examples/enable_sshd* .
: (省略)
'/usr/share/mkswu/examples/enable_sshd/root/.ssh/
authorized_keys'
-> './enable_sshd/root/.ssh/authorized_keys'
'/usr/share/mkswu/examples/enable_sshd.desc' -> './
enable_sshd.desc'
[ATDE ~/mkswu]$ cp ~/.ssh/id_rsa.pub ¥
enable_sshd/root/.ssh/authorized_keys
[ATDE ~/mkswu]$ mkswu initial_setup.desc enable_sshd.desc
enable_sshd.desc を組み込みました。
initial_setup.swu を作成しました。
```

3. イメージのインストール

「9.10.3. イメージのインストール」を参考に、作成したイメージをインストールしてください。

4. 次回以降のアップデート

次回以降のアップデートは作成した証明鍵を使用して Armadillo-X2 の SWU イメージを作成します。

.desc ファイルの内容は /usr/share/mkswu/examples/ のディレクトリや「9.10.6. mkswu の desc ファイル」を参考にしてください。

9.10.3. イメージのインストール

イメージをインストールする方法として以下に示すような方法があります。もし、作成した SWU イメージのインストールに失敗する場合は、「9.10.4. swupdate がエラーする場合の対処」をご覧ください。

・ USB メモリまたは SD カードからの自動インストール

Armadillo-X2 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-X2 は自動で再起動します。

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



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

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

```
[ATDE ~/mkswu]$ df -h
Filesystem      Size  Used Avail Use% Mounted on
```

```

: (省略)
/dev/sda1      15G  5.6G  9.1G  39% /media/USBDRIVE ❶
[ATDE ~/mkswu]$ cp initial_setup.swu /media/USBDRIVE/ ❷
[ATDE ~/mkswu]$ umount /media/USBDRIVE ❸

```

- ❶ USB メモリがマウントされている場所を確認します。
- ❷ ファイルをコピーします。
- ❸ /media/USBDRIVE をアンマウントします。コマンド終了後に USB メモリを取り外してください。

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

```

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

```

- ❶ 証明が間違ったメッセージ。

- ・ 外部記憶装置からイメージのインストール (手動)

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

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

```

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

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

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

```

- ・ ウェブサーバーからイメージのインストール (手動)

swu イメージをウェブサーバーにアップロードして、イメージのインストールを行います。以下は、http://server/initial_setup.swu のイメージをインストールする例です。

```

[armadillo ~]# swupdate -d '-u http://server/initial_setup.swu'
SWUpdate v5f2d8be-dirty

```

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

```
[INFO ] : SWUPDATE running : [main] : Running on AGX4500 Revision at1
[INFO ] : SWUPDATE running : [channel_get_file] : Total download size is 25 kB.
[INFO ] : SWUPDATE started : Software Update started !
[INFO ] : SWUPDATE running : [read_lines_notify] : No base os update: copying current os over
[INFO ] : SWUPDATE running : [read_lines_notify] : Removing unused containers
[INFO ] : SWUPDATE running : [read_lines_notify] : swupdate triggering reboot!
Killed
```

・ ウェブサーバーからの定期的な自動インストール

swupdate-url を有効にしたら、定期的にチェックしてインストールします。以下はサービスの有効化とタイミングの設定の例です。

```
[armadillo ~]# rc-update add swupdate-url ❶
[armadillo ~]# persist_file /etc/runlevels/default/swupdate-url ❷
[armadillo ~]#
    echo https://download.atmark-techno.com/armadillo-x2/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-X2 を複数台管理してアップデートすることができます。「9.10.5. hawkBit サーバーから複数の Armadillo に配信する」を参考にしてください。

9.10.4. swupdate がエラーする場合の対処

SWU イメージのインストール動作は、「9.10.1. SWU イメージとは」で述べたように swupdate が実行します。mkswu で作成した SWU イメージの内容が適切でなかったり、あるいは、ストレージの空

き容量が不足していたりするなど、いくつかの理由で swupdate のインストール動作が失敗することがあります。インストールに失敗すると、swupdate は /var/log/messages にエラーメッセージのログを残しますので、エラーメッセージを見ると、エラーの内容・原因が分かります。

エラーの原因ごとに、エラーメッセージとエラーの内容および対処方法を記した FAQ ページ (<https://armadillo.atmark-techno.com/faq/swupdate-troubleshooting-abos>) を公開しています。SWU イメージのインストールに失敗して対処法が分からないときは、この FAQ ページをご覧ください。

9.10.5. hawkBit サーバーから複数の Armadillo に配信する

hawkBit サーバーを利用することで複数の Armadillo のソフトウェアをまとめてアップデートすることができます。

手順は次のとおりです。

1. コンテナ環境の準備

Docker を利用すると簡単にサーバーを準備できます。Docker の準備については <https://docs.docker.com/get-docker/> を参照してください。

Docker の準備ができたなら、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-X2 開発用ツール [<https://armadillo.atmark-techno.com/resources/software/armadillo-x2/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 9.199 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「9.10.5.2. SWU で hawkBit を登録する」を参考にしてください。
- ❼ hawkbit_push_update でアップデートを CLI で扱う場合は、「Y」を入力してください。詳細は <sct.hawkbit_push_update>> を参照してください。
- ❽ hawkbit_push_update でアップデートを実行する場合は、「Y」を入力してください。
- ❾ ここでは http でテストのコンテナを作成するので、「N」のままで進みます。
- ❿ コンテナを起動します。初期化が終わったら <IP>:8080 でアクセス可能になります。

```

[ATDE ~]$ /usr/share/mkswu/hawkbit-compose/setup_container.sh
docker-compose の設定ファイルと hawkBit のデータをどこに保存しますか? [/home/atmark/hawkbit-
compose]
setup_container.sh へのリンクを /home/atmark/hawkbit-compose に作ります。
docker サービスに接続できませんでした。sudo でもう一度試します。
OK!
    
```



```

Hawkbit admin user name [admin]
admin ユーザーのパスワード:
パスワードを再入力してください:
パスワードが一致しません。
admin ユーザーのパスワード:
パスワードを再入力してください:
追加の管理人アカウントのユーザーネーム (空にすると追加しません)
hawkBit の「device」ユーザーを登録しますか? (自動登録用) [Y/n]
device ユーザーのパスワード:
パスワードを再入力してください:
hawkBit の「mkswu」ユーザーを登録しますか? (swu のアップロード用) [Y/n]
ユーザーにロールアウトの権限を与えますか? (インストール要求を出すこと) [Y/n]
mkswu ユーザーのパスワード:
パスワードを再入力してください:
Setup TLS reverse proxy? [y/N] y ❶
lighttpd が起動中で、リバースプロキシ設定と競合しています。
lighttpd サービスを停止しますか? [Y/n] ❷
Synchronizing state of lighttpd.service with SysV service script with /lib/systemd/systemd-
sysv-install.
Executing: /lib/systemd/systemd-sysv-install disable lighttpd
Removed /etc/systemd/system/multi-user.target.wants/lighttpd.service.
リバースプロキシの設定に証明書の domain name が必要です。
この domain はこのままデバイスからアクセスできる名前にしてください。
例えば、https://hawkbit.domain.tld でアクセスしたら hawkbit.domain.tld、
https://10.1.1.1 でしたら 10.1.1.1 にしてください。
証明書の domain name: 10.1.1.1 ❸
証明書の有効期限を指定する必要があります。Let's encrypt を使用する場合、
この値は新しい証明書が生成されるまでしか使用されないの、デフォルトの値
のままにしておくことができます。Let's encrypt を使用しない場合、
数年ごとに証明書を新しくすることが最も好まれます。
証明書の有効期間は何日間になりますか? [3650] ❹
クライアントの TLS 認証を設定するために CA が必要です。
署名 CA のファイルパス (空にするとクライアント TLS 認証を無効になります) [] ❺
サーバーが直接インターネットにアクセス可能であれば、Let's Encrypt の証明書
を設定することができます。TOS への同意を意味します。
https://letsencrypt.org/documents/LE-SA-v1.2-November-15-2017.pdf
certbot コンテナを設定しますか? [y/N] ❻

/home/atmark/hawkbit-compose/data/nginx_certs/proxy.crt を /usr/local/share/ca-
certificates/ にコピーして、 update-ca-certificates を実行する必要があります。
この base64 でエンコードされたコピーを examples/hawkbit_register.sh の
SSL_CA_BASE64 に指定する手順が推奨されます。

LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUJlekNDQVNHZ0F3SUJBZ0lVQTByZ0cwcTJF
SFNnampb0tUZWg3aGlaSVVVd0NnWUllb1pJemowRUF3SXcKRXpFuk1B0EdBMVVFQXd3SU1UQXVN
UzR4TGpFd0hoY05Nakl3TXpJMU1EVXhOVFU0V2hjTk16SXdNekl5TURVeApOVFU0V2pBVE1SRXdE
d1lEVlFRERBZ3hNQzR4TGpFdU1UQlpNQk1HQnlxR1NNNDlBZ0VHQ0Nxr1NNNDlBd0VICkEwSUFc
SDFFRhBN3NOTlFJUDlTdLhLUUnNmWj12dVVFVWRkRkMVE2TzViRlV2RTh4UjUwUjBCLzNlajMzd0VI
NEoKYmZqb296bEpXaExlSG5SbGZsaHEXVDlKdm5TaLV6QlJNQjBHQTFVZERnUdCQlFBUMYvSkdT
dkVJek5xZ2JMNQpQamY2VGRpSk1EQWZCZ05W5FNRRUdEQVdnQlFBUMYvSkdTdkVJek5xZ2JMNVBq
ZjZUZGkTURBUEJnTlZIUk1CCkFm0EVCVEFEQVFILO1Bb0dDQ3FHU000OUJBTUNBMGdBTUVVQ0LD
Nis3ZzJlZk1SRXl0RVk5WDhDNC8vUEw1U1kKWUlGZHUxVFZiUEZrSlV0SUFpRUE4bm1VSnVQSFZl
SHg2N2ErZFRwSXZlQmJUSG1KbWd6dU13bTJ2RjRppRnZRPQotLS0tLUVORCBDRVJUSUZJQ0FURS0t
LS0tCg== ❷
    
```



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。

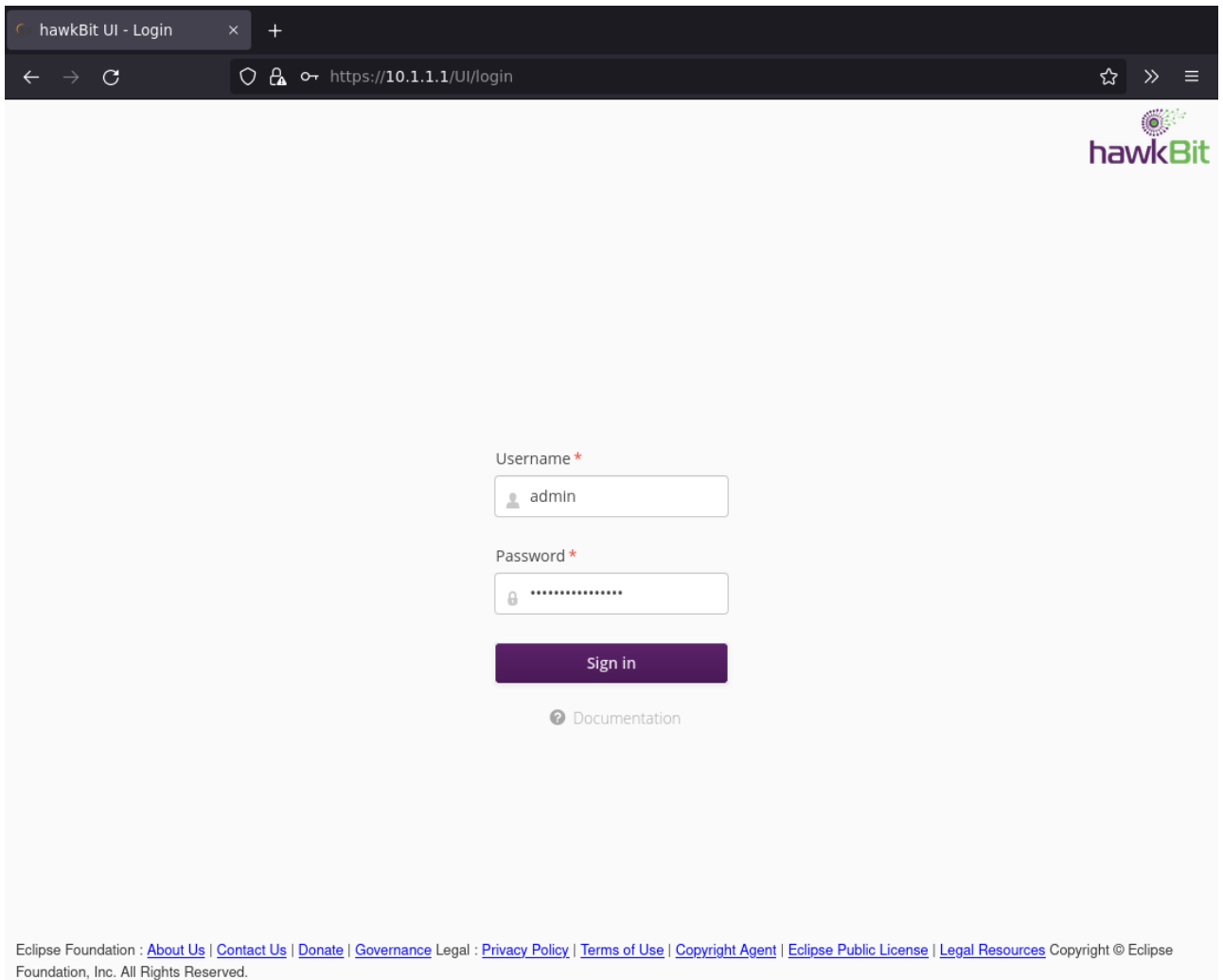
コンテナの設定が完了しました。 `docker-compose` コマンドでコンテナの管理が可能です。
`/home/atmark/hawkbit-compose/setup_container.sh` を再び実行すると設定の変更が可能です。
hawkBit コンテナを起動しますか？ [Y/n]

図 9.200 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「9.10.5.2. SWU で hawkBit を登録する」を参照してください。

3. hawkBit へのログイン

作成したコンテナによって `http://<サーバーの IP アドレス>:8080` か `https://<サーバーのアドレス>` にアクセスすると、ログイン画面が表示されます。

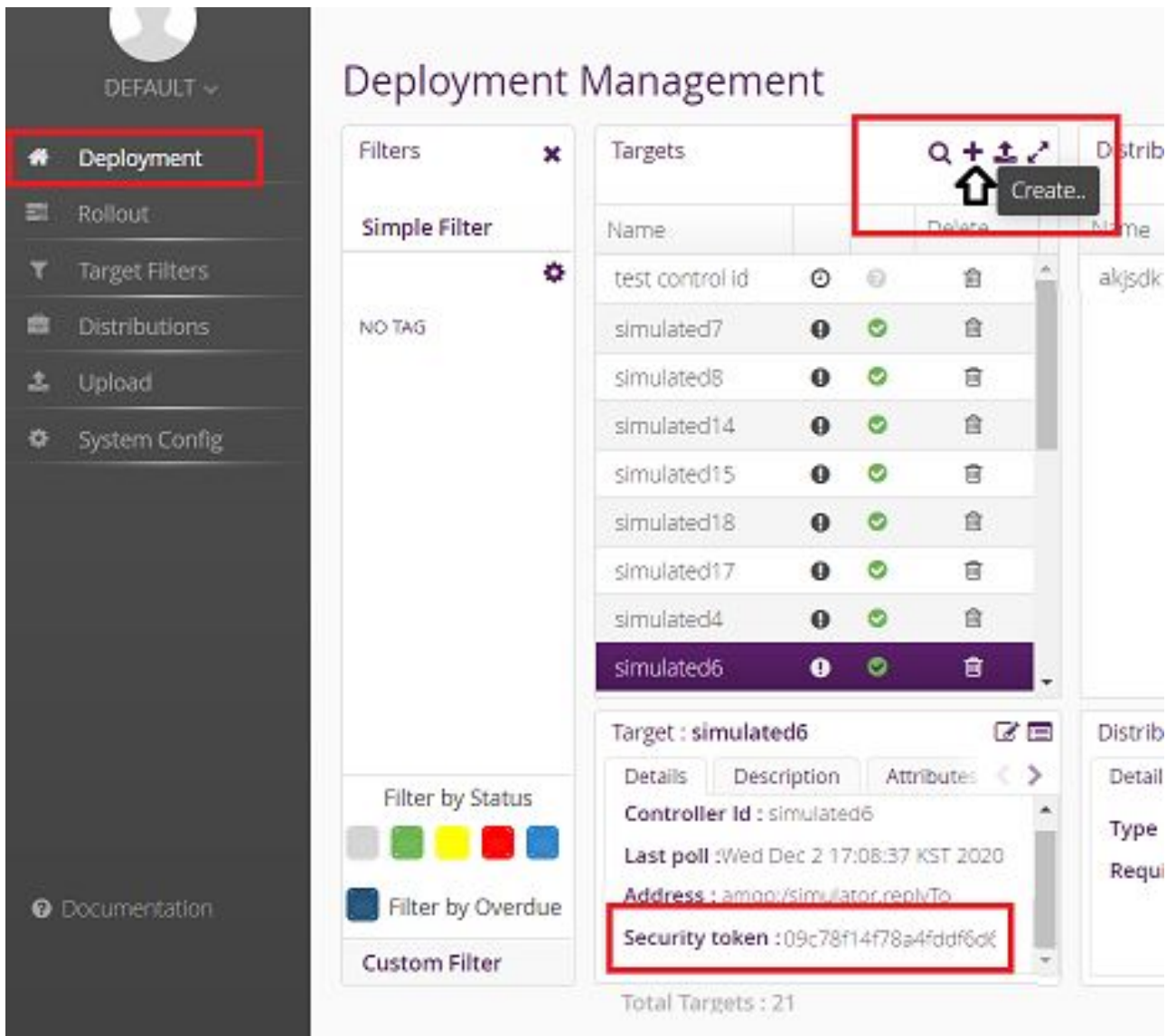


デフォルトでは次のアカウントでログインできます。

ユーザー	admin
パスワード	admin

4. Armadillo を Target に登録する

左側のメニューから Deployment をクリックして、Deployment の画面に移ります。



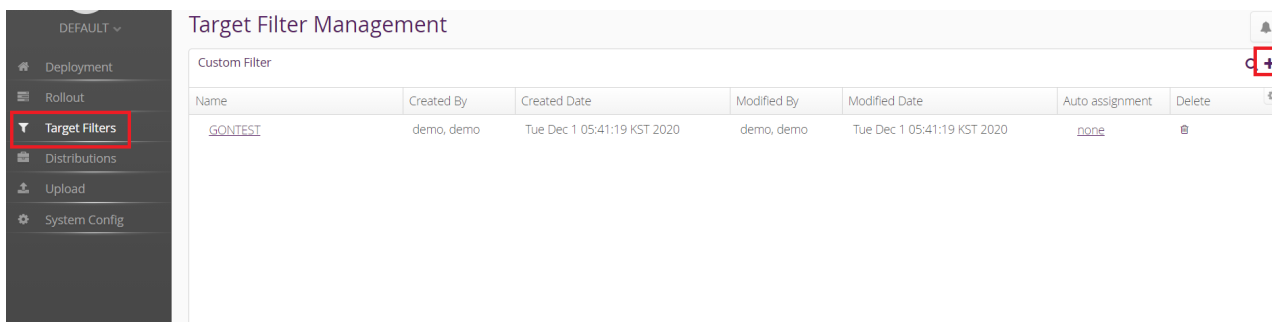
"+"をクリックして Target を作成します。

作成したターゲットをクリックすると、下のペインに "Security token:<文字列>" と表示されるので、<文字列>の部分メモします。

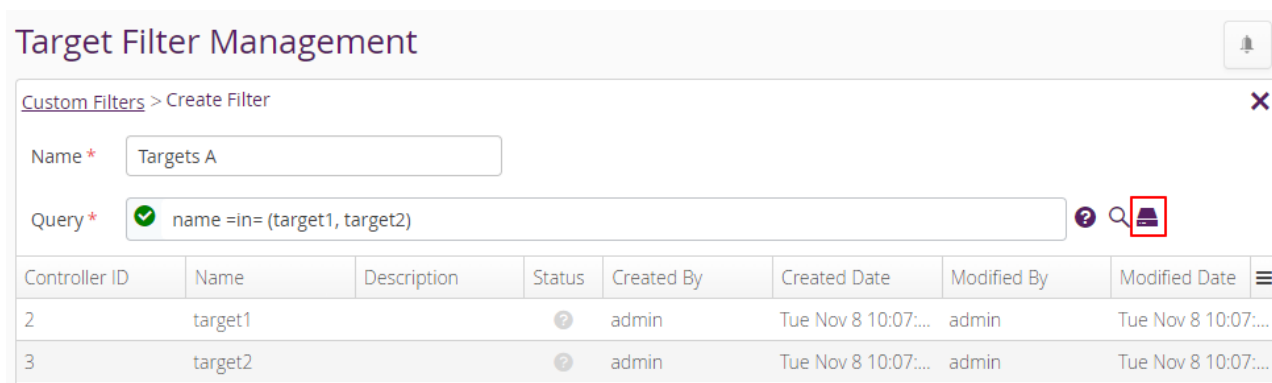
メモした<文字列>を Armadillo の /etc/swupdate.cfg に設定すると Hawkbit への接続認証が通るようになります。

5. Target Filter を作成する

左側のメニューから"Target Filters"をクリックして、Target Filters の画面に移ります。



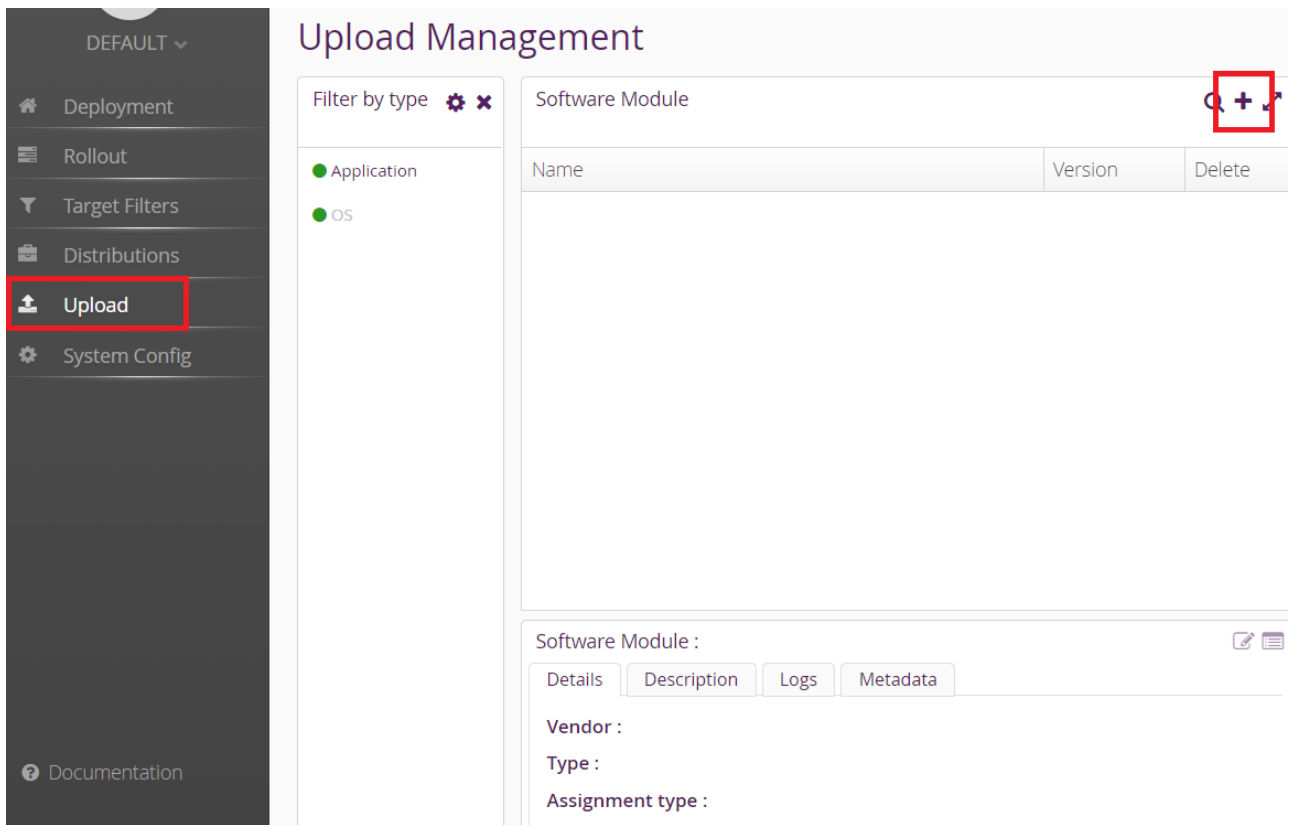
"+" をクリックして新規に Target Filter を作成します。



Filter name と フィルタリング条件を入力して保存します。

6. Software module を作成する

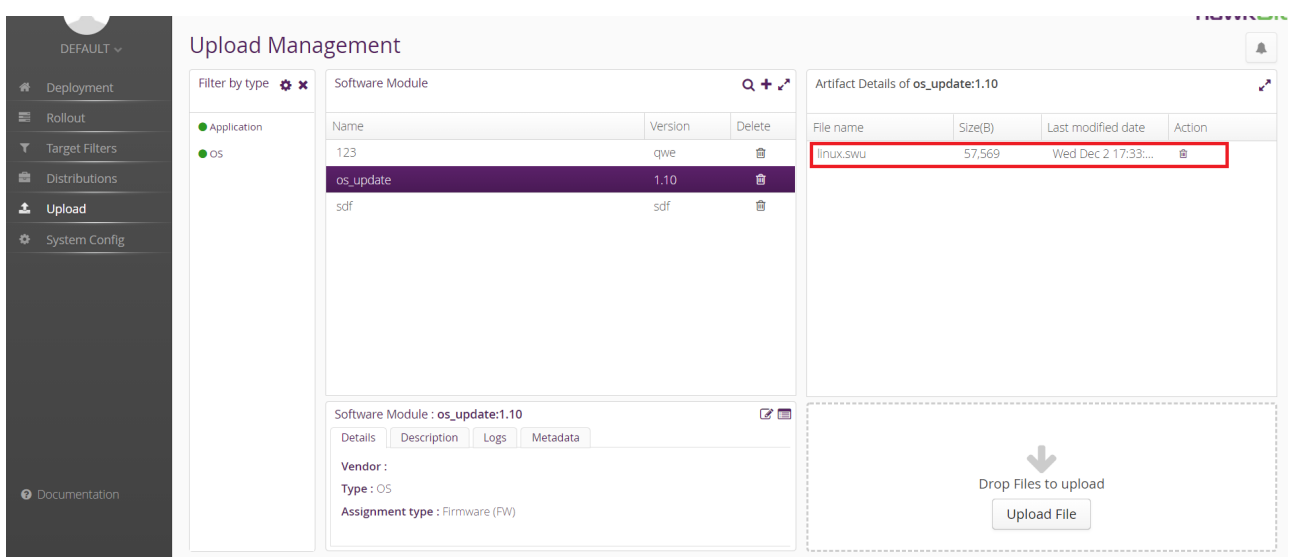
左側のメニューから"Upload"をクリックして、Upload Management の画面に移ります。



