Armadillo-900 開発セット
製品マニュアル

A9900-U00D0

Version 1.1.0
2025/05/28

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-900 開発セット 製品マニュアル

株式会社アットマークテクノ

Version 1.1.0
2025/05/28

Armadillo-900 開発セット 製品マニュアル

目次

図目次

表目次

1. はじめに

このたびは Armadillo-900 開発セット をご利用いただき、ありがとうございます。

Armadillo-900 開発セット は、実装タイプの CPU モジュール「Armadillo-900」を搭載したマザーボードや製品を作るための、開発ボードといくつかの周辺機器一式をそろえた開発セットです。

付属の開発ボードには、組み込み機器・IoT 機器で良く用いられる、Ethernet、USB、LTE、WLAN や Bluetooth®、TH、CAN 等のインターフェースを搭載しており、回路図・部品表はすべて無償公開しています。無償公開の設計情報を参考に、お客様の欲しいインターフェース部分のみを抜き出して、オリジナルのマザーボードを早く・簡単に開発することができます。

「Armadillo-900」に搭載されている SoC「i.MX 8ULP」は、28nm FD-SOI プロセスで製造され、高い処理能力ながらも、省電力性に優れており間欠動作が可能です。

また、i.MX 8ULP には複数の異なるタイプの CPU コアが搭載されており(ヘテロジニアスマルチコア)、通信機能や GUI には Arm Cortex-A35 上で動作する Linux アプリケーションで、高速な周期処理やアナログ信号の計測には Arm Cortex-M33 上で動作する FreeRTOS アプリケーションで処理することが可能です。

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をコンテナとして管理できる機能を利用して、開発のベースとできる様々なコンテナを用意しています。様々なアプリケーションのベースとできる「Debian コンテナ」、Web ブラウザ上でビジュアルプログラミングが可能な「Node-RED™コンテナ」等があります。

Armadillo Base OS そのものがセキュリティ要件適合評価及びラベリング制度(JC-STAR)の★1 に適合しやすい構造であるため、ユーザーは工数を大幅に増やすことなくセキュリティ対応 IoT 機器を開発することが可能です。IoT 機器の運用管理クラウドサービス「Armadillo Twin」と組み合わせることで、現場に設置した IoT 機器の死活監視のほか、遠隔操作や、ソフトウェアのアップデート(OTA)も可能です。Armadillo Base OS はアップデートの状態を二面化しているので電源やネットワークの遮断によって中断してもアップデート前の状態に復旧します。

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

1.1. 本書について

1.1.1. 本書で扱うこと

本書では、Armadillo-900 開発セット の使い方、製品仕様(ソフトウェアおよびハードウェア)、オリジナルの製品を開発するために必要となる情報、その他注意事項について記載しています。Linux あるいは組み込み機器に不慣れな方でも読み進められるよう、コマンドの実行例なども記載しています。

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

1.1.2. 本書で扱わないこと

本書では、一般的な Linux のプログラミング、デバッグ方法やツールの扱い方、各種モジュールの詳細仕様など、一般的な情報や、他に詳しい情報があるものは扱いません。また、(Armadillo-900 開発セット を使用した)最終製品あるいはサービスに固有な情報や知識も含まれていません。

1.1.3. 本書で必要となる知識と想定する読者

本書は、読者として Armadillo-900 開発セット 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-900 開発セット 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-900 開発セット は組込みプラットフォームとして実績のある Armadillo をベースとしているため、標準で有効になっている機能以外にも様々な機能を実現することができます。

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

1.1.4. 本書の構成

本書には、Armadillo-900 開発セット をベースに、オリジナルの製品を開発するために必要となる情報を記載しています。また、取扱いに注意が必要な事柄についても説明しています。

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

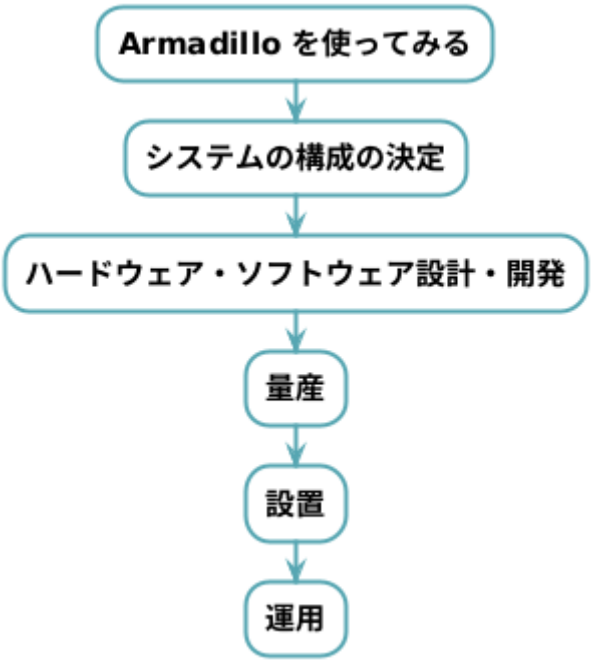


図 1.1 製品化までのロードマップ

- 「Armadillo を使ってみる」

はじめに Armadillo の使用方法を確認しつつ動作確認を行います。Linux コマンドやコンテナの使用方法、ネットワーク設定など Armadillo での開発に慣れることを目的に、開発時に必要になることが想定される知識について「3. 起動と終了」から「5. 動作確認方法」で紹介します。

- **「システム構成の決定」、「ハードウェア・ソフトウェア設計・開発」**

  システムが必要とする要件から使用するクラウド、デバイス、ソフトウェア仕様を決定および、ハードウェア・ソフトウェアの開発時に必要な情報について、「6. ハードウェアの設計情報」と「7. 開発編」で紹介します。

- **「量産」**

  開発完了後の製品を量産する方法について、「8. 量産編」で紹介します。

- **「設置」、「運用」**

  設置時の勘所や、量産した Armadillo を含めたハードウェアを設置し、運用する際に利用できる情報について、「9. 運用編」で紹介します。

また、本書についての概要を「1. はじめに」に、Armadillo-900 開発セットについての概要を「2. 製品概要」に、開発～運用までの一連の流れの中で説明しきれなかった機能についてを、「10. 応用編」で紹介します。

1.1.5. フォント

本書では以下のような意味でフォントを使いわけています。

表 1.1 使用しているフォント

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

1.1.6. コマンド入力例

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

表 1.2 表示プロンプトと実行環境の関係

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

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

表 1.3 コマンド入力例での省略表記

表記	説明
[VERSION]	ファイルのバージョン番号

1.1.7. アイコン

本書では以下のようにアイコンを使用しています。

注意事項を記載します。

役に立つ情報を記載します。

用語の説明や補足的な説明を記載します。

1.1.8. ユーザー限定コンテンツ

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

・ 各種信頼性試験データ・納入仕様書等製造関連情報

限定コンテンツを取得するには、「4.6. ユーザー登録」を参照してください。

1.1.9. 本書および関連ファイルのバージョンについて

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

Armadillo サイト - Armadillo-900 開発セット ドキュメントダウンロード

https://armadillo.atmark-techno.com/armadillo-900/resources/documents

Armadillo サイト - Armadillo-900 開発セット ソフトウェアダウンロード

https://armadillo.atmark-techno.com/armadillo-900/resources/software

1.2. 注意事項

1.2.1. 安全に関する注意事項

本製品を安全にご使用いただくために、特に以下の点にご注意ください。

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

を保持できなくなった場合には、直ちに電池を交換してください。そのまま使用すると、電池が漏液、発熱、破裂したり、ケガや製品の故障の原因となります。万一、漏れた液が身体に付着した場合は多量の水で洗い流してください。

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

1.2.2. 取扱い上の注意事項

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

破損しやすい箇所	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。
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, microSD) [3]以外へのコネクタ着脱は、絶対に行わないでください。
静電気	本製品には CMOS デバイスを使用しており、静電気により破壊されるおそれがあります。本製品を開封するときは、低湿度状態にならないよう注意し、静電防止用マットの使用、導電靴や人体アースなどによる作業者の帯電防止対策、備品の放電対策、静電気対策を施された環境下で行ってください。また、本製品を保管する際は、静電気を帯びやすいビニール袋やプラスチック容器などは避け、導電袋や導電性の容器・ラックなどに収納してください。
ラッチアップ	電源および入出力からの過大なノイズやサージ、電源電圧の急激な変動等により、使用している CMOS デバイスがラッチアップを起こす可能性があります。いったんラッチアップ状態となると、電源を切断しないかぎりこの状態が維持されるため、デバイスの破損につながることがあります。ノイズの影響を受けやすい入出力ラインには、保護回路を入れることや、ノイズ源となる装置と共通の電源を使用しない等の対策をとることをお勧めします。
衝撃	落下や衝撃などの強い振動を与えないでください。
使用場所の制限	無線機能を搭載した製品は、テレビ・ラジオに近接する場所で使用すると、受信障害を招く恐れがあります。
振動	振動が発生する環境では、Armadillo が動かないよう固定して使用してください。

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

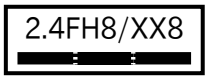
電池の取り扱い

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

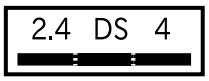
電波に関する注意事項(2.4GHz帯無線)

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

この無線機(LBES5PL2EL)の無線 LAN 機能は、2.4GHz 帯の周波数帯を使用します。全帯域を使用し、かつ移動体識別装置の帯域が回避可能です。変調方式として DS-SS および OFDM 方式を採用し、想定される与干渉距離は 40m 以下です。



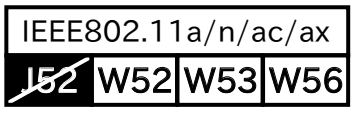
この無線機(LBES5PL2EL)の Bluetooth 機能は、2.4GHz 帯の周波数帯を使用します。全帯域を使用し、かつ移動体識別装置の帯域が回避不可です。変調方式として FH-SS 方式を採用し、想定される与干渉距離は 80m 以下です。



この無線機(LBES5PL2EL)の IEEE 802.15.4 機能は、2.4GHz 帯の周波数帯を使用します。全帯域を使用し、かつ移動体識別装置の帯域が回避不可です。変調方式として DS-SS 方式を採用し、想定される与干渉距離は 40m 以下です。

電波に関する注意事項(5GHz帯無線)

この無線機(LBES5PL2EL)は 5GHz 帯の周波数帯を使用します。



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

電波に関する注意事項(LTE)

この無線機(SIM7672G)は LTE 通信を行います。LTE 通信機能は、心臓ペースメーカーや除細動器等の植込み型医療機器の近く(15cm程度以内)で使用しないでください。

電気通信事業法に関する注意事項について

本製品の有線 LAN を、 電気通信事業者の通信回線(インターネットサービスプロバイダーが提供している通信網サービス等)に直接接続することはできません。 接続する場合は、 必ず電気通信事業法の認定を受けた端末設備(ルーター等)を経由して接続してください。

1.2.3. 製品の保管について

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

・ 保管環境として推奨する温度・湿度条件は以下のとおりです。

表 1.4 推奨温湿度環境について

推奨温湿度環境	5～35℃/70%RH 以下[a] [b]

[a]半田付け作業を考慮した保管温度範囲となっております。半田付けを行わない、または、すべての半田付けが完了している場合の推奨温度・湿度条件は、製品の動作温度・湿度範囲となります。
[b]温度変化の少ない場所に保管してください。保管時の急激な温度変化は結露が生じ、金属部の酸化、腐食などが発生し、はんだ濡れ性に影響が出る場合があります。

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

1.2.4. ソフトウェア使用に関しての注意事項

本製品に含まれるソフトウェアについて

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

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

以下のソフトウェアは、オープンソースソフトウェアではありません。

ボード情報取得ツール(get-board-info)

パスワードの設定について

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

基本的には初回ログイン時または、専用の開発環境での開発開始時に機器のパスワードの再設定を求められます。この時、推測されやすい単純なパスワードを使用すると外部からの不正アクセスや機器の乗っ取りの原因になりますので、必ず推測されにくい複雑なパスワードを設定してください。パスワードの設定については、「3.3.4. ログイン」や「7.1.3. Armadillo に初期設定をインストールする」、「5.1.3. ABOS Web のパスワード登録」を参照してください。

ソフトウェアのアップデートについて

弊社は Armadillo のソフトウェアのアップデートを定期的に実施します。詳細は「2.1.5. Armadillo Base OS のメンテナンスポリシーとアップデートの推奨」を参照してください。

本製品のソフトウェアは常に最新版にアップデートした上でご使用ください。本製品を購入後、開発前に最新版のソフトウェアをインストールする方法については「3.2. Armadillo の初期化と ABOS のアップデート」を、運用時に本製品をアップデートする方法については「9.3. Armadillo のソフトウェアをアップデートする」を参照してください。

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

1.2.5. 本製品を廃棄する場合について

本製品を廃棄する場合は、必ず「9.6. Armadillo を廃棄する」に記載の注意事項をご一読の上、機器内に保存されているログや個人情報などのデータを正しく全て削除した上で廃棄してください。

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

1.2.6. 電波障害について

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

1.2.7. 無線モジュールの安全規制について

本製品に搭載されている LTE モジュール は、 電気通信事業法に基づく設計認証を受けています。

また、本製品に搭載されている LTE モジュール SIM7672G 、WLAN+BT+TH コンボモジュール LBES5PL2EL は、 電波法に基づく工事設計認証を受けています。

これらの無線モジュールを国内で使用するときに無線局の免許は必要ありません。

以下の事項を行うと法律により罰せられることがあります。

・ 無線モジュールやアンテナを分解/改造すること。

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

認証番号は次のとおりです。

表 1.5 LTE モジュール:SIM7672G 適合証明情報

項目	内容
型式又は名称	SIM7672G
電波法に基づく工事設計認証における認証番号	201-230741
電気通信事業法に基づく設計認証における認証番号	D230149201

図 1.2 LTE モジュール:SIM7672G 認証マーク

表 1.6 WLAN+BT+TH コンボモジュール:LBES5PL2EL 適合証明情報

項目	内容
型式又は名称	LBES5PL2EL
電波法に基づく工事設計認証における認証番号	001-P02018

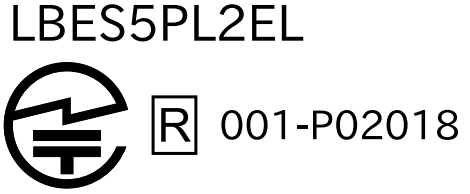


図 1.3 WLAN+BT+TH コンボモジュール:LBES5PL2EL 認証マーク

1.2.8. LED について

本製品に搭載されている LED は部品の特性上、LED ごとに色味や輝度の差が発生する場合がありますので、あらかじめご了承ください。

1.2.9. 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、ご購入から標準で 1 年間の交換保証を行っております。添付品およびソフトウェアは保証対象外となりますのでご注意ください。

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

製品保証サービス https://armadillo.atmark-techno.com/support/warranty

製品保証規定 https://armadillo.atmark-techno.com/support/warranty/policy

1.2.10. 輸出について

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

1.2.11. 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。

- Bluetooth®のワードマークおよびロゴは、Bluetooth SIG, Inc.が所有する登録商標です。

1.3. 謝辞

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

2. 製品概要

2.1. 製品の特長

2.1.1. Armadillo とは

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

- Arm プロセッサ搭載・省電力設計

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

- 小型・手のひらサイズ

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

- 標準 OS として Linux をプリインストール

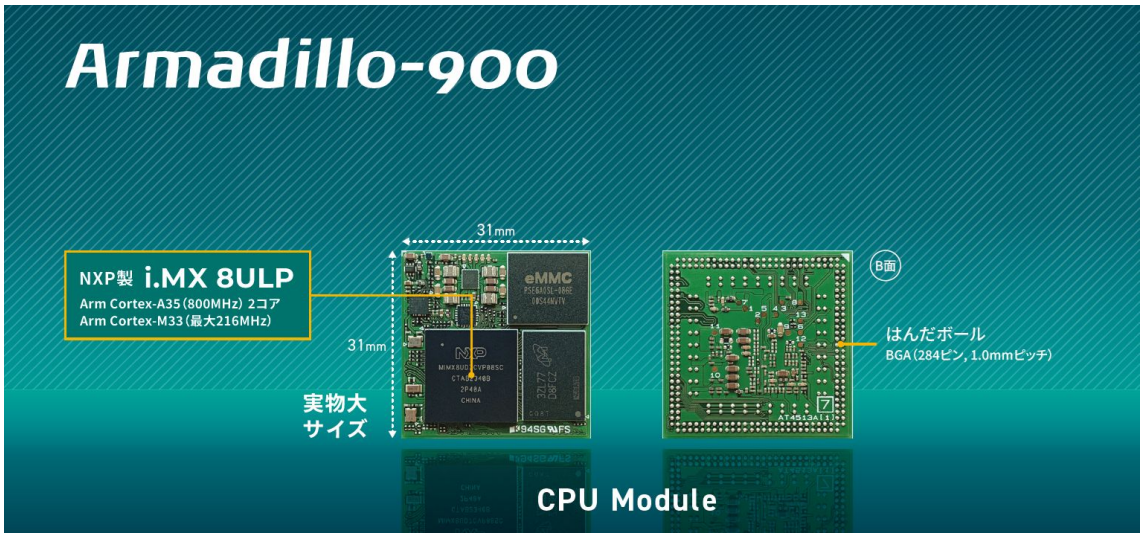
  標準 OS に Linux を採用しており、豊富なソフトウェア資産と実績のある安定性を提供します。ソースコードをオープンソースとして公開しています。

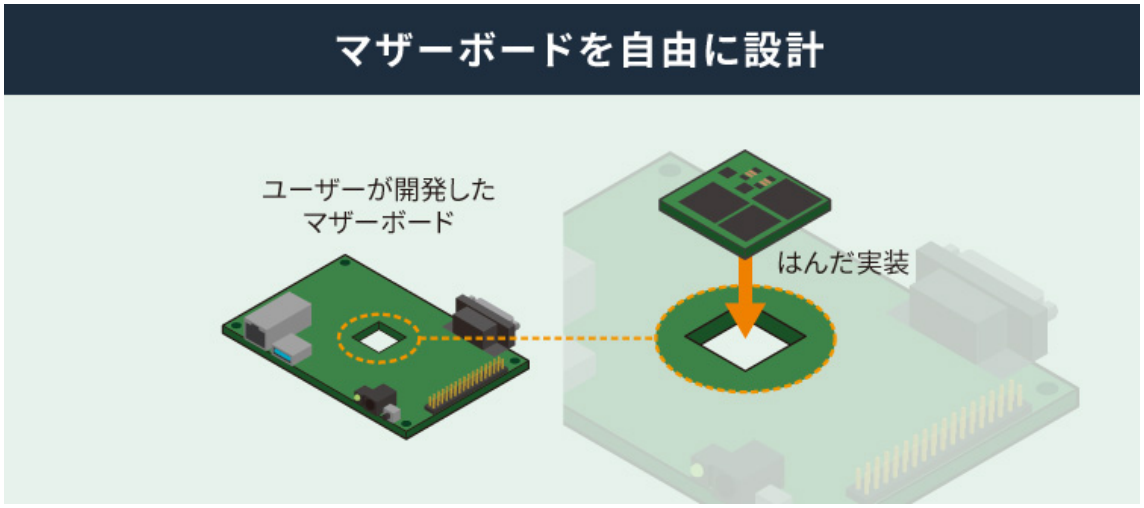
- 開発環境

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

2.1.2. Armadillo-900 とは

「Armadillo-900」は実装型の CPU モジュールで、31mm 角のサイズに、SoC、メモリ、電源回路など IoT 機器として主要な機能が高密度に集積されています。IoT 機器のハードウェア設計者は、「Armadillo-900」を利用することで SoC 周りの煩雑で高度な設計が不要となり、要求仕様に合わせたインターフェース部分(マザーボード)の開発に特化することができます。また、搭載されている Armadillo Base OS はセキュリティ機能が充実しているため、セキュアな IoT 製品を短期間に開発・製造することが可能です。





Armadillo-900 には以下の特長があります。

- i.MX 8ULP 搭載 省電力なシステム設計が可能

  搭載する SoC「i.MX 8ULP」は、28nm FD-SOI プロセスで製造され、高い処理能力ながらも、省電力性に優れています。多彩な電力コントロール機能を持つことも 1 つの特長です。Shutdown、Deep Sleep、Sleep [1]、Active の 4 つの動作モードを提供し、これらの動作モードを組み合わせることで間欠動作を実現し、バッテリーを搭載したモバイル端末への適用も可能です。

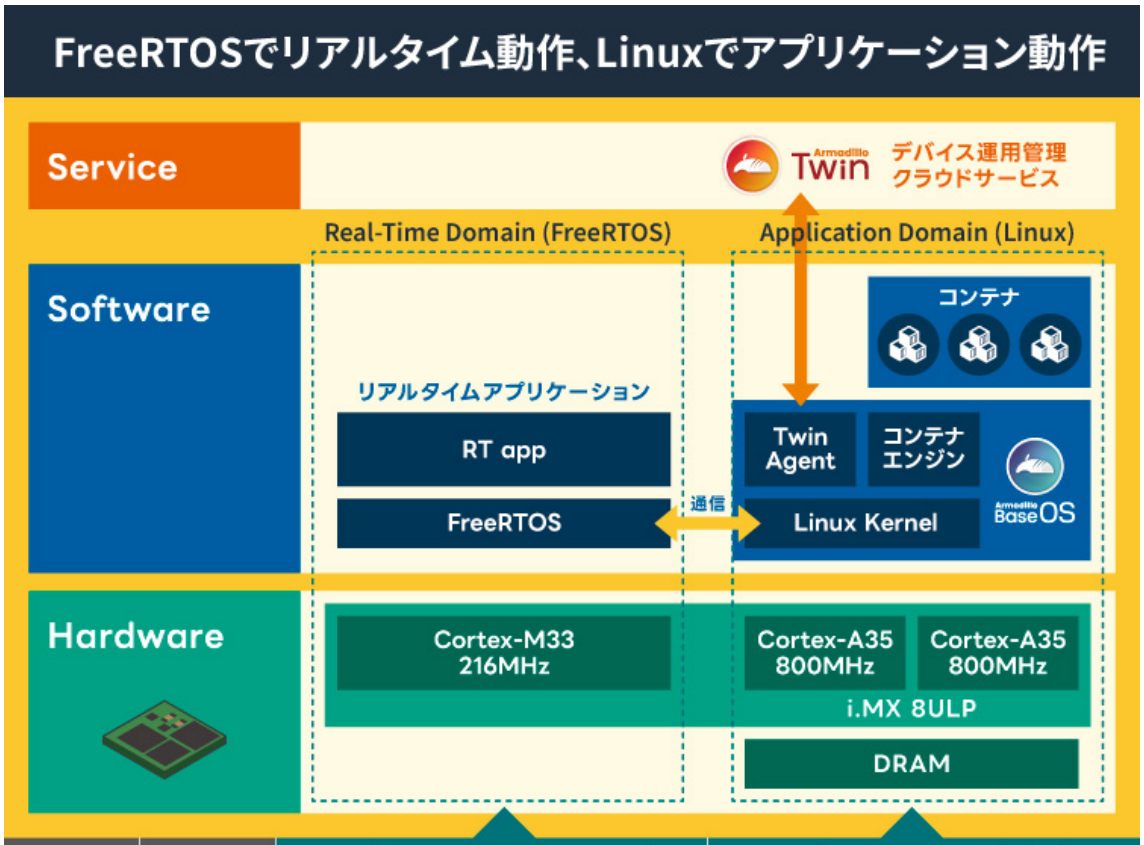
- ヘテロジニアスマルチコアを活用した Linux+FreeRTOS 環境で柔軟なシステム開発

  i.MX 8ULP には複数の異なるタイプの CPU コアが搭載されており(ヘテロジニアスマルチコア)、通信機能や GUI には Arm Cortex-A35 上で動作する Linux アプリケーションで、高速な周期処理やアナログ信号の計測には Arm Cortex-M33 上で動作する FreeRTOS アプリケーションで処理することが可能です。多様な要求仕様へ柔軟に対応し、IoT 機器のシステム開発を実現します。

- コンテナ型でセキュアな OS、JC-STAR★1 適合の IoT 機器も簡単に開発

[1]Sleep モードの使用方法については現時点で本マニュアルには記載しておりません。使い方の説明が必要な場合には Armadillo フォーラムにてお問い合わせください。

Armadillo Base OS は IoT 機器向けの Linux ベースのコンテナアーキテクチャの OS です。Armadillo Base OS そのものがセキュリティ要件適合評価及びラベリング制度(JC-STAR)の★1 に適合しやすい構造であるため、ユーザーは工数を大幅に増やすことなくセキュリティ対応 IoT 機器を開発することが可能です。IoT 機器の運用管理クラウドサービス「Armadillo Twin」と組み合わせることで、現場に設置した IoT 機器の死活監視のほか、遠隔操作や、ソフトウェアのアップデート(OTA)も可能です。



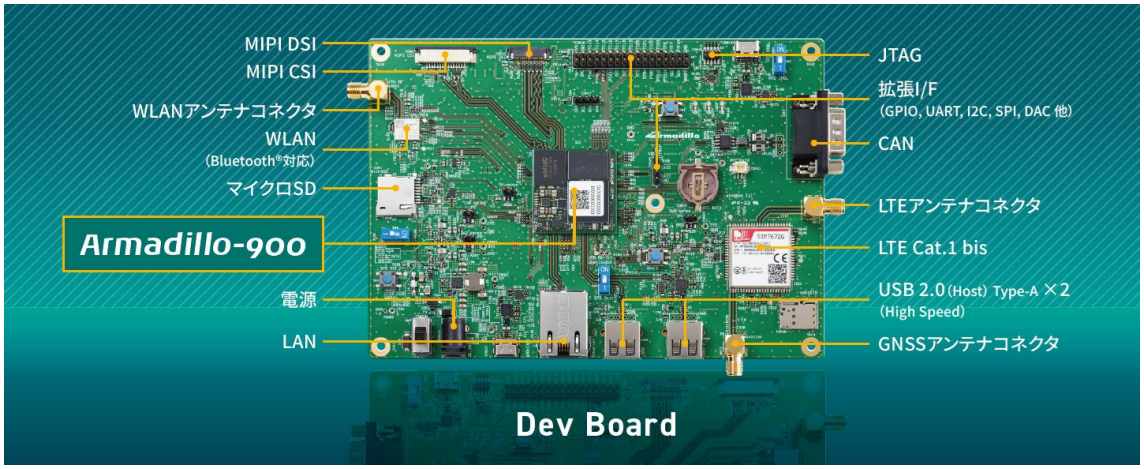
動作モード	消費電力	Real-Time Domain (FreeRTOS)			Application Domain (Linux)		
		状態	起床要因	起床時間	状態	起床要因	起床時間
Active	720mW～	動作※1 160MHz	―	―	動作	―	―
Sleep	60mW	動作 38.4MHz	―	―	待機	I/Fアクセス RTDからの要求	～1秒
Deep Sleep	6mW	待機	I/Fアクセス	～1秒	待機	I/Fアクセス	～1秒
Shutdown	0.05mW	停止	PMIC ONOFF (RTC, SW等)	～1秒※2	停止	PMIC ONOFF (RTC, SW等)	約12秒

2.1.3. Armadillo-900 開発セットとは

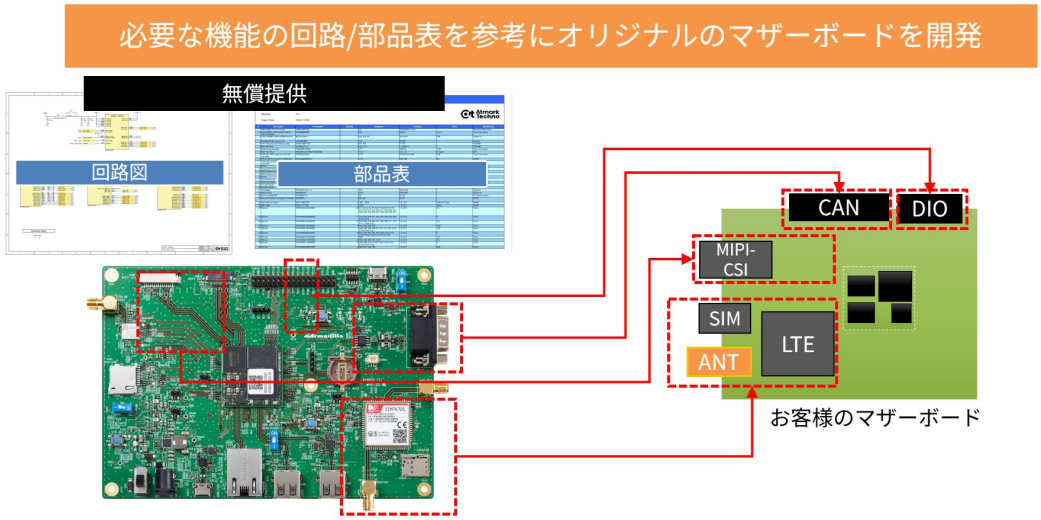
Armadillo-900 開発セットは、Armadillo-900 の評価やオリジナルのマザーボードの開発を行なう為の評価・開発セットです。

Armadillo-900 を実装し、組み込み機器で良く使用される様々なインターフェースや機能を搭載した開発ボードと開発ボードを動作させる為の、いくつかのオプション品が一式のセットになっています。

マザーボードの試作が出来上がるまでの間、インターフェースの動作確認やソフトウェアの開発・テストを進める事ができます。



また、開発ボードの回路図や部品表などの設計情報はアットマークテクノの Web サイトに無償公開しており、必要な機能の回路図や部品を参考にすることでオリジナルのマザーボードを早く・簡単に開発することができます。



2.1.4. Armadillo Base OS とは

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

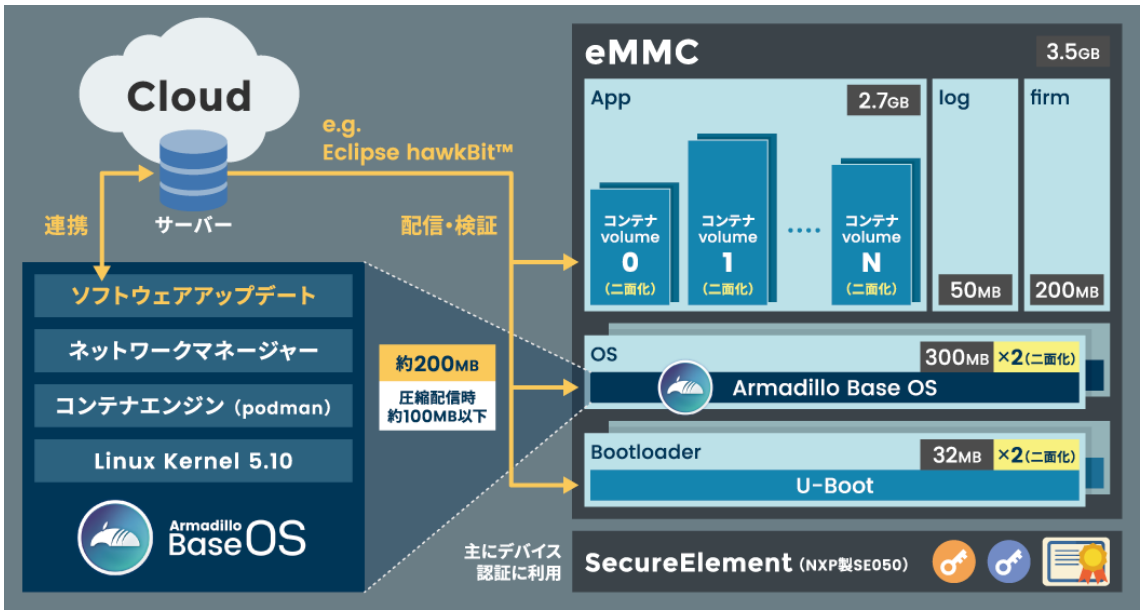


図 2.1 Armadillo Base OS とは

- OS のコンパクト化

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

- コンテナによるアプリケーション運用

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

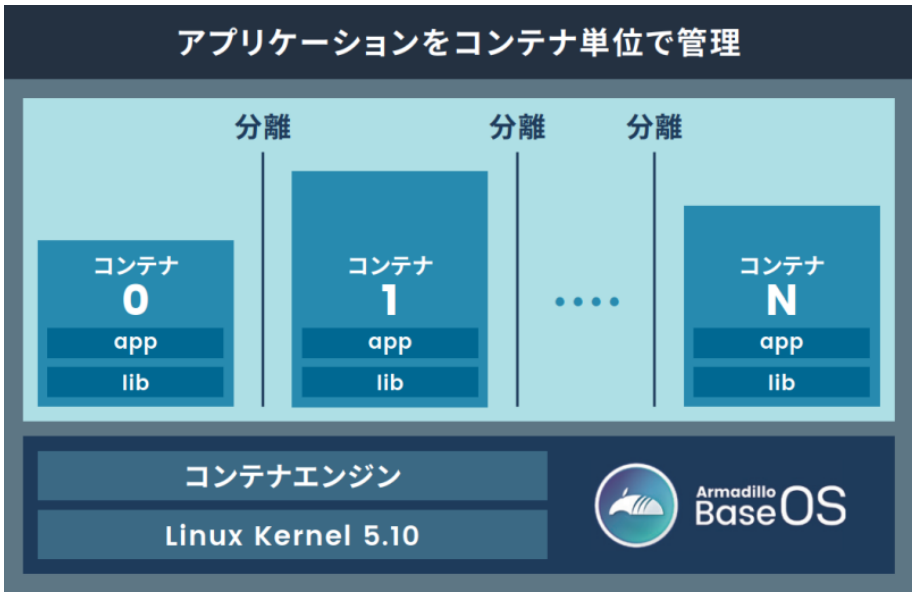


図 2.2 コンテナによるアプリケーションの運用

- アップデート機能を標準搭載

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

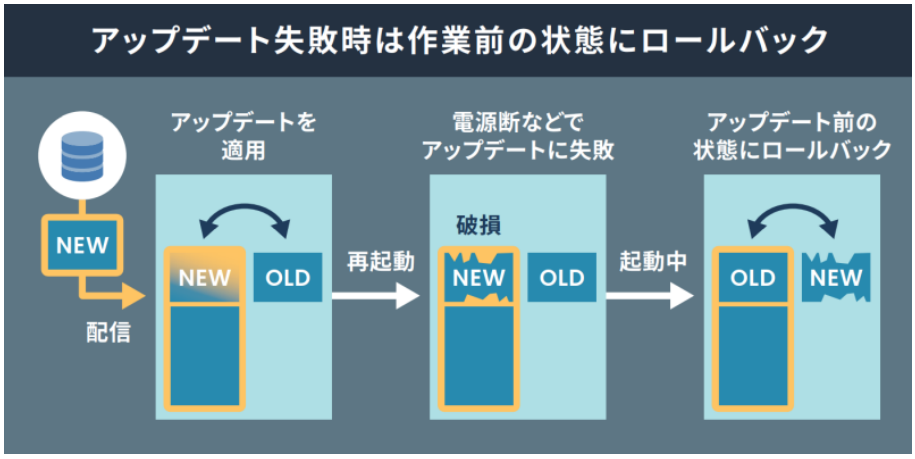


図 2.3 ロールバックの仕組み

- 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[2][3]」を利用可能な状態で提供しています。

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

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

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

> **Armadillo サイト 製品アップデート**
>
> https://armadillo.atmark-techno.com/news/software-updates

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

2.1.5.1. 後方互換性について

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

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

製品開発を開始するにあたり、Armadillo Base OS に関してより詳細な情報が必要な場合は、「7.3. 開発前に知っておくべき Armadillo Base OS の機能・特徴」 を参照してください。

2.1.6. Armadillo Twin とは

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

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

図 2.4 Armadillo Twin　とは

2.1.6.1. サービスの特徴

- ソフトウェアアップデート (OTA)

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

- 遠隔稼働監視

  登録されたデバイスの死活監視をはじめ、CPU の使用率や温度、メモリの使用量、モバイル回線の電波状況、ストレージの空き容量や寿命を監視することができます。各値にはアラートの設定を

行うことができ、異常を検知した場合はアラートメールを管理者に送信します。メールを受けた管理者は本サービスの遠隔操作機能を利用し、即座に対応を行うことができるため、システムの安定運用を行うことができます。そのほか、本サービスに登録したデバイスは、自由にラベル名を付けたりグループを作成して管理することができるため、どのデバイスをどの場所に設置したか画面上で把握することが容易になります。また、デバイス本体に搭載されているセキュアエレメントを利用した個体認証により、不正なデバイスの登録を防ぎます。

- 遠隔操作

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

2.1.6.2. 提供する機能一覧

Armadillo Twin は、下記の機能を提供します。

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

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

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

2.2. 製品ラインアップ

2.2.1. Armadillo-900 開発セット

Armadillo-900 量産ボードを搭載した開発セットです。

表 2.1 Armadillo-900 開発セット ラインアップ

名称	型番
Armadillo-900 開発セット	A9900-U00D0

内容物は以下のとおりです。

図 2.5 Armadillo-900 開発セットの内容物

❶ Armadillo-900 開発セット 本体

❷ LTE 用外付けアンテナ

❸ WLAN/BT/TH 用外付けアンテナ

❹ USB(A オス-Type C)ケーブル

❺ AC アダプタ(12V/2.0A)

❻ 基板固定用スペーサー(8 個)

2.2.2. Armadillo-900 量産ボード

Armadillo-900 開発セットに搭載されている小型 CPU モジュールです。Armadillo-900 量産ボードの詳細については、今後公開する製品マニュアルを参照してください。

表 2.2 Armadillo-900 ラインアップ

名称	型番
Armadillo-900 量産ボード	A9000-U00Z

2.3. 仕様

Armadillo-900 開発セットの主な仕様は以下のとおりです。

表 2.3 仕様

型番	A9900-U00D0
プロセッサ	NXP Semiconductors i.MX 8ULP ARM Cortex-A35 x 2 ・命令/データキャッシュ 32KByte/32KByte ・L2 キャッシュ 512KByte ・内部 SRAM 64KByte ・メディアプロセッシングエンジン(NEON)搭載 ARM Cortex-M33 x 1 ・命令/データキャッシュ 32KByte/32KByte ・内部共有メモリ 768KByte
システムクロック	CPU コアクロック(ARM Cortex-A35): 800MHz CPU コアクロック(ARM Cortex-M33): 216MHz DDR クロック: 528MHz 源発振クロック: 32.768kHz, 24MHz
RAM	LPDDR4x: 1GByte バス幅: 32bit
ROM	eMMC: 3.8GB(3.6GiB) [a]
LAN(Ethernet)	RJ45 x 1 100BASE-TX/10BASE-T, AUTO-MDIX 対応
無線 LAN/Bluetooth®/IEEE 802.15.4	WLAN+BT+TH コンボモジュール [b] IEEE 802.11a/b/g/n/ac/ax and Bluetooth, IEEE 802.15.4
モバイル通信	LTE Cat.1 bis モジュール [c] 2 キャリア（NTT ドコモ、KDDI）対応 SIM スロット: nanoSIM 対応
USB	USB 2.0 (Host) Type A x 2 (High Speed)
SD	microSD スロット x 1
カレンダ時計	リアルタイムクロック搭載 バックアップ用電池 CR1220 接続可能 最大月差: 8 秒(経年変化除く)
カメラ	MIPI CSI-2 (2 レーン) FFC コネクタ (15 ピン/1.0mm ピッチ) x 1
ビデオ	MIPI DSI (4 レーン) FFC コネクタ (22 ピン/0.5mm ピッチ) x 1
CAN	D-Sub 9 ピン x 1
スイッチ	ユーザースイッチ x 1 ONOFF 信号制御用スイッチ x 1 リセットスイッチ x 1 ブートモード切替スイッチ x 1 電源スイッチ x 1 RTD 用コンソール JTAG 切替スイッチ x 1
LED	SYS(Green) x 1 APP(Green) x 1 WWAN(Green) x 1
拡張インターフェース[d]	GPIO x 26、SPI x 1、UART x 3、I2C x 2
メンテナンスポート	APD 用コンソール USB Type C x 1 RTD 用コンソール USB Type C x 1 [e] JTAG ピンヘッダ(10 ピン) x 1 [e]
セキュアエレメント	NXP Semiconductors SE050
入力電源	DC 8～26.4V
動作温度範囲	+10～+40℃ (結露なきこと)
外形サイズ(基板)	170 x 120 mm (突起部、アンテナを除く)

[a]pSLC での数値です。出荷時 pSLC に設定しています。
[b]無線 LAN/Bluetooth/IEEE 802.15.4 通信を利用する時は、WLAN/BT/TH 用外付けアンテナを接続する必要があります。
[c]モバイル通信を利用する時は、LTE 用外付けアンテナを接続する必要があります。

[d]i.MX 8ULP のピンマルチプクレスの設定で、優先的に機能を割り当てた場合に拡張可能な最大数を記載しています。
[e]RTD 用コンソールと JTAG は排他で切替スイッチで切り替えます。

2.4. 外観

Armadillo-900 開発セットの外観は以下のとおりです。

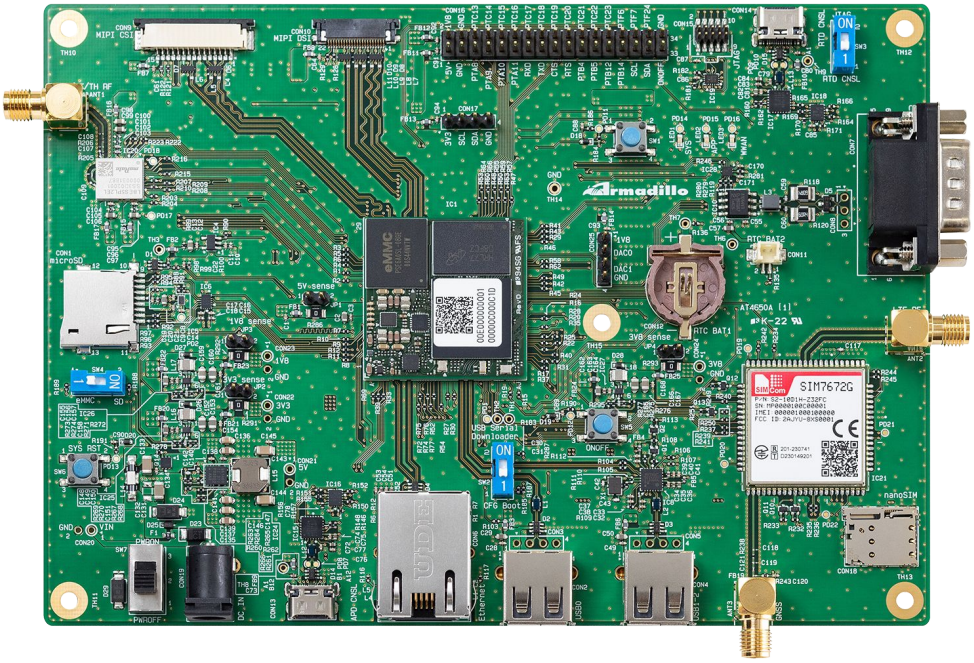


図 2.6 Armadillo-900 開発セット 外観(上面)

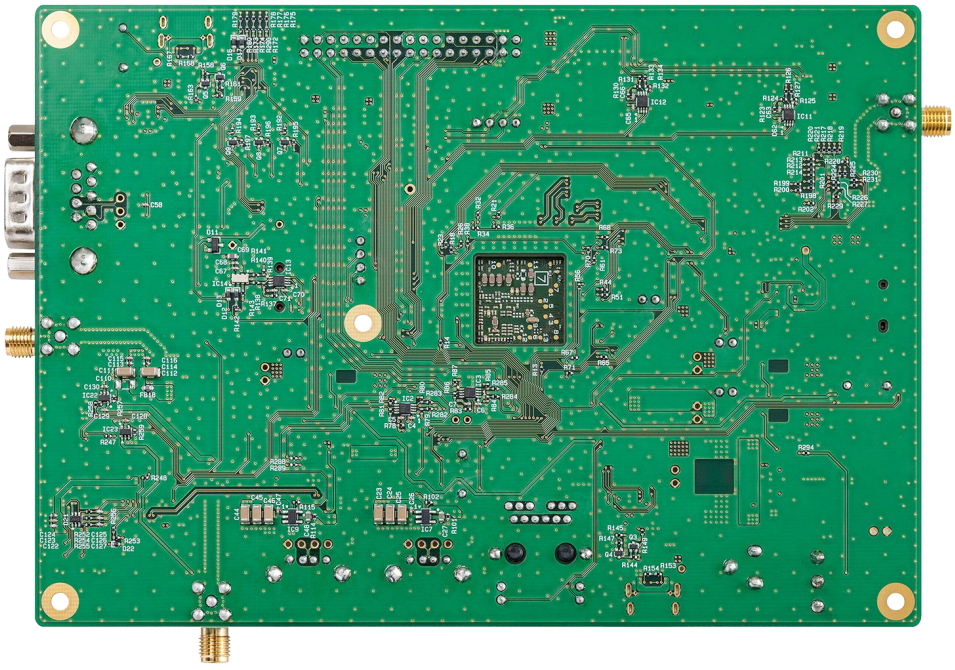


図 2.7 Armadillo-900 開発セット 外観(下面)

2.5. インターフェースレイアウト

Armadillo-900 開発セット のインターフェースレイアウトです。

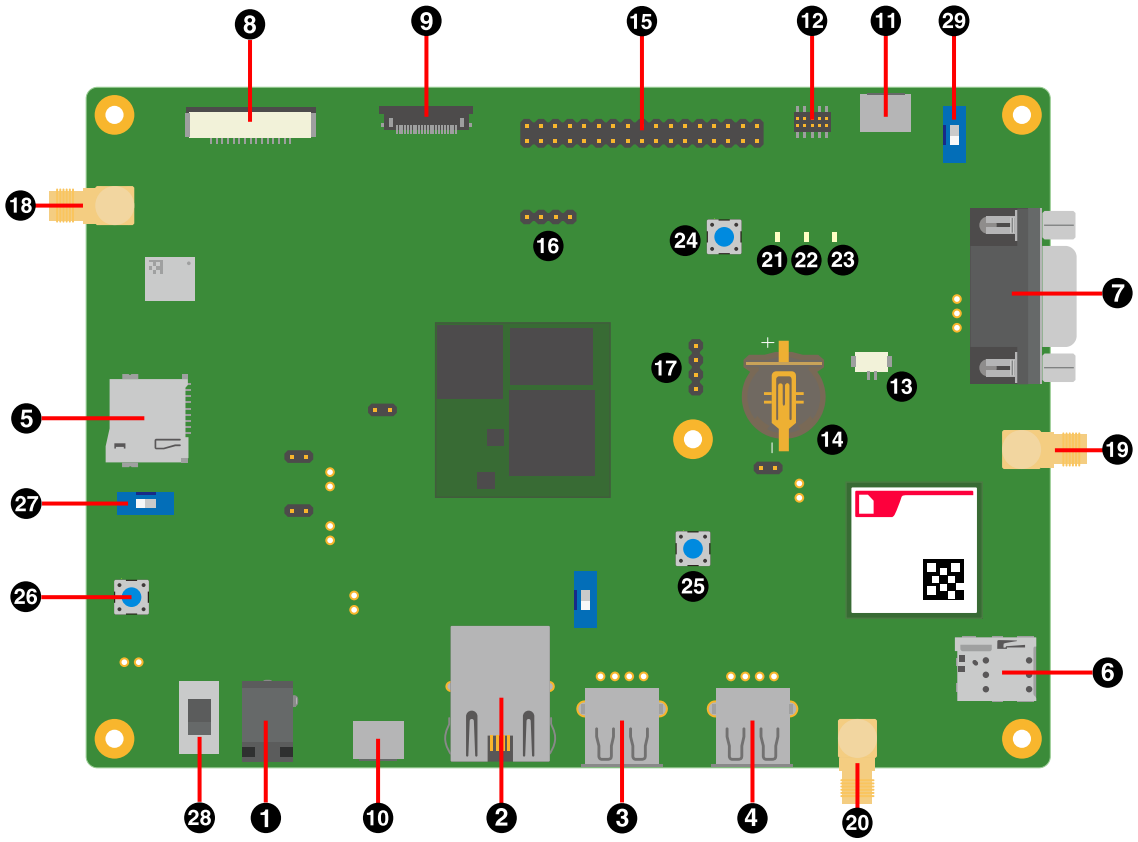


図 2.8 Armadillo-900 開発セット インターフェースレイアウト

表 2.4 各部名称と機能

番号	名称	形状	説明
1	電源入力インターフェース	DC ジャック	Armadillo-900 開発セットへの電源供給で使用します。付属の AC アダプタ(12V/2A)を接続します。対応プラグ:内径 2.1mm、外形 5.5mm
2	LAN インターフェース	RJ45 コネクタ	有線 LAN を利用する場合に使用します。LAN ケーブルを接続します。
3	USB インターフェース 1	USB 2.0 Type A コネクタ	USB デバイスを利用する場合に使用します。USB メモリ等を接続します。
4	USB インターフェース 2	USB 2.0 Type A コネクタ	USB デバイスを利用する場合に使用します。USB メモリ等を接続します。
5	SD インターフェース	microSD スロット	外部ストレージが必要な場合や、ブートローダーを破壊してしまった時の復旧等で使用します。microSD カードを挿入します。
6	nanoSIM インターフェース	nanoSIM スロット	LTE データ通信を利用する場合に使用します。nanoSIM カードを挿入します。
7	CAN インターフェース	D-Sub(9 ピン)コネクタ	CAN 通信を利用する場合に使用します。
8	MIPI CSI インターフェース	FFC コネクタ(15 ピン/1.0mm ピッチ)	MIPI CSI カメラを利用する場合に使用します。FFC(15 ピン/1.0mm ピッチ)を接続します。

番号	名称	形状	説明
9	MIPI DSI インターフェース	FFC コネクタ(22 ピン/0.5mm ピッチ)	MIPI DSI ディスプレイを利用する場合に使用します。FFC(22 ピン/0.5mm ピッチ)を接続します。
10	APD 用コンソールインターフェース	USB Type C コネクタ	アプリケーションドメイン用コンソール入出力を利用する場合に使用します。USB Type C ケーブルを接続します。
11	RTD 用コンソールインターフェース	USB Type C コネクタ	リアルタイムドメイン用コンソール入出力を利用する場合に使用します。USB Type C ケーブルを接続します。
12	JTAG インターフェース	ピンヘッダ 10 ピン(1.27mm ピッチ)	JTAG を利用する場合に使用します。
13	RTC バックアップインターフェース 1	2 ピンコネクタ	リアルタイムクロックのバックアップ給電が必要な場合に使用します。
14	RTC バックアップインターフェース 2	電池ボックス	リアルタイムクロックのバックアップ給電が必要な場合に使用します。対応電池: CR1220 等
15	拡張インターフェース	ピンヘッダ 34 ピン(2.54mm ピッチ)	機能拡張する場合に使用します。
16	I2C(3.3V)インターフェース	ピンヘッダ 4 ピン(2.54mm ピッチ)	3.3V の I2C を利用する場合に使用します。
17	DAC インターフェース	ピンヘッダ 4 ピン(2.54mm ピッチ)	アナログ電圧出力を利用する場合に使用します。
18	WLAN/BT/TH アンテナインターフェース	SMA コネクタ	WLAN、Bluetooth、IEEE 802.15.4 データ通信を利用する場合に使用します。付属の WLAN/BT/TH 用外付けアンテナを接続します。
19	LTE アンテナインターフェース	SMA コネクタ	LTE データ通信を利用する場合に使用します。付属の LTE 用外付けアンテナを接続します。
20	GNSS アンテナインターフェース	SMA コネクタ	GNSS を利用する場合に使用します。
21	システム LED	LED(緑色、面実装)	電源の入力状態を表示する緑色 LED です。
22	アプリケーション LED	LED(緑色、面実装)	アプリケーションの状態を表示する緑色 LED です。
23	ワイヤレス WAN	LED(緑色、面実装)	LTE 通信の状態を表示する緑色 LED です。
24	ユーザースイッチ	タクトスイッチ	ユーザーが利用可能なスイッチです。
25	ONOFF 信号制御用スイッチ	タクトスイッチ	ONOFF 信号を制御するために使用します。
26	リセットスイッチ	タクトスイッチ	リセット信号を制御するために使用します。
27	ブートモード切替スイッチ	DIP スイッチ	ブートモードを切り替えるために使用します。
28	電源スイッチ	スライドスイッチ	電源を ON/OFF するために使用します。
29	RTD 用コンソール JTAG 切替スイッチ	DIP スイッチ	RTD 用コンソールと JTAG を切り替えるために使用します。

2.6. ブロック図

Armadillo-900 開発セットのブロック図は次のとおりです。

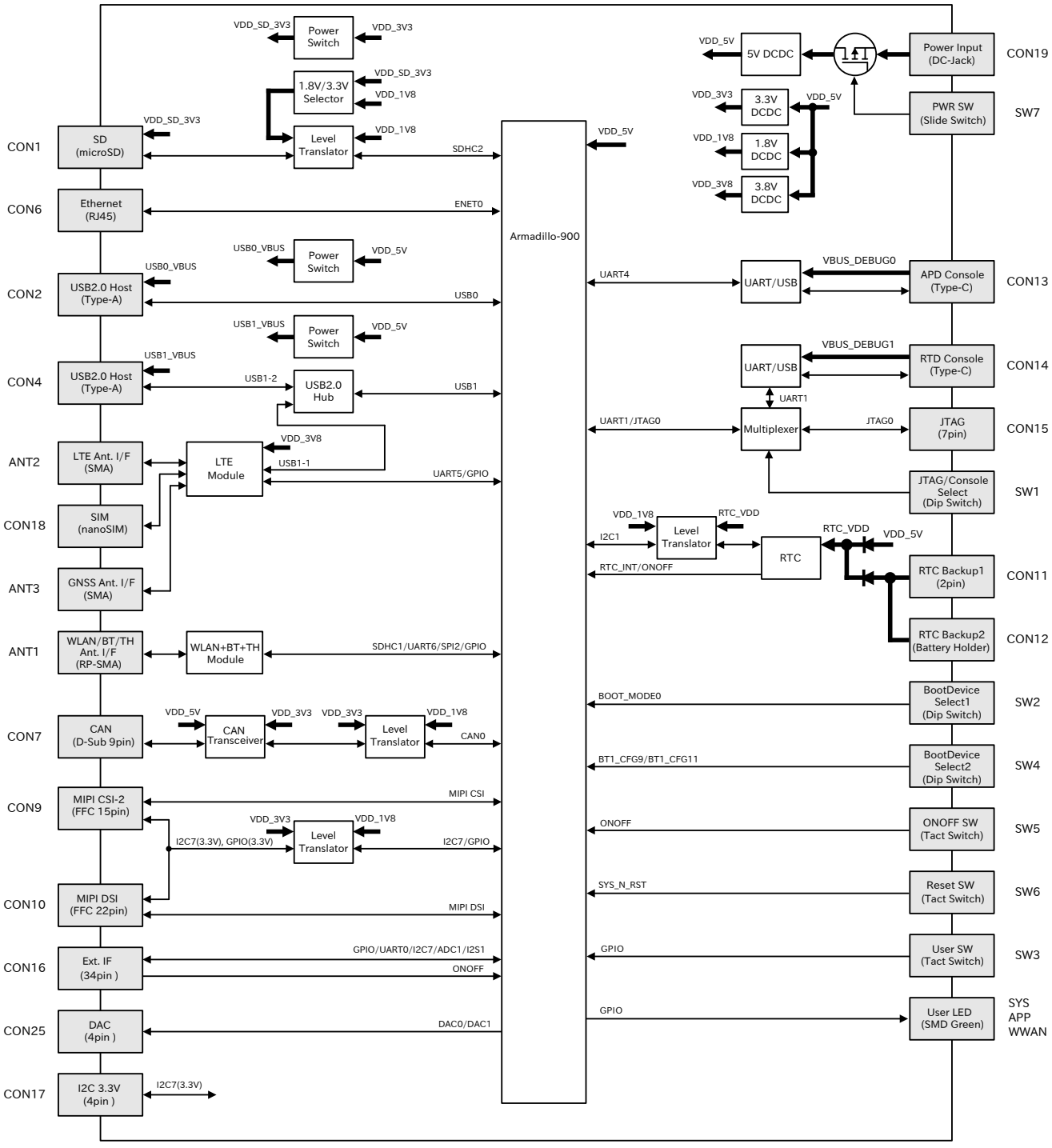


図 2.9 ブロック図

2.7. 使用可能なストレージデバイス

Armadillo-900 開発セット でストレージとして使用可能なデバイスを次に示します。

表 2.5 ストレージデバイス

デバイス種類	ディスクデバイス	先頭パーティション	インターフェース
オンボード eMMC	/dev/mmcblk0	/dev/mmcblk0p1	オンボード
オンボード eMMC (GPP)	/dev/mmcblk0gp3	なし	オンボード
SD/SDHC/SDXC カード	/dev/mmcblk2	/dev/mmcblk2p1	microSD スロット(CON1)
USB メモリ	/dev/sd* [a]	/dev/sd*1	USB ホストインターフェース (CON2)
USB メモリ	/dev/sd* [b]	/dev/sd*1	USB ホストインターフェース (CON4)

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

GPP(General Purpose Partition)について

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

Armadillo-900 開発セット では、8 MiB の GPP を 4 つ作成しています。各領域の用途を「表 2.6. eMMC の GPP の用途」に示します。

表 2.6 eMMC の GPP の用途

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

[a]詳細は「10.22.4. ログ用パーティションについて」を参照ください。

2.8. ストレージデバイスのパーティション構成

Armadillo-900 開発セットの eMMC のパーティション構成を「表 2.7. eMMC メモリマップ」に示します。

表 2.7 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	100MiB	firm	ファームウェア用パーティション
5	2.8GiB	app	アプリケーション用パーティション
10	50MiB	secboot0	セキュアブート用の A 面パーティション[a]
11	50MiB	secboot1	セキュアブート用の B 面パーティション[a]

[a]セキュアブートを有効にした場合に署名済み Linux カーネルイメージが格納されます。

Armadillo-900 開発セットの eMMC のブートパーティションの構成を「表 2.8. eMMC ブートパーティション構成」に示します。

表 2.8 eMMC ブートパーティション構成

ディスクデバイス	サイズ	説明
/dev/mmcblk0boot0	4 MiB	A/B アップデートの A 面
/dev/mmcblk0boot1	4 MiB	A/B アップデートの B 面

Armadillo-900 開発セットの eMMC の GPP(General Purpose Partition)の構成を「表 2.9. eMMC GPP 構成」に示します。

表 2.9 eMMC GPP 構成

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

[a]詳細は「10.22.4. ログ用パーティションについて」を参照ください。

2.9. ソフトウェアのライセンス

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

手順について、詳細は以下の Howto を参照してください。

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

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

3. 起動と終了

3.1. 準備するもの

Armadillo-900 開発セット を使用する前に、まずは次のものを用意してください。

作業用 PC	Linux または Windows が動作し、ネットワークインターフェースと 1 つ以上の USB ポートを持つ PC です。
Armadillo-900 開発セット 一式	詳しくは「2.2.1. Armadillo-900 開発セット」をご参照ください。
1 GB 以上の microSD カード	Armadillo の初期化・ABOS のアップデートの際に使用します。
ネットワーク環境	Armadillo の初期化インストールディスクイメージなどを作業用 PC にダウンロードする手順があります。

3.2. Armadillo の初期化と ABOS のアップデート

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

3.2.1. 初期化インストールディスクの作成

初期化インストールディスクイメージを microSD カードに書き込み、初期化インストールディスクを作成します。ここでは、Windows で作成する方法を紹介します。（ATDE や Linux で実施する方法については「10.23. ATDE・Linux でインストールディスクを作成する」を参照してください）

1. 1 GB 以上の microSD カードを用意してください。

2. 標準のインストールディスクイメージをダウンロードします。Armadillo-900 開発セット インストールディスクイメージ [https://armadillo.atmark-techno.com/resources/software/armadillo-900/disc-image] から「Armadillo Base OS インストールディスクイメージ」をダウンロードしてください。

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

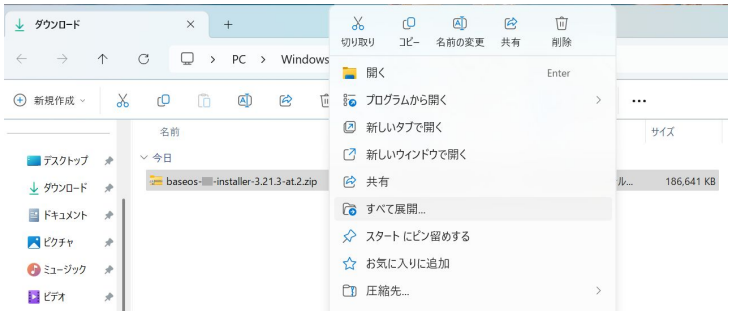


図 3.1 zip ファイルを展開

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

5. PC に microSD カードを接続します。

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

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

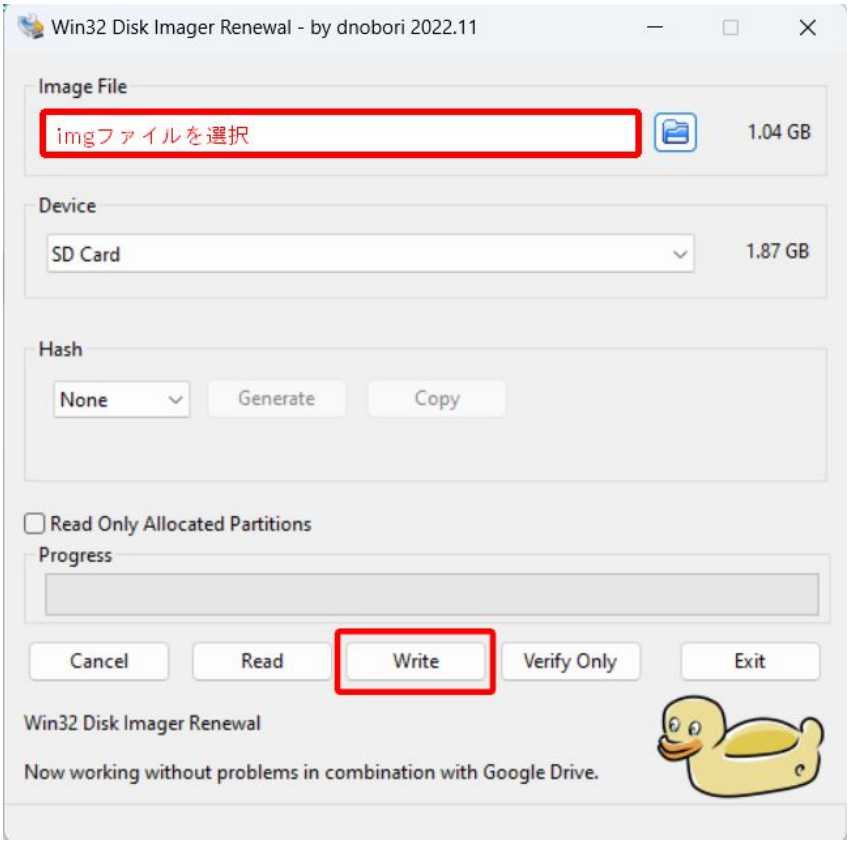


図 3.2 Win32 Disk Imager Renewal 設定画面

7. [Write] ボタンを押すと警告が出ますので、問題なければ[はい]を選択して書き込みます。

8. 書き込み終了ダイアログが表示されたらインストールディスクの作成は完了です。microSD カードを取り外してください。

3.2.2. インストールディスクを使用する

作成したインストールディスクを使用して、インストールを実施します。

以下の手順に沿って、 「図 3.3. Armadillo-900 開発セットを初期化する接続」 のとおりに接続します。

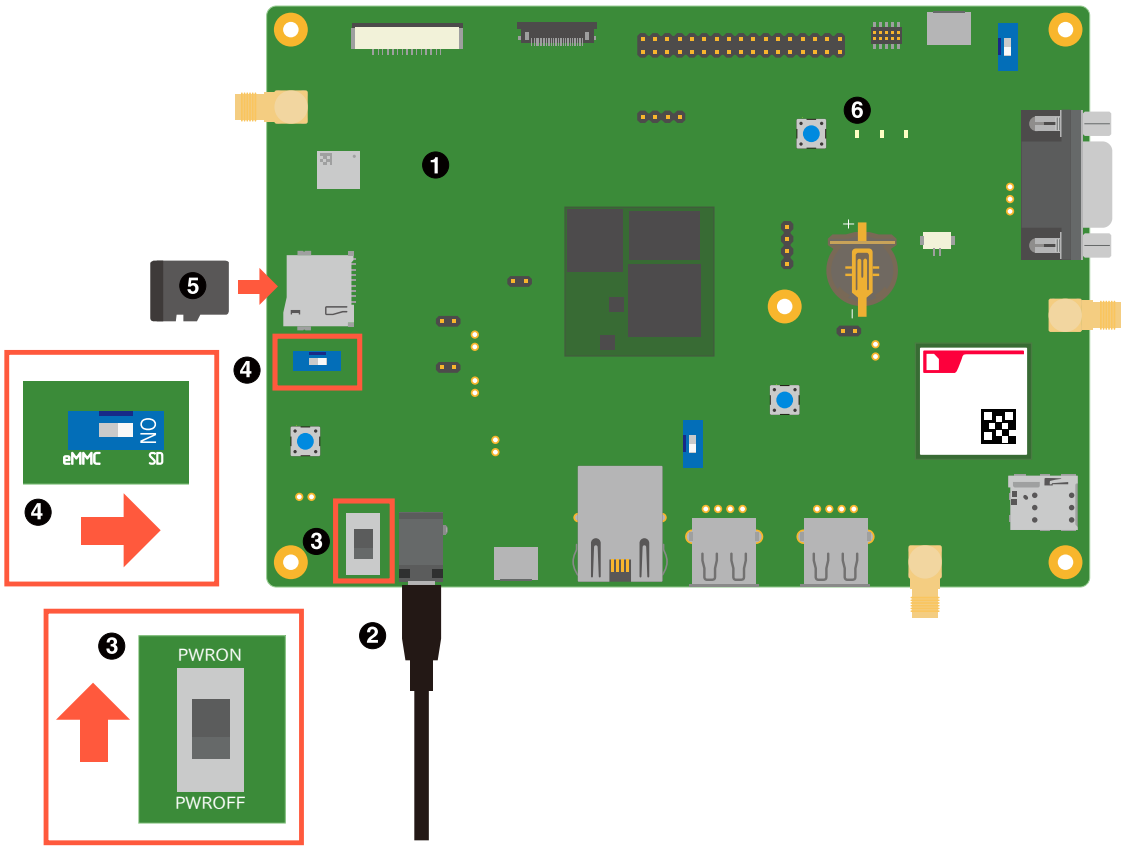


図 3.3 Armadillo-900 開発セットを初期化する接続

❶ Armadillo-900 開発セット

❷ AC アダプタ(12V/2.0A)

❸ 電源スイッチ

❹ 起動デバイス設定スイッチ

❺ microSD カード

❻ システム LED(SYS)

1. 起動デバイス設定スイッチを「SD」に切り替えます。(起動デバイス設定スイッチについての詳細は 「5.5.29. 起動デバイスを変更する」 をご参照ください)
2. microSD カードを CON1 に挿入します。
3. AC アダプタを接続して電源スイッチを ON にするとシステム LED(SYS) が点滅します。
4. 1 分程度で eMMC のソフトウェアの初期化が完了し、電源が切れます(システム LED(SYS) が消灯)。
5. システム LED(SYS) が消灯したら、電源を取り外し、続いて 起動デバイス設定スイッチを eMMC に設定し、microSD カードを外してください。

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

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

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

3.3. シリアルコンソールを使用する

Armadillo ではシリアルコンソールを通じて Linux コマンドを直接実行させることができます。

3.3.1. Armadillo と開発用 PC を接続

Armadillo-900 開発セットのシリアルコンソールを使用するために、 「図 3.4. シリアルコンソールを使用する配線例」 のとおりに配線を行ってください。この配線図は Armadillo-900 開発セットのシリアルコンソールを使用するための最低限の配線ですので、これに加えて他のインターフェースを接続しても問題ありません。

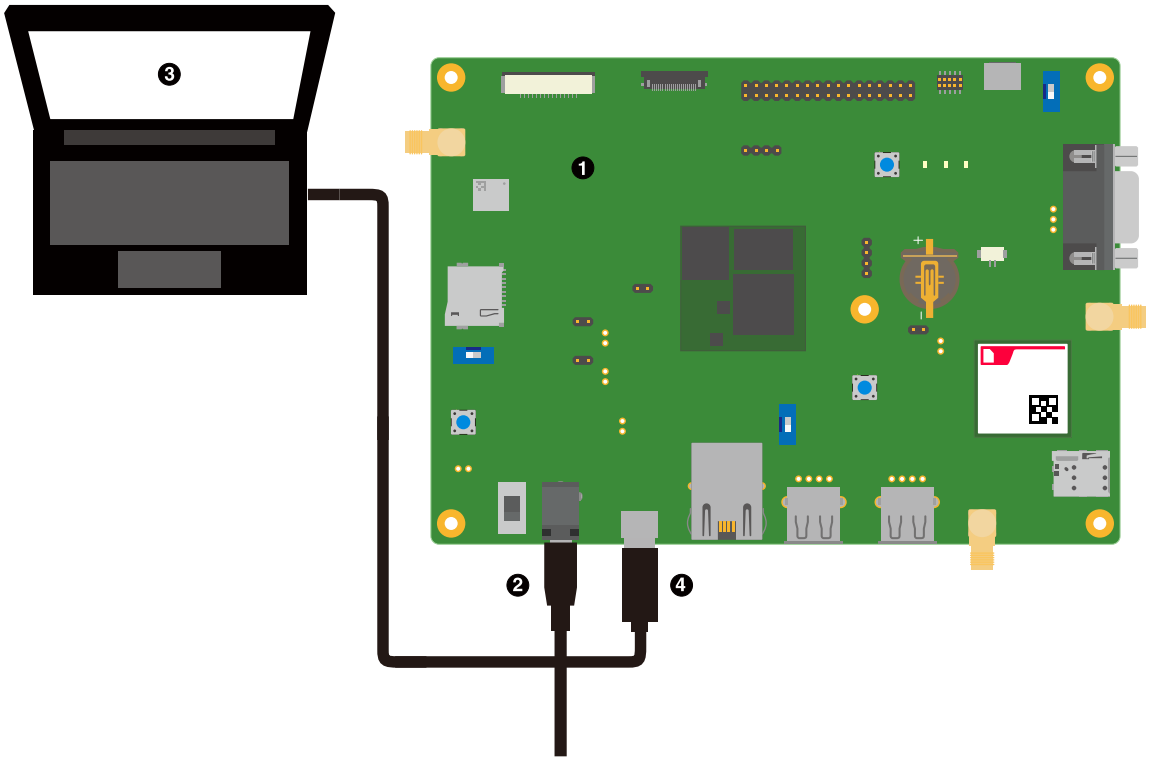


図 3.4 シリアルコンソールを使用する配線例

❶ Armadillo-900 開発セット

❷ AC アダプタ(12V/2.0A)

❸ 作業用 PC

❹ シリアル通信用 USB ケーブル(A-Type-C)

3.3.2. Tera Term の起動

本項では Windows 環境で Tera Term を使用したシリアルコンソールの操作方法について記載しています。(Linux や ATDE の場合は 「10.24. シリアル通信ソフトウェア(minicom)」 を参照してください。)

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

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

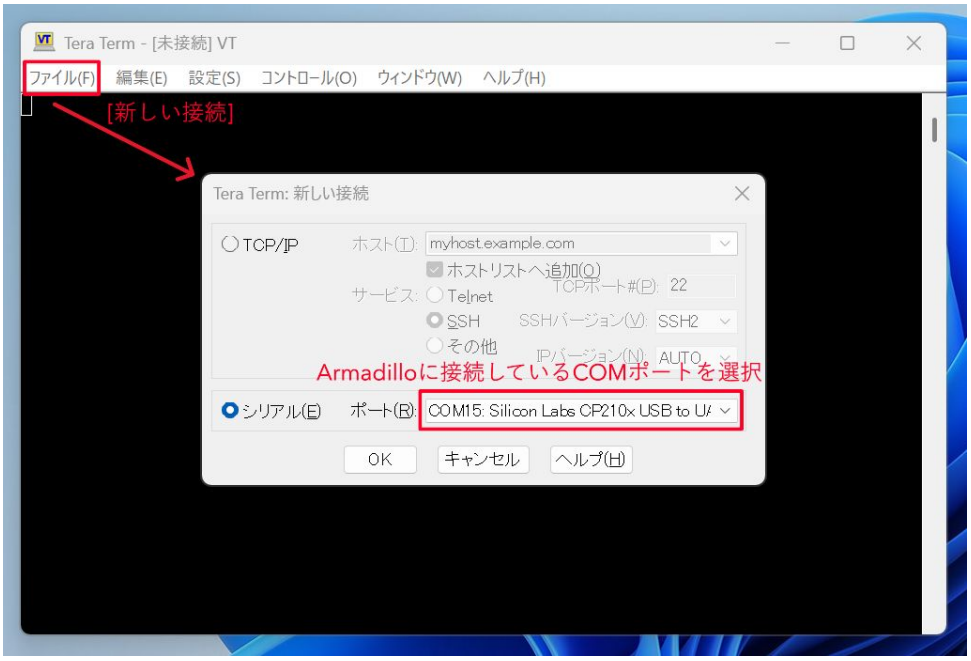


図 3.5 Armadillo を接続している COM ポートを指定

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

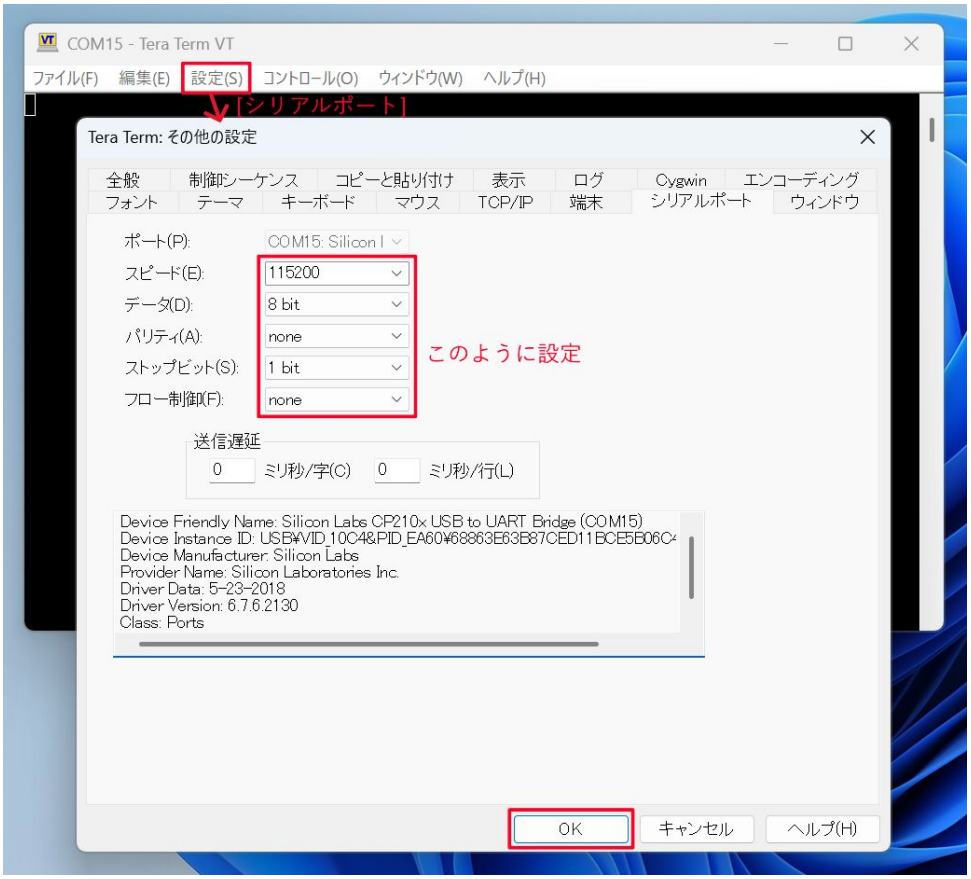


図 3.6 Armadillo を接続しているシリアルポートの設定

3.3.3. Armadillo の起動

電源を接続して Armadillo-900 開発セット を起動してください。シリアルコンソールに以下のような起動ログが出力されます。

以下に起動ログの例を示します。

```
U-Boot SPL 2023.04-at3 (Apr 14 2025 - 05:58:59 +0000)
Normal Boot
ELE firmware version 1.0.0-344acb84

: (省略)

Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0000000000 [0x411fd040]
[    0.000000] Linux version 5.10.236-0-at (builder@6c9022be5e1e) (aarch64-alpine-linux-musl-gcc
(Alpine 14.2.0) 14.2.0, GNU ld (GNU Binutils) 2.42) #1-Alpine SMP PREEMPT Mon Apr 14 01:01:38 UTC
2025
[    0.000000] Machine model: Atmark-Techno Armadillo-900 module EVA Board

: (省略)

Welcome to Alpine Linux 3.21
Kernel 5.10.236-0-at on an aarch64 (/dev/ttyLP0)
```

```
armadillo login:
```

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

3.3.4. ログイン

起動が完了するとログインプロンプトが表示されます。初期状態では「root」ユーザーと、一般ユーザーである「atmark」ユーザーが存在しますが、「atmark」ユーザーは初期状態ではロックされていますので、「root」ユーザーでログインしてください。(「atmark」ユーザーでログインする方法は 「10.1. ログインできるユーザーについて」 を参照してください。)

パスワードを設定します。

```
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。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。

❷ 新しいパスワードを再入力します。

Armadillo BaseOS ではルートファイルシステムに overlayfs を採用しています。そのため、そのままではシステムが終了するとパスワードの設定内容も消え、起動する度にパスワードを設定する必要があります。設定内容を反映させるためには、設定後に以下のように persist_file 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「4.3. persist_file について」を参照してください。

3.3.5. Armadillo の終了方法

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

```
armadillo:~#  * WARNING: clock skew detected!
local                    | * Stopping local ...zramswap                | * Deactivating zram swap    ⏎
device ...podman-atmark            | * Stopping all podman containers ... [ ok ]