"+" をクリックして Software module を作成します。type には OS/Application、version には任意の文字列を指定します。

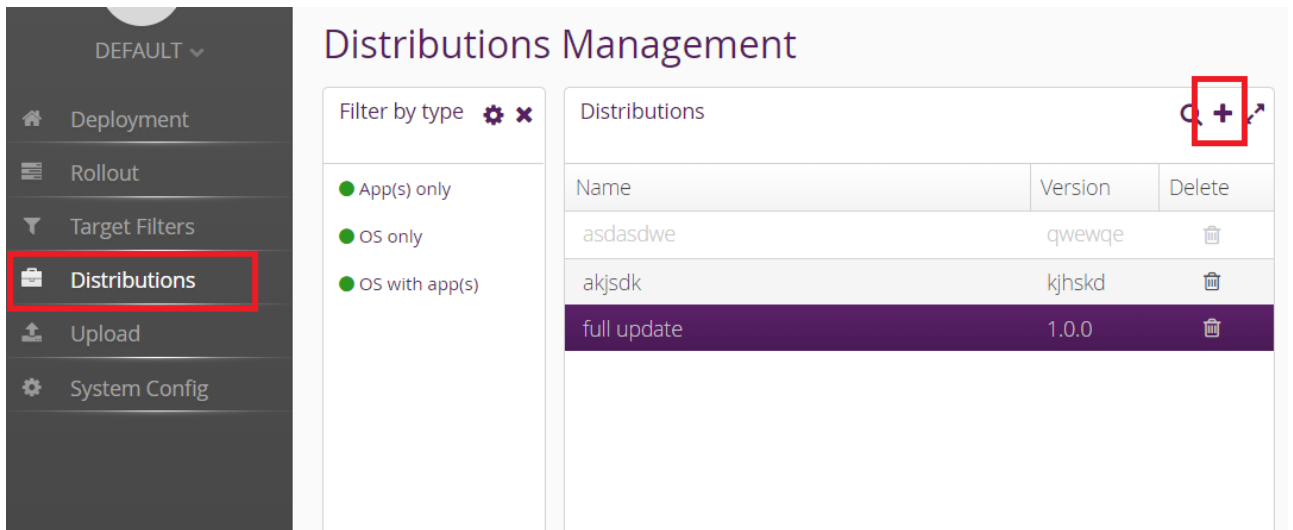
7. swu パッケージをアップロードして Software module に関連付ける

先程作成した Software module を選択して、ハイライトされた状態で、"Upload File"ボタンをクリックするか、ファイルをドラッグアンドドロップしてアップロードします。



8. Distribution を作成して Software module を関連付ける

左側のメニューから"Distribution"をクリックして、Distribution Management の画面に移ります。



The screenshot shows the 'Distributions Management' interface. On the left, a sidebar menu lists 'Deployment', 'Rollout', 'Target Filters', 'Distributions' (highlighted with a red box), 'Upload', and 'System Config'. The main content area is titled 'Distributions Management' and contains a table of distributions. The table has columns for 'Name', 'Version', and 'Delete'. The table contains three rows: 'asdasdwe' with version 'qwewqe', 'akjsdk' with version 'kjhsdk', and 'full update' with version '1.0.0'. A red box highlights the '+' icon in the top right corner of the table, indicating the button to create a new distribution.

Name	Version	Delete
asdasdwe	qwewqe	
akjsdk	kjhsdk	
full update	1.0.0	

"+" をクリックして Distribution を作成します。type には OS/OSwithApp/Apps、version には任意の文字列を指定します。

Create new Distribution ✕

Select Type ▼

Name *

Version *

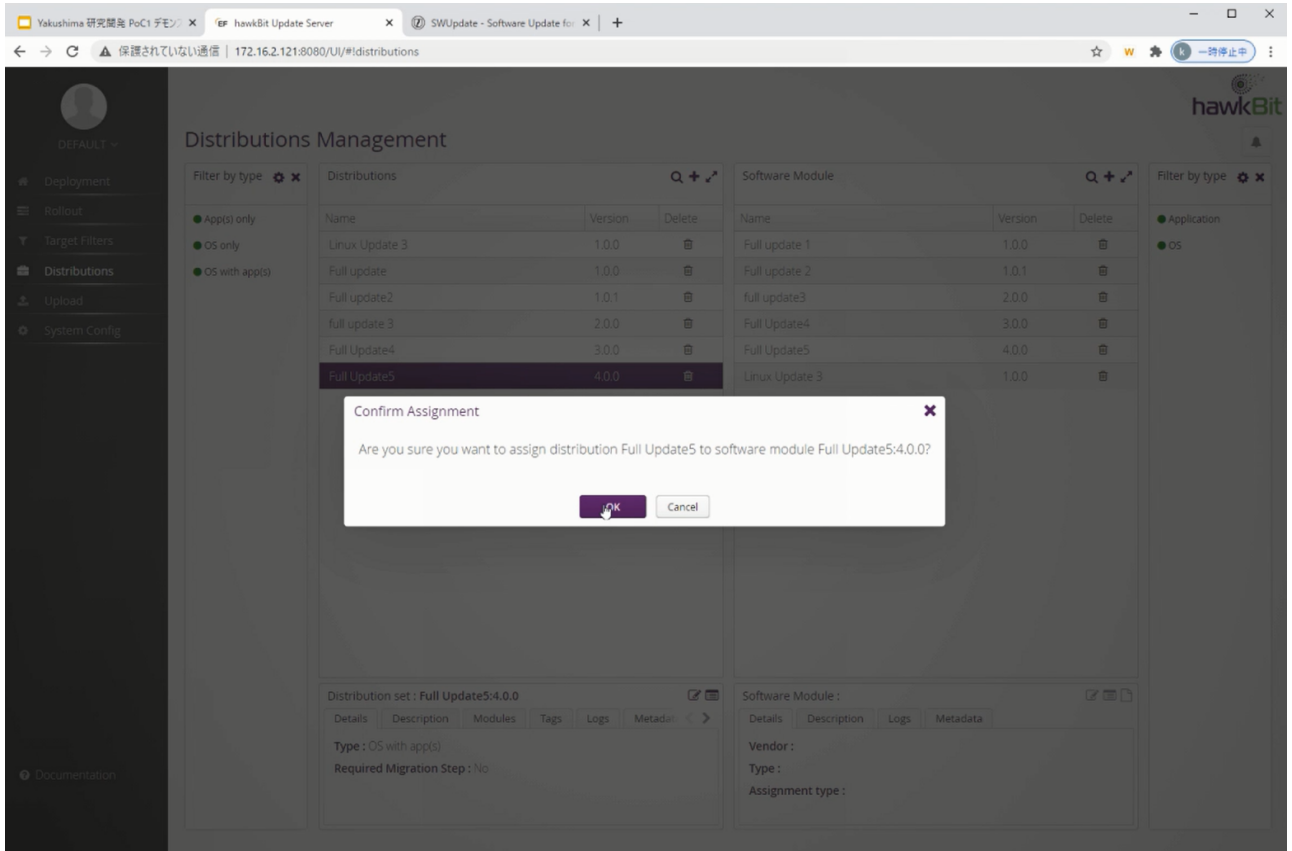
Description

Required Migration Step

* Mandatory Field

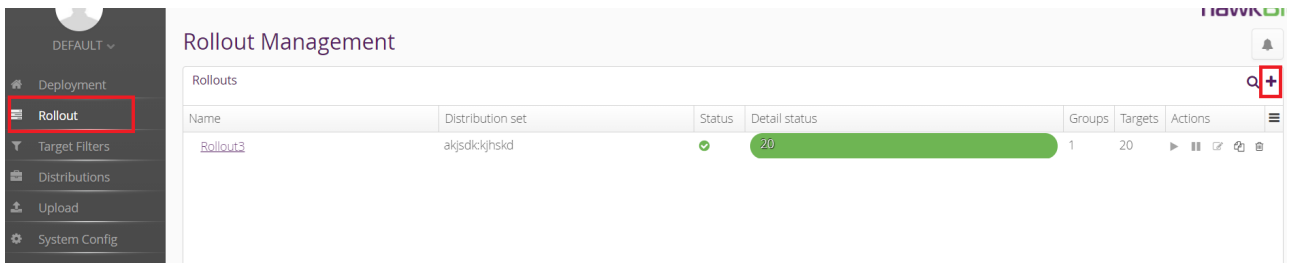
 Save  Cancel

"Software module"のペインから先程作成した Software をドラッグして、作成した Distribution の上にドロップします。



9. Rollout を作成してアップデートを開始する

左側のメニューから"Rollout"をクリックして、Rollout Management の画面に移ります。



"+"をクリックして Rollout を作成します。

Create new Rollout ✕

Name * *

Distribution set * ▼


Custom Target Filter * ▼

Description

Action type * ⚡ Forced ⏸ Soft ⌚ Time Forced ⬇️ Download Only

Start type * 📁 Manual ▶ Auto ⌚ Scheduled

Number of Groups Advanced Group definition



Total Targets : 20
20 in Group 1

Generate the groups automatically with the specified thresholds.

Number of groups * Targets per group :20

Trigger threshold * %

Error threshold * % Count

* Mandatory Field

📁 Save ✕ Cancel ?

項目	説明
Name	任意の文字列を設定します。
Distribution Set	先程作成した Distribution を選択します。
Custom Target Filter	先程作成した Target Filter を選択します。
Action Type	アップデート処理をどのように行うかを設定します。 ・ Forced/Soft: 通常のアップデート ・ Time Forced: 指定した時刻までにアップデートする ・ Download only: ダウンロードのみ行う
Start Type	Rollout の実行をどのように始めるかを設定します。 ・ Manual: 後で手動で開始する ・ Auto: Target からのハートビートで開始する ・ Scheduled: 決まった時間から開始する

10. アップデートの状態を確認する

Rollout Management の画面の Detail Status で、各 Rollout のアップデートの状態を確認できます。

アップデート中は黄色、アップデートが正常に完了すると緑色になります。

9.10.5.1. hawkBit のアップデート管理を CLI で行う

一つのアップデートを登録するには、hawkBit の Web UI で必要な手順が長いので CLI で行うことで効率よく実行できます。

サーバーの設定の段階では、「mkswu」のユーザーを作成する必要があります。作成していない場合は `setup_container.sh --add-user mkswu` で作成してください。

1. hawkbit_push_update の実行例

```
[ATDE ~/mkswu]$ ls enable_sshd.swu ❶
enable_sshd.swu

[ATDE ~/mkswu]$ hawkbit_push_update --help
Usage: /usr/bin/hawkbit_push_update [options] file.swu

rollout creation:
  --no-rollout: only upload the file without creating a rollout ❷
  --new: create new rollout even if there already is an existing one ❸
  --failed: Apply rollout only to nodes that previously failed update ❹

post action:
  --start: start rollout immediately after creation ❺

[ATDE ~/mkswu]$ hawkbit_push_update --start enable_sshd.swu ❻
Uploaded (or checked) image extra_os.sshd 1 successfully
Created rollout extra_os.sshd 1 successfully
Started extra_os.sshd 1 successfully
```

- ❶ この例ではあらかじめ作成されている `enable_sshd.swu` を hawkBit に登録します。
- ❷ `--no-rollout` を使う場合に SWU を「distribution」として登録します。デフォルトでは rollout も作成します。テストする際、デバイスがまだ登録されていなければ rollout の段階で失敗します。
- ❸ 同じ SWU で rollout を二回作成した場合にエラーが出ます。もう一度作成する場合は `--new` を使ってください。
- ❹ 一度 rollout をスタートして、Armadillo で失敗した場合には失敗したデバイスだけに対応した rollout を作れます。
- ❺ 作成した rollout をすぐ実行します。このオプションには追加の権限を許可する必要があります。
- ❻ スタートまで行う実行例です。実行結果は Web UI で表示されます。

9.10.5.2. SWU で hawkBit を登録する

デバイスが多い場合は、SWU を一度作って armadillo を自己登録させることができます。

サーバーの設定の段階では、「device」のユーザーを作成する必要があります。作成していない場合は `setup_container.sh --add-user device` で作成してください。

1. hawkbit_register.desc で hawkBit の自己登録を行う例

```
[ATDE ~]$ cd mkswu/
```



```
[ATDE ~/mkswu]$ cp /usr/share/mkswu/examples/hawkbit_register.* . ❶

[ATDE ~/mkswu]$ vi hawkbit_register.sh ❷
# Script configuration: edit this if required!
# user given here must have CREATE_TARGET, READ_TARGET_SECURITY_TOKEN permissions
HAWKBIT_USER=device
HAWKBIT_PASSWORD="CS=wC, zJmrQeeKT.3" ❸
HAWKBIT_URL=https://10.1.1.1 ❹
HAWKBIT_TENANT=default
# set custom options for suricatta block or in general in the config
CUSTOM_SWUPDATE_SURICATTA_CFG="" # e.g. "polldelay = 86400;"
CUSTOM_SWUPDATE_CFG=""
# set to non-empty if server certificate is invalid
SSL_NO_CHECK_CERT=
# or set to cafile that must have been updated first
SSL_CAFILE=
# ... or paste here base64 encoded crt content
SSL_CA_BASE64="
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUJlakNDQVNHZ0F3SUJBZ0lVYTMvYXpNSHZ0
bFFnaFZnZDhIZWhMaEwxNm5Bd0NnWUllb1pJemowRUF3SXcKRXpFuk1BOEdBMVVFQXd3SU1UQXVn
UzR4TGpFd0hoY05Nakl3TWpFNE1EVTFNakV6V2hjTk16SXdNakUyTURVMQpNakV6V2pBVE1SRXdE
d1lEVlFRFRERBz3hNzR4TGpFdU1UQlNk1HQnlxR1NNNDLBZ0VHQ0NzR1NNNDLB0VICKwSUFc
RFJGcnJVv3hHNnBhdWVoejRkRzVqYkVWtm5scHUwYXBHT1c3UUVBPUYU4cWp1ZzJWYjk2UHNScWJY
Sk8KbEFdVVo20StaMhk3c1BqeDJHYnhDNms0czFHaLV6QLJNQjBHQTFVZERnUvdCQlJtZzhxL2FV
OURRc3EvTGE1TgpaWFdkTHROUmNEQWZCZ05WSFNRRUdEQVdnQlJtZzhxL2FVOURRc3EvTGE1TlpY
V2RMdE5SY0RBUEJnTlZlUk1CCkFm0EVCVEFEQVFIL01Bb0dDQ3FHU000UJBTUNBMGNBTUVRQ0LB
ZTRCQ0xKREpWZnFTQVdRcVBqNTFmMjJvQkYKRmVBbVlGY2VBMU45dE8rN0FpQXVvUEV1VGFxWjhH
UFYyRUg1UWd0MFRKS05SckJDOEtpNkZwcFlkRUowYWc9PQotLS0tLUVORCBDRVJUSUZJQ0FURS0t
LS0tCg== ❺
"

# ... or add your own options if required
CURLLOPT=-s

: (省略)

[ATDE ~/mkswu]$ cat hawkbit_register.desc ❻
: (省略)
swdesc_script hawkbit_register.sh --version extra_os.hawkbit 1

[ATDE ~/mkswu]$ mkswu hawkbit_register.desc ❼
hawkbit_register.swu を作成しました。

[ATDE ~/mkswu]$ mkswu initial_setup.desc hawkbit_register.desc ❽
hawkbit_register.desc を組み込みました。
initial_setup.swu を作成しました。
```

- ❶ hawkbit_register.sh と .desc ファイルをカレントディレクトリにコピーします。
- ❷ hawkbit_register.sh を編集して、設定を記載します。
- ❸ hawkBit の設定の時に入力した「device」ユーザーのパスワードを入力します。この例のパスワードは使用しないでください。
- ❹ hawkBit サーバーの URL を入力します。
- ❺ TLS を使用の場合に、コンテナ作成の時の証明書を base64 で入力します。

- ⑥ hawkbit_register.desc の中身を確認します。hawkbit_register.sh を実行するだけです。
- ⑦ SWU を作成して、initial_setup がすでにインストール済みの Armadillo にインストールできます。
- ⑧ または、initial_setup.desc と合わせて hawkbit_register を含んだ initial_setup.swu を作成します。

9.10.6. mkswu の desc ファイル

.desc ファイルを編集すると、いくつかのコマンドが使えます。

例

```
[ATDE ~/mkswu]$ tree /usr/share/mkswu/examples/nginx_start
/usr/share/mkswu/examples/nginx_start
├── etc
│   └── atmark
│       └── containers
│           └── nginx.conf
└──
```

```
[ATDE ~/mkswu]$ cat /usr/share/mkswu/examples/usb_container_nginx.desc
swdesc_option version=1

swdesc_usb_container "nginx_alpine.tar" ①
swdesc_files --extra-os "nginx_start" ②
```

- ① nginx_alpine.tar ファイルに保存されたコンテナをインストールします。
- ② nginx_start ディレクトリの中身を転送します。

コマンドは書かれた順番でインストールされます。インストールするかどうかの判断はバージョンで行います:

```
swdesc_option component=<component>
swdesc_option version=<version>
か
swdesc_xxx --version <component> <version> [options]
```

・ <component>は以下のどれかにしてください (デフォルトでは .desc ファイルのファイル名を使います)

1. base_os: rootfs (Armadillo Base OS)を最初から書き込む時に使います。現在のファイルシステムは保存されていない。
 この場合、/etc/swupdate_preserve_files に載ってるファイルのみをコピーして新しい base OS を展開します。
 この component がないと現在の rootfs のすべてがコピーされます。
2. extra_os.<文字列>: rootfs の変更を行う時に使います。<文字列> には任意の文字列を指定します。
 rootfs を変更を行う時に使います。swdesc_* コマンドに --extra-os オプションを追加すると、component に自動的に extra_os. を足します。

3. <文字列> (コンテナの名前などの任意の文字列) : rootfs の変更がないときに使います。

この component を使うと rootfs の変更ができませんのでご注意ください。

- ・ アップデートを行う際にこのバージョンと現在のバージョンを比べてアップデートの判断を行います。

<component> がまだインストールされてなかった時や <version> が上がる時にインストールします。

デフォルトではダウングレードはできませんが、 --install-if=different オプションを追加することで <version> が変わる際にインストール可能になります。

アップデートの一部をインストールすることもありますので、複数の component で管理し、いくつかの古いバージョンに対応するアップデートも作成可能です。

以下のコマンドから使ってください

- ・ swdesc_tar と swdesc_files でファイルを転送します。

```
swdesc_tar [--dest <dest>] <tar_file>
swdesc_files [--dest <dest>] [--basedir <basedir>] ¥
<file> [<more files>]
```

swdesc_tar の場合、予め用意されてある tar アーカイブをこのままデバイスで展開します。

--dest <dest> で展開先を選ぶことができます。デフォルトは / (--extra-os を含め、バージョンの component は base_os か extra_os.* の場合) か /var/app/rollback/volumes/ (それ以外の component)。後者の場合は /var/app/volumes と /var/app/rollback/volumes 以外は書けないので必要な場合に --extra-os を使ってください。

swdesc_files の場合、mkswu がアーカイブを作ってくれますが同じ仕組みです。

--basedir <basedir> でアーカイブ内のパスをどこで切るかを決めます。

- ・ 例えば、swdesc_files --extra-os --basedir /dir /dir/subdir/file ではデバイスに /subdir/file を作成します。
- ・ デフォルトは <file> から設定されます。ディレクトリであればそのまま basedir として使います。それ以外であれば親ディレクトリを使います。
- ・ swdesc_command や swdesc_script でコマンドを実行する

```
swdesc_command <command> [<more commands>]
swdesc_script <script>
```

アップデート先の環境でコマンドやスクリプトファイルを走らせます。

バージョンの component は base_os と extra_os 以外の場合、 /var/app/volumes と /var/app/rollback/volumes 以外は変更できないのでご注意ください。

コマンドの実行が失敗した場合、アップデートも失敗します。

- ・ swdesc_exec でファイルを配ってコマンドでそのファイルを使う

```
swdesc_exec <file> <command>
```

swdesc_command と同じくコマンドを走らせますが、<file> を先に転送してコマンド内で"\$1"として使えます。

- swdesc_command_nochroot, swdesc_script_nochroot, swdesc_exec_nochroot で起動中のシステム上でコマンドを実行します。

このコマンドは nochroot なしのバージョンと同じ使い方で、現在起動中のシステムに変更や確認が必要な場合にのみ使用してください。



nochroot コマンドは確認を一切しないため、Armadillo が起動できない状態になる可能性もあります。充分にご注意ください。

例が必要な場合は /usr/share/mkswu/examples/firmware_update.desc を参考にしてください。

- swdesc_embed_container, swdesc_usb_container, swdesc_pull_container で予め作成したコンテナを転送します。

```
swdesc_embed_container <container_archive>
swdesc_usb_container <container_archive>
swdesc_pull_container <container_url>
```

例は「9.5.5. コンテナの配布」を参考にしてください。

- swdesc_boot で imx-boot を更新します。

```
swdesc_boot <boot image>
```

このコマンドだけにバージョンは自動的に設定されます。

コマンドの他には、設定変数もあります。以下の設定は /home/atmark/mkswu/mkswu.conf に設定できます。

- DESCRIPTION="<text>": イメージの説明、ログに残ります。
- PRIVKEY=<path>, PUBKEY=<path>: 署名鍵と証明書
- PRIVKEY_PASS=<val>: 鍵のパスワード（自動用）

openssl の Pass Phrase をそのまま使いますので、pass:password, env:var や file:pathname のどれかを使えます。pass や env の場合他のプロセスに見られる恐れがありますので file をおすすめします。

- ENCRYPT_KEYFILE=<path>: 暗号化の鍵

以下のオプションも `mkswu.conf` に設定できますが、`.desc` ファイルにも設定可能です。`swdesc_option` で指定することで、誤った使い方した場合 `mkswu` の段階でエラーを出力しますので、必要な場合は使用してください。

- `swdesc_option CONTAINER_CLEAR`: インストールされたあるコンテナと `/etc/atmark/containers/*.conf` をすべて削除します。

このオプションは簡単な初期化と考えてください。通常の運用では、不要になったイメージは自動的に削除されますのでこのオプションを設定する必要はありません。

以下のオプションは Armadillo 上の `/etc/atmark/baseos.conf` に、例えば `MKSWU_POST_ACTION=xxx` として設定することができます。

その場合に `swu` に設定されなければ `/etc` の設定で実行されますので、アットマークテクノが用意している Base OS のアップデートでも動作の変更は可能です。`swu` に特定のオプションが設定された場合は設定されたオプションが優先されますので、一時的な変更も可能です。

- `swdesc_option POST_ACTION=container`: コンテナのみのアップデート後に再起動を行いません。コンテナの中身だけをアップデートする場合、Armadillo-X2 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。

9.10.6.1. 例: `sshd` を有効にする

`/usr/share/mkswu/examples/enable_sshd.desc` を参考にします。

`desc` ファイルを編集する必要がありませんが自分の公開鍵を指定された場所に配置してください。

```
[ATDE ~/mkswu]$ cp -r /usr/share/mkswu/examples/enable_sshd* .
[ATDE ~/mkswu]$ cat enable_sshd.desc
swdesc_option component=extra_os.sshd version=1

# add your public key in enable_sshd/root/.ssh/authorized_keys
if [ -z "$SWDESC_TEST" ]; then
    grep -qE '^ssh-' enable_sshd/root/.ssh/authorized_keys ¥
    || error "Add your keys in enable_sshd/root/.ssh/authorized_keys"
fi
swdesc_files --dest /root enable_sshd/root ❶

swdesc_command "ssh-keygen -A" ¥ ❷
```

```

"rc-update add sshd" ❸
[ATDE ~/mkswu]$ cp ~/.ssh/id_rsa.pub ¥
enable_sshd/root/.ssh/authorized_keys ❹
[ATDE ~/mkswu]$ mkswu enable_sshd.desc ❺
Enter pass phrase for /home/atmark/mkswu/swupdate.key:
enable_sshd.swu を作成しました。

```

- ❶ 自分の公開鍵を転送します。デフォルトのオプションなので enable_sshd/root ディレクトリの中身をそのまま /root に転送されます。
- ❷ 再起動する度に新しいサーバーの鍵が変わらないように、アップデートの時に一回作成します。
- ❸ サービスを有効にします。
- ❹ 自分の公開鍵を指定された場所に配置します。
- ❺ イメージを作成します。パスワードは証明鍵のパスワードです。

9.10.6.2. 例: Armadillo Base OS アップデート

ここでは、「9.7. 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。

9.10.6.3. 例: swupdate_preserve_files で Linux カーネル以外の Armadillo-X2 向けのイメージをインストールする方法

Armadillo-X2 向けのアップデートイメージに Linux カーネルが含まれています。

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

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

- ❶ swupdate_preserve_files に /boot と /lib/modules を保存するように追加します。
- ❷ 変更した設定ファイルを保存します



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

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

- ❶ スクリプトの内容を確認する場合は /usr/share/mkswu/examples/update_preserve_files.sh を参照してください。



Armadillo Base OS のカーネルを再び使用したい場合は同じスクリプトの --del オプションで行を削除することができます。

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

9.10.7. swupdate_preserve_files について

extra_os のアップデートで rootfs にファイルを配置することができますが、次の OS アップデートの際に削除される可能性があります。

デフォルトでは、/etc/atmark と、swupdate、sshd やネットワークの設定を保存しますがそれ以外はコピーされてません。

そうでないファイルを更新するには /etc/swupdate_preserve_files に記載します。「9.10.6.3. 例: swupdate_preserve_files で Linux カーネル以外の Armadillo-X2 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`

9.10.8. SWU イメージの内容の確認

`mkswu --show [file.swu]` で SWU イメージの内容を確認することができます。

出力は desc ファイルに似ていますが、そのまま desc ファイルとして利用できませんので確認用としてお使いください。

```
[ATDE ~/mkswu]$ mkswu --show enable_sshd.swu
enable_sshd.swu

# built with mkswu 4.1

swdesc_files --dest /root enable_sshd/root
--version extra_os.sshd 1
(encrypted)

swdesc_command ssh-keygen -A && rc-update add sshd default
--version extra_os.sshd 1
```

9.10.9. SWUpdate と暗号化について

`mkswu --init` の時に暗号化を有効にする場合は AES でファイルを暗号化します。

現在使われてる SWUpdate の暗号化はコマンドやメタデータを含む sw-description ファイルは暗号化されてません。そのため、通信の暗号化 (HTTPS で送信するなど) を使うことを推奨します。

9.11. Armadillo Base OS の操作

Armadillo Base OS は Alpine Linux をベースとして作られています。

このセクションでは Armadillo Base OS の機能を紹介します。

9.11.1. アップデート

Armadillo Base OS は SWUpdate によってアップデートすることができます。

アップデートする際には、rootfs ファイルシステムにインストールされたファイルをすべて消して、アップデートの中身と /etc/swupdate_preserve_files に記載されているファイルで新しい rootfs を作ります。「9.10.7. swupdate_preserve_files について」を参照してください。

アップデートでファイルを削除してしまった場合に `abos-ctrl mount-old` で前のシステムを read-only でマウントして、削除されたファイルをコピーすることもできます。

9.11.2. overlaysfs と persist_file について

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

そのため、ファイルを変更した後 Armadillo の電源を切ると変更内容は保持されません。開発中などに rootfs の変更内容を保持するには、変更したファイルに対して `persist_file` コマンドを使用します。

開発以外の時は安全のため、ソフトウェアアップデートによる更新を実行してください。アップデート手順に関しては「9.10. Armadillo のソフトウェアをアップデートする」を参照してください。

rootfs の内容を変更しても、ソフトウェアアップデートを実施した際に変更した内容が保持されない可能性があります。ソフトウェアアップデート実施後も変更内容を保持する手順に関しては「9.10.7. swupdate_preserve_files について」を参照してください。

`persist_file` コマンドの概要を「図 9.201. persist_file のヘルプ」に示します。

```
[armadillo ~]# persist_file -h
Usage: /usr/bin/persist_file [options] file [more files...]

Mode selection:
  (none) single entry copy
  -d, --delete    delete file
  -l, --list      list content of overlay
  -a, --apk       apk mode: pass any argument after that to apk on rootfs
  -R, --revert    revert change: only delete from overlay, making it
                  look like the file was reverted back to original state

Copy options:
  -r, --recurse  recursive copy (note this also removes files!)
  -p, --preserve make the copy persist through baseos upgrade
                  by adding entries to /etc/swupdate_preserve_files
  -P, --preserve-post same, but copy after upgrade (POST)

Delete options:
  -r, --recurse  recursively delete files

Common options:
  -v, --verbose  verbose mode for all underlying commands

Note this directly manipulates overlaysfs lower directories
so might need a reboot to take effect
```

図 9.201 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 9.202 persist_file 保存・削除手順例

- ❶ 追加・変更したファイルを rootfs へコピーします。
- ❷ -r を指定すると、ひとつ前の rm -f コマンドで削除したファイルが rootfs から削除されますのでご注意ください。
- ❸ すでに rootfs に存在するファイルも一度削除してからコピーするため、このようなメッセージが表示されます。

2. ソフトウェアアップデート後も変更を維持する手順例

```

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

```

図 9.203 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 9.204 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 9.205 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。

9.11.3. ロールバック状態の確認

Armadillo Base OS の ルートファイルシステムが壊れて起動できなくなった場合に自動的に前のバージョンで再起動します。

自分で確認する必要がある場合に abos-ctrl status でロールバックされてるかどうかの確認ができます。

必要な場合（例えば、自分のアプリケーションがアップデート直後に問題があった場合）、abos-ctrl rollback で手動のロールバックも可能です。ロールバックにエラーがなければ、再起動してロールバックを完了します。

なお、/var/at-log/atlog に切り替えの際に必ずログを書きますので、調査の時に使ってください。

```
[armadillo ~]# cat /var/at-log/atlog
Mar 17 14:51:35 armadillo NOTICE swupdate: Installed update to /dev/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 9.206 /var/at-log/atlog の内容の例

9.11.4. ボタンやキーを扱う

自分のアプリケーションで直接入力の処理ができない場合に Base OS から簡単な処理ができます。

buttond サービスで指定されたイベントでコマンドを実行します。

/etc/atmark/buttond.conf に BUTTOND_ARGS を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:

- ・ -s <key> -a "command" : 早押しの設定。キーを 1 秒以内に離せば早押しと認識されてコマンドを実行します。その 1 秒のタイミングは -t <time_ms> でチューニング可能です。
- ・ -l <key> -s "command" : 長押しの設定。キーを 5 秒押しつづけたらその時にコマンドを実行します。その 5 秒のタイミングは -t <time_ms> でチューニング可能です。
- ・ 一つのキーを違うタイミングで何回か設定できます。その場合、長押しの設定あればその一番長いタイミングで実行されますが、他のアクションはキーを放す時に合っているコマンドを実行します。(例: -s 1 秒、-l 2 秒、-l 10 秒では、1 秒以内に離したら一番目、2 秒以上で 10 秒以内に離したら二番目、10 秒を越えたら三番目のコマンドを実行します)

以下にデフォルトを維持したままで SW1 の早押しと長押しにそれぞれの場合にコマンドを実行させます。

```
[armadillo ~]# vi /etc/atmark/buttond.conf ❶
BUTTOND_ARGS="$BUTTOND_ARGS -s prog1 -a 'date >> /tmp/shortpress'"
BUTTOND_ARGS="$BUTTOND_ARGS -l prog1 -t 5000 -a 'date >> /tmp/longpress'"
[armadillo ~]# persist_file /etc/atmark/buttond.conf ❷
[armadillo ~]# rc-service buttond restart ❸
buttond          | * Stopping button watching daemon ...           [ ok ]
buttond          | * Starting button watching daemon ...             [ ok ]
[armadillo ~]# cat /tmp/shortpress ❹
Tue Mar 22 17:16:42 JST 2022
Tue Mar 22 17:16:43 JST 2022
[armadillo ~]# cat /tmp/longpress
Tue Mar 22 17:16:48 JST 2022
```

図 9.207 buttond で SW1 を扱う

- ❶ カスタマイズ用のコンフィグファイルを編集します。早押しで shortpress, 長押し（5 秒）で longpress に日付を出力します。
- ❷ コンフィグファイルを保存します。
- ❸ buttond サービスを再起動させます。ここで早押し二回、長押し一回行います。
- ❹ 押された回数を確認します。

USB キーボードや他の入力デバイスにも対応できます。

1. デバイスを接続してから、buttond でデバイス名とキーコードを確認します。

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

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

- ❶ 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 -s LEFTCTRL -a 'podman_start button_pressed_container'"
[armadillo ~]# persist_file /etc/atmark/buttond.conf
[armadillo ~]# rc-service buttond restart
```

9.11.5. Armadillo Base OS 側の起動スクリプト

起動時に何かスクリプトを走らせるためにはコンテナとして実行することを推奨します。「9.5.1. コンテナの自動起動」を参照してください。

コンテナで実行不可能な場合に、「local」サービスを使うことができます:/etc/local.d ディレクトリに .start ファイルを置いておくと起動時に実行されて、.stop ファイルは終了時に実行されます。

```
[armadillo ~]# vi /etc/local.d/date_test.start ❶
#!/bin/sh

date > /tmp/boottest
[armadillo ~]# chmod +x /etc/local.d/date_test.start ❷
[armadillo ~]# persist_file /etc/local.d/date_test.start ❸
[armadillo ~]# reboot
: (省略)
[armadillo ~]# cat /tmp/boottest ❹
Tue Mar 22 16:36:12 JST 2022
```

図 9.208 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。

9.11.6. u-boot の環境変数の設定

u-boot の環境変数を変更するには /boot/uboot_env.d/ ディレクトリに環境変数が書かれた設定ファイルを配置します。

ファイルの構文は fw_setenv が扱うことができるもので、以下のとおりです：

- ・ # で始まる行はコメントと扱われる為、無視されます。また、環境変数への代入を示す = がない場合も無視されます。
- ・ [変数]=[値] で変数を設定します。スペースや引用符を含め他の文字は有効ですので、変数の名前と値に不要な文字を入れないように注意してください。
- ・ [変数]= で変数を消します。値がない場合に変数が消去されます。

このファイルによるアップデート内容は swupdate でアップデートする際に適用されます。

実行中のシステムに影響がありませんので、設定ファイルを swupdate で転送しない場合はファイル永続化後に fw_setenv -s /boot/uboot_env.d/[ファイル名] で変数を書き込んでください。

swupdate でファイルを転送した場合には、変数はすぐに利用されます。

```
[armadillo ~]# vi /boot/uboot_env.d/no_prompt 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 9.209 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。



「9.7.1. 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 9.12 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
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、「12.1.4. ロールバック (リカバリー)」を参照してください。	1
bootlimit	"bootcount"のロールバックを行うしきい値を指定します。	3
bootdelay	保守モードに遷移するためのキー入力待つ時間を指定します(単位: 秒)。次の値は特別な意味を持ちます。 <ul style="list-style-type: none"> ・ -1: キー入力の有無に関らず保守モードに遷移します。 ・ -2: キー入力の有無に関らず保守モードに遷移しません。 	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、「9.12.4. DT overlay によるカスタマイズ」を参照してください。	boot/overlays.txt
mmcautodetect	mmc デバイスの自動検出機能の有効/無効を指定します。yes を指定した場合のみ、u-boot が起動された mmc デバイスが自動的に mmcdev として利用されます。	yes
mmcdev	"image"や"fdt_file"で指定されたファイルが配置してある mmc デバイスのインデックスを指定します。インデックスと mmc デバイスの対応は次の通りです。 <ul style="list-style-type: none"> ・ 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
mmccroot	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

環境変数	説明	デフォルト値
optargs	Linux カーネル起動時パラメータを指定します。"quiet"を削除すると、コンソールに起動ログが出力されるようになりますが、起動時間が長くなります。nokaslr を削除すると、KASLR(Kernel Address Space Layout Randomization)が有効となり、Linux カーネルの仮想アドレス空間がランダム化されます。	quiet 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

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

Armadillo Base OS では chronyd を使っています。

デフォルトの設定（使用するサーバーなど）は /etc/chrony/conf.d/ にあり、変更用に /etc/atmark/chrony.conf.d/ のファイルも読み込みます。/etc/atmark/chrony.conf.d ディレクトリに /etc/chrony/conf.d/ と同じファイル名の設定ファイルを置いておくことで、デフォルトのファイルを読まないようになります。

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

```
[armadillo ~]# vi /etc/atmark/chrony.conf.d/servers.conf ❶
pool my.ntp.server iburst
[armadillo ~]# persist_file /etc/atmark/chrony.conf.d/servers.conf ❷
[armadillo ~]# rc-service chronyd restart ❸
chronyd          | * Stopping chronyd ... [ ok ]
chronyd          | * Starting chronyd ... [ ok ]
armadillo:~# chronyc sources ❹
MS Name/IP address      Stratum Poll Reach LastRx Last sample
=====
^? my.ntp.server        1      6      3      2    +88ms[ +88ms] +/- 173ms
```

図 9.210 chronyd のコンフィグの変更例

- ❶ コンフィグファイルを作ります。
- ❷ ファイルを保存します
- ❸ chronyd サービスを再起動します。
- ❹ chronyc で新しいサーバーが使用されていることを確認します。

9.12. 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)です。



Armadillo-X2 は Armadillo-IoT ゲートウェイ G4 と同じ DTB で動作します。そのため、「9.12. Device Tree をカスタマイズする」や「9.12.4. DT overlay によるカスタマイズ」でも armadillo_iotg_g4- から始まる dtb を利用します。

この動作は、RAM 上にロードされた DTB をブートローダーが修正することで実現されています。

9.12.1. at-dtweb のインストール

ATDE9 に at-dtweb パッケージをインストールします。

```
[ATDE ~]$ sudo apt update
[ATDE ~]$ sudo apt install at-dtweb
```

インストール済みの場合は、以下のコマンドを実行し最新版への更新を行ってください。

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

9.12.2. at-dtweb の起動

1. at-dtweb の起動開始

at-dtweb の起動を開始するには、デスクトップ左上のアプリケーションの「システムツール」から「at-dtweb」を選択してください。

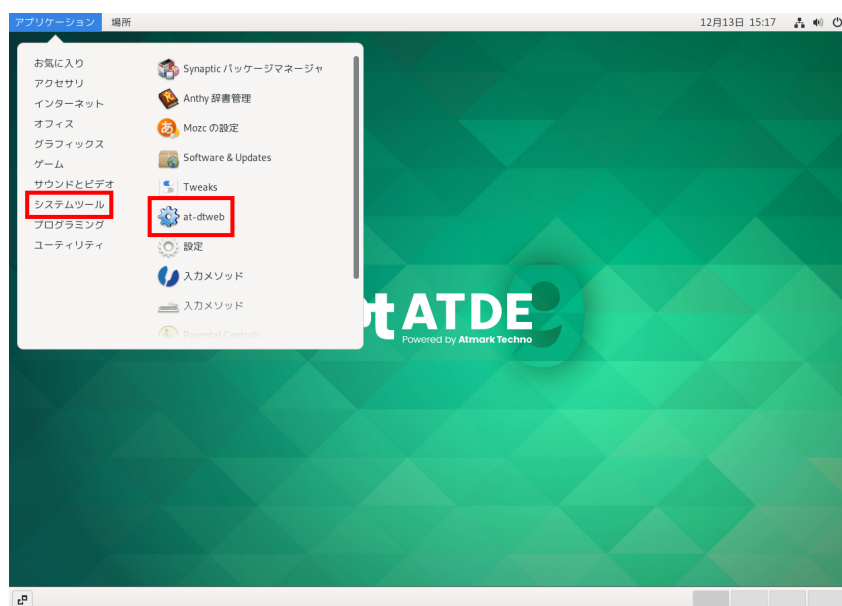


図 9.211 at-dtweb の起動開始

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-X2 を選択して、「OK」をクリックします。

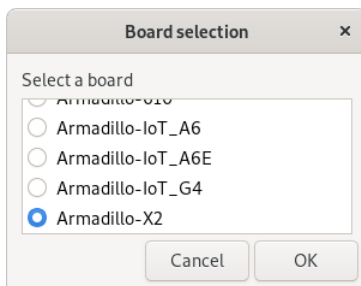


図 9.212 ボード選択画面

2. Linux カーネルディレクトリの選択

Linux カーネルディレクトリを選択します。コンフィギュレーション済みの Linux カーネルディレクトリを選択して、「OK」をクリックします。

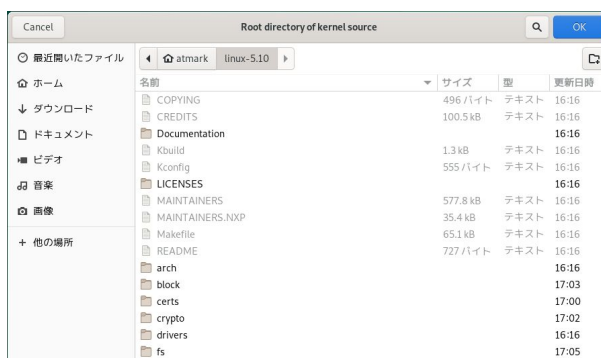


図 9.213 Linux カーネルディレクトリ選択画面

3. at-dtweb の起動完了

at-dtweb が起動し、次のように画面が表示されます。

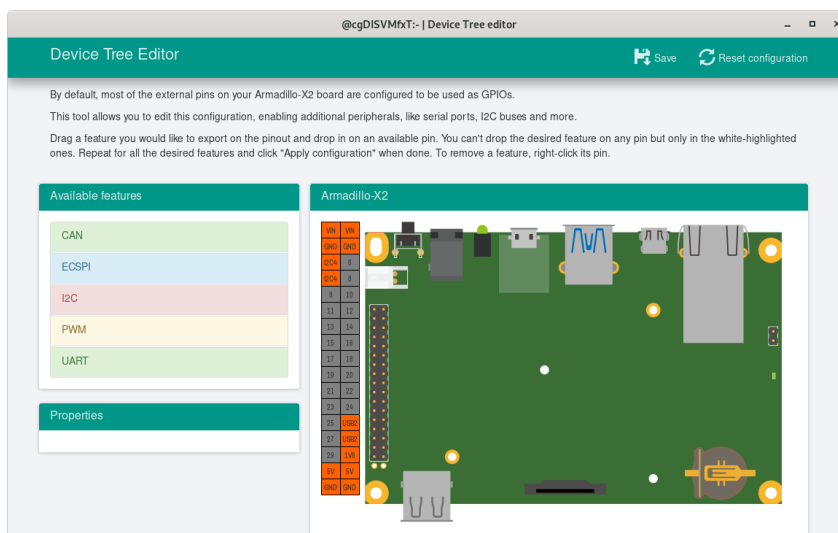


図 9.214 at-dtweb 起動画面



Linux カーネルは、事前にコンフィギュレーションされている必要があります。コンフィギュレーションの手順については「9.7. Armadillo のソフトウェアをビルドする」を参照してください。

9.12.3. Device Tree をカスタマイズ

9.12.3.1. 機能の選択

機能の選択は、ドラッグ&ドロップで行います。画面左上の「Available features」から有効にしたい機能をドラッグし、画面右側の「Armadillo-X2」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。

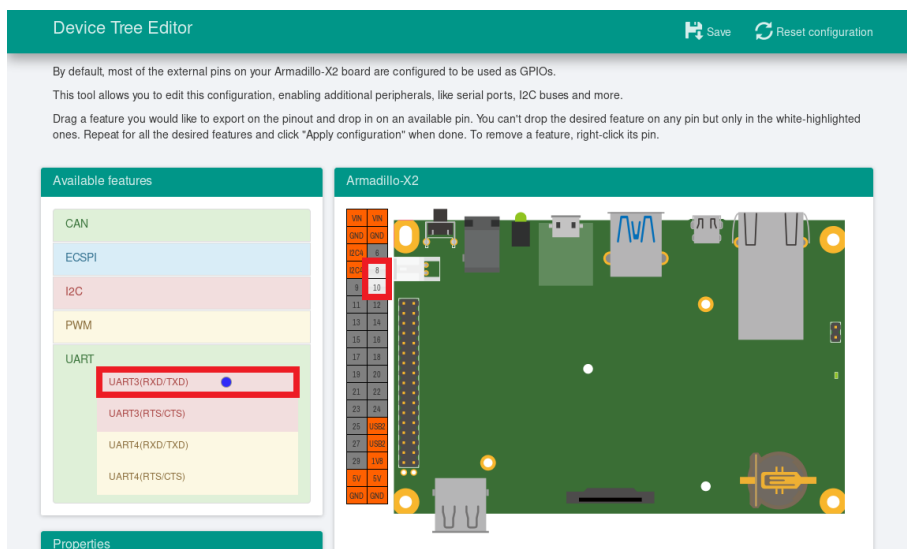


図 9.215 UART3(RXD/TXD) のドラッグ

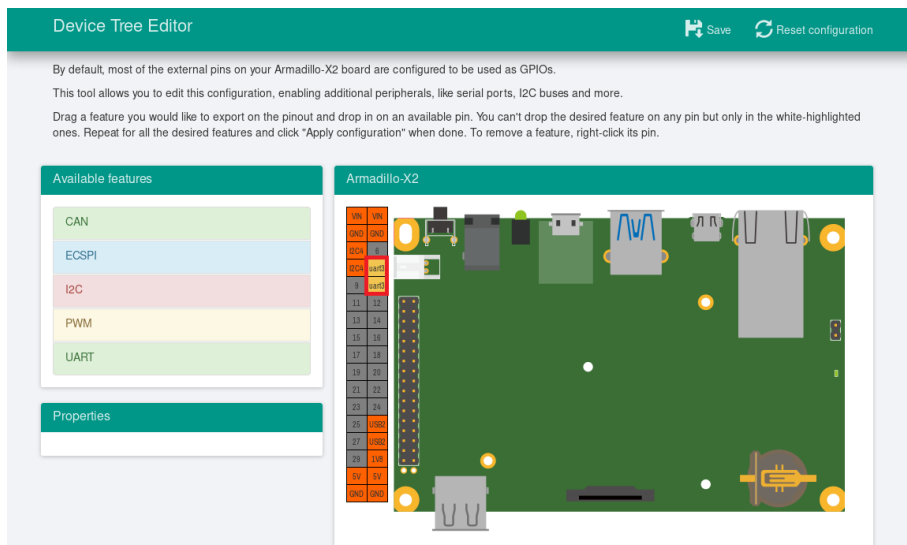


図 9.216 CON11 8/10 ピンへのドロップ

9.12.3.2. 信号名の確認

画面右側の「Armadillo-X2」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。

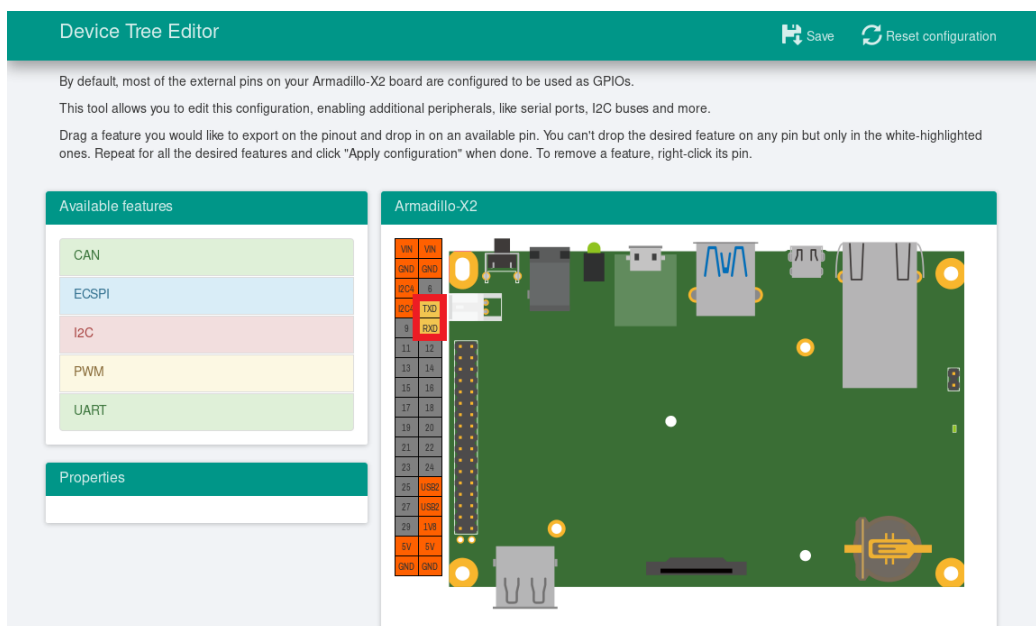


図 9.217 信号名の確認



再度ピンを左クリックすると機能名の表示に戻ります。

9.12.3.3. プロパティの設定

いくつかの機能にプロパティを設定することができます。画面右側の「Armadillo-X2」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。

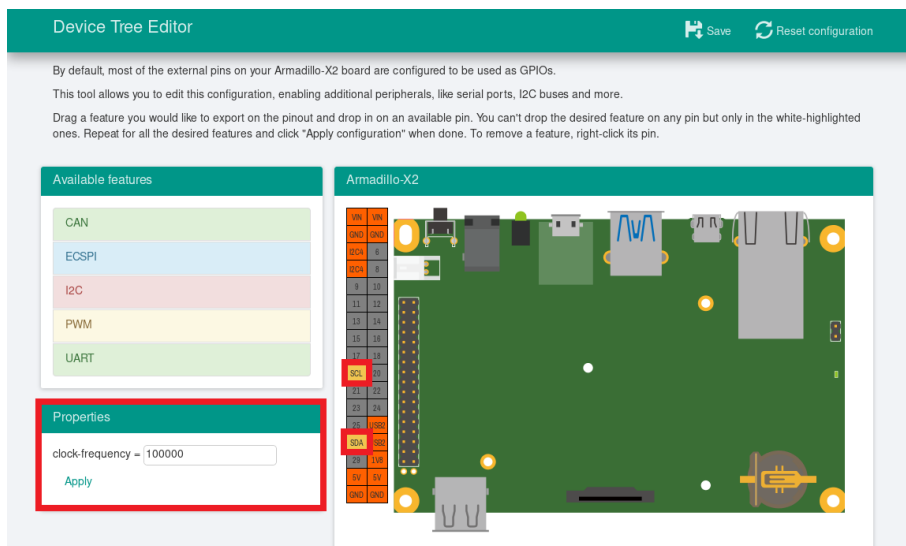


図 9.218 プロパティの設定

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。

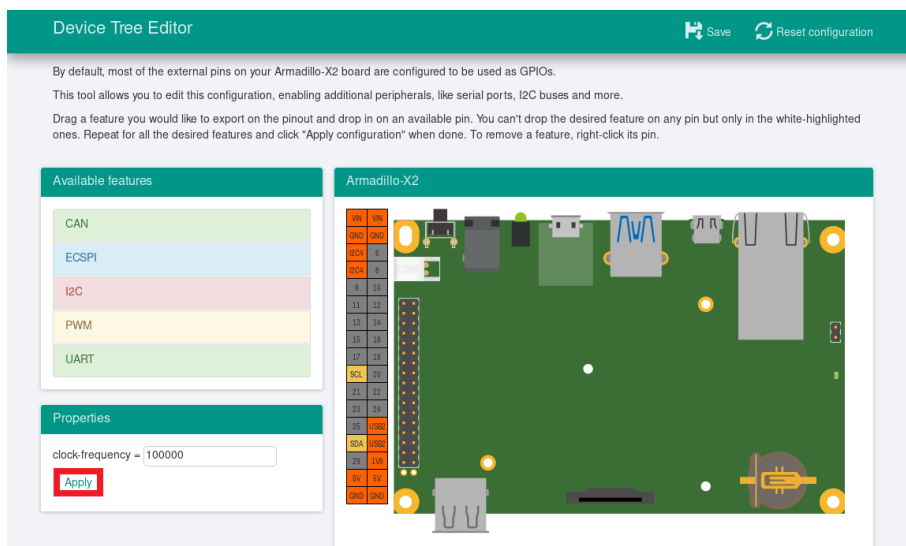


図 9.219 プロパティの保存

9.12.3.4. 機能の削除

全ての機能を削除する場合は、画面右上の「Reset configuration」をクリックします。機能ごとに削除する場合は、画面右側の「Armadillo-X2」のピンを右クリックして「Remove」をクリックします。

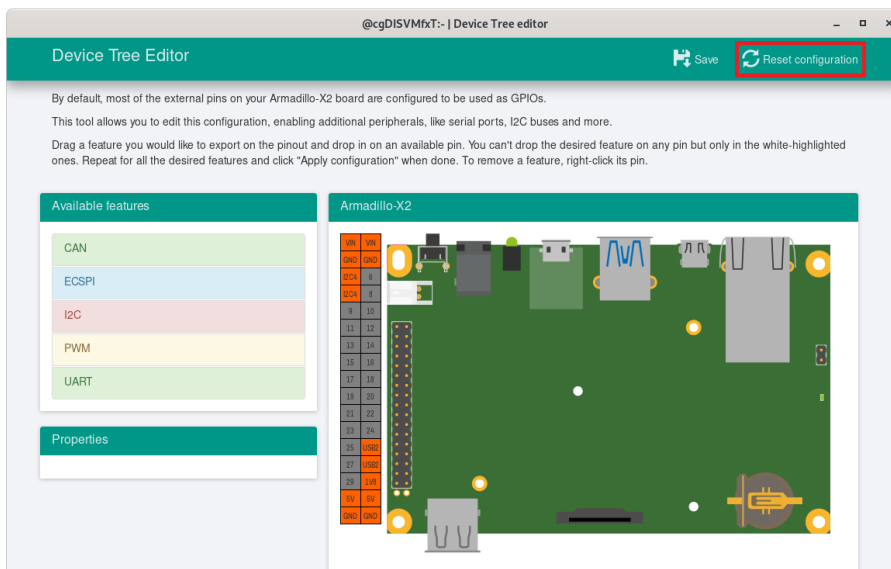


図 9.220 全ての機能の削除

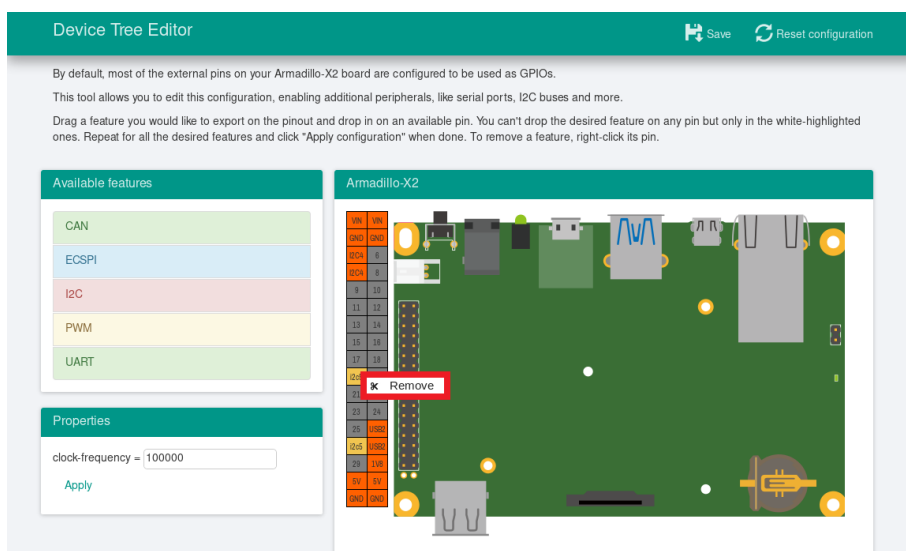


図 9.221 I2C5(SCL/SDA) の削除

9.12.3.5. Device Tree のファイルの生成

Device Tree のファイルを生成するには、画面右上の「Save」をクリックします。

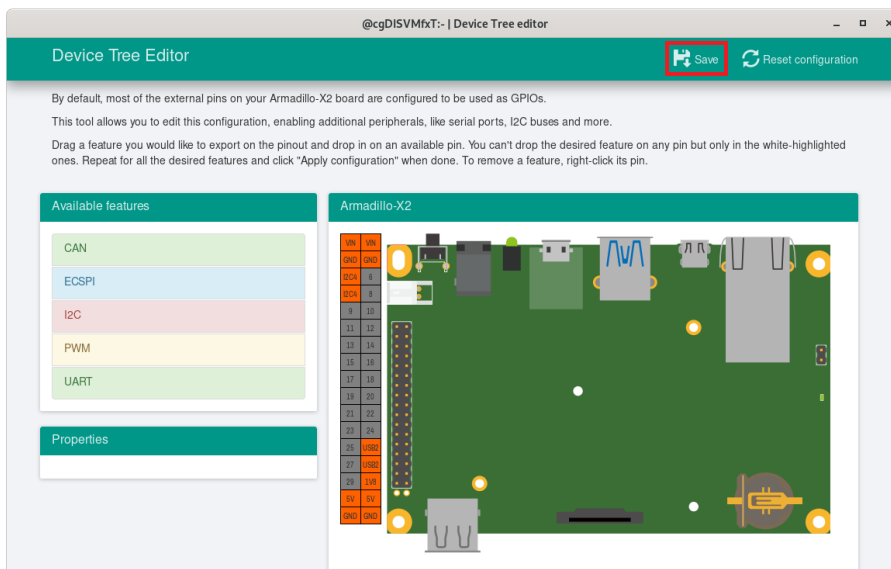


図 9.222 DTS/DTB の生成

以下の画面ようなメッセージが表示されると、dtbo ファイルおよび desc ファイルの生成は完了です。

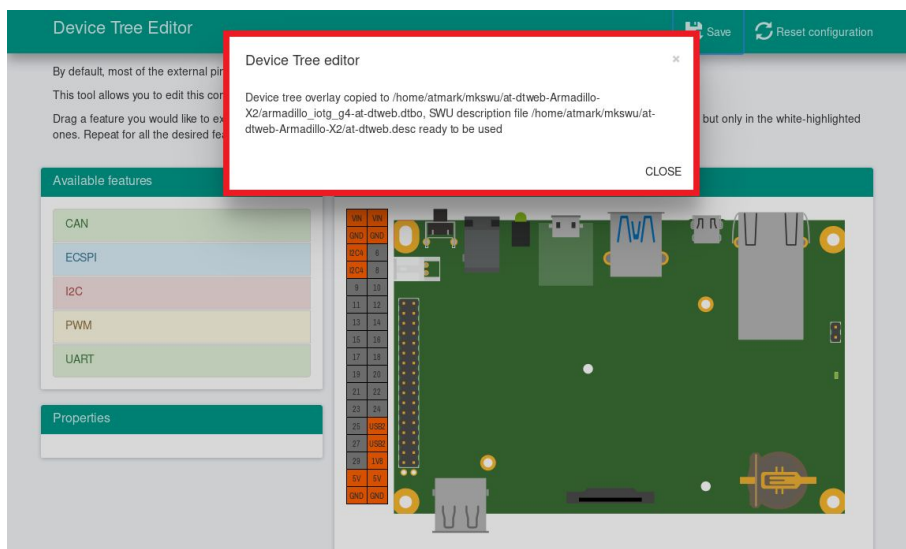


図 9.223 dtbo/desc の生成完了

ビルドが完了するとホームディレクトリ下の `mkswu/at-dtweb-Armadillo-X2` 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-X2 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-X2
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-X2
[ATDE ~]$ mkswu at-dtweb.desc ①
Enter pass phrase for /home/atmark/mkswu/swupdate.key:
at-dtweb.swu を作成しました。
```