:(省略)
```

```
The system is going down NOW!
Sent SIGTERM to all processes
Sent SIGKILL to all processes
Requesting system poweroff
```

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

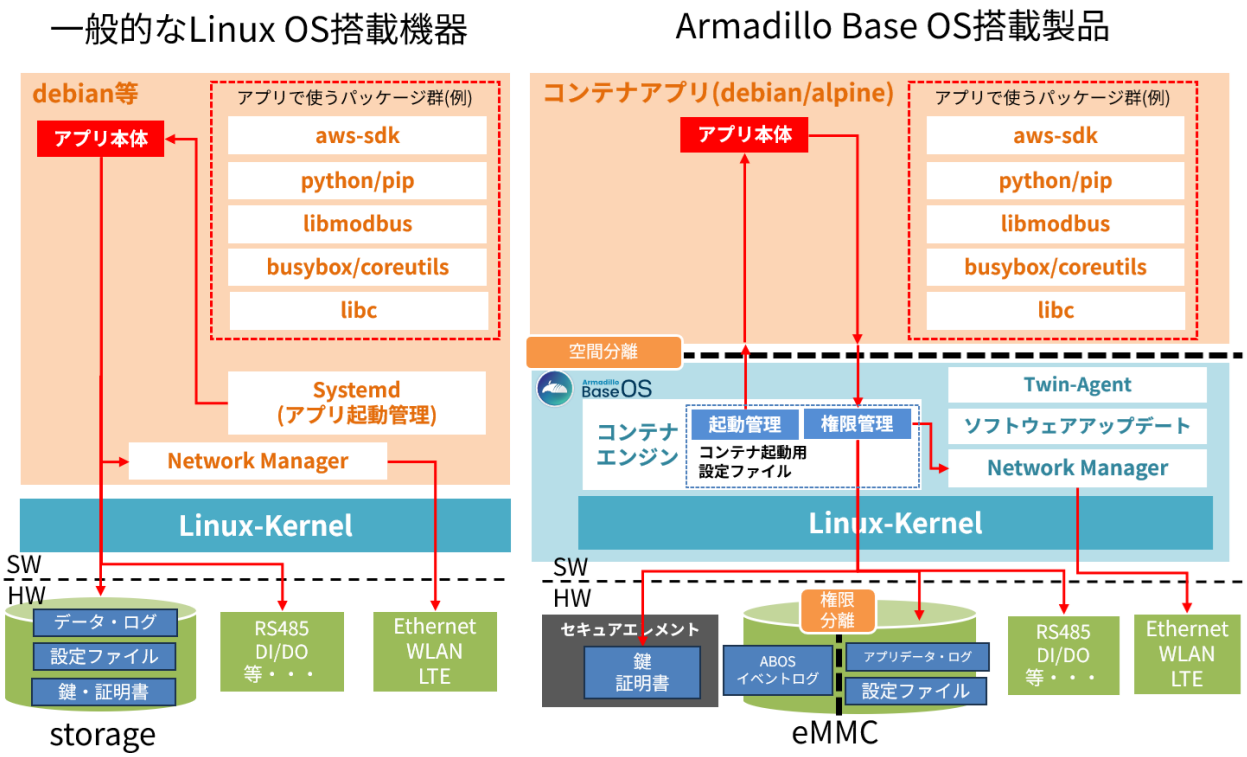
コンデンサに蓄えられた電荷が抜ける前に電源を再投入した場合、電源シーケンスが守られず、起動しない等の動作不具合の原因となります。

4. Armadillo Base OS チュートリアル

本章は、一般的な Linux OS 搭載組み込み機器と Armadillo Base OS の違いや特徴、基礎的なコンテナの使い方など、 Armadillo-900 開発セット の動作確認を進めるうえで必要となる基本知識を身に着け、使い方に慣れるためのチュートリアルです。

4.1. 一般的な Linux OS 搭載組み込み機器との違いと特長

ここでは、一般的な Linux OS 搭載組み込み機器と Armadillo Base OS 搭載機器との違いや特長について説明します。



・ コンテナによるアプリケーションの起動

上記の図に示すように、一般的な Linux OS 搭載組み込み機器ではアプリケーションの実行環境をユーザーランド上に直接用意し、Systemd などでアプリケーションを自動実行させるのが一般的です。Armadillo Base OS 搭載機器では、アプリケーションの実行環境をコンテナ内に用意して、「コンテナ起動設定ファイル」を所定の場所に配置することでコンテナ(=アプリケーション)を自動実行させます。ベースとなるコンテナを生成し、アプリケーションの動作に必要な apt 等のパッケージ類、python pip 等の各種言語用のライブラリをインストールし、アプリケーション本体を配置します。

・ アクセス権限の分離と付与

Armadillo Base OS とコンテナは空間分離されており、標準ではコンテナから Armadillo Base OS によって管理されているストレージやシリアル等のインターフェース、ネットワークなどへのアクセスはできません。「コンテナ起動設定ファイル」で明示的に権限を付与することでこれらへのアクセスが可能になります。

・セキュアエレメント標準搭載

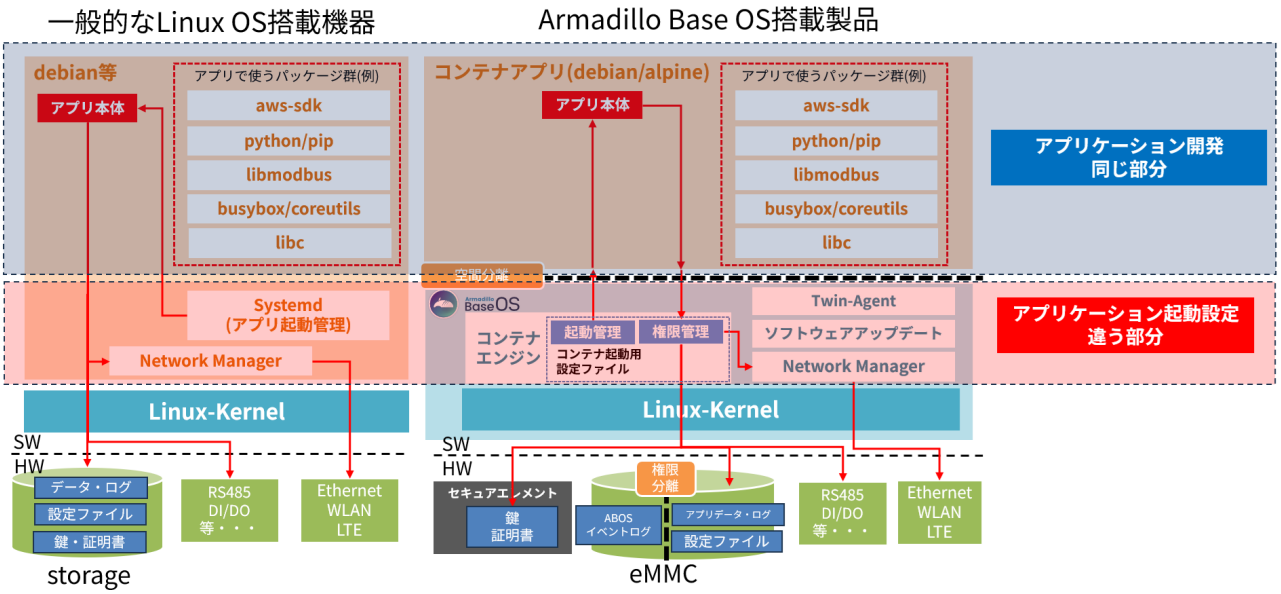
Armadillo Base OS 搭載機器は、標準でセキュアエレメントを搭載しており、対応した暗号化方式の認証鍵や証明書を安全に保存・利用することが可能です。

・ルートファイルシステム領域への書き込み制限と書き込み方法

一般的な Linux OS 搭載組み込み機器では、ストレージ（NAND フラシュメモリ）の寿命からの保護や、書き込み中の電源断等からのデータ保護のために overlayfs で運用するのが一般的です。Armadillo Base OS 搭載機器でも、Armadillo Base OS のルートファイルシステム領域が overlayfs によって保護されており、Linux の cp コマンド等で書き込みを行っても揮発性の RAM 上に保存され、不揮発性のストレージ（eMMC）側に保存されず、電源を入れ直すと消失します。ルートファイルシステム内のファイルの作成や編集後、ストレージ（eMMC）に永続的に保存するには、明示的に persist_file コマンドを実行する必要があります。

・Armadillo Base OS の障害発生時のログについて

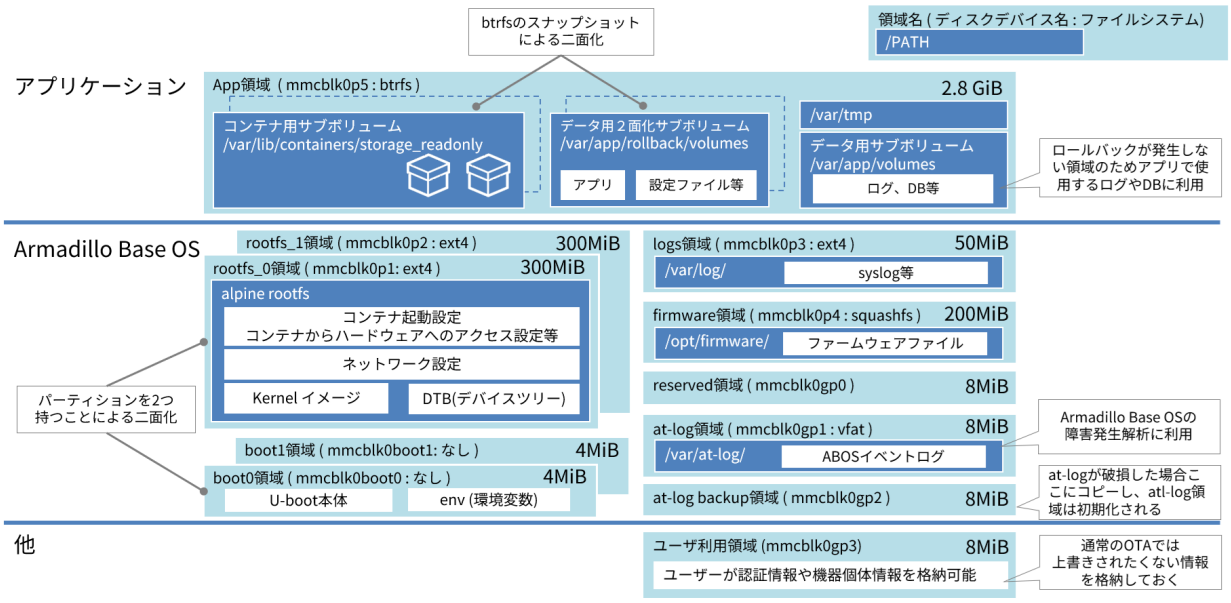
一般的な Linux OS 搭載組み込み機器ではシステムが出力するログは /var/log ディレクトリに保存されます。Armadillo Base OS 搭載機器では、これに加えて、障害発生時のログが /var/at-log ディレクトリに出力されます。これを ABOS イベントログと呼びます。/var/at-log は、ルートファイルシステムとは別のパーティションになっているので、ルートファイルシステムに障害が発生した場合でも、/var/at-log のパーティションが無事であれば、ログファイルを取り出して、不具合等の解析に利用することができます。



・コンテナと Armadillo Base OS の役割分担

上記の図に示すように、アプリケーションの起動に関わる設定を Armadillo Base OS 側で実施し、アプリケーション本体の開発をコンテナ上で実施します。コンテナの部分にのみ着目すると、Debian 等の Linux OS 搭載組み込み機器の開発と同じであり、異なる部分はアプリケーションの起動設定を Armadillo Base OS に対して行う部分となります。

4.1.1. eMMC パーティション構成の解説



- eMMC パーティション構成解説

  Armadillo Base OS の alpine ルートファイルシステムや Kernel イメージ、DTB は二面化された rootfs_0 領域、rootfs_1 領域に配置します。bootloader（U-Boot）や U-Boot の環境変数も二面化された boot0 領域、boot1 領域に配置します。これらの領域は原則リードオンリーで使用します。

  Armadillo Base OS が使用するログ領域は、Armadillo Base OS のルートファイルシステムや bootloader とは分離されておりライト可でマウントしています。alpine が出力するログを保存する「 /var/log 」と Armadillo Base OS の障害発生時のログを保存する「 /var/at-log 」の２種類があり、「 /var/at-log 」はバックアップされています。

  一部、ルートファイルシステムに含めることのできないファームウェア等を「firmware 領域」に配置します。また、ユーザーが自由に使える「ユーザー利用領域」も用意しています。

- コンテナ用の App 領域とサブボリューム

  コンテナ本体及びコンテナが使用するデータ等は App 領域に保存します。App 領域はファイルシステム（btrfs 形式）でフォーマットされており、btrfs のサブボリュームによって更に細かく領域が分けられています。

  コンテナはサブボリューム（ /var/lib/containers/storage_readonly ）内に配置され、btrfs のスナップショット機能を用いて二面化されています。コンテナ用サブボリュームは原則リードオンリーで使用します。

  アプリケーションが使用するログや設定ファイルなどのデータは、コンテナ用サブボリュームとは分離した別のデータ用のサブボリュームをライト可でマウントして使用します。データ用のサブボリュームには二面化された「 /var/app/rollback/volumes 」 と二面化していない「 /var/app/volumes 」の２種類があり、用途によって使い分けます。

4.2. vi エディタを使ってみよう

vi は UNIX 系 OS に標準でインストールされているコマンドラインベースのテキストエディタです。Armadillo にも標準でインストールされています。vi エディタはモードを持っていることが大きな特徴で、次の 2 つのモードがあります。

- コマンドモード： 入力した文字がすべてコマンドとして扱われます。起動直後はこのモードです。i や a などを入力すると入力モードに移行します。
- 入力モード： 文字の入力ができます。ESC キーを入力するとコマンドモードに移行します。

vi エディタを初めて使う際は操作が少し特殊に感じるかもしれませんが、慣れてくると効率的にテキストを編集できるようになります。本書では Armadillo の設定ファイルの編集などで vi エディタを使用しているため、ここでは vi エディタの使用方法について簡単に解説します。

4.2.1. vi の起動

まずは、以下のコマンドを入力して vi を起動します。file にファイル名のパスを指定すると、ファイルの編集を行います。file が存在しない場合は自動的に新規で作成されます。また、起動直後はコマンドモードの状態です。

```
[armadillo ~]# vi [file]
```

図 4.1 vi の起動

4.2.2. 文字の入力

文字を入力するために、i か a を入力して入力モードへ移行します。（入力モードに移行すると、画面左下の - が I に変わります。）入力モードに移行した後は、キーを入力することで文字がそのまま入力されます。下表や下図のように、i と a では文字入力の開始位置が異なります。状況に応じて適宜使い分けてください。

表 4.1 入力モードに移行するコマンド

コマンド	動作
i	カーソルのある文字の直前から文字入力を開始
a	カーソルのある文字の直後から文字入力を開始

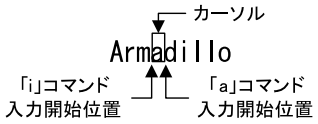


図 4.2 入力モードに移行するコマンドによる開始位置の違い

入力モードからコマンドモードに戻りたいときは、ESC キーを入力します。コマンドモードの状態で ESC キーを入力しても何も起こらないため、現在のモードがどちらなのか分からなくなった際は、ひとまず ESC キーを入力してコマンドモードへ戻ると良いです。

4.2.3. カーソルの移動

入力モード・コマンドモードどちらの状態でも方向キーを入力してカーソルの移動ができます。これに加えて、コマンドモードでは下表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 4.2 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

4.2.4. 文字の削除

以下のいずれかの方法で文字を削除できます。ただし、コンソールの環境によっては入力モードで BS(Backspace)キーを入力しても「^H」文字が表示されるだけで文字が削除できない場合があります。その場合は後者の方法で文字を削除してください。

- 入力モードで BS(Backspace)キーを入力する。
- コマンドモードで下表のコマンドを入力する。

表 4.3 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

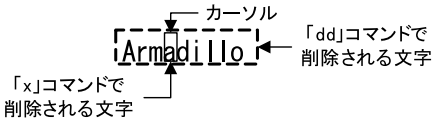


図 4.3 文字を削除するコマンドによる削除範囲の違い

4.2.5. 保存と終了

ファイルの保存や終了はコマンドモードで下表のコマンド入力後に Enter キーを押すことでできます。これらのコマンドは「：」(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左下にカーソルが移り、入力したコマンドが表示されます。

表 4.4 保存・終了コマンド

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

4.3. persist_file について

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

「図 4.4. OverlayFS の構成」のように、Armadillo Base OS では「下位層」である eMMC 上に存在するファイルシステムに対して「上位層」である RAM 上のファイルシステムを統合しています。

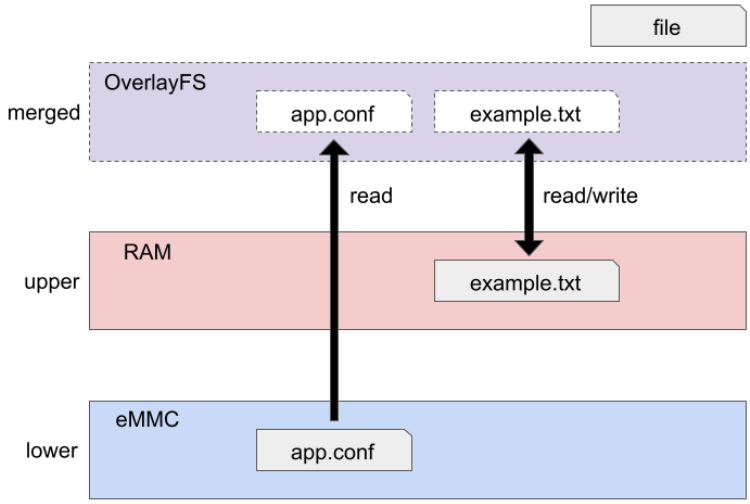


図 4.4 OverlayFS の構成

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

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

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

開発中などに rootfs の変更内容を保持するには、変更したファイルに対して `persist_file` コマンドを使用する必要があります。

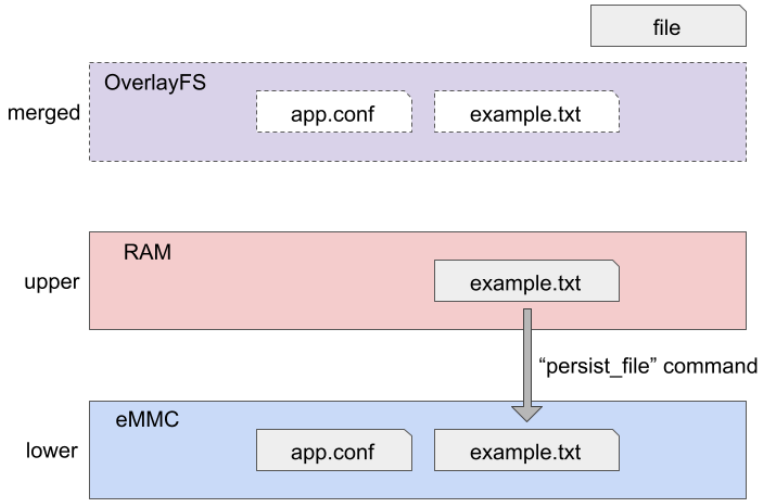


図 4.5 `persist_file` コマンドの説明

OverlayFS からファイルを書き込むと RAM 上のファイルに書き込みが行われます。eMMC 上にのみファイルが存在する場合、「図 4.6. 書き込み時の OverlayFS の挙動」に示すように、そのファイルは eMMC から読み込まれて編集した内容は RAM 上に書き込まれます。

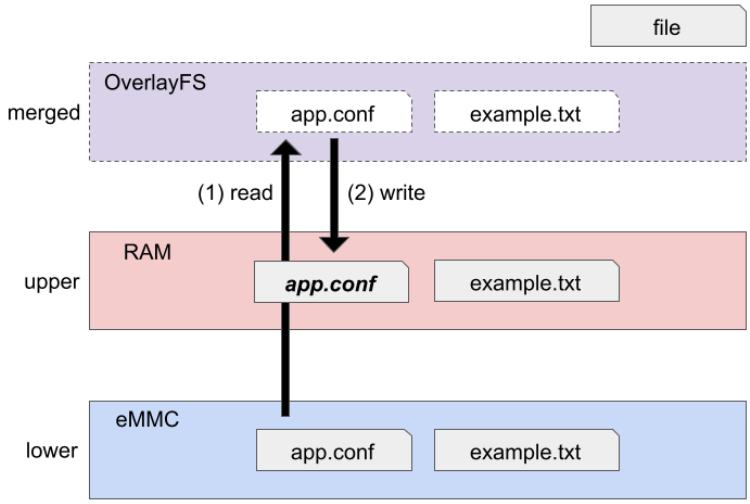


図 4.6 書き込み時の OverlayFS の挙動

編集内容を eMMC 上のファイルに反映させるためには、「図 4.7. 書き込み後の `persist_file` コマンドの実行」に示すように、`persist_file` コマンドを実行する必要があります。

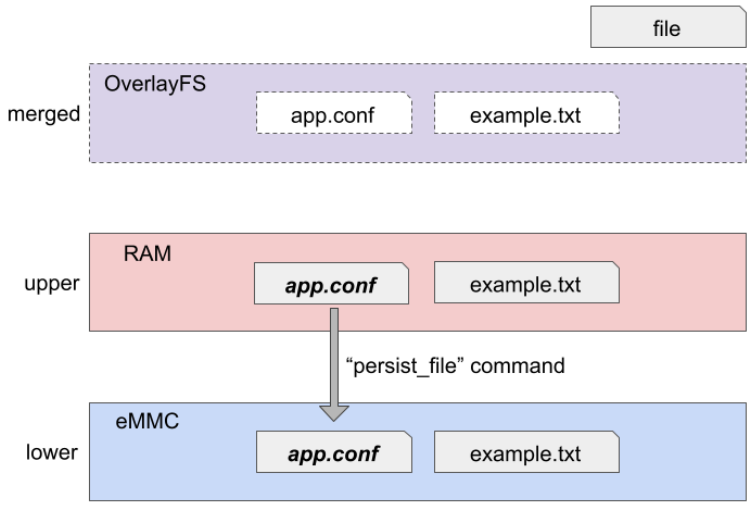


図 4.7 書き込み後の `persist_file` コマンドの実行

開発以外の時は安全のため、ソフトウェアアップデートによる更新を実行してください。SWUpdate に関しては 「7.3.3. アップデート機能について」 を参照してください。

rootfs の内容を変更しても、ソフトウェアアップデートを実施した際に変更した内容が保持されない可能性があります。ソフトウェアアップデート実施後も変更内容を保持する手順に関しては 「10.4. swupdate_preserve_files について」 を参照してください。

以下、persist_file コマンドの使い方について説明します。

persist_file コマンドの概要を 「図 4.8. persist_file のヘルプ」 に示します。

```
[armadillo ~]# persist_file -h
Usage: /usr/bin/persist_file [options] file [more files...]

Mode selection:
  (none) single entry copy
  -d, --delete   delete file
  -l, --list     list content of overlay
  -a, --apk      apk mode: pass any argument after that to apk on rootfs
  -R, --revert   revert change: only delete from overlay, making it
                 look like the file was reverted back to original state

Copy options:
  -r, --recurse  recursive copy (note this also removes files!)
  -p, --preserve make the copy persist through baseos upgrade
                 by adding entries to /etc/swupdate_preserve_files
  -P, --preserve-post   same, but copy after upgrade (POST)

Delete options:
  -r, --recurse  recursively delete files

Common options:
  -v, --verbose  verbose mode for all underlying commands

Note this directly manipulates overlayfs lower directories
so might need a reboot to take effect
```

図 4.8 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 4.9 persist_file 保存・削除手順例

❶ 追加・変更したファイルを rootfs へコピーします。

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

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

2. ソフトウェアアップデート後も変更を維持する手順例

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

図 4.10 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 4.11 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 4.12 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。

4.4. Podman を使ってみよう

4.4.1. 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や、その仮想化技術のことです。一般的な仮想化では仮想環境内に OS （ゲスト OS）が搭載されていますが、コンテナによる仮想化では、OS をコンテナ内に搭載しない（ゲスト OS が無い）という特徴があります。これにより、一般的な仮想化のように、

- 異なる個体のハードウェアでも同一の環境を簡単に再現できる（再現性と移植性）
- セキュリティリスクを低減できる

といった恩恵がありながら、これに加えて、

- 軽量である
- アプリケーションの起動が素早い
- 環境を手軽に構築・削除できる

といったメリットもあります。

Podman [https://podman.io] 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す。Podman は同じコンテナ管理ソフトウェアとしてよく知られている Docker とコマンドインターフェースの互換性がある一方で、Docker よりもコンテナ間の独立性が高いなどのセキュリティ的な長所があります。

ABOS では Podman を採用しており、ユーザーアプリケーションはコンテナ内で実行されることを想定しています。そのため、Podman は ABOS を搭載した Armadillo における根幹の機能であり、Armadillo を使って機器を開発したり、アプリケーションを開発・運用したりする上で、Podman やコンテナに関する理解が非常に重要となります。

ここでは、Podman やコンテナに慣れる・上手に活用できるようになる目的で、Podman の使用方法や動作について説明します。

4.4.2. コンテナの基本的な操作

以下で紹介する Podman のコマンドの多くは、 --help オプションを付けることでより詳細な情報を確認することができます。

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

図 4.13 podman pull --help 実行例

4.4.2.1. コンテナイメージをダウンロードする

コンテナを作成するためには、その元となるイメージ（コンテナイメージ）が必要です。まずは、例として、Alpine Linux のコンテナイメージを Docker Hub [https://hub.docker.com] からダウンロードしてきます。

コンテナイメージのダウンロードは podman pull [source] コマンドでできます。これはインターネット上のレジストリ（Docker Hub など）にある指定したイメージ（ source ）をローカルにダウンロードするコマンドです。

```
[armadillo ~]# podman pull docker.io/alpine
Trying to pull docker.io/library/alpine:latest...
Getting image source signatures
Writing manifest to image destination
8d591b0b7dea080ea3be9e12ae563eebf9869168ffced1cb25b2470a3d9fe15e
```

図 4.14 コンテナイメージのダウンロード

4.4.2.2. コンテナイメージ一覧を表示する

先ほどダウンロードしたイメージが本当にダウンロードできたかどうかを podman images コマンドで確認します。これは、ローカルにあるイメージ一覧を表示するコマンドです。

```
[armadillo ~]# podman images
REPOSITORY                TAG         IMAGE ID      CREATED       SIZE
docker.io/library/alpine  latest      8d591b0b7dea  2 months ago  8.47 MB
```

図 4.15 イメージ一覧

先ほどダウンロードしたイメージがあることを確認できます。

4.4.2.3. イメージからコンテナを作成して起動する

Podman や Docker に詳しい方は podman run コマンドでコンテナの作成＆起動ができることをご存知かもしれませんが、ABOS では podman_start コマンドを積極的に使用してください。このコマンドはアットマークテクノが用意したコマンドで、「10.8.3. コンテナ起動設定ファイルを作成する」 で紹介するコンテナの自動起動などでも使用します。

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 podman_start コマンドで作成します。

まず、podman_start コマンドを使用するためには、コンテナについての設定（コンテナイメージの指定・実行するコマンド・デバイスへのアクセス権限など）を記述したコンフィグファイルを指定の場所に指定のファイル名であらかじめ用意しておきます。デフォルトでは指定の場所は /etc/atmark/containers/ ディレクトリになります。ファイル名は"コンテナの名前としたい文字列"に .conf を末尾に付加した文字列になります。

ここでは、my_container というコンテナの名前でコンフィグファイルを作成します。

```
[armadillo ~]# vi /etc/atmark/containers/my_container.conf
set_image docker.io/alpine ❶
set_command ls / ❷
```

図 4.16 コンテナの設定ファイルを用意する

❶ set_image に元となるコンテナイメージを指定します。ここでは、先ほどダウンロードしたばかりの docker.io/alpine イメージを指定しています。

❷ set_command に、実行するコマンドを指定します。ここでは、簡単な例として "ls /" コマンドを指定しています。

コンフィグファイルに記述できる設定内容や、より詳細な説明については 「10.8.3. コンテナ起動設定ファイルを作成する」 を参照してください。

コンフィグファイルを作成したら、いよいよ podman_start コマンドでコンテナの作成＆起動を行います。後ろの引数にコンテナの名前（コンフィグファイル名の .conf を除いた部分）を指定してください。

```
[armadillo ~]# podman_start my_container
Starting 'my_container'
e9fbb65655be14aee6fb2e2006893050feb3ee8b7a7eceefeb7ff3fd18eb161e
[armadillo ~]#
```

図 4.17 コンテナを作成＆起動する

"ls /" を実行するだけの "my_container" という名前のコンテナが作成＆起動しました。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。（後述のコマンドで確認できます）

ここでは簡単な例のために podman_start コマンドを手動で呼び出しましたが、コンフィグファイルが /etc/atmark/containers/ に置かれていると、Armadillo 起動時に podman_start コマンドでそのコンフィグファイルからコンテナが自動的に作成＆起動します。自動起動が不要な場合にはコンフィグファイルに set_autostart no を記述してください。

4.4.2.4. コンテナのログを確認する

コンテナ内のログは podman logs コマンドで確認できます。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起動したため、そのコマンド結果が表示されることを確認できるはずです。ここで表示されているのは、コンテナ内部の "/" ディレクトリ内のディレクトリ・ファイル一覧です。

```
[armadillo ~]# podman logs my_container
bin
dev
:（省略）
usr
var
[armadillo ~]#
```

図 4.18 コンテナのログを確認する

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。

4.4.2.5. コンテナー覧を表示する

作成したコンテナの一覧は podman ps -a コマンドで確認できます。

```
[armadillo ~]# podman ps -a
CONTAINER ID  IMAGE                            COMMAND     CREATED        STATUS                    ⏎
PORTS       NAMES
e9fbb65655be  docker.io/library/alpine:latest  ls /        40 seconds ago  Exited (0) 39 seconds    ⏎
ago             my_container
[armadillo ~]#
```

図 4.19 コンテナー覧の表示実行例

"ls /"が正常終了したため、STATUS が「Exited (0)」になっていることが確認できます。

また、コンテナ名やコンテナ ID を確認することもできます。-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。

4.4.2.6. 実行中のコンテナの中に入る

実行中のコンテナに入り、コンテナ内で CLI 操作するには 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、コンテナ内部で CLI 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。

まずは、先ほどまで使用していた /etc/atmark/containers/my_container.conf を書き換えます。

```
[armadillo ~]# vi /etc/atmark/containers/my_container.conf
set_image docker.io/alpine
set_command sleep infinity
[armadillo ~]# podman_start my_container
Starting 'my_container'
2782f3ac49b94eeaa37154c21609e102fbff25d5e388e5be5b705b3684aa866b
[armadillo ~]# podman exec -it my_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 4.20 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＆起動し、その後コンテナ内で 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で CLI 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 infinity 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
[armadillo ~]#
```

図 4.21 コンテナ内部のシェルから抜ける

4.4.2.7. コンテナを停止する

exit コマンドで抜けても sleep infinity は終了していないため、コンテナはまだ実行中です。podman stop コマンドで動作中のコンテナを停止します。

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

図 4.22 コンテナを停止する実行例

4.4.2.8. コンテナを起動する

作成済みのコンテナを再度起動するためには podman start コマンドを実行します。podman_start コマンドとは違い、podman と start の間はスペースであることに注意してください。

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

図 4.23 コンテナを起動する実行例

4.4.2.9. コンテナを削除する

作成済みコンテナの削除は podman rm コマンドで行います。--force オプション無しの場合はあらかじめコンテナを停止しておく必要があります。

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

図 4.24 コンテナを削除する

最後の podman ps -a コマンドの出力結果から、コンテナが削除されていることが確認できます。

4.4.2.10. コンテナイメージを削除する

コンテナイメージの削除には 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あらかじめ削除しておく必要があります。podman rmi コマンドではイメージの名前か 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 images
REPOSITORY                TAG         IMAGE ID      CREATED       SIZE
docker.io/library/alpine  latest      8d591b0b7dea  2 months ago  8.47 MB
[armadillo ~]# podman rmi docker.io/library/alpine
Untagged: docker.io/library/alpine:latest
Deleted: 8d591b0b7dea080ea3be9e12ae563eebf9869168ffced1cb25b2470a3d9fe15e
[armadillo ~]# podman images
REPOSITORY  TAG         IMAGE ID    CREATED     SIZE
[armadillo ~]#
```

図 4.25 コンテナイメージを削除する

最後の podman images コマンドの出力結果から、コンテナが削除されていることが確認できます。

podman images で R/O が true として表示される（Read-Only）コンテナイメージについては、 podman rmi を実行するとエラーとなります。Read-Only のコンテナイメージについては abos-ctrl podman-rw rmi をご使用ください。 abos-ctrl podman-rw については 「10.8.1.4. コンテナイメージを eMMC に保存する」 を参照してください。

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

図 4.26 Read-Only のイメージを削除する実行例

4.4.2.11. コンテナイメージを作成する

ここまでは、 podman pull でダウンロードしてきたコンテナイメージを元にコンテナを作成してきましたが、コンテナイメージを元にして新たにカスタマイズしたコンテナイメージを作成することもできます。これには、どのようにカスタマイズするかを記述した Dockerfile と podman build コマンドを使います。

1. まずは Dockerfile を用意します

```
[armadillo ~]# vi Dockerfile
FROM docker.io/alpine:latest ❶

RUN apk upgrade && apk add python3 && rm -f /var/cache/apk/* ❷
```

図 4.27 Dockerfile の用意

❶ FROM に元となるコンテナイメージを指定します。指定したコンテナイメージは自動的にダウンロードされるため、あらかじめダウンロードしておく必要はありません。

❷ RUN に、コンテナイメージ作成時に実行するコマンドを指定します。ここでは、簡単な例として Python3 をインストールしています。

2. Dockerfile からコンテナイメージを作成します

```
[armadillo ~]# podman build -t my_image ./ ❶
STEP 1/2: FROM docker.io/alpine:latest
STEP 2/2: RUN apk upgrade && apk add python3 && rm -f /var/cache/apk/*

:（省略）

COMMIT my_image
--> ba6d17aa8020
Successfully tagged localhost/my_image:latest
63fe9d38336c9a1b9600ae1f86d85fea2e8f4b3ac72093111a9c15de4bbb73a1
```

図 4.28 コンテナイメージの作成

❶ カレントディレクトリにある Dockerfile から my_image という名前でコンテナイメージを作成します。

podman images で、たった今作成したコンテナイメージと、 FROM で指定したことで自動的にダウンロードされたコンテナイメージがあることを確認できます。

```
[armadillo ~]# podman images
REPOSITORY                TAG         IMAGE ID      CREATED         SIZE
localhost/my_image        latest      63fe9d38336c  11 seconds ago  53.1 MB
docker.io/library/alpine  latest      8d591b0b7dea  2 months ago    8.47 MB
[armadillo ~]#
```

図 4.29 作成したコンテナイメージの確認

4.4.2.12. コンテナをコンテナイメージとして保存する

ここまでは、コンテナイメージからコンテナを作成してきました。一方で、コンテナからコンテナイメージを作成することもできます。

コンテナはコンテナイメージから一方向的に作成されるため、コンテナに対する変更内容はコンテナが停止してしまうと失なわれてしまいます。ですが、コンテナからコンテナイメージを作成することで、コンテナが停止してもコンテナに対する変更内容を残しておくことができます。言い換えれば、コンテナをコンテナイメージとして保存できます。

これには、 podman commit コマンドを使用します。

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

図 4.30 コンテナをコンテナイメージとして保存する

❶　コンテナ my_container を hoge_image という名前のコンテナイメージとして保存します。

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、元となるコンテナイメージから作り直してください。

4.4.2.13. コンフィグファイルとコンテナイメージを永続化する

ここまでで、コンテナイメージのダウンロード・作成（ podman pull, podman build ）とコンフィグファイルの作成を行いました。ですが、 podman pull や podman build で用意したコンテナイメージや、作成・変更したコンフィグファイルはデフォルト状態ではメモリ上にしか保存されません。これは、Armadillo Base OS の意図した仕様です。

そのため、このまま Armadillo を終了すると、これらのデータは消えてしまいます。Armadillo を終了してもデータが消えないようにするためには、コンフィグファイルは persist_file コマンドで保存し、コンテナイメージは abos-ctrl podman-storage コマンドで保存します。

（ここではチュートリアルのために簡単な内容のみ記載しています。コンテナイメージを eMMC に保存するためのより本格的な内容については 「10.8.1.4. コンテナイメージを eMMC に保存する」 を参照してください。）

まず、コンフィグファイルは次の persist_file コマンドで保存します。

```
[armadillo ~]# persist_file my_container.conf
```

図 4.31 コンフィグファイルを保存する

次に、コンテナイメージは次の abos-ctrl podman-storage コマンドで保存します。

```
[armadillo ~]# abos-ctrl podman-storage ❶
List of images configured on development storage:
REPOSITORY                TAG         IMAGE ID      CREATED         SIZE
localhost/my_image        latest      63fe9d38336c  32 minutes ago  53.1 MB
docker.io/library/alpine  latest      8d591b0b7dea  2 months ago    8.47 MB

What should we do? ([C]opy (default), [N]othing, [D]elete)
C ❷
```

```
：（省略）

Podman is in tmpfs mode
[armadillo ~]# podman images ❸
REPOSITORY                TAG         IMAGE ID      CREATED         SIZE        R/O
localhost/my_image        latest      63fe9d38336c  33 minutes ago  53.1 MB     true
docker.io/library/alpine  latest      8d591b0b7dea  2 months ago    8.46 MB     true
[armadillo ~]#
```

図 4.32 コンテナイメージを保存する

❶ abos-ctrl podman-storage をオプション無しで実行します。

❷ 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（C）します。

❸ コピーされたイメージを確認します。イメージが読み取り専用（R/O, Read only）になっていることが確認できます。

4.4.2.14. 開発時に有用な—privileged オプション

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

コンフィグファイルに以下の行を追加してください。

```
add_args --privileged
```

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

Podman やコンテナについてのより詳細な使用方法については 「10.8. コンテナについて」 を参照してください。

4.5. ストレージを使用する

4.5.1. Armadillo Base OS から直接使用する

ここでは、microSD/microSDHC/microSDXC カードを microSD カードと表記し、microSD 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。

SDXC/microSDXC カードを使用する場合は、事前に「4.5.1.1. ストレージのパーティション変更とフォーマット」を参照してフォーマットを行う必要があります。

これは、Linux カーネルが exFAT ファイルシステムを扱うことができないためです。通常、購入したばかりの SDXC/microSDXC カードは exFAT ファイルシステムでフォーマットされています。

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 [device] [dir]
```

図 4.33 mount コマンド書式

[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/mmcblk2p1 、パーティション 2 の場合は /dev/mmcblk2p2 となります。

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

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

microSD カード内のファイルは、/mnt ディレクトリ以下に見えるようになります。

```
[armadillo ~]# mount /dev/mmcblk2p1 /mnt
[armadillo ~]# ls /mnt
```

図 4.34 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 4.35 ストレージのアンマウント

4.5.1.1. 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 4.36. fdisk コマンドによるパーティション変更」に示します。

```
[armadillo ~]# fdisk /dev/{sd-dev}p1

The number of cylinders for this disk is set to 12608.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
```

```
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help): p ❶
Disk /dev/mmcblk2p1: 394 MB, 413138944 bytes, 806912 sectors
12608 cylinders, 4 heads, 16 sectors/track
Units: sectors of 1 * 512 = 512 bytes

Device         Boot StartCHS    EndCHS        StartLBA     EndLBA    Sectors  Size Id Type
/dev/mmcblk2p1p1    0,1,1       1023,3,16           16     806911     806896  393M 83 Linux

Command (m for help): d ❷
Selected partition 1

Command (m for help): n ❸
Partition type
   p   primary partition (1-4)
   e   extended
p ❹
Partition number (1-4): 1 ❺
First sector (16-806911, default 16): ❻
Using default value 16
Last sector or +size{,K,M,G,T} (16-806911, default 806911): +100M ❼

Command (m for help): n
Partition type
   p   primary partition (1-4)
   e   extended
p
Partition number (1-4): 2 ❽
First sector (204816-806911, default 204816):
Using default value 204816
Last sector or +size{,K,M,G,T} (204816-806911, default 806911):
Using default value 806911

Command (m for help): w ❾
The partition table has been altered.
Calling ioctl() to re-read partition table
fdisk: WARNING: rereading partition table failed, kernel still uses old table: Invalid argument

[armadillo ~]# fdisk /dev/{sd-dev}p1

The number of cylinders for this disk is set to 12608.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help): p ❿
Disk /dev/mmcblk2p1: 394 MB, 413138944 bytes, 806912 sectors
12608 cylinders, 4 heads, 16 sectors/track
Units: sectors of 1 * 512 = 512 bytes

Device         Boot StartCHS    EndCHS        StartLBA     EndLBA    Sectors  Size Id Type
```

```
/dev/mmcblk2p1p1    0,1,1       1023,3,16          16     204815     204800  100M 83 Linux
/dev/mmcblk2p1p2    1023,3,16   1023,3,16      204816     806911     602096  293M 83 Linux
```

図 4.36 fdisk コマンドによるパーティション変更

❶ 現在のパーティション構成を表示します。

❷ 現在のパーティションを削除します。

❸ 新しくパーティションを作成します。

❹ プライマリーパーティションを選択します。

❺ パーティション 1 を作成します。

❻ パーティション 1 のセクターの始まりを位置を指定します。

❼ パーティション 1 の最後のセクターの位置を指定します。

❽ 同様にパーティション 2 を作成します。

❾ 上記で設定したパーティション構成を microSD カードに書き込みます。

❿ 作成したパーティション構成を確認します。

fdisk コマンドの詳細な使い方は、man ページ等を参照してください。

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/mmcblk2p1
```

図 4.37 EXT4 ファイルシステムの構築

4.5.2. コンテナ内からストレージを使用する

ここでは、 sd_example という名称の alpine ベースのコンテナを作成し、その中で microSD カードを使用します。CON1 に microSD カードを挿入してください。

はじめに alpine コンテナイメージをダウンロードします。

```
[armadillo ~]# podman pull docker.io/alpine
Trying to pull docker.io/library/alpine:latest...
Getting image source signatures
Writing manifest to image destination
8d591b0b7dea080ea3be9e12ae563eebf9869168ffced1cb25b2470a3d9fe15e
```

図 4.38 alpine コンテナイメージをダウンロードする

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

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

図 4.39 sd_example.conf の内容

❶ add_hotplugs に mmc を指定することで、 コンテナ内で microSD カードをホットプラグで認識します

❷ コンテナ内で microSD カードをマウントするための権限を与えます

コンテナを起動し、コンテナの中に入ります。

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

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

図 4.40 sd_example.conf に基づきコンテナを生成

コンテナ内で microSD カードは、 /dev/mmcblk2 として認識されますので /mnt にマウントします。

```
[container]# mount /dev/mmcblk2p1 /mnt
```

図 4.41 コンテナ内で microSD カードをマウント

4.6. 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/

4.6.1. 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-900 開発セット 購入製品登録

https://armadillo.atmark-techno.com/armadillo-900/register

5. 動作確認方法

5.1. ABOS Web を用いたネットワーク設定方法

この節では、ABOS Web を用いたネットワーク設定を行います。

コマンドラインによるネットワーク設定を行いたい場合は「5.2. コマンドラインを用いたネットワーク設定方法」をご参照ください。

5.1.1. ABOS Web とは

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

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

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

ABOS Web で設定できる項目はネットワーク関連以外にもありますが、それらについては「10.9. Web UI から Armadillo をセットアップする (ABOS Web)」で紹介します。

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

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

5.1.2. ABOS Web へのアクセス

Armadillo と PC を有線 LAN で接続し、Armadillo の電源を入れて PC で Web ブラウザを起動した後、Web ブラウザのアドレスバーに次の URL を入力してください： https://armadillo.local:58080

ABOS Web は、初期状態では同一サブネットのネットワークのみアクセス可能です。
サブネット外からのアクセスを許可したい場合は、 /etc/atmark/abos_web/init.conf を作成し、ABOS Web のサービスを再起動してください。

以下の例ではコンテナとループバックからのアクセスのみを許可します：

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

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

5.1.3. ABOS Web のパスワード登録

「7.1.3.1. initial_setup.swu の作成」で ABOS Web のログイン用パスワードを設定していない場合、 ABOS Web 初回ログイン時に、 "初回ログイン"のパスワード登録画面が表示されますので、パスワードを設定してください。

図 5.1 パスワード登録画面

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

図 5.2 パスワード登録完了画面

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

ABOS Web に対応した Armadillo が正常に起動していれば、Web ブラウザに ABOS Web のログイン画面が表示されます。

図 5.3 ログイン画面

ログイン画面で ABOS Web のパスワードを入力して認証されれば、ABOS Web の設定画面に表示が変わり、設定操作を行うことができます。

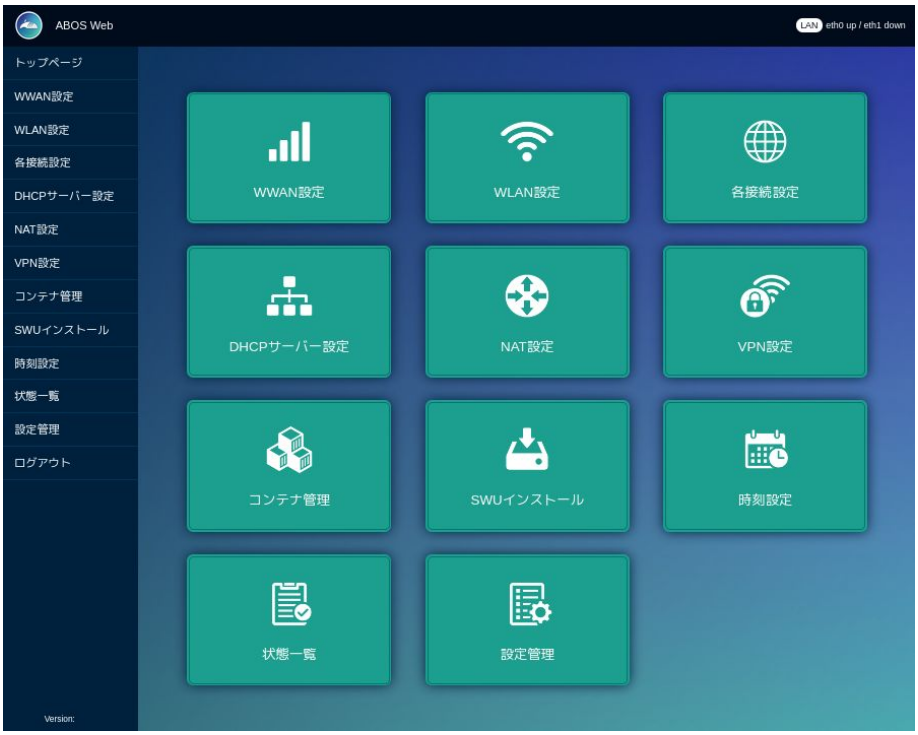


図 5.4 トップページ

5.1.4. ABOS Web のパスワード変更

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

図 5.5 ログイン画面

5.1.5. ABOS Web の設定操作

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

5.1.6. ログアウト

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

5.1.7. WWAN 設定

LTE をはじめとする WWAN 通信モジュールを搭載した Armadillo の、WWAN 設定を行います。この設定画面では、WWAN 接続設定の登録と、WWAN 接続の状態（現在のアドレス情報）の表示、登録済み WWAN 接続設定の編集と削除を行うことができます。設定項目のうち、"MCC/MNC" は、通常は空欄にしてください。MCC/MNC 以外の項目を正しく設定しても WWAN 通信が動作しない場合、特に SIM カードがマルチキャリア SIM の場合は、ご契約の通信事業者に MCC/MNM を問い合わせ、通信事業者から提示された MCC/MNC の値を設定してください。それぞれの入力フィールドに設定値を入力して "接続して保存" ボタンをクリックすると、WWAN 接続の設定を登録して、WWAN 接続動作を実行します。WWAN 通信設定が行われ、ネットワーク接続が確立した状態では、割当たっている IP アドレスなどを "現在の WWAN 接続情報" に表示します。「図 5.6. WWAN 設定画面」に、WWAN 設定を行った状態を示します。

図 5.6 WWAN 設定画面

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

5.1.8. WLAN 設定

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

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

5.1.8.1. WLAN 設定（クライアントとしての設定）

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

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

「図 5.7. WLAN クライアント設定画面」に、WLAN クライアント設定を行った状態を示します。

図 5.7 WLAN クライアント設定画面

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

5.1.8.2. WLAN 設定（アクセスポイントとしての設定）

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

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

「図 5.8. WLAN アクセスポイント設定画面」に、WLAN アクセスポイント設定を行った状態を示します。

図 5.8 WLAN アクセスポイント設定画面

アクセスポイントモードのセキュリティ方式は、WPA2 を使用します。

5.1.9. 各接続設定（各ネットワークインターフェースの設定）

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

図 5.9 現在の接続情報画面

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

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

5.1.9.1. LAN 接続設定

LAN 接続の接続名は、デフォルトでは "Wired connection 1" です。LAN ポートを二つ搭載した Armadillo では、二つめの LAN ポートに対応する "Wired connection 2" も有効です。Armadillo を LAN と WWAN との間で IPv4 ルーターとして運用する場合は、LAN 接続の設定で IPv4 アドレスを固

定 IP アドレスに設定して下さい。「図 5.10. LAN 接続設定で固定 IP アドレスに設定した画面」に、LAN 接続の設定編集画面で固定 IP アドレスに設定した状態を示します。

図 5.10 LAN 接続設定で固定 IP アドレスに設定した画面

5.1.9.2. WWAN 接続設定

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

5.1.9.3. WLAN 接続設定

WLAN 接続の接続名は、デフォルトでは、クライアントモードが "abos_web_wlan"、アクセスポイントモードが "abos_web_br_ap" です。

5.1.10. DHCP サーバー設定

ネットワークインターフェースごとに、接続したネットワーク上で Armadillo を DHCP サーバーとして動作させる設定を行うことができます。接続済みの DHCP サーバー情報を、画面上部の"現在の

DHCP 情報"に表示します。DHCP サーバーの設定を登録する場合は、"DHCP 情報入力"欄に設定内容を入力して"設定"ボタンをクリックしてください。「図 5.11. eth0 に対する DHCP サーバー設定」に、一つめの LAN ポート（eth0）に対する設定を行った状態を示します。

図 5.11 eth0 に対する DHCP サーバー設定

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

5.1.11. NAT 設定

この設定画面では、ルーター機能での宛先インターフェース設定と、Armadillo を接続した LAN 上の機器用のポートフォワーディング設定を行うことができます。Armadillo を LAN や WLAN と WWAN との間でルーターとして運用する場合は、NAT 設定の宛先インターフェースを WWAN のインターフェースに設定してください。そして、LAN や WLAN 上の機器を、WWAN 接続したインターネットにサーバーとして公開したい場合は、ポートフォワーディング設定を使ってください。

5.1.11.1. NAT 設定

ルーター機能での宛先インターフェース設定を行なえます。「図 5.12. LTE を宛先インターフェースに指定した設定」に、宛先インターフェースに ppp0 を指定した場合の画面を示します。

図 5.12 LTE を宛先インターフェースに指定した設定

5.1.11.2. ポートフォワーディング設定

受信インターフェースごとに、ポートフォワーディング設定を登録できます。「図 5.13. LTE からの受信パケットに対するポートフォワーディング設定」 に、受信インターフェース ppp0 について、ポート 8080 番宛の tcp パケットをポートフォワーディングする設定を行った状態を示します。

図 5.13 LTE からの受信パケットに対するポートフォワーディング設定

5.1.12. VPN 設定

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

「図 5.14. VPN 設定」に、VPN 接続設定を行った状態を示します。

図 5.14 VPN 設定

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

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

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

5.1.13. 状態一覧

各設定画面で行った設定の現在状態を、設定ごとに区切って一覧表示します。

5.2. コマンドラインを用いたネットワーク設定方法

基本的に、 Armadillo-900 開発セット のネットワーク設定は、「5.1. ABOS Web を用いたネットワーク設定方法」で紹介したとおり、 ABOS Web で行います。しかし、 ABOS Web で対応できない複雑なネットワーク設定を行いたい場合などは、コマンドラインからネットワークの設定を行うことも可能です。

ここでは、コマンドラインによるネットワークの設定方法について説明します。

5.2.1. 接続可能なネットワーク

表 5.1 ネットワークとネットワークデバイス

ネットワーク	搭載モデル	ネットワークデバイス	出荷時の設定
Ethernet	全モデル	eth0	DHCP
LTE	Cat.1 bis	ppp0	無し
無線 LAN	Cat.1 bis[a], WLAN	mlan0	クライアントモード

[a]型番によっては、搭載/非搭載が異なります。

5.2.2. IP アドレスの確認方法

Armadillo-900 開発セット 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 fq_codel state UP qlen 1000
    link/ether 00:11:0c:00:0c:7d brd ff:ff:ff:ff:ff:ff
    inet 172.16.1.73/16 brd 172.16.255.255 scope global dynamic noprefixroute eth0
       valid_lft 7183sec preferred_lft 7183sec
    inet6 fe80::a32e:4e20:df35:b238/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: can0: <NOARP40000> mtu 16 qdisc noop state DOWN qlen 10
    link/[280]
4: mlan0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN qlen 1000
    link/ether 52:b6:62:ac:78:a8 brd ff:ff:ff:ff:ff:ff
5: uap0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether fe:84:a7:51:db:d2 brd ff:ff:ff:ff:ff:ff
```

図 5.15 IP アドレスの確認

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

```
[armadillo ~]# ip addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP qlen 1000
    link/ether 00:11:0c:00:0c:7d brd ff:ff:ff:ff:ff:ff
    inet 172.16.1.73/16 brd 172.16.255.255 scope global dynamic noprefixroute eth0
       valid_lft 7062sec preferred_lft 7062sec
    inet6 fe80::a32e:4e20:df35:b238/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

図 5.16 IP アドレス(eth0)の確認

5.2.3. ネットワークの設定方法

Armadillo-900 開発セット 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。

5.2.3.1. 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 5.17. 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 5.17 nmcli のコマンド書式

5.2.4. 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。

5.2.4.1. コネクションの一覧表示

登録されているコネクションの一覧表示するには、「図 5.18. コネクションの一覧表示」に示すコマンドを実行します。[1]

[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
NAME                UUID                                  TYPE      DEVICE
Wired connection 1  xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx  ethernet  eth0
```

図 5.18 コネクションの一覧表示

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。

5.2.4.2. 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、「図 5.19. コネクションの有効化」に示すコマンドを実行します。

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

図 5.19 コネクションの有効化

コネクションを無効化するには、「図 5.20. コネクションの無効化」に示すコマンドを実行します。

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

図 5.20 コネクションの無効化

5.2.4.3. コネクションの作成

コネクションを作成するには、「図 5.21. コネクションの作成」に示すコマンドを実行します。

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

図 5.21 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-900 開発セット 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 「4.3. 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 5.22 コネクションファイルの永続化

別の Armadillo-900 開発セット からコネクションファイルをコピーした場合は、コネクションファイルのパーミッションを 600 に設定してください。600 に設定後、 nmcli c reload コマンドでコネクションファイルを再読込します。

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

swu イメージを使用してコネクションファイルのアップデートを行う場合は、swu イメージに含めるコネクションファイルのパーミッションを 600 に設定してから、swu イメージを作成してください。アップデート実行時には swu イメージ作成時のパーミッションが維持されるため、上記のコマンド実行手順は不要です。swu イメージに関しては 「7.3.3.6. SWU イメージのインストール」 を参考にしてください。

5.2.4.4. コネクションの削除

コネクションを削除するには、「図 5.23. コネクションの削除」に示すコマンドを実行します。

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

図 5.23 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 5.24 コネクションファイル削除時の永続化

5.2.4.5. 固定 IP アドレスに設定する

「表 5.2. 固定 IP アドレス設定例」の内容に設定する例を、「図 5.25. 固定 IP アドレス設定」に示します。

表 5.2 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 5.25 固定 IP アドレス設定

5.2.4.6. DHCP に設定する

DHCP に設定する例を、「図 5.26. DHCP の設定」に示します。

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

図 5.26 DHCP の設定

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

5.2.4.7. DNS サーバーを指定する

DNS サーバーを指定する例を、「図 5.27. 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 5.27 DNS サーバーの指定

5.2.4.8. 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 5.28 コネクションの修正の反映

5.2.4.9. デバイスの一覧表示

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、「図 5.29. デバイスの一覧表示」に示す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 5.29 デバイスの一覧表示

5.2.4.10. デバイスの接続

デバイスを接続するには、「図 5.30. デバイスの接続」に示すコマンドを実行します。

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

図 5.30 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存在するかを確認してください。

5.2.4.11. デバイスの切断

デバイスを切断するには、「図 5.31. デバイスの切断」に示すコマンドを実行します。

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

図 5.31 デバイスの切断

5.2.5. 有線 LAN の接続を確認する

有線 LAN で正常に通信が可能かを確認します。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 -c 3 192.0.2.20
PING 192.0.2.20 (192.0.2.20): 56 data bytes
64 bytes from 192.0.2.20: seq=0 ttl=64 time=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 5.32 有線 LAN の PING 確認

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。確実に有線 LAN の接続確認をするために、有線 LAN 以外のインターフェースを無効化してください。

5.2.6. LTE

本章では、Armadillo-900 開発セット に搭載されている LTE モジュールの使用方法について説明します。

Armadillo-900 開発セット に搭載しております SIMCom 製 LTE 通信モジュール SIM7672G は、ドコモの相互接続性試験を完了しています。ド

コモの相互接続性試験を完了した SIM7672G のファームウェアバージョンは 2388B01SIM767XM5A_M です。SIM7672G のファームウェアのバージョンは以下のように確認できます。「Revision:」がバージョンになります。

```
[armadillo ~]# send-at /dev/ttyLP1 AT+SIMCOMATI echo sim7672
AT+SIMCOMATI
Manufacturer: SIMCOM INCORPORATED
Model: SIM7672G-MNGV
Revision: 2388B01SIM767XM5A_M
SIM767XM5_B01V02_250409
IMEI: 865031061369335
OK
```

上記のバージョンでない場合は、ファームウェアのアップデートをお願いします。ファームウェア本体、およびアップデート手順は https://armadillo.atmark-techno.com/resources/software/partner-solution/sim7672g-software で公開しています。

上記のバージョン以外のファームウェアを使用しても技適違反にはなりませんが、通信トラブル等があった際にキャリアのサポートが得られない等の可能性があります。

5.2.6.1. LTE データ通信設定を行う前に

LTE データ通信を利用するには、通信事業者との契約が必要です。契約時に通信事業者から貸与された nanoSIM(UIM カード)と APN 情報を準備します。

Armadillo-900 開発セット 動作検証済み nanoSIM (料金プラン)に関しては、 Armadillo サイトの「Armadillo-IoT ゲートウェイ 動作確認済み SIM 一覧」を確認ください。

Armadillo-IoT ゲートウェイ 動作確認済み SIM 一覧 [https://armadillo.atmark-techno.com/howto/armadillo-iot-tested-sim]

Armadillo-900 開発セット の電源が切断されていることを確認してから nanoSIM(UIM カード)を取り付けてください。

本製品は、nanoSIM スロットを搭載しています。

標準/microSIM サイズの SIM カードを nanoSIM サイズにカットしたもの、サイズの異なるものを使用すると、nanoSIM スロットが故障する原因となります。これらを使用し本製品が故障した場合は、保証期間内であっても保証適用外となります。

nanoSIM(UIM カード)の切り欠きを挿入方向に向け、刻印面を上にして挿入してください。挿入位置などは、「図 5.91. LTE 用外付けアンテナ接続/nanoSIM カード挿入」 を参照してください。

APN の設定を行うには、「表 5.3. APN 設定情報」に示す情報が必要です。各設定値の文字長を超える設定はできませんので、SIM の料金プランを選択する際にはご注意ください。

表 5.3 APN 設定情報

項目	Armadillo-900 開発セット (SIM7672G 搭載)
APN	最大 99 文字
ユーザー名	最大 64 文字
パスワード	最大 64 文字
認証方式	PAP または CHAP
PDP Type	IP のみをサポート

5.2.6.2. 省電力などの設定

LTE モジュール SIM7672G 起動時に設定する内容を、設定ファイル(/etc/atmark/sim7672-boot.conf)に記載します。

設定ファイルの記載例として、サンプルファイル(/etc/atmark/sim7672-boot.conf.example)がありますので、こちらをリネームまたはコピーしてご利用ください。

/etc/atmark/sim7672-boot.conf に設定できる内容を「表 5.4. sim7672-boot.conf の設定内容」に示します。

sim7672-boot.conf のフォーマットは以下の通りです。

- パラメータは、「パラメータ名=値」のフォーマットで記載してください。
- 行頭に # が存在する場合、その行を無視します。
- パラメーターが存在しない場合、その項目に関して何も設定をしません。

表 5.4 sim7672-boot.conf の設定内容

パラメーター名	初期値	設定可能値	説明
psm	disable	disable または tau,act-time	Power Save Mode の設定
edrx	disable	disable または eDRX の値	eDRX の設定

PSM (Power Save Mode) の設定値を「表 5.5. psm の tau と act-time に設定可能な値」に示します。disable にしない場合、tau (Periodic TAU cycle (T3412)) は act_time (Active time (T3324)) より大きい値にする必要があります。

表 5.5 psm の tau と act-time に設定可能な値

パラメーター名	設定可能値
tau (s=秒,m=分,h=時間)	2s,4s,6s…62s,90s,120s,150s…930s,16m,17m,18m…31m,40m,50m,60m…310m,6h,7h,8h…31h,40h,50h,60h…310h,320h,640h,960h…9920h
act-time (s=秒,m=分,h=時間)	2s,4s,6s…62s,1m,2m,3m…31m,36m,42m,48m…186m

eDRX (extended Discontinuous Reception) の設定値を「表 5.6. edrx に設定可能な値」に示します。

PSM と eDRX はどちらかの設定が有効となります。両方共設定した場合は PSM が優先されます。

表 5.6 edrx に設定可能な値

パラメーター名	設定可能値
edrx (秒)	5.12, 10.24, 20.48, 40.96, 61.44, 81.92, 102.4, 122.88, 143.36, 163.84, 327.68, 655.36, 1310.72, 2621.44, 5242.88, 10485.76

5.2.6.3. LTE のコネクションを作成する

「表 5.7. APN 情報設定例」の内容に設定する例を「図 5.33. LTE のコネクションの作成」に示します。

表 5.7 APN 情報設定例

項目	設定
APN	[apn]
ユーザー名	[user]
パスワード	[password]
ネットワークデバイス	ttyCommModem

```
[armadillo ~]# nmcli connection add type gsm ifname ttyCommModem apn [apn] user [user] password
[password]
Connection 'gsm-[wwan]' (xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx) successfully added.
```

図 5.33 LTE のコネクションの作成

コネクション設定を永続化するには、以下のコマンドを入力してください。設定を永続化すると、Armadillo 起動時に自動的にデータ接続を行うようになります。

同一インタフェースへの設定が複数存在する場合、 gsm-ttyCommModem-1.nmconnection など後ろに数値が付与されますので、「図 5.33. LTE のコネクションの作成」 入力時のメッセージで生成されたファイル名を確認した上で永続化を実施ください。

```
[armadillo ~]# persist_file /etc/NetworkManager/system-connections/gsm-ttyCommModem.nmconnection
```

図 5.34 LTE のコネクションの設定の永続化

5.2.6.4. ユーザー名とパスワード設定が不要な LTE のコネクションを作成する

ユーザー名とパスワード設定が不要な SIM カードをご利用の場合、「図 5.35. ユーザー名とパスワード設定が不要な LTE のコネクションの作成」 に示すとおり user と password を省略して設定してください。

```
[armadillo ~]# nmcli connection add type gsm ifname ttyCommModem apn [apn]
Connection 'gsm-[wwan]' (xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx) successfully added.
```

図 5.35 ユーザー名とパスワード設定が不要な LTE のコネクションの作成

5.2.6.5. MCC/MNC を指定した LTE のコネクションを作成する

マルチキャリア SIM などを使用する際、MCC (Mobile Country Code) と MNC (Mobile Network Code) を指定してコネクションを作成すると LTE ネットワークに接続できることがあります。指定する場合は 「図 5.36. MCC/MNC を指定した LTE コネクションの作成」 に示すコマンドを実行してください。

[mccmnc] には 44010 などの数字を入力してください。実際に設定する値に関しては、ご契約の通信事業者へお問い合わせください。

```
[armadillo ~]# nmcli connection add type gsm ifname ttyCommModem apn [apn] user [user] password
[password] gsm.network-id [mccmnc]
```

図 5.36 MCC/MNC を指定した LTE コネクションの作成

5.2.6.6. PAP 認証を有効にした LTE のコネクションを作成する

LTE のコネクションの認証方式は、デフォルトで CHAP に設定されています。PAP 認証を有効にしたコネクションを作成する場合は「図 5.37. PAP 認証を有効にした LTE コネクションの作成」 に示すコマンドを実行してください。

```
[armadillo ~]# nmcli connection add type gsm ifname ttyCommModem apn [apn] user [user] password
[password] ppp.refuse-eap true ppp.refuse-chap true ppp.refuse-mschap true ppp.refuse-mschapv2 true
ppp.refuse-pap false
```

図 5.37 PAP 認証を有効にした LTE コネクションの作成

すでに LTE コネクションを作成済みの場合はコネクション設定を削除した後に、「図 5.37. PAP 認証を有効にした LTE コネクションの作成」を実施してください。

5.2.6.7. LTE コネクションを確立する

LTE コネクションの作成直後や設定変更後に再起動をせずにコネクションを確立するには、「図 5.38. LTE のコネクション確立」に示すコマンドを実行します。

```
[armadillo ~]# nmcli connection up gsm-ttyCommModem
Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkManager/
ActiveConnection/x)
```

図 5.38 LTE のコネクション確立

5.2.6.8. LTE の接続を確認する

ご利用になるサービスとの通信を検証する、ICMP に対応しているアドレス (8.8.8.8 など) と PING 通信を行うなどの方法で LTE の接続を確認してください。

```
[armadillo ~]# ping -c 3 8.8.8.8 -I [network device]
```

図 5.39 LTE の PING 確認

[network device] には、「表 5.1. ネットワークとネットワークデバイス」を参照し、使用している LTE のネットワークデバイスを指定してください。

5.2.6.9. LTE コネクションを切断する

LTE コネクションを切断するには、「図 5.40. LTE コネクションを切断する」に示すコマンドを実行します。LTE コネクションを切断する前に、LTE 再接続サービスを停止しないと再接続処理が実行される為、事前に停止します。

```
[armadillo ~]# rc-service connection-recover stop ❶
connection-recover| * Stopping connection-recover ... [ ok ]
[armadillo ~]# nmcli connection down gsm-ttyCommModem ❷
```

図 5.40 LTE コネクションを切断する

❶ LTE 再接続サービスを停止します。

❷ LTE コネクションを切断します。

5.2.7. LTE 再接続サービス

LTE 再接続サービスは、LTE のデータ接続の状態を定期的に監視し、切断を検出した場合に再接続を行うサービスです。

初期状態でこのサービスが有効になっております。

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

コネクション状態確認時 PING 送付先の初期値は 8.8.8.8 ですが、この IP アドレスに対して ping 導通ができない場合、 ping 導通可能な IP アドレスを指定する必要があります。

SIM カードが挿入されており、NetworkManager に有効な LTE コネクションの設定がされているとき、初期設定では 120 秒に一度コネクションの状態を監視します。オプションで SIM カードの認識ができないときに Armadillo の再起動を実施することも可能です。

コネクションが無効になっている場合、切断状態と判定しコネクションを有効にします。

コネクションが有効になっている場合、特定の宛先に PING を実行します。PING がエラーになったとき切断状態と判定し、コネクションの無効化・有効化を行うことで再接続を実施します。

コネクションの無効化・有効化による再接続を実施しても PING がエラーになる場合、電波のオン・オフまたは LTE モジュールの電源をオン・オフを実施して LTE 再接続を試みます。どちらを実施するかは設定ファイルの WWAN_FORCE_RESTART_COUNT に依存します。

WWAN_FORCE_RESTART_COUNT が初期値の 10 である場合、1 から 9 回目は電波のオン・オフを実施し、10 回目は LTE モジュールの電源オン・オフを実施します。それ以降も NG が続く場合、同じく 10 回に一度 LTE モジュールの電源オン・オフを実施します。

LTE モジュールが検出できない状態が 2 回連続で発生した場合、LTE モジュールの再起動を実施します。

LTE 接続中状態が 3 回連続で発生した場合、LTE モジュールの再起動を実施します。

工場出荷状態で本サービスは有効化されており、システム起動時にサービスが自動的に開始されます。PING を実行する宛先は、初期設定では "8.8.8.8" です。

設定ファイルの記載例として、サンプルファイル(/etc/atmark/connection-recover.conf.example)がありますので、こちらをリネームまたはコピーしてご利用ください。

設定ファイルの概要を「表 5.8. 再接続サービス設定パラメーター」に示します。必要に応じて設定値を変更してください。

設定ファイルが存在しない場合は初期値で動作します。

表 5.8 再接続サービス設定パラメーター

パラメーター名	初期値	意味	変更
PRODUCT_NAME	-	製品名	不可
CHECK_INTERVAL_SEC	120	監視周期(秒)	可
PING_DEST_IP	8.8.8.8	コネクション状態確認時 PING 送付先	可
DEVICE	-	ネットワークデバイス名	不可
TYPE	-	ネットワークタイプ	不可
NETWORK_IF	-	ネットワーク I/F 名	不可
FORCE_REBOOT	FALSE	TRUE に設定すると PING 導通チェックが 4 回連続 NG だった場合、 Armadillo を再起動します。	可
REBOOT_IF_SIM_NOT_FOUND	FALSE	TRUE に設定すると SIM を検出できない状態が 2 回連続で発生した場合、 Armadillo を再起動します。	可
WWAN_FORCE_RESTART_COUNT	10	PING 導通確認を設定した回数連続で失敗した場合 LTE モジュールを再起動します。設定した回数に満たない場合 、電波のオフ・オン実施のみで LTE 再接続を試みます。	可

設定ファイル変更後、変更内容を永続化するには「図 5.41. LTE 再接続サービスの設定値を永続化する」に示すコマンドを実行してください。

```
[armadillo ~]# persist_file /etc/atmark/connection-recover.conf
```

図 5.41 LTE 再接続サービスの設定値を永続化する

LTE 再接続サービスの状態を確認するには、「図 5.42. LTE 再接続サービスの状態を確認する」に示すコマンドを実行してください。

```
[armadillo ~]# rc-service connection-recover status
 * status: started
```

図 5.42 LTE 再接続サービスの状態を確認する

LTE 再接続サービスを停止するには、「図 5.43. LTE 再接続サービスを停止する」に示すコマンドを実行してください。

```
[armadillo ~]# rc-service connection-recover stop
connection-recover| * Stopping connection-recover ... [ ok ]
```

図 5.43 LTE 再接続サービスを停止する

LTE 再接続サービスを開始するには、「図 5.44. LTE 再接続サービスを開始する」に示すコマンドを実行してください。

```
[armadillo ~]# rc-service connection-recover start
connection-recover| * Starting connection-recover ... [ ok ]
```

図 5.44 LTE 再接続サービスを開始する

独自に接続状態を確認するサービスを実装されるなどの理由で標準の LTE 再接続サービスが不要な場合、「図 5.45. LTE 再接続サービスを無効にする」に示す手順で再接続サービスを永続的に無効にできます。

```
[armadillo ~]# rc-service connection-recover stop ❶
connection-recover| * Stopping connection-recover ... [ ok ]
[armadillo ~]# rc-update del connection-recover default ❷
service connection-recover removed from runlevel default
[armadillo ~]# persist_file -d /etc/runlevels/default/connection-recover ❸
```

図 5.45 LTE 再接続サービスを無効にする

❶ 再接続サービスを停止します。

❷ 再接続サービスを無効にします。

❸ サービス設定ファイルの削除を永続化します。

LTE 再接続サービスを無効化した後、再度有効にする場合、「図 5.46. LTE 再接続サービスを有効にする」に示す手順を実行してください。

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

図 5.46 LTE 再接続サービスを有効にする

❶ 再接続サービスを有効にします。

❷ 再接続サービスを開始します。

❸ サービス設定ファイルを永続化します。

5.2.7.1. ModemManager - mmcli について

ここでは ModemManager と mmcli について説明します。

Armadillo-900 開発セット にはネットワークを管理する NetworkManager とは別に、モデムを管理する ModemManager がインストールされています。ModemManager はモバイルブロードバンドデバイス(LTE モジュールなど)の操作および、接続状況の管理などを行います。

ModemManager のコマンドラインツールである **mmcli** を使用することで、LTE 通信の電波強度や SIM カードの情報(電話番号や IMEI など)を取得することが可能です。mmcli の詳しい使いかたについては **man mmcli** を参照してください。

ModemManager はモデムデバイスに応じたプラグインを選択して動作します。 Armadillo-900 開発セット では simtech-sim7672 という名称のプラグインで動作しています。

5.2.7.2. mmcli - 認識されているモデムの一覧を取得する

認識されているモデムの一覧を取得するには、「図 5.47. 認識されているモデムの一覧を取得する」に示すコマンドを実行します。

Armadillo Base OS では、Armadillo Base OS が使用している LTE モジュール番号を取得するコマンド **mm-modem-num** を用意しております。

```
[armadillo:~]# mmcli -L
    /org/freedesktop/ModemManager1/Modem/0 [SIMCOM INCORPORATED] SIM7672G-MNGV
```

図 5.47 認識されているモデムの一覧を取得する

5.2.7.3. mmcli - モデムの情報を取得する

モデムの情報を取得するには、「図 5.48. モデムの情報を取得する」に示すコマンドを実行します。

```
armadillo:~# mmcli -m $(mm-modem-num)
  --------------------------------
  General  |                 path: /org/freedesktop/ModemManager[number1]/Modem/[number2]
           |            device id: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  --------------------------------
  Hardware |         manufacturer: SIMCOM INCORPORATED
           |                model: SIM7672G-MNGV
           |    firmware revision: XXXXXXXXXXXXXXXXXXX
           |            supported: lte
           |              current: lte
           |         equipment id: XXXXXXXXXXXXXXX
:（省略）
```

図 5.48 モデムの情報を取得する

モデムの情報を取得するには、SIM カードが取り付けられている必要があります。正しく SIM カードが取り付けられていることを確認してください。

5.2.7.4. mmcli - SIM の情報を取得する

SIM の情報を取得するには、「図 5.49. SIM の情報を取得する」に示すコマンドを実行します。

```
[armadillo ~]# mmcli -m $(mm-modem-num)
:（省略）
```

```
SIM      |     primary sim path: /org/freedesktop/ModemManager1/SIM/[number] # [number] を次のコマ ⏎
ンドで使用
:（省略）

[armadillo ~]# mmcli -i [number]
  ------------------------------
  General    |              path: /org/freedesktop/ModemManager1/SIM/0
  ------------------------------
  Properties |            active: yes
             |              imsi: XXXXXXXXXXXXXX
             |             iccid: XXXXXXXXXXXXXXXXXX
             |       operator id: XXXXX
             |     operator name: XXXXXXXXXX
```

図 5.49 SIM の情報を取得する

5.2.7.5. mmcli - 回線情報を取得する

回線情報を取得するには、「図 5.50. 回線情報を取得する」に示すコマンドを実行します。

```
[armadillo ~]# mmcli -m $(mm-modem-num)
:（省略）
  Bearer   |               paths: /org/freedesktop/ModemManager1/Bearer/[number] # [number] を次の ⏎
コマンドで使用
:（省略）

[armadillo ~]# mmcli -b [number]
  ------------------------------------
  General            |           path: /org/freedesktop/ModemManager1/Bearer/[bearer number]
                     |           type: default
  ------------------------------------
  Status             |      connected: yes
                     |      suspended: XX
                     |    multiplexed: XX
                     |     ip timeout: XX
  ------------------------------------
  Properties         |            apn: XXXXXXXXXX
                     |        ip type: XXXXX
```

図 5.50 回線情報を取得する

5.2.8. 無線 LAN

本章では、Armadillo-900 開発セット に搭載されている無線 LAN モジュールの使用方法について説明します。

例として、WPA2-PSK(AES)のアクセスポイントに接続します。WPA2-PSK(AES)以外のアクセスポイントへの接続方法などについては、man nm-settings を参考にしてください。また、以降の説明では、アクセスポイントの ESSID を[essid]、パスフレーズを[passphrase]と表記します。

5.2.8.1. 無線 LAN アクセスポイントに接続する

無線 LAN アクセスポイントに接続するためには、次のようにコマンドを実行してコネクションを作成します。

```
[armadillo ~]# nmcli device wifi connect [essid] password [passphrase]
```

図 5.51 無線 LAN アクセスポイントに接続する

作成されたコネクションの ID は nmcli connection コマンドで確認できます。

```
[armadillo ~]# nmcli connection
NAME                UUID                                  TYPE      DEVICE
[essid]             e051a1df-6bd7-4bcf-9c71-461af666316d  wifi      mlan0
Wired connection 1  f147b8e8-4a17-312d-a094-8c9403007f6a  ethernet  --
```

図 5.52 無線 LAN のコネクションが作成された状態

NetworkManager の仕様により、無線 LAN の接続にはランダムな MAC アドレスが使用されます。搭載されている無線 LAN モジュール固有の MAC アドレスを使用したい場合は、以下の例のように NetworkManager の設定を変更し、再起動を行ってください。

```
[armadillo ~]# echo "[device-mac-randomization]" > /etc/NetworkManager/
conf.d/no-random-mac.conf
[armadillo ~]# echo "wifi.scan-rand-mac-address=no" >> /etc/
NetworkManager/conf.d/no-random-mac.conf
[armadillo ~]# echo "[connection-mac-randomization]" >> /etc/
NetworkManager/conf.d/no-random-mac.conf
[armadillo ~]# echo "wifi.cloned-mac-address=permanent" >> /etc/
NetworkManager/conf.d/no-random-mac.conf
[armadillo ~]# persist_file /etc/NetworkManager/conf.d/no-random-
mac.conf
```

5.2.8.2. 無線 LAN の接続を確認する

無線 LAN で正常に通信が可能か確認します。

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 192.0.2.20
```

図 5.53 無線 LAN の PING 確認

無線 LAN 以外のコネクションが有効化されている場合、ネットワーク通信に無線 LAN が使用されない場合があります。確実に無線 LAN の接続確認をする場合は、事前に無線 LAN 以外のコネクションを無効化してください。

5.2.9. 無線 LAN アクセスポイント (AP) として設定する

無線 LAN をアクセスポイント (以降 AP) として設定する方法を説明します。AP 設定は hostapd というソフトウェアと、 DNS/DHCP サーバである dnsmasq というソフトウェアを使用します。

hostapd と dnsmasq は Armadillo Base OS にデフォルトでインストール済みとなっているため、インストール作業は不要です。インストールされていない場合は、 Armadillo Base OS を最新バージョンに更新してください。

アクセスポイントモード (AP) と ステーションモード (STA) の同時利用はできません。

5.2.9.1. bridge インターフェースを追加する

NetworkManager を使用し bridge インターフェース (br0) を追加します。同時に AP の IP アドレスも設定します。ここでは 192.0.2.1 を設定しています。

```
[armadillo ~]# nmcli con add type bridge ifname br0
[armadillo ~]# nmcli con mod bridge-br0 ipv4.method manual ipv4.address "192.0.2.1/24"
[armadillo ~]# nmcli con up bridge-br0
[armadillo ~]# persist_file /etc/NetworkManager/system-connections/bridge-br0.nmconnection ❶
```