① SWU イメージを生成します。



Armadillo-X2 は Armadillo-IoT ゲートウェイ G4 と同じ DTB で動作します。そのため、「9.12. Device Tree をカスタマイズする」や「9.12.4. DT overlay によるカスタマイズ」でも `armadillo_iotg_g4-` から始まる dtb を利用します。

この動作は、RAM 上にロードされた DTB をブートローダーが修正することで実現されています。

SWU イメージを使ったアップデートの詳細は「9.10. Armadillo のソフトウェアをアップデートする」を参照してください。

9.12.4. DT overlay によるカスタマイズ

Device Tree は「DT overlay」(dtbo) を使用することでも変更できます。

DT overlay を使用することで、通常の dts の更新が自動的に入りつづける状態で dts の変更でしかできない設定を行うことができます。

`/boot/overlays.txt` に `fdt_overlays` を dtbo 名で設定することで、u-boot が起動時にその DT overlay を通常の dtb と結合して起動します。

複数の DT overlay を使う場合は以下の例のようにスペースで別けたファイル名を記載することができます。

```
[armadillo ~]# vi /boot/overlays.txt ①
fdt_overlays=armadillo_iotg_g4-nousb.dtbo armadillo_iotg_g4-sw1-wakeup.dtbo

[armadillo ~]# persist_file -vp /boot/overlays.txt ②
'/boot/overlays.txt' -> '/mnt/boot/overlays.txt'
Added "/boot/overlays.txt" to /etc/swupdate_preserve_files

[armadillo ~]# reboot ③
: (省略)
Applying fdt overlay: armadillo_iotg_g4-nousb.dtbo ④
Applying fdt overlay: armadillo_iotg_g4-sw1-wakeup.dtbo
: (省略)

[armadillo ~]# cat /sys/firmware/devicetree/base/regulator-usb1-vbus/status; echo
broken ⑤

[armadillo ~]# cat /sys/devices/platform/gpio-keys/power/wakeup ⑥
enabled
```

図 9.224 /boot/overlays.txt の変更例

- ① `/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「9.12.4. DT overlay によるカスタマイズ」を参照してください。

- ② /boot/overlays.txt を保存し、アップデートの場合でも保存します。
- ③ overlay の実行のために再起動します。
- ④ 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。
- ⑤ Linux から「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。
- ⑥ sw1-wakeup も有効になっていることを確認できます。

9.12.4.1. 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。

9.12.4.2. 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. 「9.7.2. 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] ①
[PC ~/linux-[VERSION]]$ vim ¥
    arch/arm64/boot/dts/freescale/armadillo_iotg_g4-customize.dts ②
/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 9.225 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 がロードされていることを確認します。

9.13. 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-X2 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「9.13.3. 実装仕様に関する技術情報」を参照してください。

9.13.1. 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 9.13 データリテンションの挙動

<code>/etc/conf.d/micron_emmc_reten</code>	initiating condition
<code>use_system_time=yes</code>	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
<code>use_system_time=no (default)</code>	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。

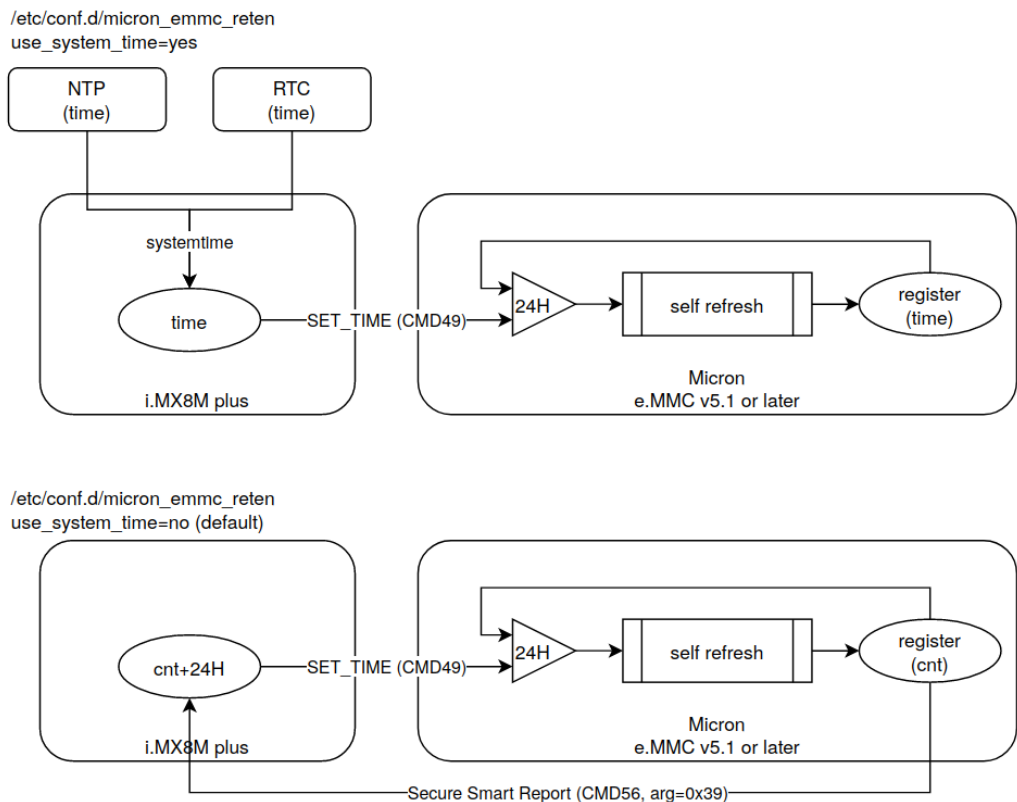


図 9.226 データリテンション開始トリガーの方式

`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。

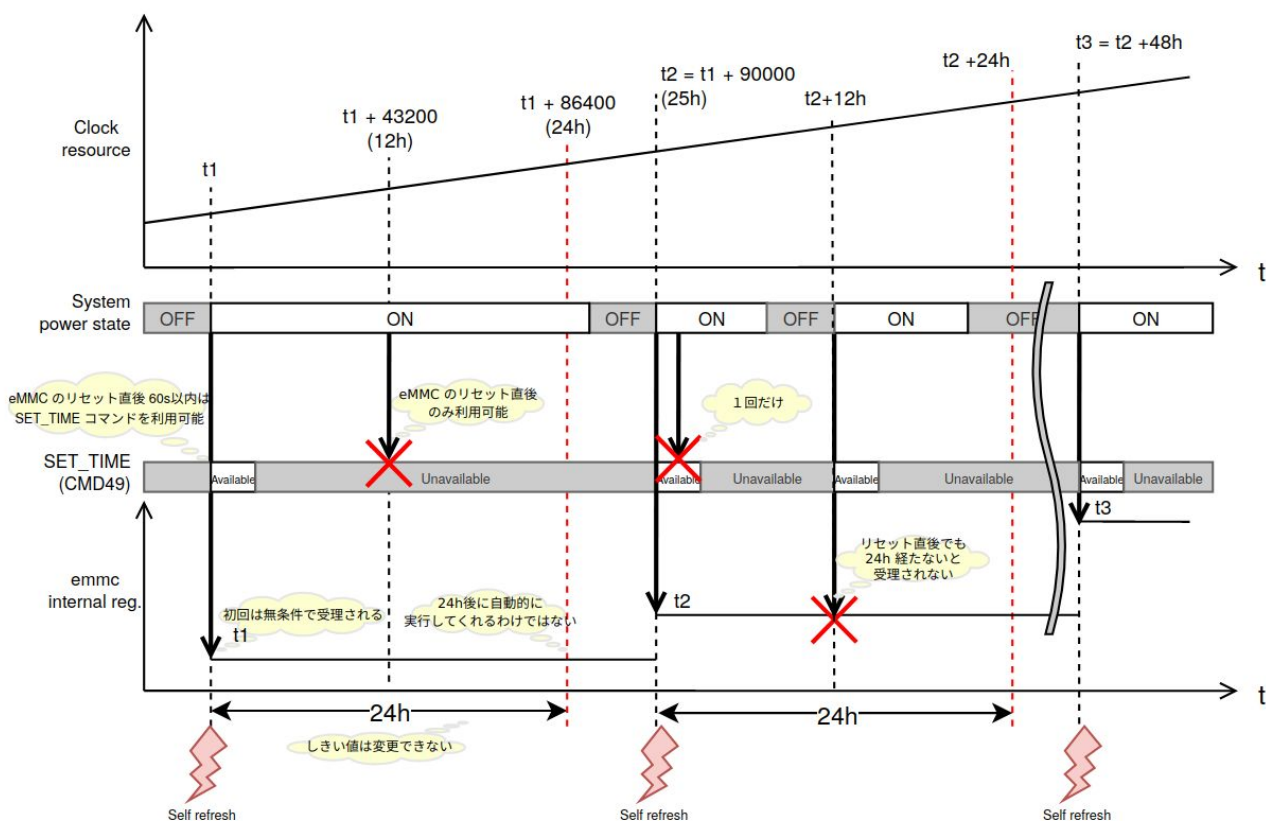


図 9.227 データリテンションの開始トリガーの動作例


9.13.2. 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]:                 0xffffffff8148931 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

9.13.3. 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、セルフ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 9.14 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。

9.14. 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する必要があります。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「9.4.3. アットマークテクノが提供するイメージを使う」を参照してください。

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。

9.14.1. 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 9.228 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

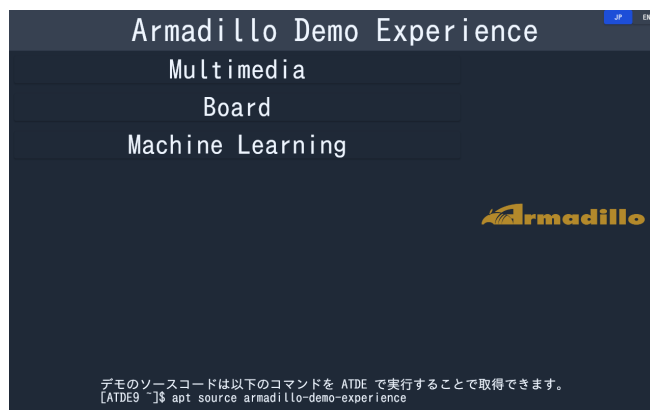
9.14.2. デモアプリケーションランチャを起動する

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 9.229 デモアプリケーションランチャの起動

以下のようなアプリ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 9.15 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
```

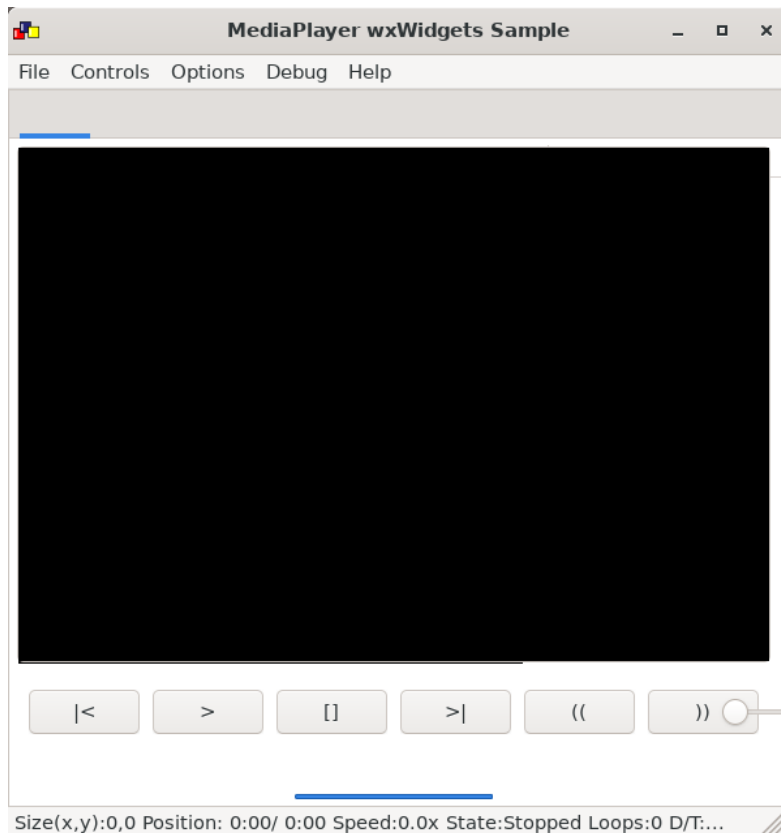
図 9.230 パッケージのバージョンを確認する

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。

9.14.3. 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。



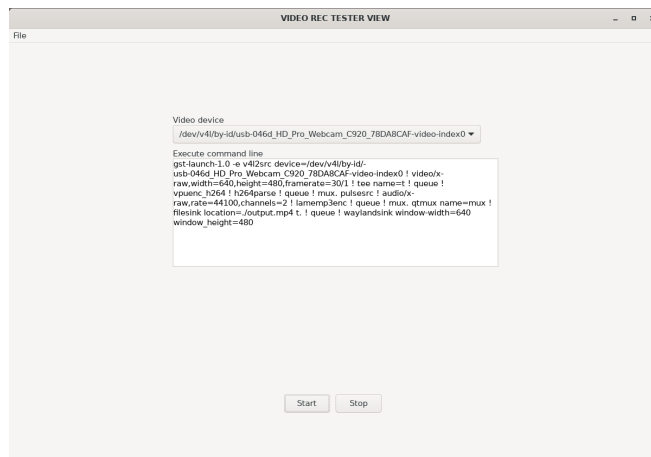
音声も出力したい場合は、pulseaudio をインストールして起動する必要があります。

```
[container /]# apt install pulseaudio
[container /]# pulseaudio --start --exit-idle-time=-1
```

図 9.231 pulseaudio のインストールと起動

9.14.4. video recoder

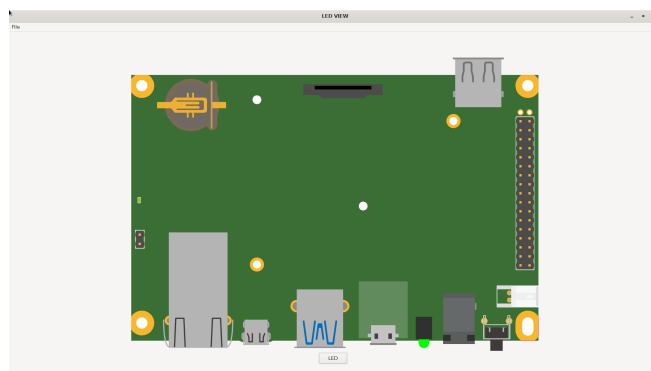
video recoder は gstreamer を使用してカメラからの映像を録画することができます。そのため、このアプリケーションを使用するためには、Armadillo 本体にカメラを接続する必要があります。カメラが接続されていると Video device の項目でカメラを選択できるようになります。カメラを選択し、Start ボタンを押すと別ウィンドウが表示され録画が開始されます。アプリケーション上のテキストボックスには、Start ボタンを押したときに起動する gstreamer のコマンドを表示しています。テキストボックスの内容はキーボードで編集可能です。このアプリケーションは、GUI 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、「9.14.3. mediaplayer」を参照して pulseaudio を起動してください。

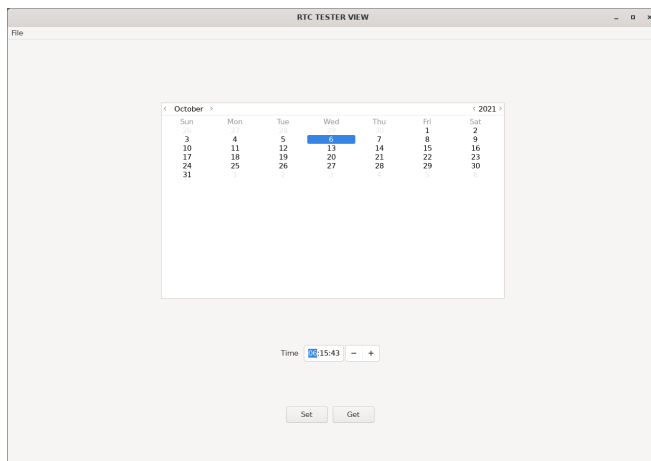
9.14.5. led switch tester

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。



9.14.6. 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。



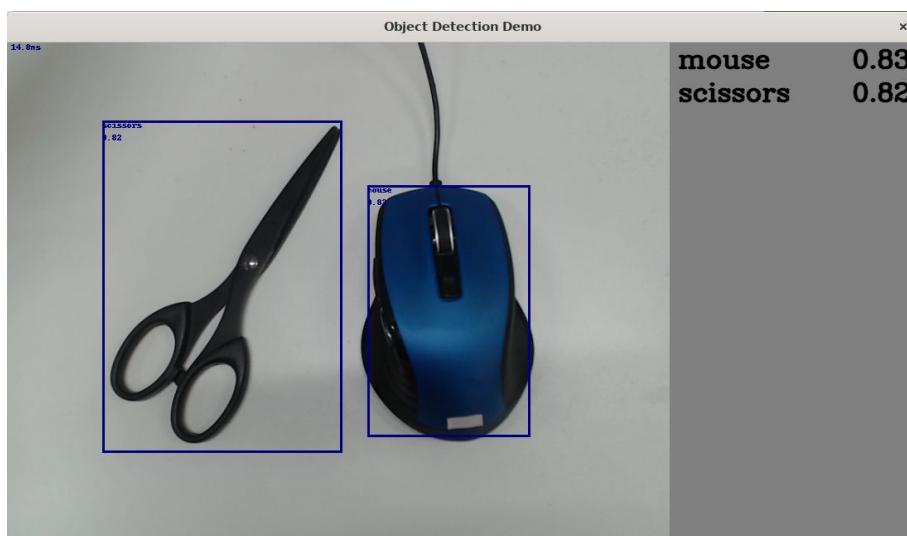
9.14.7. object detection demo

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 9.232 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 application that used NPU and TensorFlow
Lite on the Armadillo-X2 board."
}]}]}
    
```



図 9.233 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。

9.14.8. 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":[{"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 application that used NPU and TensorFlow
Lite on the Armadillo-X2 board."
}]}]}
:
: (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 application that uses NPU on the Armadillo-
X2 board."
  }
}]

```

図 9.234 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。

9.14.9. 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 application that uses NPU on the
Armadillo-X2 board."
    }]
}]
```