図 5.54 bridge インターフェースを作成する

❶ 設定ファイルを永続化します。

また、NetworkManager のデフォルト状態では定期的に mlan0 のスキャンを行っています。スキャン中は AP の性能が低下するため mlan0 を NetworkManager の管理から外します。

```
[armadillo ~]# vi /etc/NetworkManager/conf.d/90_disable_mlan0.conf
[device_mlan0]
match-device=interface-name:mlan0
managed=0

[armadillo ~]# persist_file /etc/NetworkManager/conf.d/90_disable_mlan0.conf
[armadillo ~]# nmcli d set mlan0 managed no ❶
```

図 5.55 mlan0 インターフェースを NetworkManager の管理から外す

❶ nmcli で NetworkManager をリスタートせずに設定します。

5.2.9.2. hostapd を設定する

hostapd の設定ファイルの雛形として用意してある /etc/hostapd/hostapd.conf.example をコピーして使用します。

```
[armadillo ~]# cp /etc/hostapd/hostapd.conf.example /etc/hostapd/hostapd.conf
[armadillo ~]# vi /etc/hostapd/hostapd.conf
```

```
hw_mode=a ❶
channel=44 ❷
ssid=myap ❸
wpa_passphrase=myap_pass ❹
interface=uap0
bridge=br0
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
driver=nl80211
country_code=JP
ctrl_interface=/var/run/hostapd
ctrl_interface_group=0
disassoc_low_ack=1
preamble=1
wmm_enabled=1
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
ieee80211ac=1
ieee80211ax=1
ieee80211n=1
ieee80211d=1
ieee80211h=1
logger_syslog=-1
logger_syslog_level=2
logger_stdout=-1
logger_stdout_level=2

[armadillo ~]# persist_file /etc/hostapd/hostapd.conf ❺
[armadillo ~]# rc-service hostapd start ❻
[armadillo ~]# rc-update add hostapd ❼
[armadillo ~]# persist_file /etc/runlevels/default/hostapd ❽
```

図 5.56 hostapd.conf を編集する

❶ 5GHz であれば a を、2.4GHz であれば g を設定します。

❷ 使用するチャンネルを設定します。

❸ 子機から接続するための任意の SSID を設定します。この例では myap を設定しています。

❹ 子機から接続するための任意のパスフレーズを設定します。この例では myap_pass を設定しています。

❺ 設定ファイルを永続化します。

❻ hostapd を起動します。

❼ Armadillo 起動時に hostapd が自動的に起動されるようにします。

❽ hostapd 自動起動の設定を永続化します。

5.2.9.3. dnsmasq を設定する

dnsmasq の設定ファイルを以下の内容で作成し /etc/dnsmasq.d/ 下に配置します。ファイル名は任意ですが、拡張子は .conf としてください。ここでは dhcp.conf としています。

```
[armadillo ~]# vi /etc/dnsmasq.d/dhcp.conf
interface=br0
bind-dynamic
dhcp-range=192.0.2.10, 192.0.1.2, 24h ❶

[armadillo ~]# persist_file /etc/dnsmasq.d/dhcp.conf ❷
[armadillo ~]# rc-service dnsmasq restart ❸
```

図 5.57 dnsmasq の設定ファイルを編集する

❶ 子機に割り当てる IP アドレスの範囲とリース期間を設定します。

❷ 設定ファイルを永続化します。

❸ dnsmasq を再起動します。

hostapd と dnsmasq の起動完了後、子機から hostapd.conf で設定した SSID とパスフレーズで接続できます。

5.2.10. ファイアウォールの設定方法

開放しているポートが存在すると攻撃者の標的になる可能性があります。開発したサーバーが使用するポートに対して、アクセスできる IP アドレスを制限することでセキュリティ上のリスクを低減することができます。

ここでは、iptables コマンドを使用した、パケットフィルタリングによるアクセス制限方法を紹介します。

デフォルトでは iptables サービスは無効になっています。サービスを有効にするためには以下のコマンドを実行してください。

```
[armadillo ~]# rc-update add iptables ❶
[armadillo ~]# persist_file /etc/runlevels/default/iptables ❷
```

❶ サービスを有効にします。

❷ サービスの有効を永続化します。

「図 5.58. 特定のポートに対する IP アドレスのフィルタリング」の例では、Armadillo の特定のポートに対して、特定の IP アドレスからのアクセスのみを受け入れるようにします。この例では、<送信元 IP アドレス> は Armadillo にパケットを送信する IP アドレス、<ポート番号> はパケットを受け入れる Armadillo のポート番号、<プロトコル> は通信プロトコルのことを指します。また、<ポート番号> はパケットを受け入れる Armadillo のポート番号のことを指します。

```
[armadillo ~]# iptables -I INPUT -s <送信元 IP アドレス> -p <プロトコル> --dport <ポート番号> -j ⏎
ACCEPT ❶
[armadillo ~]# iptables -A INPUT -p <プロトコル> --dport <ポート番号> -j REJECT ❷
[armadillo ~]# iptables -L ❸
```

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     〈プロトコル〉 --  〈送信元 IP アドレス〉 anywhere            〈プロトコル〉 dpt:〈ポート番号〉
REJECT     〈プロトコル〉 --  anywhere             anywhere             〈プロトコル〉 dpt:〈ポート番号
〉 reject-with icmp-port-unreachable
... 省略

[armadillo ~]# /etc/init.d/iptables save ❹
[armadillo ~]# persist_file /etc/iptables/rules-save ❺
```

図 5.58 特定のポートに対する IP アドレスのフィルタリング

❶ <ポート番号> に <送信元 IP アドレス> から送られてきたパケットを受け入れるように設定します。

❷ <ポート番号> に <送信元 IP アドレス> 以外から送信されてきたパケットを拒否するように設定します。

❸ 想定通りに設定されているか確認します。

❹ 上記の設定を設定ファイル /etc/iptables/rules-save に保存します。

❺ 保存した設定ファイルを永続化します。

「図 5.58. 特定のポートに対する IP アドレスのフィルタリング」はあくまで一例ですが、このように iptables コマンドを用いることで開発したサーバーにアクセスできる IP アドレスを制限することができます。

上記の設定を削除する場合は「図 5.59. 特定のポートに対する IP アドレスのフィルタリングの設定を削除」に示すコマンドを実行してください。

```
[armadillo ~]# armadillo:~# iptables -L --line-number ❶
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    ACCEPT     〈プロトコル〉 --  〈送信元 IP アドレス〉 anywhere            〈プロトコル〉 dpt:〈ポート番号〉
2    REJECT     〈プロトコル〉 --  anywhere             anywhere             〈プロトコル〉 dpt:〈ポー
ト番号〉 reject-with icmp-port-unreachable
... 省略

[armadillo ~]# iptables -D INPUT 2 ❷
[armadillo ~]# iptables -D INPUT 1 ❸
[armadillo ~]# /etc/init.d/iptables save ❹
[armadillo ~]# persist_file /etc/iptables/rules-save ❺
```

図 5.59 特定のポートに対する IP アドレスのフィルタリングの設定を削除

❶ 削除する設定の番号（num）を確認します。ここでは 1 番と 2 番の設定を削除します。

❷ 2 番の設定を削除します。

❸ 1 番の設定を削除します。

❹ 上記の設定を設定ファイル /etc/iptables/rules-save に保存します。

❺ 保存した設定ファイルを永続化します。

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

Armadillo Base OS では「表 5.9. Armadillo Base OS のデフォルトで開放しているポート」に示すポートをデフォルトで開放しています。

表 5.9 Armadillo Base OS のデフォルトで開放しているポート

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

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

ABOS Web のサービスを停止する方法は「10.9.9. ABOS Web を停止する」を、起動する方法は「10.9.10. ABOS Web を起動する」を参照してください。

「図 5.60. avahi-daemon を停止する」に avahi のサービスを停止する方法を示します。

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

図 5.60 avahi-daemon を停止する

❶ OpenRC に avahi のサービスが登録されていることを確認します。

❷ avahi のサービスが起動していることを確認します。

❸ avahi のサービスを停止します。

❹ サービスを管理している OpenRC から avahi のサービスの登録を解除します。

❺ サービス設定ファイルの削除を永続化します。

「図 5.61. avahi-daemon を起動する」に avahi サービスを起動する方法を示します。

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

図 5.61 avahi-daemon を起動する

❶ OpenRC に avahi のサービスが登録されていないことを確認します。

❷ サービスを管理している OpenRC に avahi のサービスを登録します。

❸ avahi のサービスを起動します。

❹ サービス設定ファイルを永続化します。

5.4. USB デバイスの接続を許可する

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

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

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

この機能について、CUI で実行したい場合は「10.25. 不正な USB デバイスの接続を拒否する」をご参照ください。

工場出荷状態における USB 接続制御機能

製品のセキュリティ向上のため、 ABOS バージョン 3.21.3-at.10 以降では、USB 接続制御機能がデフォルトで有効化されています。デフォルトで接続が許可されている USB デバイスは次の通りです。

表 5.10 デフォルトで許可されている USB デバイス

デバイス名/デバイスクラス名	利用目的
MassStorage クラス	SWUpdate で使用するため
Hub クラス	Armadillo 内部で使用しているため
LTE モジュール(SIM7672G)	製品として標準で搭載しているため

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

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

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

「図 5.62. USB 接続制御の設定画面」に設定画面を示します。

図 5.62 USB 接続制御の設定画面

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

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

「図 5.63. 接続済みの USB デバイス」に示すように、「接続済みの USB デバイス」では、現在接続されている USB デバイスが一覧として表示されます。

図 5.63 接続済みの USB デバイス

"詳細表示" ボタンを押すと、「図 5.64. 接続済み USB デバイスの詳細情報」に示す画面が表示されて、その USB デバイスに関するより詳細な情報が表示されます。

図 5.64 接続済み USB デバイスの詳細情報

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

block の場合、「図 5.65. USB デバイスを許可する」に表示されている "許可" ボタンを押すことで allow に変更できます。

図 5.65 USB デバイスを許可する

"許可" ボタンを押すと、「図 5.66. 許可済み USB デバイス」に示すように「許可済みの USB デバイス」の欄に許可ルールが追加されます。

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

図 5.66 許可済み USB デバイス

「図 5.67. 接続済みの個体と同じメーカー・製品の全ての個体を許可する」に示すように、「同じメーカー・製品の全ての個体を許可」のチェックボックスを選択した状態で "許可" ボタンを押すと、選択した USB デバイスと同じメーカー・製品であれば、どの個体であっても接続が許可されます。

図 5.67 接続済みの個体と同じメーカー・製品の全ての個体を許可する

その場合、「図 5.68. 全ての個体を許可する場合の表示」に示すように、「許可済みの USB デバイス」の「全ての個体を許可」の欄に「✓」が表示されます。

図 5.68 全ての個体を許可する場合の表示

「図 5.68. 全ての個体を許可する場合の表示」に示すように、USB デバイスの許可ルールを選択した状態で "削除" ボタンを押すと、「図 5.69. 許可ルールを削除する」に示す画面に遷移します。

図 5.69 許可ルールを削除する

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

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

「図 5.70. USB デバイスクラス」に表示されている「許可済みの USB デバイスクラス」では、許可する USB デバイスをデバイスクラス単位で設定することができます。

図 5.70 USB デバイスクラス

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

デバイスクラスを選択した後、"削除" ボタンを押すことでそのデバイスクラスの許可ルールを削除できます。削除の手順は「図 5.69. 許可ルールを削除する」と同様です。

"追加" ボタンを押すと、「図 5.71. USB デバイスクラスの追加」に示す画面が表示されます。追加可能な USB デバイスクラスを選択し、"追加" ボタンを押してください。

図 5.71 USB デバイスクラスの追加

5.5. 各インターフェースの使用方法

各デバイスの接続方法と、使用方法について紹介します。「図 5.72. Armadillo-900 開発セットのインターフェース」に Armadillo-900 開発セット のインターフェースを示します。

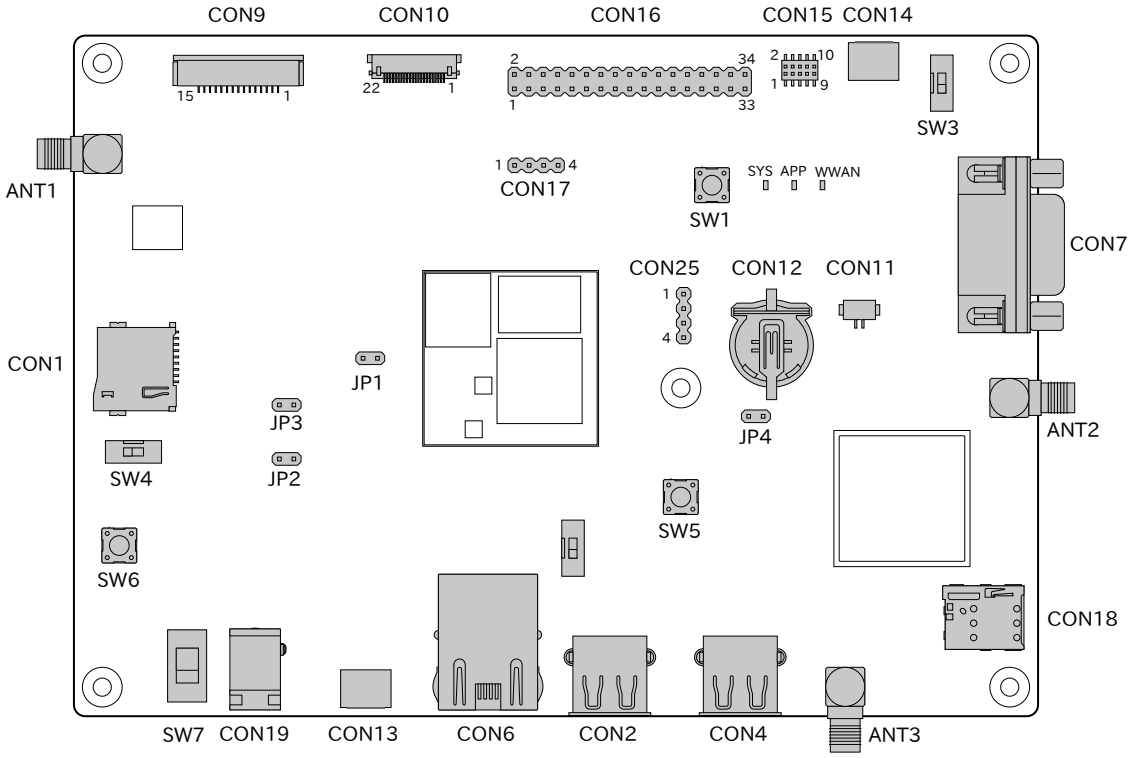


図 5.72 Armadillo-900 開発セットのインターフェース

表 5.11 Armadillo-900 開発セット 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	693071020811	Wurth Elektronik
CON2	USB インターフェース 1	SS-52100-001	Bel Fuse
CON4	USB インターフェース 2	SS-52100-001	Bel Fuse
CON6	LAN インターフェース	S26-KA-0009	U.D. ELECTRONIC
CON7	CAN インターフェース	CD6109P21G4	Cvilux
CON9	MIPI CSI インターフェース	1-84952-5	TE Connectivity
CON10	MIPI DSI インターフェース	SFV22R-2STBE1HLF	Amphenol
CON11	RTC バックアップインターフェース 1	DF13C-2P-1.25V(21)	Hirose Electric
CON12	RTC バックアップインターフェース 2	BH-44C-5	Adam Tech
CON13	APD 用コンソールインターフェース	CX90B-16P	Hirose Electric
CON14	RTD 用コンソールインターフェース	CX90B-16P	Hirose Electric
CON15	JTAG インターフェース	20021121-00010C1LF	Amphenol
CON16	拡張インターフェース	61303421121	Wurth Elektronik
CON17	I2C(3.3V)インターフェース	61300411121	Wurth Elektronik
CON18	nanoSIM インターフェース	SF72S006VBDR2500	Japan Aviation Electronics Industry
CON19	電源入力インターフェース	PJ-102AH	Same Sky
CON25	DAC インターフェース	61300411121	Wurth Elektronik
ANT1	WLAN/BT/TH アンテナインターフェース	SA5JP-LP001CG-39	CONNEKT PRECISION ELECTRONICS
ANT2	LTE アンテナインターフェース	SA5JJ-LP001PG-25	CONNEKT PRECISION ELECTRONICS
ANT3	GNSS アンテナインターフェース	SA5JJ-LP001PG-25	CONNEKT PRECISION ELECTRONICS
SYS	システム LED	SML-D12M1WT86	ROHM
APP	アプリケーション LED	SML-D12M1WT86	ROHM
WWAN	ワイヤレス WAN LED	SML-D12M1WT86	ROHM
SW1	ユーザースイッチ	PTS645SM43SMTR92LFS	C&K
SW3	RTD 用コンソール JTAG 切替スイッチ	DS01-254-S-01BE	Same Sky
SW4	ブートモード切替スイッチ	DS01-254-S-01BE	Same Sky
SW5	ONOFF 信号制御用スイッチ	PTS645SM43SMTR92LFS	C&K
SW6	リセットスイッチ	PTS645SM43SMTR92LFS	C&K
SW7	電源スイッチ	SLW-1276864-4A-D	Same Sky

5.5.1. 電源を入力する

付属の AC アダプタを CON19 電源入力インターフェースに接続し、SW7 電源スイッチを ON にして電源を入力します。

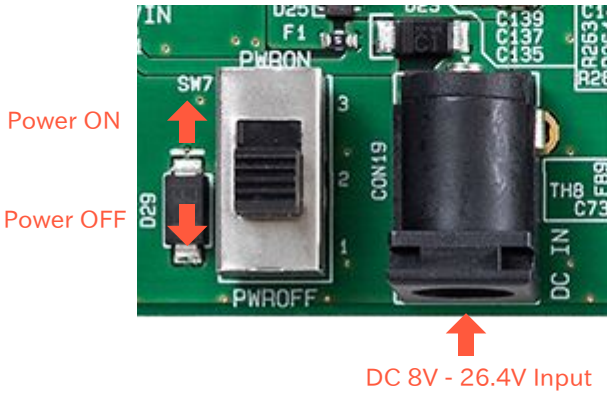


図 5.73 電源入力

付属の AC アダプタ以外を使用する場合は、「図 5.74. 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。定格入力電圧範囲は、DC8V～26.4V で、対応プラグは内径 2.1mm、外形 5.5mm のものとなります。

図 5.74 AC アダプタの極性マーク

AC アダプタの DC プラグを DC ジャックに接続してから、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 ※SW7 電源スイッチで切断した場合も同様
- 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。

5.5.2. SD カードを使用する

CON1 SD インターフェースに microSD/microSDHC/microSDXC カード（以下 microSD カード）を挿入することで、microSD カードが利用可能です。

機能
- カードタイプ: microSD/microSDHC/microSDXC
- バス幅: 1bit or 4bit

・スピードモード: Default Speed(24MHz), High Speed(49MHz), UHS-I (195MHz)

・カードディテクトサポート

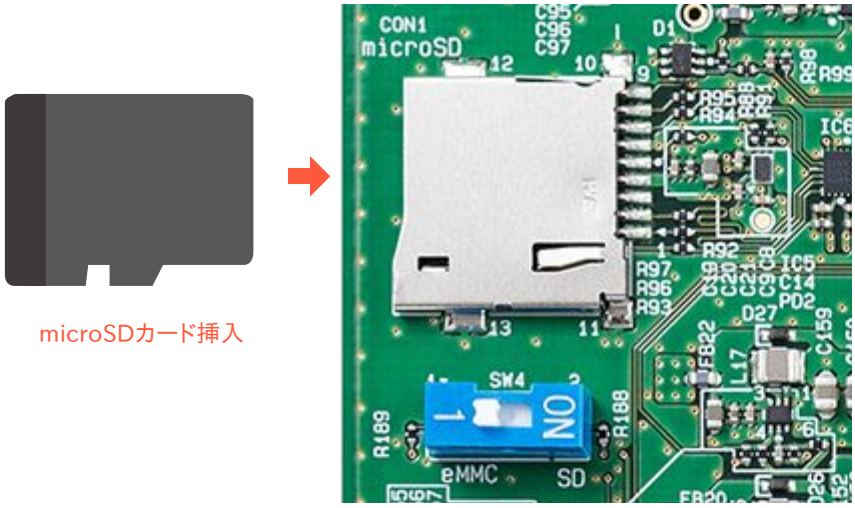


図 5.75 microSD カード挿入

5.5.2.1. ハードウェア仕様

SD インターフェースの回路は、microSD カードのバス電圧 1.8V/3.3V に対応するため 1.8V/3.3V 電圧セレクタとレベルシフタでバス電圧を切り替える構成になっています。また、microSD カードの電源(VDD_SD_3V3)はパワースイッチで ONOFF を制御しています。

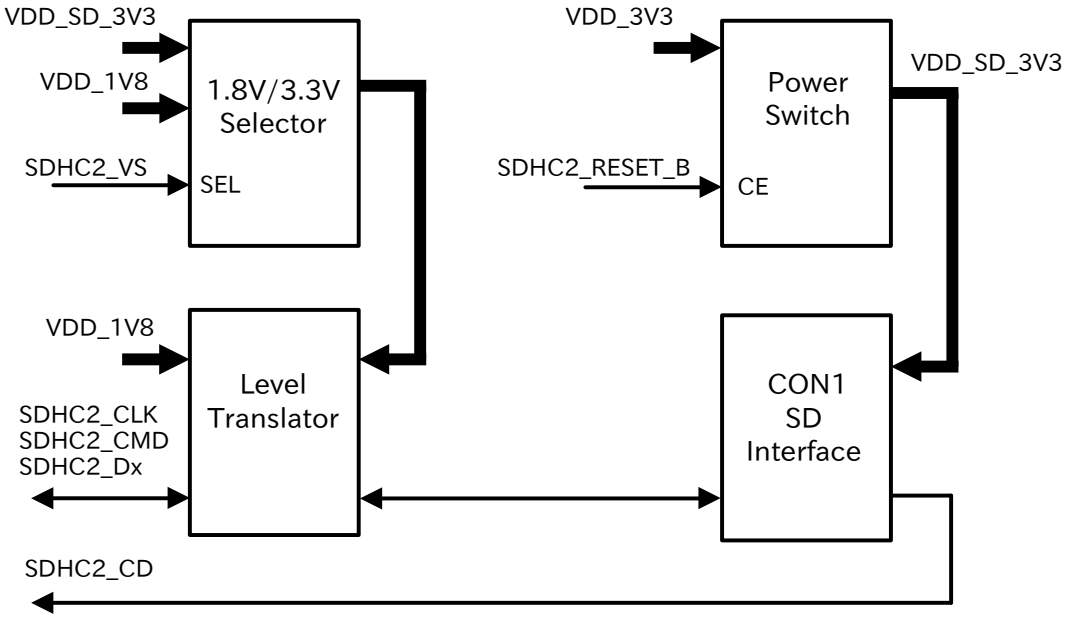


図 5.76 SD インターフェース回路構成

表 5.12 CON1 信号配列

ピン番号	ピン名	I/O	A900 との接続	説明
1	DAT2	In/Out	SDHC2_D2	SD データバス(bit2) レベルシフタ経由で A900 に接続
2	CD/DAT3	In/Out	SDHC2_D3	SD データバス(bit3) レベルシフタ経由で A900 に接続
3	CMD	In/Out	SDHC2_CMD	SD コマンド/レスポンス レベルシフタ経由で A900 に接続
4	VDD_SD_3V3	Power	-	電源(VDD_SD_3V3)
5	CLK	Out	SDHC2_CLK	SD クロック レベルシフタ経由で A900 に接続
6	VSS	Power	-	電源(GND)
7	DAT0	In/Out	SDHC2_D0	SD データバス(bit0) レベルシフタ経由で A900 に接続
8	DAT1	In/Out	SDHC2_D1	SD データバス(bit1) レベルシフタ経由で A900 に接続

表 5.13 その他 SD 信号

信号名	I/O	A900 との接続	説明
SDHC2_VS	Out	SDHC2_VS	SD バス電圧選択信号 Low = SD バス電圧 3.3V High = SD バス電圧 1.8V
SDHC2_CD	In	SDHC2_CD	SD カードディテクト信号 Low = SD カード接続 High = SD カード未接続
SDHC2_WP	In	SDHC2_WP	SD ライトプロテクト信号 未使用
SDHC2_RESET_B	Out	PTA2	SD カード用電源(VDD_SD_3V3)ONOFF 制御用信号 Low = OFF High = ON

5.5.2.2. 使用方法

microSD カードの使用方法については、「4.5. ストレージを使用する」をご参照ください。

5.5.3. Ethernet を使用する

CON6 LAN インターフェースにカテゴリ 5 以上の Ethernet ケーブルを接続することで、10BASE-T/100BASE-TX での Ethernet 通信が可能です。AUTO-MDIX 機能を搭載のため、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。

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

図 5.77 Ethernet ケーブル接続

5.5.3.1. ハードウェア仕様

LAN インターフェースの回路は、各信号をトランス内蔵 RJ45 コネクタに直接接続する構成になっています。また、LED 制御信号で RJ45 コネクタ内蔵の LED を制御しています。

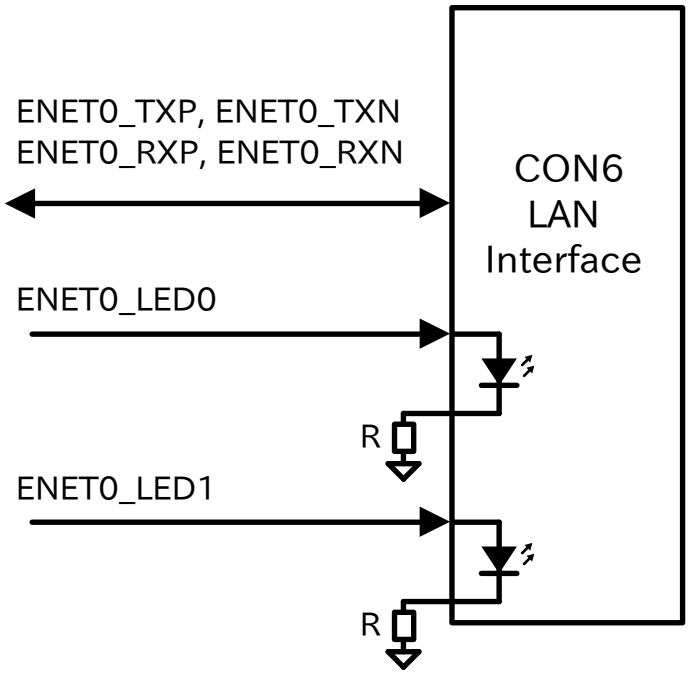


図 5.78 LAN インターフェース回路構成

表 5.14 CON6 信号配列

ピン番号	ピン名	I/O	A900 との接続	説明
1	TX+	In/Out	ENET0_TXP	送信データ(+)
2	TX-	In/Out	ENET0_TXN	送信データ(-)
3	RX+	In/Out	ENET0_RXP	受信データ(+)
4	-	-	-	5 ピンと接続後に 75Ω 終端
5	-	-	-	4 ピンと接続後に 75Ω 終端
6	RX-	In/Out	ENET0_RXN	受信データ(-)
7	-	-	-	8 ピンと接続後に 75Ω 終端
8	-	-	-	7 ピンと接続後に 75Ω 終端

表 5.15 その他 Ethernet 信号

信号名	I/O	A900 との接続	説明
ENET0_LED0	Out	ENET0_LED0	LED 制御信号 LAN リンクアクティビティ LED に接続
ENET0_LED1	Out	ENET0_LED1	LED 制御信号 LAN スピード LED に接続

表 5.16 CON6 LAN LED の動作

名称(色)	状態	説明
LAN スピード LED(緑)	消灯	10Mbps で接続されている、もしくは Ethernet ケーブル未接続
	点灯	100Mbps で接続されている
LAN リンクアクティビティ LED(黄)	消灯	リンクが確立されていない
	点灯	リンクが確立されている
	点滅	リンクが確立されており、データを送受信している

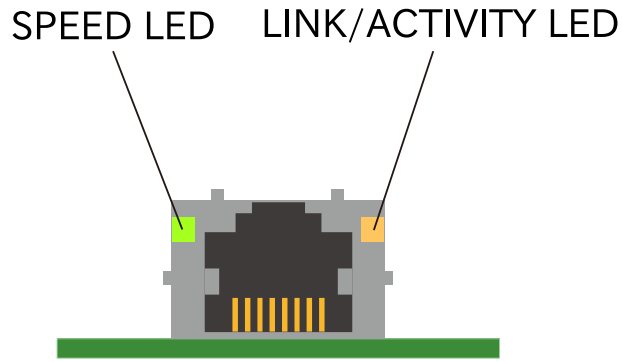


図 5.79 CON6 LAN LED

5.5.3.2. ソフトウェア仕様

ネットワークデバイス
- eth0

5.5.3.3. 使用方法

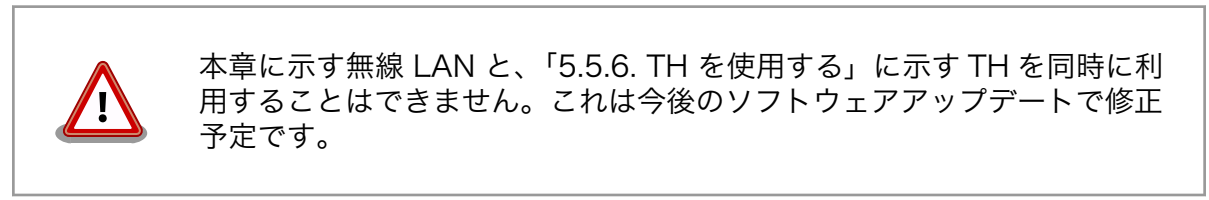
有線 LAN の設定方法は「5.1. ABOS Web を用いたネットワーク設定方法」を参照ください。

5.5.4. 無線 LAN を使用する

ANT1 WLAN/BT/TH アンテナインターフェースに、付属の WLAN/BT/TH 用外付けアンテナを接続することで、無線 LAN 通信が可能です。アンテナコネクタからアンテナまでの経路は 50Ω 同軸ケーブ

ルでの延長が可能です。ただし、ケーブルロスが発生することにご注意ください。無線 LAN、Bluetooth、IEEE 802.15.4 通信でアンテナは共通です。

機能
- IEEE 802.11a/b/g/n/ac/ax 準拠
- 最大リンク速度: 601Mbps
- 動作モード: インフラストラクチャモード(STA/AP), アドホックモード
- チャンネル(2.4GHz): 1-13
- チャンネル(5GHz): 36-48, 52-64, 100-140



本章に示す無線 LAN と、「5.5.6. TH を使用する」に示す TH を同時に利用することはできません。これは今後のソフトウェアアップデートで修正予定です。

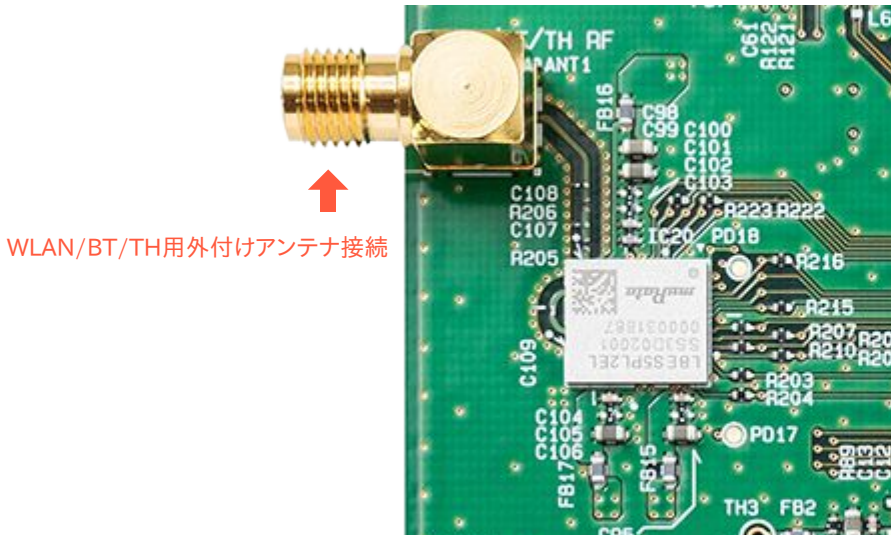


図 5.80 WLAN/BT/TH 用外付けアンテナ接続

5.5.4.1. ハードウェア仕様

無線 LAN 通信では、WLAN/BT/TH コンボモジュールとの通信に SDIO を使用する構成になっています。WLAN/BT/TH コンボモジュールは Murata Manufacturing 製 LBES5PL2EL が搭載されています。

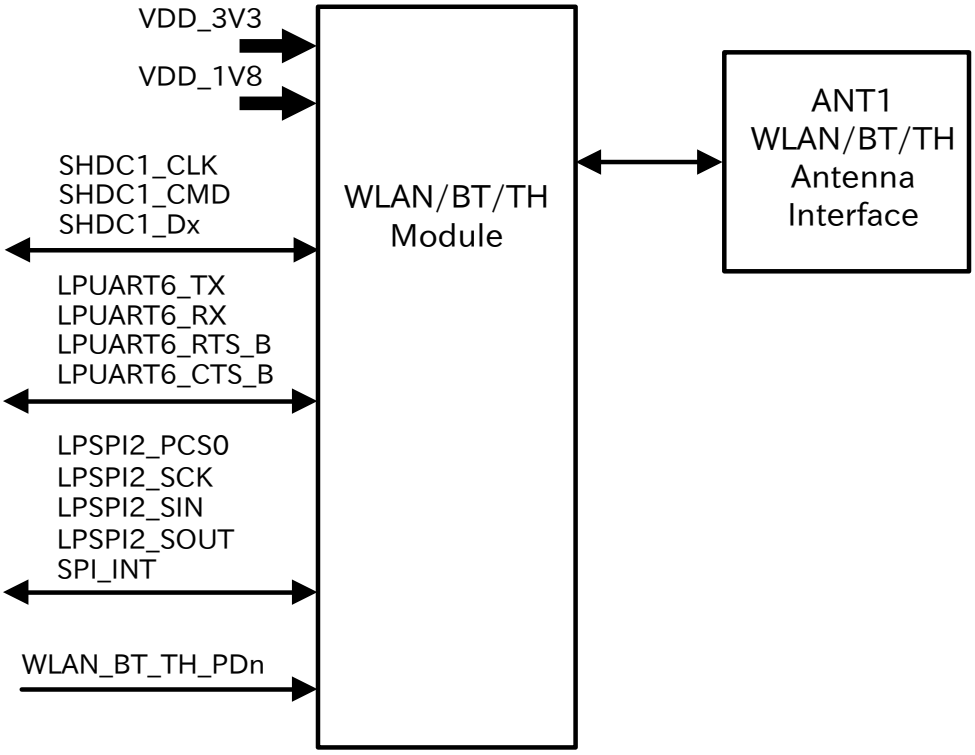


図 5.81 WLAN/BT/TH コンボモジュール回路構成

表 5.17 無線 LAN 信号

信号名	I/O	A900 との接続	説明
SDHC1_CLK	Out	PTD22	SDIO クロック
SDHC1_CMD	In/Out	PTD23	SDIO コマンド/レスポンス
SDHC1_D0	In/Out	PTD21	SDIO データバス(bit0)
SDHC1_D1	In/Out	PTD20	SDIO データバス(bit1)
SDHC1_D2	In/Out	PTD19	SDIO データバス(bit2)
SDHC1_D3	In/Out	PTD18	SDIO データバス(bit3)

5.5.4.2. ソフトウェア仕様

ネットワークデバイス
- `mlan0` (STA)
- `uap0` (AP)

5.5.4.3. 使用方法

無線 LAN の設定方法は

5.5.4.4. 注意事項

LBES5PL2EL のファームウェアは、 Armadillo-900 開発セット にインストールされている linux-firmware-imx-wifi-iw612 パッケージに含まれています。

5.5.5. Bluetooth を使用する

ANT1 WLAN/BT/TH アンテナインターフェースに、付属の WLAN/BT/TH 用外付けアンテナを接続することで、Bluetooth 通信が可能です。無線 LAN、Bluetooth、IEEE 802.15.4 通信でアンテナは共通です。

機能
- Bluetooth® version 5.3
- BLE(Bluetooth Low Energy)
- EDR(Enhanced Data Rate)

本章に示す Bluetooth と、「5.5.6. TH を使用する」に示す TH を同時に利用することはできません。これは今後のソフトウェアアップデートで修正予定です。

5.5.5.1. ハードウェア仕様

Bluetooth 通信では、WLAN/BT/TH コンボモジュールとの通信に UART を使用する構成になっています。WLAN/BT/TH コンボモジュールは Murata Manufacturing 製 LBES5PL2EL が搭載されています。

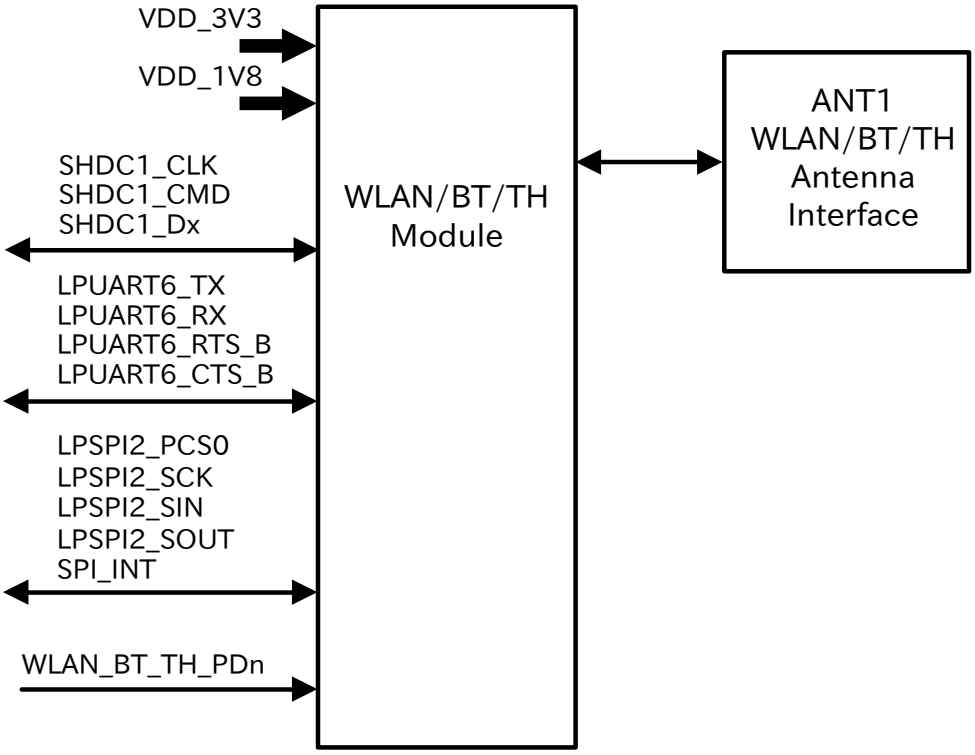


図 5.82 WLAN/BT/TH コンボモジュール回路構成

表 5.18 Bluetooth 信号

信号名	I/O	A900 との接続	説明
LPUART6_TX	Out	PTE10	UART TXD 信号
LPUART6_RX	In	PTE11	UART RXD 信号
LPUART6_CTS_B	In	PTE8	UART CTS 信号
LPUART6_RTS_B	Out	PTE9	UART RTS 信号

5.5.5.2. ソフトウェア仕様

Bluetooth® version 5.0 以降で追加された Coded PHY(Long Range)などの機能は、各種ディストリビューションが公開しているパッケージから利用することはできません。Linux で標準的に利用されている BlueZ も非対応です。

デバイスファイル
- hci0

5.5.5.3. 使用方法

コンテナ内から Bluetooth を使用するには、コンテナ作成時にホストネットワークを使用するために、NET_ADMIN の権限を渡す必要があります。「図 5.83. Bluetooth を扱うコンテナの作成例」に、alpine イメージから Bluetooth を扱うコンテナを作成する例を示します。

```
[armadillo ~]# podman pull docker.io/alpine
Trying to pull docker.io/library/alpine:latest...
Getting image source signatures
Writing manifest to image destination
8d591b0b7dea080ea3be9e12ae563eebf9869168ffced1cb25b2470a3d9fe15e
[armadillo ~]# vi /etc/atmark/containers/bt_example.conf
set_image docker.io/alpine
set_command sleep infinity
set_network host
add_args --cap-add=NET_ADMIN
[armadillo ~]# podman_start bt_example
Starting 'bt_example'
45fe1eb6b25529f0c84cd4b97ca1aef8451785fc9a87a67d54873c1ed45b70a4
```

図 5.83 Bluetooth を扱うコンテナの作成例

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

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

図 5.84 Bluetooth を起動する実行例

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

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

図 5.85 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。

5.5.6. TH を使用する

ANT1 WLAN/BT/TH アンテナインターフェースに、付属の WLAN/BT/TH 用外付けアンテナを接続することで、IEEE 802.15.4 通信が可能です。無線 LAN、Bluetooth、IEEE 802.15.4 通信でアンテナは共通です。

機能 ・ IEEE 802.15.4 準拠

5.5.6.1. ハードウェア仕様

IEEE 802.15.4 通信では、WLAN/BT/TH コンボモジュールとの通信に SPI を使用する構成になっています。WLAN/BT/TH コンボモジュールは Murata Manufacturing 製 LBES5PL2EL が搭載されています。

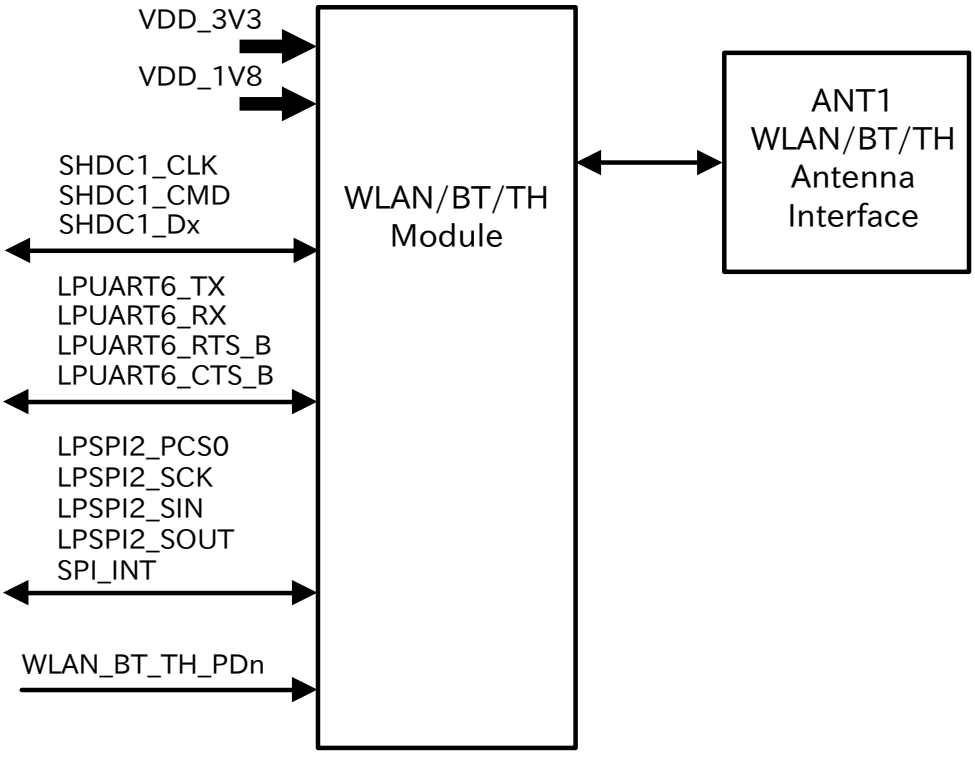


図 5.86 WLAN/BT/TH コンボモジュール回路構成

表 5.19 IEEE 802.15.4 信号

信号名	I/O	A900 との接続	説明
LPSPI2_SIN	In	PTC0	SPI SIN 信号
LPSPI2_SOUT	Out	PTC1	SPI SOUT 信号
LPSPI2_SCK	Out	PTC2	SPI SCK 信号
LPSPI2_PCS0	Out	PTC3	SPI PCS 信号

信号名	I/O	A900 との接続	説明
SPI_INT	In	PTF3	SPI INT 信号

5.5.6.2. ソフトウェア仕様

ネットワークデバイス
・ wpan0

5.5.6.3. 使用方法

この節では、2台の Armadillo-900 開発セット を用いて TH の使用方法を説明します。

1台は TH ネットワークのリーダーとなるデバイスとして使用します。

もう1台は構築した TH ネットワークに参加するデバイス（ジョイナー）として使用します。

コンテナ内から TH を使用するには、コンテナ作成時にホストネットワークを使用するために、NET_ADMIN と NET_RAW 権限を渡す必要があります。

「図 5.87. TH を扱うコンテナの作成例」に、alpine イメージから TH を扱うコンテナを作成する例を示します。リーダーとなるデバイス、ジョイナーとなるデバイス両方で同様の設定を行ってください。

```
[armadillo ~]# podman pull docker.io/alpine
Trying to pull docker.io/library/alpine:latest...
Getting image source signatures
Writing manifest to image destination
8d591b0b7dea080ea3be9e12ae563eebf9869168ffced1cb25b2470a3d9fe15e
[armadillo ~]# vi /etc/atmark/containers/th_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/spidev0.0
add_devices /dev/gpiochip2
add_devices /dev/gpiochip5
add_devices /dev/net/tun
set_network host
add_args --cap-add=NET_ADMIN
add_args --cap-add=NET_RAW
[armadillo ~]# podman_start th_example
Starting 'th_example'
45fe1eb6b25529f0c84cd4b97ca1aef8451785fc9a87a67d54873c1ed45b70a4
```

図 5.87 TH を扱うコンテナの作成例

コンテナ内で必要なソフトウェアをインストールして、ot-daemon を起動します。以下の作業もリーダーとなるデバイス、ジョイナーとなるデバイス両方で行ってください。

```
[armadillo ~]# podman exec -it th_example sh
[container ~]# apk update
[container ~]# apk add openthread-nxp-iwxxx --allow-untrusted --repository=https://download.atmark-techno.com/alpine/v`cat /etc/alpine-release | cut -d. -f1-2`/atmark/
[container ~]# ot-daemon "spinel+spi:///dev/spidev0.0?gpio-reset-device=/dev/gpiochip2&gpio-reset-line=4&gpio-int-device=/dev/gpiochip5&gpio-int-line=3&spi-mode=0&spi-speed=1000000&spi-reset-delay=0&spi-cs-delay=500" > ot.log 2>&1 &
[container ~]# ps
PID   USER     TIME  COMMAND
```

```
:（省略）
   11 root      0:00 ot-daemon spinel+spi:///dev/spidev0.0?gpio-reset-device=/d ❶
   12 root      0:00 ps
```

図 5.88 ot-daemon を起動する実行例

❶ ot-daemon が起動していることを確認してください。

これにより、OT CLI で ネットワークの構築や参加ができるようになります。

以下、リーダーとなるデバイスで行う作業です。

ot-ctl コマンドでネットワークを構築する例を示します。

```
[container ~]# ot-ctl dataset init new ❶
Done
[container ~]# ot-ctl dataset ❷
Active Timestamp: 1
Channel: 24
Channel Mask: 0x07fff800
Ext PAN ID: 12c4da5219f33729
Mesh Local Prefix: fd38:81d:4c42:8a86::/64
Network Key: f0ce0da58d072dc29cece7cc64c0307b
Network Name: OpenThread-6901
PAN ID: 0x6901
PSKc: 59bf1e7f2c2e3b194edc2f2a7418edc0
Security Policy: 672 onrc 0
Done
[container ~]# ot-ctl dataset commit active ❸
Done
[container ~]# ot-ctl ifconfig up ❹
Done
[container ~]# ot-ctl thread start ❺
Done
[container ~]# ot-ctl state ❻
leader
Done
[container ~]# ot-ctl commissioner start ❼
Commissioner: petitioning
Done
[container ~]# ot-ctl commissioner joiner add ¥* J01NME ❽
Done
```

図 5.89 ot-ctl コマンドによるネットワーク構築の例

❶ 新しいデータセットを生成します。

❷ 生成したデータセットを表示します。

❸ 生成したデータセットを有効化します。

❹ インターフェースを有効化します。

❺ ネットワークを開始します。

❻ インターフェースの状態を表示します。"leader"と表示されるまで待ちます。

❼ コミッショナーを開始します。

❽ ジョイナーを追加します。認証情報は"J01NME"としています。

ジョイナーを追加コマンドのワイルドカード("*")の指定は、全てのデバイスへ接続許可を与える事を意味します。正式運用時の使用は非推奨です。

ジョイナーを追加コマンドで指定する認証情報は、読みやすさのため I, O, Q, Z を除く大文字英数字のみ指定可能です。長さは 6〜32 文字です。

以下、ジョイナーとなるデバイスで行う作業です。

```
[container ~]# ot-ctl ifconfig up ❶
Done
[container ~]# ot-ctl joiner start J01NME ❷
Done
[container ~]# ot-ctl thread start ❸
Done
[container ~]# ot-ctl state ❹
router
Done
```

図 5.90 ot-ctl コマンドによるネットワーク参加の例

❶ インターフェースを有効化します。

❷ ジョイナーを開始し、認証情報"J01NME"を持つネットワークへの接続情報を取得します。"Join success!"と表示されると成功です。

❸ ネットワークに参加します。

❹ インターフェースの状態を表示します。"router"と表示されるまで待ちます。

5.5.7. LTE を使用する

ANT2 LTE アンテナインターフェースに、付属の LTE 用外付けアンテナを接続することで、LTE 通信が可能です。LTE データ通信には、CON18 nanoSIM インターフェースに nanoSIM カードを挿入する必要があります。nanoSIM カードを挿入する際は、nanoSIM(UIM カード)の切り欠きを挿入方向に向け、刻印面を上にして挿入してください。アンテナコネクタからアンテナまでの経路は 50Ω 同軸ケーブルでの延長が可能です。ただし、ケーブルロスが発生することにご注意ください。

機能 ・LTE 通信 Cat.1 bis

・10Mbps(DL)、5Mbps(UL)

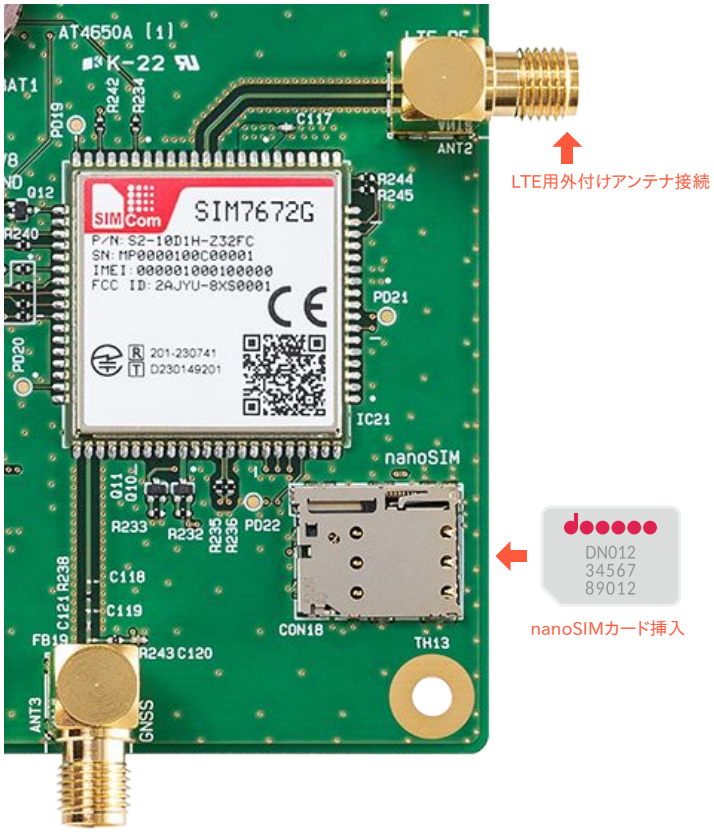


図 5.91 LTE 用外付けアンテナ接続/nanoSIM カード挿入

5.5.7.1. ハードウェア仕様

LTE 通信では、LTE モジュールとの通信に USB と UART を使用する構成になっています。LTE モジュールは、SIMCom 製 SIM7672G が搭載されています。

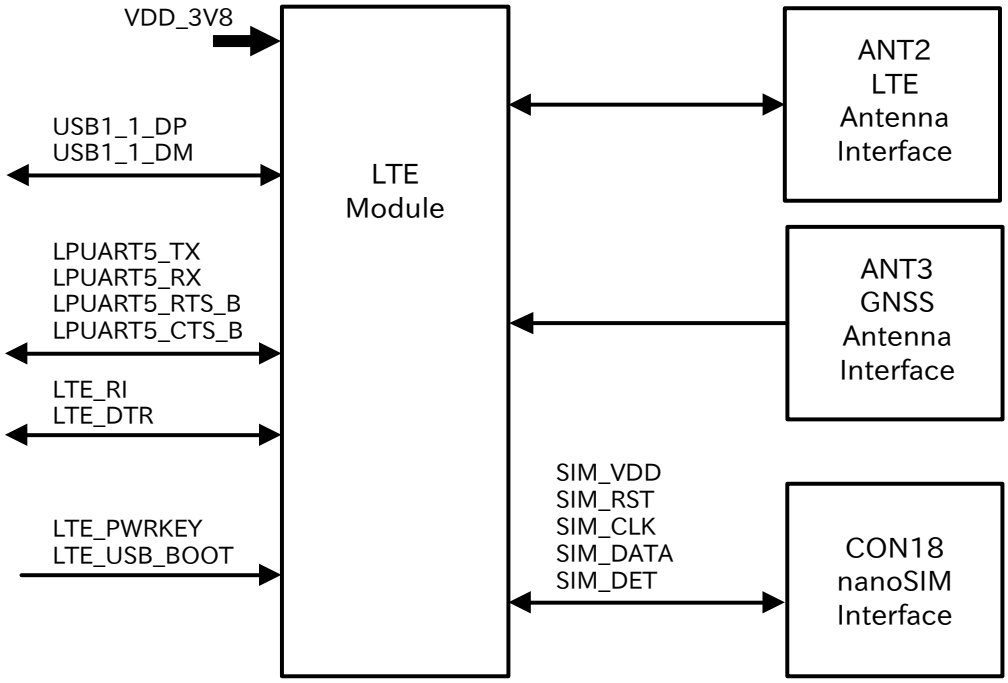


図 5.92 LTE インターフェース回路構成

表 5.20 CON18 信号配列

ピン番号	ピン名	I/O	A900 との接続	説明
C1	SIM_VCC	Power	-	SIM 電源、LTE モジュールの SIM_VDD に接続
C2	SIM_RST	Out	-	SIM リセット、LTE モジュールの SIM_RST に接続
C3	SIM_CLK	Out	-	SIM クロック、LTE モジュールの SIM_CLK に接続
C5	GND	Power	-	電源(GND)
C6	SIM_VPP	-	-	未接続
C7	SIM_I/O	In/Out	-	SIM データ、LTE モジュールの SIM_DATA に接続

表 5.21 LTE 信号

信号名	I/O	A900 との接続	説明
USB1_1_DP	In/Out	-	USB 差動信号(+) USB Hub ポート 1 経由で A900 の USB1_DP 接続
USB1_1_DM	In/Out	-	USB 差動信号(-) USB Hub ポート 1 経由で A900 の USB1_DM 接続
LPUART5_TX	Out	PTE6	UART TX 信号
LPUART5_RX	In	PTE7	UART RX 信号
LPUART5_RTS_B	Out	PTD15	UART RTS 信号
LPUART5_CTS_B	In	PTD12	UART CTS 信号
LTE_RI	In	PTB6	RI 信号
LTE_DTR	Out	PTD14	DTR 信号
LTE_PWRKEY	Out	PTF25	PWRKEY 信号
LTE_USB_BOOT	Out	PTF26	USB BOOT 信号
LTE_STATUS	In	PTD16	STATUS 信号

信号名	I/O	A900 との接続	説明
LTE_VBUS_CTL	Out	PTC12	VBUS CONTROL 信号

5.5.7.2. ソフトウェア仕様

デバイスファイル
- /dev/ttyACM0
  - ModemManager が /dev/ttyCommModem のシンボリックリンクを作成し ATコマンド用ポートとして使用します。
- /dev/ttyLP1

ネットワークデバイス
- ppp0

ttyACM0 は、他の USB デバイスを接続している場合、番号が変わる可能性があります。

5.5.7.3. 使用方法

LTE モデム SIMCom 製 SIM7672G は、Armadillo 起動時に自動的に電源が投入され、 Armadillo 終了時には自動的に電源が切られるようになっていますが、これらは以下のコマンドによる手動操作も可能です。

ただし、以下のコマンドを実行する前には、「図 5.43. LTE 再接続サービスを停止する」の手順を参考に再接続サービスをあらかじめ停止してください。「5.2.7. LTE 再接続サービス」 では、通信状態に応じて LTE モデムのリセットなどを自動的に実施するので、処理の重複を避けるためにその操作があらかじめ必要です。

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

図 5.93 LTE モデムをリセットまたは LTE モデムの電源を入れる

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

図 5.94 LTE モデムの電源を切る

ネットワークの設定方法については「5.1. ABOS Web を用いたネットワーク設定方法」を参照してください。

LTE 再接続サービスの設定、省電力設定に関しては「5.2.6. LTE」を参照してください。

5.5.8. GNSS を使用する

ANT3 GNSS アンテナインターフェースに、GNSS 用アンテナを接続することで、GNSS 通信が可能です。GNSS 用アンテナは付属していません。ただし、付属の LTE 用外付けアンテナは GNSS にも対応しているため、パッシブアンテナとして GNSS 通信の動作確認をすることが可能です。

機能 ・ GNSS 対応バンド： GPS L1、GLONASS G1、BeiDou B1I、GALILEO E1

図 5.95 GNSS 用外付けアンテナ接続

5.5.8.1. ハードウェア仕様

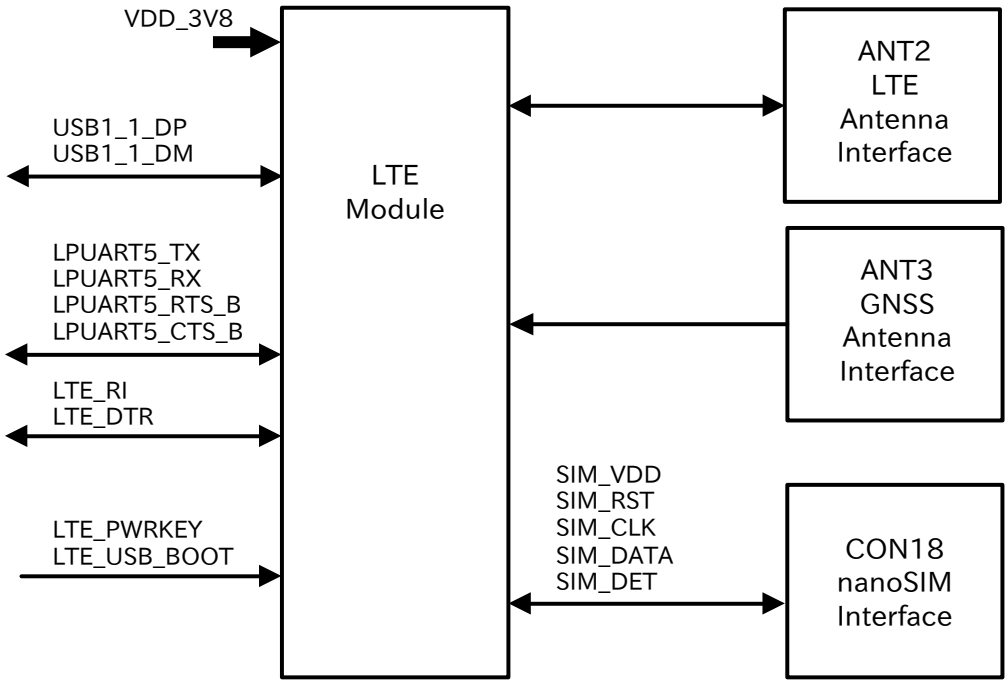


図 5.96 LTE インターフェース回路構成

表 5.22 推奨アンテナ仕様

項目	内容
対応周波数	L1(1559～1609MHz)
インピーダンス	50Ω
コネクタ	SMA-P スタンダード
電源電圧	DC 3.3V (アクティブアンテナの場合)

5.5.8.2. ソフトウェア仕様

デバイスファイル
・ /dev/ttyLP1

5.5.8.3. 使用方法

まず、アクティブアンテナを使用する場合は、以下のコマンドでアンテナ用の電源を供給します。パッシブアンテナを使用する場合にはこの操作は必要ありません。

```
[armadillo ~]# gpioset -t0 GNSS_ANT_PWR_EN=1 ❶
```

図 5.97 アンテナ用電源を ON にする

❶ アンテナ用電源を ON にします。OFF にする際は =1 の代わりに =0 を入力してください。

次に GNSS 機能を有効にします。

```
[armadillo ~]# send-at /dev/ttyLP1 AT+CGNSSPWR=1 echo sim7672
AT+CGNSSPWR=1
OK
```

図 5.98 GNSS を有効にする

GNSS・GPS 衛星の信号の捕捉には、上記コマンドで GNSS 機能を有効にしてから 40 秒ほどかかります。また、信号を捕捉するためには、屋内ではなく空が見える屋外にアンテナを配置してください。

GNSS・GPS 衛星の信号を捕捉できるようになると、以下のコマンドで GNSS・GPS の情報を取得できます。

```
[armadillo ~]# send-at /dev/ttyLP1 AT+CGNSSINFO echo sim7672 ❶
AT+CGNSSINFO
+CGNSSINFO: 3,06,02,01,06,43.074890,N,141.348003,E,
090525,095431.000,35.5,0.01,0.00,1.91,1.25,1.44,13
OK
[armadillo ~]# send-at /dev/ttyLP1 AT+CGPSINFO echo sim7672 ❷
AT+CGPSINFO
+CGPSINFO: 4304.4934,N,14120.8802,E,090525,095425.000,35.6,0.01,0.00
OK
```

図 5.99 GNSS と GPS の情報を取得する

❶ GNSS の情報を取得します。出力されるデータのフォーマットは 「表 5.23. AT+CGNSSINFO の出力フォーマット」 を参照してください。

❷ GPS の情報を取得します。出力されるデータのフォーマットは 「表 5.24. AT+CGPSINFO の出力フォーマット」 を参照してください。

表 5.23 AT+CGNSSINFO の出力フォーマット

名称	例	説明
mode	3	Fix モード。2（2D）か 3（3D）を示します。
GPS-SVs	06	GPS 衛星の視認数。
GLONASS-SVs	02	GLONASS 衛星の視認数。
GALILEO-SVs	01	GALILEO 衛星の視認数。
BEIDOU-SVs	06	BEIDOU 衛星の視認数。
lat	43.074890	現在地の緯度。フォーマットは DEG 形式で dd.dddddd です。
N/S	N	北（N）か南（S）かを示します。
log	141.348003	現在地の経度。フォーマットは DEG 形式で ddd.dddddd です。
E/W	E	東（E）か西（W）かを示します。
date	090525	現在の日付。フォーマットは ddmmyy です。
UTC time	095431.000	現在の UTC 時刻。フォーマットは hhmmss.sss です。
alt	35.5	高度（MSL）。単位は m です。
speed	0.01	対地速度。単位は knot です。
course	0.00	移動の方位。単位は deg です。
PDOP	1.91	位置精度低下率。
HDOP	1.25	水平精度低下率。

名称	例	説明
VDOP	1.44	垂直精度低下率。
NoSV	13	測位に関わる衛星数。

表 5.24 AT+CGPSINFO の出力フォーマット

名称	例	説明
lat	4304.4934	現在地の緯度。フォーマットは DMM 形式で ddmm.mmmm です。
N/S	N	北（N）か南（S）かを示します。
log	14120.8802	現在地の経度。フォーマットは DMM 形式で dddmm.mmmm です。
E/W	E	東（E）か西（W）かを示します。
date	090525	現在の日付。フォーマットは ddmmyy です。
UTC time	095425.000	現在の UTC 時刻。フォーマットは hhmmss.sss です。
alt	35.6	高度（MSL）。単位は `m` です。
speed	0.01	対地速度。単位は `knot` です。
course	0.00	移動の方位。単位は `deg` です。

5.5.9. USB デバイスを使用する

CON2 USB インターフェース 1 または、CON4 USB インターフェース 2 に USB メモリ等の USB デバイスを接続することで、USB デバイスを利用可能です。

機能
- Universal Serial Bus Specification Revision 2.0 準拠
- Enhanced Host Controller Interface (EHCI)準拠
- 転送レート：USB2.0 High-Speed（480Mbps），Full-Speed（12Mbps），Low-Speed (1.5Mbps)

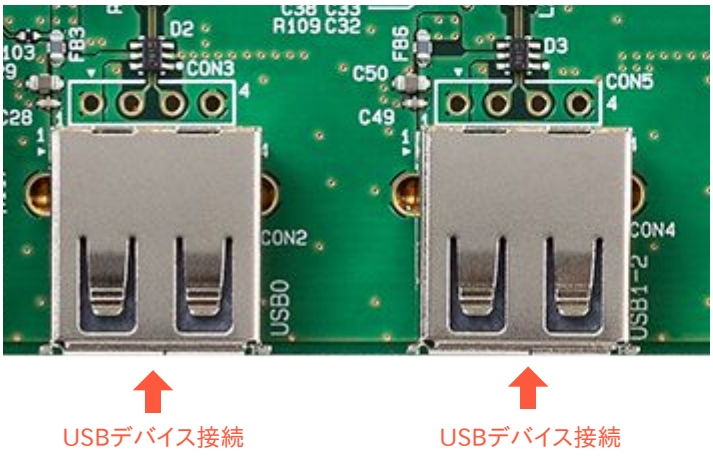


図 5.100 USB デバイス接続

5.5.9.1. ハードウェア仕様

USB インターフェース 1 の回路は、USB コネクタへ各信号を直接接続する構成になっています。USB の電源(USB0_VBUS)はパワースイッチで ONOFF を制御しています。USB インターフェース 2 の回路は、USB ハブのポート 2 を USB コネクタへ接続する構成になっています。USB の電源(USB1_VBUS)はパワースイッチで ONOFF を制御しています。

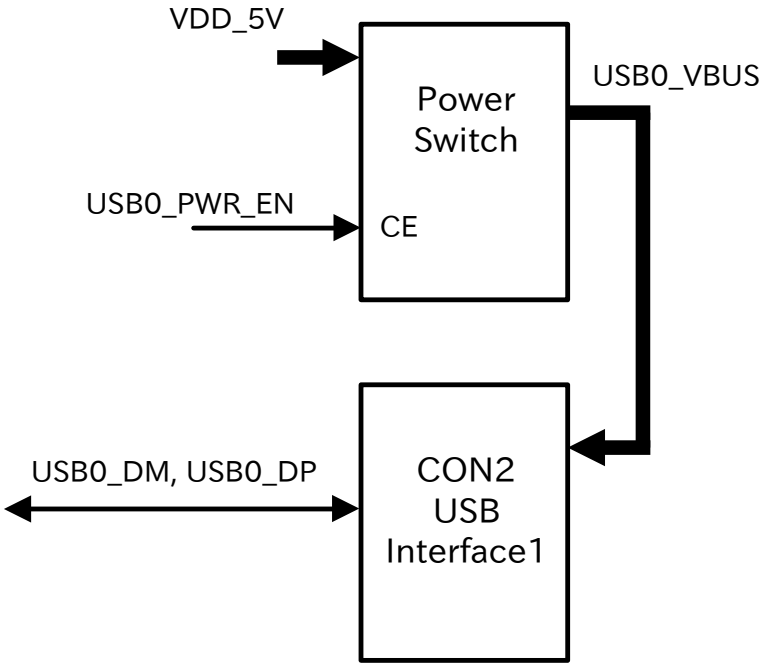


図 5.101 USB インターフェース 1 回路構成

表 5.25 CON2 信号配列

ピン番号	ピン名	I/O	A900 との接続	説明
1	VBUS	Power	-	電源(USB0_VBUS)
2	D-	In/Out	USB0_DM	USB 差動信号(-)
3	D+	In/Out	USB0_DP	USB 差動信号(+)
4	GND	Power	-	電源(GND)

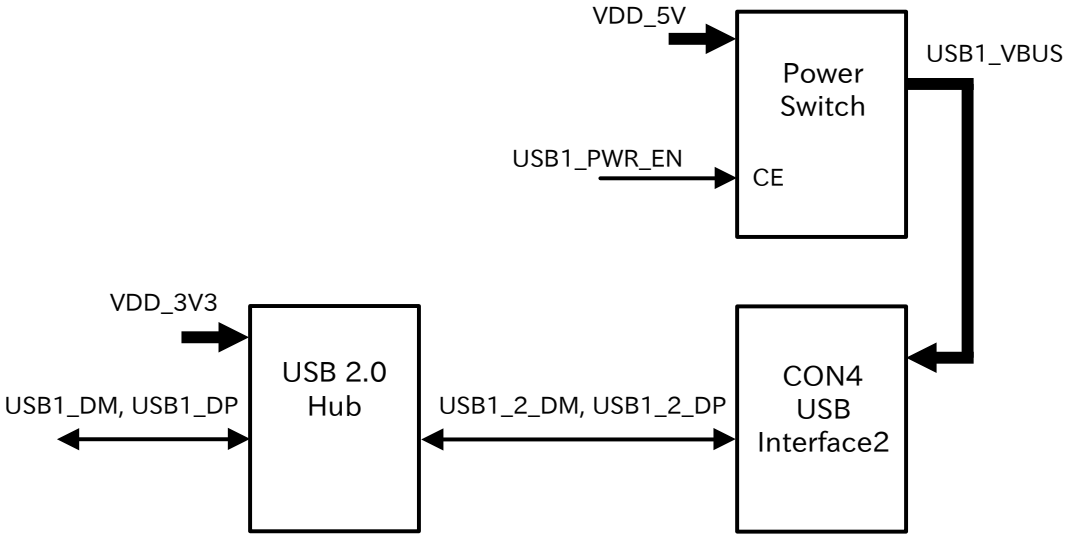


図 5.102 USB インターフェース 2 回路構成

表 5.26 CON4 信号配列

ピン番号	ピン名	I/O	A900 との接続	説明
1	VBUS	Power	-	電源(USB1_VBUS)

ピン番号	ピン名	I/O	A900 との接続	説明
2	D-	In/Out	-	USB 差動信号(-) USB Hub ポート 2 経由で A900 の USB1_DM 接続
3	D+	In/Out	-	USB 差動信号(+) USB Hub ポート 2 経由で A900 の USB1_DP 接続
4	GND	Power	-	電源(GND)

5.5.9.2. ソフトウェア仕様

デバイスファイル
- メモリデバイスの場合は、デバイスを認識した順番で/dev/sdN (N は'a'からの連番)となります。
- I/O デバイスの場合は、ファンクションに応じたデバイスファイルとなります。

5.5.9.3. 使用方法

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

設定を変更するには「5.4. USB デバイスの接続を許可する」をご確認ください。

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

- USB シリアルデバイスを扱う

USB シリアルデバイスをコンテナ内から扱う場合には、Podman のイメージからコンテナを作成する際に add_hotplugs に ttyUSB を設定する必要があります。この設定により、コンテナ起動後に USB シリアルデバイスを接続した場合でも正しく認識されます。以下は、alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# podman pull docker.io/alpine
Trying to pull docker.io/library/alpine:latest...
Getting image source signatures
Writing manifest to image destination
8d591b0b7dea080ea3be9e12ae563eebf9869168ffced1cb25b2470a3d9fe15e
[armadillo ~]# vi /etc/atmark/containers/usb_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_hotplugs ttyUSB
[armadillo ~]# podman_start usb_example
Starting 'usb_example'
34cb0e60d6274ac1df87aed58a461bcf56d0c117c4d377af130605ea399e0950
```

図 5.103 USB シリアルデバイスを扱うためのコンテナ作成例

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

```
[armadillo ~]# podman exec -it usb_example sh
[container ~]# setserial -a /dev/serial/by-id/usb-067b_2303-if00-port0
/dev/serial/by-id/usb-067b_2303-if00-port0, Line 4, UART: 16654, Port: 0x0000, IRQ: 0
```

```
        Baud_base: 460800, close_delay: 0, divisor: 0
        closing_wait: infinite
        Flags: spd_normal
```

図 5.104 setserial コマンドによる USB シリアルデバイス設定の確認例

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

- USB カメラを扱う

  USB カメラをコンテナ内から扱う場合には、Podman のイメージからコンテナを作成する際に add_hotplugs に video4linux を設定する必要があります。この設定により、コンテナ起動後に USB カメラを接続した場合でも正しく認識されます。以下は、alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# podman pull docker.io/alpine
Trying to pull docker.io/library/alpine:latest...
Getting image source signatures
Writing manifest to image destination
8d591b0b7dea080ea3be9e12ae563eebf9869168ffced1cb25b2470a3d9fe15e
[armadillo ~]# vi /etc/atmark/containers/usbcam_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_hotplugs video4linux
[armadillo ~]# podman_start usbcam_example
Starting 'usbcam_example'
ffe06090b45826cc0b1c7710e9e850ba9521d36b70de4288d0dfe1fe91a35632
[armadillo ~]# podman exec -it usbcam_example sh
[container ~]# ls /dev/v4l/by-id/usb-046d_HD_Pro_Webcam_C920_78DA8CAF-video-index0
/dev/v4l/by-id/usb-046d_HD_Pro_Webcam_C920_78DA8CAF-video-index0
```

図 5.105 USB カメラを扱うためのコンテナ作成例

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

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

- USB メモリを扱う

  ここでは、USB メモリを扱う方法について 2 つの例を示します。

  - ホスト OS 側でマウントした USB メモリをコンテナから扱う

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

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

```
[armadillo ~]# ls /mnt
sample.txt
```

図 5.106 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 ~]# podman pull docker.io/alpine
Trying to pull docker.io/library/alpine:latest...
Getting image source signatures
Writing manifest to image destination
8d591b0b7dea080ea3be9e12ae563eebf9869168ffced1cb25b2470a3d9fe15e
[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 5.107 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 5.108 USB メモリに保存されているデータの確認例

- USB メモリをコンテナ内からマウントする

  USB メモリをコンテナ内からマウントして扱う場合には、Podman のイメージからコンテナを作成する際に add_hotplugs に sd を設定する必要があります。この設定により、コンテナ起動後に USB メモリを接続した場合でも正しく認識されます。加えて、コンテナ内からマウントするためには適切な権限も設定する必要があります。以下は、 alpine イメージからコンテナを作成する例です。権限として SYS_ADMIN を渡しています。

```
[armadillo ~]# podman pull docker.io/alpine
Trying to pull docker.io/library/alpine:latest...
Getting image source signatures
Writing manifest to image destination
8d591b0b7dea080ea3be9e12ae563eebf9869168ffced1cb25b2470a3d9fe15e
[armadillo ~]# vi /etc/atmark/containers/usbmem_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_args --cap-add=SYS_ADMIN
add_hotplugs sd
```

```
[armadillo ~]# podman_start usbmem_example
Starting 'usbmem_example'
387a2256530e9b35b5361ca681a99fba8f46d78b6a6cb8ecd60096246b9198a8
```

図 5.109 USB メモリをマウントするためのコンテナ作成例

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