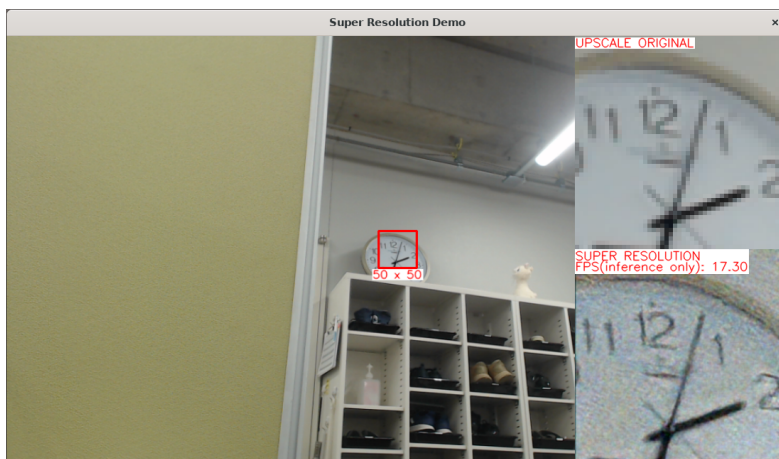
図 9.235 ビデオデバイスの変更

❶ --camera_id の値を環境に合わせて変更します。

9.14.10. 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/
```

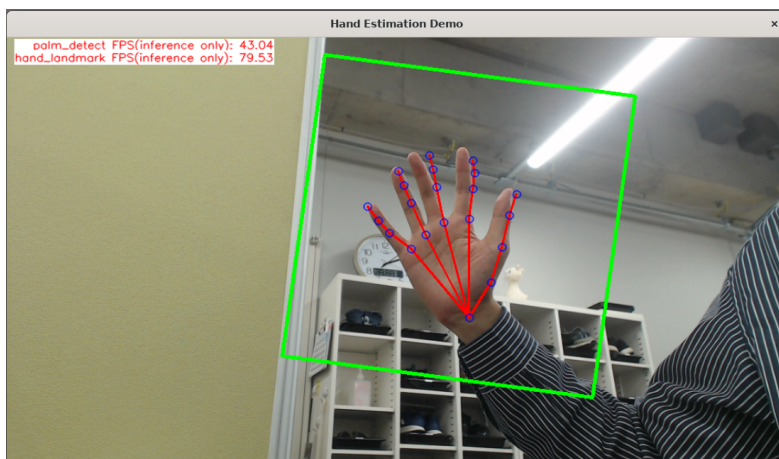
```
super_resolution.py --model /usr/share/armadillo-demo-experience/AI-demo/super_resolution/
super_resolution.tflite --camera_id 2", ❶
    "source": "",
    "screenshot": "super_resolution_demo.png",
    "compatible": "armadillo-x2",
    "description": "This is a super resolution application that uses NPU on the Armadillo-
X2 board."
    }]
}]
```

図 9.236 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。

9.14.11. 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."
    }]
}]
```

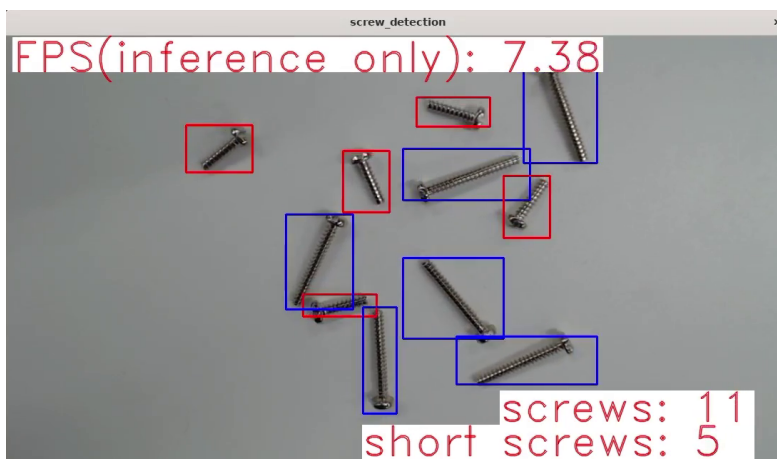
図 9.237 ビデオデバイスの変更

❶ --camera_id の値を環境に合わせて変更します。

9.14.12. 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 9.238 各種コンフィグの変更

- ① 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 9.16. 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 9.16 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。

9.15. 動作中の Armadillo の温度を測定する

この章では、Armadillo Base OS 搭載製品を組み込んだユーザー製品の熱設計時に役立つ温度プロファイラツールである「atmark-thermal-profiler」について紹介します。

9.15.1. 温度測定的重要性

Armadillo は製品ごとに動作温度範囲が設定されていますが、それらは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認する必要があります。

Armadillo Base OS 搭載製品では、動作中の Armadillo の各種温度等を取得し CSV 形式で出力する atmark-thermal-profiler を利用することができますので、温度測定に役立てることができます。

9.15.2. atmark-thermal-profiler をインストールする

atmark-thermal-profiler は apk パッケージで公開されていますので、apk add コマンドでインストールすることが可能です。

```

[armadillo ~]# apk upgrade
[armadillo ~]# apk add atmark-thermal-profiler

```

図 9.239 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。

9.15.3. atmark-thermal-profiler を実行・停止する

「[図 9.240. 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 9.240 atmark-thermal-profiler を実行する

「[図 9.241. 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 9.241 atmark-thermal-profiler を停止する

9.15.4. atmark-thermal-profiler が出力するログファイルを確認する

atmark-thermal-profiler は、インストール直後から自動的に温度や CPU 負荷率、Load Average などの情報を 30 秒に 1 度の周期で集め、`/var/log/thermal_profile.csv` に追記していきます。

```
[armadillo ~]# head /var/log/thermal_profile.csv
DATE, ONESHOT, CPU_TMEP, SOC_TEMP, LOAD_AVE, CPU_1, CPU_2, CPU_3, CPU_4, CPU_5, USE_1, USE_2, USE_3, USE_4, USE_5
2022-11-30T11:11:05+09:00, 0, 54, 57, 0.24, /usr/sbin/rngd -b -p /run/rngd.pid -q -0 jitter:buffer_size:
4133 -0 jitter:refill_thresh:4133 -0 jitter:thread_count:1, /usr/sbin/chronyd -f /etc/chrony/
chrony.conf, [kworker/1:3H-kb], podman network inspect podman, /usr/sbin/NetworkManager -n, 22, 2, 2, 0, 0,
: (省略)
```

↵

↵

↵

図 9.242 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 9.17. thermal_profile.csv の各列の説明](#)」に記載します。

表 9.17 thermal_profile.csv の各列の説明

ヘッダ	説明
DATE	その行のデータ取得日時です。"年-月-日 T時:分:秒+タイムゾーン" の形式で出力されます。
ONESHOT	この列が 1 の行のデータは、サーマルシャットダウンを含むシャットダウンが実行された時に取得されたことを示します。
CPU_TEMP	計測時点の CPU 温度を示します。単位は°Cです。
SOC_TEMP	計測時点の SoC 温度を示します。単位は°Cです。製品によっては非対応で、その場合は空白になります。
LOAD_AVE	計測時点から直近 1 分間の Load Average です。
CPU_1	計測時点の CPU 使用率 1 位のプロセスです。
CPU_2	計測時点の CPU 使用率 2 位のプロセスです。
CPU_3	計測時点の CPU 使用率 3 位のプロセスです。
CPU_4	計測時点の CPU 使用率 4 位のプロセスです。
CPU_5	計測時点の CPU 使用率 5 位のプロセスです。
USE_1	計測時点の CPU 使用率 1 位のプロセスの CPU 使用率です。
USE_2	計測時点の CPU 使用率 2 位のプロセスの CPU 使用率です。
USE_3	計測時点の CPU 使用率 3 位のプロセスの CPU 使用率です。
USE_4	計測時点の CPU 使用率 4 位のプロセスの CPU 使用率です。
USE_5	計測時点の CPU 使用率 5 位のプロセスの CPU 使用率です。

9.15.5. 温度測定結果の分析

atmark-thermal-profiler を使用して得られたログファイルの内容を分析してみます。

9.15.5.1. サーマルシャットダウン温度の確認

予め、使用している Armadillo が何°Cでサーマルシャットダウンするか確認しておきます。ここでは、Armadillo Base OS を搭載している Armadillo-IoT ゲートウェイ G4 を例とします。他の製品では得られる結果が異なる場合があることに注意してください。

```
[armadillo ~]# cat /sys/class/thermal/thermal_zone0/trip_point_1_temp
105000 ①
[armadillo ~]# cat /sys/class/thermal/thermal_zone1/trip_point_1_temp
105000 ②
```

図 9.243 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)

- ① CPU のサーマルシャットダウン温度です。ミリ°Cで表記されているので、105°Cでサーマルシャットダウンすることがわかります。
- ② SoC のサーマルシャットダウン温度です。ミリ°Cで表記されているので、105°Cでサーマルシャットダウンすることがわかります。

9.15.5.2. 温度測定結果のグラフ化

atmark-thermal-profiler が出力するログ(thermal_profile.csv)は CSV ファイルなので、各種表計算ソフトでインポートしてグラフ化することが可能です。これにより Armadillo 動作中の温度の変化が可視化され、得られる情報が見やすくなります。

「図 9.244. 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。

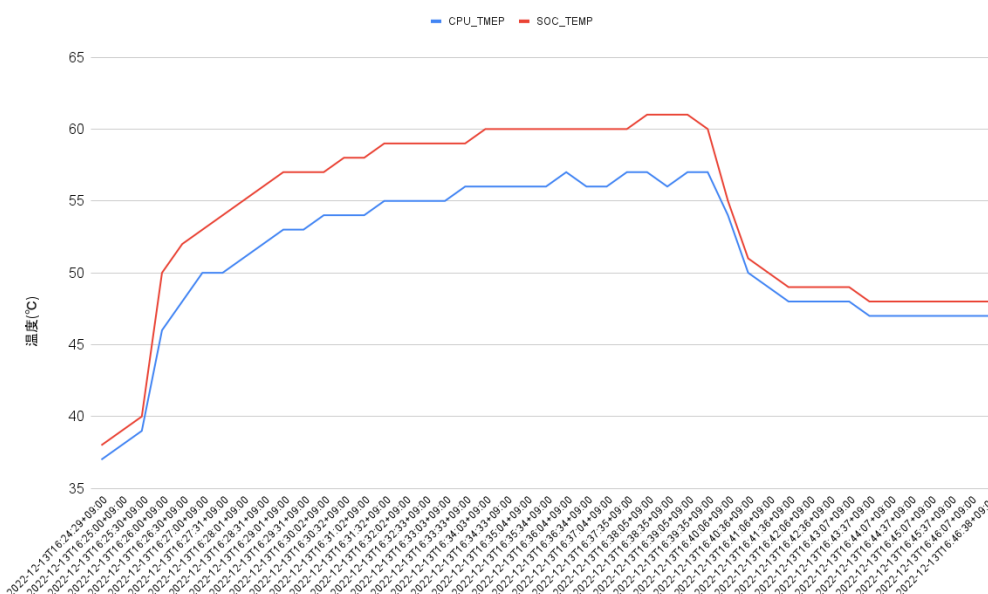


図 9.244 Armadillo-IoT ゲートウェイ G4 で取得した温度のグラフ

グラフの縦軸は温度(°C)で、横軸は時間です。青い線は CPU の温度、赤い線は SoC の温度を表しています。このグラフと、「9.15.5.1. サーマルシャットダウン温度の確認」で得たサーマルシャットダウン温度を見比べると、CPU に負荷をかけた際であっても SoC の温度は 60°C 前後ほどまでしか上がらず、この条件で動く Armadillo が温度的にどれほど余裕を持っているかをひと目で確認できます。

9.15.5.3. CPU 使用率の確認

atmark-thermal-profiler は、時間毎の温度だけでなく CPU 使用率と CPU 使用率の高いプロセスについても取得して記録します。CPU 使用率については thermal_profile.csv の CPU_1~CPU_5 列と、USE_1~USE_5 列を参照してください。各列について詳しくは「表 9.17. 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。

9.16. 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。

9.16.1. 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

「9.7.2. 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。

- ③ 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。
- ④ 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 ①  
[armadillo ~]# persist_file /etc/runlevels/default/kdump-tools ②
```

- ① kdump-tools サービスの自動起動を有効にします。
- ② 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。「9.11.6. 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 ①  
optargs=quiet nokaslr crashkernel=128M ②  
[armadillo ~]# persist_file -v /boot/uboot_env.d/kdump ③  
'/boot/uboot_env.d/kdump' -> '/mnt/boot/uboot_env.d/kdump'  
[armadillo ~]# fw_setenv -s /boot/uboot_env.d/kdump ④  
Environment OK, copy 0  
[armadillo ~]# fw_printenv | grep optargs= ⑤  
optargs=quiet nokaslr crashkernel=128M  
[armadillo ~]# reboot ⑥
```

- ① コンフィグファイルを生成します。
- ② 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。
- ③ ファイルを永続化します。
- ④ 変数を書き込みます。
- ⑤ 書き込んだ変数を確認します。
- ⑥ 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。

9.16.2. 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 ①  
[armadillo ~]# echo c > /proc/sysrq-trigger ②
```

```
[ 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 0x00000000003 [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"
```

9.16.3. 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、「9.16.2. 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。

「9.16.1. 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、「9.4.3. アットマークテクノが提供するイメージを使う」を参照して 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ますようになります。

```
[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
```

10. 動作ログ

10.1. 動作ログについて

Armadillo-X2 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。

10.2. 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。

10.3. 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 10.1 動作ログのフォーマット

10.4. 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。

11. 製品機能

本章では、Armadillo-X2 で利用できる各種機能の仕様について説明します。

11.1. SD ホスト

Armadillo-X2 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-X2 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。

- デバイスファイル ・ /dev/mmcblk1
- 関連するソースコード ・ drivers/mmc/host/sdhci-esdhc-imx.c
- ・ drivers/mmc/host/sdhci-of-esdhc.c
- Device Tree ドキュメント ・ Documentation/devicetree/bindings/mmc/fsl-imx-esdhc.yaml
- ・ Documentation/devicetree/bindings/mmc/mmc-controller.yaml

カーネルコンフィギュレーション

```

Device Drivers --->
<*> MMC/SD/SDIO card support --->                                <MMC>
<*>  MMC block device driver                                       <MMC_BLOCK>
(32)  Number of minors per block device   <MMC_BLOCK_MINORS>
*** MMC/SD/SDIO Host Controller Drivers ***
<*>  Secure Digital Host Controller Interface support              <MMC_SDHCI>
<*>    SDHCI platform and OF driver helper   <MMC_SDHCI_PLTFM>
<*>    SDHCI OF support for the Freescale eSDHC controller      <MMC_SDHCI_OF_ESDHC>
<*>    SDHCI support for the Freescale eSDHC/uSDHC i.MX
controller support                                               <MMC_SDHCI_ESDHC_IMX>
    
```



11.2. Ethernet

Armadillo-X2 の Ethernet(LAN)は、i.MX 8M Plus の ENET(Ethernet MAC)を利用しています。

Armadillo-X2 では、LAN インターフェース 1(CON3)が ENET を利用しています。

- 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) ^[1]
 - ・ Auto Negotiation サポート
 - ・ キャリア検知サポート
 - ・ リンク検出サポート

- 関連するソースコード
- ・ drivers/net/ethernet/freescale/fec_main.c
 - ・ drivers/net/phy/marvell.c
 - ・ drivers/net/mdio/of_mdio.c

- Device Tree ドキュメント
- ・ Documentation/devicetree/bindings/net/fsl-fec.txt
 - ・ Documentation/devicetree/bindings/net/mdio-mux.txt
 - ・ Documentation/devicetree/bindings/net/ethernet-phy.yaml

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

カーネルコンフィギュレーション

```

Device Drivers --->
[*] Network device support ---> <NETDEVICES>
  [*] Ethernet driver support ---> <ETHERNET>
    [*] Freescale devices <NET_VENDOR_FREESCALE>
    <*> FEC ethernet controller (of ColdFire and some i.MX CPUs) <FEC>
  -*- PHY Device support and infrastructure ---> <PHYLIB>
    [*] Support LED triggers for tracking link state <LED_TRIGGER_PHY>
    <*> Marvell Alaska PHYs <MARVELL_PHY>
```

11.3. USB ホスト

Armadillo-X2 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-X2 では、USB インターフェース(CON4)が USB1 を利用しています。USB2 には「11.4. USB ハブ」に示す USB2422 が接続されています。

- 機能
- ・ USB specification rev 3.0 準拠
 - ・ xHCI(eXtensible Host Controller Interface)互換

^[1]1000Mbps(1000BASE-T)は Half-Duplex に非対応です。

- ・ 転送レート: Super-speed(5 Gbps), high-speed(480 Mbps), full-speed(12 Mbps), low-speed(1.5 Mbps)
- 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。
- 関連するソースコード
- ・ drivers/usb/dwc3/dwc3-imx8mp.c
 - ・ drivers/phy/freescale/phy-fsl-imx8mq-usb.c
- Device Tree ドキュメント
- ・ Documentation/devicetree/bindings/usb/dwc3-imx8mp.txt
 - ・ Documentation/devicetree/bindings/phy/fsl,imx8mq-usb-phy.txt

カーネルコンフィギュレーション

```

Device Drivers --->
[*] USB support --->                                <USB_SUPPORT>
  <*> Support for Host-side USB                        <USB>
      *** USB Host Controller Drivers ***
  <*> xHCI HCD (USB 3.0) support                       <USB_XHCI_HCD>
  -* Generic xHCI driver for a platform device
                                          <USB_XHCI_PLATFORM>
  <*> DesignWare USB3 DRD Core Support                 <USB_DWC3>
      DWC3 Mode Selection (Dual Role mode) --->
                                          <USB_DWC3_DUAL_ROLE>
  <*> NXP iMX8MP Platform                             <USB_DWC3_IMX8MP>
PHY Subsystem --->
  -* PHY Core                                          <GENERIC_PHY>
  <*> Freescale i.MX8M USB3 PHY                       <PHY_FSL_IMX8MQ_USB>
    
```

11.4. USB ハブ

Armadillo-X2 には、Microchip 製 USB2422 が搭載されています。USB2422 は、「13.2.7. CON11 (拡張インターフェース)」および「13.2.11. CON17 (USB インターフェース 1)」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)
- 関連するソースコード
- ・ drivers/usb/misc/usb251xb.c
- Device Tree ドキュメント
- ・ Documentation/devicetree/bindings/usb/usb251xb.txt

カーネルコンフィギュレーション

```

Device Drivers --->
[*] USB support --->                                <USB_SUPPORT>
      *** USB Miscellaneous drivers ***
  <*> USB251XB Hub Controller Configuration Driver
                                          <USB_HUB_USB251XB>
    
```

11.5. UART

Armadillo-X2 の UART は、i.MX 8M Plus の UART(Universal Asynchronous Receiver/Transmitter)を利用しています。

Armadillo-X2 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。

- 関連するソースコード
- ・ drivers/tty/serial/imx.c
 - ・ drivers/tty/serial/imx_earlycon.c

- Device Tree ドキュメント
- ・ Documentation/devicetree/bindings/serial/fsl-imx-uart.yaml

デバイスファイル

シリアルインターフェース	デバイスファイル
UART1	/dev/ttymx0
UART2	/dev/ttymx1

カーネルコンフィギュレーション

```
Device Drivers --->
Character devices --->
  [*] Enable TTY <TTY>
    Serial drivers --->
      <*> IMX serial port support <SERIAL_IMX>
      <*> Console on IMX serial port <SERIAL_IMX_CONSOLE>
      [*] Earlycon on IMX serial port <SERIAL_IMX_EARLYCON>
```

11.6. HDMI

Armadillo-X2 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-X2 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^[2]、ALSA(Advanced Linux Sound Architecture)デバイスとして利用することができます。

^[2] 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。

- 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)
 - ・ 走査方式: プログレッシブ



上記を満たしていても、画像出力できない場合があります。次の VIC ^[3] 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

- 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)

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

- 関連するソースコード
- ・ drivers/gpu/drm/imx/dw_hdmi-imx.c
 - ・ drivers/phy/freescale/phy-fsl-samsung-hdmi.c
 - ・ drivers/clk/imx/clk-blk-ctrl.c
 - ・ drivers/gpu/drm/imx/imx8mp-hdmi-pavi.c
 - ・ drivers/gpu/imx/lcdifv3/lcdifv3-common.c
 - ・ sound/soc/fsl/imx-cdnhdmi.c
 - ・ sound/soc/fsl/fsl_aud2htx.c

- Device Tree ドキュメント
- ・ Documentation/devicetree/bindings/phy/fsl,samsung-hdmi-phy.yaml
 - ・ Documentation/devicetree/bindings/sound/imx-audio-cdnhdmi.txt
 - ・ Documentation/devicetree/bindings/sound/fsl,aud2htx.txt

カーネルコンフィギュレーション

```
Device Drivers --->
Graphics support --->
```

^[3]CEA-861 規格の VIC コード。


```

<*) i.MX LCDIFV3 core support <IMX_LCDIFV3_CORE>
-* NXP i.MX8MP HDMI Audio Video (PVI/PAI) <IMX8MP_HDMI_PAVI>
<*) Freescale i.MX DRM HDMI <DRM_IMX_HDMI>
-* Common Clock Framework ---> <COMMON_CLK>
<*) IMX8MP CCM Clock Driver <CLK_IMX8MP>
<*) Sound card support ---> <SOUND>
<*) Advanced Linux Sound Architecture ---> <SND>
<*) ALSA for SoC audio support ---> <SND_SOC>
    SoC Audio for Freescale CPUs --->
        <*) SoC Audio support for i.MX boards with CDN HDMI port
            <SND_SOC_IMX_CDNHDMI>
PHY Subsystem --->
    -* AUDIO TO HDMI TX module support <SND_SOC_FSL_AUD2HTX>
    <*) Samsung HDMI PHY support <PHY_SAMSUNG_HDMI_PHY>
    
```



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
```

11.7. MIPI CSI-2

Armadillo-X2 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-X2 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^[4]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

^[4]Armadillo-X2 にカメラは付属していません。

- ・ カラーフォーマット (RAW): RAW6, RAW7, RAW8, RAW10, RAW12, RAW14
- デバイスファイル
 - ・ /dev/video2 ^[5]
- 関連するソースコード
 - ・ drivers/staging/media/imx/imx8-mipi-csi2-sam.c
 - ・ drivers/staging/media/imx/imx8-media-dev.c
 - ・ drivers/staging/media/imx/imx8-isi-core.c
 - ・ drivers/staging/media/imx/imx8-isi-cap.c
- Device Tree ドキュメント
 - ・ Documentation/devicetree/bindings/media/imx8-media-dev.txt

カーネルコンフィギュレーション


```

Device Drivers --->
[*] Staging drivers --->                                <STAGING>
  [*] Media staging drivers --->                          <STAGING_MEDIA>
    <*> i.MX V4L2 media core driver                        <VIDEO_IMX_CAPTURE>
        i.MX8QXP/QM Camera ISI/MIPI Features support --->
    <*> IMX8 MIPI CSI2 SAMSUNG Controller <IMX8_MIPI_CSI2_SAM>
    <*> IMX8 Image Sensor Interface Core Driver <IMX8_ISI_CORE>
    <*> IMX8 Image Sensor Interface Capture Device Driver
                                                <IMX8_ISI_CAPTURE>
    <*> IMX8 Media Device Driver                        <IMX8_MEDIA_DEVICE>
```

11.8. リアルタイムクロック

Armadillo-X2 のリアルタイムクロックは、Armadillo-X2 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
- 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。


- 関連するソースコード
 - ・ drivers/rtc/rtc-rv8803.c
 - ・ drivers/rtc/rtc-snvs.c

^[5]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。


- Device Tree ドキュメント
- ・ Documentation/devicetree/bindings/rtc/epson, rx8900. txt
 - ・ Documentation/devicetree/bindings/crypto/fsl-sec4. txt

カーネルコンフィギュレーション

```
Device Drivers --->
[*] Real Time Clock --->                                <RTC_CLASS>
    (rtc0) RTC used to synchronize NTP adjustment         <RTC_SYSTOHC_DEVICE>
<*> Micro Crystal RV8803, Epson RX8900                 <RTC_DRV_RV8803>
<*> Freescale SNVS RTC support                          <RTC_DRV_SNVS>
```

 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 (SNVS) 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。

11.9. ユーザースイッチとイベント信号

Armadillo-X2 に搭載されているユーザースイッチには、GPIO が接続されています。

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

ユーザースイッチと信号には、次に示すキーコードが割り当てられています。

表 11.1 キーコード

ユーザースイッチ	キーコード	イベントコード	X11 キーコード
SW1	KEY_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 [6]
- 関連するソースコード . drivers/input/keyboard/gpio_keys.c
- Device Tree ドキュメント . Documentation/devicetree/bindings/input/gpio-keys.yaml
- カーネルコンフィギュレーション

```

Device Drivers --->
Input device support --->
  *- Generic input layer (needed for keyboard, mouse, ...)
                                     <INPUT>
  [*] Keyboards --->                 <INPUT_KEYBOARD>
  <*> GPIO Buttons                    <KEYBOARD_GPIO>
    
```

11.10. LED

Armadillo-X2 に搭載されているユーザー LED には、GPIO が接続されています。

Linux では、GPIO 接続用 LED ドライバ(leds-gpio)で制御することができます。

- sysfs LED クラスディレクトリ . /sys/class/leds/led1
- 関連するソースコード . drivers/leds/leds-gpio.c
- Device Tree ドキュメント . Documentation/devicetree/bindings/leds/leds-gpio.yaml
- カーネルコンフィギュレーション

```

Device Drivers --->
[*] LED Support --->                 <NEW_LEDS>
  <*> LED Support for GPIO connected LEDs <LEDS_GPIO>
    
```