```
[armadillo ~]# podman exec -it usbmem_example sh
[container ~]# mount /dev/disk/by-label/[MYUSBMEMORY] /mnt ❶
[container ~]# ls /mnt
sample.txt
[container ~]# cat /mnt/sample.txt
test
```

図 5.110 コンテナ内から USB メモリをマウントする例

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

コンテナ内からマウントするデバイスの指定には /dev/sdN を使用することもできますが、他にもストレージデバイスを接続している場合などには N の値が変わることがあります。このため、USB メモリにラベルを設定している場合は、上記の例のように /dev/disk/by-label/ 下にあるラベルと同名のファイルを指定することで確実に目的のデバイスを使用することができます。

5.5.10. CAN を使用する

CON7 CAN インターフェースに CAN 通信対応機器を接続することで、CAN 通信が可能です。

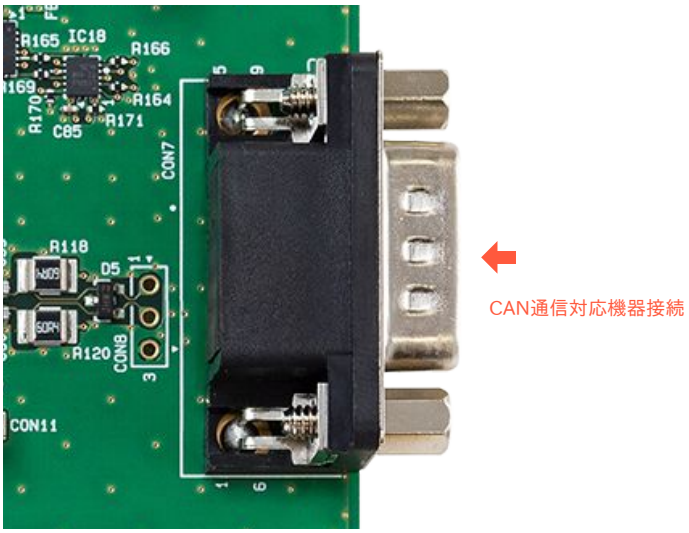


図 5.111 CAN 通信対応機器接続

5.5.10.1. ハードウェア仕様

CAN インターフェースの回路は、CAN TX/RX 信号をレベルシフタで電圧変換、CAN トランシーバで差動信号に変換する構成になっています。

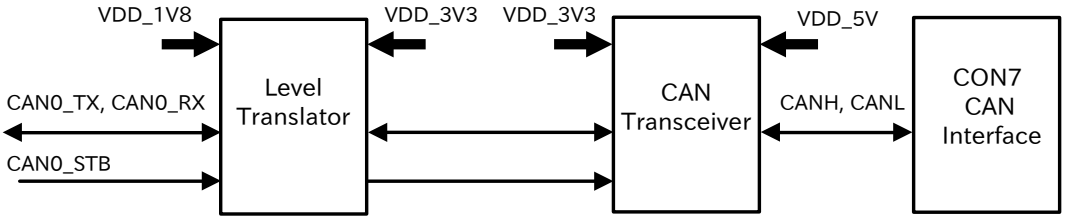


図 5.112 CAN インターフェース回路構成

表 5.27 CON7 信号配列

ピン番号	ピン名	I/O	A900 との接続	説明
1	-	-	-	-
2	CANL	In/Out	-	CAN 差動信号(-)
3	GND	Power	-	電源(GND)
4	-	-	-	-
5	GND	Power	-	電源(GND)
6	GND	Power	-	電源(GND)
7	CANH	In/Out	-	CAN 差動信号(+)
8	-	-	-	-
9	-	-	-	-

表 5.28 その他 CAN 信号

信号名	I/O	A900 との接続	説明
CAN0_TX	Out	PTA12	CAN TX 信号 レベルシフタ経由で CAN トランシーバに接続
CAN0_RX	In	PTA13	CAN RX 信号 レベルシフタ経由で CAN トランシーバに接続
CAN0_STB	Out	PTA14	CAN スタンバイ信号 レベルシフタ経由で CAN トランシーバに接続

5.5.11. MIPI CSI カメラ を使用する

CON9 MIPI CSI インターフェースに FFC(15 ピン/1.0mm ピッチ)を介して MIPI CSI カメラを接続することで、MIPI CSI カメラを利用可能です。

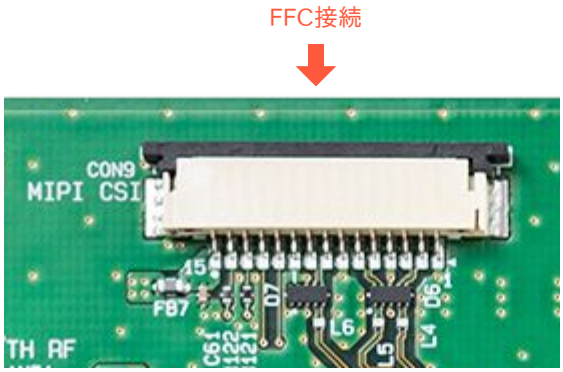


図 5.113 MIPI CSI FFC 接続

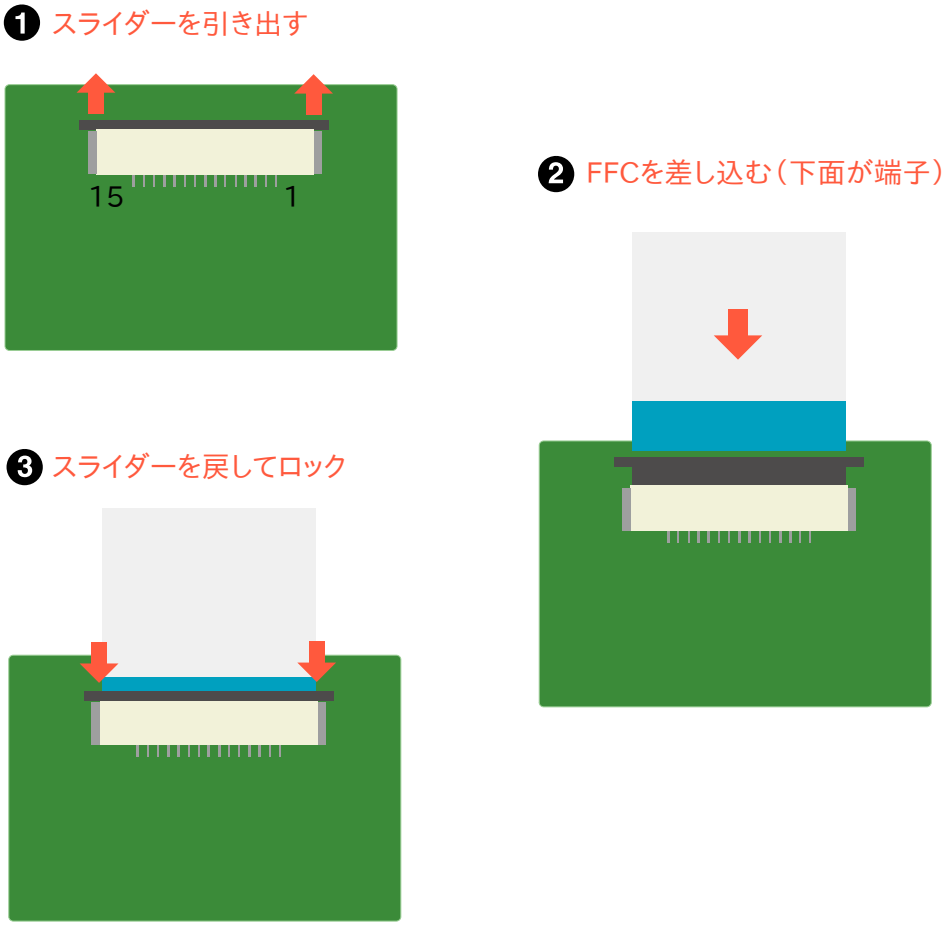


図 5.114 MIPI CSI インターフェース FFC 接続方法

5.5.11.1. ハードウェア仕様

MIPI CSI インターフェースの回路は、MIPI CSI の各信号を FFC コネクタに直接接続する構成になっています。また、3.3V の I2C、GPIO を利用できるように、レベルシフタで電圧変換した I2C、GPIO をそれぞれ FFC コネクタに接続しています。

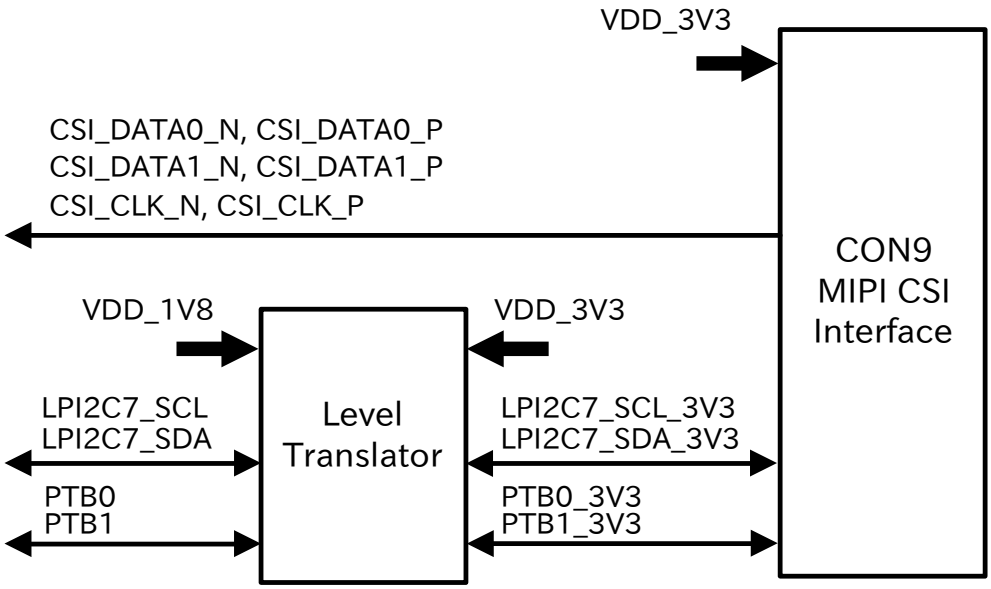


図 5.115 MIPI CSI インターフェース回路構成

表 5.29 CON9 信号配列

ピン番号	ピン名	I/O	A900 との接続	説明
1	GND	Power	-	電源(GND)
2	CSI_DATA0_N	In	CSI_DATA0_N	MIPI CSI DATA0(-)
3	CSI_DATA0_P	In	CSI_DATA0_P	MIPI CSI DATA0(+)
4	GND	Power	-	電源(GND)
5	CSI_DATA1_N	In	CSI_DATA1_N	MIPI CSI DATA0(-)
6	CSI_DATA1_P	In	CSI_DATA1_P	MIPI CSI DATA0(+)
7	GND	Power	-	電源(GND)
8	CSI_CLK_N	In	CSI_CLK_N	MIPI CSI CLK(-)
9	CSI_CLK_P	In	CSI_CLK_P	MIPI CSI CLK(+)
10	GND	Power	-	電源(GND)
11	PTB0_3V3	In/Out	PTB0	3.3V GPIO 信号 レベルシフタ経由で A900 に接続
12	PTB1_3V3	In/Out	PTB1	3.3V GPIO 信号 レベルシフタ経由で A900 に接続
13	LPI2C7_SCL_3V3	In/Out	PTF4	3.3V I2C SCL 信号 レベルシフタ経由で A900 に接続
14	LPI2C7_SDA_3V3	In/Out	PTF5	3.3V I2C SDA 信号 レベルシフタ経由で A900 に接続
15	VDD_3V3	Power	-	電源(VDD_3V3)

5.5.12. MIPI DSI ディスプレイ を使用する

CON10 MIPI DSI インターフェースに FFC(22 ピン/0.5mm ピッチ)を介して MIPI DSI ディスプレイを接続することで、MIPI DSI ディスプレイを利用可能です。

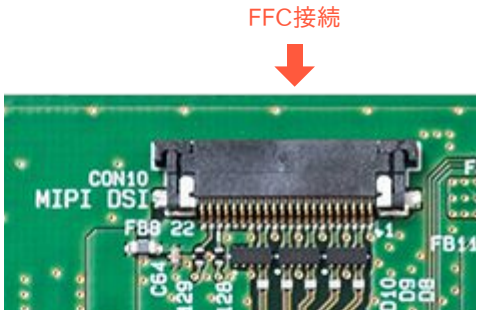


図 5.116 MIPI DSI FFC 接続

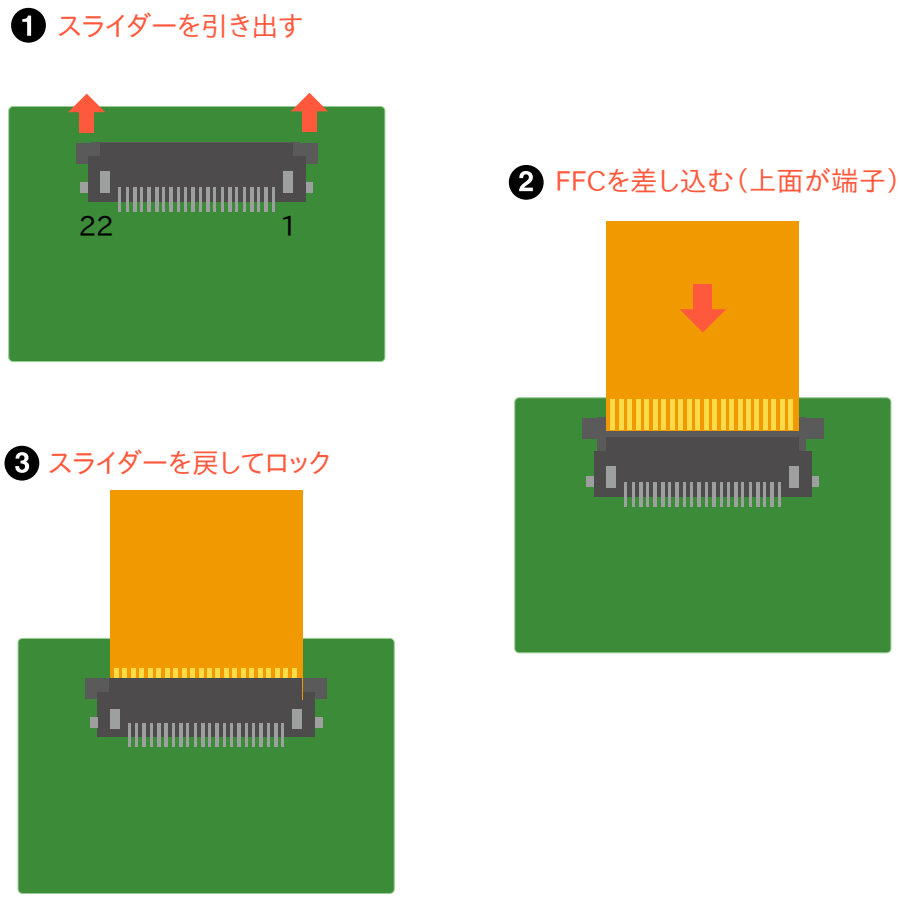


図 5.117 MIPI DSI インターフェース FFC 接続方法

5.5.12.1. ハードウェア仕様

MIPI DSI インターフェースの回路は、MIPI DSI の各信号を FFC コネクタに直接接続する構成になっています。また、3.3V の I2C、GPIO を利用できるように、レベルシフタで電圧変換した I2C、GPIO をそれぞれ FFC コネクタに接続しています。

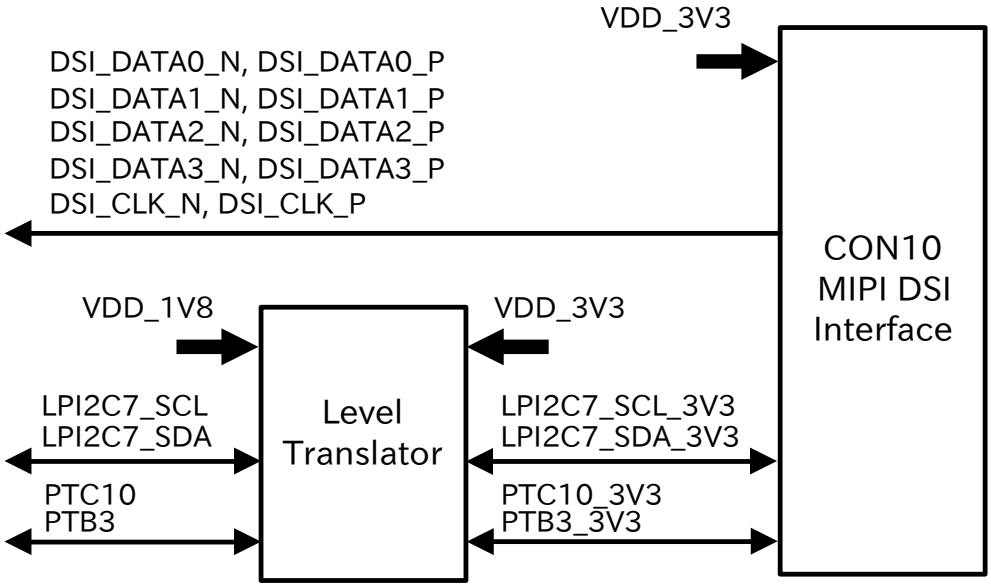


図 5.118 MIPI DSI インターフェース回路構成

表 5.30 CON10 信号配列

ピン番号	ピン名	I/O	A900 との接続	説明
1	GND	Power	-	電源(GND)
2	DSI_DATA0_N	Out	DSI_DATA0_N	MIPI DSI DATA0(-)
3	DSI_DATA0_P	Out	DSI_DATA0_P	MIPI DSI DATA0(+)
4	GND	Power	-	電源(GND)
5	DSI_DATA1_N	Out	DSI_DATA1_N	MIPI DSI DATA0(-)
6	DSI_DATA1_P	Out	DSI_DATA1_P	MIPI DSI DATA0(+)
7	GND	Power	-	電源(GND)
8	DSI_CLK_N	Out	DSI_CLK_N	MIPI DSI CLK(-)
9	DSI_CLK_P	Out	DSI_CLK_P	MIPI DSI CLK(+)
10	GND	Power	-	電源(GND)
11	DSI_DATA2_N	Out	DSI_DATA2_N	MIPI DSI DATA2(-)
12	DSI_DATA2_P	Out	DSI_DATA2_P	MIPI DSI DATA2(+)
13	GND	Power	-	電源(GND)
14	DSI_DATA3_N	Out	DSI_DATA3_N	MIPI DSI DATA3(-)
15	DSI_DATA3_P	Out	DSI_DATA3_P	MIPI DSI DATA3(+)
16	GND	Power	-	電源(GND)
17	PTC10_3V3	In/Out	PTC10	3.3V GPIO 信号 レベルシフタ経由で A900 に接続
18	PTB3_3V3	In/Out	PTB3	3.3V GPIO 信号 レベルシフタ経由で A900 に接続
19	GND	Power	-	電源(GND)
20	LPI2C7_SCL_3V3	In/Out	PTF4	3.3V I2C SCL 信号 レベルシフタ経由で A900 に接続
21	LPI2C7_SDA_3V3	In/Out	PTF5	3.3V I2C SDA 信号 レベルシフタ経由で A900 に接続
22	VDD_3V3	Power	-	電源(VDD_3V3)

5.5.13. APD 用コンソール を使用する

CON13 APD 用コンソールインターフェースに USB Type C ケーブル経由で PC 等に接続することで、アプリケーションドメイン用のコンソール入出力を利用可能です。

図 5.119 USB Type C ケーブル接続

5.5.13.1. ハードウェア仕様

APD 用コンソールインターフェースの回路は、USB シリアル変換 IC で UART 信号を USB 信号に変換する構成になっています。USB シリアル変換 IC は、USB Type C コネクタから給電される電源によって動作します。

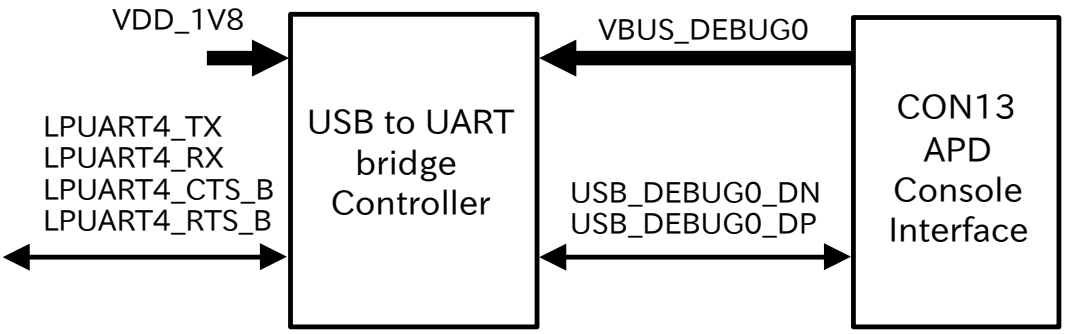


図 5.120 APD 用コンソールインターフェース回路構成

表 5.31 CON13 信号配列

ピン番号	ピン名	I/O	A900 との接続	説明
A1	GND	Power	-	電源(GND)
B1	GND	Power	-	電源(GND)
A4	VBUS	Power	-	電源(VBUS_DEBUG0)
B4	VBUS	Power	-	電源(VBUS_DEBUG0)
A5	CC1	-	-	5.1kΩ プルダウン
B5	CC2	-	-	5.1kΩ プルダウン
A6	DP1	-	-	USB2.0 差動信号(+)
B6	DP2	-	-	USB2.0 差動信号(+)
A7	DN1	-	-	USB2.0 差動信号(-)
B7	DN2	-	-	USB2.0 差動信号(-)
A9	VBUS	Power	-	電源(VBUS_DEBUG0)
B9	VBUS	Power	-	電源(VBUS_DEBUG0)
A12	GND	Power	-	電源(GND)
B12	GND	Power	-	電源(GND)

表 5.32 その他 APD コンソール信号

信号名	I/O	A900 との接続	説明
LPUART4_TX	Out	LPUART4_TX	UART TXD 信号
LPUART4_RX	In	LPUART4_RX	UART RXD 信号
LPUART4_CTS_B	In	LPUART4_CTS_B	UART CTS 信号
LPUART4_RTS_B	Out	LPUART4_RTS_B	UART RTS 信号

5.5.14. RTD 用コンソール を使用する

CON14 RTD 用コンソールインターフェースに USB Type C ケーブル経由で PC 等に接続することで、リアルタイムドメイン用のコンソール入出力を利用可能です。RTD 用コンソールは JTAG と同じ信号を使用するため同時に利用することはできません。RTD 用コンソールを使用する場合は、SW3 RTD 用コンソール JTAG 切替スイッチを RTD 用コンソールの方向に切り替える必要があります。

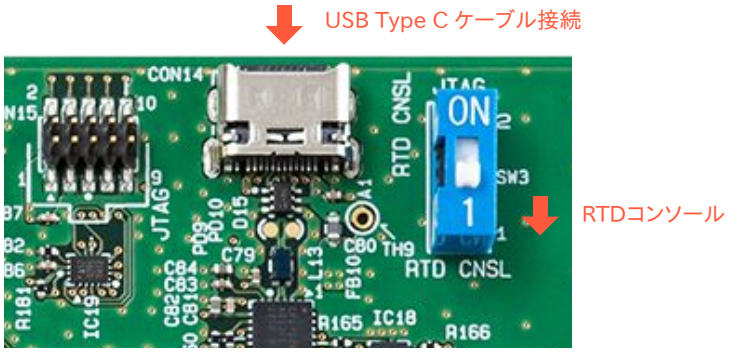


図 5.121 USB Type C ケーブル接続/切替スイッチ制御

5.5.14.1. ハードウェア仕様

RTD 用コンソールインターフェースの回路は、USB シリアル変換 IC で UART 信号を USB 信号に変換する構成になっています。USB シリアル変換 IC は、USB Type C コネクタから給電される電源によって動作します。また、RTD 用コンソールインターフェースで使用する UART 信号は、JTAG 信号と同じピンをマルチプレクスして UART の機能に割り当てるためバススイッチによって接続先のインターフェースを切り替える構成になっています。

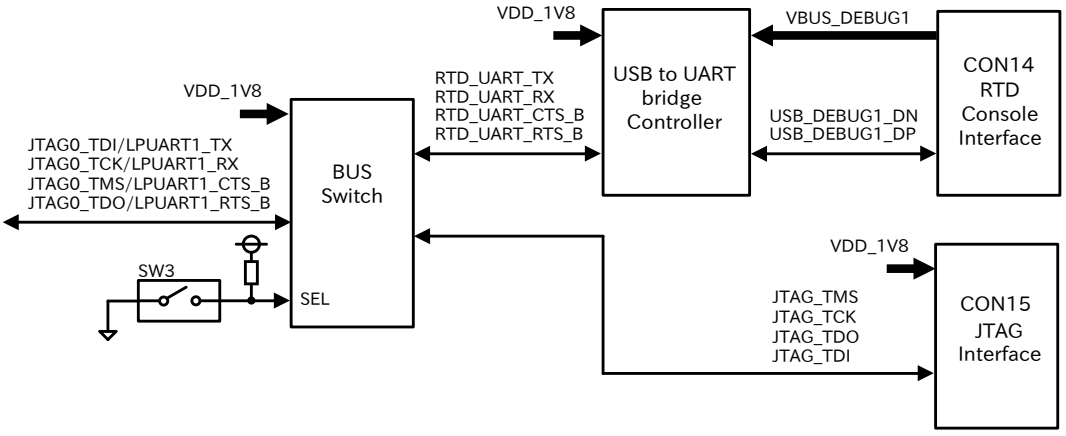


図 5.122 RTD 用コンソールインターフェース回路構成

表 5.33 CON14 信号配列

ピン番号	ピン名	I/O	A900 との接続	説明
A1	GND	Power	-	電源(GND)
B1	GND	Power	-	電源(GND)
A4	VBUS	Power	-	電源(VBUS_DEBUG1)
B4	VBUS	Power	-	電源(VBUS_DEBUG1)
A5	CC1	-	-	5.1kΩ プルダウン
B5	CC2	-	-	5.1kΩ プルダウン
A6	DP1	-	-	USB2.0 差動信号(+)
B6	DP2	-	-	USB2.0 差動信号(+)
A7	DN1	-	-	USB2.0 差動信号(-)
B7	DN2	-	-	USB2.0 差動信号(-)
A9	VBUS	Power	-	電源(VBUS_DEBUG1)
B9	VBUS	Power	-	電源(VBUS_DEBUG1)
A12	GND	Power	-	電源(GND)
B12	GND	Power	-	電源(GND)

5.5.14.2. ソフトウェア仕様

セキュリティのため、デフォルトでは出力が無効化になっています。

以下のいずれかの方法でログ出力を確認できます。:

1. Linux が起動している場合、ログは /var/log/messages に保存されています。

```
[armadillo ~]# grep rtos /var/log/messages
Apr 23 16:10:32 armadillo user.notice rtos-logger: Start SRTM communication
Apr 23 16:10:32 armadillo user.notice rtos-logger: waiting message from uboot
Apr 23 16:10:32 armadillo user.notice rtos-logger: M33> uboot: handshake
Apr 23 16:10:32 armadillo user.notice rtos-logger: initializing i2c 0 (lpi2c1)
Apr 23 16:10:32 armadillo user.notice rtos-logger: uboot: booting into linux
Apr 23 16:10:32 armadillo user.notice rtos-logger: Handle Peer Core Linkup
Apr 23 16:10:32 armadillo user.notice rtos-logger: Watchdog start (timeout 60000)
Apr 23 16:10:32 armadillo user.notice rtos-logger: first watchdog ping
Apr 23 16:10:32 armadillo user.notice rtos-logger: initializing i2c 1 (lpi2c1)
Apr 23 16:10:32 armadillo user.notice rtos-logger: initializing tty 1 as LPUART 0
Apr 23 16:10:32 armadillo user.notice rtos-logger: initialized tty 0 for M33 console
Apr 23 16:10:32 armadillo user.notice rtos-logger: spi 0: init ok
```

また、「 abos-ctrl rtos interact 」コマンドを実行するとログが停止されますがコンソールのように操作できます。

```
[armadillo ~]# abos-ctrl rtos interact
Could not lock rtos console for reading
stop rtos-logger service ? [Y/n]
WARNING: It will not be restarted automatically!

rtos-logger              | * WARNING: you are stopping a boot service
rtos-logger              | * Stopping rtos-                                    ⏎
logger ...                                                              [ ok ]
Entering console. Press ctrl+D to exit
M33> help
help    - this help
reset   - cold reset
```

```
log     - print buffer log
clear   - clear buffer log
quiet   - disable background messages
verbose - enable background messages
version - print firmware version
M33>
```

2. u-boot の環境変数でコンソールを有効化できます。こちらの設定では起動時の初期段階以外の出力を確認できます：

```
[armadillo ~]# fw_setenv m33_console 22/23
Environment OK, copy 0
[armadillo ~]# reboot
```

こちらの設定を永続化したい場合は例えば /boot/uboot_env.d/70_m33_console ファイルを作成し、変数を設定してください：

```
[armadillo ~]# echo 'm33_console=22/23' > /boot/uboot_env.d/70_m33_console
[armadillo ~]# persist_file -v /boot/uboot_env.d/70_m33_console
'/mnt/boot/uboot_env.d/70_m33_console' -> '/target/boot/uboot_env.d/70_m33_console'
```

設定可能な値は以下の通りです：

値	動作
none	デフォルトの無効化状態
22/23	CON14 (LPUART1 の tx=PTA22, rx=PTA23) に出力されます
<tx>/<rx>	1.8V TTL UART 変換ケーブルを利用すると他の PTA のピンでも利用可能です：LPUART0 (tx: 2, 14, 18, rx: 3, 15, 19) か LPUART1 (tx: 6, 10, 22, rx: 7, 11, 23) 。

3. imx-boot ソースの変更ですべてのログを確認できます。m33_firmware_at/debug_console.c ファイルの DEFAULT_UART_CONSOLE_TX と DEFAULT_UART_CONSOLE_RX 値を設定すると、m33 ファームウェア起動時に有効化されます。この場合、u-boot の環境変数が無視されます。ビルド手順については 「10.19.1. ブートローダーをビルドする」 を参照ください。

5.5.15. JTAG を使用する

CON15 JTAG インターフェースに JTAG ケーブル (SWJ-PRB-MIL20-10HP 等の 2x5 ピン/1.27mm ピッチコネクタ) を介して、PALMiCE4 [https://www.computex.co.jp/products/palmice4/index.htm] 等の JTAG エミュレータに接続することで、JTAG を利用可能です。JTAG は RTD 用コンソールの信号と同じ信号を使用するため同時に利用することはできません。JTAG を使用する場合は、SW3 RTD 用コンソール JTAG 切替スイッチを JTAG の方向に切り替える必要があります。

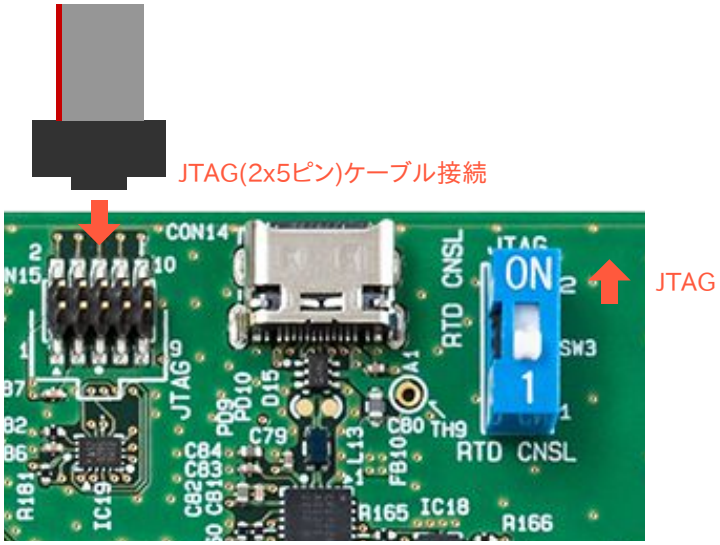


図 5.123 JTAG ケーブル接続/切替スイッチ制御

5.5.15.1. ハードウェア仕様

JTAG インターフェースの回路は、JTAG 信号を 10 ピンのピンヘッダに接続する構成になっています。ただし、JTAG 信号は RTD 用コンソールインターフェースで使用する UART 信号と同じピンをマルチプレクスして JTAG の機能に割り当てるためバススイッチによって接続先のインターフェースを切り替える構成になっています。

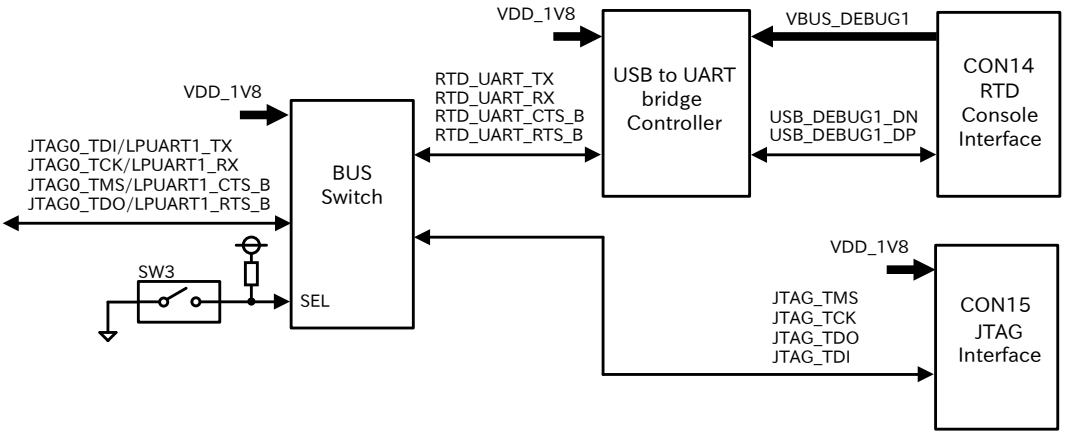


図 5.124 RTD 用コンソールインターフェース回路構成

表 5.34 CON13 信号配列

ピン番号	ピン名	I/O	A900 との接続	説明
1	VDD_1V8	Power	-	電源(VDD_1V8)
2	JTAG_TMS		JTAG0_TMS/ LPUART1_CTS_B	JATG TMS 信号 マルチプレクサ経由で A900 に接続
3	GND	Power	-	電源(GND)
4	JTAG_TCK		JTAG0_TCK/ LPUART1_RX	JTAG TCK 信号 マルチプレクサ経由で A900 に接続
5	GND	Power	-	電源(GND)

ピン番号	ピン名	I/O	A900 との接続	説明
6	JTAG_TDO	-	JTAG0_TDO/ LPUART1_RTS_B	JTAG TDO 信号 マルチプレクサ経由で A900 に接続
7	-	-	-	
8	JTAG_TDI	-	JTAG0_TDI/ LPUART1_TX	JTAG TDI 信号 マルチプレクサ経由で A900 に接続
9	GND	Power	-	電源(GND)
10	SYS_N_RST	-	SYS_N_RST	リセット信号

5.5.16. I2C デバイスを使用する

CON16 拡張インターフェースまたは、CON17 I2C(3.3V)インターフェースを使用することで I2C 通信を利用することが可能です。CON16 の信号電圧は 1.8V、CON17 の信号電圧は 3.3V になります。

図 5.125 I2C(3.3V)インターフェース

5.5.16.1. ハードウェア仕様

表 5.35 CON17 信号配列

ピン番号	ピン名	I/O	A900 との接続	説明
1	VDD_3V3	Power	-	電源(VDD_3V3)
2	LPI2C7_SCL_ 3V3	In/Out	PTF4	I2C SCL 信号(3.3V) レベルシフタ経由で A900 に接続
3	LPI2C7_SDA_ 3V3	In/Out	PTF5	I2C SDA 信号(3.3V) レベルシフタ経由で A900 に接続
4	GND	Power	-	電源(GND)

5.5.17. ソフトウェア仕様

Armadillo-900 開発セット の I2C インターフェースは、i.MX 8ULP の I2C(I2C Controller) を利用します。また、i2c-gpio を利用することで、I2C バスを追加することができます。Armadillo-900 開発セット で利用している I2C バスと、接続される I2C デバイスを次に示します。

表 5.36 I2C デバイス

I2C バス	I2C デバイス	
	アドレス	デバイス名
1(I2C1)	0x32	RV8803 (RTC)
	0x2C	USB2422(USB Hub)
6(I2C6)	0x48	SE050(セキュアエレメント)

Armadillo-900 開発セット の標準状態では、CONFIG_I2C_CHARDEV が有効となっているためユーザードライバで I2C デバイスを制御することができます。ユーザードライバを利用する場合は、Linux カーネルで I2C デバイスに対応するデバイスドライバを無効にする必要があります。

機能 ・ 最大転送レート: 384kbps

デバイスファイル ・ /dev/i2c-1 (I2C1)

- /dev/i2c-6 (I2C6)
- /dev/i2c-7 (I2C7)

5.5.18. 使用方法

コンテナ内で動作するアプリケーションから I2C を扱うためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/i2c-N を渡す必要があります。以下は、/dev/i2c-7 を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# podman pull docker.io/alpine
Trying to pull docker.io/library/alpine:latest...
Getting image source signatures
Writing manifest to image destination
8d591b0b7dea080ea3be9e12ae563eebf9869168ffced1cb25b2470a3d9fe15e
[armadillo ~]# vi /etc/atmark/containers/i2c_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/i2c-7
[armadillo ~]# podman_start i2c_example
Starting 'i2c_example'
efa1eb129c1f036a709755f0d53b21a0f2a39307ecae32b24aac98c0b6567bf0
```

図 5.126 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 7
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

図 5.127 i2cdetect コマンドによる確認例

5.5.19. DAC を使用する

CON25 DAC インターフェースを使用することで、デジタル/アナログ変換出力を利用することが可能です。出力電圧範囲は 0〜1.8V です。

図 5.128 DAC インターフェース

5.5.19.1. ハードウェア仕様

表 5.37 CON25 信号配列

ピン番号	ピン名	I/O	A900 との接続	説明
1	VDD_1V8	Power	-	電源(VDD_3V3)
2	DAC0_OUT	Out	DAC0_OUT	DAC 出力
3	DAC1_OUT	Out	DAC1_OUT	DAC 出力
4	GND	Power	-	電源(GND)

5.5.20. RTC を使用する

CON11 RTC バックアップインターフェース 1、CON12 RTC バックアップインターフェース 2 にバックアップ用電池を接続することで、電源が切断されても時刻データを保持することが可能です。CON12 には CR1220 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、240nA(Typ.)となっておりますので、電池寿命までの時刻保持が期待できます。

最大月差は周囲温度-20℃～60℃で 8 秒です。(経年変化を除く)

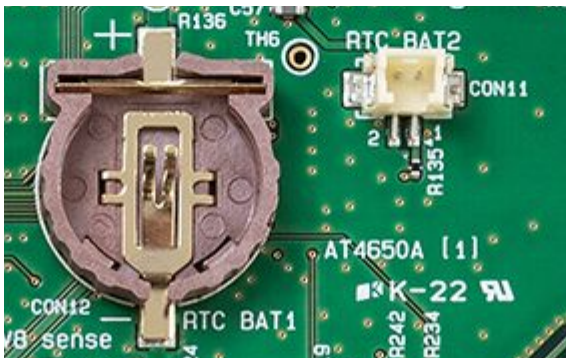


図 5.129 RTC バックアップインターフェース

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。

5.5.20.1. ハードウェア仕様

RTC バックアップインターフェースの回路は、VDD_5V 電源と RTC バックアップインターフェースから入力された電源をダイオード OR し、RTC IC の電源を保持する構成になっています。VDD_5V は Armadillo-900 開発セットに電源が投入されている間は常に供給されます。RTC バックアップインターフェース 1 と RTC バックアップインターフェース 2 に同時に電源を入力することはできません。RTC IC は、Micro Crystal 製 RV-8803-C7 が搭載されています。

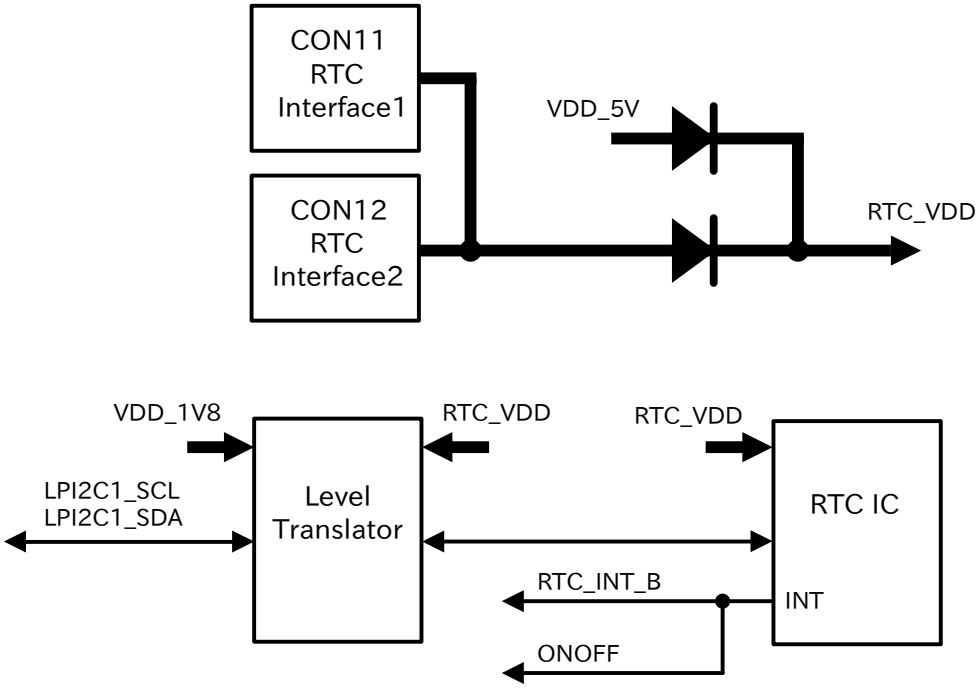


図 5.130 RTC バックアップインターフェース回路構成

表 5.38 CON11 信号配列

ピン番号	ピン名	I/O	A900 との接続	説明
1	RTC_VDD	Power	-	リアルタイムクロックのバックアップ用電源入力 (RTC_VDD)
2	GND	Power	-	電源(GND)

表 5.39 CON12 信号配列

ピン番号	ピン名	I/O	A900 との接続	説明
1	RTC_VDD	Power	-	リアルタイムクロックのバックアップ用電源入力 (RTC_VDD)
2	GND	Power	-	電源(GND)

5.5.20.2. ソフトウェア仕様

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

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。

5.5.20.3. 使用方法

・ コンテナで使用する

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/rtc 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/rtc 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 ~]# podman pull docker.io/alpine
Trying to pull docker.io/library/alpine:latest...
Getting image source signatures
Writing manifest to image destination
8d591b0b7dea080ea3be9e12ae563eebf9869168ffced1cb25b2470a3d9fe15e
[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/rtc
[armadillo ~]# podman_start rtc_example
Starting 'rtc_example'
025209e0d96f43c2911239a8397b7002c3eaab057e031d8abb765df5707d75bd
```

図 5.131 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 5.132 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。

・ Armadillo 上で RTC に時刻を設定する

Linux の時刻には、Linux カーネルが管理するシステムクロックと、RTC が管理するハードウェアクロックの 2 種類があります。RTC に時刻を設定するためには、まずシステムクロックを設定します。その後に、ハードウェアクロックをシステムクロックと一致させる手順となります。

システムクロックは、date コマンドを用いて設定します。date コマンドの引数には、設定する時刻を [MMDDhhmmCCYY.ss] というフォーマットで指定します。時刻フォーマットの各フィールドの意味を次に示します。

表 5.40 時刻フォーマットのフィールド

フィールド	意味
MM	月
DD	日(月内通算)
hh	時
mm	分
CC	年の最初の 2 桁(省略可)
YY	年の最後の 2 桁(省略可)
ss	秒(省略可)

2023 年 3 月 2 日 12 時 34 分 56 秒に設定する例を次に示します。

```
[armadillo ~]# date
Sat Jan  1 09:00:00 JST 2000
[armadillo ~]# date 030212342023.56
Fri Mar  2 12:34:56 JST 2023
[armadillo ~]# date
Fri Mar  2 12:34:57 JST 2023
```

図 5.133 システムクロックを設定

システムクロックを設定後、ハードウェアクロックを hwclock コマンドを用いて設定します。

```
[armadillo ~]# hwclock ❶
2000-01-01 00:00:00.000000+09:00
[armadillo ~]# hwclock --utc --systohc ❷
[armadillo ~]# hwclock --utc ❸
2023-03-02 12:57:20.534140+09:00
```

図 5.134 ハードウェアクロックを設定

❶ 現在のハードウェアクロックを表示します。

❷ ハードウェアクロックを協定世界時(UTC)で設定します。

❸ ハードウェアクロックが UTC で正しく設定されていることを確認します。

インターネットに接続できている場合は、chronyd により自動的に日時設定が行われます。そのため、手動で日時設定を行う必要はありません。

5.5.21. ユーザースイッチを使用する

SW1 はユーザーが自由に利用できる押しボタンスイッチです。

5.5.21.1. ハードウェア仕様

表 5.41 SW1 信号配列

部品番号	名称	説明
SW1	ユーザースイッチ	A900 の PTB2 ピンに接続

5.5.21.2. ソフトウェア仕様

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

ユーザースイッチと信号には、次に示すキーコードが割り当てられています。

表 5.42 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。

5.5.21.3. 使用方法

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 ~]# podman pull docker.io/alpine
Trying to pull docker.io/library/alpine:latest...
Getting image source signatures
Writing manifest to image destination
8d591b0b7dea080ea3be9e12ae563eebf9869168ffced1cb25b2470a3d9fe15e
[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 5.135 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 148 (KEY_PROG1)
Properties:
Testing ... (interrupt to exit)
Event: time 1744349334.534281, type 1 (EV_KEY), code 148 (KEY_PROG1), value 1 ❶
Event: time 1744349334.534281, ---------- SYN_REPORT ------------
Event: time 1744349334.822277, type 1 (EV_KEY), code 148 (KEY_PROG1), value 0 ❷
Event: time 1744349334.822277, ---------- SYN_REPORT ------------
```

図 5.136 evtest コマンドによる確認例

❶ SW1 のボタン **プッシュ** イベントを検出したときの表示

❷ SW1 のボタン **リリース** イベントを検出したときの表示

Armadillo Base OS では、スイッチの制御を簡単に実装できる **buttond** デーモンを用意しております。

5.5.22. リセットスイッチを使用する

SW6 リセットスイッチを使用することで、Armadillo-900 開発セットに搭載された Armadillo-900 上の PMIC をコールドリセットすることができます。デフォルトでは、1 秒以上スイッチを押下することで動作します。リセットスイッチを用いたリセットは、ソフトウェアの動作状況に関わらず動作するため、ソフトウェアの更新等で再起動が必要な場合には使用しないでください。

5.5.22.1. ハードウェア仕様

表 5.43 SW6 信号配列

部品番号	名称	説明
SW6	リセットスイッチ	A900 の SYS_N_RST ピンに接続

5.5.23. LED を使用する

LED は SYS、APP、WWAN が実装されており、Armadillo Base OS にて「表 5.45. LED 状態と製品状態の対応について」に示す状態を表示しています。

5.5.23.1. ハードウェア仕様

表 5.44 LED 信号配列

部品番号	名称(色)	説明
SYS	システム LED(緑)	電源(VDD_3V3)の入力状態を表示、A900 の PTC5 に接続 (Low: 消灯、High: 点灯)
APP	アプリケーション LED(緑)	アプリケーションの状態を表示、A900 の PTF30 に接続 (Low: 消灯、High: 点灯)
WWAN	ワイヤレス WAN LED(緑)	LTE 通信の状態を表示、A900 の PTF31 に接続 (Low: 消灯、High: 点灯)

5.5.23.2. ソフトウェア仕様

Linux では、GPIO 接続用 LED ドライバ(leds-gpio)で制御することができます。

sysfs LED クラスディレクトリ
- /sys/class/leds/app
- /sys/class/leds/sys
- /sys/class/leds/wwan

表 5.45 LED 状態と製品状態の対応について

LED 状態\LED 名称	SYS	APP	WWAN
OFF	電源 OFF	アプリ起動不可	SIM 未検出または認識中、または LTE モデム未検出
ON	電源 ON	アプリ起動可能	LTE 接続済み
Blink Slow	シャットダウン中	アプリ起動完了 [a]	SIM 検出、LTE 未接続 [b]
Blink Fast	アップデート中	アプリエラー [a]	SIM 検出、LTE 未接続、電波品質が低い [b]

[a]APP LED の「起動完了」と「エラー」の点滅動作は、アプリ自身が行います。
[b]LTE コネクションが未作成、設定間違いの場合もこの状態となります

WLAN/LAN モデルでは WWAN LED を自由に使用することができます。

5.5.23.3. 使用方法

- コンテナで使用する

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 ~]# podman pull docker.io/alpine
Trying to pull docker.io/library/alpine:latest...
Getting image source signatures
Writing manifest to image destination
8d591b0b7dea080ea3be9e12ae563eebf9869168ffced1cb25b2470a3d9fe15e
[armadillo ~]# vi /etc/atmark/containers/led_example.conf
set_image docker.io/alpine
```

```
set_command sleep infinity
add_volumes /sys
[armadillo ~]# podman_start led_example
Starting 'led_example'
c770f76d7714f4cceb1229be2240382bded236c2c51bb6b866bc0098c2cb987a
```

図 5.137 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/app/brightness
[container ~]# echo 1 > /sys/class/leds/app/brightness
```

図 5.138 LED の点灯/消灯の実行例

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

- LED を点灯/消灯する

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

  brightness に 0 以外の値を書き込むと LED が点灯します。

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

図 5.139 LED を点灯させる

Armadillo-900 開発セット の LED には輝度制御の機能がないため、0(消灯)、1~255(点灯)の 2 つの状態のみ指定することができます。

  brightness に 0 を書き込むと LED が消灯します。

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

図 5.140 LED を消灯させる

  brightness を読み出すと LED の状態が取得できます。

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

図 5.141 LED の状態を表示する

・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要な値は以下の通りです。

表 5.46 LED トリガの種類

設定	説明
none	トリガを設定しません
mmc0	eMMC のアクセスランプにします
mmc2	microSD スロットのアクセスランプにします
timer	任意のタイミングで点灯/消灯を行います。この設定にすることにより、LED クラスディレクトリ以下に delay_on, delay_off ファイルが出現し、それぞれ点灯時間, 消灯時間をミリ秒単位で指定します
heartbeat	心拍のように点灯/消灯を行います
default-on	主に Linux カーネルから使用します。LED が点灯します
panic	カーネルパニック時に LED が点滅します

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

```
[armadillo ~]# cat /sys/class/leds/[LED]/trigger
[none] bluetooth-power rfkill-any rfkill-none kbd-scrolllock kbd-numlock kbd-capslock kbd-
kanalock kbd-shiftlock kbd-altgrlock kbd-ctrllock kbd-altlock kbd-shiftllock kbd-shiftrlock
kbd-ctrllock kbd-ctrlrlock timer oneshot mtd nand-disk heartbeat activity default-on panic
pattern mmc0 mmc1 mmc2 29950000.ethernet-1:00:link 29950000.ethernet-1:00:100Mbps
29950000.ethernet-1:00:10Mbps rfkill0 hci0-power rfkill1
```

図 5.142 対応している LED トリガを表示

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

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

図 5.143 LED のトリガに timer を指定する

5.5.24. SMS を利用する

Armadillo-900 開発セット は、LTE モジュール を使用した SMS の送受信を行うことができます。SMS の送信、受信した SMS の確認および削除などの操作は ModemManager の mmcli コマンドで行うことができます。

本章では mmcli コマンドでの SMS の使用方法について説明します。

5.5.24.1. 初期設定

SMS が利用可能な SIM を挿入して Armadillo-900 開発セット の電源を入れると、 ModemManager が必要な初期設定を行い、 SMS が利用可能になります。

SMS の受信は自動的に行われます。

「図 5.144. 言語設定」に示すコマンドを実行し、言語設定を行います。

```
[armadillo ~]# export LANG="ja_JP.UTF-8"
```

図 5.144 言語設定

5.5.24.2. SMS を送信する

SMS を作成するには、「図 5.145. SMS の作成」に示すコマンドを実行します。

```
[armadillo ~]# mmcli -m $(mm-modem-num) --messaging-create-sms="number=[送信先電話番号],text='[SMS
本文]'"
```

図 5.145 SMS の作成

SMS の作成に成功すると、以下のように SMS 番号が表示されます。SMS 番号は送信時に使用します。

```
Successfully created new SMS: /org/freedesktop/ModemManager1/SMS/[SMS 番号]
```

図 5.146 SMS 番号の確認

「図 5.147. SMS の送信」に示すコマンドを実行し、SMS 送信を行います。 [SMS 番号] には、 SMS の作成時に表示された番号を指定します。

```
[armadillo ~]# mmcli -s [SMS 番号] --send
```

図 5.147 SMS の送信

5.5.24.3. SMS を受信する

SMS を送信可能な端末から Armadillo-900 開発セット に SMS を送信すると、 Armadillo-900 開発セット は自動的に SMS を受信します。

また、 LTE モジュールの内蔵ストレージに 10 件 SMS を保存した状態で Armadillo-900 開発セットに SMS を送信した場合は、Armadillo-900 開発セット は受信を行いません。

受信を行うには、 LTE モジュールの内蔵ストレージに保存している SMS を削除するか、他のストレージに移動する必要があります。

5.5.24.4. SMS 一覧を表示する

「図 5.148. SMS の一覧表示」のコマンドを実行することで、 SMS 一覧を表示できます。

末尾が "(sent)" となっているものが送信した SMS で "(received)" となっているものが受信した SMS です。

```
[armadillo ~]# mmcli -m $(mm-modem-num) --messaging-list-sms
Found 7 SMS messages:
        /org/freedesktop/ModemManager1/SMS/0 (received)
```

```
        /org/freedesktop/ModemManager1/SMS/1 (received)
        /org/freedesktop/ModemManager1/SMS/2 (received)
        /org/freedesktop/ModemManager1/SMS/3 (received)
        /org/freedesktop/ModemManager1/SMS/4 (sent)
        /org/freedesktop/ModemManager1/SMS/5 (received)
        /org/freedesktop/ModemManager1/SMS/6 (sent)
```

図 5.148 SMS の一覧表示

5.5.24.5. SMS の内容を表示する

SMS の内容を表示するには、「図 5.149. SMS の内容を表示」に示すコマンドを実行します。

```
[armadillo ~]# mmcli -s [SMS 番号]
  ----------------------------------
  Content    |              number: XXXXXXXXXX
             |                text: hello world
  ----------------------------------
  Properties |            PDU type: deliver
             |               state: received
             |             storage: me
             |                smsc: +XXXXXXXXXXX
             |           timestamp: XXXXXXXXXXX+XX
```

図 5.149 SMS の内容を表示

受信した SMS は自動的に LTE モジュールの内蔵ストレージに保存されます。Armadillo-900 開発セット に搭載されている、LTE モジュールには、最大 10 件まで SMS を保存することが可能です。

SMS の内容を表示した際の「storage: **me**」は、 LTE モジュールの内蔵ストレージに SMS が保存されていることを意味しています。

「storage: **sm**」と表示された場合、 SIM カードのストレージに SMS が保存されています。 SIM カードのストレージに保存できる SMS の件数は SIM カードによって異なります。

ストレージに保存されている SMS は、Armadillo-900 開発セット の電源を切断してもデータが保持されます。

5.5.24.6. SMS を削除する

SMS を削除するには、「図 5.150. SMS の削除」に示すコマンドを実行します。

```
[armadillo ~]# mmcli -m $(mm-modem-num) --messaging-delete-sms=[SMS 番号]
```

図 5.150 SMS の削除

5.5.24.7. SMS を他のストレージに移動する

SIM カードのストレージに SMS を移動するには、「図 5.151. SIM カードのストレージに SMS を移動」に示すコマンドを実行します。

```
[armadillo ~]# mmcli -s [SMS 番号] --store-in-storage="sm"
```

図 5.151 SIM カードのストレージに SMS を移動

LTE モジュールの内蔵ストレージに SMS を移動するには、「図 5.152. LTE モジュールの内蔵ストレージに SMS を移動」に示すコマンドを実行します。

```
[armadillo ~]# mmcli -s [SMS 番号] --store-in-storage="me"
```

図 5.152 LTE モジュールの内蔵ストレージに SMS を移動

5.5.25. ボタンやキーを扱う

buttond サービスを使用することで、ボタンやキー入力をトリガーとする処理を簡単に実装できます。

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

- --short <key> --action "command": 短押しの設定。キーを 1 秒以内に離せば短押しと認識し "command"を実行します。認識する最大時間は --time <time_ms> オプションで変更可能です。
- --long <key> --action "command": 長押しの設定。キーを 5 秒押し続けたタイミングで "command" を実行します。長押しと認識する最低時間は --time <time_ms> オプションで変更可能です。
- 1 つのキーに対して複数の設定が可能です。長押しの設定が複数ある場合、押したままの状態だと一番長い時間に設定されている "command" を実行します。途中でキーを離した場合は、キーを離した時間に応じた "command" を実行します。(例：buttond --short <key> --action "cmd1" --long <key> --time 2000 --action "cmd2" --long <key> --time 10000 --action "cmd3" <file> を実行した場合、1 秒以内に離すと "cmd1"、2 秒以上 10 秒以内に離すと "cmd2"、10 秒を越えたら "cmd3" を実行します)。
  - 短押し設定を複数指定する場合、時間の短い設定を先に指定してください。 0.5 秒、1 秒を設定したい場合、1 秒 → 0.5 秒の順番で指定すると 0.5 秒が無視されます。
- --exit-timeout <time_ms>: 設定した時間の後に buttond を停止します。起動時のみに対応したい場合に使えます。
- キーの設定の --exit-after オプション：キーのコマンドを実行した後に buttond を停止します。キーの対応を一回しか実行しないように使えます。

5.5.25.1. SW1 の短押しと長押しの対応

以下にデフォルトを維持したままで SW1 の短押しと長押しのそれぞれの場合にコマンドを実行させる例を示します。

```
[armadillo ~]# vi /etc/atmark/buttond.conf ❶
BUTTOND_ARGS="$BUTTOND_ARGS --short prog1 --action 'date >> /tmp/shortpress'"
BUTTOND_ARGS="$BUTTOND_ARGS --long prog1 --time 5000 --action 'date >> /tmp/longpress'"
[armadillo ~]# persist_file /etc/atmark/buttond.conf ❷
[armadillo ~]# rc-service buttond restart ❸
buttond         | * Stopping button watching daemon ...                                [ ok ]
buttond         | * Starting button watching daemon ...                                [ ok ]
```

```
[armadillo ~]# cat /tmp/shortpress ❹
Tue Mar 22 17:16:42 JST 2022
Tue Mar 22 17:16:43 JST 2022
[armadillo ~]# cat /tmp/longpress
Tue Mar 22 17:16:48 JST 2022
```

図 5.153 buttond で SW1 を扱う

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

❷ 設定ファイルを保存します。

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

❹ 押された回数を確認します。

5.5.25.2. USB キーボードの対応

USB キーボードや他の入力デバイスにも対応できます。

1. デバイスを接続してから、 buttond でデバイス名とキーコードを確認します。例では左側の ctrl キーを押しています。

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

図 5.154 buttond で USB キーボードのイベントを確認する

❶ buttond を -vvv で冗長出力にして、すべてのデバイスを指定します。

❷ 希望のキーを押すと、LEFTCTRL が三つのパスで認識されました。/dev/by-id/usb-0566_3029-event-kbd 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="$BUTTOND_ARGS -i /dev/input/by-id/usb-0566_3029-event-kbd" ❶
BUTTOND_ARGS="$BUTTOND_ARGS --short LEFTCTRL --action 'date >> /tmp/keyboardpress'" ❷
[armadillo ~]# persist_file /etc/atmark/buttond.conf
[armadillo ~]# rc-service buttond restart
```

図 5.155 buttond で USB キーボードを扱う

❶ 上記で検出した USB キーボードのパスを記述します。

❷ USB キーボードの左側の ctrl キーを押して 1 秒以内に放すと、/tmp/keyboardpress ファイルに押したタイミングの時間が記録されます。

USB キーボードの左側の ctrl キーを押して、想定通りに /tmp/keyboardpress に日付が書き込まれているか、以下のコマンドで確認できます。

```
[armadillo ~]# cat /tmp/keyboardpress
Wed Apr 30 11:06:12 JST 2025 ❶
```

図 5.156 buttond の挙動の確認

❶ buttond が想定通りに動いていると日付が出力されます。

5.5.25.3. Armadillo 起動時にのみボタンに反応する方法

Armadillo 起動時にのみ、例として SW1 の長押しに反応する方法を紹介します。

/etc/local.d/boot_switch.start に稼働期間を指定した buttond を起動させる設定を記載します。

buttond が起動してから 10 秒以内に SW1 を一秒以上長押しすると myapp のコンテナの親プロセスに USR1 信号を送ります（アプリケーション側で信号を受信して、デバッグモードなどに切り替える想定です）。SW1 が Armadillo 起動前に押された場合は、buttond の起動一秒後に実行されます。

```
[armadillo ~]# vi /etc/local.d/boot_switch.start
#!/bin/sh

buttond /dev/input/by-path/platform-gpio-keys-event ¥ ❶
        --exit-timeout 10000 ¥ ❷
        --long PROG1 --time 1000 --exit-after ¥ ❸
                --action "podman exec myapp kill -USR1 1" & ❹
[armadillo ~]# chmod +x /etc/local.d/boot_switch.start
[armadillo ~]# persist_file /etc/local.d/boot_switch.start
```

図 5.157 buttond で SW1 を Armadillo 起動時のみ受け付ける設定例

❶ SW1 の入力を /dev/input/by-path/platform-gpio-keys-event ファイルの PROG1 として認識できます。

❷ buttond 起動後 10 秒経過すると終了します。

❸ SW1 を一度検知した後すぐに終了します。

❹ サービスとして動作させる必要がないため & を付けてバックグラウンド起動します。

5.5.26. 動作中の Armadillo の温度を測定する

Armadillo-900 開発セット の温度センサーは、i.MX 8ULP の TEMPMON(Temperature Monitor) を利用しています。

デフォルトでは、i.MX 8ULP の測定温度が 95℃以上になった場合、Linux カーネルが /sbin/poweroff コマンドを実行し、システムを停止します。

/sys/class/thermal/thermal_zone0/temp ファイルの値を読み出すことによって、i.MX 8ULP の測定温度を取得することができます。

```
[armadillo ~]# cat /sys/class/thermal/thermal_zone0/temp
32000 ❶
```

図 5.158 i.MX 8ULP の測定温度を取得する

❶ 温度はミリ℃ の単位で表示されます。この例では 32.000℃ を示しています。

ここでは、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。

5.5.26.1. 温度測定の重要性

Armadillo は製品ごとに動作温度範囲が設定されていますが、それらはあくまでも標準筐体に放熱材と共に取り付けて使用した場合の目安であり、実運用時には自作の筐体の使用や放熱の有無などで記載のスペック通りにならない場合があります。また、 Armadillo には CPU または SoC が特定の温度以上になると、自動的にシャットダウンするサーマルシャットダウン機能が搭載されています。そのため、現実的には Armadillo を組み込んだ製品を運用時と同等の環境で動作させつつ、実際に温度を計測して実運用時の CPU 及び SoC 温度がどの程度まで上がるか、サーマルシャットダウンは起こらないかを確かめる必要があります。

Armadillo Base OS 搭載製品では、動作中の Armadillo の各種温度等を取得し CSV 形式で出力する atmark-thermal-profiler を利用することができますので、温度測定に役立てることができます。

5.5.26.2. 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 5.159 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。

5.5.26.3. atmark-thermal-profiler を実行・停止する

「図 5.160. 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 5.160 atmark-thermal-profiler を実行する

「図 5.161. 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 5.161 atmark-thermal-profiler を停止する

5.5.26.4. 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
:（省略）
2025-04-25T07:33:35+09:00,0,32,,0.17,top -b -n 1,podman exec -it example /bin/bash,/usr/sbin/ ⏎
NetworkManager -n,/usr/sbin/ModemManager,/usr/bin/abos-web,9,0,0,0,0,
:（省略）
```

図 5.162 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 5.47. thermal_profile.csv の各列の説明」に記載します。

表 5.47 thermal_profile.csv の各列の説明

ヘッダ	説明
DATE	その行のデータ取得日時です。 "年-月-日 T 時:分:秒+タイムゾーン" の形式で出力されます。
ONESHOT	この列が 1 の行のデータは、サーマルシャットダウンを含むシャットダウンが実行された時に取得されたことを示します。
CPU_TEMP	計測時点の CPU 温度を示します。単位は℃です。
SOC_TEMP	計測時点の SoC 温度を示します。単位は℃です。製品よっては非対応で、その場合は空白になります。
LOAD_AVE	計測時点から直近 1 分間の Load Average です。

ヘッダ	説明
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。

5.5.26.5. 温度測定結果の分析

予め、使用している Armadillo が何℃でサーマルシャットダウンするか確認しておきます。

```
[armadillo ~]# cat /sys/class/thermal/thermal_zone0/trip_point_1_temp
95000 ❶
```

図 5.163 サーマルシャットダウン温度の確認

❶ CPU のサーマルシャットダウン温度です。ミリ℃で表記されているので、95℃でサーマルシャットダウンすることがわかります。

5.5.26.6. 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 5.164. 取得した温度のグラフ」は一定期間 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。

作成方法の手順は以下です。(2025 年 4 月現在)

1. thermal_profile.csv をドライブにアップロード
2. [ファイル] > [インポート] > thermal_profile.csv を選択 > [挿入]
3. 範囲選択した後、[挿入] > [グラフ]
4. グラフエディタの設定の[グラフの種類]で折れ線グラフを選択
5. 不必要なパラメータはグラフエディタの[系列]から削除

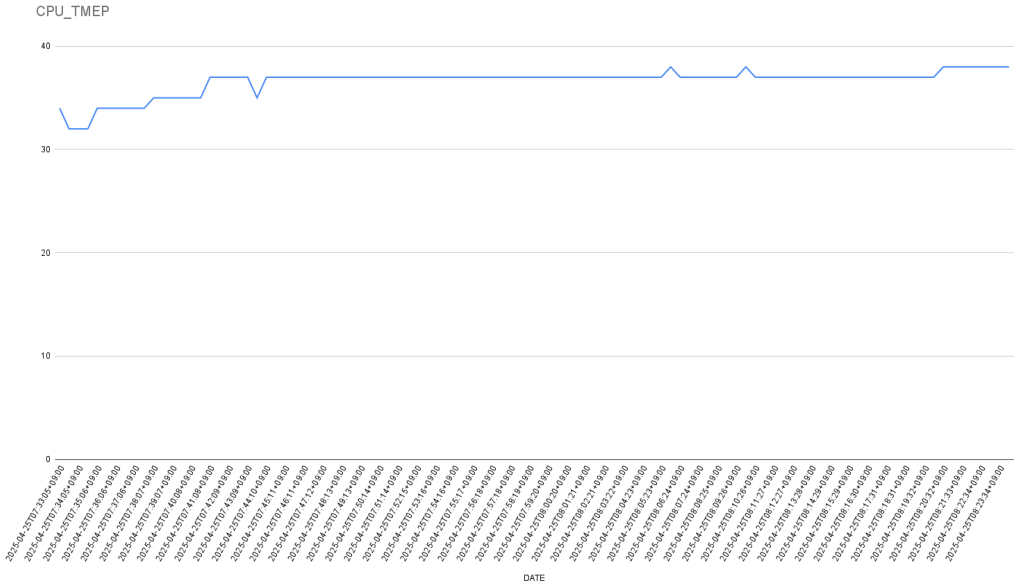


図 5.164 取得した温度のグラフ

ここでは、グラフの縦軸は温度(℃)で、横軸は時間です。青い線は CPU の温度を表しています。作成したグラフと、「5.5.26.5. 温度測定結果の分析」で得たサーマルシャットダウン温度を見比べることでどれほど温度に余裕があるか視覚的に比較できます。

5.5.26.7. 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 5.47. 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。

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

atmark-thermal-profiler の他に、Armadillo Twin からも温度や CPU 負荷率等の情報を確認することができます。詳細は Armadillo Twin ユーザーマニュアル 「デバイス監視アラートを管理する」[https://manual.armadillo-twin.com/management-device-monitoring-alert/] をご確認ください。

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

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

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

時刻取得に関する設定は 2 つのファイルに分かれています：

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

例えば、 NTP サーバーを変更する際は 「図 5.165. chronyd のコンフィグの変更例」 に示す通り/etc/chrony/conf.d/initstepslew.conf と /etc/chrony/conf.d/servers.conf に記載します：

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

図 5.165 chronyd のコンフィグの変更例

❶ 起動時のサーバー設定です。不要な場合は空のファイルを生成してください。

❷ 運用時のサーバー設定です。複数の行または「pool」の設定も可能です。

❸ ファイルを保存します。

❹ chronyd サービスを再起動します。

❺ chronyc で新しいサーバーが使用されていることを確認します。

NTP の設定は ABOS Web や Rest API を使って行うこともできます。詳細は、「10.9.5. 時刻設定」および 「10.9.6.13. Rest API : 時刻の設定」 を参照してください。

5.5.28. 拡張インターフェースを使用する

CON16 拡張インターフェースは、各ピンの機能をマルチプレクスすることで GPIO、I2C、UART などの機能を使用することが可能です。2.54mm ピッチのピンソケットを接続可能です。

拡張できる機能の詳細につきましては、「Armadillo-900 開発セット マルチプレクス表」 [https://armadillo.atmark-techno.com/resources/documents/armadillo-900/manual-multiplex]をご参照ください。

各ピンの設定は Device Tree で行います。Device Tree の変更については「10.18. Device Tree をカスタマイズする」をご参照ください。

5.5.28.1. ハードウェア仕様

表 5.48 CON16 信号配列

ピン番号	ピン名	I/O	A900 との接続	説明
1	VDD_5V	Power	-	電源(VDD_5V)
2	VDD_1V8	Power	-	電源(VDD_1V8)
3	GND	Power	-	電源(GND)
4	GND	Power	-	電源(GND)
5	PTA8	In/Out	PTA8	拡張入出力(1.8V)
6	PTC13	In/Out	PTC13	拡張入出力(1.8V)
7	PTA9	In/Out	PTA9	拡張入出力(1.8V)
8	PTC14	In/Out	PTC14	拡張入出力(1.8V)
9	PTA10	In/Out	PTA10	拡張入出力(1.8V)
10	PTC15	In/Out	PTC15	拡張入出力(1.8V)
11	PTA11	In/Out	PTA11	拡張入出力(1.8V)
12	PTC16	In/Out	PTC16	拡張入出力(1.8V)
13	LPUART0_RX	In/Out	PTA15	拡張入出力(1.8V)
14	PTC17	In/Out	PTC17	拡張入出力(1.8V)
15	LPUART0_TX	In/Out	PTA18	拡張入出力(1.8V)
16	PTC18	In/Out	PTC18	拡張入出力(1.8V)
17	LPUART0_CTS_B	In/Out	PTA16	拡張入出力(1.8V)
18	PTC19	In/Out	PTC19	拡張入出力(1.8V)
19	LPUART0_RTS_B	In/Out	PTA17	拡張入出力(1.8V)
20	PTC20	In/Out	PTC20	拡張入出力(1.8V)
21	PTB4	In/Out	PTB4	拡張入出力(1.8V)
22	PTC21	In/Out	PTC21	拡張入出力(1.8V)
23	PTB5	In/Out	PTB5	拡張入出力(1.8V)
24	PTC22	In/Out	PTC22	拡張入出力(1.8V)
25	PTB12	In/Out	PTB12	拡張入出力(1.8V)
26	PTC23	In/Out	PTC23	拡張入出力(1.8V)
27	PTB14	In/Out	PTB14	拡張入出力(1.8V)
28	PTF6	In/Out	PTF6	拡張入出力(1.8V)
29	LPI2C7_SCL	Out	PTF4	I2C SCL 信号 4.7kΩ プルアップ
30	PTF7	In/Out	PTF7	拡張入出力(1.8V)
31	LPI2C7_SDA	In/Out	PTF5	I2C SDA 信号 4.7kΩ プルアップ
32	PTF24	In/Out	PTF24	拡張入出力(1.8V)
33	ONOFF	In	ONOFF	ONOFF 信号
34	GND	Power	-	電源(GND)

5.5.28.2. ソフトウェア仕様

表 5.49 デフォルトのマルチプレクス

ピン番号	ピン名	A900 との接続	デフォルトのマルチプレクス
1	VDD_5V	-	-
2	VDD_1V8	-	-
3	GND	-	-
4	GND	-	-
5	PTA8	PTA8	GPIO
6	PTC13	PTC13	GPIO
7	PTA9	PTA9	GPIO

ピン番号	ピン名	A900 との接続	デフォルトのマルチプレクス
8	PTC14	PTC14	GPIO
9	PTA10	PTA10	GPIO
10	PTC15	PTC15	GPIO
11	PTA11	PTA11	GPIO
12	PTC16	PTC16	GPIO
13	LPUART0_RX	PTA15	GPIO
14	PTC17	PTC17	GPIO
15	LPUART0_TX	PTA18	GPIO
16	PTC18	PTC18	GPIO
17	LPUART0_CTS_B	PTA16	GPIO
18	PTC19	PTC19	GPIO
19	LPUART0_RTS_B	PTA17	GPIO
20	PTC20	PTC20	GPIO
21	PTB4	PTB4	GPIO
22	PTC21	PTC21	GPIO
23	PTB5	PTB5	GPIO
24	PTC22	PTC22	GPIO
25	PTB12	PTB12	GPIO
26	PTC23	PTC23	GPIO
27	PTB14	PTB14	GPIO
28	PTF6	PTF6	GPIO
29	LPI2C7_SCL	PTF4	I2C SCL 信号
30	PTF7	PTF7	GPIO
31	LPI2C7_SDA	PTF5	I2C SDA 信号
32	PTF24	PTF24	GPIO
33	ONOFF	ONOFF	-
34	GND	-	-

5.5.29. 起動デバイスを変更する

SW4 はブートモード切替スイッチです。SW4 を操作することで、起動デバイスを設定することができます。

5.5.29.1. ハードウェア仕様

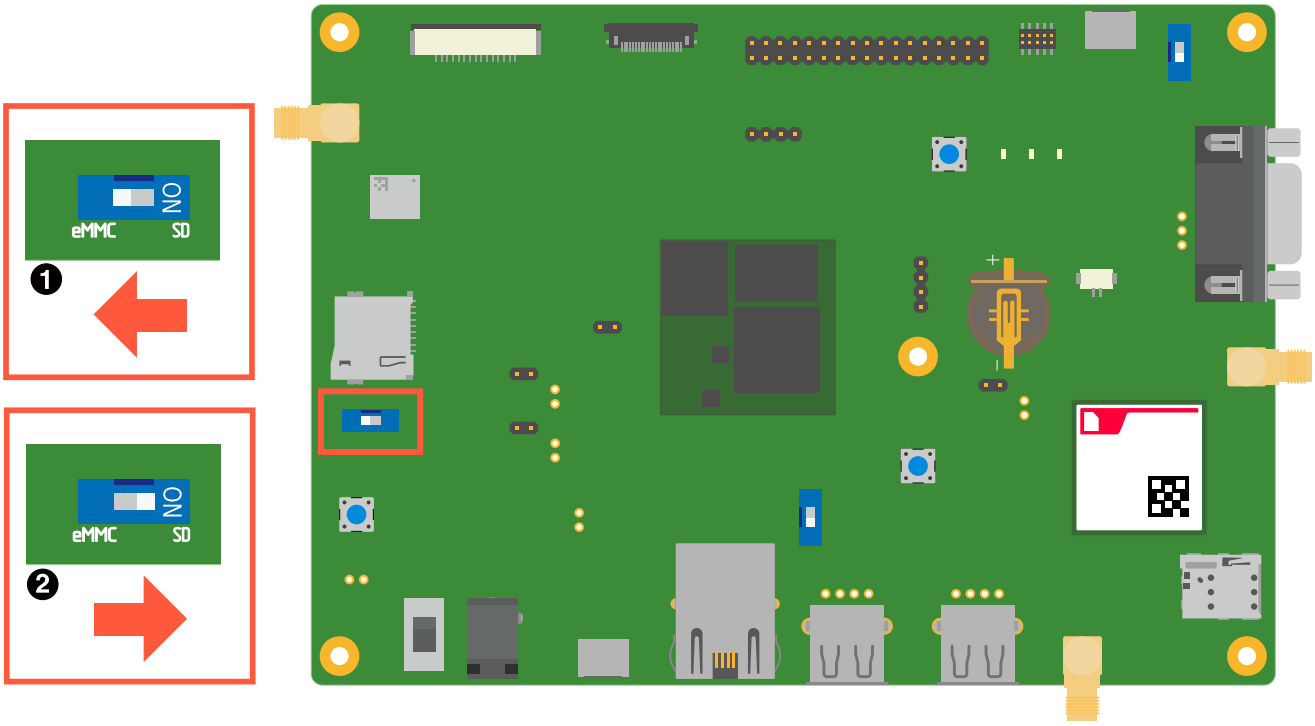


図 5.166 スイッチの状態と起動デバイス

❶ 起動デバイスは eMMC になります。

❷ 起動デバイスは microSD になります。

起動デバイス設定スイッチの両脇の基板上に、白い文字で eMMC/SD とシルク記載しているので、操作の目印にご利用ください。

5.6. パワーマネジメント・省電力・間欠動作

Armadillo-900 開発セット では、特にバッテリー駆動時などで必要となる、省電力・間欠動作の機能を用意しています。どのタイミングで DeepSleep・Shutdown モードへ遷移するか、何をトリガーとして起床するかを設定できます。本節では、Armadillo-900 開発セット の 省電力・間欠動作機能や動作モード、状態遷移について説明します。

5.6.1. 間欠動作モード・起床条件と状態遷移図

Armadillo-900 開発セット の動作モード・起床条件と状態遷移を「図 5.167. 状態遷移図」に示します。また、動作モード毎のデバイス状態を「表 5.50. 動作モード別デバイス状態」に示します。

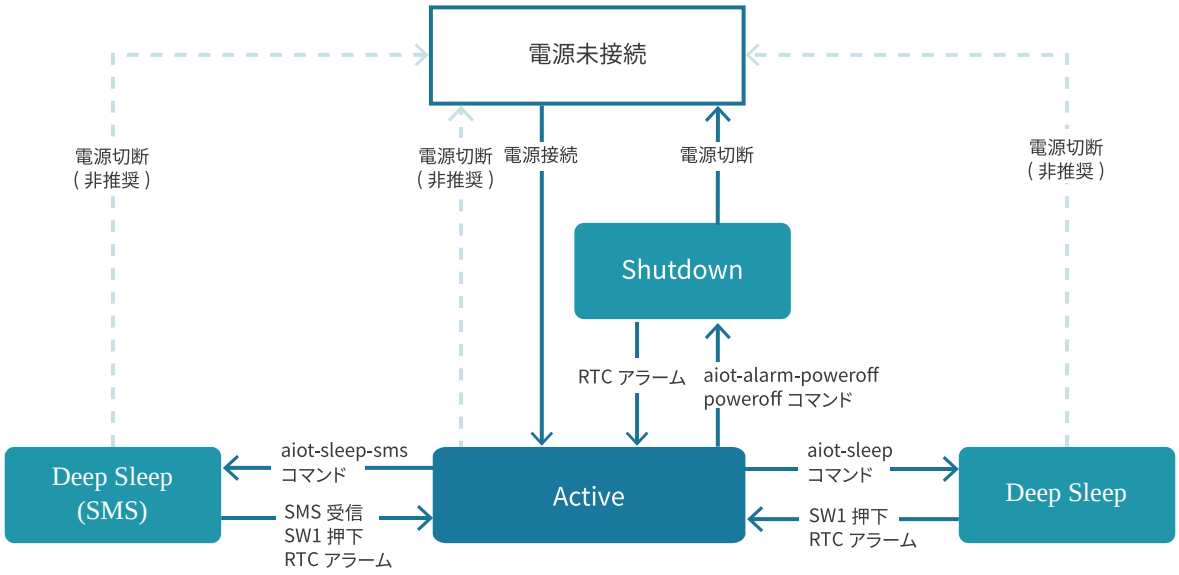


図 5.167 状態遷移図

表 5.50 動作モード別デバイス状態

動作モード	CPU	LTE	LED	有線 LAN	USB など
Active	動作	通信	動作	動作	通電
Shutdown	停止	停止	消灯	停止	停止
DeepSleep	suspend-to-RAM	動作 [a]	消灯	停止	通電
DeepSleep(SMS)	suspend-to-RAM	動作 [a]	消灯	停止	通電

[a]LTE 通信は停止し、LTE モジュールは動作している状態です。

5.6.1.1. 間欠動作モード・起床条件

次に、各動作モードと利用することのできる起床条件について説明します。

5.6.1.2. Active モード

「CPU:動作」、「LTE:動作」 状態のモードです。

Armadillo-900 開発セットの電源投入後 Linux カーネルが起動し、まずは Active モードに遷移します。

任意のアプリケーションの実行や、外部センサー・デバイスの制御、LTE や Ethernet での通信が可能ですが、最も電力を消費するモードです。Active モードの時間をより短くすることで、消費電力を押さえることができます。

5.6.1.3. Shutdown モード

「CPU:停止」、「LTE:停止」の状態であり最も消費電力を抑えることのできるモードです。

その反面、CPU を停止させ、Linux カーネルをシャットダウンしている状態であるため、 Active モードに遷移する場合は Linux カーネルの起動分の時間がかかります。

Shutdown モードから Active モードに遷移するには、RTC のアラーム割り込みを使用するか、一度電源を切断・再接続を行う必要があります。

5.6.1.4. Deep Sleep モード

「CPU:待機」、「LTE:停止」 状態のモードです。

CPU(i.MX 8ULP)はパワーマネジメントの Suspend-to-RAM 状態になり、Linux カーネルは Pause の状態になります。Shutdown モードと比較すると消費電力は高いですが、Linux カーネルの起動は不要であるため数秒程度で Active モードに遷移が可能です。ユーザスイッチの押下、RTC アラーム割り込みによって Active モードへの遷移ができます。

LTE 接続中に Deep Sleep モードをご利用になる場合、 Deep Sleep モードから Active モードへ遷移するタイミングで ping による LTE 通信の疎通確認を実施します。

ping 疎通確認先の IP アドレスは以下の順序・ルールで決定します。「5.2.7. LTE 再接続サービス」 で使用している設定ファイルを参照しています。

1. /etc/atmark/connection-recover.conf が存在してファイル内に PING_DEST_IP があれば、この値を使用します。
2. 存在しない場合は、8.8.8.8 を疎通先として使用します。