11.11. I2C

Armadillo-X2 の 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。

Armadillo-X2 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 11.2 I2C デバイス


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

[6]USB キーボードなどを接続してインプットデバイスを追加している場合は、番号が異なる可能性があります

Armadillo-X2 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。

関連するソースコード ・ drivers/i2c/busses/i2c-imx.c

Device Tree ドキュメン
ト ・ Documentation/devicetree/bindings/i2c/i2c-imx.yaml

カーネルコンフィギュレー
ション

```
Device Drivers --->
I2C support --->
  *- I2C support                                     <I2C>
  <*> I2C device interface                           <I2C_CHARDEV>
  I2C Hardware Bus support --->
  <*> IMX I2C interface                               <I2C_IMX>
```

11.12. GPIO

Armadillo-X2 の GPIO は、i.MX 8M Plus の GPIO(General Purpose Input/Output)を利用しています。

関連するソースコード ・ drivers/gpio/gpio-mxc.c

Device Tree ドキュメント ・ Documentation/devicetree/bindings/gpio/fsl-imx-gpio.yaml

デバイスファイル

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

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



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

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

カーネルコンフィギュレーション

```
Device Drivers --->
[*] GPIO Support --->                                <GPIOLIB>
    (512) Maximum number of GPIOs for fast path
                                                <GPIOLIB_FASTPATH_LIMIT>

Memory mapped GPIO drivers --->
-* i.MX GPIO support                                <GPIO_MXC>
```

11.13. 温度センサー

Armadillo-X2 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) |
| 関連するソースコード | ・ drivers/thermal/imx8mm_thermal.c |
| Device Tree ドキュメント | ・ Documentation/devicetree/bindings/thermal/imx8mm-thermal.yaml |

カーネルコンフィギュレーション

```
Device Drivers --->
-* Thermal drivers --->                                <THERMAL>
    <*> Temperature sensor driver for Freescale i.MX8MM SoC
                                                <IMX8MM_THERMAL>
```

11.14. ウォッチドッグタイマー

Armadillo-X2 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。

- 関連するソースコード ・ drivers/watchdog/imx2_wdt.c
- Device Tree ドキュメント ・ Documentation/devicetree/bindings/watchdog/fsl-imx-wdt.yaml

カーネルコンフィギュレーション

```
Device Drivers --->
[*] Watchdog Timer Support --->           <WATCHDOG>
  <*> IMX2+ Watchdog                       <IMX2_WDT>
```

ウォッチドッグタイマーの設定変更は、ioctl システムコール経由で行うことができます。詳細な情報については、Linux カーネルのソースコードに含まれているドキュメント(Documentation/watchdog/watchdog-api.rst)を参照してください。



ウォッチドッグタイマーを停止することはできません。

11.15. パワーマネジメント

Armadillo-X2 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-X2 の消費電力を抑えることができます。

パワーマネジメント状態を省電力モードに遷移させると、アプリケーションの実行は一時停止し、Linux カーネルはサスペンド状態となります。起床要因が発生すると、Linux カーネルのリジューム処理が行われた後、アプリケーションの実行を再開します。

sysfs ファイル ・ /sys/power/state

関連するソースコード ・ kernel/power/

カーネルコンフィギュレーション

```
Power management options --->
[*] Suspend to RAM and standby           <SUSPEND>
-* Device power management core functionality <PM>
```

Armadillo-X2 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 11.3 対応するパワーマネジメント状態

パワーマネジメント状態	文字列	説明
Suspend-to-RAM	mem	Suspend-to-Idle よりも消費電力を抑えることができる
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	<pre>[armadillo ~]# echo enabled > /sys/class/tty/ ttymxc1/power/wakeup</pre>	↵
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	<pre>[armadillo ~]# echo enabled > /sys/devices/ platform/soc@0/32f10100.usb/power/wakeup [armadillo ~]# echo enabled > /sys/bus/usb/devices/ usb1/power/wakeup</pre>	↵
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	

12. ソフトウェア仕様

12.1. SWUpdate

12.1.1. SWUpdate とは

SWUpdate は、デバイス上で実行されるプログラムで、ネットワーク/ストレージ経由でデバイスのソフトウェアを更新することができます。Stefano Babic, DENX software engineering, Germany によってオープンソースで開発が進められています。

Armadillo-X2 では、SWUpdate を利用することで次のような機能を実現しています。

- ・ A/B アップデート(アップデートの 2 面化)
- ・ リカバリーモード
- ・ ソフトウェアの圧縮、暗号化、署名付与
- ・ Web サーバー機能
- ・ hawkBit への対応
- ・ ダウングレードの禁止

12.1.2. swu パッケージ

swu パッケージは、SWUpdate 独自のソフトウェアの配布フォーマットです。SWUpdate では、1 回のアップデートは 1 つの swu パッケージで行われます。

swu パッケージには、次のような様々なものを含めることができます。

- ・ アップデート対象のイメージファイル
- ・ アップデート対象のイメージファイルのチェックサム
- ・ アップデート前後に実行するスクリプト
- ・ 書き込み先ストレージの情報
- ・ U-Boot 環境変数の書き換え情報
- ・ ソフトウェアのバージョン情報
- ・ etc...

12.1.3. A/B アップデート(アップデートの 2 面化)

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

常に使用していない方のパーティションを書き換えるため次の特徴を持ちます。

- ・ ○ アップデートによって動作中のソフトウェアは破壊されない
- ・ ○ 書き込みが電源断などで中断後しても、すぐに復帰出来る
- ・ ○ 機器が動作中に書き込みが出来る
- ・ × 使用 Flash メモリ量が増える

12.1.4. ロールバック (リカバリー)

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

ロールバック状態の確認は「9.11.3. ロールバック状態の確認」を参照してください。

ロールバックする条件は次の通りです:

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

bootcount 機能は u-boot の「upgrade_available」変数で管理されています。bootcount 機能を利用しないようにするには、「9.11.6. u-boot の環境変数の設定」を参照して変数を消します。

- ・ ユーザーのスクリプトなどから、「abos-ctrl rollback」コマンドを実行した場合。

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

12.2. hawkBit

12.2.1. hawkBit とは

hawkBit は、サーバー上で実行されるプログラムで、ネットワーク経由でデバイスのソフトウェアを更新(配信)することができます。

hawkBit は次のような機能を持っています。

- ・ ソフトウェアの管理
- ・ デバイスの管理
 - ・ デバイス認証 (セキュリティトークン、証明書)
 - ・ デバイスのグループ化
- ・ アップデート処理の管理
 - ・ 進捗のモニタリング
 - ・ スケジューリング、強制アップデート
- ・ RESTful API での直接操作

12.2.2. データ構造

hawkBit は、配信するソフトウェアを次のデータ構造で管理します。

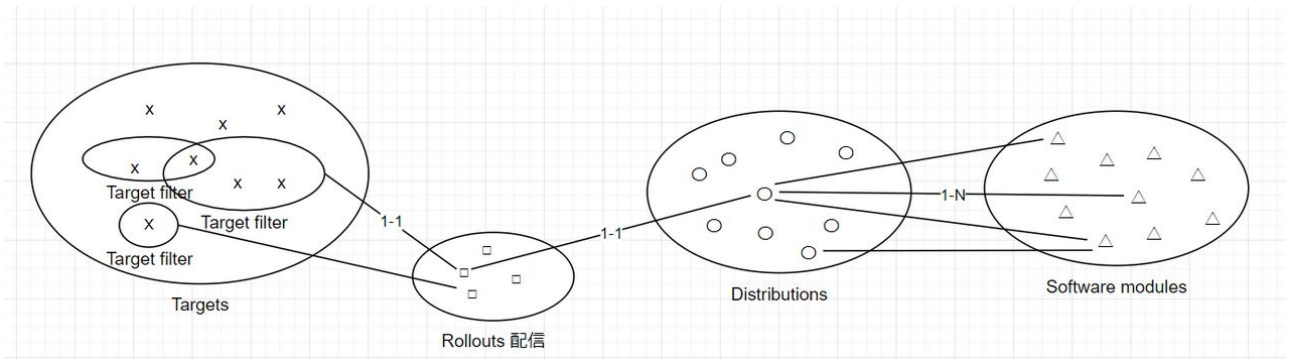


図 12.1 hawkBit が扱うソフトウェアのデータ構造

13. ハードウェア仕様

13.1. 電氣的仕様

13.1.1. 絶対最大定格

表 13.1 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	-
入出力電圧	VI,VO(VDD_3V3)	-0.3	OVDD+0.3	V	OVDD=VDD_3V3
	VI,VO(VDD_1V8)				OVDD=VDD_1V8
	VI,VO(NVCC_SNV5_1V8)				OVDD=NVCC_SNV5_1V8
	VI,VO(VEXT_3V3)	-0.5	7.0	V	-
USB コンソール電源電圧	VBUS_CN5L	-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	Topr	-20	70	°C	結露なきこと



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

13.1.2. 推奨動作条件

表 13.2 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_CN5L	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°C、対応電池: CR1220 等

13.1.3. 電源出力仕様

表 13.3 電源出力仕様

項目	記号	Min.	Typ.	Max.	単位	備考
5V 電源	VDD_5V	4.85	5	5.15	V	-
	USB1_VBUS					
	USB2_VBUS					
	HDMI_5V					
3.3V 電源	VDD_3V3	3.135	3.3	3.465	V	-
	VDD_SD					
	VEXT_3V3					
1.8V 電源	VDD_1V8	1.71	1.8	1.89	V	-
	NVCC_SNV5_1V8					

13.1.4. 入出インターフェースの電氣的仕様

表 13.4 拡張入出力ピンの電氣的仕様(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
ハイレベル入力電圧	VIH	0.7xOVDD	-	OVDD+0.3	V	-
ローレベル入力電圧	VIL	-0.3	-	0.3xOVDD	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 13.5 拡張入出力ピンの電氣的仕様(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	0.7xOVDD	-	OVDD+0.3	V	-
ローレベル入力電圧	VIL	-0.3	-	0.3xOVDD	V	-

13.1.5. 電源回路の構成

Armadillo-X2 の電源回路の構成は「図 13.1. 電源回路の構成」のとおりです。

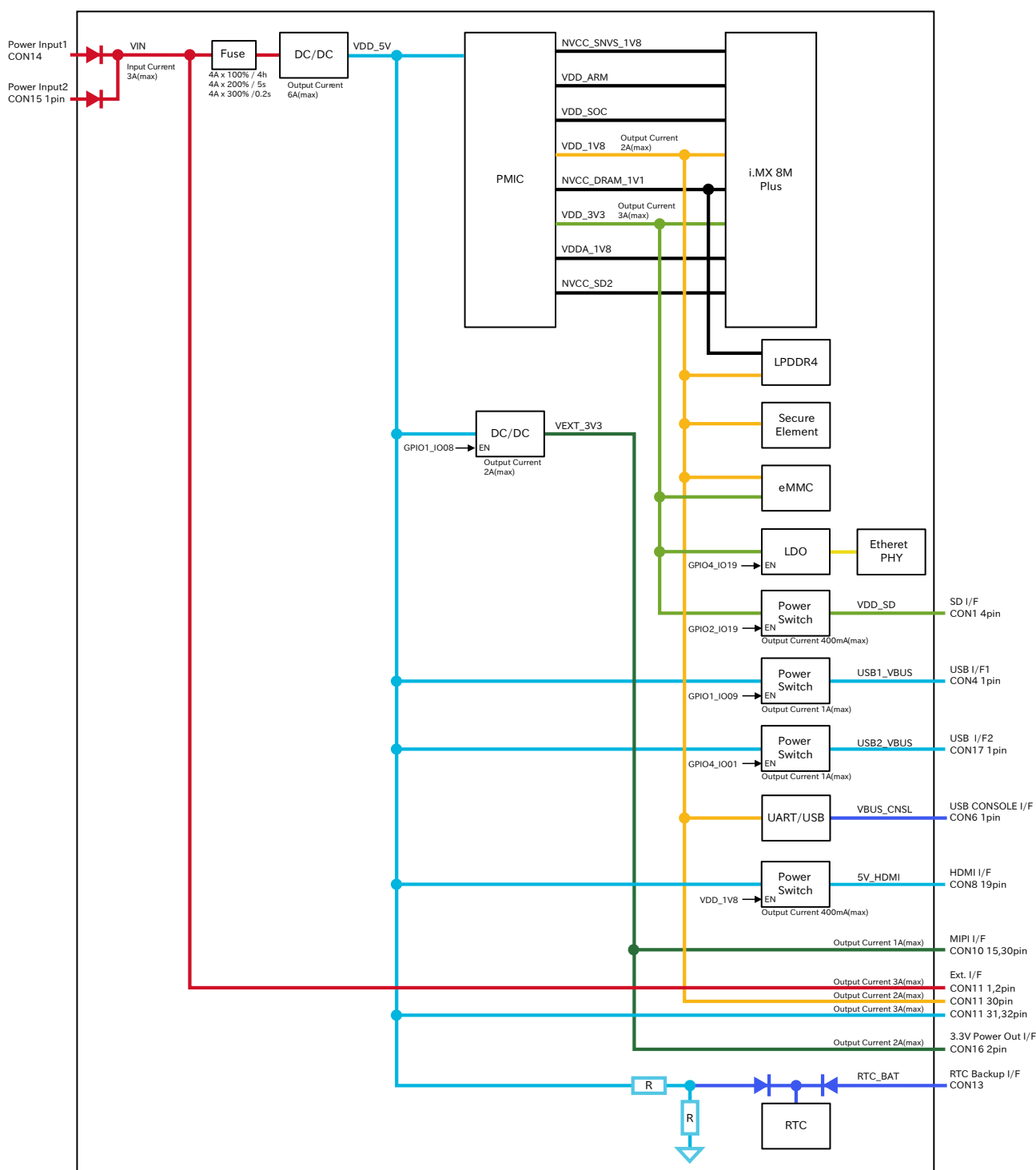


図 13.1 電源回路の構成

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。

13.1.6. 電源シーケンス

電源シーケンスは「図 13.2. 電源シーケンス」のとおりです。

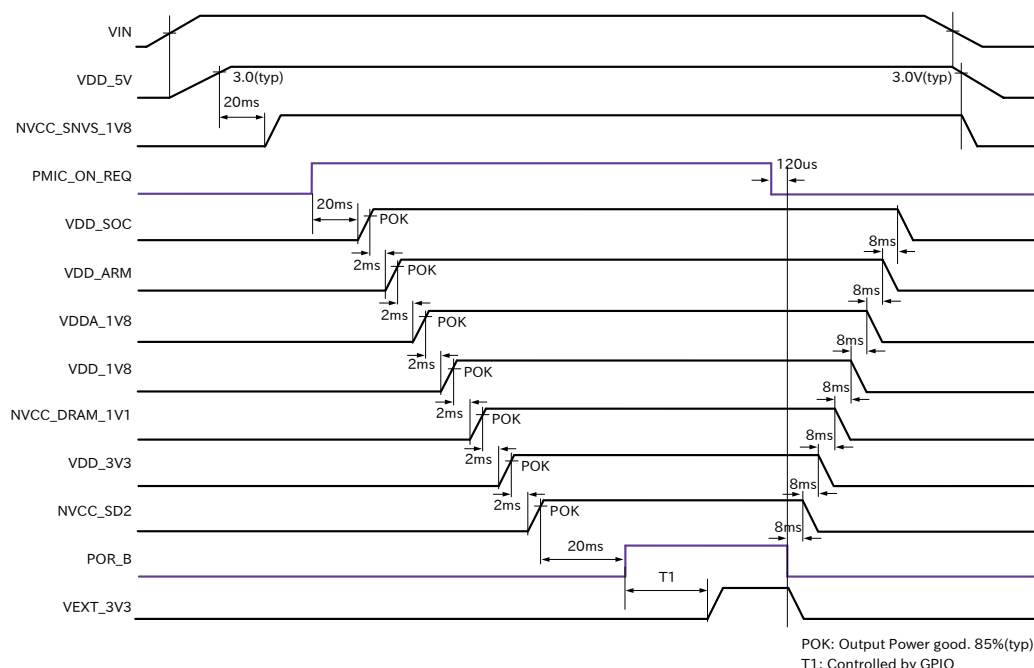


図 13.2 電源シーケンス

13.1.6.1. 電源オン時

Armadillo-X2 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。

13.1.6.2. 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-X2 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。

13.1.7. リセット回路の構成

Armadillo-X2 のリセット回路の構成は「図 13.3. リセット回路の構成」のとおりです。

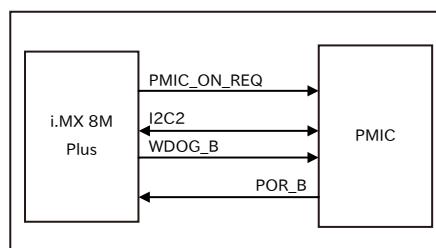


図 13.3 リセット回路の構成

13.1.8. リセットシーケンス

リセットシーケンスは「図 13.4. リセットシーケンス」のとおりです。

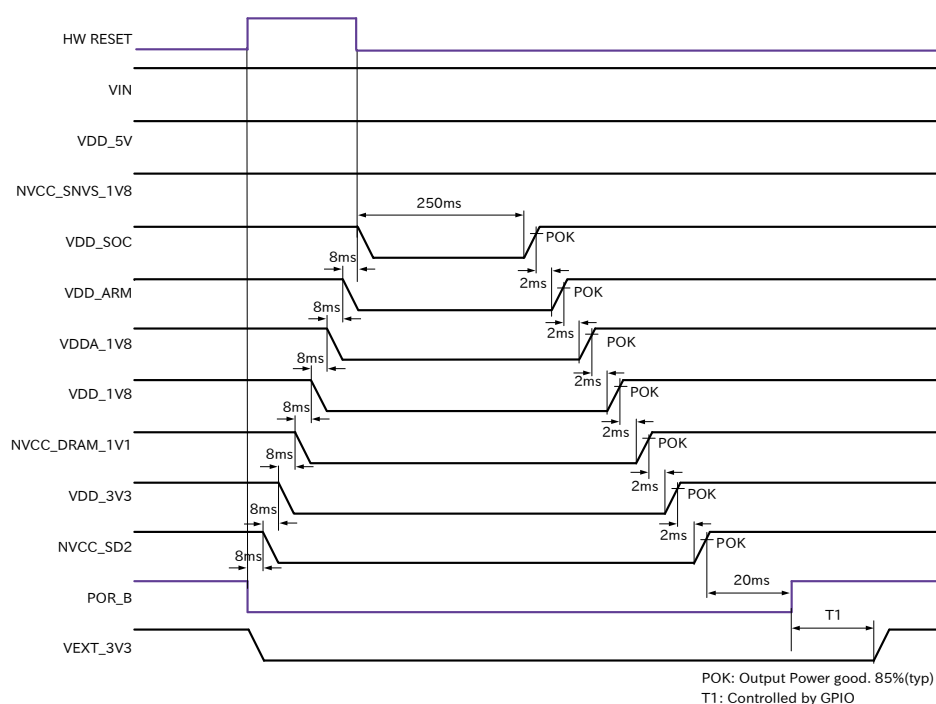


図 13.4 リセットシーケンス

Armadillo-X2 のハードウェアリセットには、I2C によるリセット^[1]、ウォッチドックタイマーによるリセットがあります。

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。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。

13.1.9. 外部からの電源制御

13.1.9.1. リアルタイムクロックからの電源制御

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。

^[1]reboot コマンド時は、I2C によりリセットされます。

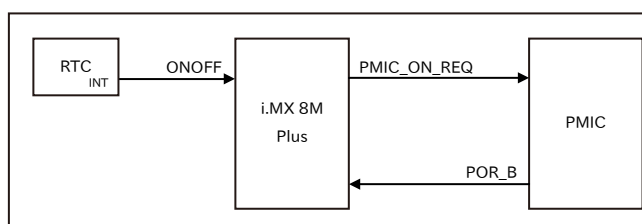


図 13.5 ONOFF 回路の構成

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。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。

表 13.6 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



オフ状態にして Armadillo-X2 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

13.2. インターフェース仕様

Armadillo-X2 のインターフェース仕様について説明します。

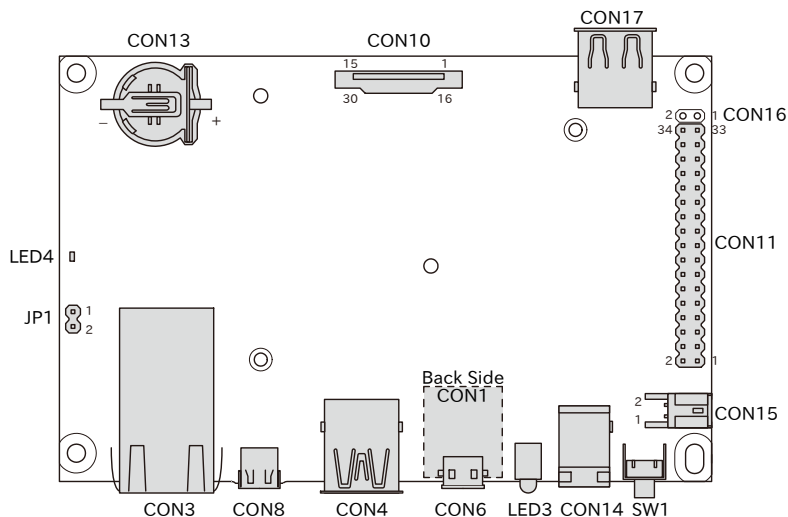


図 13.6 Armadillo-X2 のインターフェース

表 13.7 Armadillo-X2 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
CON3	LAN インターフェース	56F-1304DYDZ2NL	YUAN DEAN SCIENTIFIC
CON4	USB インターフェース 1	GSB3111311HR	Amphenol ICC
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.
CON8	HDMI インターフェース	DC3RX19JA2R1700	Japan Aviation Electronics Industry
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	Würth Elektronik
CON13	RTC バックアップインターフェース	BH-44C-5	Adam Tech
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.
CON16	3.3V 電源出力インターフェース	61300211121	Würth Elektronik
CON17	USB インターフェース 2	SS-52100-001	Bel Fuse Inc.
JP1	起動デバイス設定ジャンパ	61300211121	Würth 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



「表 13.7. Armadillo-X2 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。



Armadillo-X2 の各インターフェースのピンアサインは Armadillo-IoT ゲートウェイ G4 と同じです。

13.2.1. CON1 (SD インターフェース)

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 13.8 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ありますので、取り扱いにはご注意ください。

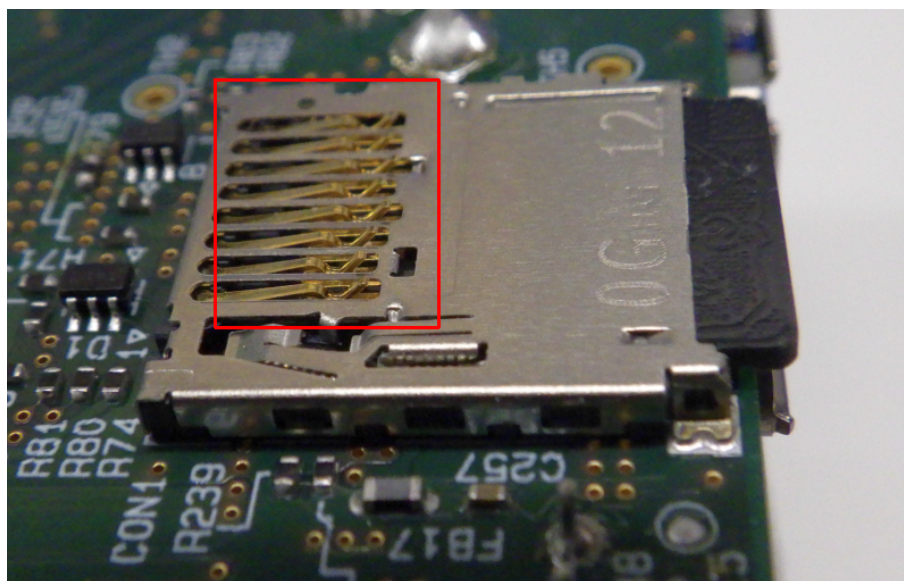


図 13.7 CON1 microSD スロット 取り扱い上の注意事項

13.2.2. CON3 (LAN インターフェース)

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 Ethernet PHY を経由して i.MX 8M Plus の Ethernet MAC(ENET)に接続されています。

表 13.9 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 13.10 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 13.11 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

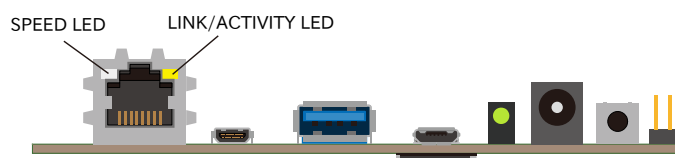


図 13.8 CON3 LAN LED 配置

13.2.3. CON4 (USB インターフェース 1)

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 13.12 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

13.2.4. CON6 (USB コンソールインターフェース)

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 13.13 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)

13.2.5. CON8 (HDMI インターフェース)

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 13.14 CON7 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

ピン番号	ピン名	I/O	説明
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)
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)

13.2.6. CON10 (MIPI-CSI インターフェース)

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。

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 13.15 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)

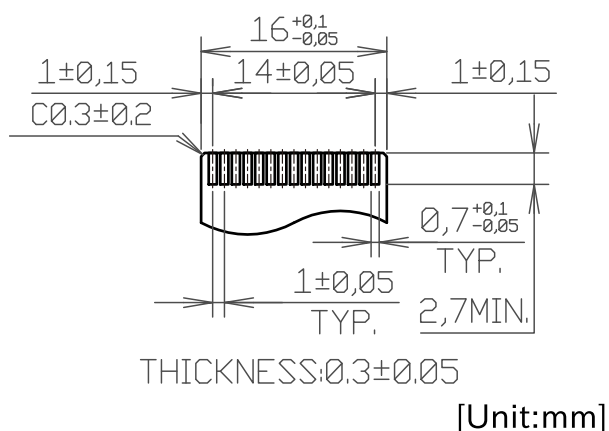


図 13.9 CON10 接続可能なフレキシブルフラットケーブルの形状


表 13.16 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	-

ピン番号	ピン名	I/O	説明	電圧グループ
7	GND	Power	電源(GND)	-
8	CSI1_CLK_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_CLK_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
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_CLK_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_CLK_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)	-

13.2.7. CON11 (拡張インターフェース)

CON11 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。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。

表 13.17 CON11 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]	Würth Elektronik	許容電流 3A(端子 1 本あたり)
対向コネクタ	6130xx21821 [a]	Würth Elektronik	-

[a]xx にはピン数が入ります。

表 13.18 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
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