5.6.1.5. Deep Sleep(SMS) モード

「CPU:待機」、「LTE:待機」 状態のモードです。

Deep Sleep モードとの違いは、SMS の受信によって、 Active モードへの遷移も可能である点です。LTE:待機(PSM)の状態であるため、 Deep Sleep モードよりも電力を消費します。

5.6.2. Shutdown モードへの遷移と起床

Shutdown モードへ遷移するには、poweroff コマンド、または aiot-alarm-poweroff コマンドを実行します。

5.6.2.1. poweroff コマンド

poweroff コマンドを実行して Shutdown モードに遷移した場合、電源の切断・接続のみで Active に遷移が可能です。poweroff コマンドの実行例を次に示します。

```
[armadillo ~]# poweroff
podman-atmark            | * Stopping all podman containers ...loca
l                    | * Stopping local ... [ ok ]
avahi-daemon             | * Stopping avahi-daemon ...zramswap                 |
 * Deactivating zram swap device ...modemmanager             | * Stopping modemm
anager ...sim7672-boot             | * Stopping sim7672-boot ...wwan-led
         | * Stopping wwan-led ... [ ok ] [ ok ]

※省略

Sent SIGTERM to all processes
Sent SIGKILL to all processes
Requesting system poweroff
[  232.390025] failed to disconnect on suspend
```

5.6.2.2. aiot-alarm-poweroff コマンド

aiot-alarm-poweroff コマンドを実行することで、 Shutdown モードに遷移後、RTC のアラーム割り込みをトリガで起床（ Active モードに遷移）することができます。なお、RTC を起床要因に使って間欠動作させる場合は、 「5.5.20. RTC を使用する」 を参考に、必ず RTC の日時設定を行ってください。

RTC 未設定によるエラーが発生した場合、 Shutdown モードへの遷移は行われません。

```
[armadillo ~]# aiot-alarm-poweroff +[現在時刻からの経過秒数]
```

図 5.168 aiot-alarm-poweroff コマンド書式

Shutdown モードに遷移し、300 秒後にアラーム割り込みを発生させるには、次のようにコマンドを実行します。

```
[armadillo ~]# aiot-alarm-poweroff +300
aiot-alarm-poweroff: alarm_timer +300 second
```

現在時刻からの経過秒数は 180 秒以上を指定する必要があります。

5.6.3. Deep Sleep への遷移と起床

aiot-sleep コマンドを実行することで、 Deep Sleep モードに遷移することができます。ユーザースイッチによる起床は標準で有効になっています。

5.6.3.1. RTC アラーム割り込み以外での起床

SW1 が押下された時に Deep Sleep モードから起床するには、次に示すコマンドを実行します。

```
[armadillo ~]# aiot-sleep
modemmanager             | * Stopping modemmanager ... [ ok ]
connection-recover       | * Stopping connection-recover ... [ ok ]
OK
aiot-sleep: Power Management suspend-to-ram

※ SW1 を押下

I/TC: Secondary CPU 1 initializing
I/TC: Secondary CPU 1 switching to normal world boot
[  247.110778] fec 29950000.ethernet eth0: Graceful transmit stop did not complete!
aiot-sleep: change mode CPU Idle
OK
modemmanager             | * Starting modemmanager ... [ ok ]
connection-recover       | * Starting connection-recover ... [ ok ]
```

5.6.3.2. RTC アラーム割り込みでの起床

RTC アラーム割り込みでの起床を行う場合、パラメーター設定が異なります。なお、RTC を起床要因に使って間欠動作させる場合は、 「5.5.20. RTC を使用する」 を参考に、必ず RTC の日時設定を行ってください。

RTC アラーム割り込みでの起床は、毎分 00 秒で起床する分指定 (Armadillo-900 搭載の RTC アラーム割り込みを用いた起床) と秒指定 (SoC 内蔵の RTC アラーム割り込みを用いた起床) の 2 種類があります。現状のソフトウェアでは、分指定にのみ対応しています。

分指定のコマンド書式を 「図 5.169. コマンド書式 (RTC アラーム割り込みでの起床の場合: 分指定)」 に示します。

```
[armadillo ~]# echo +<現在時刻からの経過秒数> > /sys/class/rtc/rtc0/wakealarm
```

図 5.169 コマンド書式 (RTC アラーム割り込みでの起床の場合: 分指定)

現在時刻からの経過秒数 は 60 秒以上を指定する必要があります。

300 秒後に RTC アラーム割り込みを発生させ、 Deep Sleep モードから起床させるコマンド実行例を以下に示します。

```
[armadillo ~]# echo +300 > /sys/class/rtc/rtc0/wakealarm
[armadillo ~]# aiot-sleep
modemmanager             | * Stopping modemmanager ... [ ok ]
connection-recover       | * Stopping connection-recover ... [ ok ]
OK
aiot-sleep: Power Management suspend-to-ram

※ 約 300 秒待つ

I/TC: Secondary CPU 1 initializing
I/TC: Secondary CPU 1 switching to normal world boot
[  247.110778] fec 29950000.ethernet eth0: Graceful transmit stop did not complete!
aiot-sleep: change mode CPU Idle
OK
modemmanager             | * Starting modemmanager ... [ ok ]
connection-recover       | * Starting connection-recover ... [ ok ]
```

5.6.4. Deep Sleep(SMS)モードへの遷移と起床

aiot-sleep-sms コマンドを実行することで、 Deep Sleep(SMS) モードに遷移することができます。ユーザースイッチによる起床は標準で有効になっています。aiot-sleep-sms コマンドを実行した場合 SMS 受信による起床は強制的に有効になります。

aiot-sleep-sms コマンドの実行例を次に示します。

```
[armadillo ~]# aiot-sleep-sms
aiot-sleep-sms: terminate dialup
Connection 'gsm-ttyCommModem' successfully deactivated (D-Bus active path: /org/
freedesktop/NetworkManager/ActiveConnection/4)
modemmanager             | * Stopping modemmanager ... [ ok ]
connection-recover       | * Stopping connection-recover ... [ ok ]
```

```
OK
aiot-sleep-sms: Power Management suspend-to-ram

※ SMS 受信

I/TC: Secondary CPU 1 initializing
I/TC: Secondary CPU 1 switching to normal world boot
[  290.472971] fec 29950000.ethernet eth0: Graceful transmit stop did not complete!
aiot-sleep-sms: change mode CPU Idle
OK
modemmanager             | * Starting modemmanager ... [ ok ]
aiot-sleep-sms: redial.
connection-recover       | * Starting connection-recover ... [ ok ]
```

ご利用の SMS 送信サービスの SMS 送信制限により SMS の送信ができないことがあります。また、ネットワーク状態によって SMS の受信を検知できなかったり、検知が遅れることがあります。

起床要因として SMS のみを設定されるシステムを想定されている場合は、上記検知できない可能性を考慮して RTC など別な起床要因で周期的に起床することを推奨します。

また「5.2.6.2. 省電力などの設定」の初期値では、SMS 受信を検知して起床するまでに最長で 3 分かかります。より短時間で起床する必要がある場合は psm と edrx を disable に設定する対応をご検討ください。

aiot-sleep-sms で Deep Sleep(SMS) モードへ遷移する際、LTE モジュールの SMS 保存用ストレージに空きがない場合 SMS 受信での起床ができなくなるため、LTE モジュールのストレージから 1 件 SMS を削除してから Deep Sleep(SMS) モードへ遷移します。

SMS で受信した内容が必要な場合は、 SMS の内容を別なファイルなどに保存してから aiot-sleep-sms を実施してください。

5.6.5. 消費電流 を測定する

Armadillo-900 開発セットでは、4 つの電源系統について、それぞれ 2 つの方法で電流測定が可能です。電流計を使用して電流測定を行う場合は、「表 5.51. 電流測定可能箇所」に示す 0Ω 抵抗を外しピンヘッダに電流計を接続します。シャント抵抗と電圧計を使用して電流測定を行う場合は、「表 5.51. 電流測定可能箇所」に示す 0Ω 抵抗をシャント抵抗に変更し、シャント抵抗の両端に電圧計を接続します。使用可能なシャント抵抗のサイズは 3.2 x 1.6mm です。

表 5.51 電流測定可能箇所

電源系統	ピンヘッダ	0Ω 抵抗	説明
VSYS_5V	JP1	R286	Armadillo-900 への入力電源(5V)
VDD_3V3	JP2	R291	3.3V 系電源
VDD_1V8	JP3	R292	1.8V 系電源

電源系統	ピンヘッダ	0Ω 抵抗	説明
VDD_3V8	JP4	R293	LTE モジュールへの入力電源(3.8V)

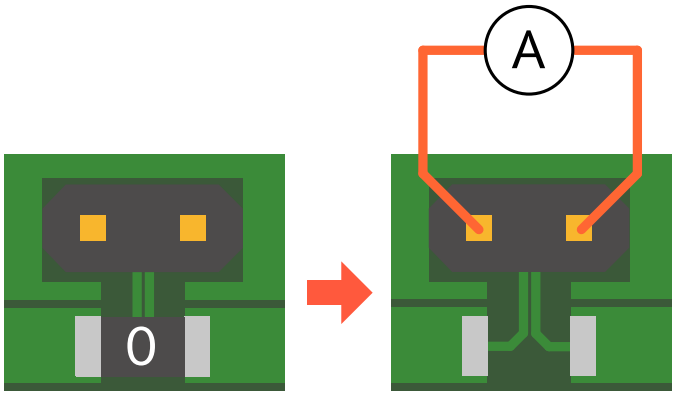


図 5.170 電流測定(ピンヘッダに電流計を接続)

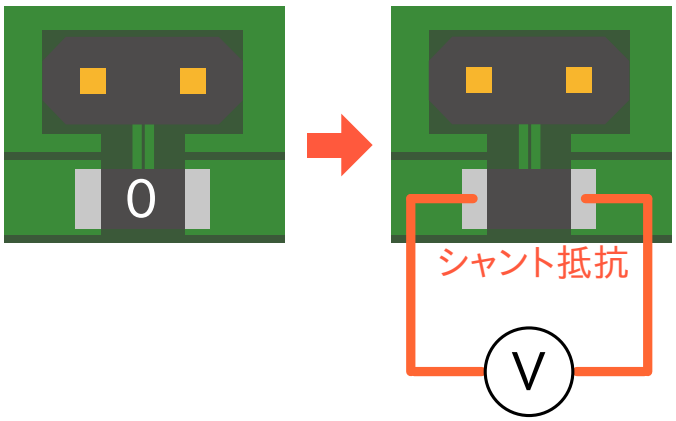


図 5.171 電流測定(シャント抵抗に電圧計を接続)

5.7. Twin に接続する

5.7.1. Armadillo Twin を体験する

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

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

5.7.2. Armadillo Twin を契約する

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

5.7.3. Armadillo Twin に Armadillo を登録する

5.7.3.1. Armadillo の設置前に登録する場合

Armadillo を Armadillo Twin に登録する場合、ケース裏や基板本体に貼付されているシール上の QR コードを使用します。登録方法についての詳細は Armadillo Twin ユーザーマニュアル 「Armadillo Twin にデバイスを登録する」 [https://manual.armadillo-twin.com/register-device/] をご確認ください。

5.7.3.2. Armadillo の設置後に登録する場合

Armadillo 設置後の登録については、弊社営業までお問い合わせください。

5.7.4. Armadillo Twin から複数の Armadillo をアップデートする

Armadillo Twin を使用することで、自身でサーバー構築を行うことなくネットワーク経由で SWU イメージを配信し、デバイスのソフトウェアを更新することができます。Armadillo Twin を使用したソフトウェアアップデートを行うためには、Armadillo Twin へのデバイスの登録が完了している必要があります。Armadillo Twin へのデバイスの登録方法については、「5.7.3. Armadillo Twin に Armadillo を登録する」 をご確認ください。また、Armadillo Twin を使用したソフトウェアアップデートの実施方法については、 Armadillo Twin ユーザーマニュアル 「デバイスのソフトウェアをアップデートする」 [https://manual.armadillo-twin.com/update-software/] をご確認ください。

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

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

6. ハードウェアの設計情報

6.1. ハードウェア設計情報のダウンロード

Armadillo-900 量産ボードを使用した基板設計の参考資料として、Armadillo-900 開発ボードの回路図・部品表を公開しています。「アットマークテクノ Armadillo サイト」 [https://armadillo.atmark-techno.com/]から「購入者向けの限定公開データ」としてダウンロード可能です。

Armadillo-900 開発ボードの回路図・部品表に記載している部品については、既に販売終了していたり、終息部品になっている可能性があります。部品採用を決定する前に、各部品の現在の状況について十分ご確認ください。

6.2. 電気的仕様

6.2.1. 絶対最大定格

表 6.1 絶対最大定格

項目	記号	Min.	Max.	単位	備考
電源電圧	VIN	-0.3	26.4	V	CON19
入出力電圧(GPIO 信号)	VI,VO (VDD_1V8, VDD_VBAT18)	-0.3	OVDD +0.3	V	
USB コンソール電源電圧	VBUS_DEBUB0, VBUS_DEBUG1	-0.3	5.8	V	CON13, CON14
RTC バックアップ電源電圧	RTC_VDD	-0.3	5.5	V	CON11, CON12
動作温度範囲	Topr	10	40	℃	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。

6.2.2. 入出力仕様

・ 電源出力仕様

表 6.2 電源出力仕様

項目	記号	Min.	Typ.	Max.	単位	備考
5V 電源	VDD_5V USB0_VBUS USB1_VBUS	4.85	5	5.15	V	
3.3V 電源	VDD_3V3 VDD_SD_3V3	3.135	3.3	3.465	V	

項目	記号	Min.	Typ.	Max.	単位	備考
1.8V 電源	VDD_1V8 VDD_VBAT18	1.71	1.8	1.89	V	

・ 拡張インターフェース(CON16)の許容電流

表 6.3 拡張インターフェース(CON16)の許容電流

項目	記号	Max.	単位	備考
5V 電源	VDD_5V	0.5	A	
1.8V 電源	VDD_1V8	0.25	A	

・ 拡張インターフェース(CON16)の入出力仕様

表 6.4 拡張インターフェース(CON16)の入出力仕様

項目	記号	Min.	Max.	単位	備考
ハイレベル出力電圧	VOH(PTAx)	VDD_1V8-0.5	VDD_1V8	V	IOH = -10mA(DSE=1), -5mA(DSE=0)
	VOH(PTBx)	VDD_1V8-0.5	VDD_1V8	V	IOH = -10mA(DSE=1), -5mA(DSE=0)
	VOH(PTCx)	0.8xVDD_1V8	VDD_1V8	V	IOH = -0.1mA(DSE=1), -2mA(DSE=0)
	VOH(PTFx)	VDD_1V8-0.5	VDD_1V8	V	IOH = -10mA(DSE=1), -5mA(DSE=0)
ローレベル出力電圧	VOL(PTAx)	0	0.5	V	IOL = 10mA(DSE=1), 5mA(DSE=0)
	VOL(PTBx)	0	0.5	V	IOL = 10mA(DSE=1), 5mA(DSE=0)
	VOL(PTCx)	0	0.5	V	IOL = 0.1mA(DSE=1), 2mA(DSE=0)
	VOL(PTFx)	0	0.5	V	IOL = 10mA(DSE=1), 5mA(DSE=0)
ハイレベル入力電圧	VIH(PTAx)	0.75xVDD_1V8	VDD_1V8	V	
	VIH(PTBx)	0.75xVDD_1V8	VDD_1V8	V	
	VIH(PTCx)	0.7xVDD_1V8	VDD_1V8	V	
	VIH(PTFx)	0.75xVDD_1V8	VDD_1V8	V	
ローレベル入力電圧	VIL(PTAx)	0	0.3xVDD_1V8	V	
	VIL(PTBx)	0	0.3xVDD_1V8	V	
	VIL(PTCx)	0	0.3xVDD_1V8	V	
	VIL(PTFx)	0	0.3xVDD_1V8	V	
Pull-up/Pull-down 抵抗	R(PTAx)	25	50	kΩ	
	R(PTBx)	25	50	kΩ	
	R(PTCx)	20	50	kΩ	
	R(PTFx)	25	50	kΩ	

・ I2C(3.3V)インターフェース(CON17)の許容電流

表 6.5 I2C(3.3V)インターフェース(CON17)の許容電流

項目	記号	Max.	単位	備考
3.3V 電源	VDD_3V3	0.25	A	

・ DAC インターフェース(CON25)の許容電流

表 6.6 DAC インターフェース(CON25)の許容電流

項目	記号	Max.	単位	備考
1.8V 電源	VDD_1V8	0.25	A	

6.2.3. 電源回路の構成

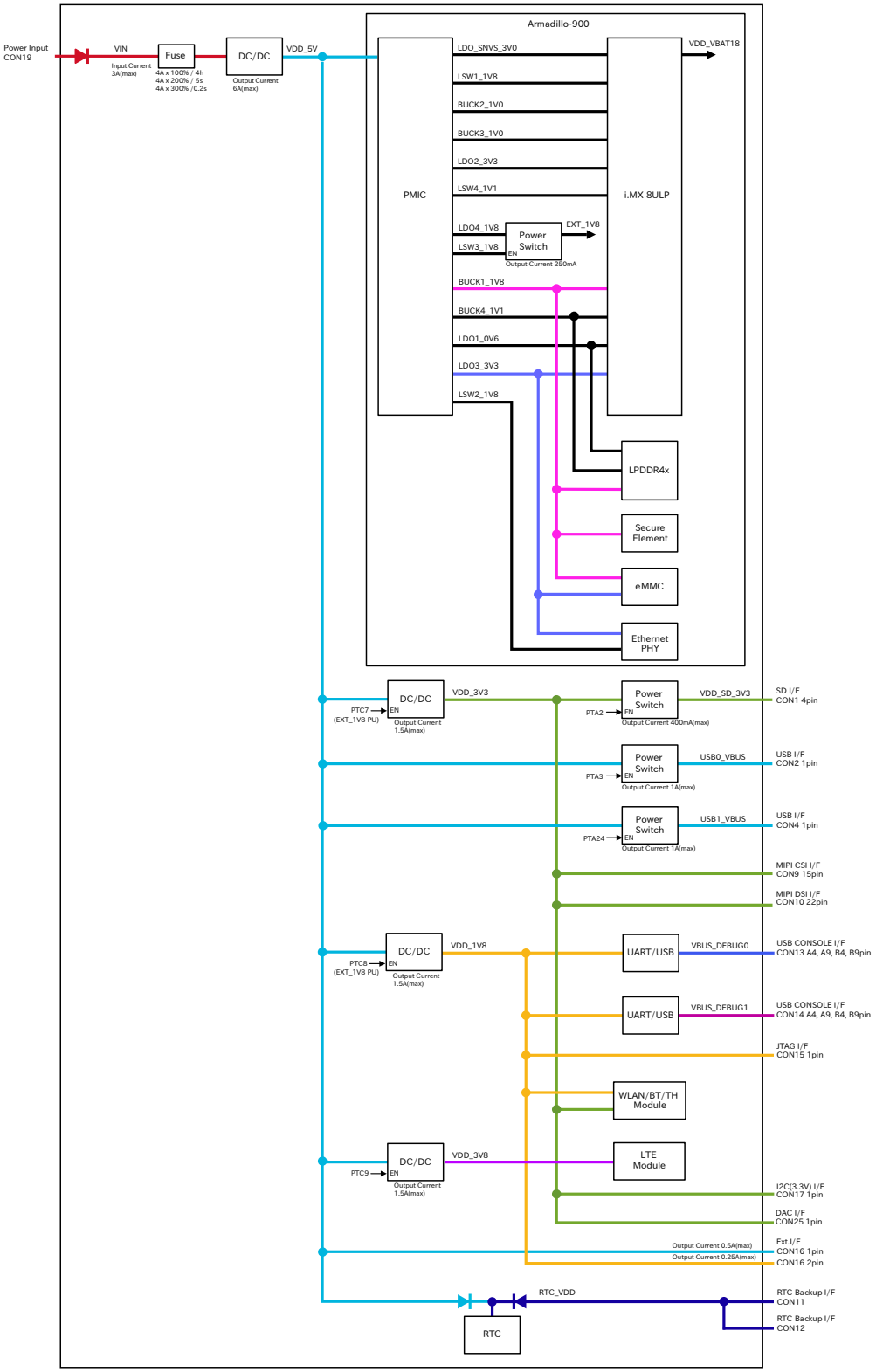


図 6.1 電源回路の構成

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。

6.2.4. 電源シーケンス

電源シーケンスは次のとおりです。

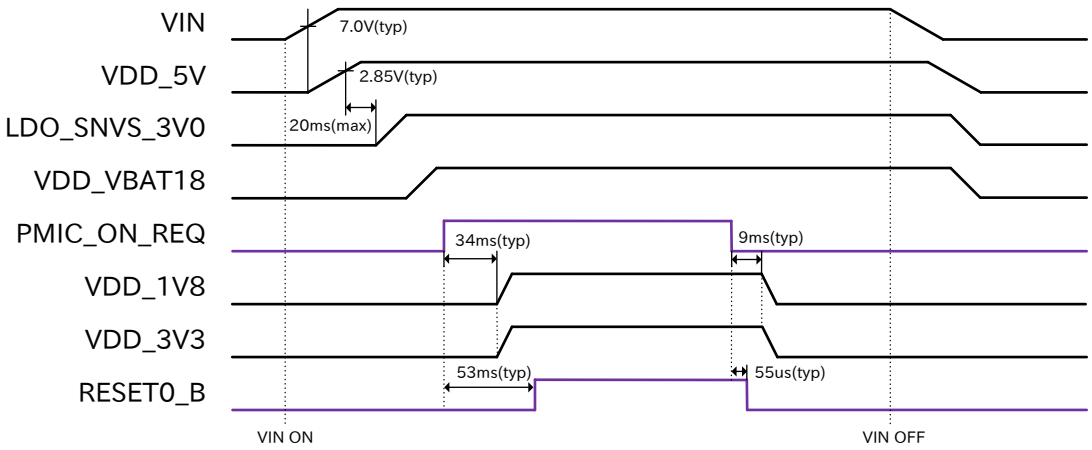


図 6.2 電源シーケンス

・電源オン時

Armadillo-900 開発セットに電源(VIN)を投入すると、VDD_5V、LDO_SNVS_3V0、VDD_VBAT18 の順で電源が立ち上がり、i.MX 8ULP からパワーマネジメント IC(PMIC)に PMIC_ON_REQ 信号が出力されます。パワーマネジメント IC(PMIC)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、各電源を立ち上げます。電源オンシーケンスが完了すると、RESET0_B 信号が解除され、Armadillo-900 開発セットはアクティブモードとなります。

・電源オフ時

poweroff コマンドにより、i.MX 8ULP が PMIC_ON_REQ 信号の出力を Low にすると、RESET0_B 信号がアサートされ Armadillo-900 開発セットはシャットダウンモードになります。また、PMIC_ON_REQ 信号が Low になると、パワーマネジメント IC(PMIC)は電源オフシーケンスを開始し、LDO_SNVS_3V0、VDD_VBAT18 以外の電源を立ち下げます。Armadillo-900 開発セットの電源(VIN)が切断されると、VIN、VDD_5V、LDO_SNVS_3V0、VDD_VBAT18 の順で電源が立ち下がります。

6.2.5. 各動作モードにおける電源供給状況

各動作モードにおける電源供給状況は以下の通りです。

表 6.7 各動作モードにおける電源供給状況

動作モード	VDD_5V	VDD_VBAT18	VDD_3V3	VDD_1V8
電源未接続	OFF	OFF	OFF	OFF
Shutdown	ON	ON	OFF	OFF
DeepSleep(SMS)	ON	ON	ON	ON
DeepSleep	ON	ON	ON	ON
Active	ON	ON	ON	ON

6.2.6. reboot コマンドによる再起動時の電源供給について

reboot コマンドで再起動した場合の各電源供給状況は以下の通りです。

表 6.8 reboot コマンドで再起動した場合の各電源供給状況

電源	供給状況
VDD_5V	供給を保持
VDD_VBAT18	供給を保持
VDD_3V3	一度 OFF した後 ON
VDD_1V8	一度 OFF した後 ON

6.2.7. 外部からの電源制御

- ONOFF ピンからの電源制御

  SW5 ONOFF 制御用スイッチ、拡張インターフェースの ONOFF 信号(CON16 33 ピン)およびリアルタイムクロックの割り込み信号は、i.MX 8ULP の ONOFF ピンに接続されています。

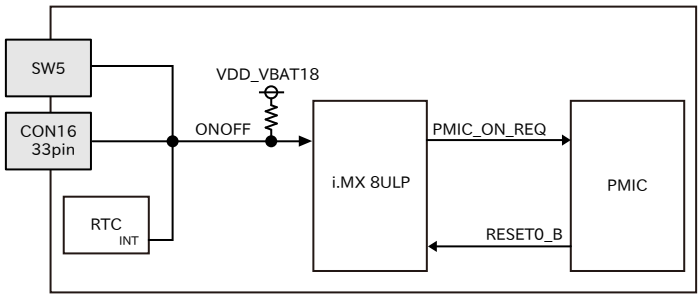


図 6.3 ONOFF 回路の構成

ONOFF 信号を一定時間以上、Low レベルとすることで、Armadillo-900 開発セットのアクティブモードとシャットダウンモードを切り替えることができます。アクティブモードで ONOFF 信号が 5 秒以上 Low レベルになると、PMIC_ON_REQ 信号がディアサートされ、シャットダウンモードに移行します。VIN、VDD_5V、LDO_SNVS_3V0、VDD_VBAT18 以外の電源は切断されます。また、シャットダウンモードで ONOFF 信号が 500 ミリ秒以上 Low レベルになると、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。

表 6.9 アクティブモード/シャットダウンモードを切り替える際に必要な Low レベル保持時間

状態	Low レベル保持時間
アクティブモードからシャットダウンモード	5 秒以上
シャットダウンモードからアクティブモード	500 ミリ秒以上

ONOFF 信号を制御する場合は、オープンドレイン出力等で GND とショートする回路を接続してください。シャットダウンモードは、VDD_VBAT18 が供給されている限り保持されます。

シャットダウンモードで Armadillo-900 開発セットの電源(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シャットダウンモードであることが保持されます。シャットダウンモード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

7. 開発編

Armadillo-900 開発セット では基本的に ATDE という Armadillo 専用開発環境と、 Visual Studio Code (以降 VS Code と記載します) 向け Armadillo 開発用エクステンションを用いてアプリケーション開発を行っていきます。

7.1. 開発の準備

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

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

7.1.1. 仮想環境のセットアップ

作業用 PC をセットアップします。アットマークテクノでは、製品のソフトウェア開発や動作確認を簡単に行うために、Oracle VirtualBox 仮想マシンのデータイメージを提供しています。このデータイメージを ATDE(Atmark Techno Development Environment) と呼びます。ATDE の起動には仮想化ソフトウェアである VirtualBox を使用します。

Oracle VirtualBox には以下の特徴があります。

- 基本パッケージが GPLv3(GNU General Public License Version 3) で提供されている
- VMware 形式の仮想ディスク(.vmdk)ファイルに対応している

7.1.1.1. VirtualBox のインストール

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

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

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

ホスト OS が Windows の場合はこの操作は必要ありません。

7.1.1.2. ATDE のアーカイブを取得

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

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

VirtualBox　　ATDE9 v20240925 以降

ATDE9 は Debian GNU/Linux 11 (コードネーム bullseye) をベースに、Armadillo-900 開発セット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-900 開発セット の動作確認を行うために必要なツールが事前にインストールされています。

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

7.1.1.3. ATDE のインポート

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

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

7.1.1.4. ATDE の起動

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

ATDE にログイン可能なユーザーを、「表 7.1. ユーザー名とパスワード」に示します [1]。

表 7.1 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

7.1.1.5. コマンドライン端末(GNOME 端末)の起動

Armadillo を利用した開発では、 CUI (Character-based User Interface)環境を提供するコマンドライン端末を通じて、 Armadillo や ATDE に対して操作を行う場面が多々あります。コマンドライン端末にはいくつかの種類がありますが、ここでは GNOME デスクトップ環境に標準インストールされている GNOME 端末を起動します。

GNOME 端末を起動するには、「図 7.1. 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 7.1 GNOME 端末の起動

「図 7.2. GNOME 端末のウィンドウ」のようにウィンドウが開きます。

[1]特権ユーザーで GUI ログインを行うことはできません

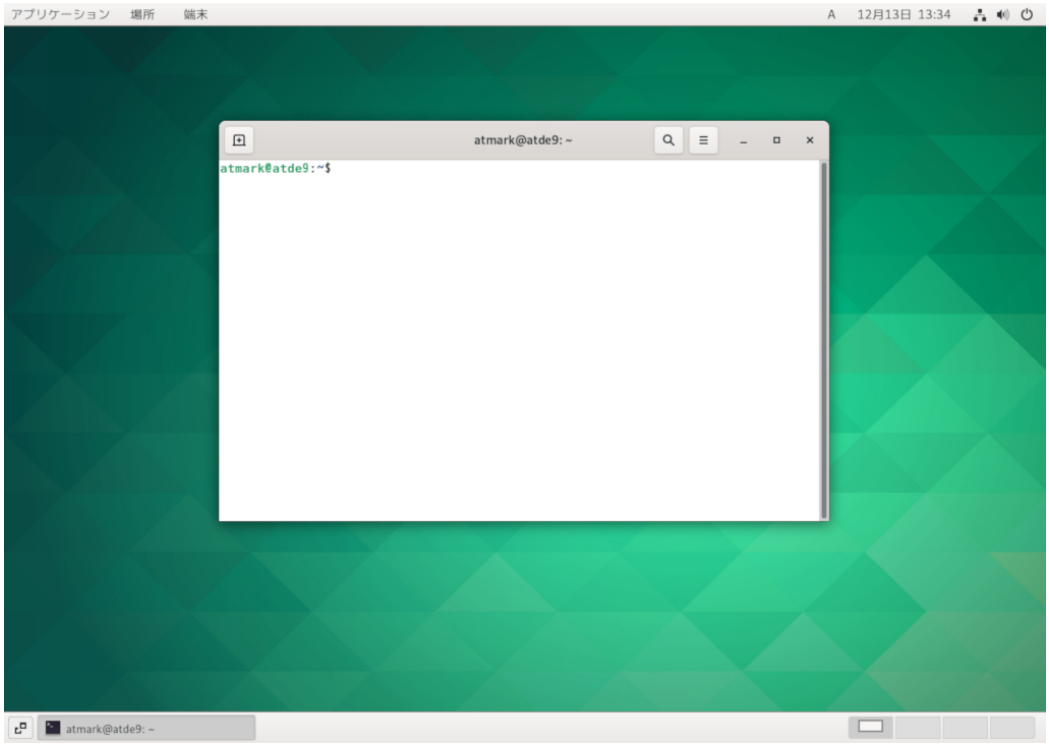


図 7.2 GNOME 端末のウィンドウ

7.1.1.6. ソフトウェアのアップデート

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

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

図 7.3 ソフトウェアをアップデートする

7.1.1.7. 取り外し可能デバイスの使用

VirtualBox は、ゲスト OS (ATDE)による取り外し可能デバイス(USB デバイスや DVD など)の使用をサポートしています。デバイスによっては、ホスト OS (VirtualBox を起動している OS)と ATDE で同時に使用することができません。そのようなデバイスを ATDE で使用するためには、ATDE にデバイスを接続する 「図 7.4. ATDE にデバイスを接続する」 の操作が必要になります。

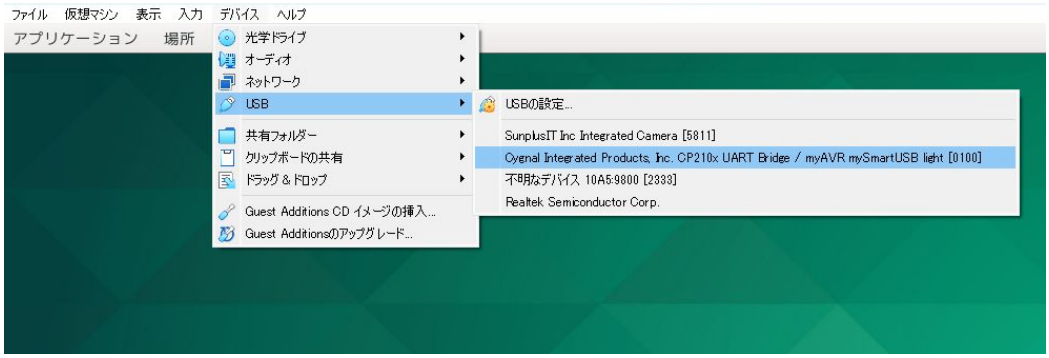


図 7.4 ATDE にデバイスを接続する

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

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

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

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

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

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

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

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

7.1.1.9. 共有フォルダーの作成

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

1. VirtualBox の上部バーから[デバイス]-[共有フォルダー]-[共有フォルダー設定]を選択します。(「図 7.5. 共有フォルダー設定を開く」)
2. 「図 7.6. 共有フォルダー設定」 の赤枠で示したアイコンをクリックします。
3. 「図 7.7. 共有フォルダーの追加」 のように、[フォルダーのパス]-[その他]を選択して、共有フォルダーに設定したいホスト OS 上のフォルダーを選択します。
4. 「図 7.7. 共有フォルダーの追加」 のように、[自動マウント]と[永続化する]にチェックを入れます。
5. [OK]をクリックして共有フォルダーを追加します。

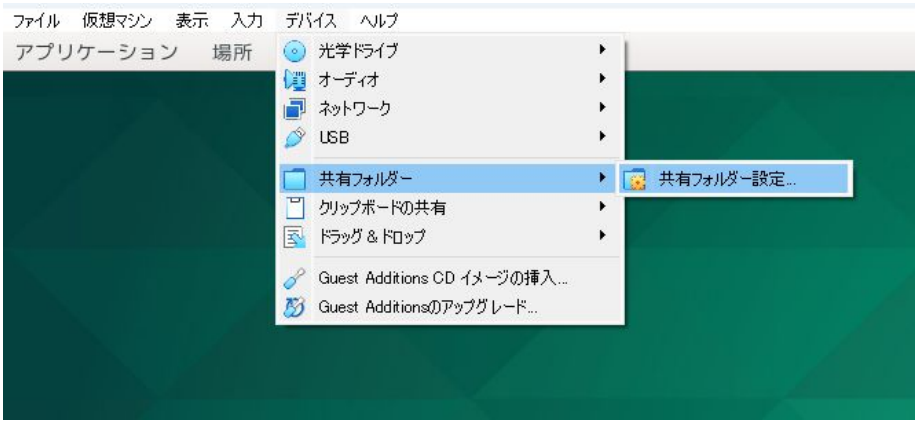


図 7.5 共有フォルダー設定を開く

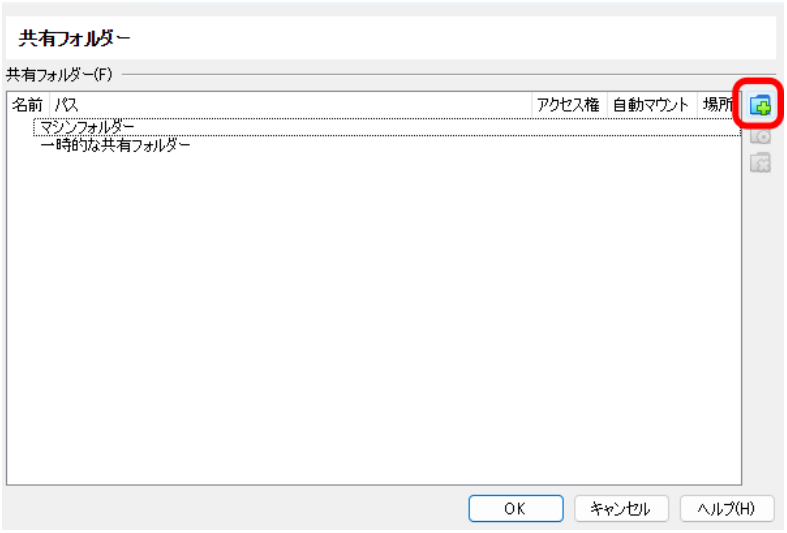


図 7.6 共有フォルダー設定

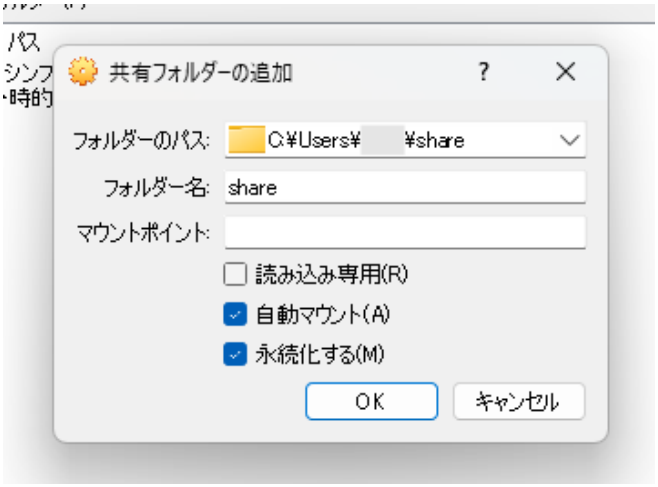


図 7.7 共有フォルダーの追加

図 7.8 「ファイル」に表示される共有フォルダー

追加した共有フォルダーは、「図 7.8. 「ファイル」に表示される共有フォルダー」のように「ファイル」からアクセスするか、または /media/sf_share（共有フォルダー名） からアクセスできます。(share というフォルダー名で作成すると、ATDE 上では sf_share として表示されます。)

7.1.2. VS Code のセットアップ

作業用 PC のセットアップです。Armadillo-900 開発セット の開発には、 VS Code を使用します。ATDE のバージョン v20230123 以上には、 VS Code がインストール済みのため新規にインストールする必要はありませんが、使用する前には「図 7.3. ソフトウェアをアップデートする」にしたがって最新版へのアップデートを行ってください。

以下の手順は全て ATDE 上で実施します。

7.1.2.1. VS Code を起動する

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

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

図 7.9 VS Code を起動する

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

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

VS Code 上でアプリケーションを開発するために、ABOSDE (Armadillo Base OS Development Environment) というエクステンションをインストールします。

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

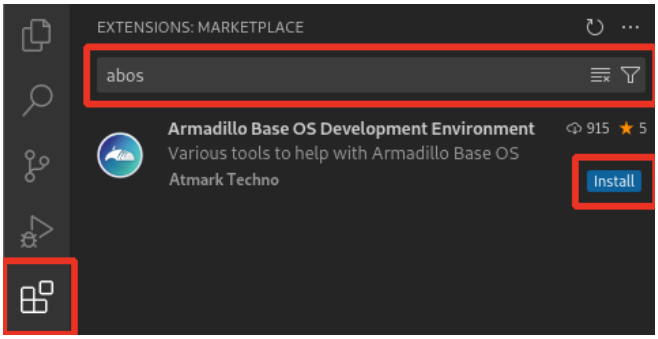


図 7.10 VS Code に開発用エクステンションをインストールする

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

7.1.3. Armadillo に初期設定をインストールする

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

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

7.1.3.1. initial_setup.swu の作成

「図 7.11. initial_setup.swu を作成する」 に示すように、VS Code の左ペインの [COMMON PROJECT COMMAND] から [Generate Initial Setup Swu] を実行してください。

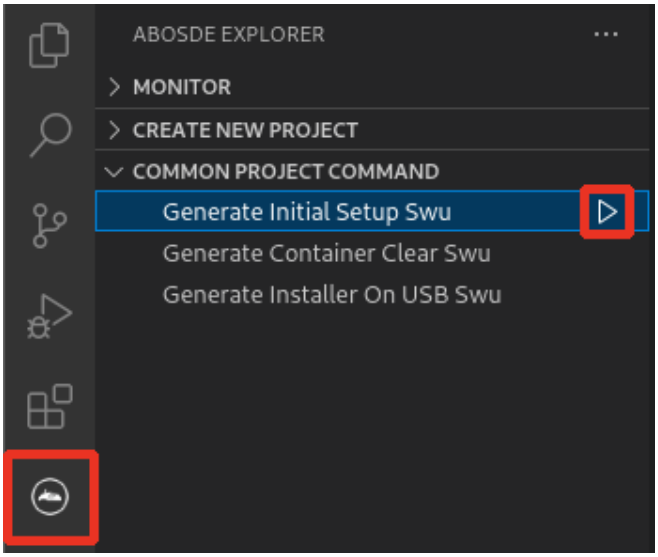


図 7.11 initial_setup.swu を作成する

初回実行時には各種設定の入力を求められます。入力する設定の内容を 「図 7.12. initial_setup.swu 初回生成時の各種設定」 に示します。

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

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

mkdir: ディレクトリ '/home/atmark/mkswu' を作成しました
設定ファイルを更新しました：/home/atmark/mkswu/mkswu.conf
証明書のコモンネーム(一般名)を入力してください: [COMMON_NAME] ❶
証明書の鍵のパスワードを入力ください（4-1024 文字）❷
証明書の鍵のパスワード（確認）：
Generating an EC private key
writing new private key to '/home/atmark/mkswu/swupdate.key.tmp'
 -----
アップデートイメージを暗号化しますか？（N/y）❸
アットマークテクノが作成したイメージをインストール可能にしますか？（Y/n）❹
root パスワード: ❺
root のパスワード（確認）：
atmark ユーザのパスワード（空の場合はアカウントをロックします）：❻
atmark のパスワード（確認）：
BaseOS/プリインストールコンテナを armadillo.atmark-techno.com サーバーから自動アップデートしますか？
(N/y) ❼
abos-web のパスワードを設定してください。
abos-web のパスワード（空の場合はサービスを無効にします）：❽
abos-web のパスワード（確認）：
/home/atmark/mkswu/initial_setup.swu を作成しました。
```

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

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

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

図 7.12 initial_setup.swu 初回生成時の各種設定

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

❷ 証明鍵を保護するパスフレーズを 2 回入力します。

❸ SWU イメージ自体を暗号化する場合に「y」を入力します。詳細は 「10.6. SWUpdate と暗号化について」 を参考にしてください。

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

❺ root のパスワードを 2 回入力します。使用するパスワードは以下のルールに従ってください。

- 辞書に載っている言葉を使用しない
- 単調な文字列を使用しない
- 8 文字以上のパスワード長にする

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

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

❽ abos-web を使用する場合はパスワードを設定してください。ここで設定したパスワードは abos-web から変更できます。使用できるパスワードの制限は root と同様です。詳細は 「5.1.4. ABOS Web のパスワード変更」 を参考にしてください。

❾ 作成したファイルを確認します。「swupdate.aes-key」は暗号化の場合にのみ作成されます。

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

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

上の手順で作成した SWU イメージ（initial_setup.swu）を Armadillo へインストールします。インストール方法は様々ありますが（「7.3.3.6. SWU イメージのインストール」）、ここでは ABOS Web を使用した手動インストールを行います。

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

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

まず、「図 7.13. ABOS にアクセスするための接続」のとおりに Armadillo に配線を行い、電源を入れてください。

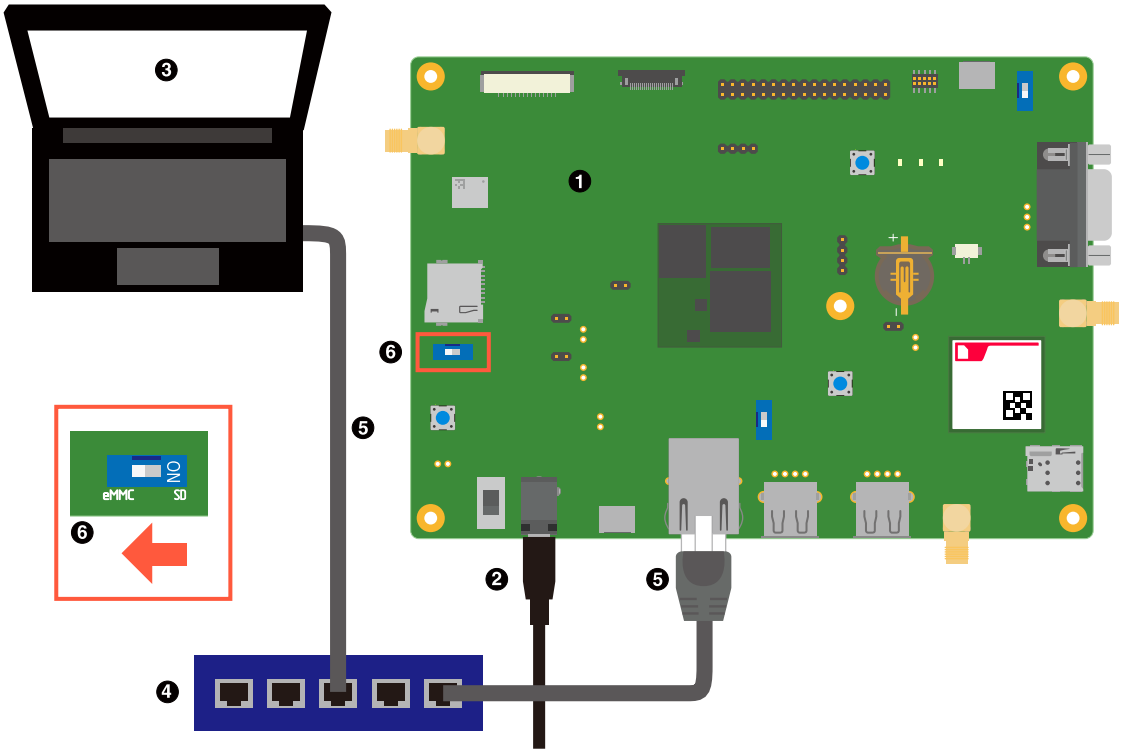


図 7.13 ABOS にアクセスするための接続

1. Armadillo-900 開発セット
2. AC アダプタ(12V/2.0A)
3. 作業用 PC
4. LAN HUB
5. Ethernet ケーブル
6. 起動デバイス設定スイッチ

1 分ほど待機して、ABOSDE でローカルネットワーク上の Armadillo をスキャンします。「図 7.14. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする」の赤枠で囲われているボタンをクリックしてください。

Armadillo が正常に起動していた場合、「図 7.15. ABOSDE に表示されている Armadillo を更新する」の一覧に起動した Armadillo が armadillo.local という名称で表示されます。表示されない場合は 1 分ほど待機してから「図 7.15. ABOSDE に表示されている Armadillo を更新する」の赤枠で囲われているマークをクリックしてスキャンを再度試みてください。

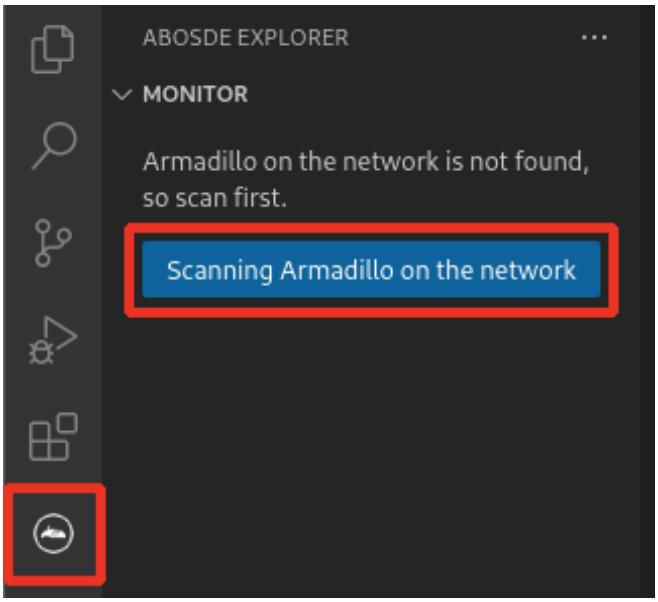


図 7.14 ABOSDE で ローカルネットワーク上の Armadillo をスキャンする

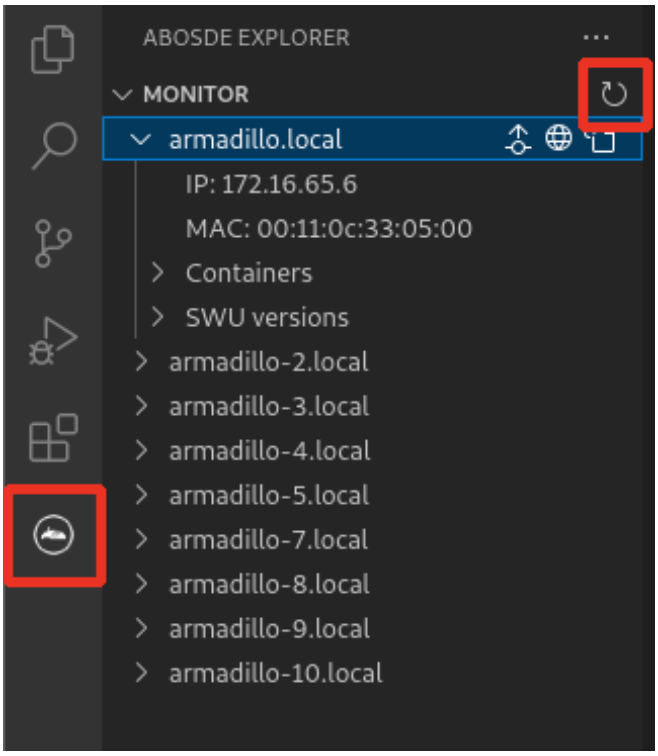


図 7.15 ABOSDE に表示されている Armadillo を更新する

ただし、ATDE のネットワークをブリッジ接続以外に設定している場合は Armadillo がリストに表示されない場合があります。表示するためには ATDE のネットワークをブリッジ接続に設定してください。また、ABOS Web が動作する Armadillo が同じ LAN 上に複数あると、ABOS Web に接続する URL のホスト名部分（armadillo.local）が、2 台目では armadillo-2.local、3 台目では armadillo-3.local のように、違うものが自動的に割り当てられます。目的の Armadillo がどのホスト名なのか不明な場合には、Armadillo のラベルに記載されている MAC アドレスと一致するもの（ 「図 7.16. ABOSDE を使用して ABOS Web を開く」 の赤枠に表示されます ）を探してください。

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

続いて、「図 7.16. ABOSDE を使用して ABOS Web を開く」 の赤枠で囲われているマークをクリックして、ABOS Web を Web ブラウザで開きます。

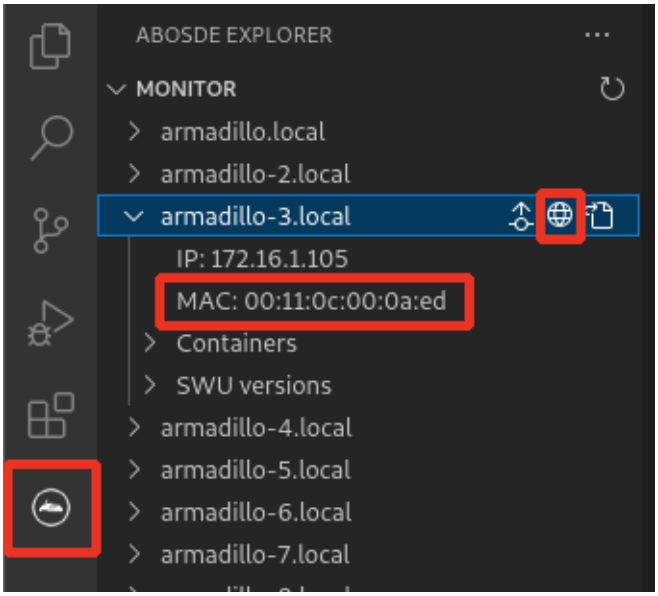


図 7.16 ABOSDE を使用して ABOS Web を開く

7.1.3.3. ABOS Web へアクセス

ABOS Web が正常に起動していれば、Web ブラウザに パスワード登録画面（ 「図 7.17. パスワード登録画面」 ）が表示されます。`initial_setup.swu` を作成する手順で設定したパスワードを入力して、ABOS Web のログイン用パスワードを設定します。

図 7.17 パスワード登録画面

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

図 7.18 パスワード登録完了画面

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

図 7.19 ログイン画面

ログインに成功すると、ABOS Web の設定画面（ 「図 7.20. トップページ」 ）に表示が変わり、設定操作を行うことができます。これで、ABOS Web へのアクセスが完了しました。

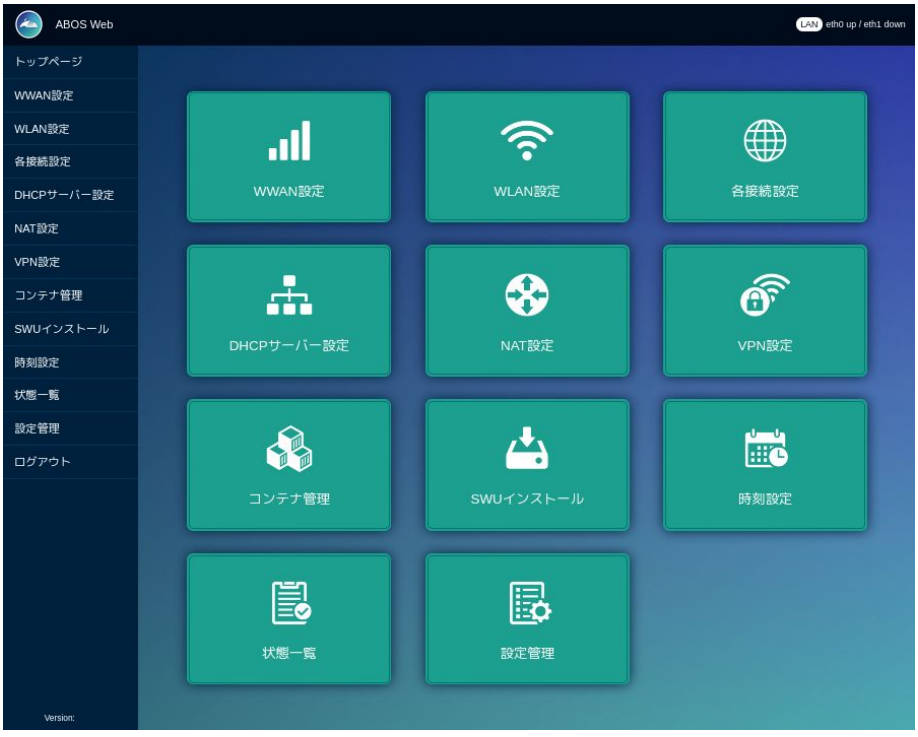


図 7.20 トップページ

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

ABOS Web のトップページから"SWU インストール"をクリックして、 「図 7.21. SWU インストール」の画面に遷移します。

図 7.21 SWU インストール

"参照…"から ~/mkswu/initial_setup.swu を選択し、"インストール"をクリックしてください。数分ほど待機すると 「図 7.22. SWU インストールに成功した画面」 のように"インストールが成功しました。"と表示され、Armadillo が再起動します。(ABOS Web も再起動されるので、再起動完了後にページを更新するとログイン画面に戻ります)

図 7.22 SWU インストールに成功した画面

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

ABOS Web にブラウザから直接アクセスする

ABOSDE を使わずに、 直接 Web ブラウザのアドレスバーに ABOS Web の URL を入力することでも ABOS Web にアクセスできます。ATDE で Web ブラウザを起動した後、Web ブラウザのアドレスバーに次の URL を入力してください： https://armadillo.local:58080

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

また、Web ブラウザから直接アクセスする方法では、ホスト名ではなく IP アドレスを指定することもできます。例えば、Armadillo の（ネットワークコネクタの）IP アドレスが 192.0.2.80 である場合は、次の URL を入力してください： https://192.0.2.80:5808

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

ABOS Web からログアウトする

ログアウトを行う場合は、サイドメニューから "ログアウト" を選択してください。ログアウトすると、ログイン画面が再び表示されますので、ABOS Web をすぐに使わないのであれば、Web ブラウザを閉じてください。

7.1.4. Python アプリケーションで動作確認する

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

7.1.4.1. プロジェクトの作成

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

VS Code の左ペインの [A9E] から [Python New Project] を実行（右に表示されている三角形ボタン）し、表示されるディレクトリ選択画面からプロジェクトを保存するディレクトリを選択してください。保存先を選択すると、プロジェクト名を入力するダイアログが表示されるので、任意のプロジェクト名を入力してエンターキーを押してください。この操作により、選択した保存先に、入力したプロジェクト名と同名のディレクトリが作成されます。

また、ここでは次のように設定しています。

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

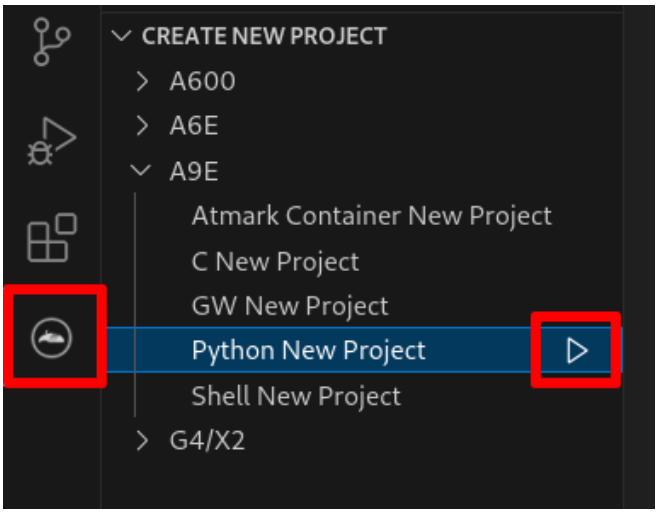


図 7.23 プロジェクトを作成する

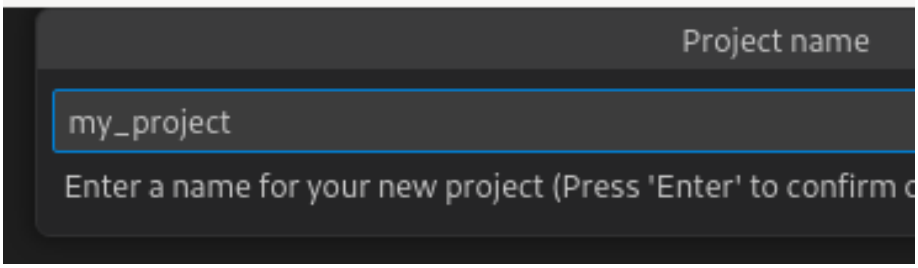


図 7.24 プロジェクト名を入力する

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

7.1.4.2. 初期設定

プロジェクトを作成する度に、初期設定を行う必要があります。初期設定では主に Armadillo と SSH で接続するための秘密鍵と公開鍵の生成を行います。以下の手順を実施してください。

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

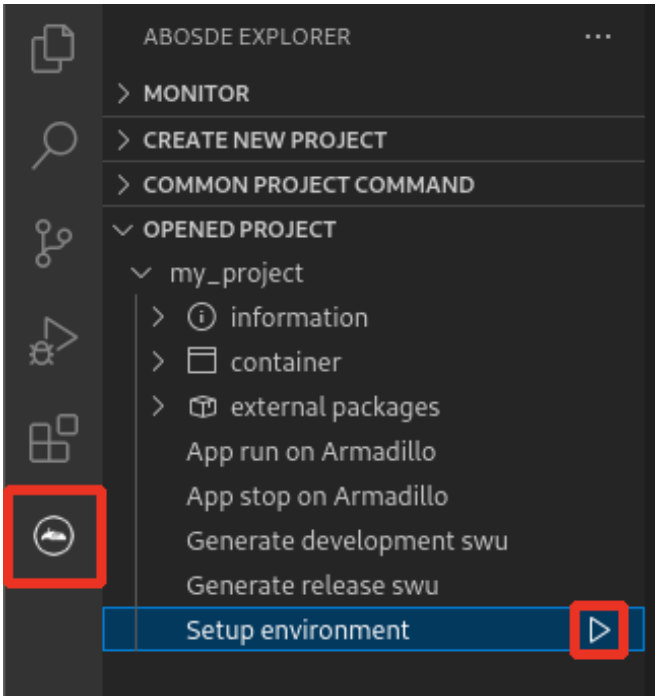


図 7.25 VS Code で初期設定を行う

選択すると、 VS Code の下部に以下のようなターミナルが表示されます。

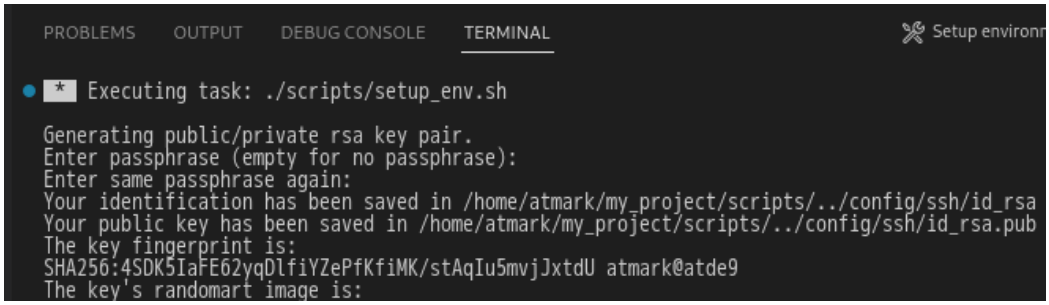


図 7.26 VS Code のターミナル

このターミナル上で以下のように入力してください。

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

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

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

図 7.27 SSH 用の鍵を生成する

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

❷ 1 でパスフレーズを設定した場合は、確認のため再度入力してください。

❸ ここで何か任意のキーを押すとターミナルが閉じます。

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

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

7.1.4.3. アプリケーション実行用コンテナイメージの作成

Armadillo 上でアプリケーションを実行するためのコンテナイメージを作成します。ここで作成したコンテナイメージは SWU イメージを使用して Armadillo へインストールします。

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

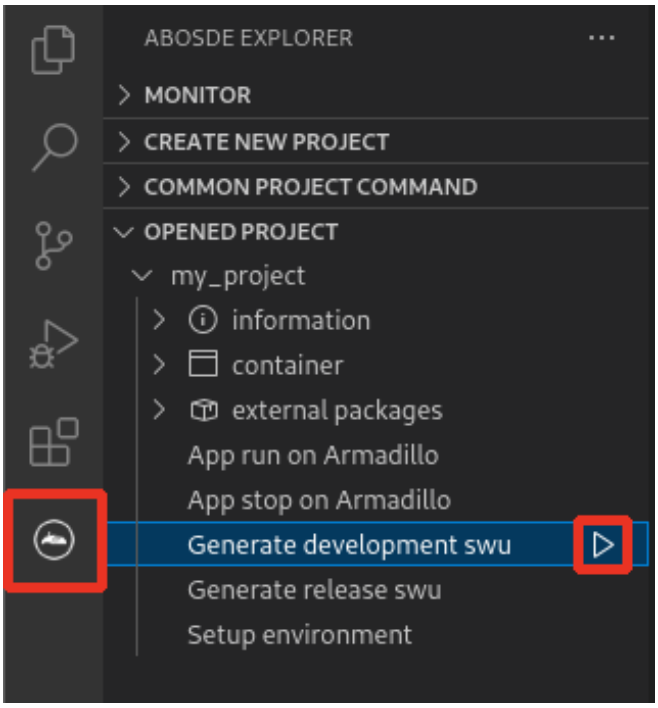


図 7.28 VS Code でコンテナイメージの作成を行う

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

```
コンテナイメージを ./swu/my_project.tar に保存しました。
./swu/app.desc のバージョンを 1 から 2 に変更しました。
./development.swu を作成しました。
次は Armadillo に ./development.swu をインストールしてください。
 *  Terminal will be reused by tasks, press any key to close it.
```

図 7.29 コンテナイメージの作成完了

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

7.1.4.4. アプリケーション実行用コンテナイメージのインストール

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

「図 7.30. ABOSDE で Armadillo に SWU をインストール」 のように、目的の Armadillo の隣にある赤枠で囲まれているボタンをクリックしてください。パスワードの入力を要求されますので、ABOS Web のパスワードを入力してください。その後、 ~/my_project/development.swu を選択してインストールを開始します。

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

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

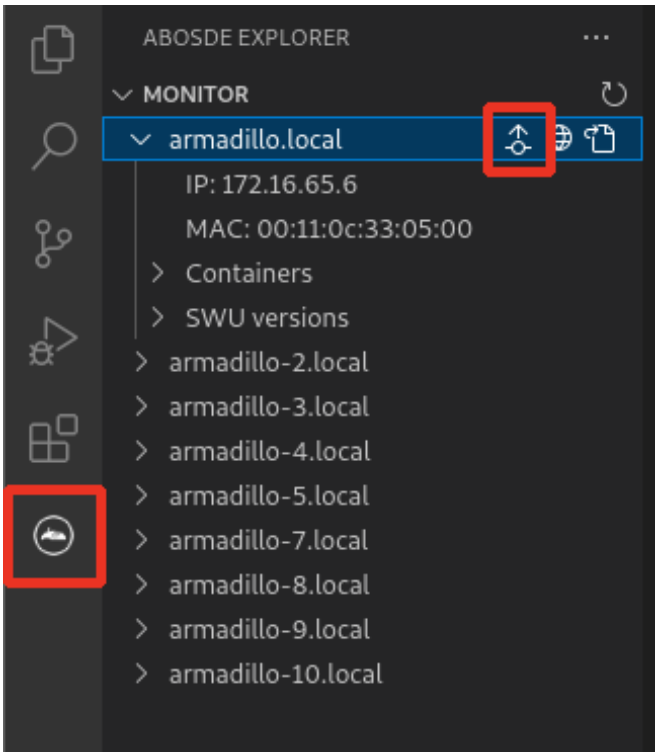


図 7.30 ABOSDE で Armadillo に SWU をインストール

7.1.4.5. ssh 接続に使用する IP アドレスの設定

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

「図 7.14. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする」 の赤枠で囲われているボタン、または 「図 7.15. ABOSDE に表示されている Armadillo を更新する」 の赤枠で囲われているマークをクリックして、ローカルネットワーク上で ABOS Web が実行されている Armadillo をスキャンしてください。

その後、目的の Armadillo について、「図 7.31. ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する」 の赤枠で囲われているマークをクリックしてください。

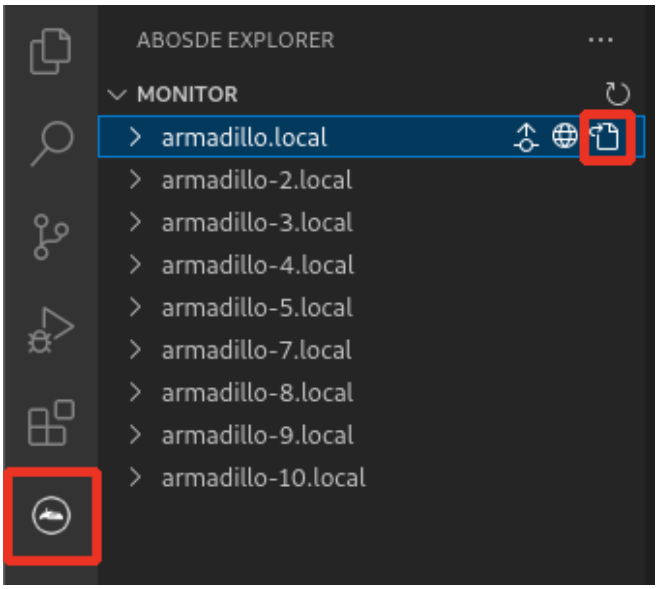


図 7.31 ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する

これにより、指定した Armadillo の IP アドレスを ssh 接続に使用する IP アドレスに設定することができます。また、プロジェクトディレクトリ内の `config/ssh_config` ファイルに指定した Armadillo の IP アドレスが記載されます。ATDE のネットワークを NAT に設定している場合や、ABOS Web を起動していない場合等、ABOSDE のリストに Armadillo が表示されない場合は、プロジェクトディレクトリに入っている `config/ssh_config` ファイルを編集して IP アドレスを書き換えてください。

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

図 7.32 ssh_config を編集する

❶ Armadillo の IP アドレスに置き換えてください。

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

7.1.4.6. アプリケーションの実行

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

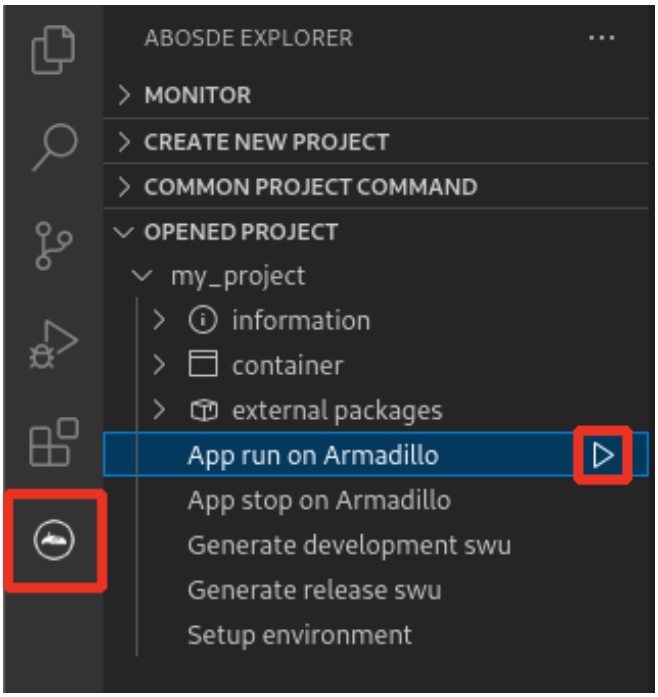


図 7.33 Armadillo 上でアプリケーションを実行する

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

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

図 7.34 実行時に表示されるメッセージ

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

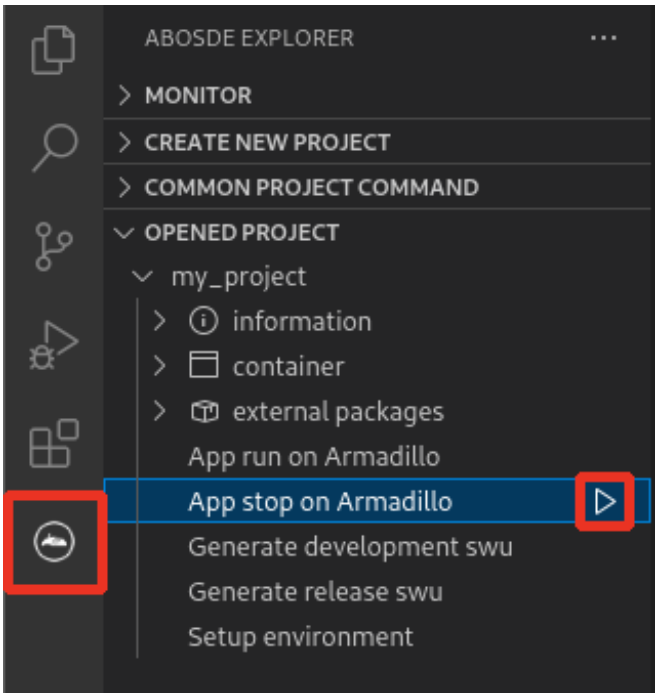


図 7.35 アプリケーションを終了する

7.1.4.7. アプリケーションの削除

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

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

この SWU イメージを Armadillo へインストールします。

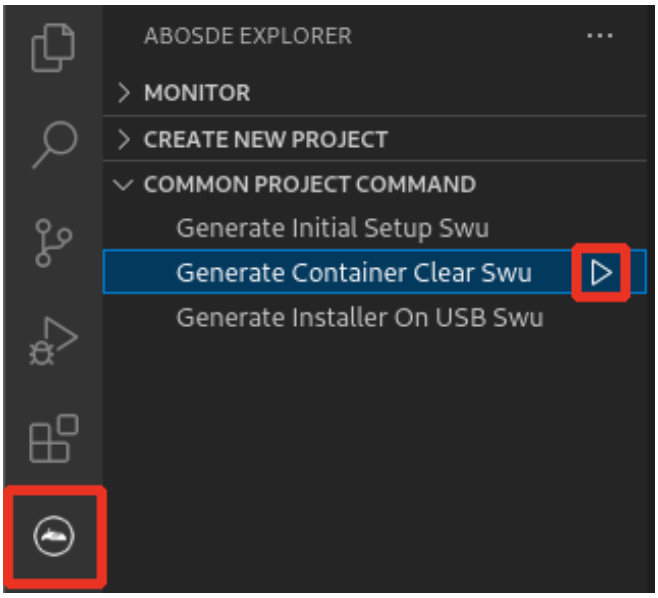


図 7.36 Armadillo 上のコンテナイメージを削除する

「図 7.30. ABOSDE で Armadillo に SWU をインストール」 のように、目的の Armadillo の隣にある赤枠で囲まれているボタンをクリックしてください。パスワードの入力を要求されますので、ABOS Web のパスワードを入力してください。その後、 `~/mkswu/container_clear.swu` を選択してインストールを開始します。

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

以上で開発環境のセットアップと動作確認の手順は終了です。

7.2. アプリケーション開発の流れ

基本的な Armadillo-900 開発セット でのアプリケーション開発の流れを「図 7.37. アプリケーション開発の流れ」に示します。

本章では、「図 7.37. アプリケーション開発の流れ」に示す開発時の流れに沿って手順を紹介していきます。

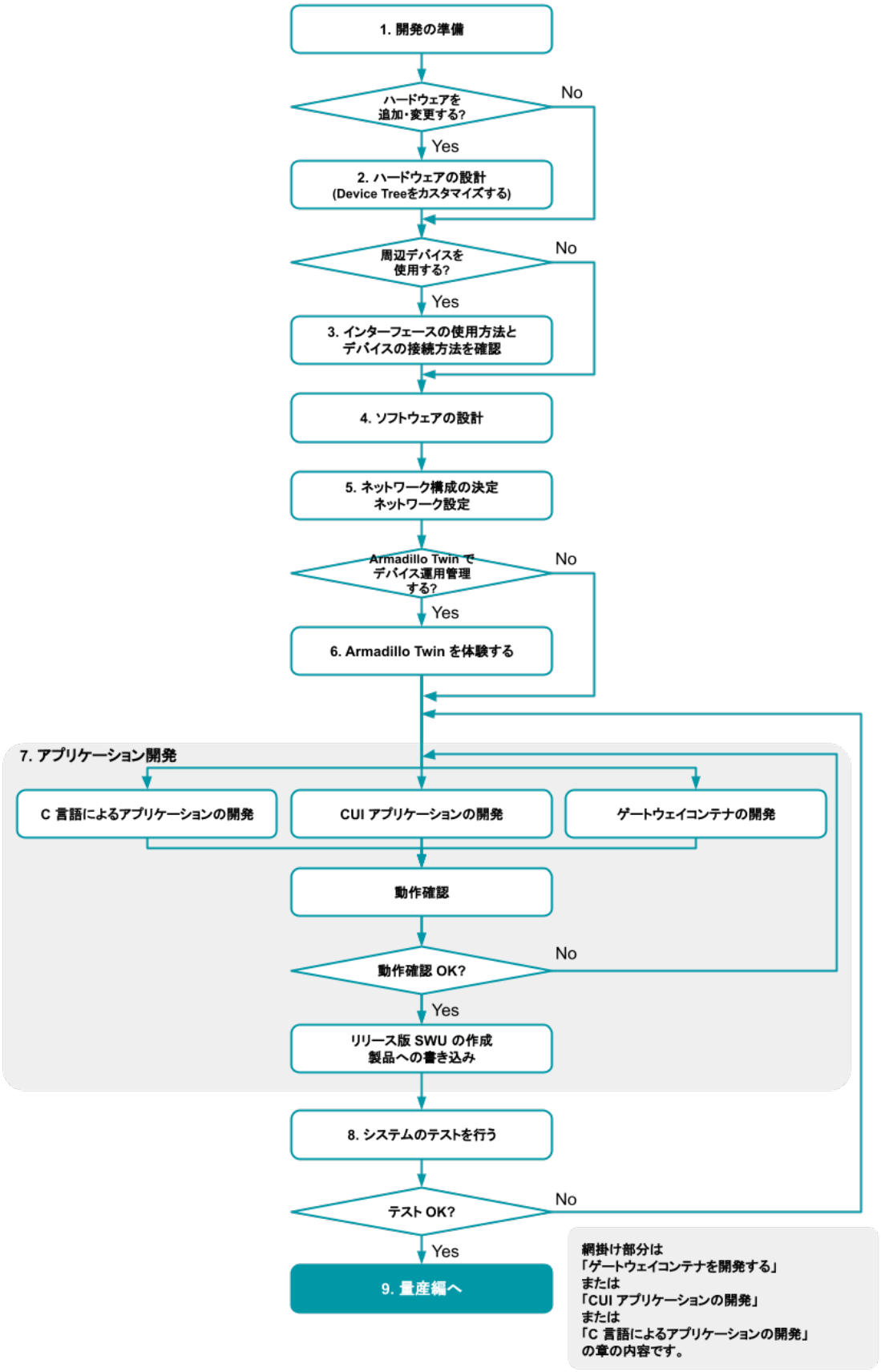


図 7.37 アプリケーション開発の流れ

1. 「7.1. 開発の準備」に従って開発環境の準備を行います。

   a.

2. Armadillo-900 開発セット に周辺デバイスを接続して使用する場合は、使用手順を「5.5. 各インターフェースの使用方法」で確認します。

3. 「7.4. ソフトウェアの設計」を行います。

4. 「5.1. ABOS Web を用いたネットワーク設定方法」を行います。

5. Armadillo Twin を使用したデバイス運用管理を検討する場合、「5.7.1. Armadillo Twin を体験する」を行います。

6. アプリケーションの開発を行います。「図 7.37. アプリケーション開発の流れ」 の網掛け部分です。

   a. CUI アプリケーションを開発する場合は、シェスクリプトまたは Python で開発することを推奨します。その場合は「7.9. CUI アプリケーションの開発」を行います。

   b. C 言語で開発された既存のアプリケーションを Armadillo 上で動作させる必要がある、あるいは開発環境の制約によって C 言語でのアプリケーション開発が必要な場合、「7.10. C 言語によるアプリケーションの開発」を行います。

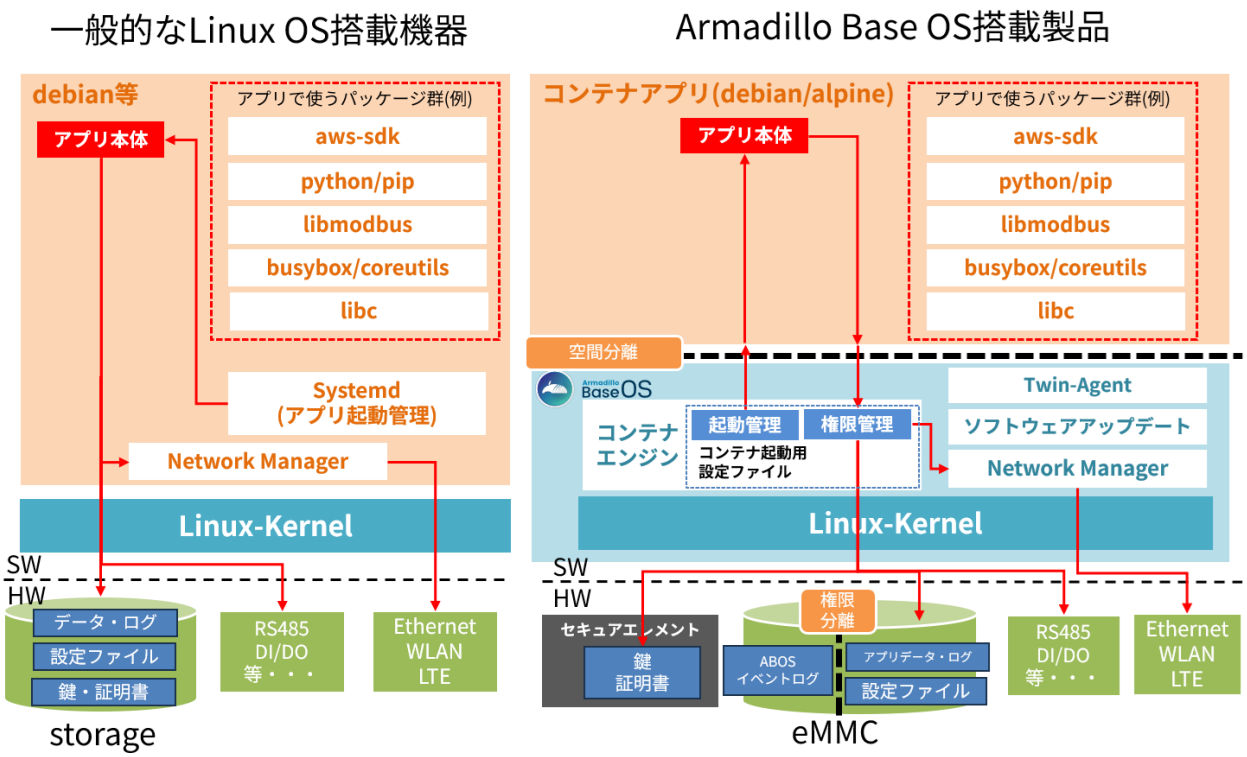
7. 開発したアプリケーションの動作確認が完了しましたら、「7.13. システムのテストを行う」を行います。

8. システムのテストが完了しましたら、「8. 量産編」へ進みます。

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

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

7.3.1. 一般的な Linux OS 搭載組み込み機器との違い

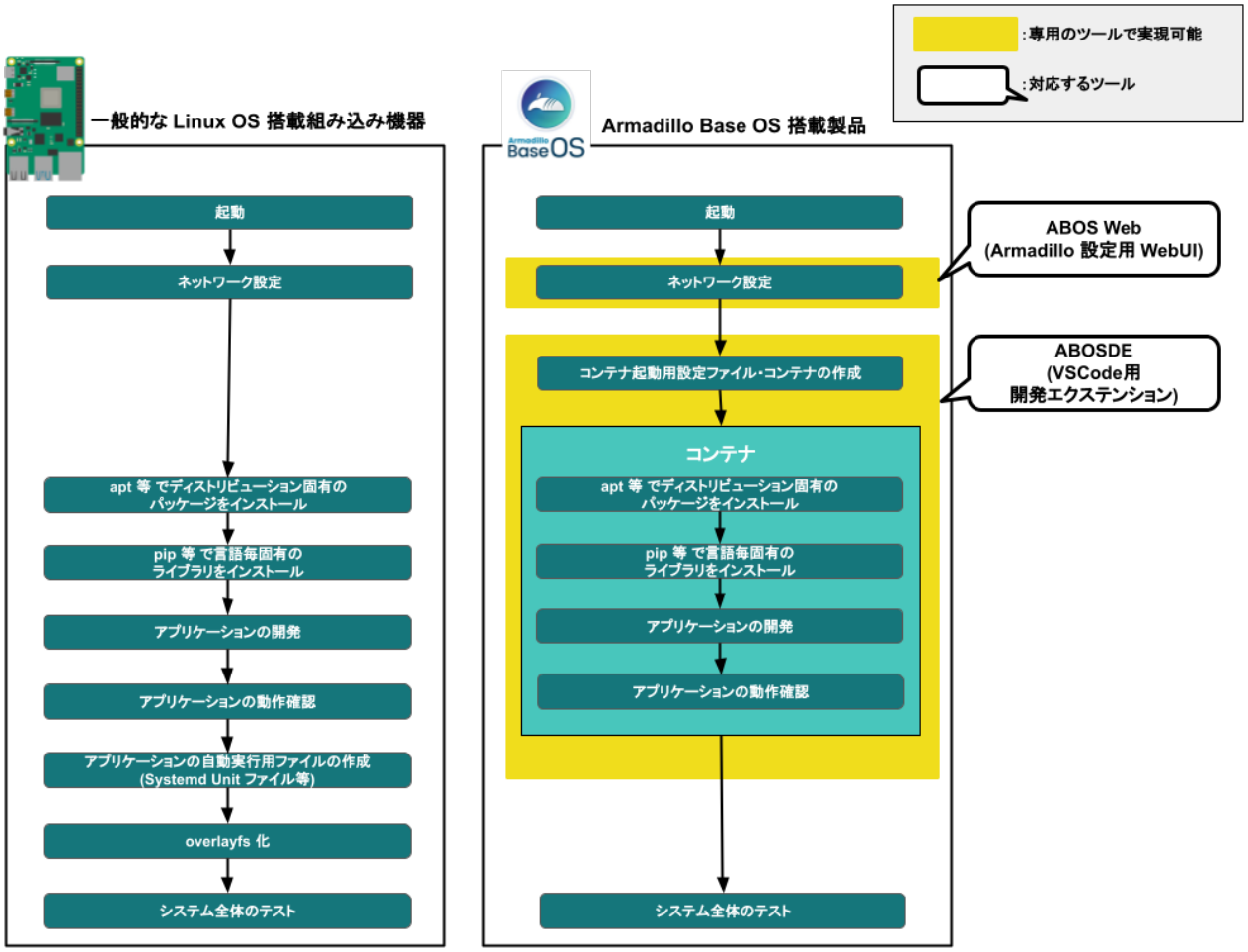


Linux OS 搭載組み込み機器ではアプリケーションの実行環境をユーザーランド上に直接用意し、Systemd などでアプリケーションを自動実行させるのが一般的です。Armadillo Base OS 搭載機器では、アプリケーションの実行環境をコンテナ内に用意して、コンテナ起動用設定ファイルを所定の場所に配置することでコンテナ(=アプリケーション)を自動実行させます。

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

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

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



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

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

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

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

7.3.3. アップデート機能について

Armadillo-900 開発セット では、開発・製造・運用時にソフトウェアを書き込む際に、 SWUpdate という仕組みを利用します。

7.3.3.1. SWUpdate とは

SWUpdate は、デバイス上で実行されるプログラムで、ネットワーク/ストレージ経由でデバイスのソフトウェアを更新することができます。Stefano Babic, DENX software engineering, Germany によってオープンソースで開発が進められています。

Armadillo-900 開発セット では、 SWUpdate を利用することで次の機能を実現しています。

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

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

なお、hawkBit 0.4.1 の配布は継続していますので、こちらを利用する場合は Armadillo-900 開発セット 開発用ツール [https://armadillo.atmark-techno.com/resources/software/armadillo-900/tools] から「Hawkbit docker-compose コンテナ」 をダウンロードして展開してください。

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

7.3.3.2. SWU イメージとは

swu パッケージは、SWUpdate 独自のソフトウェアの配布フォーマットです。SWUpdate では、1 回のアップデートは 1 つの swu パッケージで行われます。

swu パッケージには、次のような様々なものを含めることができます。

- アップデート対象のイメージファイル
- アップデート対象のイメージファイルのチェックサム
- アップデート前後に実行するスクリプト
- 書き込み先ストレージの情報
- U-Boot 環境変数の書き換え情報
- ソフトウェアのバージョン情報
- etc…

SWU イメージは swupdate (https://sbabic.github.io/swupdate/swupdate.html) によって Armadillo Base OS 上で検証とインストールが実行されます。SWU イメージを Armadillo に転送するための方法は、用途や状況に合わせて様々な方法を用意しています。例えば、 USB メモリから読み取る、ウェブサーバーからダウンロードするなどです。

7.3.3.3. 機密性、完全性、真正性の担保

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

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

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

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

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

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

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

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

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

7.3.3.4. A/B アップデート(アップデートの二面化)

A/B アップデートは、Flash メモリにパーティションを二面確保し、アップデート時には交互に利用する仕組みです。

常に使用していない方のパーティションを書き換えるため次の特徴を持ちます。

- ○ アップデートによって動作中のソフトウェアは破壊されない
- ○ 書き込みが電源断などで中断しても、すぐに復帰出来る
- ○ 機器が動作中に書き込みが出来る
- × 使用 Flash メモリ量が増える

7.3.3.5. ロールバック(リカバリー)

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

ロールバック状態の確認は 「10.14. ロールバック状態を確認する」 を参照してください。

自動ロールバックが動作する条件は以下の通りです：

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

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

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

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

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

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

7.3.3.6. SWU イメージのインストール

イメージをインストールする方法として以下に示すような方法があります。

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

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

それぞれのインストール方法の詳細については、以下に記載しております。もし、作成した SWU イメージのインストールに失敗する場合は、「10.2.5. swupdate がエラーする場合の対処」をご覧ください。

- ABOS Web を使用した手動インストール

  Armadillo-900 開発セット で動作している Web アプリケーションの ABOS Web を使用してアップデートすることができます。「10.9.4. SWU インストール」を参考にしてください。

- ABOSDE から ABOS Web を使用した手動インストール

  VS Code 拡張機能の ABOSDE を使用することで、Armadillo-900 開発セット で動作している ABOS Web 経由でアップデートすることができます。「10.10.5. Armadillo に SWU をインストールする」を参考にしてください。

- USB メモリまたは microSD カードからの自動インストール

  Armadillo-900 開発セットに 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-900 開発セットは自動で再起動します。

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

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

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

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

❶ USB メモリがマウントされている場所を確認します。

❷ ファイルをコピーします。

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

エラーの場合、/var/log/message に保存されます。例えば、コンソールで証明書が間違っているイメージのエラーは以下の様に表示されます。

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

❶ 証明書エラーのメッセージ。

・外部記憶装置からイメージのインストール（手動）

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

以下は外部記憶装置が/dev/mmcblk2p1(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/mmcblk2p1 /mnt
[armadillo ~]# swupdate -i /mnt/swu/initial_setup.swu
SWUpdate v5f2d8be-dirty

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

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

・Armadillo Twin を使用した自動インストール

Armadillo Twin で Armadillo-900 開発セット を複数台管理してアップデートすることができます。「5.7.4. Armadillo Twin から複数の Armadillo をアップデートする」を参考にしてください。

・ウェブサーバーからイメージのインストール（手動）

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

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

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

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

・ウェブサーバーからの定期的な自動インストール

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