ピン番号	ピン名	I/O	説明	電圧グループ
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	USB DP_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)	-



拡張できる機能の詳細につきましては、「アットマークテクノ Armadillo サイト」 [<https://armadillo.atmark-techno.com/>] からダウンロードできる『Armadillo-X2 マルチプレクス表』をご参照ください。

13.2.7.1. USB

USB 2.0 Host を 1 ポート拡張することが可能です。信号線は USB HUB 経由で USB コントローラ (USB2) に接続されています。

- ・ 転送速度
 - ・ High Speed(480Mbps)
 - ・ Full Speed(12Mbps)
 - ・ Low Speed(1.5Mbps)

13.2.7.2. GPIO

GPIO を最大 21 ポート拡張することが可能です。

- ・ 信号レベル : VDD_1V8

13.2.7.3. SPI

SPI 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 ESPI(ECSPI1、ECSPI2) に接続されています。

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

13.2.7.4. UART

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。

- ・ 最大データ転送レート: 4Mbps
- ・ 信号レベル: VDD_1V8

13.2.7.5. PDM MIC

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

13.2.7.6. I2S(SAI)

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

13.2.7.7. CAN

CAN を最大 2 ポート拡張することが可能です。信号線は i.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

13.2.7.8. I2C

I2C を最大 3 ポート拡張することが可能です。信号線は i.MX6ULL の I2C コントローラ(I2C4、I2C5、I2C6)に接続されています。

- ・ 最大データ転送レート: 320kbps
- ・ 信号レベル: VDD_1V8

13.2.7.9. PWM

PWM を最大 4 ポート拡張することが可能です。

- ・ 最大周波数: 66MHz
- ・ 信号レベル: VDD_1V8

13.2.8. CON13(RTC バックアップインターフェース)

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。

CON10 には CR1220、BR1220 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.)です。

表 13.19 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)



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°C~70°Cで8秒(参考値)です。

リアルタイムクロックの時間精度は周囲温度に大きく影響を受けますので、使用温度での十分な特性の確認をお願いいたします。



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。

13.2.9. CON14、CON15 (電源入力インターフェース)

CON14、CON15 は電源入力用のインターフェースです。

CON14 には DC ジャックが実装されており、「図 13.10. 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 13.10 AC アダプタの極性マーク

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 13.20 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 13.21 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-X2 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。

13.2.10. CON16 (3.3V 電源出力インターフェース)


CON16 は 3.3V 電源出力用のインターフェースです。コネクタは未実装ですので、「表 13.22. CON15 搭載コネクタ例」を参考に部品実装してご使用ください。

表 13.22 CON15 搭載コネクタ例

名称	型番	メーカー	備考
搭載コネクタ	61300211121	Würth Elektronik	許容電流 3A(端子 1 本あたり)

表 13.23 CON16 信号配列

ピン番号	ピン名	I/O	説明
1	GND	Power	電源(GND)
2	VEXT_3V3	Power	電源出力(VEXT_3V3)



Armadillo-X2 の拡張インターフェース(CON11)から機能拡張する際の、外部デバイスへの電源供給での使用を想定しています。拡張しやすいように、CON16 の 1、2 ピンは CON11 の 33、34 ピンから 2.54mm 間隔で配置しています。

13.2.11. CON17 (USB インターフェース 1)

CON17 は USB 2.0 に対応した USB インターフェースです。信号線は USB HUB 経由で i.MX 8M Plus の USB コントローラ(USB2)に接続されています。

USB デバイスに供給される電源(USB2_VBUS)は、i.MX 8M Plus の GPIO4_IO01 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
 - ・ High Speed(480Mbps)

- ・ Full Speed(12Mbps)
- ・ Low Speed(1.5Mbps)

表 13.24 CON4 信号配列

ピン番号	ピン名	I/O	説明
1	USB2_VBUS	Power	電源出力(USB1_VBUS)
2	USBDM_DN1	In/Out	USB 2.0 データ(-)、USB HUB 経由で i.MX 8M Plus の USB2 に接続
3	USBDP_DN1	In/Out	USB 2.0 データ(+)、USB HUB 経由で i.MX 8M Plus の USB2 に接続
4	GND	Power	電源(GND)

13.2.12. JP1 (起動デバイス設定ジャンパ)

JP1 は起動デバイス設定ジャンパです。JP1 の状態で、起動デバイスを設定することができます。

表 13.25 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)

表 13.26 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)

13.2.13. SW1 (ユーザースイッチ)

SW1 は、ユーザー側で自由に利用できる押しボタンスイッチです。

表 13.27 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)

13.2.14. LED3(ユーザー LED)

LED3 は、ユーザー側で自由に利用できる LED です。

表 13.28 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)

13.2.15. LED4 (電源 LED)

LED4 は、Armadillo-X2 の電源確認用の LED です。

表 13.29 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

13.3. 形状図

13.3.1. 基板形状図

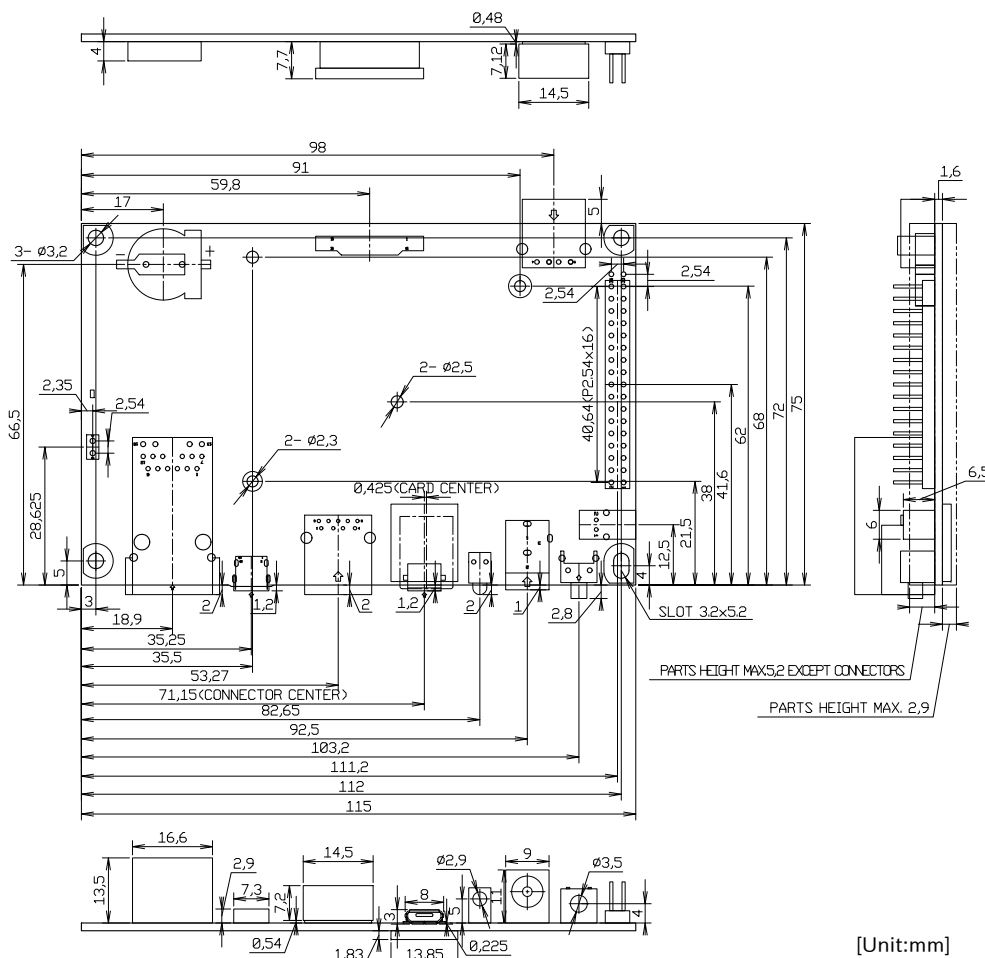



図 13.11 基板形状図



基板改版や部品変更により、基板上的部品位置、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。

13.4. 設計情報

本章では、Armadillo-X2 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。

13.4.1. 信頼性試験データについて

Armadillo-X2 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。

13.4.2. 放射ノイズ

HDMI インターフェース (CON8) にディスプレイを接続した場合や、MIPI-CSI インターフェース (CON10) にカメラを接続した場合に、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

13.4.3. ESD/雷サージ

Armadillo-X2 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-X2 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-X2 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

13.4.4. 放熱

SoC(基板裏の IC1)の放熱が必要かどうかは、使用状況により異なりますので、十分な設計評価の上、ご検討ください。SoC の表面温度が 90°C 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°C を超えることを確認しています。

- ・ 基板単体
- ・ 周囲温度: 約 65°C
- ・ microSD/HDMI/USB3.0/USB2.0/LAN 動作

Armadillo-X2 の周囲温度の上限は+70°C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/USB2.0/LAN 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、「13.5.1. Armadillo-X2 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($\phi 2.5\text{mm} \times 2$)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、「13.3. 形状図」をご確認ください。

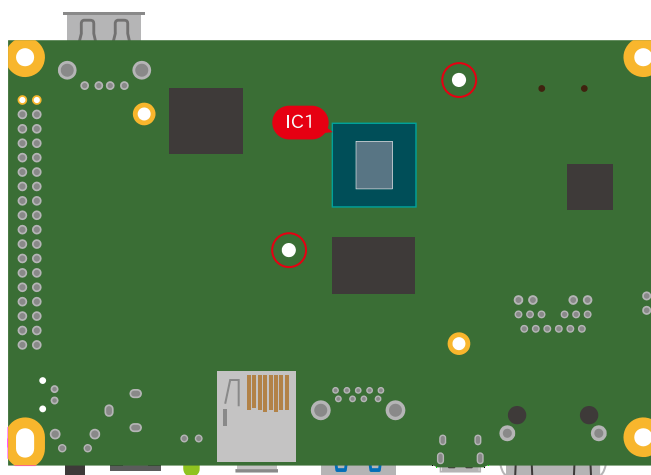


図 13.12 Armadillo-X2 の IC1 とヒートシンク固定穴の位置



Armadillo-X2 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、「11.13. 温度センサー」をご確認ください。

13.4.5. 拡張ボードの設計

Armadillo-X2 の拡張インターフェース(CON11)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。

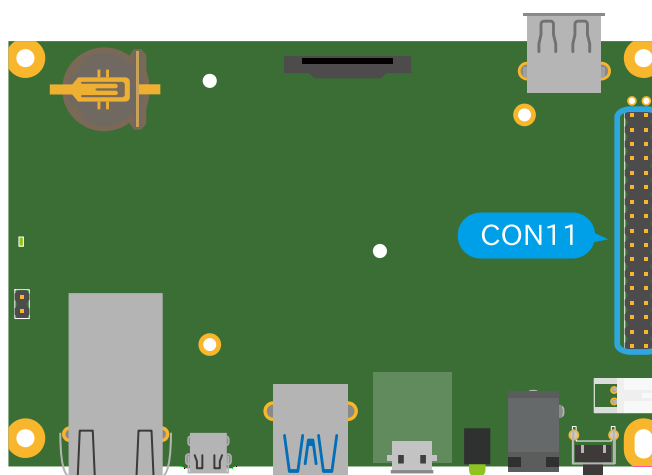


図 13.13 Armadillo-X2 の拡張インターフェース

13.4.5.1. ピンアサイン

Armadillo-X2 では、「表 3.2. 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-X2 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-X2 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「13.2.7. CON11 (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-X2 マルチプレクス表は「アットマークテクノ Armadillo サイト」 [<https://armadillo.atmark-techno.com/>] からダウンロードしてください。

13.4.5.2. 基板形状

Armadillo-X2 の拡張ボードを設計する際の推奨形状は「図 13.14. Armadillo-IoT ゲートウェイ X2 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-X2 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、 $\phi 2.3\text{mm}$ の穴を 2 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。

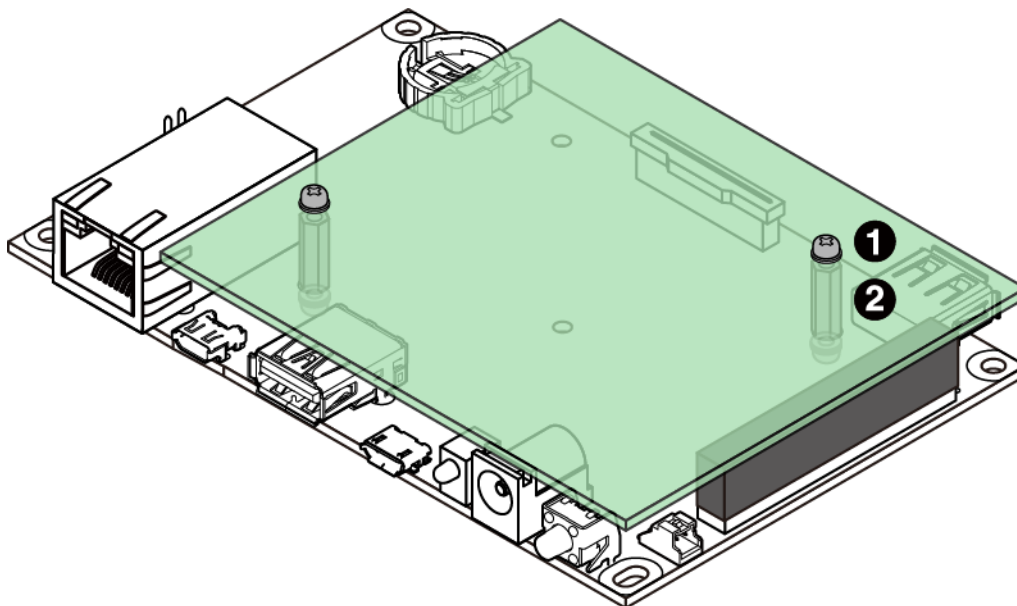


図 13.14 Armadillo-IoT ゲートウェイ X2 の拡張ボード例

- ① なべ小ねじ、ワッシャ、スプリングワッシャ付(M2、L=6mm) × 4
- ② 金属スペーサ(M2、L=11mm) × 2

基板の詳細寸法につきましては、「13.3.1. 基板形状図」をご確認ください。

13.4.6. 回路設計

拡張インターフェース(CON11)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。

13.4.6.1. 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。

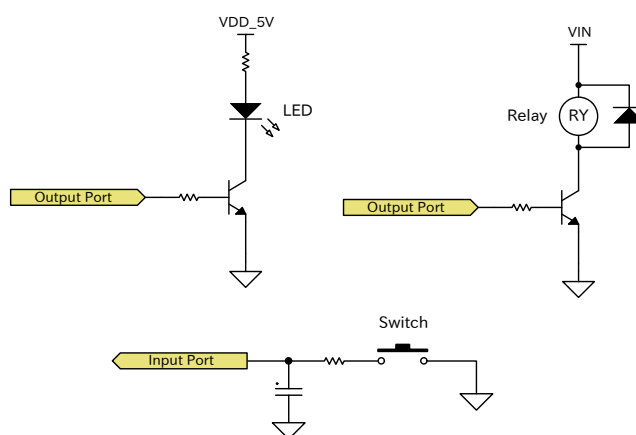



図 13.15 スイッチ、LED、リレー接続例

13.4.6.2. 電源

拡張インターフェース(CON11)から拡張ボード用に、12V 電源(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。3.3V 電源(VEXT_3V3)を 3.3V 電源出力インターフェース(CON16)から出力しており、こちらを利用することも可能です。

電源シーケンス、出力電流につきましては、「13.1.5. 電源回路の構成」をご確認ください。



3.3V 電源出力インターフェース(CON16)の 1、2 ピンは拡張インターフェース(CON11)の 33、34 ピンから 2.54mm 間隔で配置しています。

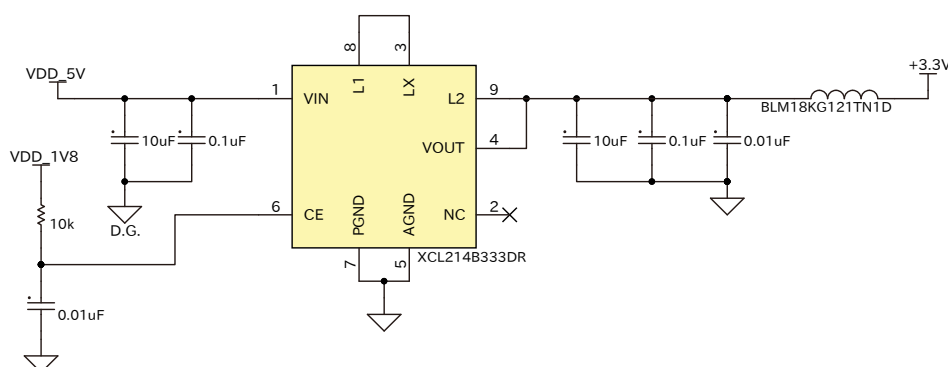



図 13.16 DC/DC コンバータ回路(VDD_5V 入力、3.3V 1.5A 出力)例

「図 13.1. 電源回路の構成」のインターフェース名(Ext. 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 13.30 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 1	USB1_VBUS	900mA
CON17	USB インターフェース 2	USB2_VBUS	500mA
CON10	MIPI-CSI インターフェース	VEXT_3V3	500mA
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
CON16	3.3V 電源出力インターフェース	VEXT_3V3	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。

13.4.6.3. レベル変換

拡張インターフェース(CON11)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 VDD_1V8、VDD_5V ピン、CON16 に VEXT_3V3 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。

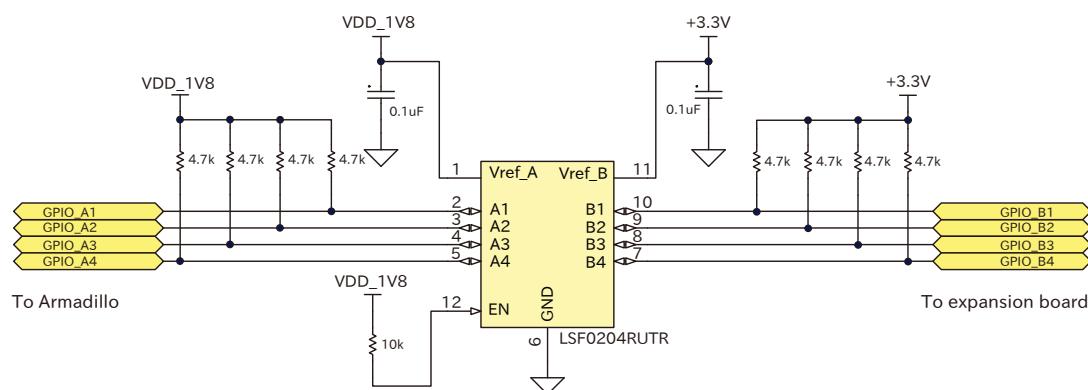



図 13.17 1.8V ↔ 3.3V 双方向レベル変換回路の例



上記レベル変換 IC は 1.8V ↔ 5V でも使用可能です。

13.5. オプション品

本章では、Armadillo-X2 のオプション品について説明します。

表 13.31 Armadillo-X2 関連のオプション品

名称	型番	備考
Armadillo-X2 オプションケース(金属製)	OP-CASEX2-MET-00	ケースモデルに付属
CPU 放熱シート 15x15x4mm 10 個セット	OP-THS-151504-00	
Armadillo-X2、G4 ケースモデル VESA 規格固定用プレート	OP-CASEX2-VESA-00	
AC アダプタ (12V/3.0A φ2.1mm) 標準品	OP-AC12V6-00	開発セットに付属

13.5.1. Armadillo-X2 オプションケース(金属製)

13.5.1.1. 概要

Armadillo-X2 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、USBx2、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。

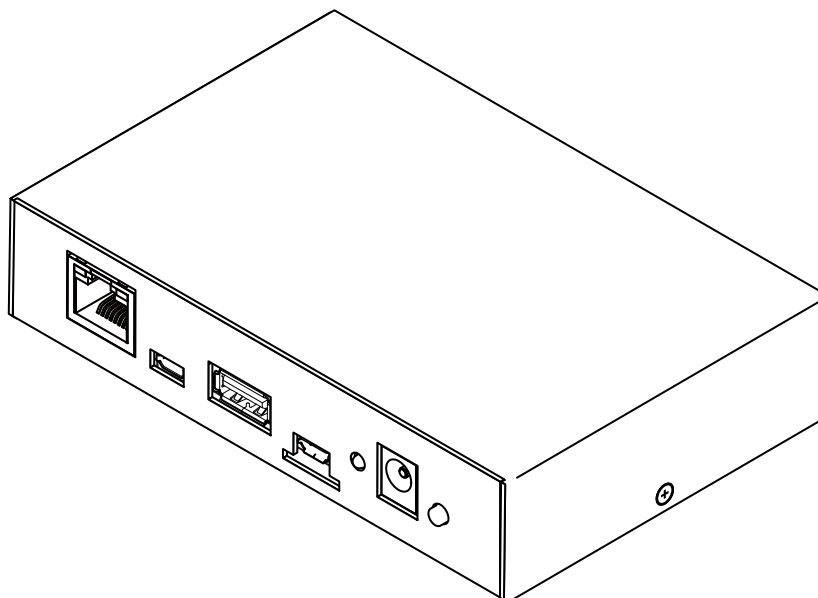


図 13.18 Armadillo-X2 オプションケース(金属製)

表 13.32 Armadillo-X2 オプションケースセット(金属製)について

製品名	Armadillo-X2 オプションケースセット(金属製)
型番	OP-CASEX2-MET-00
セット内容	アルミケース、ケースネジ×2、Armadillo-X2 固定用ネジ×4

表 13.33 Armadillo-X2 オプションケース(金属製)の仕様

材質	ヘアライン白アルマイト材
板厚	1.0 mm



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。

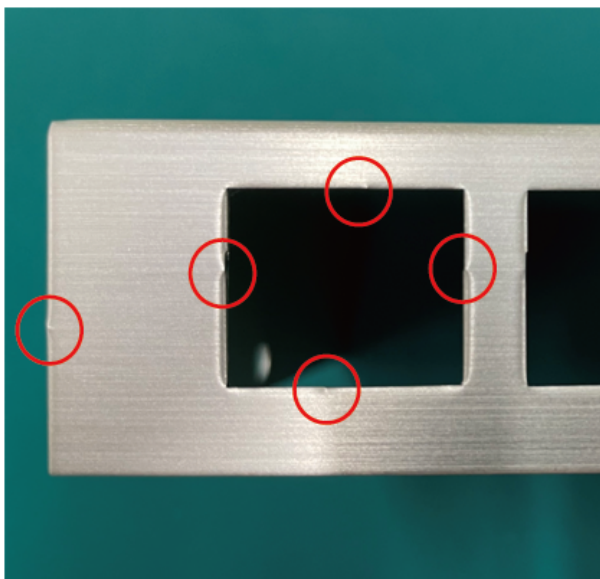


図 13.19 オプションケース(金属製)の加工痕例

13.5.1.2. 組み立て

ケース(下)に基板をねじ止めします。推奨のねじ締めトルクは 31.5cN・m です。

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。放熱する場合、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。

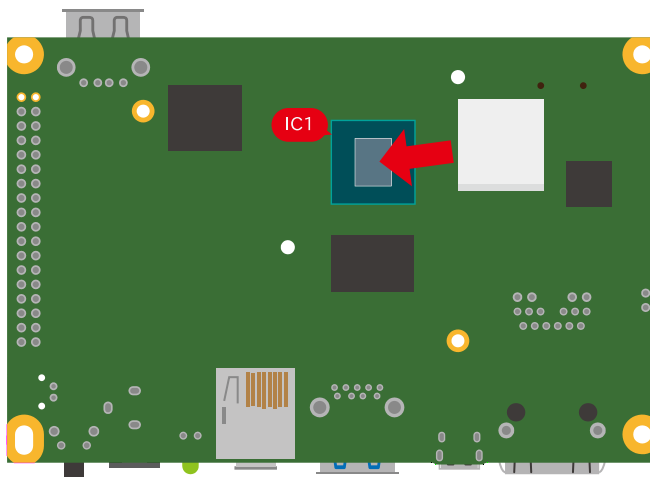


図 13.20 オプションケース(金属製) 放熱シート貼付

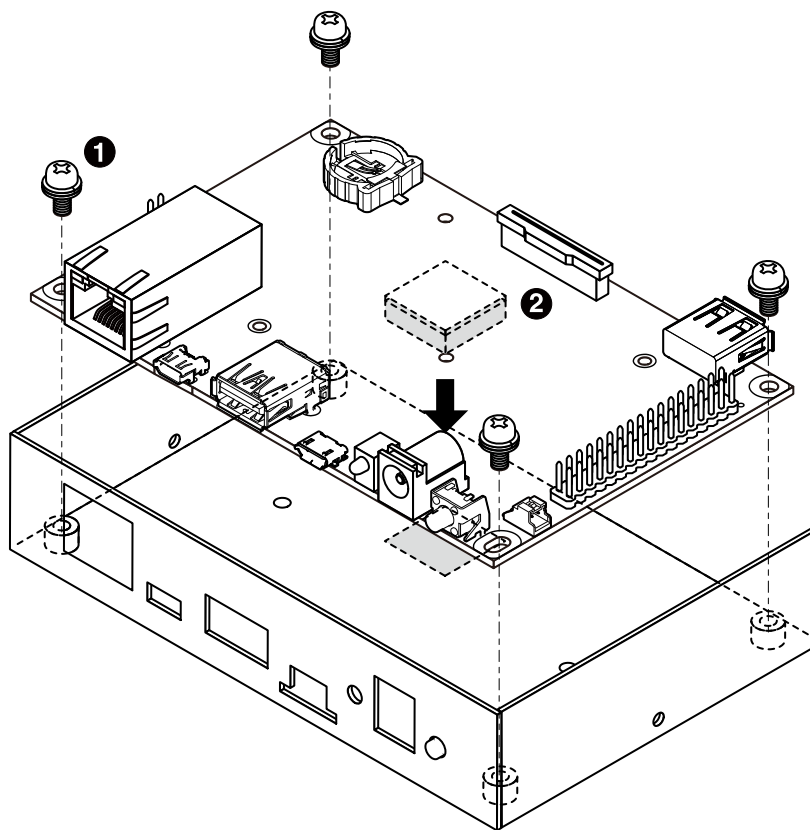


図 13.21 オプションケース(金属製) ケース(下)ねじ止め

- 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)