```
[armadillo ~]# rc-update add swupdate-url ❶
[armadillo ~]# persist_file /etc/runlevels/default/swupdate-url ❷
[armadillo ~]#
    echo https://download.atmark-techno.com/armadillo-900/image/baseos-900-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。

7.3.4. ファイルの取り扱いについて

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

その為、ファイルを変更した後 Armadillo の電源を切ると変更内容は保持されません。開発中などに rootfs の変更内容を保持するには、変更したファイルに対して persist_file コマンドを使用します。

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

図 7.38 persist_file コマンド実行例

persist_file コマンドの詳細については、「4.3. persist_file について」を参照してください。

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

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

「7.3.4. ファイルの取り扱いについて」 にて、 Armadillo Base OS 上のファイルは通常、 persist_file コマンドを実行せずに電源を切ると変更内容が保存されないと紹介しましたが、「表 7.2. 電源を切っても保持されるディレクトリ(ユーザーデータディレクトリ)」に示すディレクトリ内にあるファイルはこの限りでありません。

表 7.2 電源を切っても保持されるディレクトリ(ユーザーデータディレクトリ)

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

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

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

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

Copy-on-Write (CoW) について。

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

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

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

図 7.39 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。

7.3.5. インストールディスクについて

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

インストールディスクには以下の 2 つの種類があります。

- 初期化インストールディスク

  Armadillo-900 開発セット インストールディスクイメージ [https://armadillo.atmark-techno.com/resources/software/armadillo-900/disc-image] にある標準イメージです。Armadillo を初期化する際に使用します。

- 開発が完了した Armadillo-900 開発セット をクローンするためのインストールディスク。

  量産時など、特定の Armadillo を複製する際に使用されます。詳しくは、「8. 量産編」で説明します。

7.3.5.1. インストールディスクの作成

インストールディスクの作成方法は「3.2.1. 初期化インストールディスクの作成」を参照してください。

参照先では初期化インストールディスクの場合の手順を示していますが、「10.19. Armadillo のソフトウェアをビルドする」 でビルドしたイメージについても同じ手順になります。その際のインストールディスクイメージ（.img）は、以下のコマンドを実行して作成してください。

```
[ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_image.sh --board a900
:（省略）
[ATDE ~/build-rootfs-[VERSION]]$ ls baseos-900*img
baseos-900-[VERSION].img
[ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_image.sh --board a900 ¥
        --boot ~/uboot-[VERSION]/u-boot-dtb.imx ¥
        --installer ./baseos-900-[VERSION].img
```

コマンドの実行が完了すると、baseos-900-[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。

7.3.5.2. インストールディスクを使用する

インストールディスクを使用する方法については、「3.2.2. インストールディスクを使用する」を参照してください。

7.4. ソフトウェアの設計

Armadillo-900 開発セット を用いた製品のソフトウェア設計は、一般的な組み込み開発と基本的には変わりません。しかし、 Armadillo Base OS という独自 OS を搭載しているため、ソフトウェアの設計には特有のポイントがいくつかあります。本章では、それらの設計時に考慮すべき Armadillo Base OS 特有のポイントについて紹介していきます。

7.4.1. 開発者が開発するもの、開発しなくていいもの

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

「図 7.40. 開発者が開発するもの、開発しなくていいもの」は、 Armadillo Base OS 搭載製品において、開発者が開発するものと開発しなくていいものをまとめた図です。

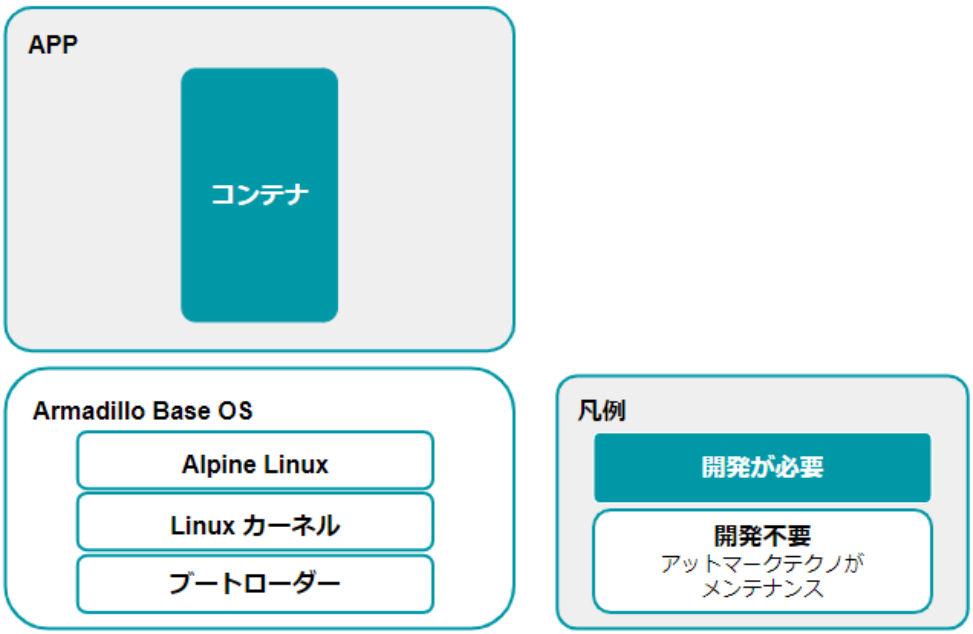


図 7.40 開発者が開発するもの、開発しなくていいもの

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

拡張ボードを追加するためにデバイスツリーのカスタマイズが必要となる場合は、デバイスツリー(dtbo)の追加が必要となります。

使用するデバイスによっては、Linux カーネルドライバの追加が必要となり、Linux カーネルのカスタマイズが必要となります。

7.4.2. ユーザーアプリケーションの設計

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

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

アットマークテクノでは、アットマークテクノが提供する Debian GNU/Linux ベースのコンテナイメージ [https://armadillo.atmark-techno.com/resources/software/armadillo-900/debian-container]を提供しておりますが、それ以外の link:Docker Hub [https://hub.docker.com/search?type=image&image_filter=official] などから使い慣れたディストリビューションのコンテナイメージを取得して開発することができます。

7.4.2.1. ユーザーデータの保存場所

アプリケーションが出力するユーザーデータで保存が必要なものは、「7.3.4.1. 電源を切っても保持されるディレクトリ(ユーザーデータディレクトリ)」にも記載があるとおり、 /var/app/volumes/ 以下に配置してください。

色々な場所にデータが保存されていますと Armadillo-900 開発セット の初期化を行う際に削除の処理が煩雑になりますので、 /var/app/volumes/ 以下に集約してください。

7.4.2.2. アプリケーション設定情報の保存場所

開発したアプリケーションやコンテナがバージョンアップした際にも必要となる設定情報は、 「7.3.4.1. 電源を切っても保持されるディレクトリ(ユーザーデータディレクトリ)」 にも記載があるとおり、 /var/app/rollback/volumes/ 以下に保存してください。

7.4.2.3. LTE 通信を使用する場合に考慮すべきこと

LTE 通信は、周辺の状況や工事などによって長時間通信ができなくなる可能性があります。そのため、クラウドやサーバーへ送信すべきデーターを即時に送信できない可能性があります。

データーの再送処理や動作しているコンテナ内にキャッシュする処理を実装して、上記状況に備えてください。

7.4.3. ログの設計

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

7.4.3.1. ログの保存場所

ユーザーアプリケーションが出力するログは、「7.3.4.1. 電源を切っても保持されるディレクトリ(ユーザーデータディレクトリ)」にも記載があるとおり、 /var/app/volumes/ 以下に配置するのが良いです。

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

7.4.3.2. 保存すべきログ

- Ethernet、LTE、Bluetooth、WLAN などの無線系のログ

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

・ソフトウェアのバージョン

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

・A/B 面どちらであったか

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

「図 7.41. 現在の面の確認方法」に示すコマンドを実行することで、現在 A/B どちらの面で起動しているかを確認できます。

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

図 7.41 現在の面の確認方法

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

7.4.4. ウォッチドッグタイマー

Armadillo-900 開発セットのウォッチドッグタイマーは、i.MX 8ULP の WDOG(Watchdog Timer)を利用しています。

ウォッチドッグタイマーは、U-Boot によって有効化されます。標準状態でタイムアウト時間は 10 秒に設定されます。

何らかの要因でウォッチドッグタイマーのキックができなくなりタイムアウトすると、システムリセットが発生します。

ウォッチドッグタイマーの設定変更は、ioctl システムコール経由で行うことができます。詳細な情報については、Linux カーネルのソースコードに含まれているドキュメント(Documentation/watchdog/watchdog-api.rst)を参照してください。

ウォッチドッグタイマーを停止することはできません。

7.4.5. コンテナに Armadillo の情報を渡す方法

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

コンテナ起動設定ファイル（conf ファイル）に関しては「10.8.3. コンテナ起動設定ファイルを作成する」を参照してください。

・アットマークテクノが提供する情報を環境変数として渡す

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

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

add_armadillo_env については「10.8.3.6. 個体識別情報の環境変数の追加」を参照してください。

・任意の情報を環境変数として渡す

コンテナ起動設定ファイルに add_args を使用してください。

add_args については「10.8.3.20. podman run に引数を渡す設定」を参照してください。

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

```
add_args --env=<環境変数名>=<値> ❶
```

図 7.42 add_args を用いてコンテナに情報を渡すための書き方

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

add_args --env の例を示します。

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

図 7.43 add_args を用いてコンテナに情報を渡す例

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

7.5. ABOS Web をカスタマイズする

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

・ロゴ画像

・ヘッダロゴアイコン画像

・ヘッダタイトル

・favicon 画像

・背景色

・メニューの表示名

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

カスタマイズは、「設定管理」で行うことができます。

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

図 7.44 ABOS Web のカスタマイズ設定

- **ロゴ画像**

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

- **ヘッダロゴアイコン画像**

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

- **ヘッダタイトル**

  画面左上に常に表示されている文字列です。24 文字まで入力できます。

- **favicon 画像**

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

表 7.3 用意する favicon 画像

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

- **背景色**

  5 種類の中から選択できます。

- **メニューの表示名**

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

図 7.45 メニュー変更画面 (一部)

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

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

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

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

7.6. RTOS ファームウェアの開発

7.6.1. m33-firmware-at とは

Armadillo-900 シリーズには、Linux を実行する 2 つの Arm Cortex-A35 コアに加えて、1 つの Arm Cortex-M33 コアが搭載されています。

一部の I/O インタフェースは A35 コアからアクセスできないため、M33 コア上に独自の RTOS ファームウェア「m33-firmware-at」を起動し、「図 7.46. Armadillo-900 で使用している i.MX 8ULP の簡易構造」の様にコア間通信を介して Linux から操作できるようにしています。

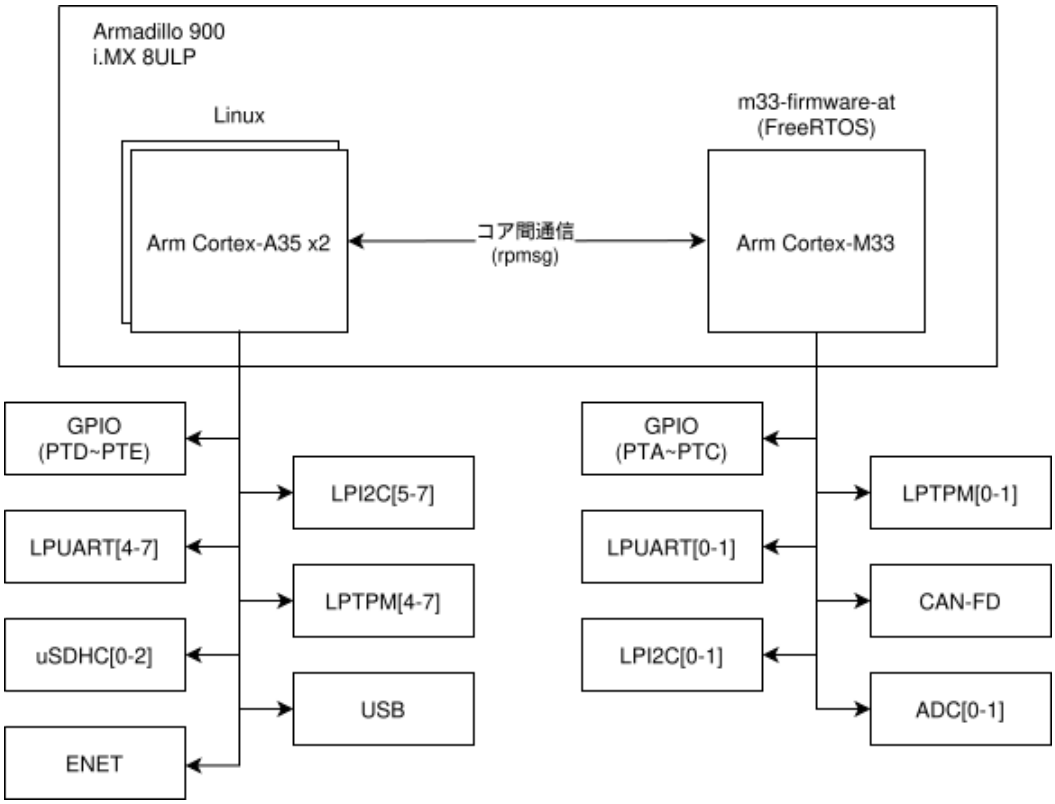


図 7.46 Armadillo-900 で使用している i.MX 8ULP の簡易構造

m33-firmware-at は主に以下の 3 つの役割を担います。

- Linux から M33 に接続されている I/O を操作可能にする
- サスペンド時に消費電力を最適化する
- 必要に応じて任意のコードを RTOS 環境で実行する

本章では、上記の 3 つ目の任意のコードを m33-firmware-at に追加する方法について解説します。

7.6.2. 開発環境の説明

ブートローダーイメージのソースコードをダウンロード、ビルド、インストールすることで、m33-firmware-at を改造できます。詳細なビルド手順は 「10.19.1. ブートローダーをビルドする」 を参照してください。

7.6.2.1. 開発を始める前の注意

ブートローダーの開発前に以下の注意点をご確認ください。

1. ブートローダーを改造すると、 Armadillo が起動しなくなる可能性があります。ハードウェアの故障でなければ、インストールディスクから復旧可能です。予め 「3.2.1. 初期化インストールディスクの作成」 通りにインストールディスクを作成しておくか、SD カードで開発を行うことを推奨します。

2. Linux と m33-firmware-at はコア間通信で連携するため、長期間更新されていない状態から片方だけを更新すると正常に動作しない可能性があります。原則として、常に適合性を維持するよ

うに更新を提供しますが、m33-firmware-at のコア部分を更新できるように開発することを推奨します。

m33-firmware-at のソースコードを github [https://github.com/atmark-techno/m33-firmware-at] に公開していますので、ブートローダーをダウンロードした後 m33-firmware-at を git リポジトリと置き換えると管理しやすくなります。

```
[ATDE ~]$ tar xf imx-boot-[VERSION].tar.zst
[ATDE ~]$ cd imx-boot-[VERSION]
[ATDE ~/imx-boot-[VERSION]]$ rm -rf m33_firmware_at
[ATDE ~/imx-boot-[VERSION]]$ git clone https://github.com/atmark-techno/m33-firmware-at ⏎
m33_firmware_at
```

3. ATDE のストレージ容量を抑えるため、デフォルトでは gcc-arm-none-eabi パッケージがインストールされていません。インストールされていない場合は、ソースコードアーカイブに含まれているバイナリが利用されるため、 m33-firmware-at のビルドは行われません。開発前に m33-firmware-at ビルド状態の確認 の手順でビルドできることを確認してください。

7.6.2.2. ソフトウェア構成

m33-firmware-at のファイル構成は以下の通りです。

1. Linux ドライバーに必要なサービス

   Linux ドライバーとの連携は srtm サービスで行います。 app_srtm.c で初期化処理を行い、各ドライバーの対応は以下のファイルで行います。

   - srtm/services/srtm_*_service.[ch]: Linux ドライバーからのプロトコル実装
   - app_*.c: ハードウェア操作の実装

2. 電源と電圧管理

   起動時の電圧管理やサスペンド時の消費電力の最適化は main.c と lpm.c で行います。

3. カスタマイズ用ディレクトリ

   custom ディレクトリに任意のコードを実装することで、 m33-firmware-at を容易に変更できます。以下のサンプルファイルを提供しています。

   - custom/cli_custom.c: デバッグシリアル接続のコマンド追加
   - custom/app_tty_custom.c: Linux の仮想 tty ドライバー

     arch/arm64/boot/dts/freescale/armadillo_900-customize.dts ファイル等に以下の内容を設定すると、仮想 tty デバイスが生成されます：

```
&{/} {
        aliases {
                ttyrpmsg2 = "/m33-custom-uart";
        };
        m33-custom-uart {
                compatible = "fsl,imx-rpmsg-tty-serial";
                port_type = <TTY_TYPE_CUSTOM>;
                port_name = "custom port"; /* 32 文字までの任意文字列 */
```

```
        };
};
```

custom_init() は tty が初期化される時に実行されます。複数のデバイスを生成する場合は port_name で区別できます。

tx/rx は Linux 側から見た名称です。 Linux 側の tty に書き込むと custom_tx() 関数が実行されます。SRTM_TtyService_NotifyAlloc() と SRTM_TtyService_NotifySend() を利用すると Linux にメッセージが送信できます。

4. ビルドシステム

   ビルドシステムは armgcc ディレクトリにあります。

   ファイルを追加する場合は armgcc/CMakeLists.txt の add_executable 命令に追加してください。

   NXP 社の mcux-sdk のドライバを追加したい場合は armgcc/config.cmake を変更してください。

7.6.3. 不具合解析

7.6.3.1. ログによる解析

m33-firmware-at のソースコードで PRINTF を使用して簡易的な解析ができます。

ログの取得については 「5.5.14. RTD 用コンソール を使用する」 を参照してください。

シリアル接続後、簡易コンソールで以下のコマンドを利用できます。

- help: 利用可能なコマンド一覧
- log: ログバッファーの内容を再表示
- version: m33-firmware-at のバージョンを表示（git からビルドした場合は git describe によるタグかコミット情報が記載されます)
- md, mw: u-boot に似た memory display/write コマンド。こちらのコマンドはデフォルト状態で利用できません。

  main.h で「#define CLI_RAW_MEM 」をコメントアウトしてイメージを更新するとコマンドが追加されます。

  -b, -w, -l, -q オプションで byte (1 byte), halfword (2 bytes), word (4 bytes), double word (8 bytes) を扱えます。

  md にアドレスとアイテム数を指定して、メモリ内容を表示します。mw にアドレスと値を指定します。

サスペンド関連の解析には、 main.h の DEBUG_SUSPEND (サスペンド中でもシリアル出力を無効化しない）と DEBUG_SUSPEND_SKIP_JTAG_PINS (サスペンド中に jtag のピンを無効化しない）設定も利用できます。

7.6.3.2. JTAG での解析

JTAG デバッガーを利用した解析ができます。

弊社では PALMiCE4 [https://www.computex.co.jp/products/palmice4/index.htm] を利用して、以下の機能を確認しています。

・メモリ、レジスターの取得と書き込み

・ステップ実行・Break（ASM コード、C ソース両方）

・ローカルまたはグローバル変数の値の取得と変更

・FreeRTOS のデバッグ機能（タスクやタイマー情報）

以下では Armadillo 固有の情報のみを記載します。ここに記載の無い内容については、コンピューテックス社にお問い合わせください。

1. C ソースの連携や FreeRTOS のデバッグ機能にはデバッグ情報が必要です。以下の手順で有効化できます。

```
[ATDE ~/imx-boot-[VERSION]]$ ./m33_firmware_at/armgcc/build.sh debug ❶
[ATDE ~/imx-boot-[VERSION]]$ ls m33_firmware_at/armgcc/debug/*
m33_firmware_at/armgcc/debug/m33-firmware-at.bin ❷
m33_firmware_at/armgcc/debug/m33-firmware-at.elf ❸
```

図 7.47 m33-firmware-at のデバッグビルド方法

❶ デバッグビルドの実行。一度実行すると、「 build.sh release 」で無効化するまでは imx-boot ディレクトリの「 make imx-boot_armadillo-900 」コマンドでデバッグビルドが利用されます。

❷ imx-boot に組み込まれるファームウェアのバイナリ

❸ デバッガーに提供する ELF ファイル（デバッグ情報あり）

2. 接続に関しては 「5.5.15. JTAG を使用する」 を参照してください。

7.7. ABOSDE によるアプリケーションの開発

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

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

ABOSDE では、以下のようなアプリケーションを開発できます。

・CUI アプリケーション

・C 言語アプリケーション

7.7.1. ABOSDE の対応言語

「表 7.4. ABOSDE の対応言語」に示すように、アプリケーション毎に対応している言語が異なります。

表 7.4 ABOSDE の対応言語

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

7.7.2. 参照する開発手順の章の選択

どのようなアプリケーションを開発するかによって ABOSDE による開発手順が異なります。「図 7.48. 参照する開発手順の章を選択する流れ」を参考に、ご自身が開発するアプリケーションに適した章を参照してください。

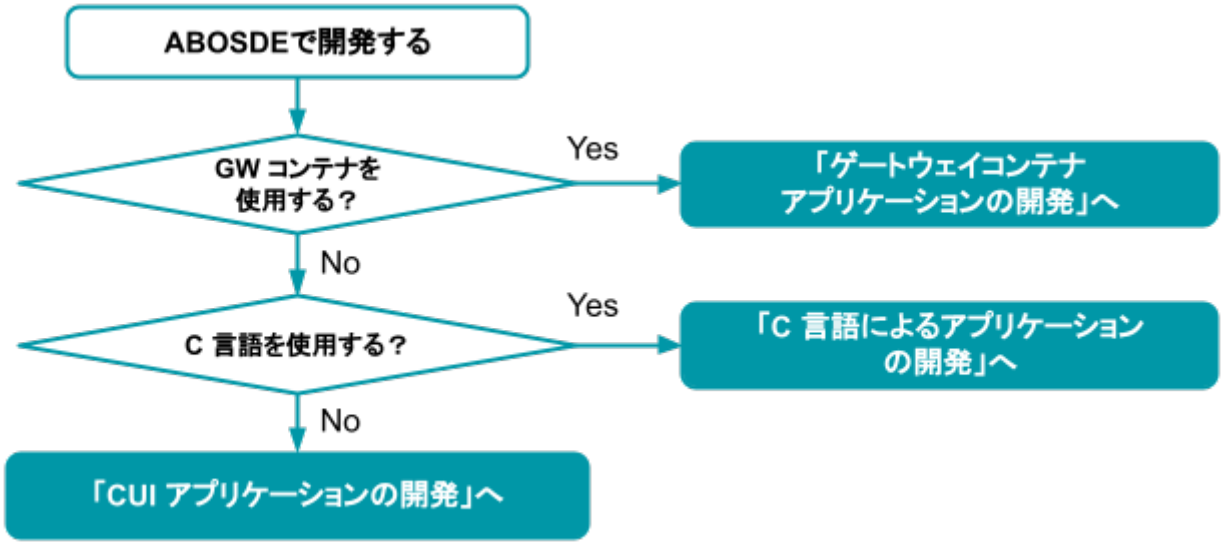


図 7.48 参照する開発手順の章を選択する流れ

CUI アプリケーション

- 対象ユーザー
  - 画面を使用しないアプリケーションを開発したい
- マニュアルの参照先
  - 「7.9. CUI アプリケーションの開発」を参照

C 言語アプリケーション

- 対象ユーザー
  - C 言語でないと実現できないアプリケーションを開発したい
  - 既存の C 言語によって開発されたアプリケーションを Armadillo で動作させたい
  - 開発環境に制約がある
- マニュアルの参照先
  - 「7.10. C 言語によるアプリケーションの開発」を参照

7.8. GUI アプリケーションの開発

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

i.MX 8ULP には、 H.264 などの動画用のハードウェアデコーダーは搭載されていないため、アプリケーション内で動画の再生を行うと期待通りのフレームレートでは再生されない可能性があります。

7.8.1. 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 7.49 Flutter アプリケーションの例

7.8.2. 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。

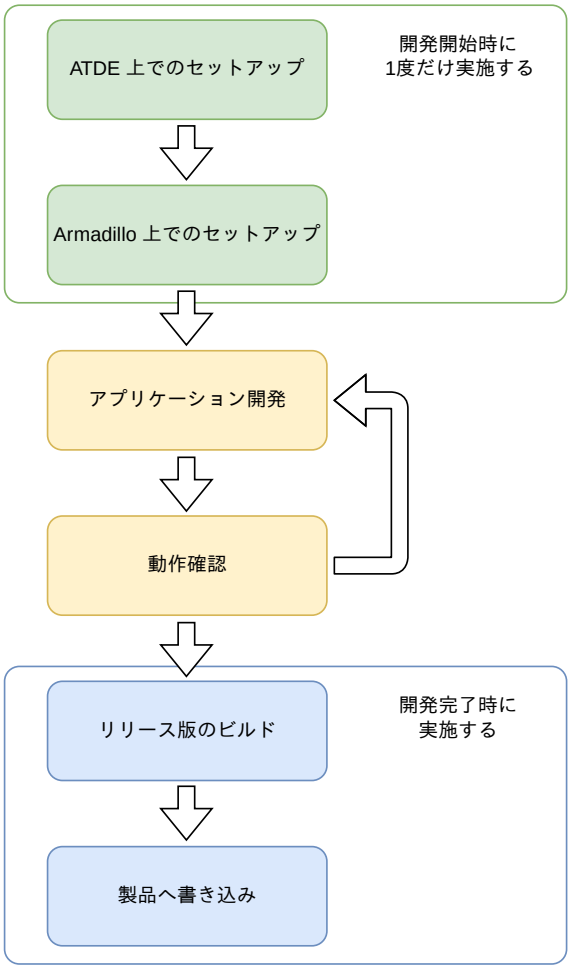


図 7.50 Flutter アプリケーション開発の流れ

7.8.3. 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 「7.1. 開発の準備」 を参照して ATDE のセットアップを完了してください。

7.8.3.1. プロジェクトの作成

Flutter アプリケーションのサンプルとして以下を用意しております。

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

各プロジェクトは以下のようなアプリケーションの画面となります。

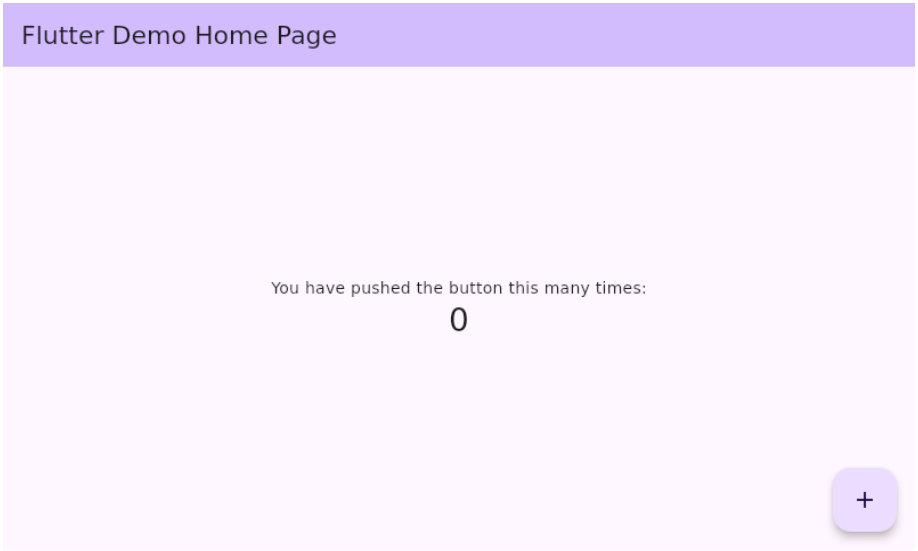


図 7.51 Flutter Demo アプリケーションの画面

図 7.52 GUI アプリケーションの画面

以降の手順でサンプルアプリケーション毎に VS Code 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。

VS Code の左ペインの [A900] から [<アプリケーション名> New Project]を実行し、表示されるディレクトリ選択画面からプロジェクトを保存するディレクトリを選択してください。実行するためには右に表示されている三角形ボタンを押してください。保存先を選択すると、プロジェクト名を入力するダイアログが表示されるので、任意のプロジェクト名を入力してエンターキーを押してください。この操作により、選択した保存先に、入力したプロジェクト名と同名のディレクトリが作成されます。

また、ここでは次のように設定しています。

・ **保存先**：ホームディレクトリ

- **プロジェクト名**：`my_project`

以下では例として [GUI アプリケーション] の作成を行っています。

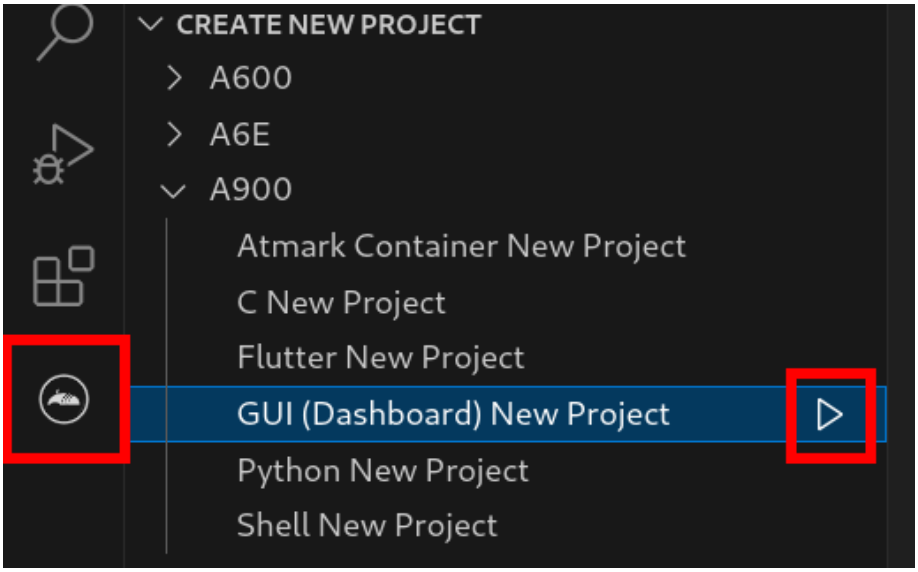


図 7.53 GUI アプリケーションのプロジェクトを作成する

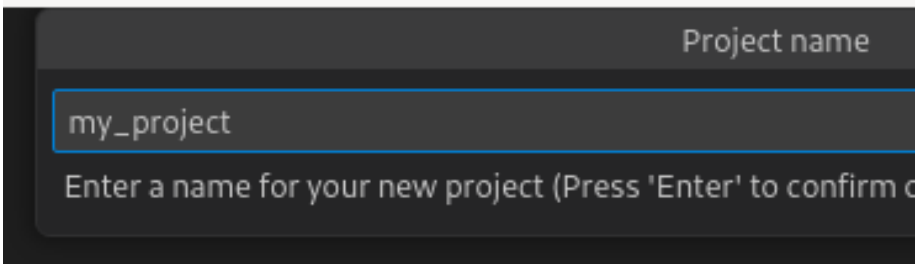


図 7.54 プロジェクト名を入力する

7.8.3.2. ディレクトリ構成

プロジェクトには下記のディレクトリがあります。

- **app**: Flutter アプリケーションです。Armadillo ではビルドしたアプリケーションが `/var/app/rollback/volumes/my_project` にコピーされます。
- **config**: 設定に関わるファイルが含まれるディレクトリです。
  - **app.conf**: コンテナのコンフィグです。記載内容については 「10.8.3. コンテナ起動設定ファイルを作成する」 を参照してください。
  - **app.desc**: SWU イメージを生成するための .desc ファイルです。記載内容については 「10.3. mkswu の .desc ファイルを編集する」 を参照してください。
  - **ssh_config**: Armadillo への ssh 接続に使用します。「7.9.7.2. ssh 接続に使用する IP アドレスの設定」 を参照してください。
- **container**: スクリプトを実行するコンテナの設定ファイルが含まれるディレクトリです。

- **packages.txt**：このファイルに記載されているパッケージがインストールされます。
- **Dockerfile**：直接編集することも可能です。

7.8.3.3. 初期設定

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

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

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

図 7.55 初期設定を行う

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

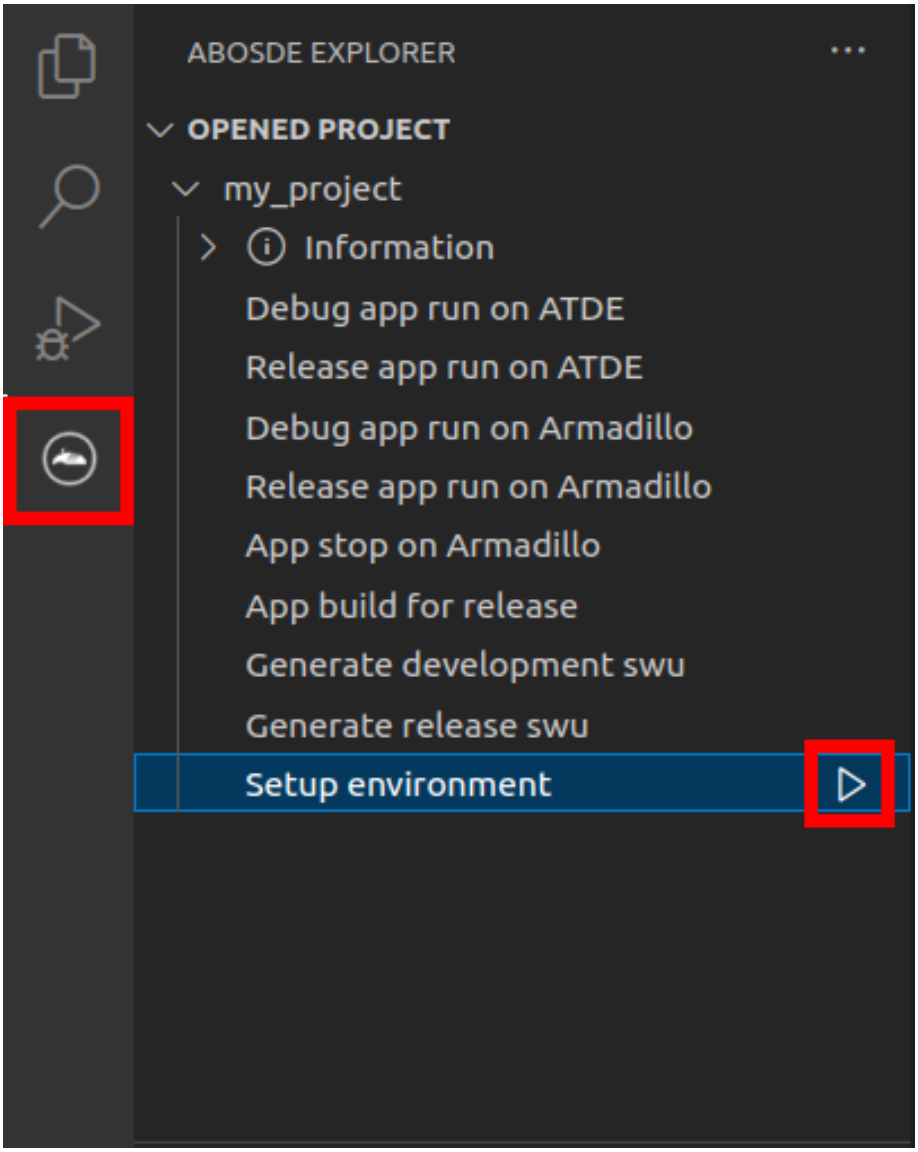


図 7.56 VS Code で初期設定を行う

選択すると、 VS Code の下部に以下のようなターミナルが表示されます。

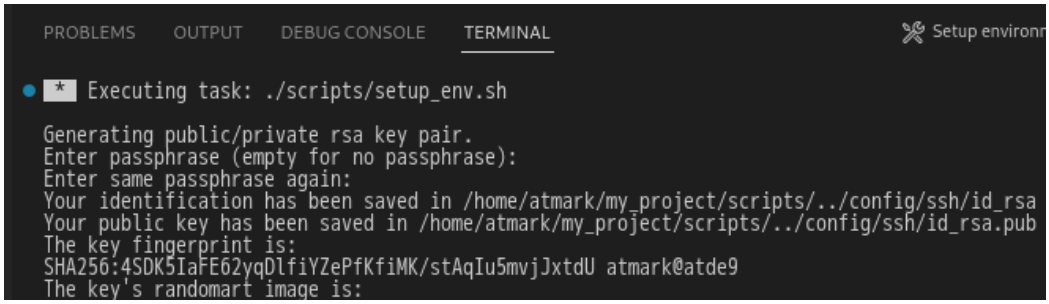


図 7.57 VS Code のターミナル

このターミナル上で以下のように入力してください。

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

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

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

図 7.58 SSH 用の鍵を生成する

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

❷ ここで何か任意のキーを押すとターミナルが閉じます。

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

7.8.3.4. アプリケーション実行用コンテナイメージの作成

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

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

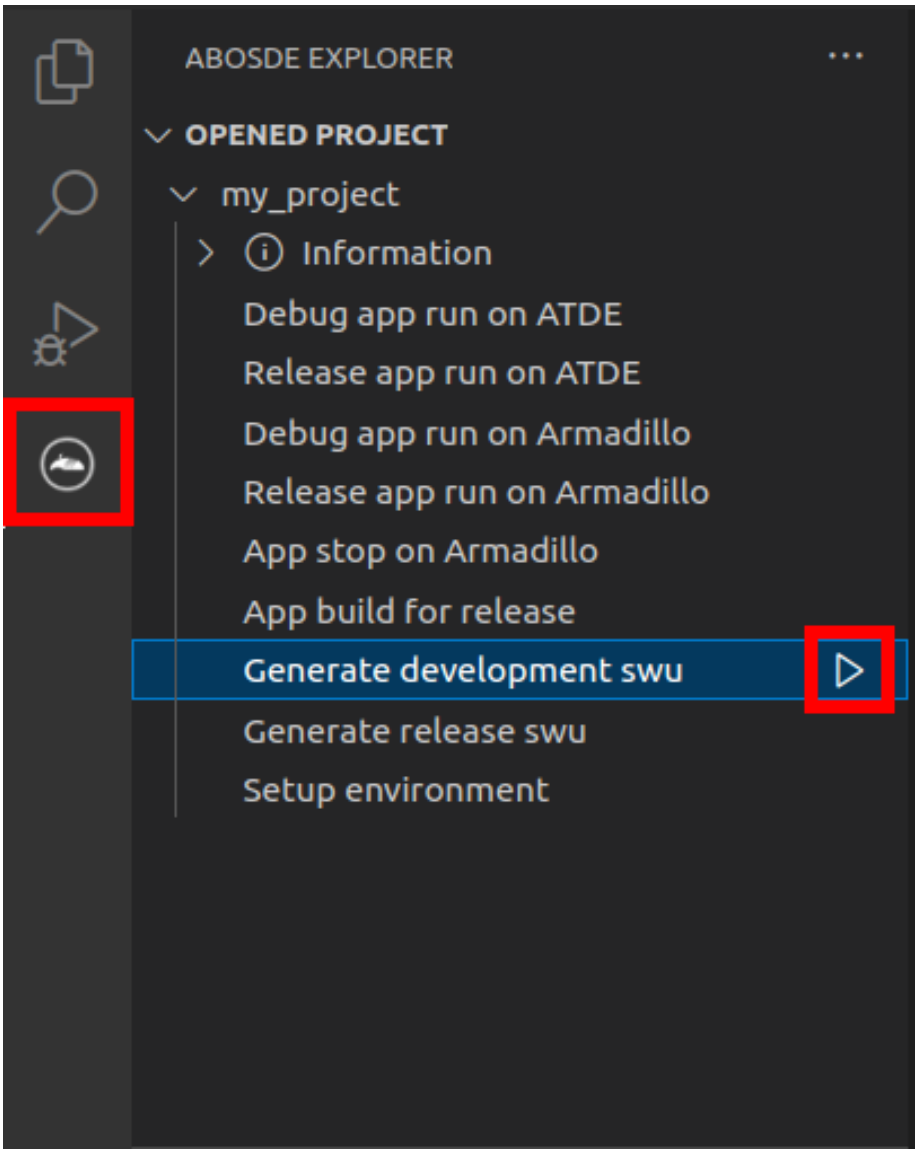


図 7.59 VS Code でコンテナイメージの作成を行う

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

```
コンテナイメージを ./swu/my_project.tar に保存しました。
./swu/app.desc のバージョンを 1 から 2 に変更しました。
./development.swu を作成しました。
次は Armadillo に ./development.swu をインストールしてください。
 *  Terminal will be reused by tasks, press any key to close it.
```

図 7.60 コンテナイメージの作成完了

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

7.8.4. コンテナのディストリビューション

使用するコンテナのディストリビューションは以下のとおりです。

ディストリビューション	・ debian:bullseye

7.8.5. Armadillo に転送するディレクトリ及びファイル

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

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

7.8.6. コンテナ内のファイル一覧表示

「図 7.61. コンテナ内のファイル一覧を表示するタブ」 の赤枠で囲われているタブをクリックすることで、`development.swu` または「7.8.12. リリース版のビルド」で作成される `release.swu` に含まれるコンテナ内のファイルおよびディレクトリを表示します。

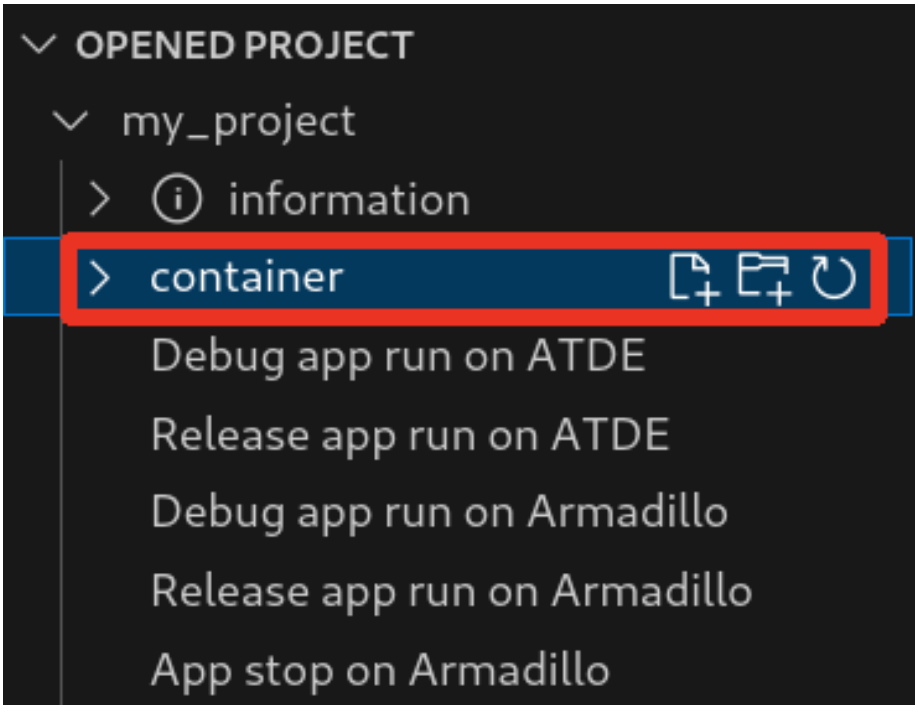


図 7.61 コンテナ内のファイル一覧を表示するタブ

クリック後の表示例を 「図 7.62. コンテナ内のファイル一覧の例」 に示します。

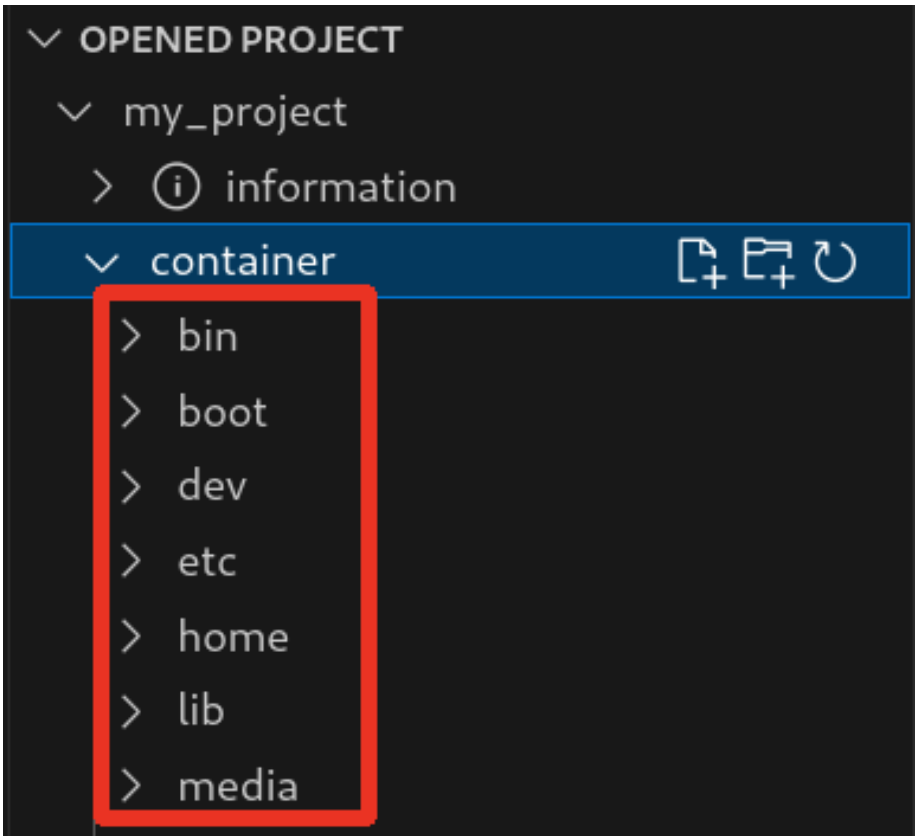


図 7.62 コンテナ内のファイル一覧の例

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

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

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

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

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

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

7.8.6.1. resources ディレクトリについて

「図 7.63. resources ディレクトリ」に示すように ATDE 上のプロジェクトディレクトリには container/resources ディレクトリがあります。

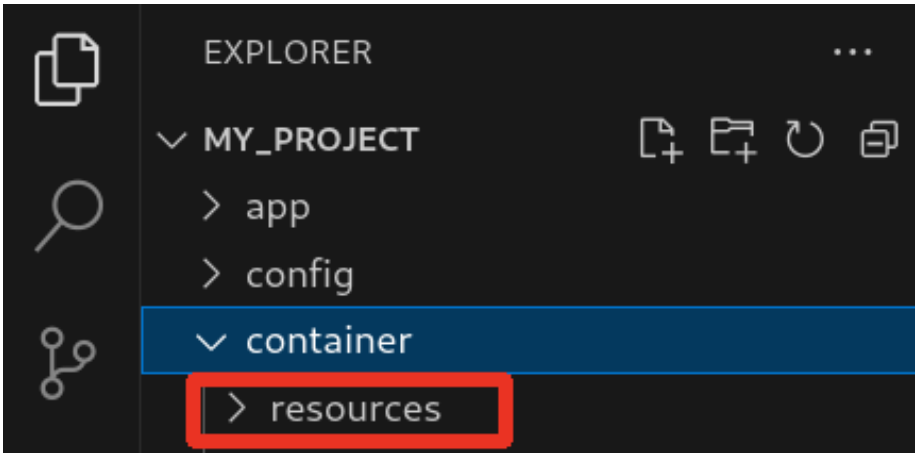


図 7.63 resources ディレクトリ

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

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

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

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

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

7.8.6.2. コンテナ内のファイル一覧の再表示

「図 7.61. コンテナ内のファイル一覧を表示するタブ」 の赤枠で囲われているボタンをクリックすることで、コンテナ内のファイル一覧を再表示します。

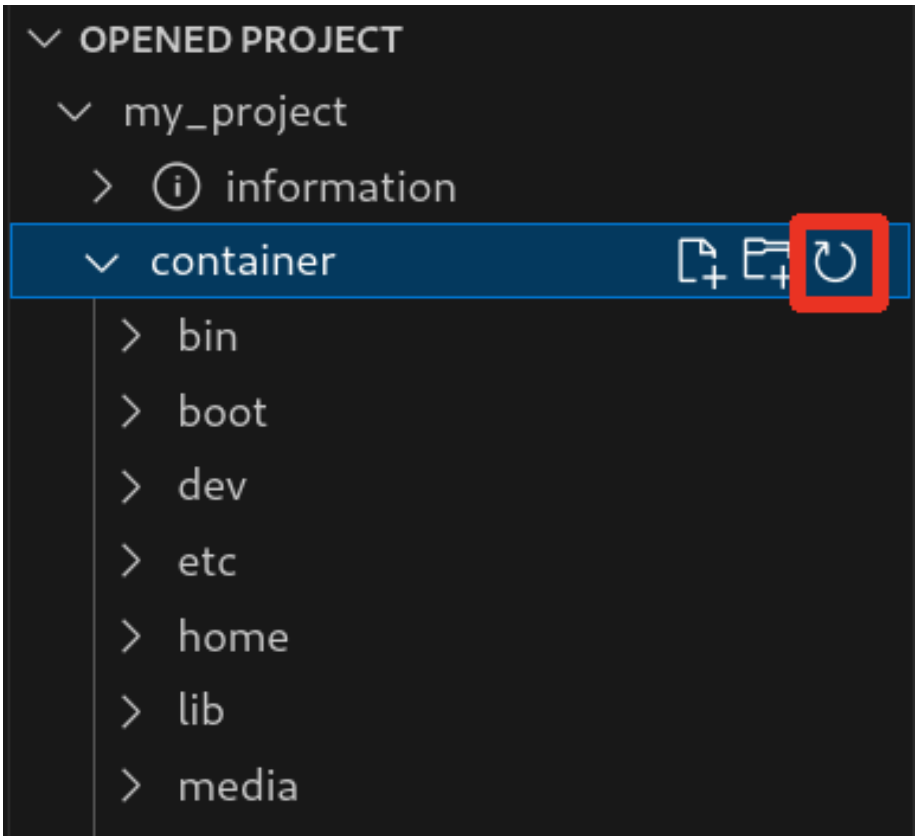


図 7.64 コンテナ内のファイル一覧を再表示するボタン

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

「図 7.65. container/resources 下にファイルを追加するボタン」 の赤枠で囲われている表記のボタンをクリックすることで、プロジェクトディレクトリにある container/resources 下にファイルを追加することが可能です。

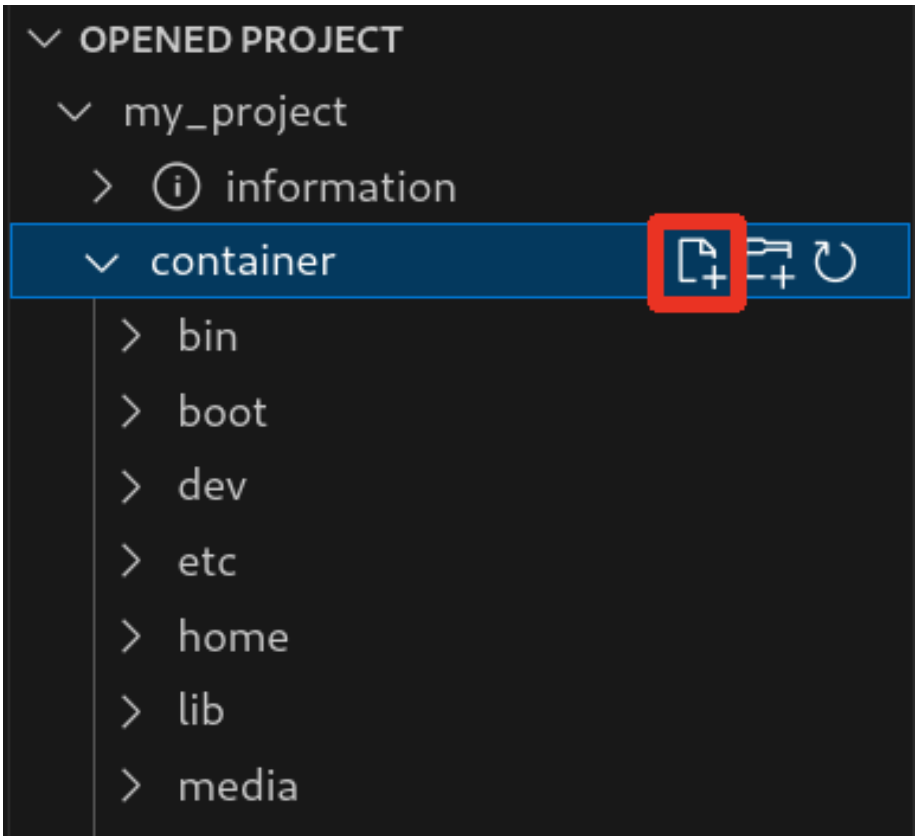


図 7.65 container/resources 下にファイルを追加するボタン

「図 7.66. ファイル名を入力」 に示すように、コマンドパレットが表示されますのでファイル名を入力してください。

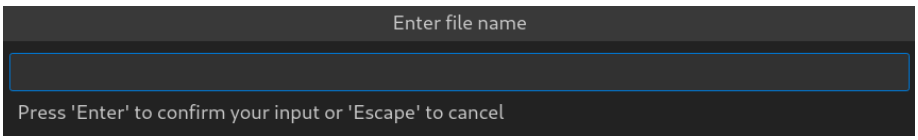


図 7.66 ファイル名を入力

例として、「add_file」というファイル名を入力したとします。「図 7.67. 追加されたファイルの表示」に示すように、追加したファイルには「A」というマークが表示されます。

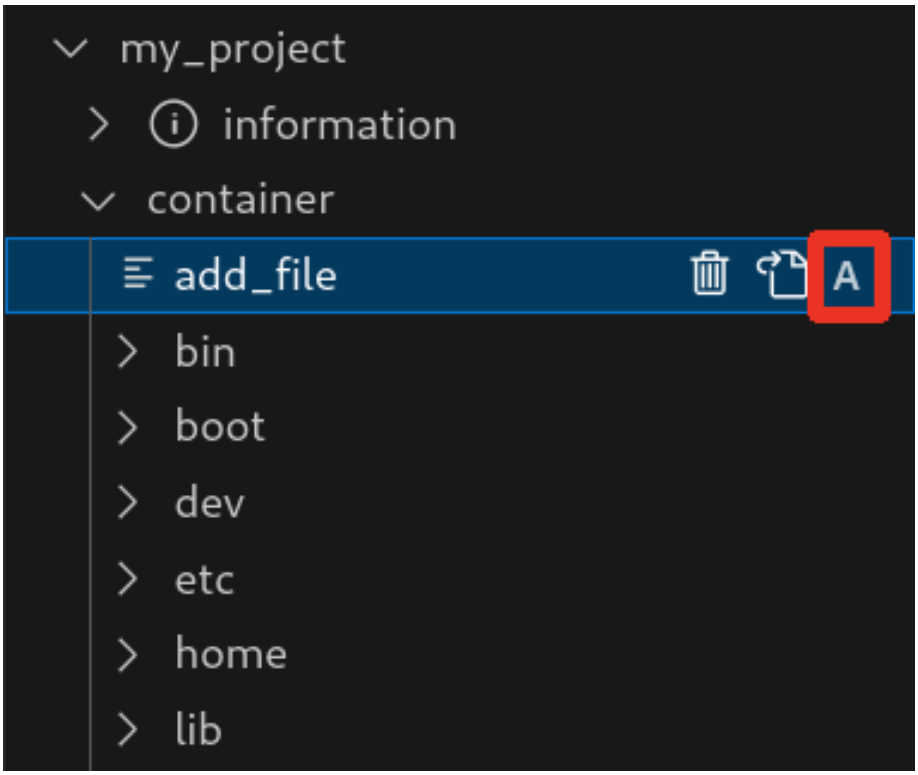


図 7.67 追加されたファイルの表示

また、「図 7.68. container/resources 下にフォルダーを追加するボタン」 の赤枠で囲われている表記のボタンをクリックすることで、ファイルの追加と同様の操作でディレクトリを追加することが可能です。

追加したディレクトリも同様に "A" というマークが表示されます。

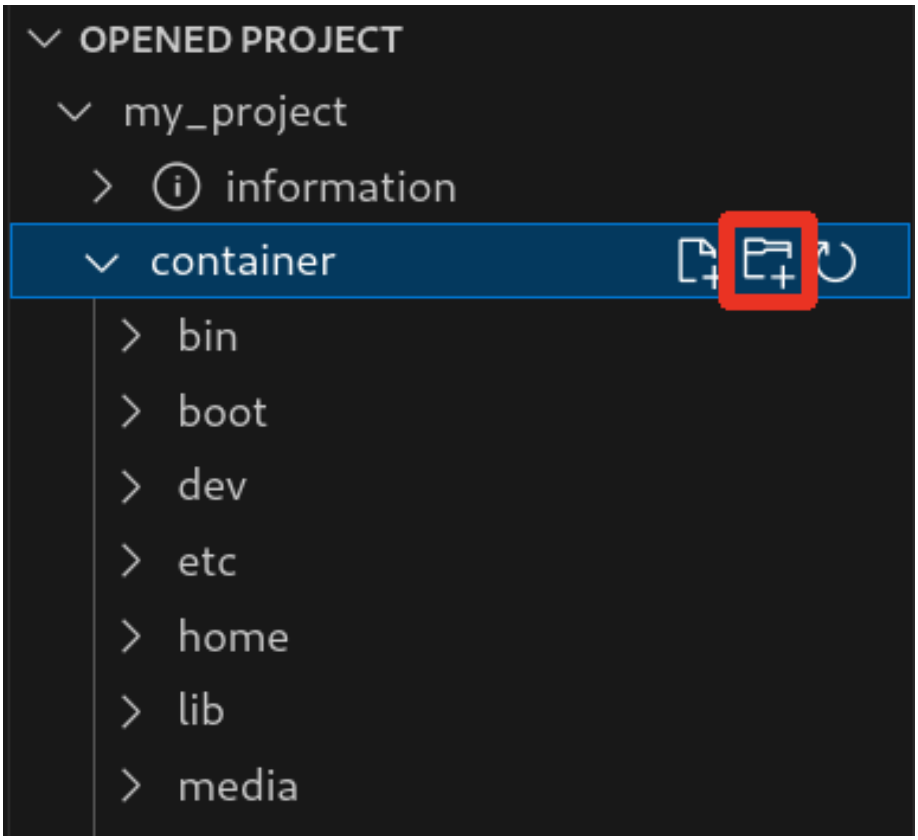


図 7.68 container/resources 下にフォルダーを追加するボタン

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

「図 7.69. container/resources 下にあるファイルを開くボタン」 の赤枠で囲われている表記のボタンをクリックすることで、プロジェクトディレクトリにある container/resources 下のファイルをエディタに表示することができます。

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

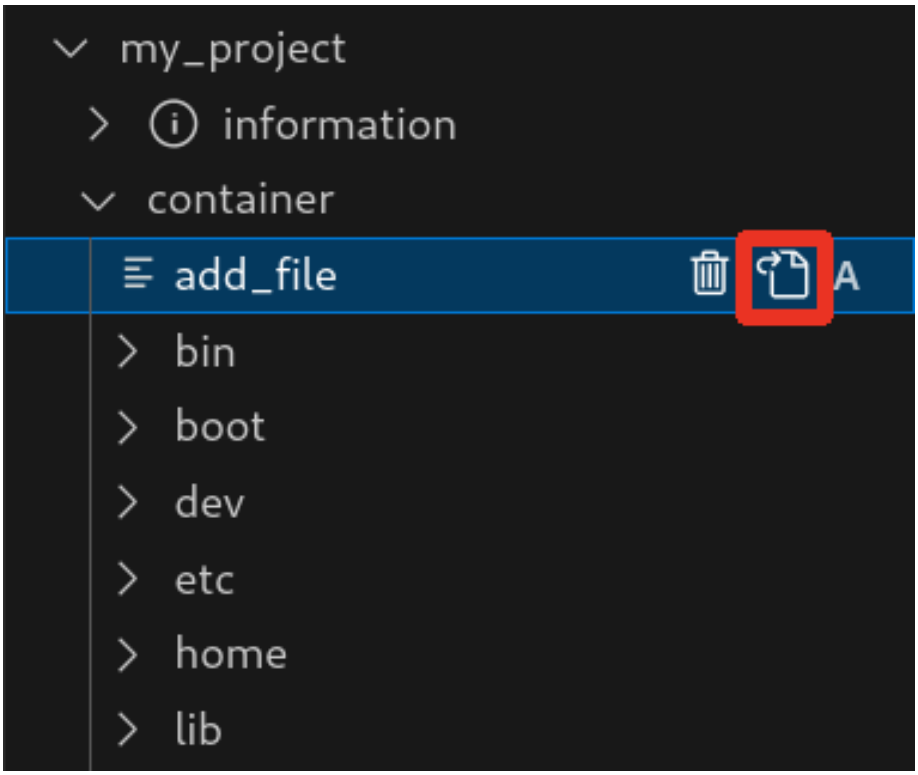


図 7.69 container/resources 下にあるファイルを開くボタン

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

「図 7.69. container/resources 下にあるファイルを開くボタン」 の赤枠で囲われている表記のボタンをクリックすることで、container/resources 下にあるファイルを削除することができます。

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

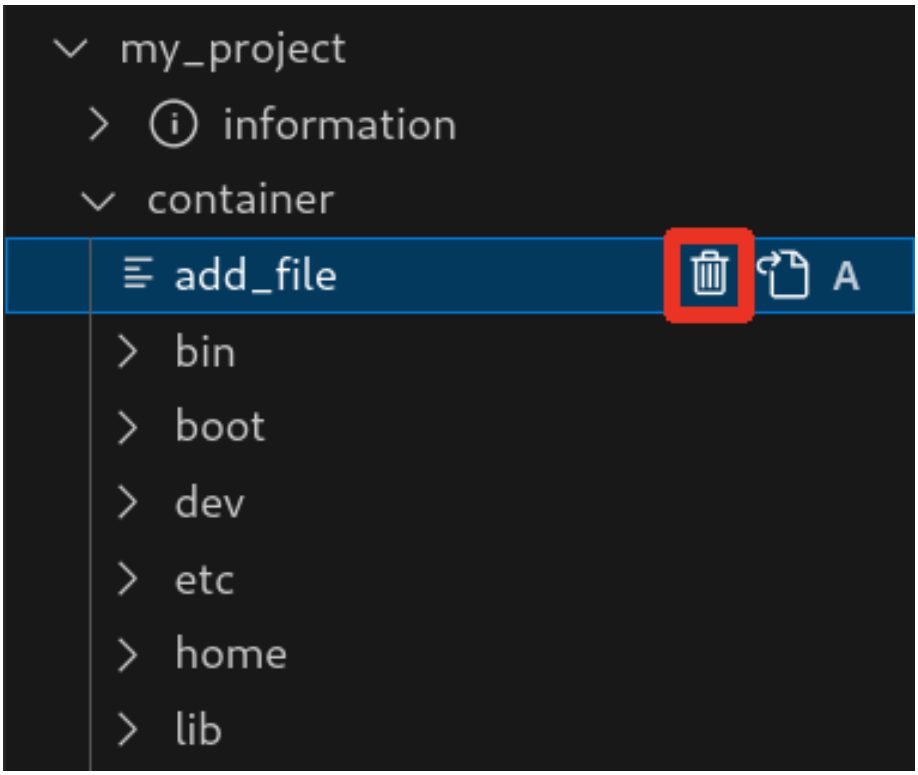


図 7.70 container/resources 下にあるファイルを削除するボタン

ディレクトリも同様に「図 7.69. container/resources 下にあるファイルを開くボタン」 の赤枠で囲われている表記のボタンをクリックすることで削除することができます。

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

「図 7.71. コンテナ内のファイルを container/resources 下に保存するボタン」 の赤枠で囲われている表記のボタンをクリックすることで、コンテナ内にあるファイルをプロジェクトディレクトリにある container/resources 下に保存します。

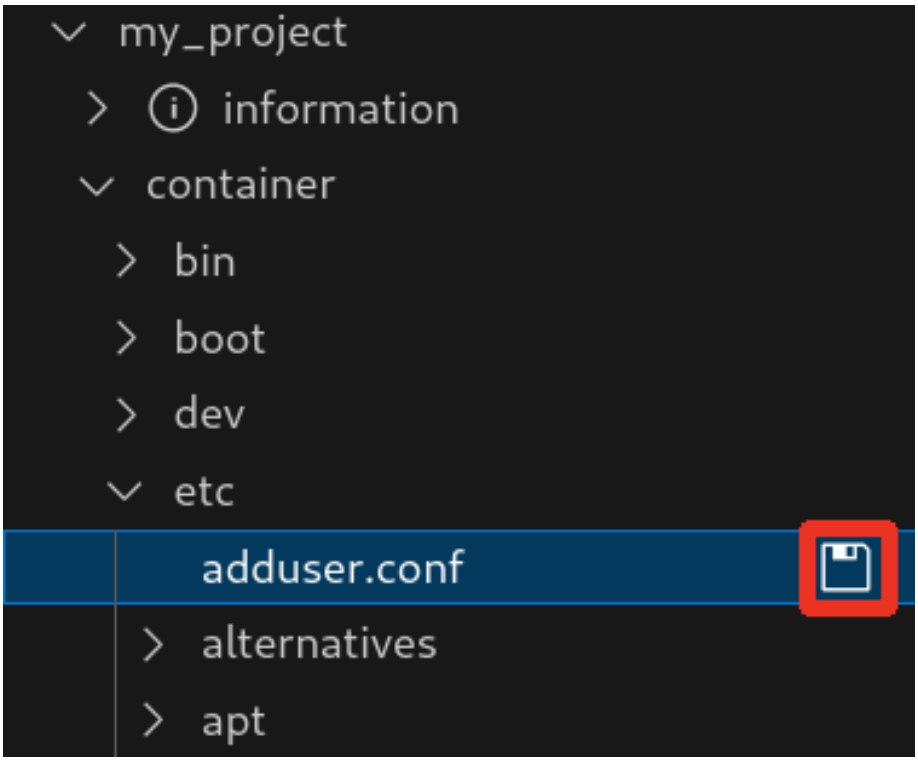


図 7.71 コンテナ内のファイルを container/resources 下に保存するボタン

ファイルが container/resources 下に保存されると、「図 7.72. 編集前のファイルを示すマーク」 に示すように、ファイル名の右側に "U" のマークが表示されます。

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

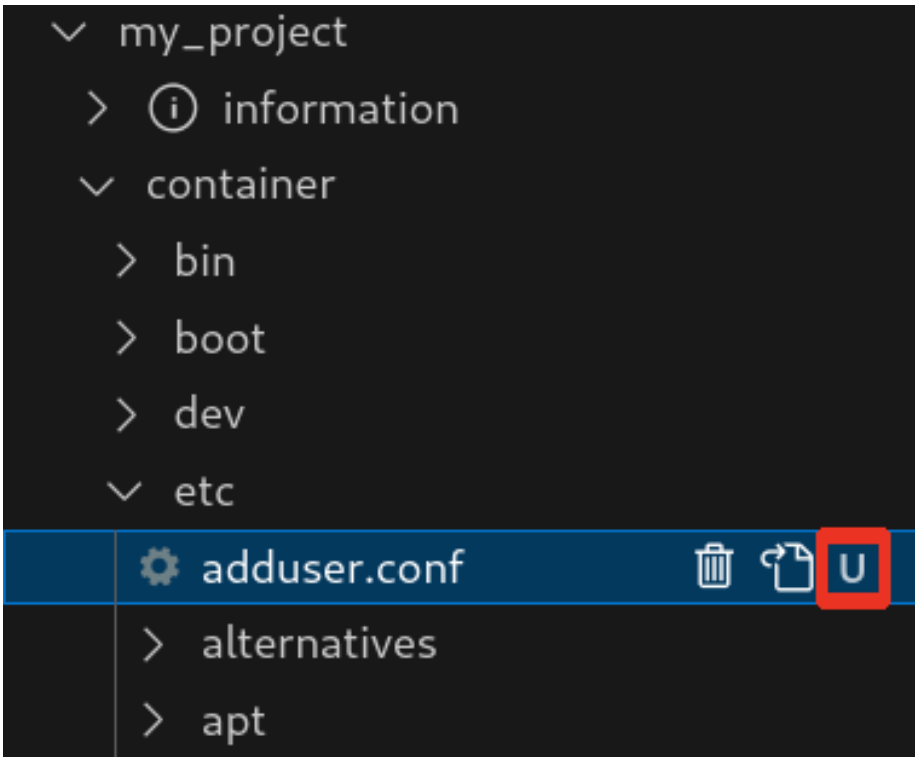


図 7.72 編集前のファイルを示すマーク

container/resources 下にあるファイルを編集して再表示すると、「図 7.73. 編集後のファイルを示すマーク」 に示すように、ファイル名の右側に "M" のマークが表示されます。

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

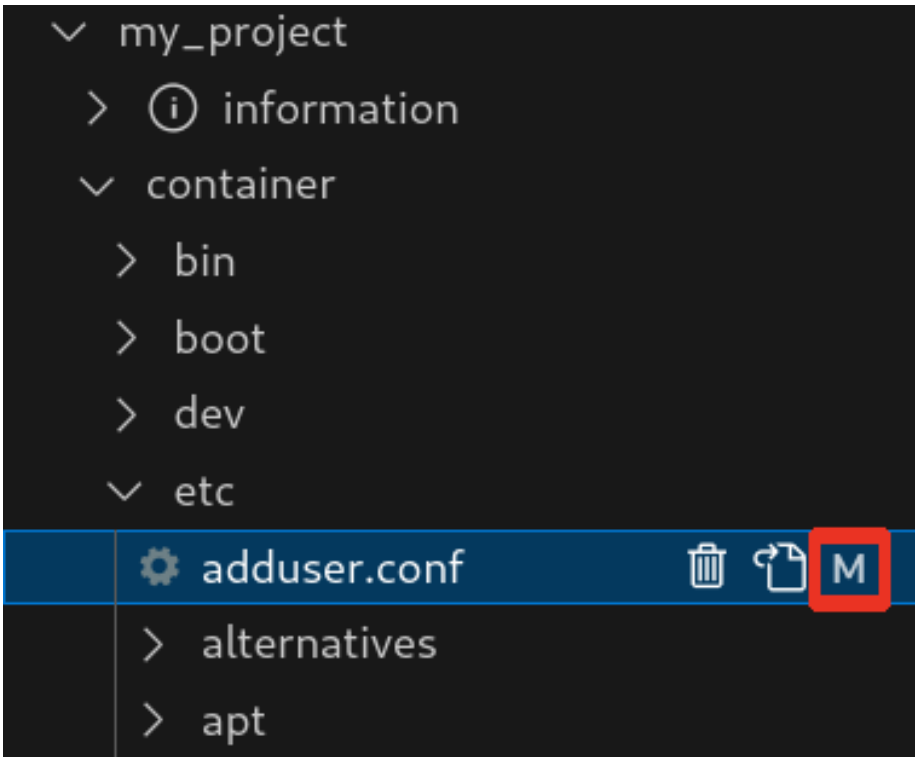


図 7.73 編集後のファイルを示すマーク

7.8.6.7. エラー表示

プロジェクトディレクトリにある container/resources 下とコンテナ内にあるファイルまたはディレクトリを比較して、同名でかつファイルの種類が異なる場合、「図 7.74. コンテナ内にコピーされないことを示すマーク」 に示すように、ファイル名の右側に "E" のマークが表示されます。

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

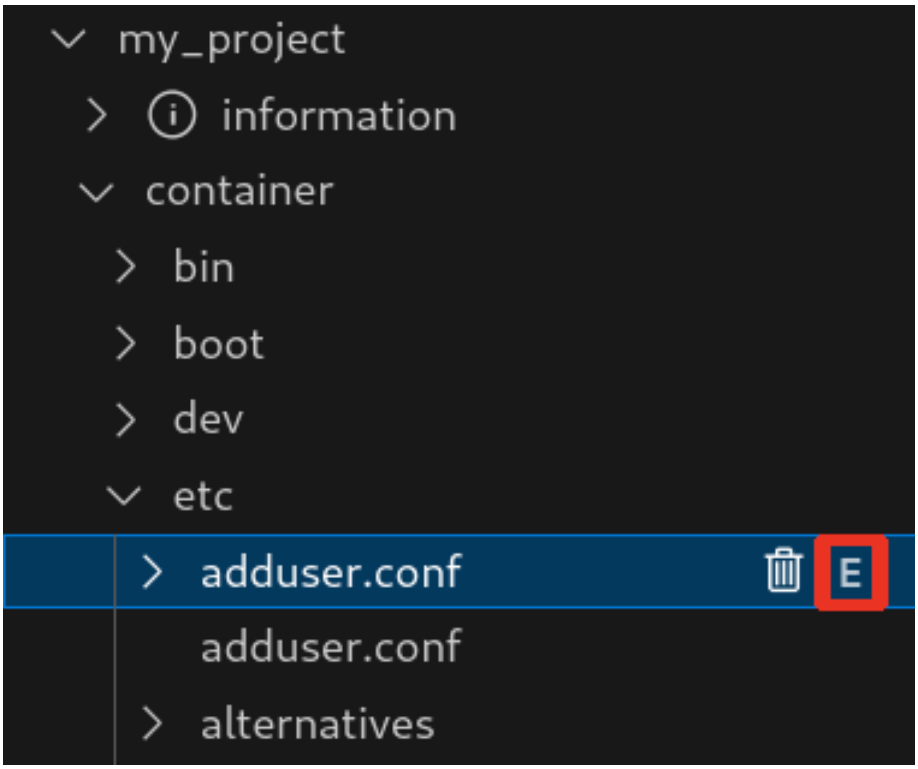


図 7.74 コンテナ内にコピーされないことを示すマーク

7.8.7. Armadillo 上でのセットアップ

7.8.7.1. ディスプレイの接続

「5.5.12. MIPI DSI ディスプレイ を使用する」 を参照して Armadillo にディスプレイを接続してください。

7.8.7.2. アプリケーション実行用コンテナイメージのインストール

「7.8.3.4. アプリケーション実行用コンテナイメージの作成」 で作成した `development.swu` を「7.3.3.6. SWU イメージのインストール」 を参照して Armadillo へインストールしてください。

インストール後に自動で Armadillo が再起動します。この際、 `weston` も自動起動します。

7.8.8. アプリケーション開発

7.8.8.1. アプリケーションのビルドモード

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

7.8.8.2. サンプルアプリケーションのビルド

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

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

図 7.75 my_project へ移動して VS Code を起動する。

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

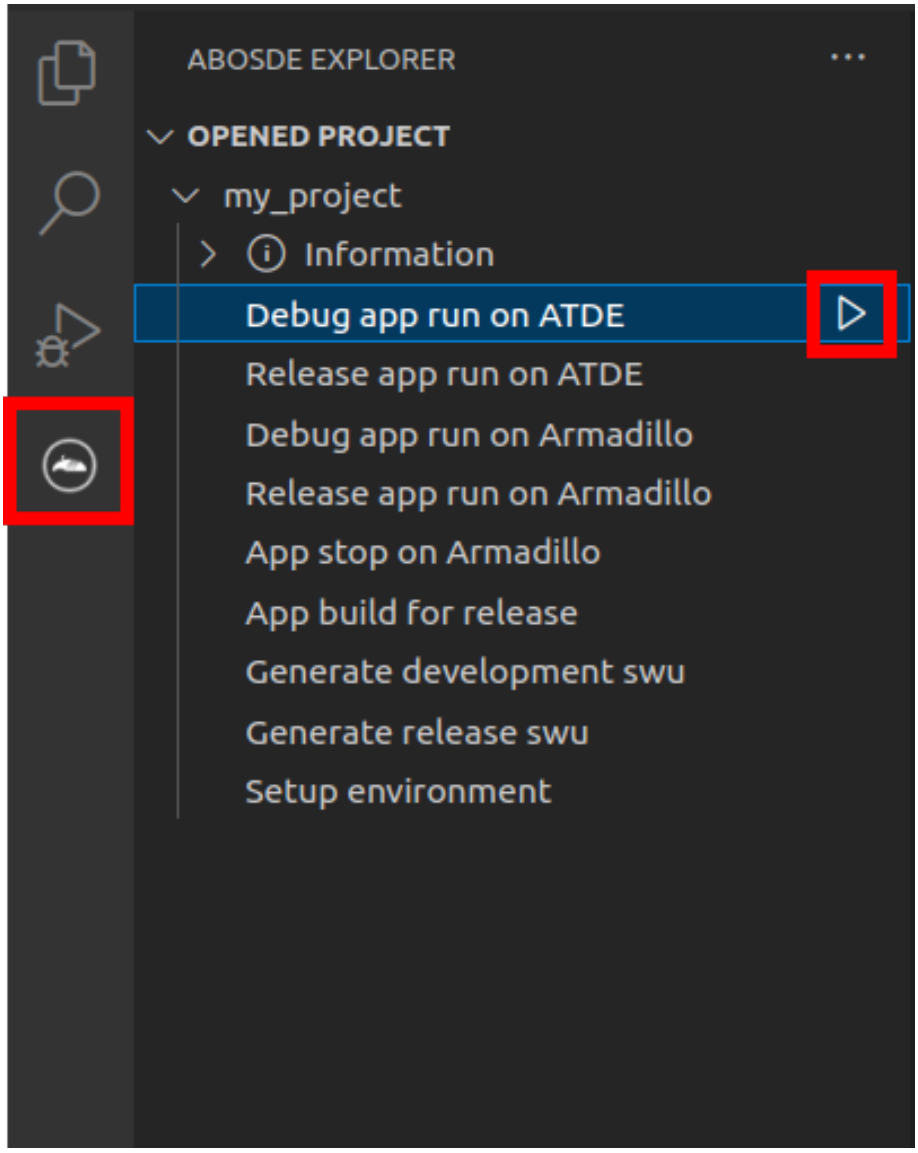


図 7.76 ATDE 上で Debug モードでビルドしたアプリケーションを実行する

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

GUI アプリケーションの場合は以下のようなアプリケーションが起動します。

図 7.77 起動したサンプルアプリケーション

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

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

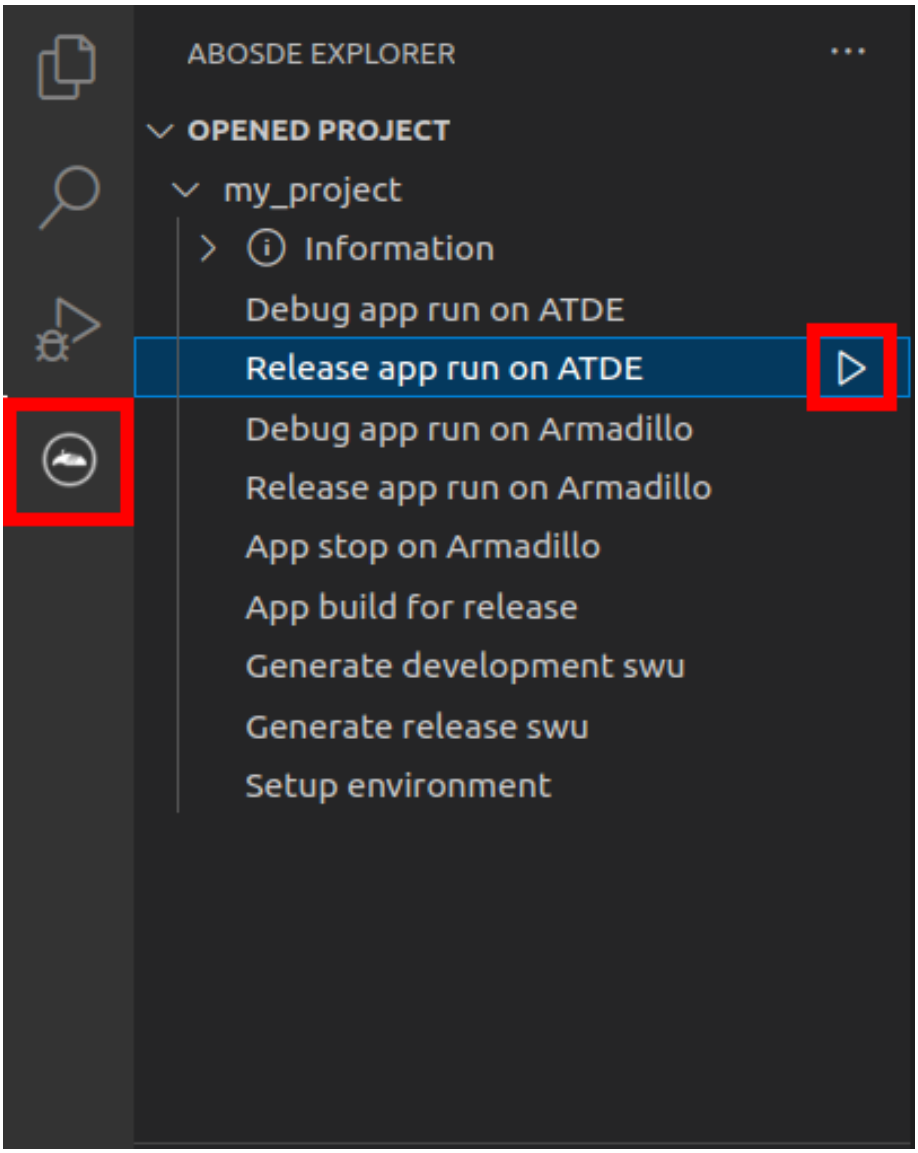


図 7.78 ATDE 上で Release モードでビルドしたアプリケーションを実行する

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

7.8.8.3. パッケージをインストールする

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

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

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

図 7.79 dart_periphery パッケージをインストールする例

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

表 7.5 組み合わせて使うパッケージ

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

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

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

図 7.80 video_player パッケージをインストールする例

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

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

図 7.81 dart_periphery パッケージをアンインストールする例

7.8.8.4. BLE パッケージをインストールする

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

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

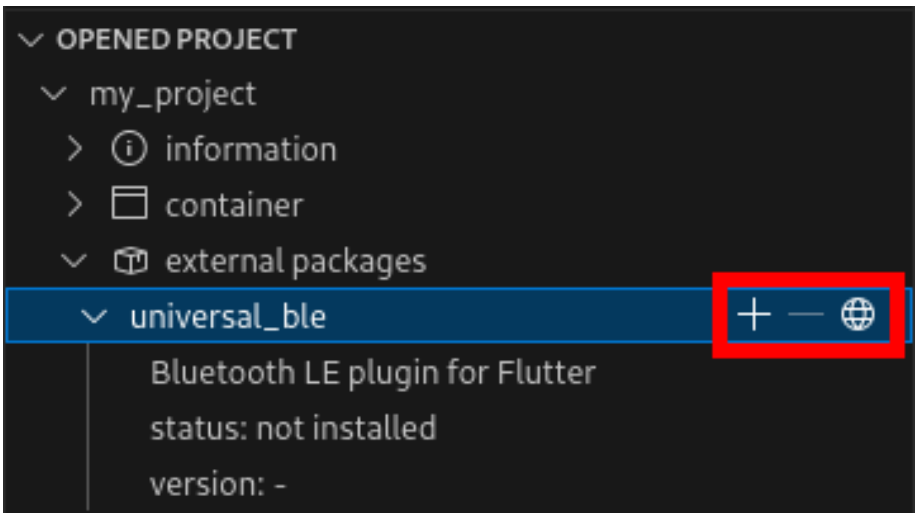


図 7.82 BLE パッケージをインストールする

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

7.8.9. 動作確認

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

7.8.9.1. ssh 接続に使用する IP アドレスの設定

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

「図 7.83. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする」 の赤枠で囲われているボタンをクリックすることで、ローカルネットワーク上で ABOS Web が実行されている Armadillo をスキャンすることができます。

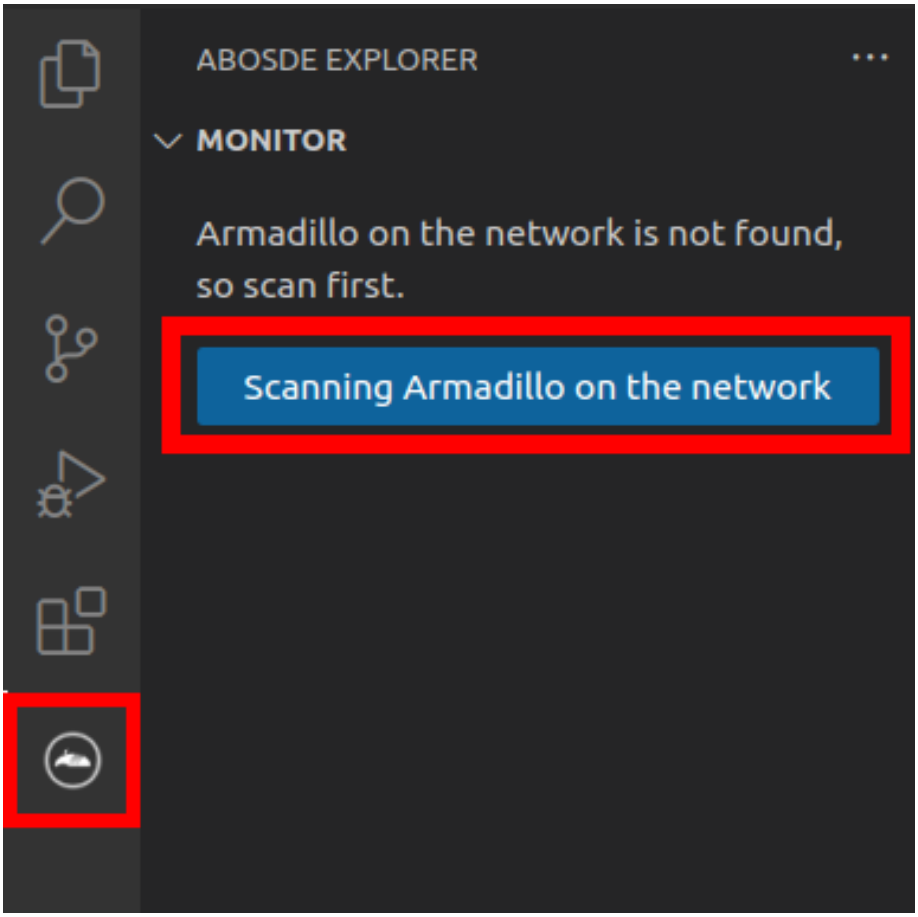


図 7.83 ABOSDE で ローカルネットワーク上の Armadillo をスキャンする

「図 7.84. ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する」 の赤枠で囲われているマークをクリックすることで、指定した Armadillo の IP アドレスを ssh 接続に使用する IP アドレスに設定することができます。

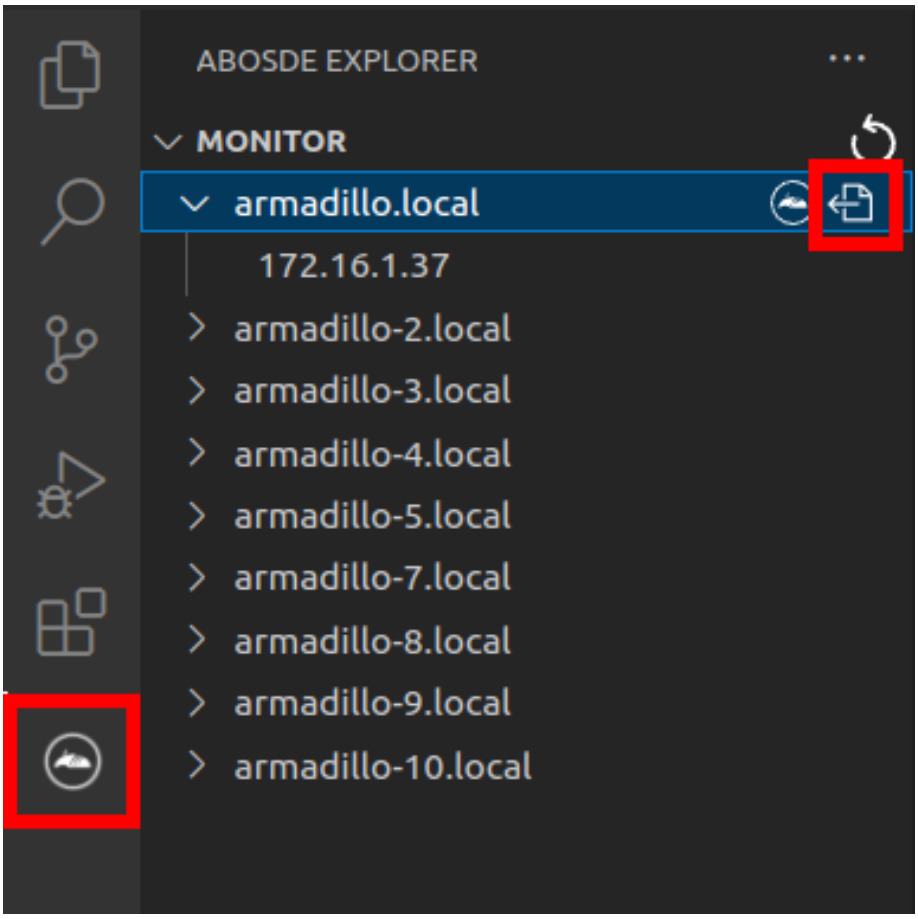


図 7.84 ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する

「図 7.85. ABOSDE に表示されている Armadillo を更新する」 の赤枠で囲われているマークをクリックすることで、ABOSDE に表示されている Armadillo を更新することができます。

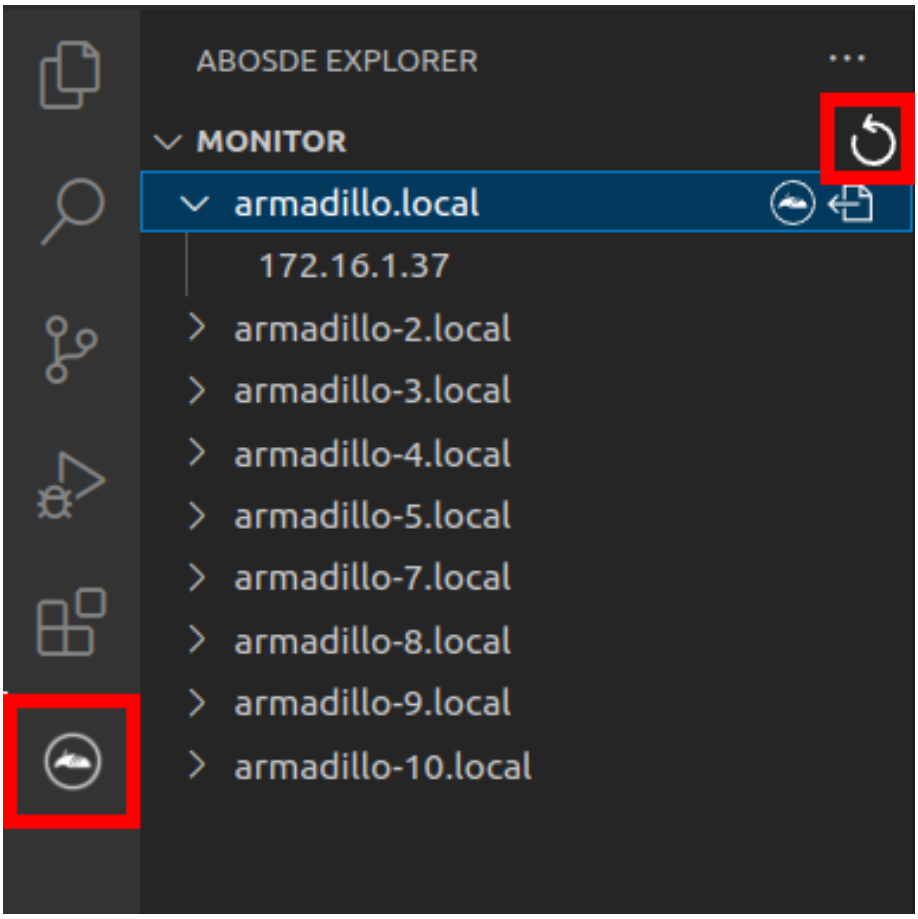


図 7.85 ABOSDE に表示されている Armadillo を更新する

ATDE のネットワークを NAT に設定している場合や、ABOS Web を起動していない場合等、ABOSDE のリストに Armadillo が表示されない場合は、プロジェクトディレクトリに入っている config/ssh_config ファイルを編集して IP アドレスを書き換えてください。

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

図 7.86 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。

7.8.9.2. アプリケーションの実行

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

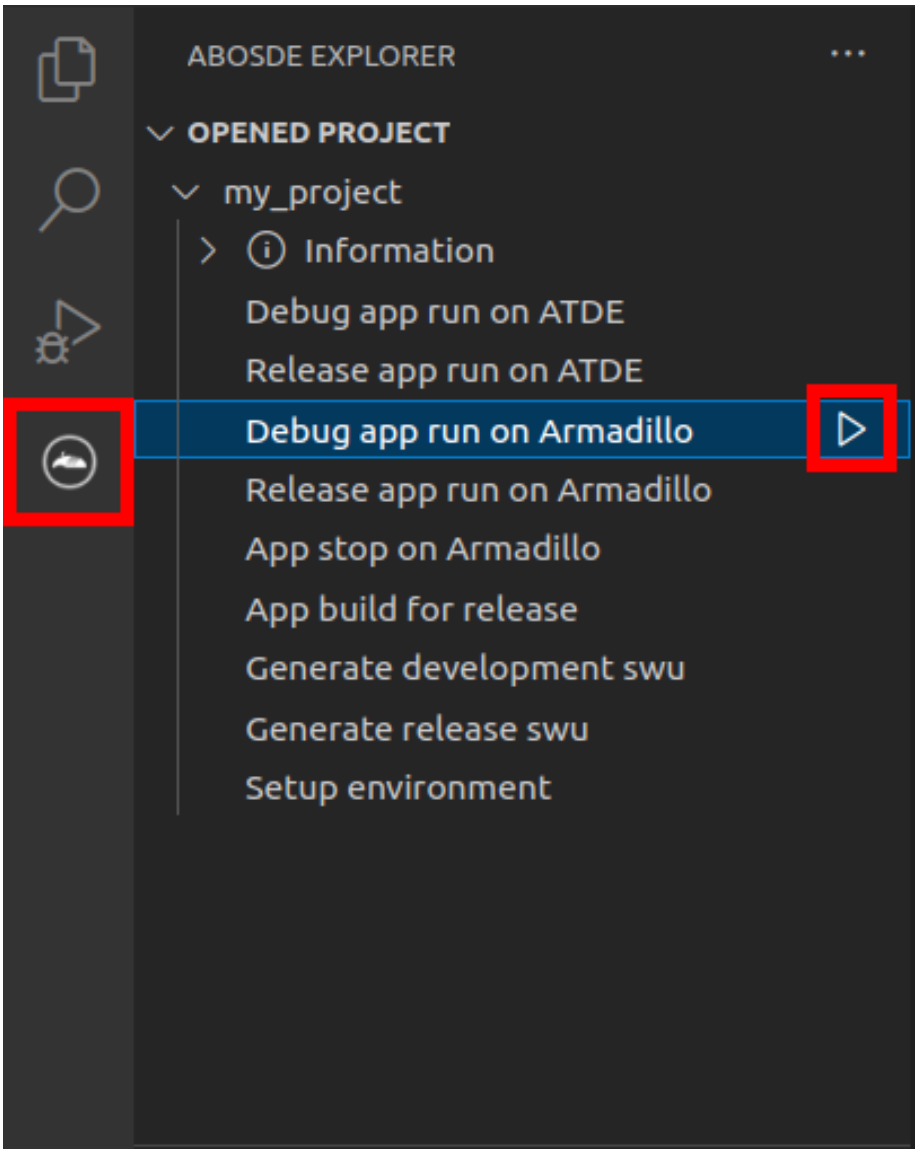


図 7.87 Armadillo 上で Debug モードでビルドしたアプリケーションを実行する

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

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

図 7.88 実行時に表示されるメッセージ

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

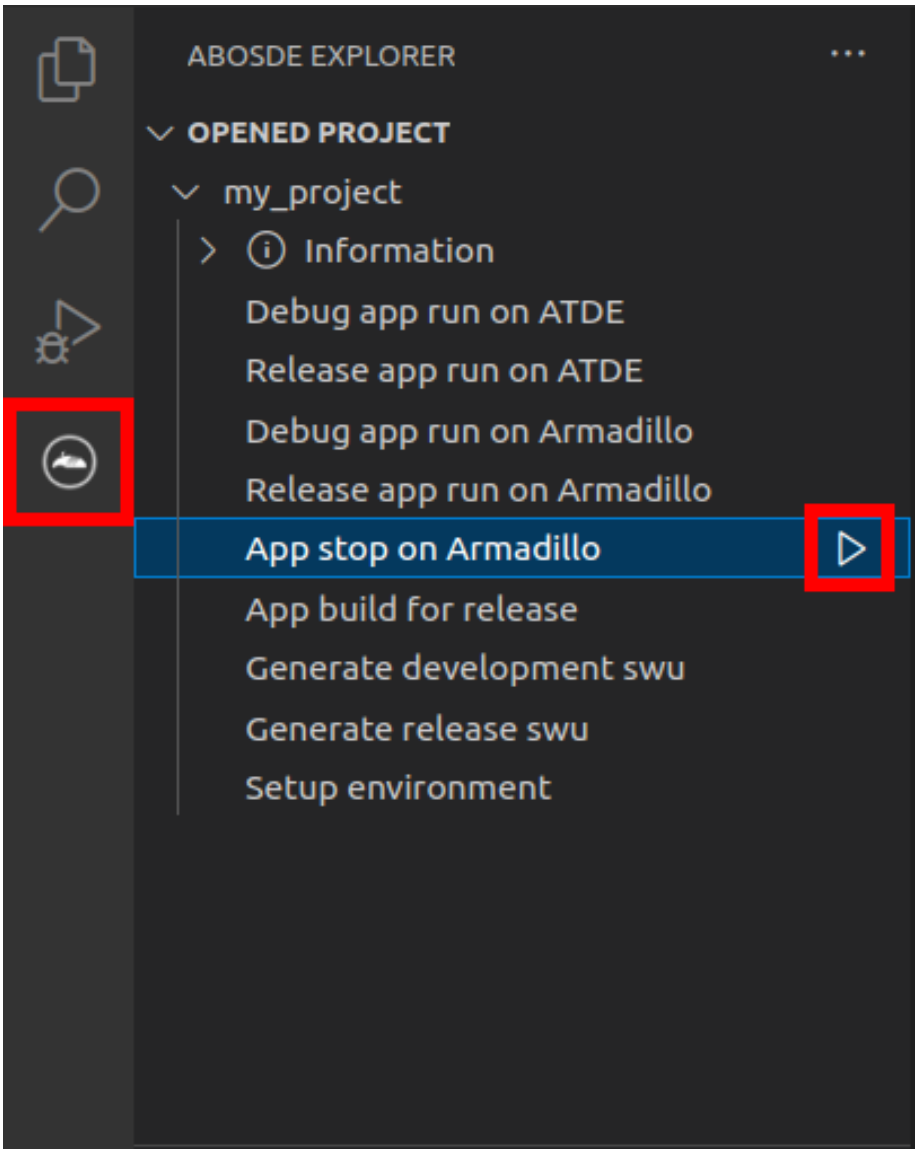


図 7.89 アプリケーションを終了する

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

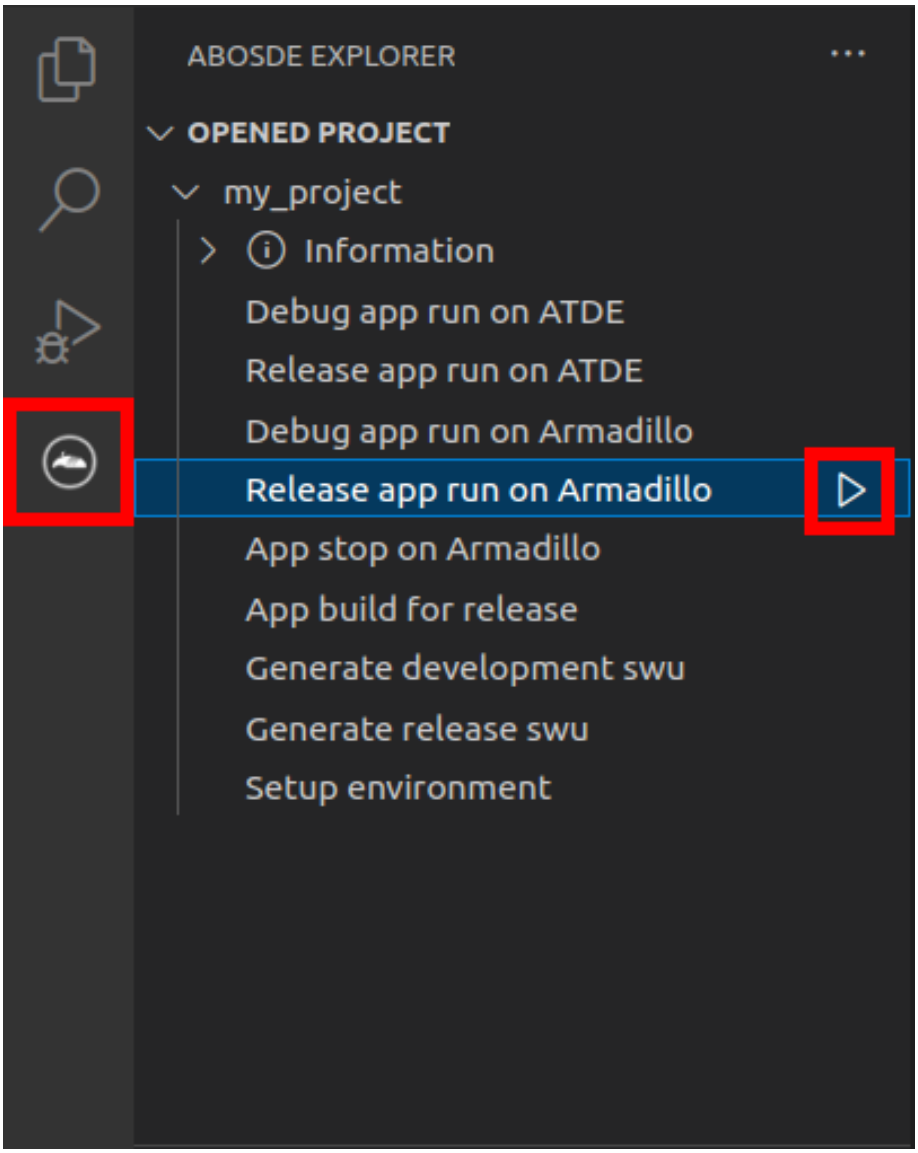


図 7.90 Armadillo 上で Release モードでビルドしたアプリケーションを実行する

7.8.9.3. ホットリロード

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

ホットリロード機能を使うには、アプリケーション実行時に表示される VS Code 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 7.91 ホットリロード機能を使う

7.8.10. アプリケーションをデバッグする

アプリケーションをデバッグモードで起動することで VS Code のデバッガ機能を使用したデバッグができるようになります。デバッガを使用すると、ブレークポイントで処理を止めて変数の値を確認したり、ステップ実行などができるようになります。

デバッグモードのアプリケーションでは、リリースモードに比べて動作が遅くなり操作に対するレスポンスが遅延することがあります。特に外部機器と接続してデータをやり取りするようなアプリケーションで、遅延が原因で接続がタイムアウトしてしまうようなことがある場合は、デバッグモードの場合はタイムアウトしないようにするなどの対応を行う必要があります。

デバッグ完了後、最終的にはリリースモードでのテストを行ってください。

7.8.10.1. VS Code に Flutter エクステンションをインストールする

VS Code に Flutter エクステンションをインストールします。マーケットプレイスの検索フォームに「flutter」と入力し、表示された「Flutter」の 「Install」ボタンをクリックしてインストールしてください。

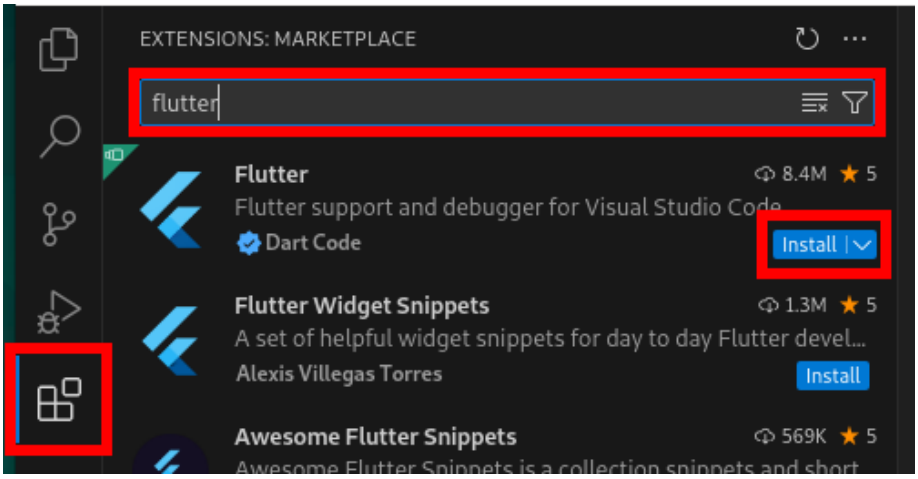


図 7.92 Flutter エクステンションをインストールする

次に Flutter Sdk Path を設定します。VS Code 上で `Ctrl + ,` キーを押して設定画面を開き、検索フォームに「Flutter Sdk Paths」と入力し、「Add Item」をクリックしてください。

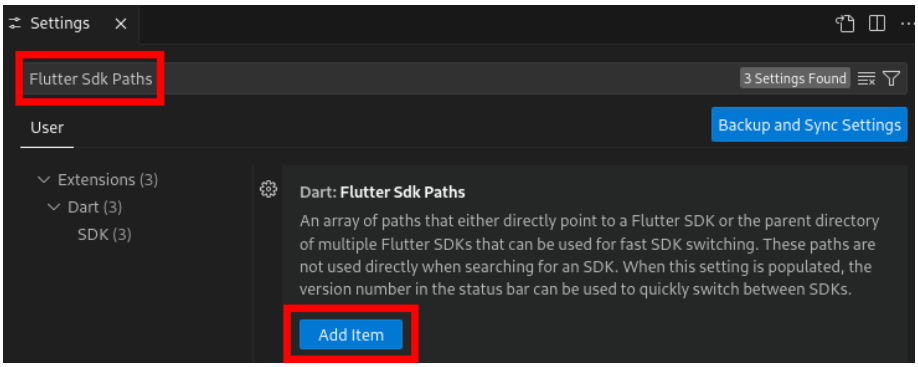


図 7.93 Flutter Sdk Paths の設定画面を表示する

表示されたフィールドに、`/opt/flutter-elinux/flutter` と入力し「OK」をクリックしてください。

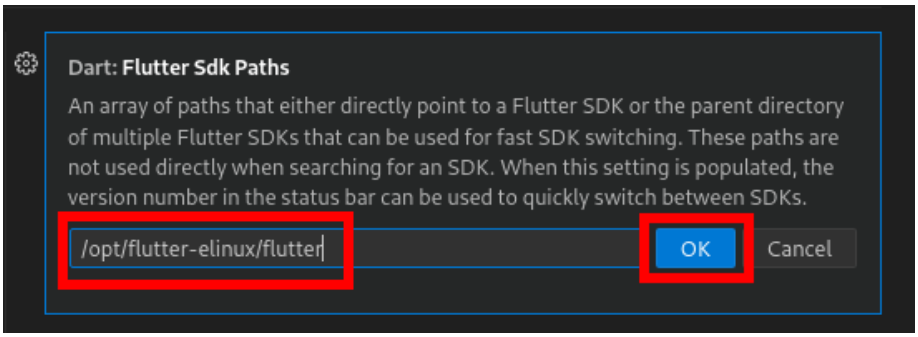


図 7.94 Flutter Sdk Paths を設定する

7.8.10.2. アプリケーションを Debug モードで実行する

VS Code の左ペインの [my_project] から [Debug app run on ATDE] または [Debug app run on Armadillo] を実行してください。

7.8.10.3. アプリケーションのデバッグを開始する

VS Code の [Run and Debug] 画面を表示し、上部の三角ボタンをクリックすると VS Code からアプリケーションのデバッグを開始します。

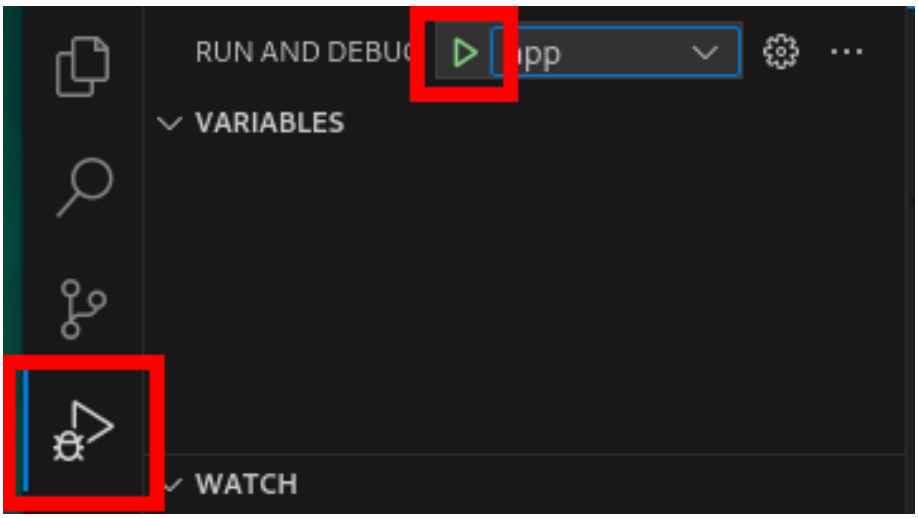


図 7.95 デバッグを開始する

7.8.10.4. デバッガの機能

デバッグを開始すると VS Code のウィンドウ上部にデバッガを操作するためのアイコンが表示されます。

図 7.96 デバッガのアイコン

デバッガの機能としては次のものがあります。

- ブレークポイントの設定

  ソースコードの行数表示の左をクリックすると丸印が付きます。処理をここで止めることができます。

- Continue

  ブレークポイントで停止したところから再び実行を継続します。

- Step Over

  クリックするごとに、ブレークポイントで停止したところから一行ずつ実行します。

- Step Into

  関数の中へ入ります。

- Step Out

  関数の中にいる場合、その関数を最後まで実行した後に関数から出ます。

- Hot Reload

  「図 7.91. ホットリロード機能を使う」 と同じ機能です。

- Restart

  アプリケーションを再起動します。

- Disconnect

  デバッグを終了します。

- Widget Inspector

  アプリケーション上に配置してある Widget のツリー構成を確認できます。

7.8.11. SBOM 生成に関する設定

SWU イメージ作成時に、同時に SBOM を生成することができます。詳細は 「7.11. SBOM 生成に関わる設定を行う」 を参照してください。

7.8.12. リリース版のビルド

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

VS Code の左ペインの [my_project] から [Generate release swu] を実行すると、リリース版のアプリケーションを含んだ SWU イメージが作成されます。事前に 「9.3.1. SWU イメージの作成」 を参照して SWU の初期設定を行ってください。

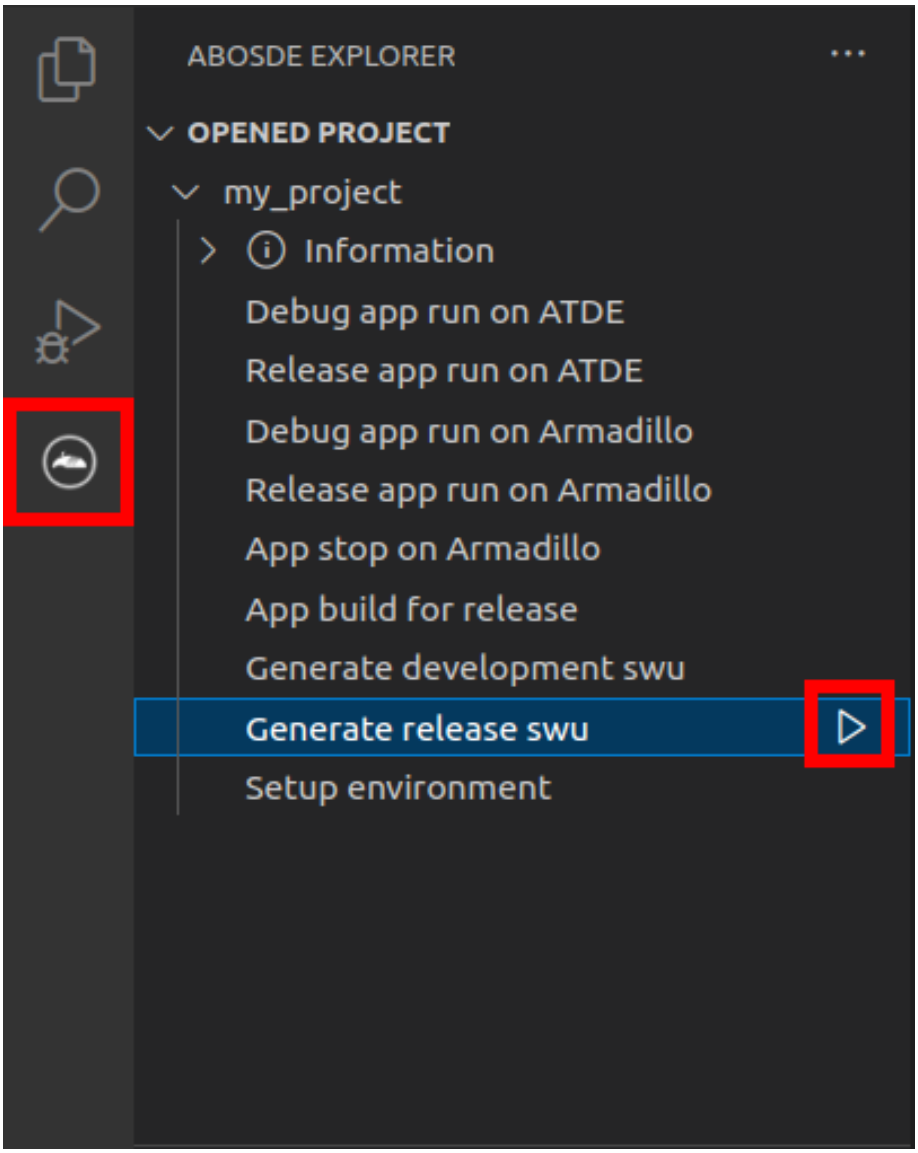


図 7.97 リリース版をビルドする

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

7.8.13. 製品への書き込み

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

この SWU イメージを 「7.3.3.6. SWU イメージのインストール」 を参照して Armadillo へインストールすると、Armadillo 起動時にアプリケーションも自動起動します。

7.8.14. Armadillo 上のコンテナイメージの削除

`development.swu` または `release.swu` を Armadillo にインストールすることで保存されたコンテナイメージを削除する方法は、 「10.8.2.1. VS Code から実行する」 を参照してください。

7.9. CUI アプリケーションの開発

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

7.9.1. 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。

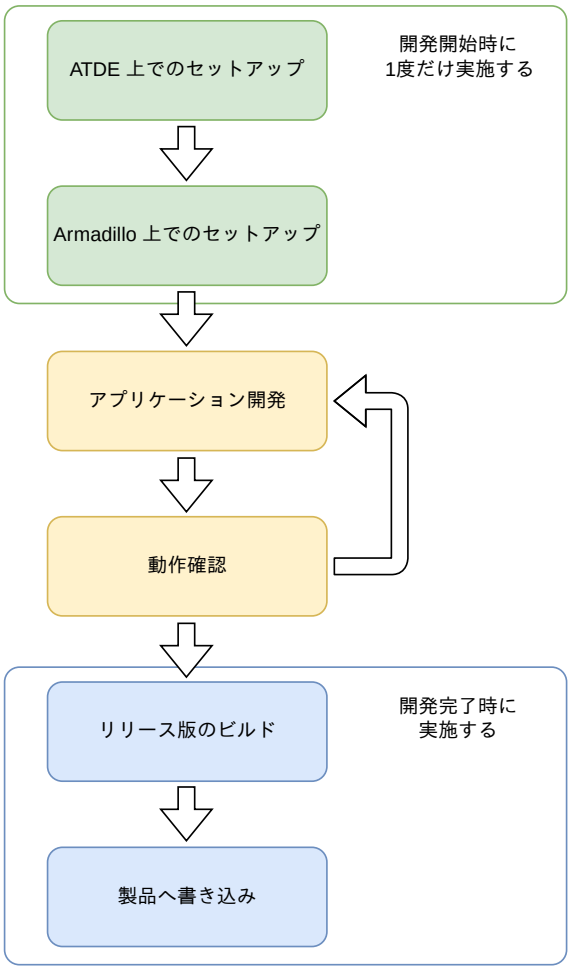


図 7.98 CUI アプリケーション開発の流れ

7.9.2. 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 「7.1. 開発の準備」 を参照して ATDE 及び、 VS Code のセットアップを完了してください。

7.9.2.1. プロジェクトの作成

VS Code の左ペインの [A9E] から [Shell New Project] を実行し、表示されるディレクトリ選択画面からプロジェクトを保存するディレクトリを選択してください。実行するためには右に表示されている三角形ボタンを押してください。Python の場合は [Python New Project] を実行してください。保存先を選択すると、プロジェクト名を入力するダイアログが表示されるので、任意のプロジェクト名を入力してエンターキーを押してください。この操作により、選択した保存先に、入力したプロジェクト名と同名のディレクトリが作成されます。

また、ここでは次のように設定しています。

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

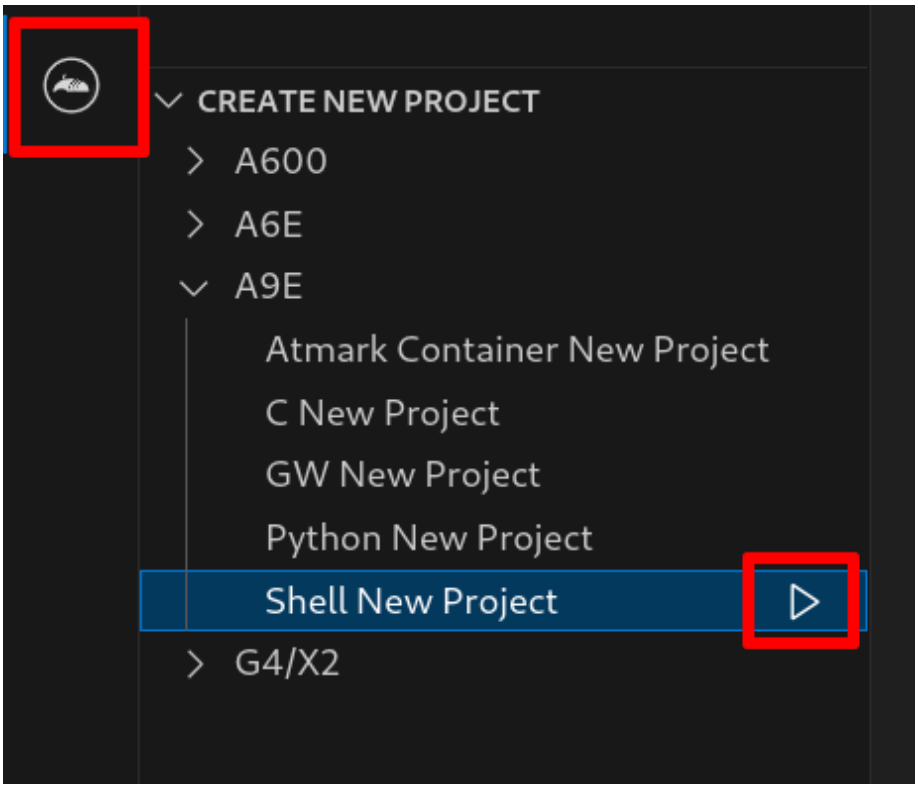


図 7.99 プロジェクトを作成する

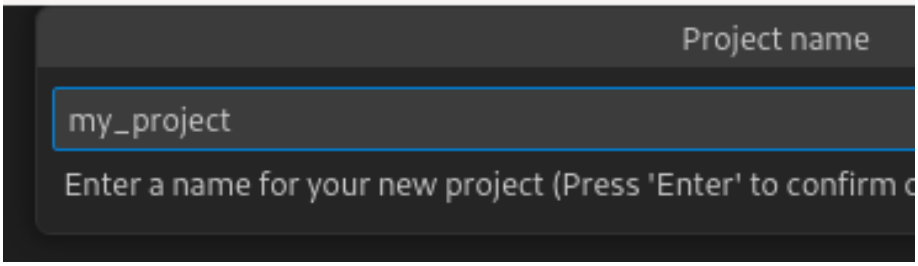


図 7.100 プロジェクト名を入力する

7.9.3. アプリケーション開発

7.9.3.1. VS Code の起動

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

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

図 7.101 VS Code で my_project を起動する

7.9.3.2. ディレクトリ構成

プロジェクトには下記のディレクトリがあります。

- **app** : アプリケーションのソースです。Armadillo ではビルドしたアプリケーションが /var/app/rollback/volumes/my_project にコピーされます。
  - **requirements.txt** : Python プロジェクトにのみ存在しており、このファイルに記載したパッケージは pip を使用してインストールされます。
- **config** : 設定に関わるファイルが含まれるディレクトリです。
  - **app.conf** : コンテナのコンフィグです。記載内容については 「10.8.3. コンテナ起動設定ファイルを作成する」 を参照してください。
  - **app.desc** : SWU イメージを生成するための .desc ファイルです。記載内容については 「10.3. mkswu の .desc ファイルを編集する」 を参照してください。
  - **ssh_config** : Armadillo への ssh 接続に使用します。「7.9.7.2. ssh 接続に使用する IP アドレスの設定」 を参照してください。
- **container** : スクリプトを実行するコンテナの設定ファイルが含まれるディレクトリです。
  - **packages.txt** : このファイルに記載されているパッケージがインストールされます。
  - **Dockerfile** : 直接編集することも可能です。

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

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

7.9.3.3. 初期設定

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

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

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

図 7.102 初期設定を行う

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

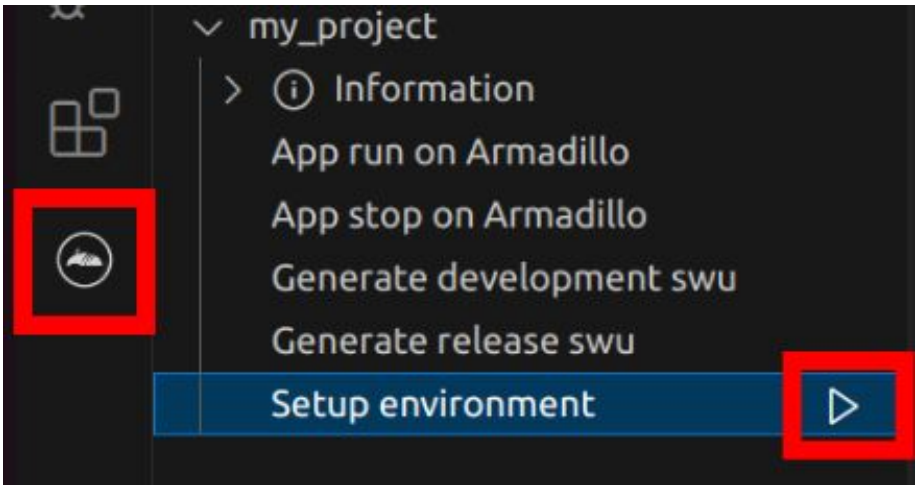


図 7.103 VS Code で初期設定を行う

選択すると、 VS Code の下部に以下のようなターミナルが表示されます。

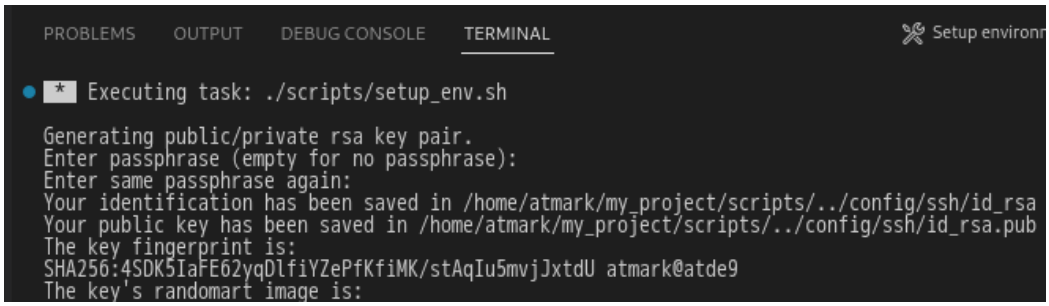


図 7.104 VS Code のターミナル

このターミナル上で以下のように入力してください。

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

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

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

図 7.105 SSH 用の鍵を生成する

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

❷ 1 でパスフレーズを設定した場合は、確認のため再度入力してください。

❸ ここで何か任意のキーを押すとターミナルが閉じます。

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

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

7.9.3.4. アプリケーション実行用コンテナイメージの作成

Armadillo 上でアプリケーションを実行するためのコンテナイメージを作成します。ここで作成したコンテナイメージは SWU イメージを使用して Armadillo へインストールするため、事前に 「9.3.1. SWU イメージの作成」 を参照して SWU の初期設定を行ってください。

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

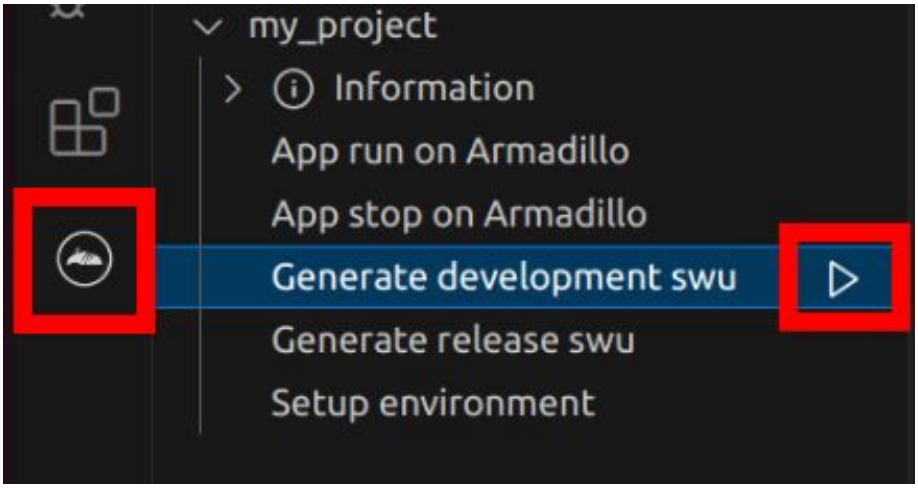


図 7.106 VS Code でコンテナイメージの作成を行う

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

```
コンテナイメージを ./swu/my_project.tar に保存しました。
./swu/app.desc のバージョンを 1 から 2 に変更しました。
./development.swu を作成しました。
次は Armadillo に ./development.swu をインストールしてください。
 *  Terminal will be reused by tasks, press any key to close it.
```

図 7.107 コンテナイメージの作成完了

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

7.9.3.5. Python アプリケーションに Bluetooth Low Energy パッケージをインストールする

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

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

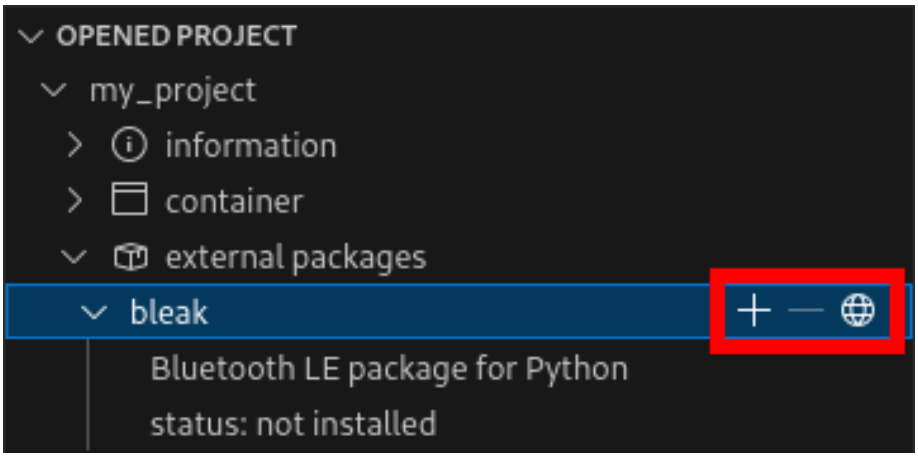


図 7.108 Bluetooth Low Energy パッケージをインストールする

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

Bluetooth Low Energy パッケージのインストールは ABOSDE のバージョン 1.8.4 以降で、かつ 2024 年 7 月 24 日以降に「7.9.2.1. プロジェクトの作成」 の手順で新たに作成したプロジェクトで使用できるようになります。

7.9.4. コンテナのディストリビューション

使用するコンテナのディストリビューションは以下のとおりです。

ディストリビューション ・ debian:bullseye-slim

7.9.5. Armadillo に転送するディレクトリ及びファイル

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

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

7.9.6. コンテナ内のファイル一覧表示

「図 7.109. コンテナ内のファイル一覧を表示するタブ」 の赤枠で囲われているタブをクリックすることで、development.swu または「7.9.9. リリース版のビルド」で作成される release.swu に含まれるコンテナ内のファイルおよびディレクトリを表示します。

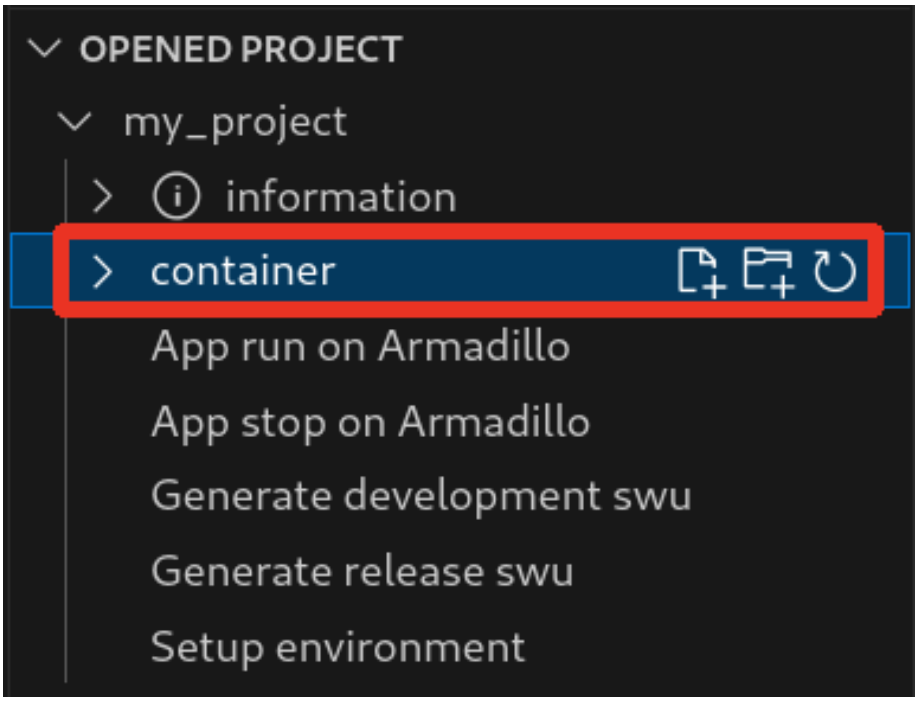


図 7.109 コンテナ内のファイル一覧を表示するタブ

クリック後の表示例を 「図 7.110. コンテナ内のファイル一覧の例」 に示します。

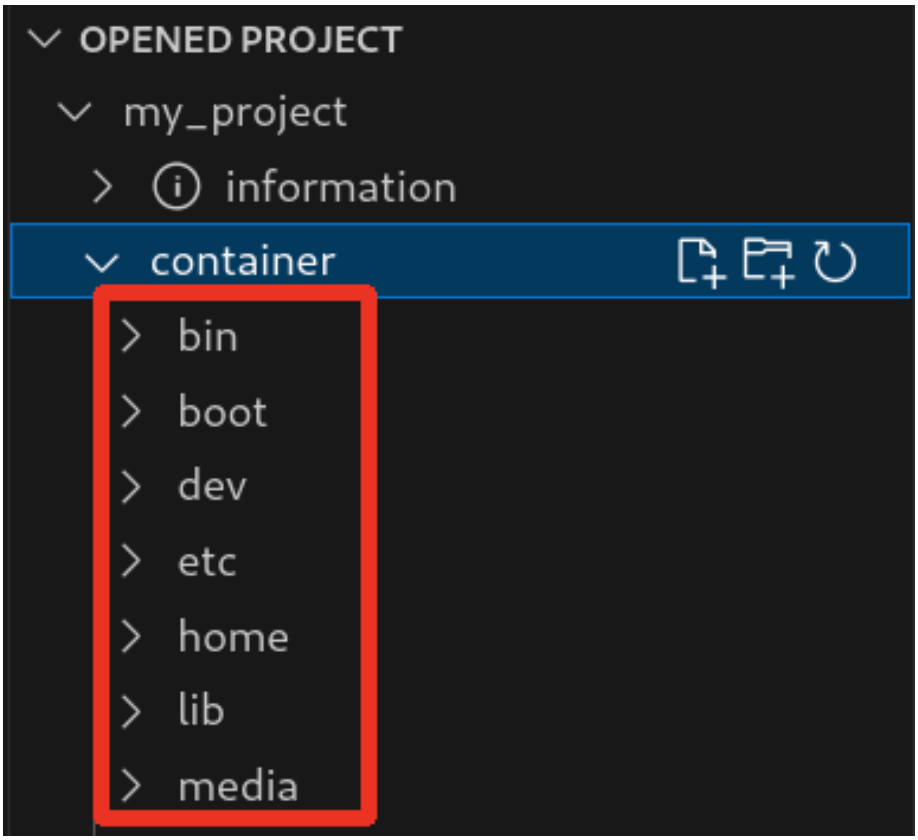


図 7.110 コンテナ内のファイル一覧の例

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

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

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

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

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

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

7.9.6.1. resources ディレクトリについて

「図 7.111. resources ディレクトリ」に示すように ATDE 上のプロジェクトディレクトリには container/resources ディレクトリがあります。

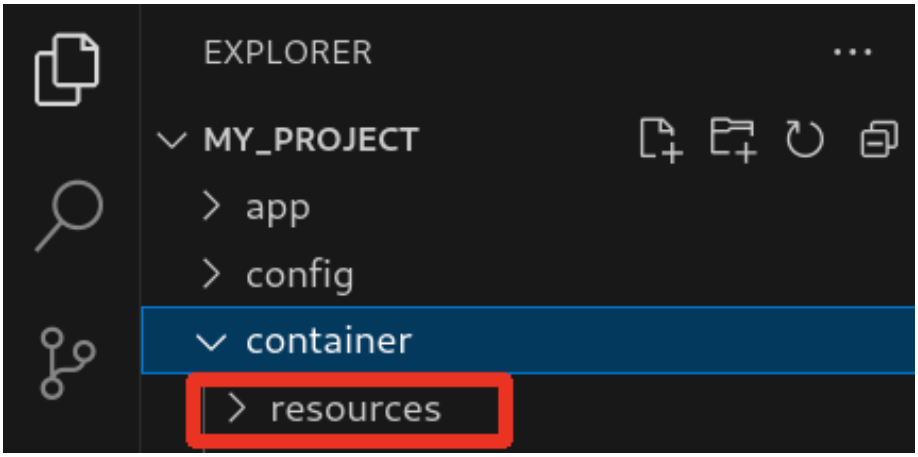


図 7.111 resources ディレクトリ

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

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

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

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

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

7.9.6.2. コンテナ内のファイル一覧の再表示

「図 7.109. コンテナ内のファイル一覧を表示するタブ」 の赤枠で囲われているボタンをクリックすることで、コンテナ内のファイル一覧を再表示します。

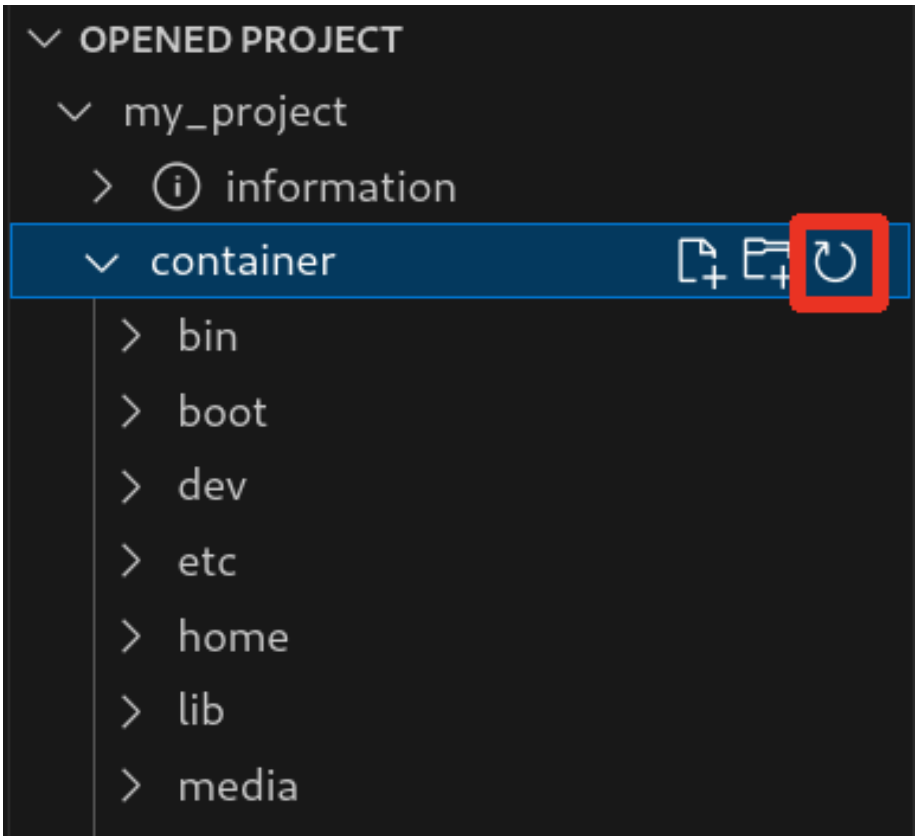


図 7.112 コンテナ内のファイル一覧を再表示するボタン

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

「図 7.113. container/resources 下にファイルを追加するボタン」 の赤枠で囲われている表記のボタンをクリックすることで、プロジェクトディレクトリにある container/resources 下にファイルを追加することが可能です。

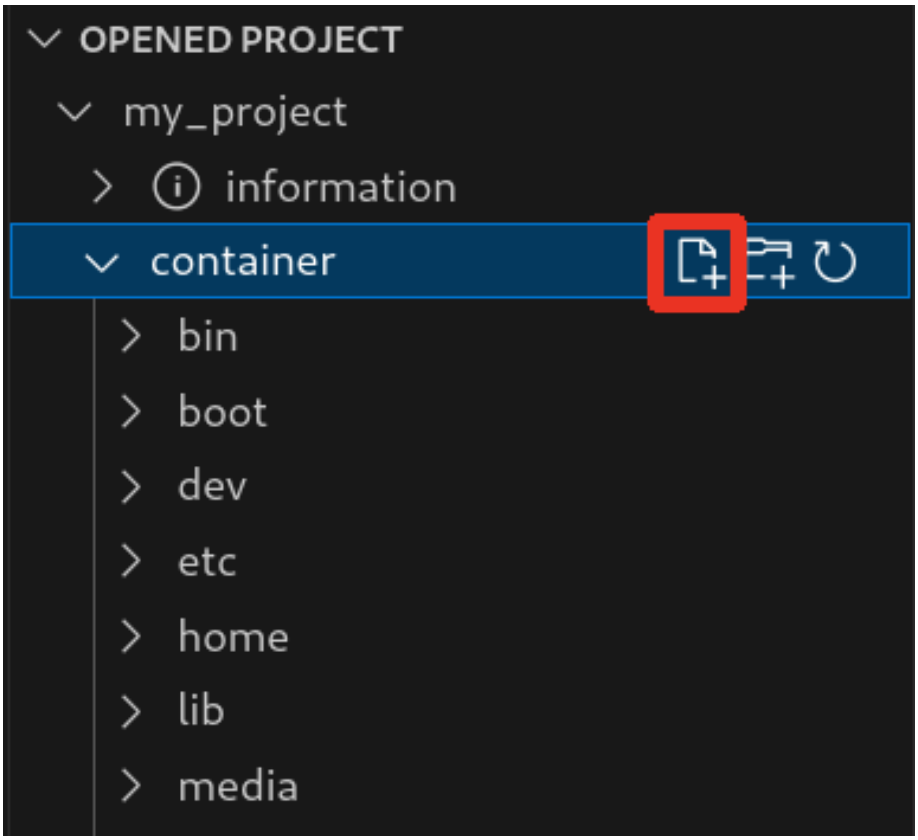


図 7.113 container/resources 下にファイルを追加するボタン

「図 7.114. ファイル名を入力」 に示すように、コマンドパレットが表示されますのでファイル名を入力してください。

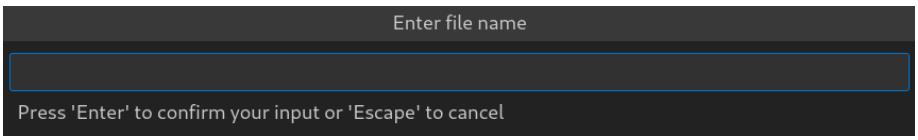


図 7.114 ファイル名を入力

例として、「add_file」というファイル名を入力したとします。

「図 7.115. 追加されたファイルの表示」 に示すように、追加したファイルには「A」というマークが表示されます。

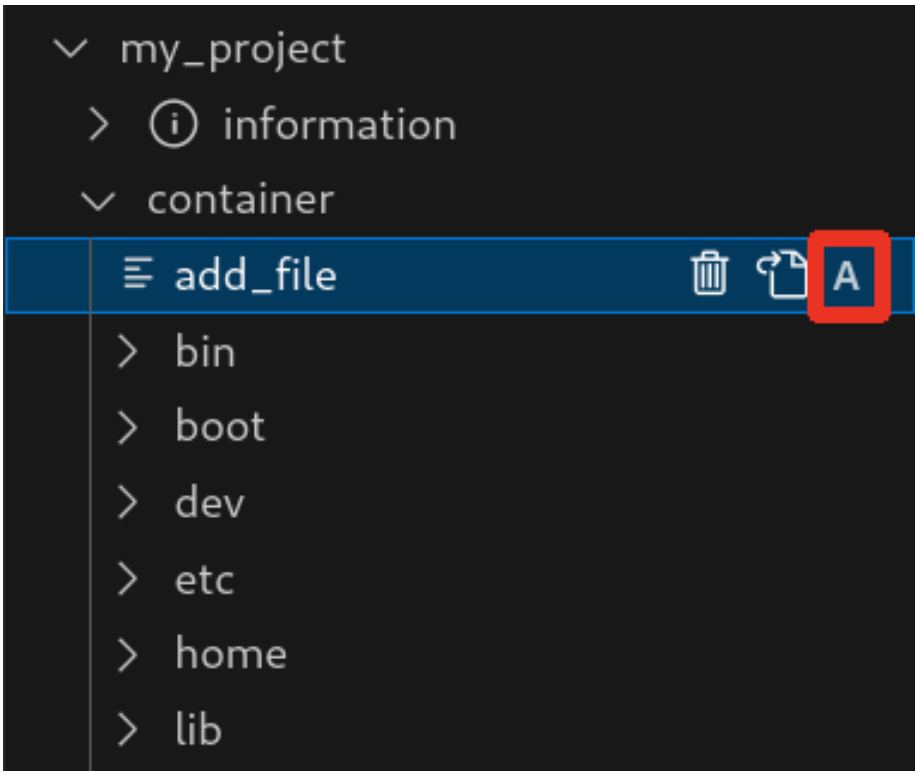


図 7.115 追加されたファイルの表示

また、「図 7.116. container/resources 下にフォルダーを追加するボタン」 の赤枠で囲われている表記のボタンをクリックすることで、ファイルの追加と同様の操作でディレクトリを追加することが可能です。

追加したディレクトリも同様に "A" というマークが表示されます。

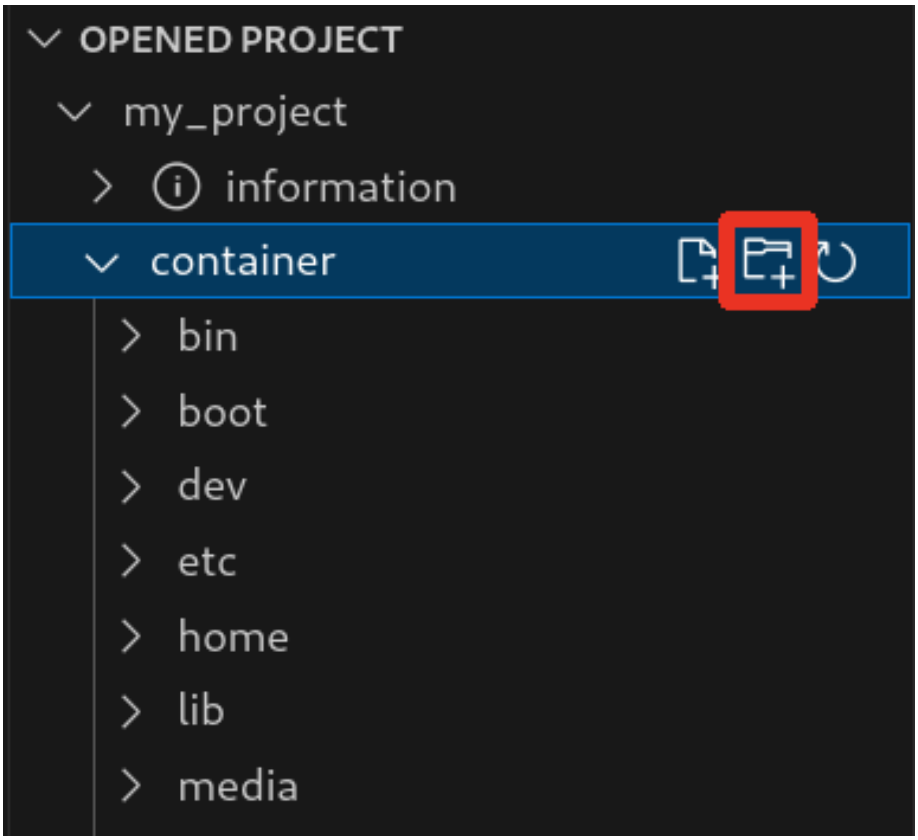


図 7.116 container/resources 下にフォルダーを追加するボタン

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

「図 7.117. container/resources 下にあるファイルを開くボタン」 の赤枠で囲われている表記のボタンをクリックすることで、プロジェクトディレクトリにある container/resources 下のファイルをエディタに表示することができます。