放熱シートは「Armadillo-X2 オプションケースセット」に含まれません。放熱シートが必要な場合は「CPU 放熱シート 15×15×4mm 10 個セット」をオプション品でラインアップしておりますので、ご検討ください。

ケース(上)とケース(下)は 2 箇所ねじ止めします。推奨のねじ締めトルクは 17.5cN・m です。

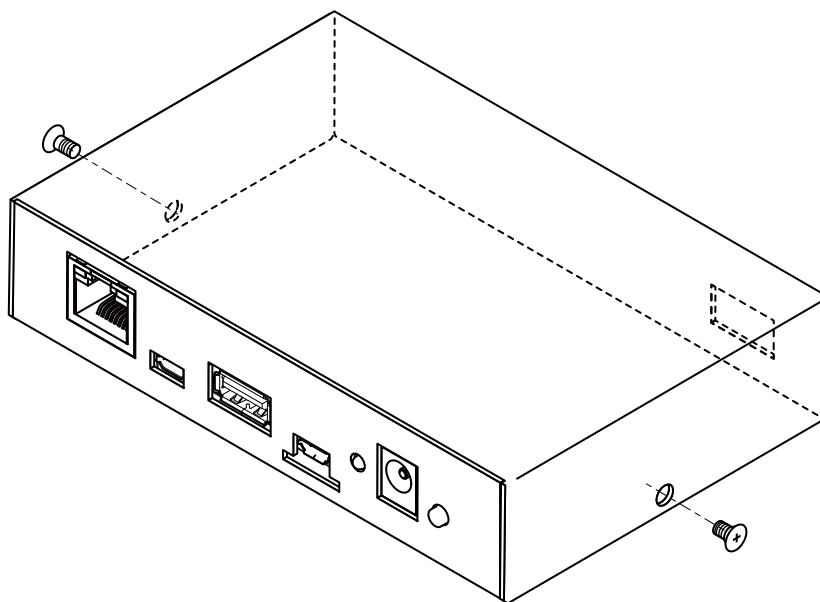



図 13.22 オプションケース(金属製) ケース(上)ねじ止め

- ① 皿ねじ(M2.6、L=4mm) × 2



ケース(上)を閉じる際にスライドさせると、LAN コネクタの接触バネ部分に干渉して折れたり曲がったりするため、接触バネに干渉しないように閉じてください。

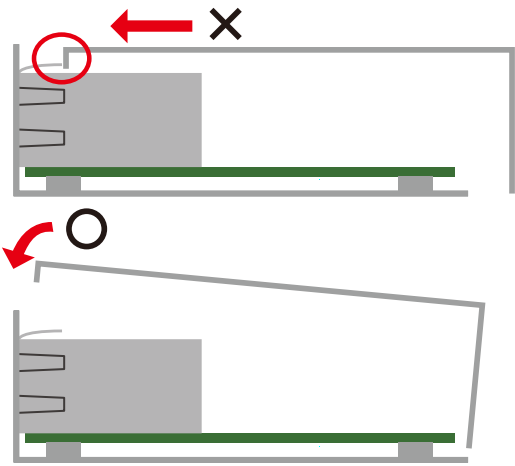


図 13.23 ケース(上)を閉じる際の注意

13.5.1.3. 形状図

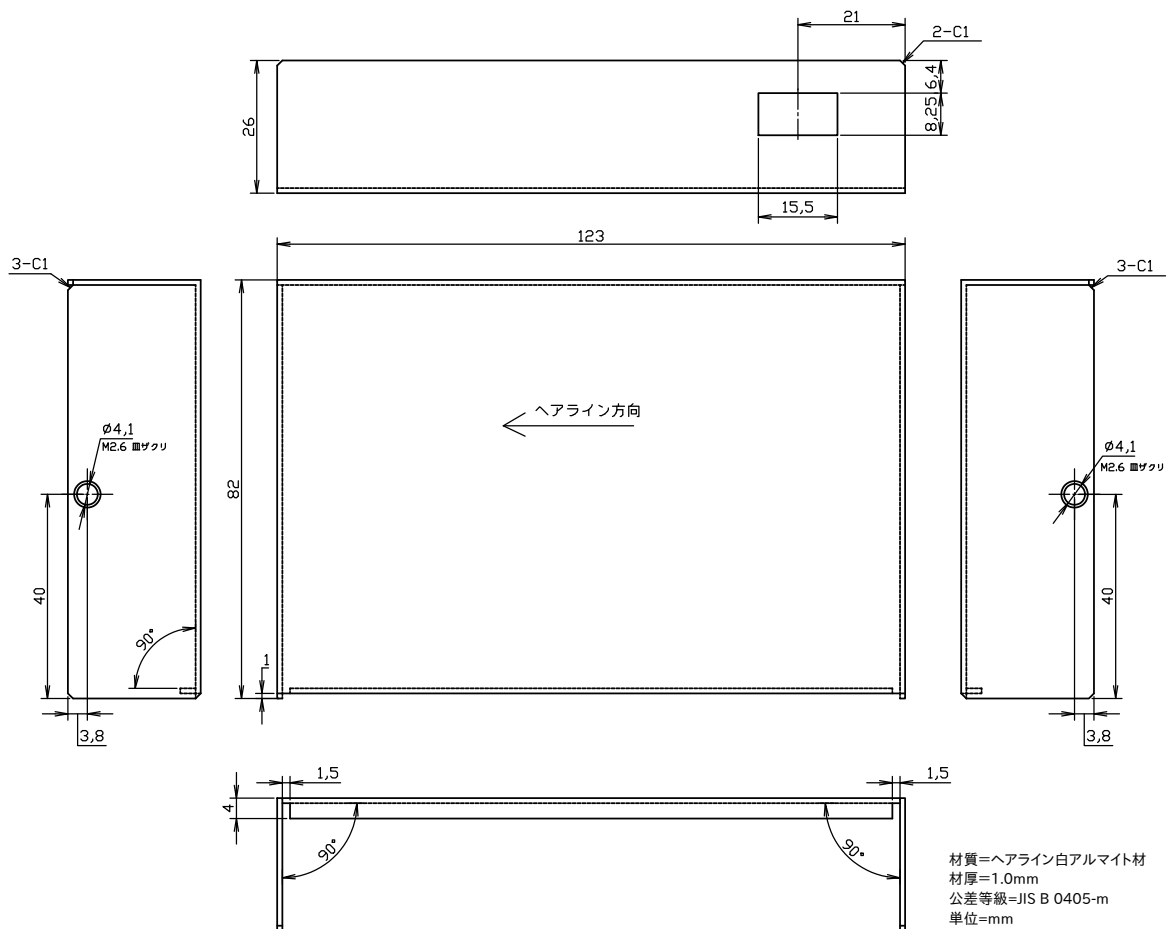


図 13.24 ケース(上)形状図

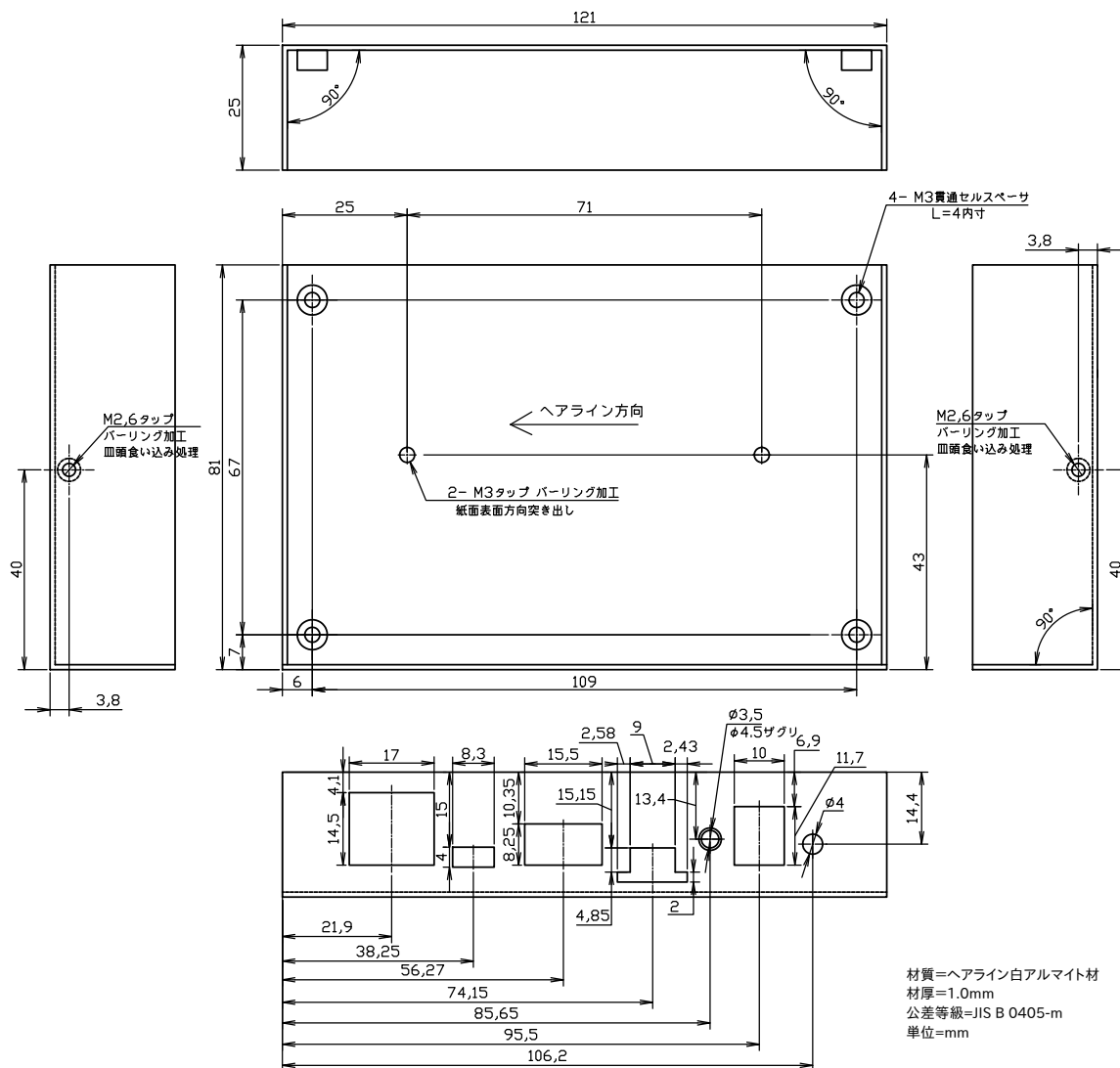


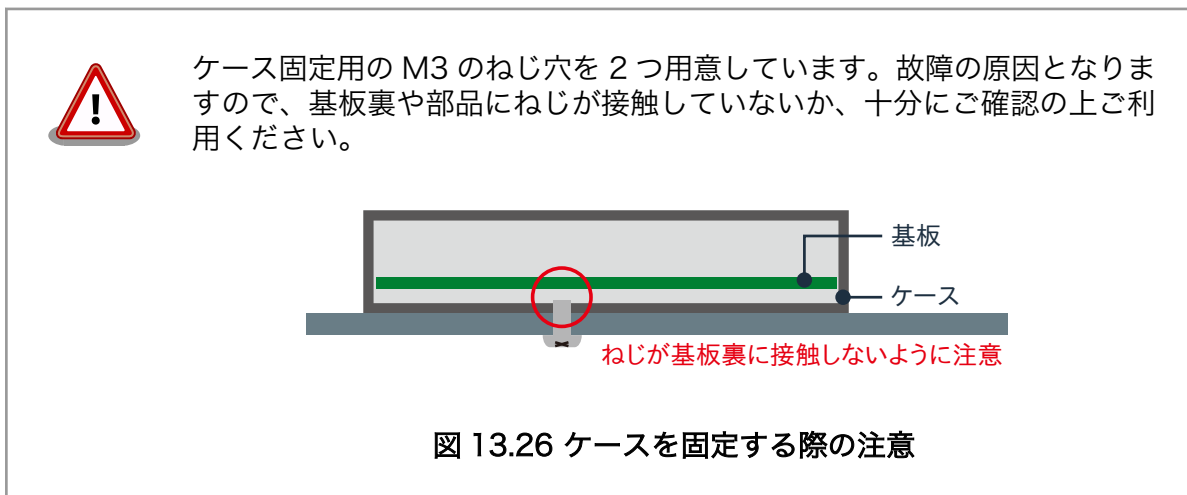
図 13.25 ケース(下)形状図



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。



DC ジャック (CON14)、USB コンソールインターフェース (CON6)、SD インターフェース (CON1) は、他コネクタの面位置より少し後ろに配置しているため、外部からの操作が不要な場合、開口を塞ぐ設計変更をするだけで、目隠しすることが可能です。



13.5.2. Armadillo-X2、G4 ケースモデル VESA 規格固定用プレート

13.5.2.1. 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。

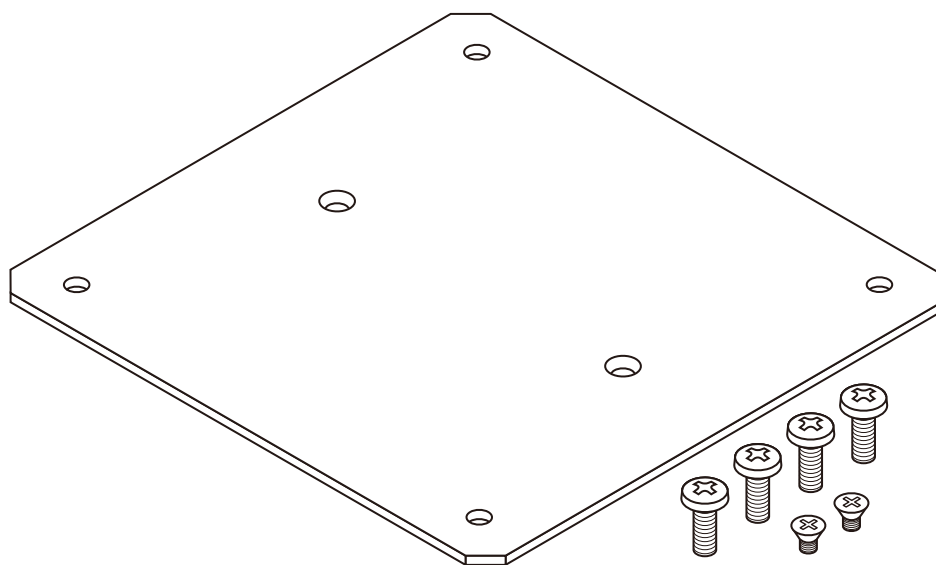


図 13.27 Armadillo-X2、G4 ケースモデル VESA 規格固定用プレート

表 13.34 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 13.35 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

13.5.2.2. 組み立て

Armadillo-X2 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。

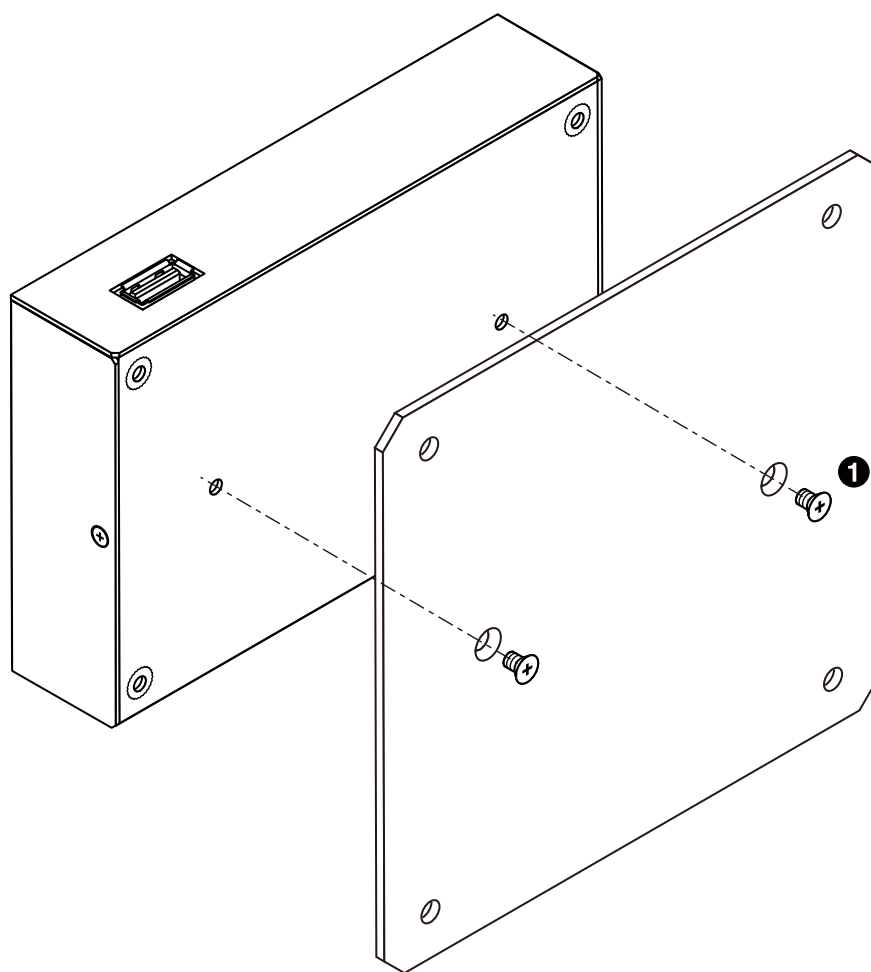


図 13.28 ケースに VESA 規格固定用プレートを取り付け

- ① 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。

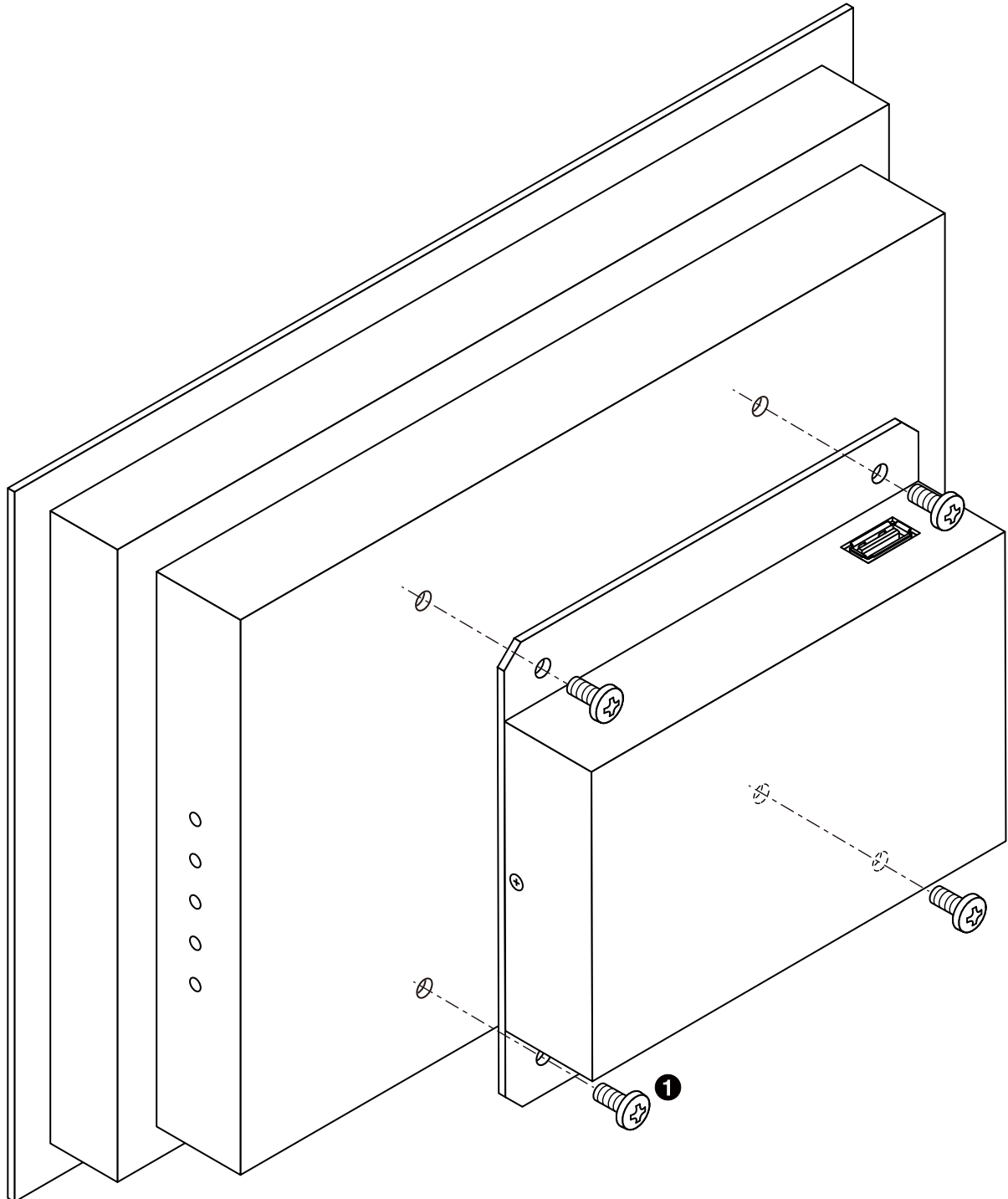


図 13.29 モニターに VESA 規格固定用プレートを取り付け

- ① バインド小ねじ(M4、L=10mm) × 4

13.5.2.3. 形状図

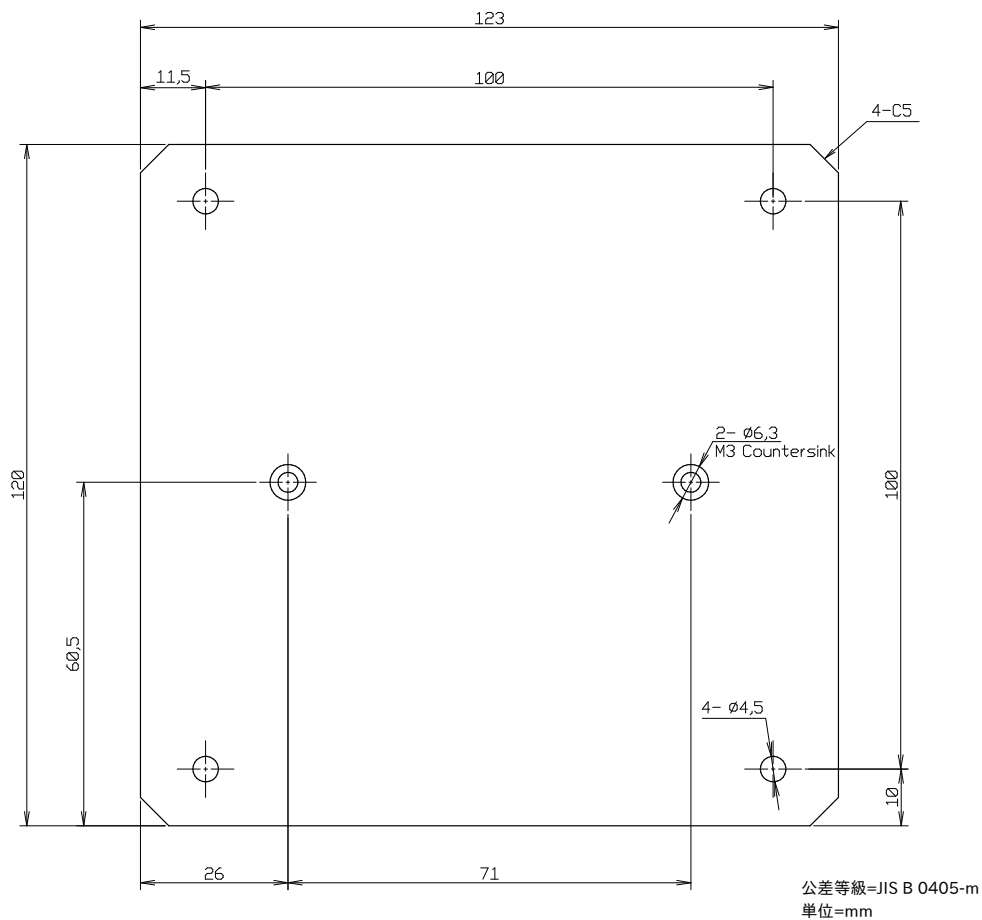


図 13.30 Armadillo-X2、G4 ケースモデル VESA 規格固定用プレート形状図



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。

改訂履歴

バージョン	年月日	改訂内容
1.0.0	2023/01/26	<ul style="list-style-type: none"> ・ 初版発行
1.1.0	2023/02/10	<ul style="list-style-type: none"> ・ 「13.5.2. Armadillo-X2、G4 ケースモデル VESA 規格固定用プレート」を追加 ・ 「13.5. オプション品」に CPU 放熱シートについての内容追加 ・ オプションケースの表記を正式名に修正 ・ 表記ゆれ修正
1.1.1	2023/02/27	<ul style="list-style-type: none"> ・ 「9.1. GUI アプリケーションを開発する」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 ・ 「9.7.2. 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 9.172. 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
1.2.0	2023/03/28	<ul style="list-style-type: none"> ・ 「7.1.2. IP アドレスの確認方法」を追加 ・ 「9.1. GUI アプリケーションを開発する」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 ・ 「9.2. CUI アプリケーションを開発する」を追加 ・ 「9.3. VSCode を使用して Armadillo のセットアップを行う」を追加 ・ 「9.5.1. コンテナの自動起動」にコンテナ起動後にホットプラグデバイスを確認させる方法を追加 ・ 「9.7.1. 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 ・ 「9.7.2. 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 ・ 「9.7.3. 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 9.12. u-boot の主要な環境変数」を追加 ・ 「9.16. 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
1.3.0	2023/04/26	<ul style="list-style-type: none"> ・ 「表 3.2. 仕様」の「カレンダー時計」に平均月差に関する説明を追加 ・ 「9.1. GUI アプリケーションを開発する」にサインページアプリケーションに関する説明を追加 ・ 「9.4.2.10. 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 ・ 「9.7.2. 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 ・ 「9.7.3. 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 ・ 「9.10.4. swupdate がエラーする場合の対処」を追加 ・ 「9.10.6.3. 例: swupdate_preserve_files で Linux カーネル以外の Armadillo-X2 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 ・ 誤記および分かりにくい表記の修正