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

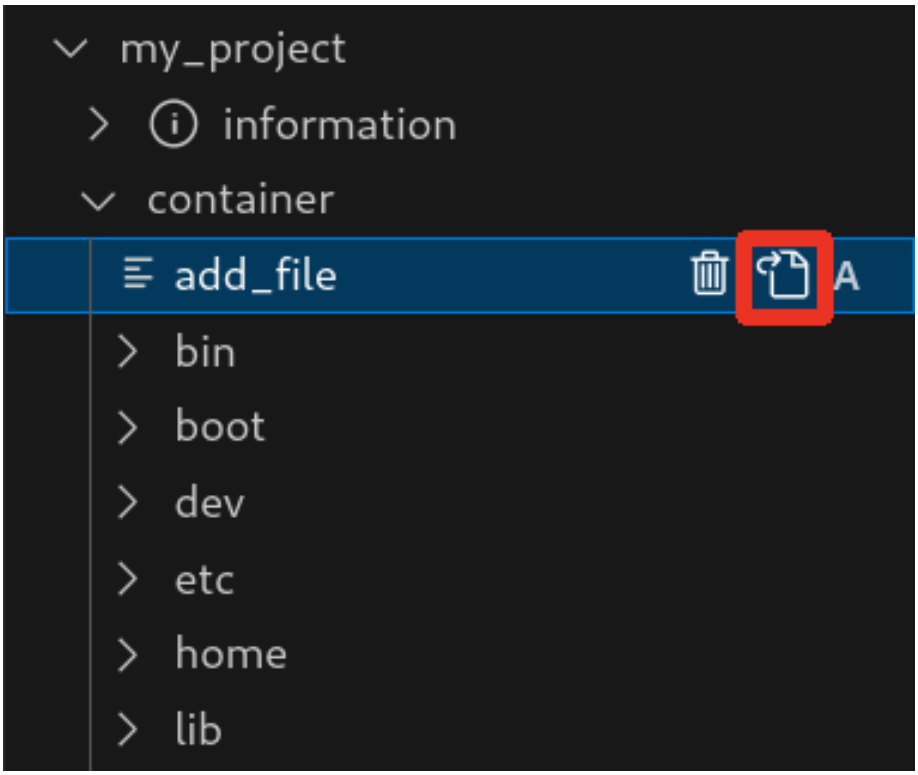


図 7.117 container/resources 下にあるファイルを開くボタン

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

「図 7.117. container/resources 下にあるファイルを開くボタン」 の赤枠で囲われている表記のボタンをクリックすることで、container/resources 下にあるファイルを削除することができます。

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

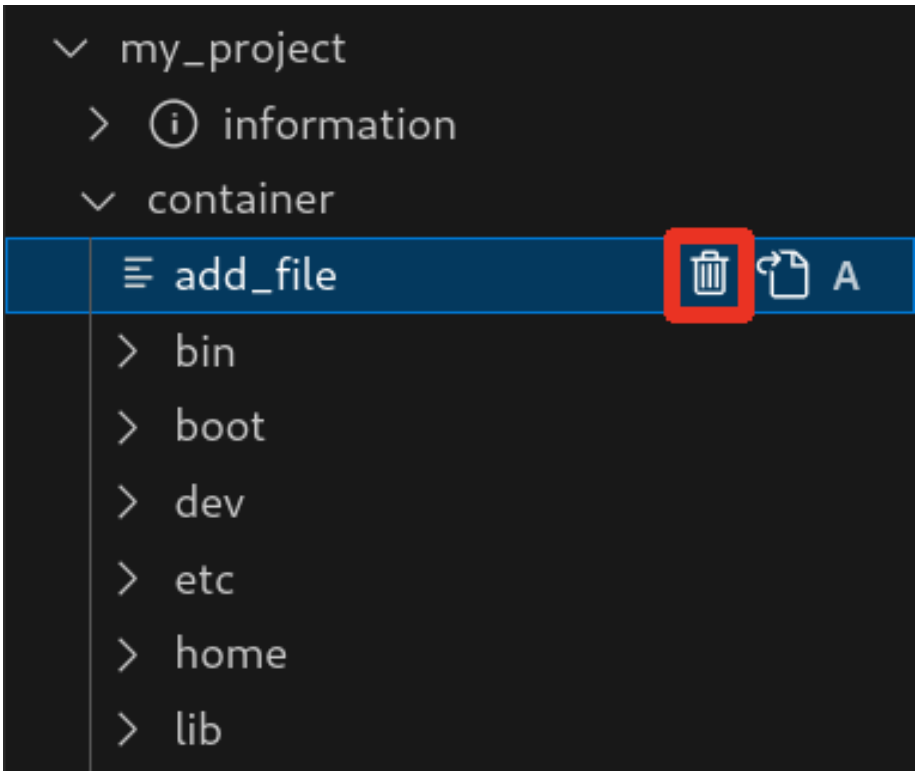


図 7.118 container/resources 下にあるファイルを削除するボタン

ディレクトリも同様に「図 7.117. container/resources 下にあるファイルを開くボタン」 の赤枠で囲われている表記のボタンをクリックすることで削除することができます。

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

「図 7.119. コンテナ内のファイルを container/resources 下に保存するボタン」 の赤枠で囲われている表記のボタンをクリックすることで、コンテナ内にあるファイルをプロジェクトディレクトリにある container/resources 下に保存します。

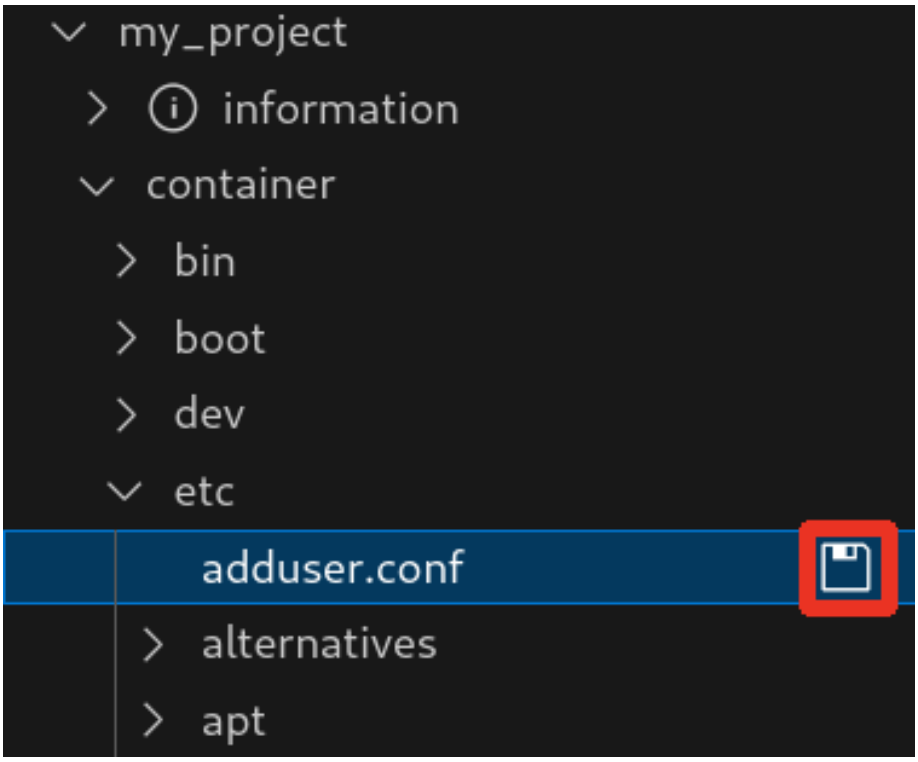


図 7.119 コンテナ内のファイルを container/resources 下に保存するボタン

ファイルが container/resources 下に保存されると、「図 7.120. 編集前のファイルを示すマーク」 に示すように、ファイル名の右側に "U" のマークが表示されます。

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

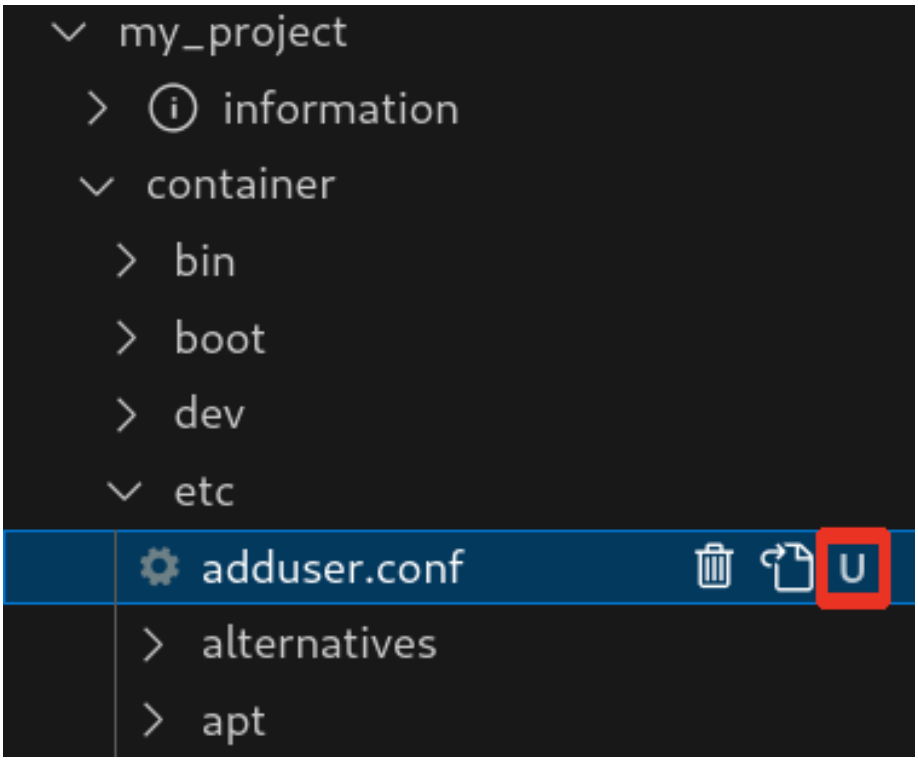


図 7.120 編集前のファイルを示すマーク

container/resources 下にあるファイルを編集して再表示すると、「図 7.121. 編集後のファイルを示すマーク」 に示すように、ファイル名の右側に "M" のマークが表示されます。

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

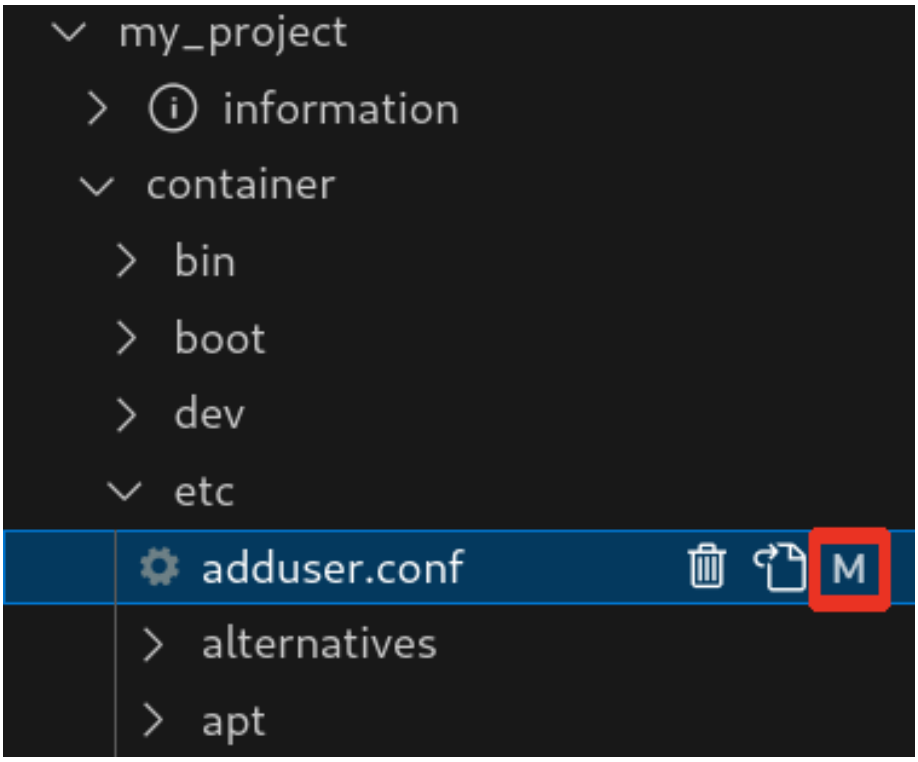


図 7.121 編集後のファイルを示すマーク

7.9.6.7. エラー表示

`container/resources` 下とコンテナ内にあるファイルまたはディレクトリを比較して、同名でかつファイルの種類が異なる場合、「図 7.122. コンテナ内にコピーされないことを示すマーク」 に示すように、ファイル名の右側に "E" のマークが表示されます。

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

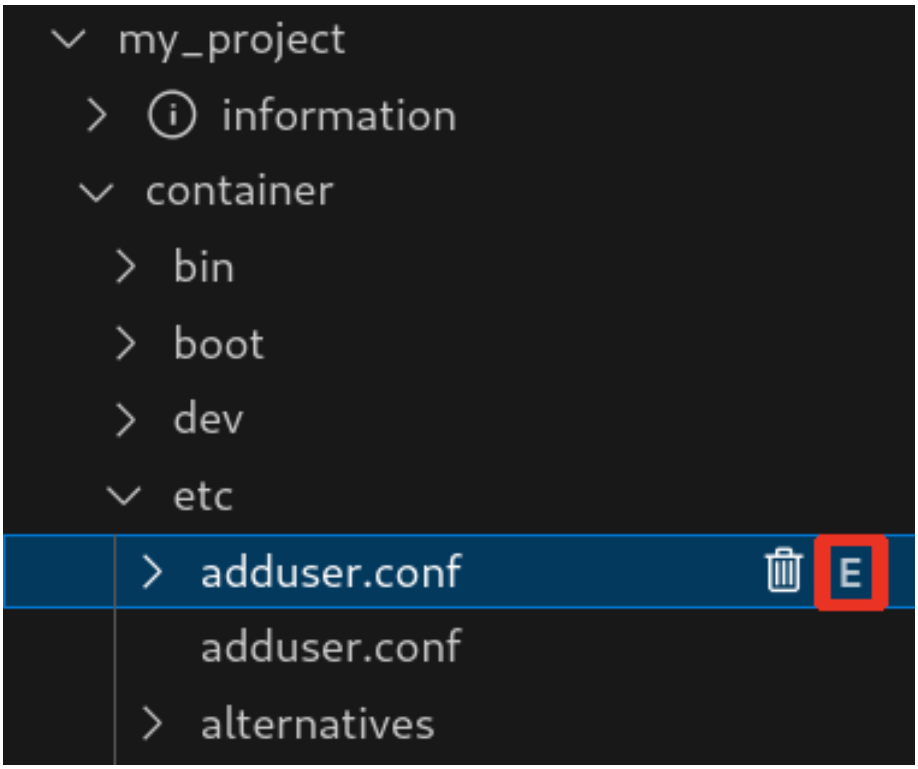


図 7.122 コンテナ内にコピーされないことを示すマーク

7.9.7. Armadillo 上でのセットアップ

7.9.7.1. アプリケーション実行用コンテナイメージのインストール

「7.9.3.4. アプリケーション実行用コンテナイメージの作成」 で作成した `development.swu` を 「7.3.3.6. SWU イメージのインストール」 を参照して Armadillo へインストールしてください。

インストール後に自動で Armadillo が再起動します。

7.9.7.2. ssh 接続に使用する IP アドレスの設定

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

「図 7.123. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする」 の赤枠で囲われているボタンをクリックすることで、ローカルネットワーク上で ABOS Web が実行されている Armadillo をスキャンすることができます。

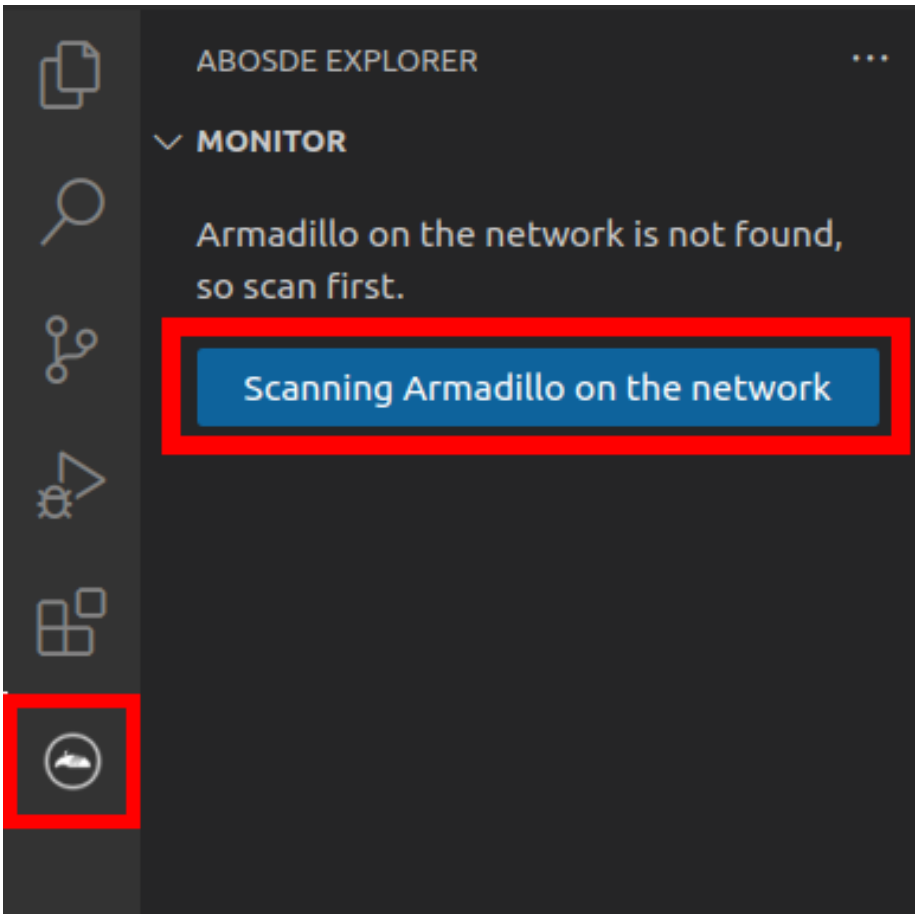


図 7.123 ABOSDE で ローカルネットワーク上の Armadillo をスキャンする

「図 7.124. ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する」 の赤枠で囲われているマークをクリックすることで、指定した Armadillo の IP アドレスを ssh 接続に使用する IP アドレスに設定することができます。

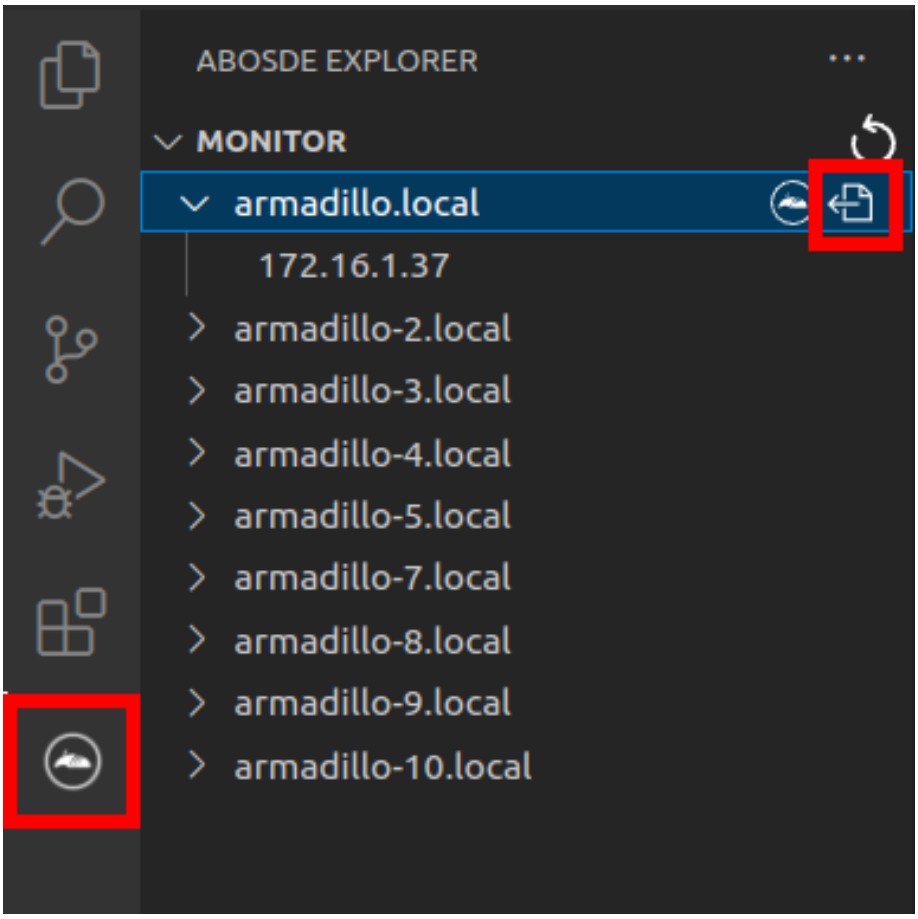


図 7.124 ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する

「図 7.125. ABOSDE に表示されている Armadillo を更新する」 の赤枠で囲われているマークをクリックすることで、ABOSDE に表示されている Armadillo を更新することができます。

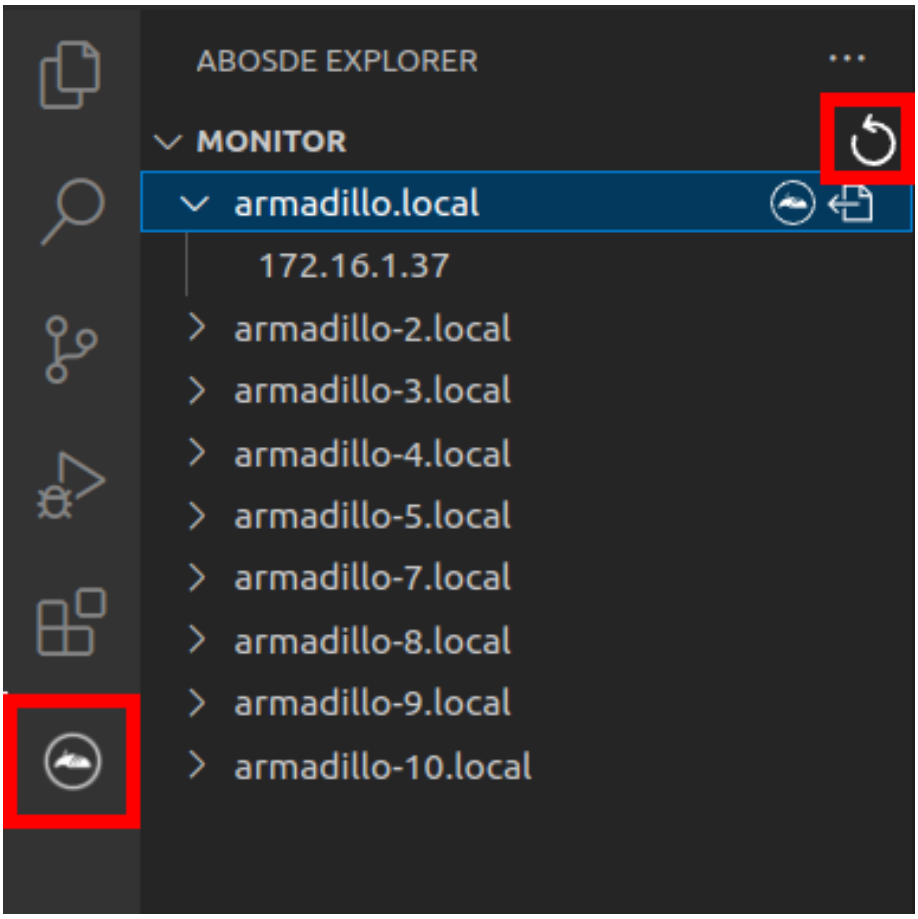


図 7.125 ABOSDE に表示されている Armadillo を更新する

ATDE のネットワークを NAT に設定している場合や、ABOS Web を起動していない場合等、ABOSDE のリストに Armadillo が表示されない場合は、プロジェクトディレクトリに入っている config/ssh_config ファイルを編集して IP アドレスを書き換えてください。

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

図 7.126 ssh_config を編集する

❶ Armadillo の IP アドレスに置き換えてください。

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

7.9.7.3. アプリケーションの実行

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

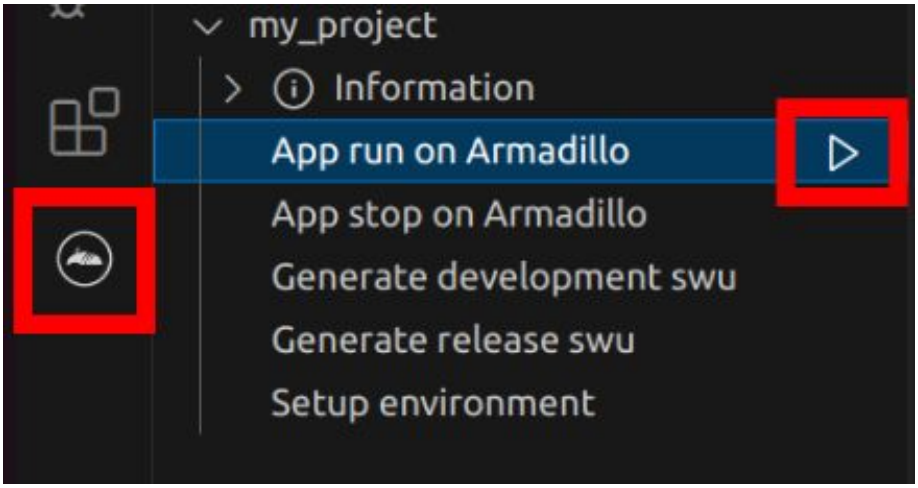


図 7.127 Armadillo 上でアプリケーションを実行する

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

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

図 7.128 実行時に表示されるメッセージ

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

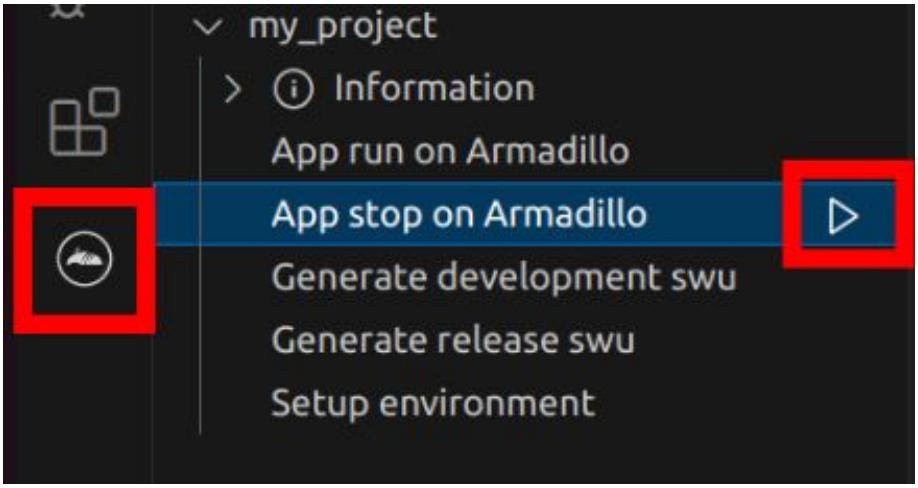


図 7.129 アプリケーションを終了する

7.9.8. SBOM 生成に関する設定

SWU イメージ作成時に、同時に SBOM を生成することができます。詳細は 「7.11. SBOM 生成に関わる設定を行う」 を参照してください。

7.9.9. リリース版のビルド

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

VS Code の左ペインの [my_project] から [Generate release swu] を実行すると、リリース版のアプリケーションを含んだ SWU イメージが作成されます。事前に 「9.3.1. SWU イメージの作成」 を参照して SWU の初期設定を行ってください。

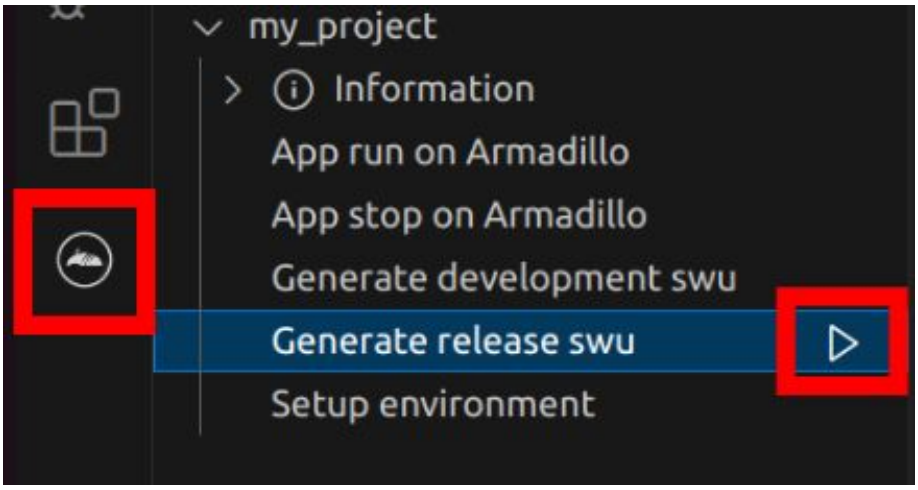


図 7.130 リリース版をビルドする

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

7.9.10. 製品への書き込み

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

この SWU イメージを 「7.3.3.6. SWU イメージのインストール」 を参照して Armadillo へインストールすると、Armadillo 起動時にアプリケーションも自動起動します。

7.9.11. Armadillo 上のコンテナイメージの削除

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

7.10. C 言語によるアプリケーションの開発

ここでは C 言語によるアプリケーション開発の方法を紹介します。

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

・既存の C 言語によって開発されたアプリケーションを Armadillo で動作させたい

・C 言語でないと実現できないアプリケーションを開発したい

上記に当てはまらず、開発するアプリケーションがシェルスクリプトまたは Python で実現可能であるならば、「7.9. CUI アプリケーションの開発」を参照してください。

7.10.1. C 言語によるアプリケーション開発の流れ

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

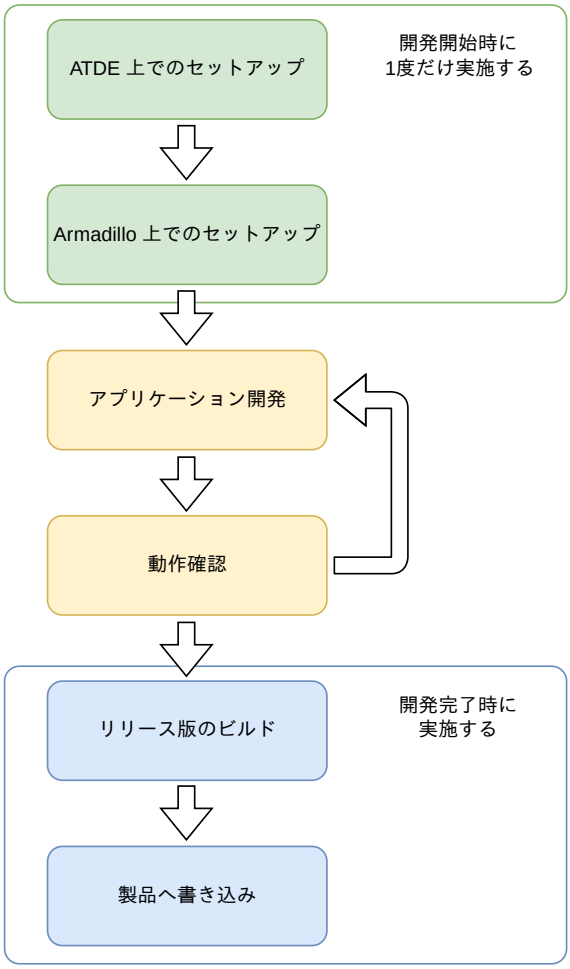


図 7.131 C 言語によるアプリケーション開発の流れ

7.10.2. 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 「7.1. 開発の準備」 を参照して ATDE 及び、 VS Code のセットアップを完了してください。

7.10.2.1. プロジェクトの作成

VS Code の左ペインの [A9E] から [C New Project] を実行し、表示されるディレクトリ選択画面からプロジェクトを保存するディレクトリを選択してください。実行するためには右に表示されている三角形ボタンを押してください。保存先を選択すると、プロジェクト名を入力するダイアログが表示され

るので、任意のプロジェクト名を入力してエンターキーを押してください。この操作により、選択した保存先に、入力したプロジェクト名と同名のディレクトリが作成されます。

また、ここでは次のように設定しています。

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

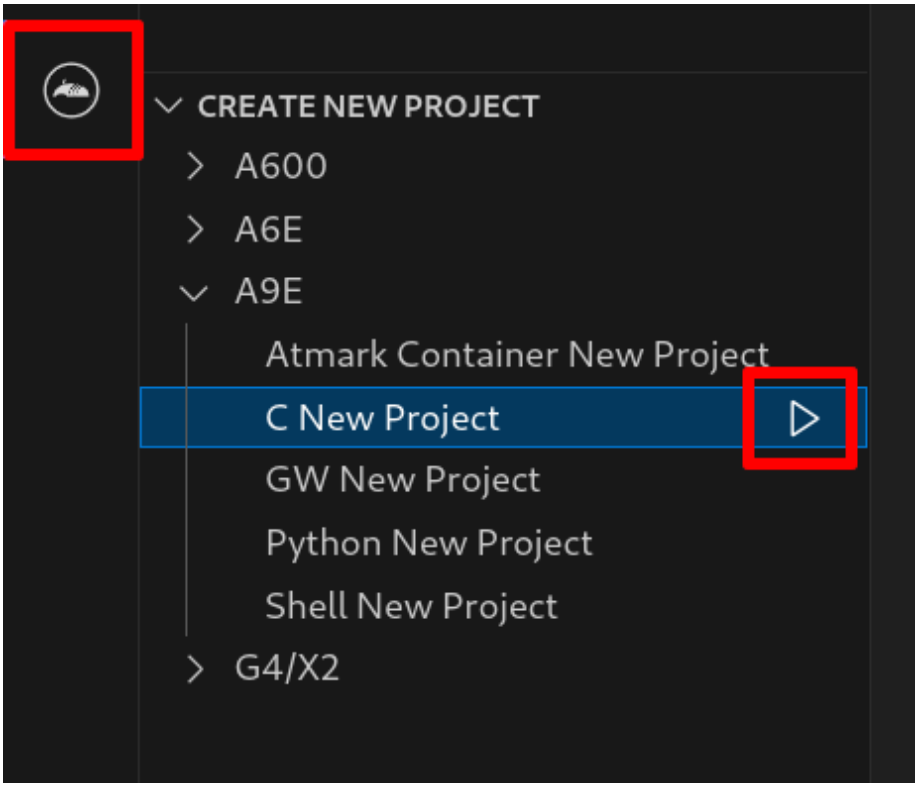


図 7.132 プロジェクトを作成する

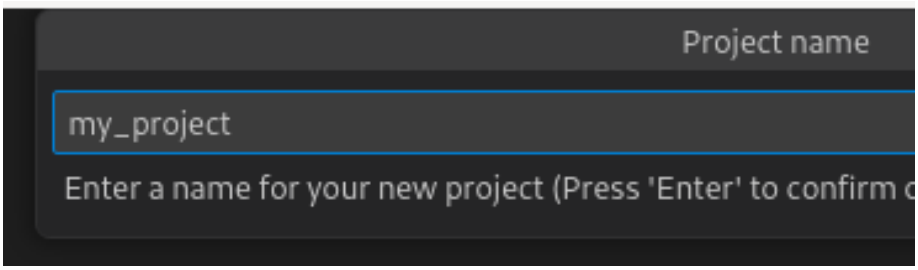


図 7.133 プロジェクト名を入力する

7.10.3. アプリケーション開発

7.10.3.1. VS Code の起動

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

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

図 7.134 VS Code で my_project を起動する

7.10.3.2. ディレクトリ構成

プロジェクトには下記のディレクトリがあります。

- **app** : 各ディレクトリの説明は以下の通りです。
  - **src** : アプリケーションのソースファイル（拡張子が .c ）と Makefile を配置してください。
  - **build** : ここに配置した実行ファイルが Armadillo 上で実行されます。
  - **lib** : 共有ライブラリの検索パスとしてこのディレクトリを指定しているので、ここに共有ライブラリ（拡張子が .so ）を配置することができます。
- **config** : 設定に関わるファイルが含まれるディレクトリです。
  - **app.conf** : コンテナのコンフィグです。記載内容については 「10.8.3. コンテナ起動設定ファイルを作成する」 を参照してください。
  - **app.desc** : SWU イメージを生成するための .desc ファイルです。記載内容については 「10.3. mkswu の .desc ファイルを編集する」 を参照してください。
  - **ssh_config** : Armadillo への ssh 接続に使用します。「7.10.7.2. ssh 接続に使用する IP アドレスの設定」 を参照してください。
- **container** : スクリプトを実行するコンテナの設定ファイルが含まれるディレクトリです。
  - **packages.txt**: このファイルに記載されているパッケージがインストールされます。
  - **Dockerfile**: 直接編集することも可能です。

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

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

7.10.3.3. 初期設定

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

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

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

図 7.135 初期設定を行う

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

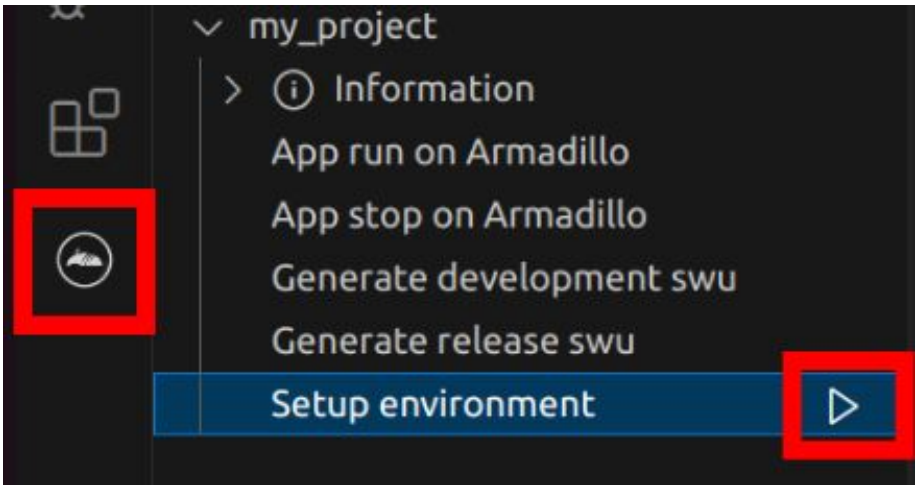


図 7.136 VS Code で初期設定を行う

選択すると、 VS Code の下部に以下のようなターミナルが表示されます。

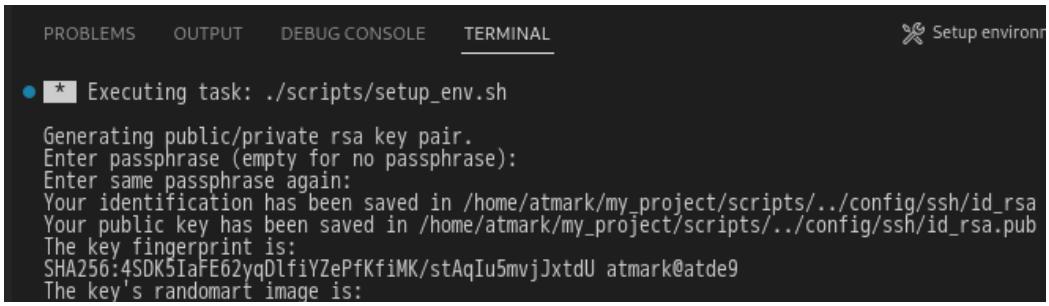


図 7.137 VS Code のターミナル

このターミナル上で以下のように入力してください。

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

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

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

図 7.138 SSH 用の鍵を生成する

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

❷ 1 でパスフレーズを設定した場合は、確認のため再度入力してください。

❸ ここで何か任意のキーを押すとターミナルが閉じます。

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

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

7.10.3.4. packages.txt の書き方

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

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

「図 7.139. C 言語による開発における packages.txt の書き方」に C 言語による開発の場合における packages.txt の書き方の例を示します。ここでは、パッケージ名を package_A 、package_B 、package_C としています。

```
package_A
package_B

[build]    ❶
package_C
```

図 7.139 C 言語による開発における packages.txt の書き方

❶ このラベル以降のパッケージはビルド時のみに使用されます。

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

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

7.10.3.5. ABOSDE での開発における制約

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

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

7.10.3.6. 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 「9.3.1. SWU イメージの作成」 を参照して SWU の初期設定を行ってください。

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

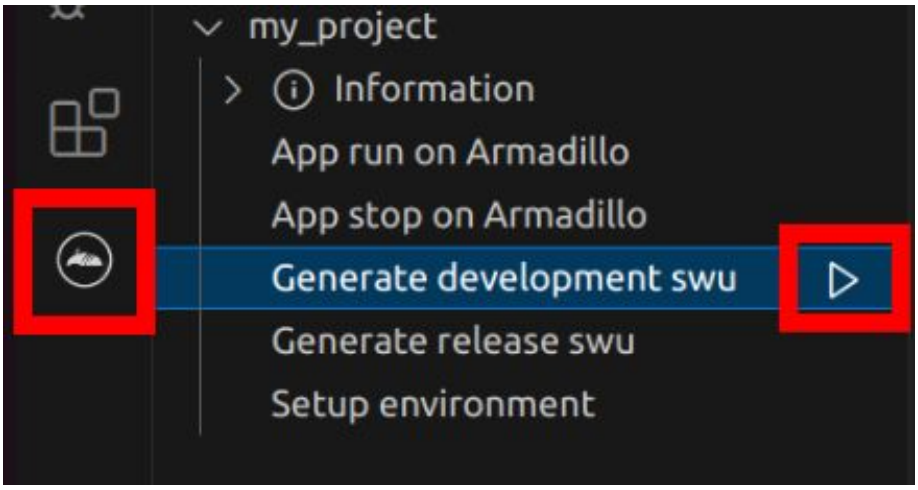


図 7.140 VS Code でコンテナイメージの作成を行う

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

```
コンテナイメージを ./swu/my_project.tar に保存しました。
./swu/app.desc のバージョンを 1 から 2 に変更しました。
./development.swu を作成しました。
次は Armadillo に ./development.swu をインストールしてください。
 *  Terminal will be reused by tasks, press any key to close it.
```

図 7.141 コンテナイメージの作成完了

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

7.10.4. コンテナのディストリビューション

使用するコンテナのディストリビューションは以下のとおりです。

ディストリビューション ・ debian:bullseye-slim

7.10.5. コンテナ内のファイル一覧表示

「図 7.142. コンテナ内のファイル一覧を表示するタブ」 の赤枠で囲われているタブをクリックすることで、development.swu または「7.10.9. リリース版のビルド」で作成される release.swu に含まれるコンテナ内のファイルおよびディレクトリを表示します。

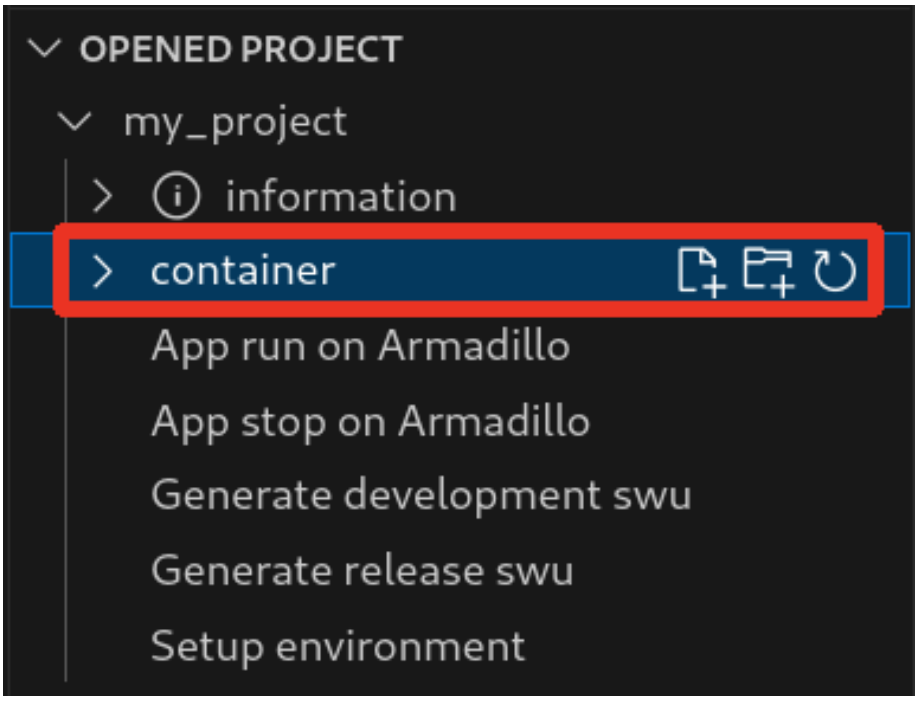


図 7.142 コンテナ内のファイル一覧を表示するタブ

クリック後の表示例を 「図 7.143. コンテナ内のファイル一覧の例」 に示します。

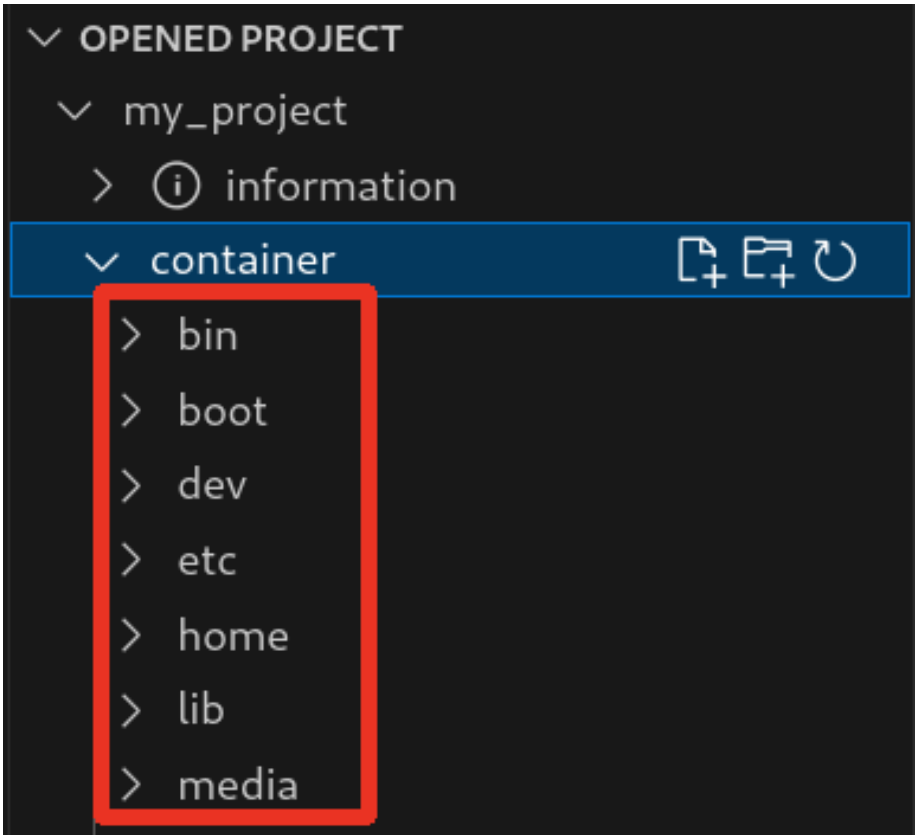


図 7.143 コンテナ内のファイル一覧の例

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

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

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

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

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

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

7.10.5.1. resources ディレクトリについて

「図 7.144. resources ディレクトリ」に示すように ATDE 上のプロジェクトディレクトリには container/resources ディレクトリがあります。

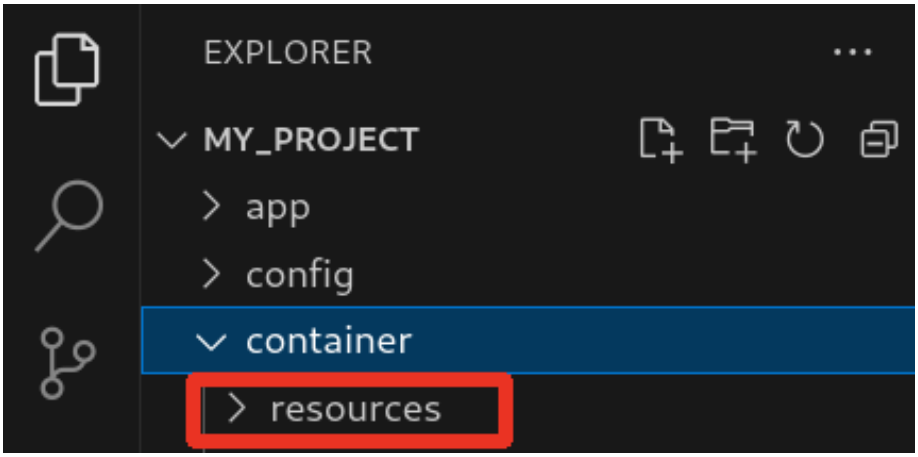


図 7.144 resources ディレクトリ

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

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

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

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

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

7.10.5.2. コンテナ内のファイル一覧の再表示

「図 7.142. コンテナ内のファイル一覧を表示するタブ」 の赤枠で囲われているボタンをクリックすることで、コンテナ内のファイル一覧を再表示します。

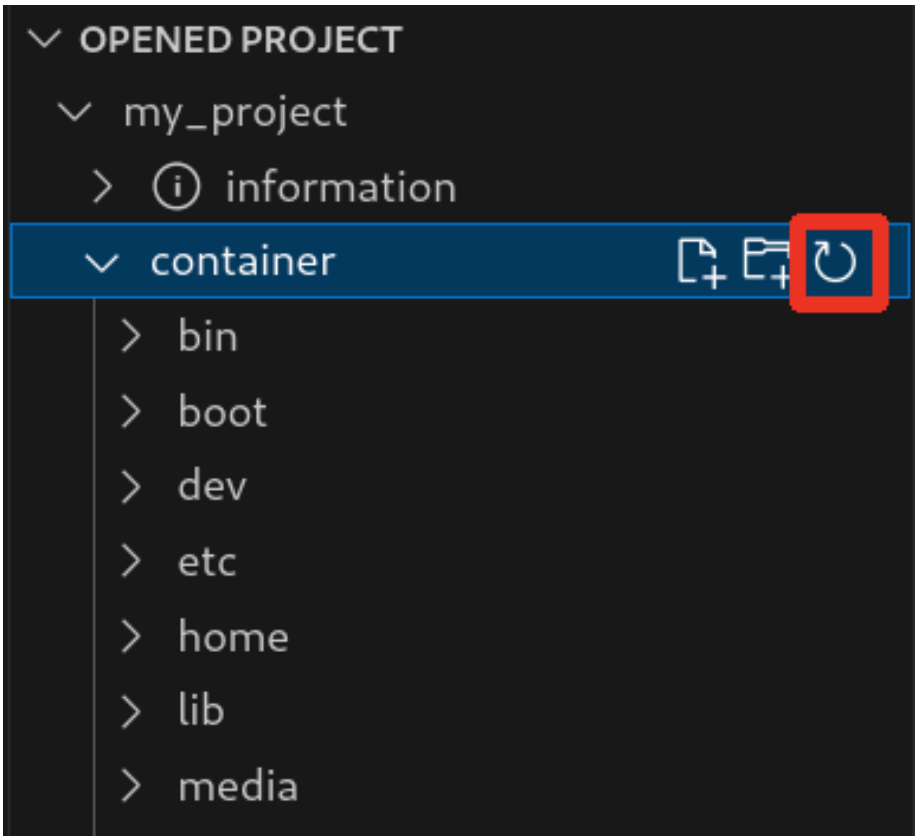


図 7.145 コンテナ内のファイル一覧を再表示するボタン

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

「図 7.146. container/resources 下にファイルを追加するボタン」 の赤枠で囲われている表記のボタンをクリックすることで、プロジェクトディレクトリにある container/resources 下にファイルを追加することが可能です。

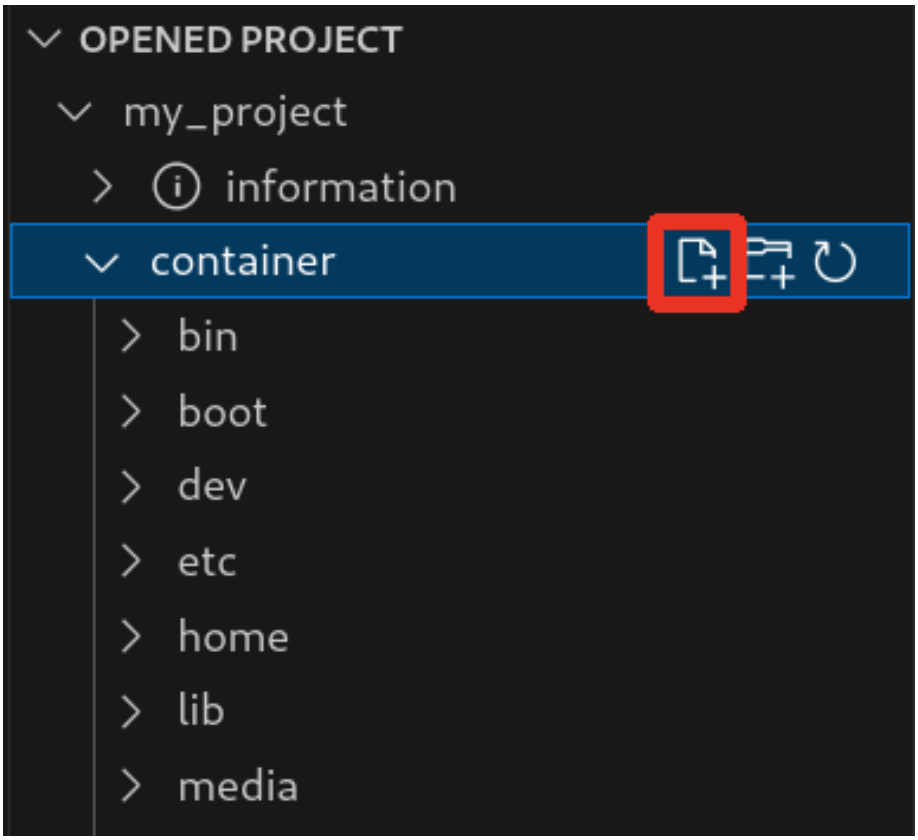


図 7.146 container/resources 下にファイルを追加するボタン

「図 7.147. ファイル名を入力」 に示すように、コマンドパレットが表示されますのでファイル名を入力してください。

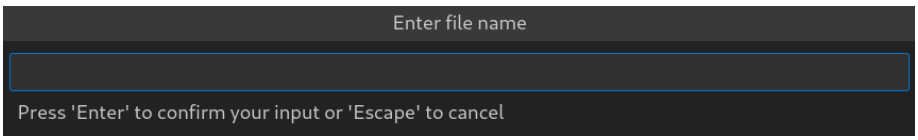


図 7.147 ファイル名を入力

例として、「add_file」というファイル名を入力したとします。「図 7.148. 追加されたファイルの表示」に示すように、追加したファイルには「A」というマークが表示されます。

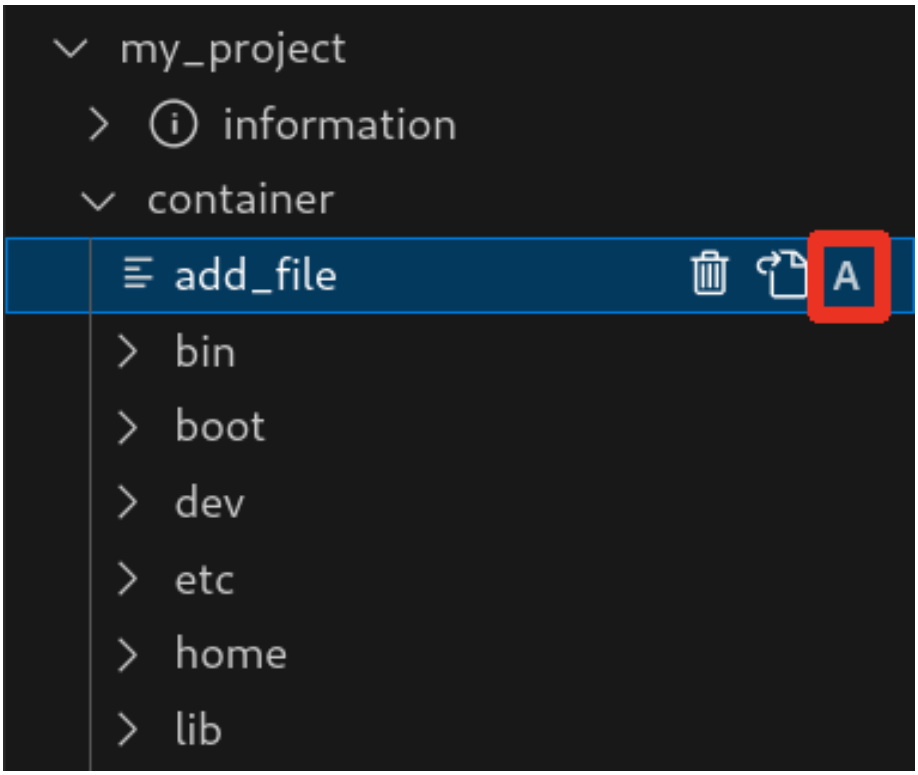


図 7.148 追加されたファイルの表示

また、「図 7.149. container/resources 下にフォルダーを追加するボタン」 の赤枠で囲われている表記のボタンをクリックすることで、ファイルの追加と同様の操作でディレクトリを追加することが可能です。

追加したディレクトリも同様に "A" というマークが表示されます。

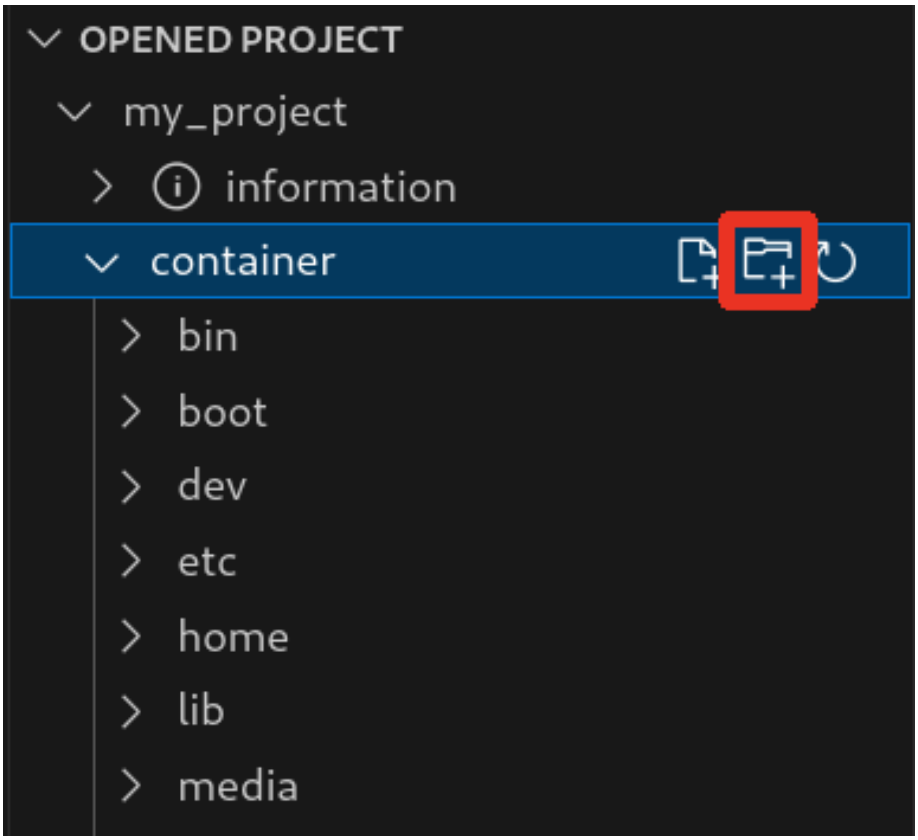


図 7.149 container/resources 下にフォルダーを追加するボタン

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

「図 7.150. container/resources 下にあるファイルを開くボタン」 の赤枠で囲われている表記のボタンをクリックすることで、プロジェクトディレクトリにある container/resources 下のファイルをエディタに表示することができます。

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

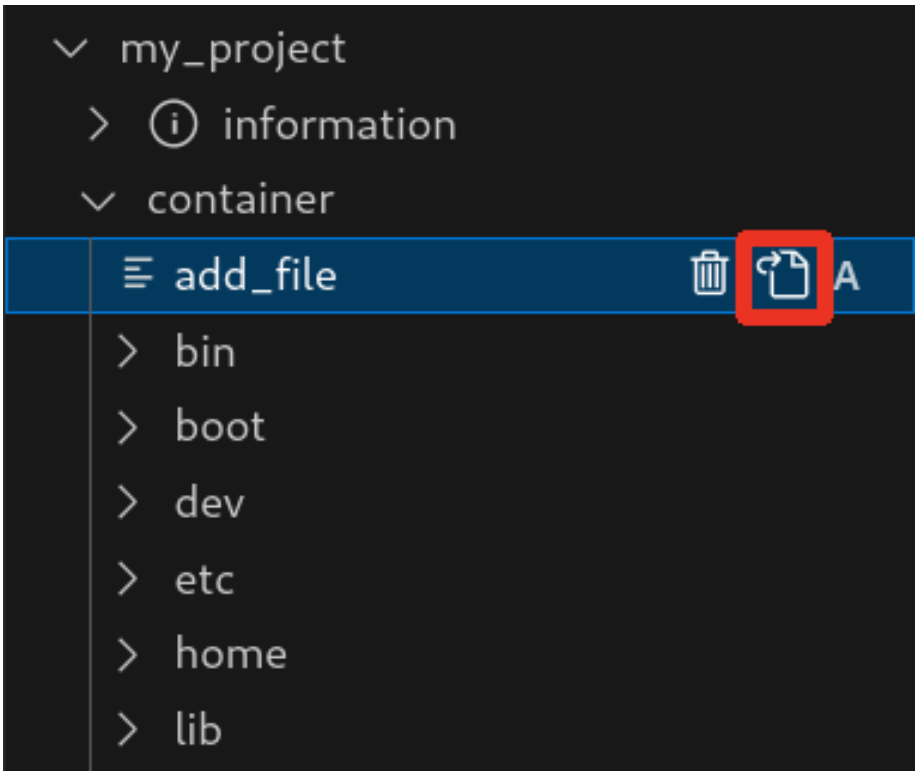


図 7.150 container/resources 下にあるファイルを開くボタン

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

「図 7.150. container/resources 下にあるファイルを開くボタン」 の赤枠で囲われている表記のボタンをクリックすることで、container/resources 下にあるファイルを削除することができます。

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

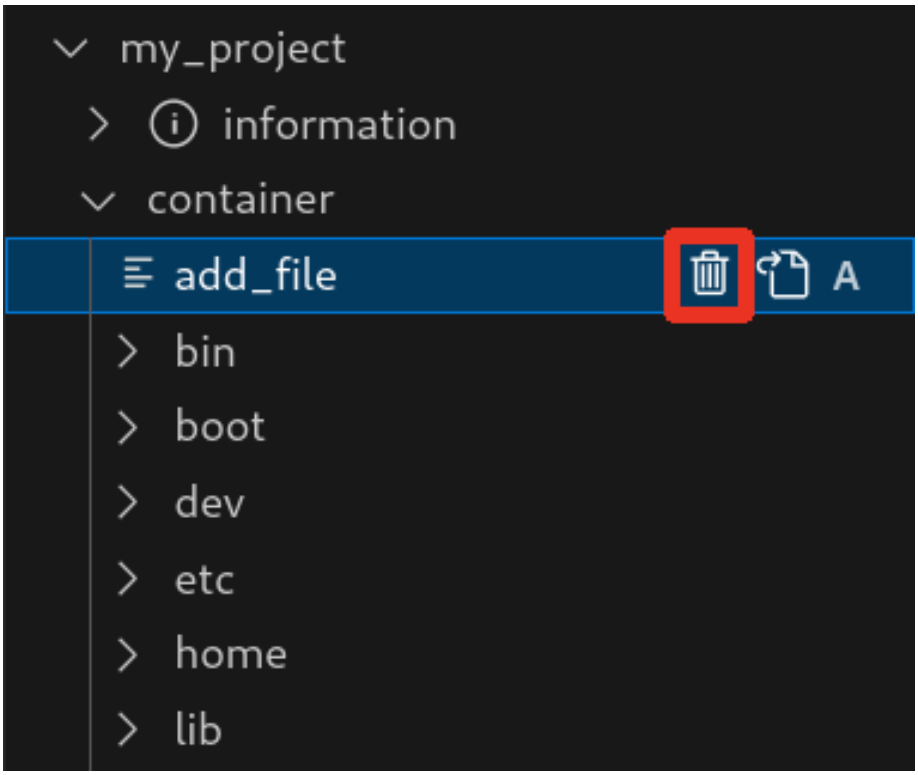


図 7.151 container/resources 下にあるファイルを削除するボタン

ディレクトリも同様に「図 7.150. container/resources 下にあるファイルを開くボタン」 の赤枠で囲われている表記のボタンをクリックすることで削除することができます。

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

「図 7.152. コンテナ内のファイルを container/resources 下に保存するボタン」 の赤枠で囲われている表記のボタンをクリックすることで、コンテナ内にあるファイルをプロジェクトディレクトリにある container/resources 下に保存します。

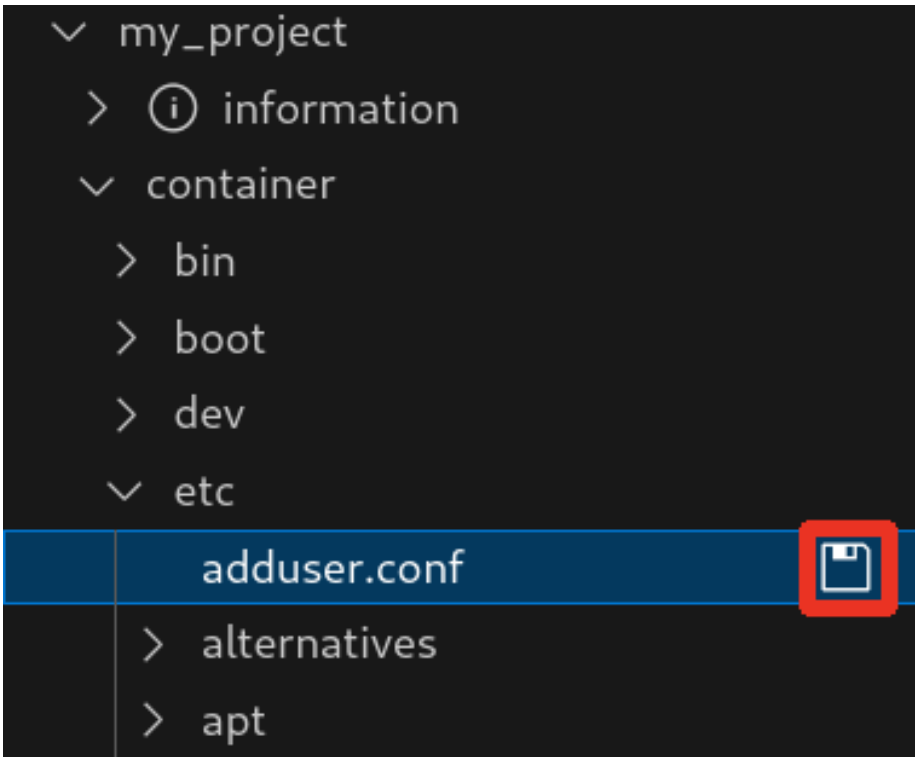


図 7.152 コンテナ内のファイルを container/resources 下に保存するボタン

ファイルが container/resources 下に保存されると、「図 7.153. 編集前のファイルを示すマーク」 に示すように、ファイル名の右側に "U" のマークが表示されます。

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

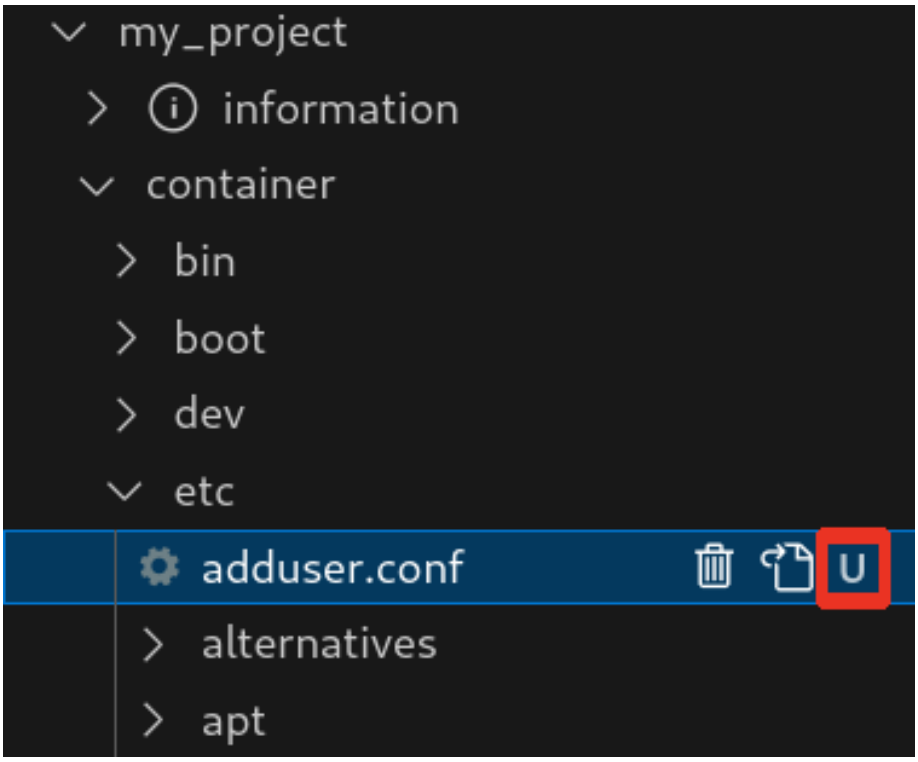


図 7.153 編集前のファイルを示すマーク

container/resources 下にあるファイルを編集して再表示すると、「図 7.154. 編集後のファイルを示すマーク」 に示すように、ファイル名の右側に "M" のマークが表示されます。

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

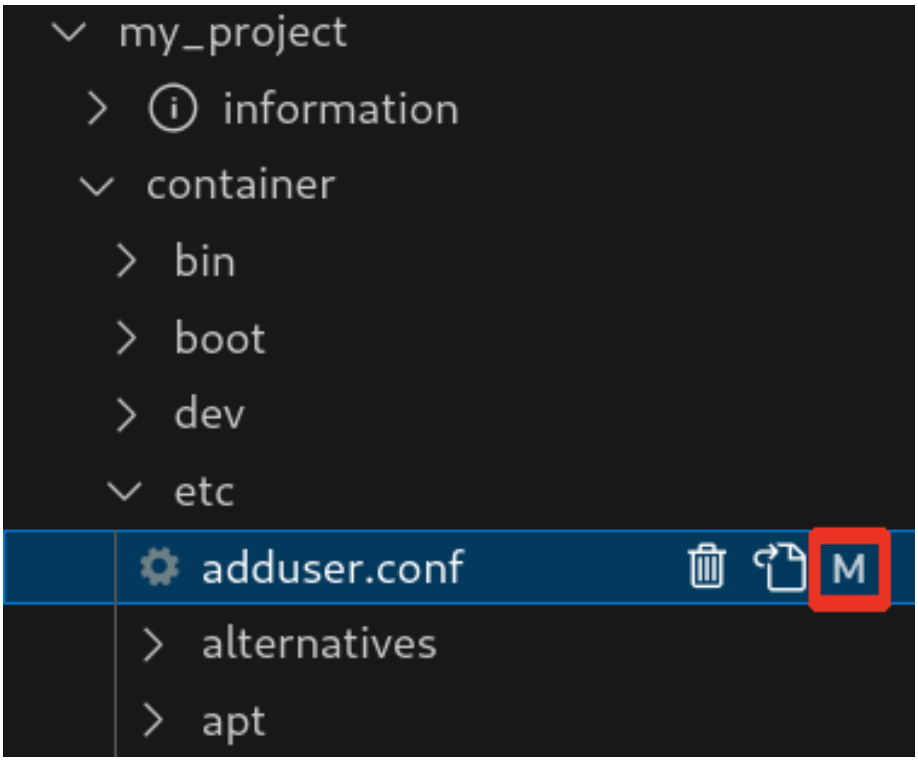


図 7.154 編集後のファイルを示すマーク

7.10.5.7. エラー表示

`container/resources` 下とコンテナ内にあるファイルまたはディレクトリを比較して、同名でかつファイルの種類が異なる場合、「図 7.155. コンテナ内にコピーされないことを示すマーク」 に示すように、ファイル名の右側に "E" のマークが表示されます。

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

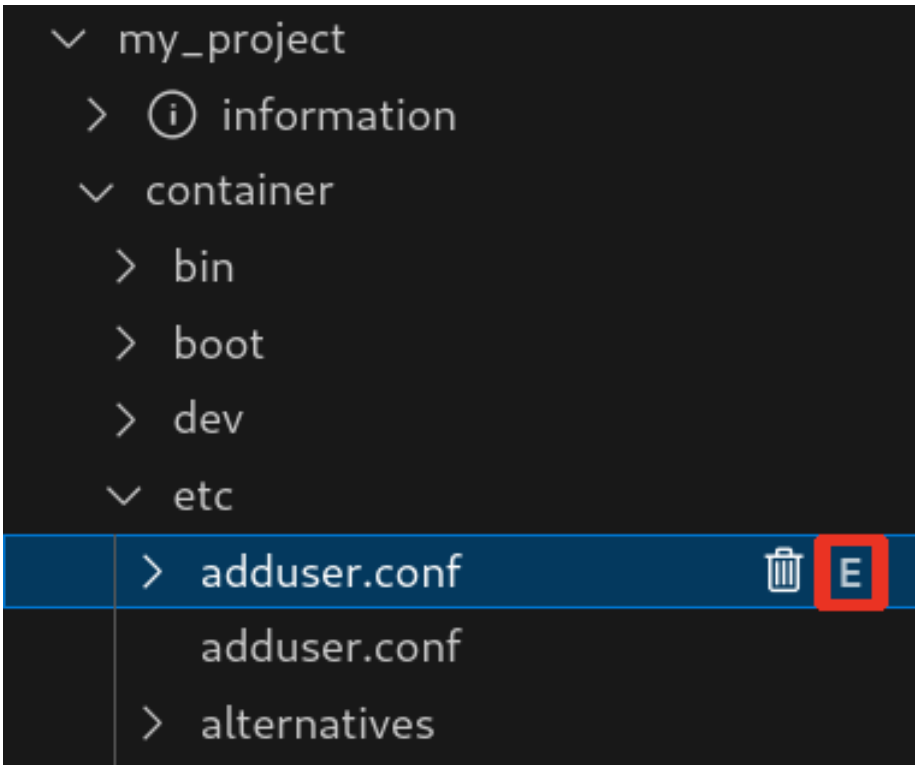


図 7.155 コンテナ内にコピーされないことを示すマーク

7.10.6. Armadillo に転送するディレクトリ及びファイル

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

Armadillo に転送するディレクトリ及びファイル

- my_project/app/build
- my_project/app/lib

7.10.7. Armadillo 上でのセットアップ

7.10.7.1. アプリケーション実行用コンテナイメージのインストール

「7.10.3.6. アプリケーション実行用コンテナイメージの作成」 で作成した `development.swu` を「7.3.3.6. SWU イメージのインストール」 を参照して Armadillo へインストールしてください。

インストール後に自動で Armadillo が再起動します。

7.10.7.2. ssh 接続に使用する IP アドレスの設定

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

「図 7.156. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする」 の赤枠で囲われているボタンをクリックすることで、ローカルネットワーク上で ABOS Web が実行されている Armadillo をスキャンすることができます。

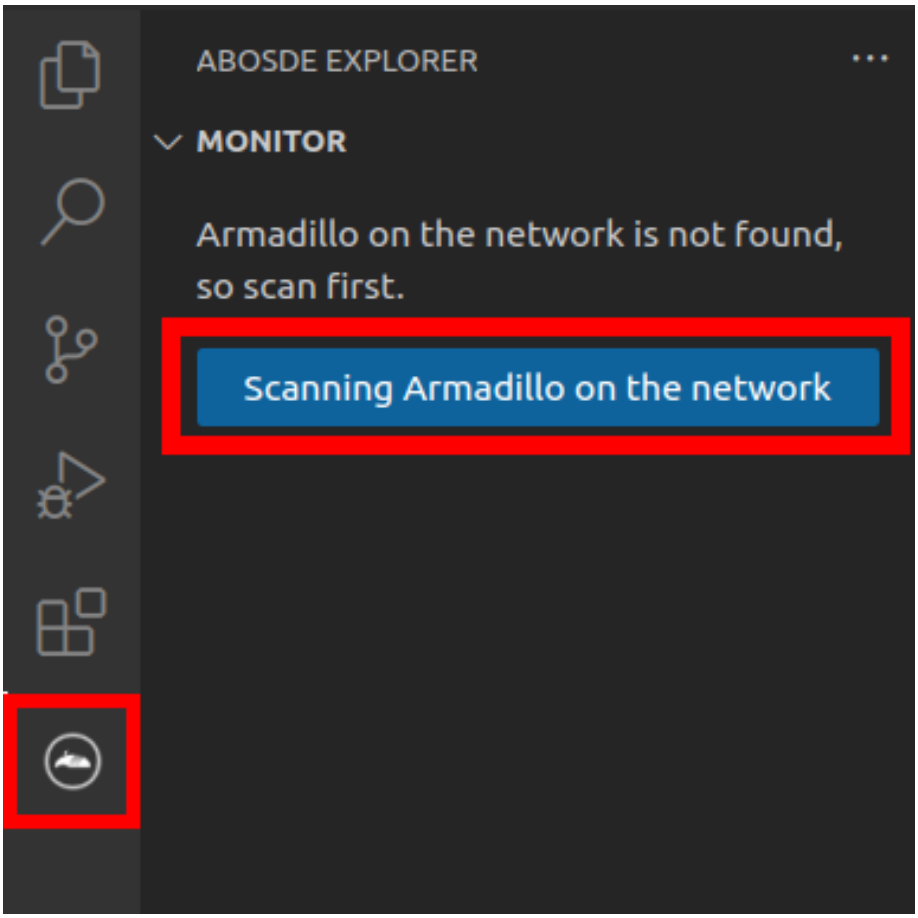


図 7.156 ABOSDE で ローカルネットワーク上の Armadillo をスキャンする

「図 7.157. ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する」 の赤枠で囲われているマークをクリックすることで、指定した Armadillo の IP アドレスを ssh 接続に使用する IP アドレスに設定することができます。

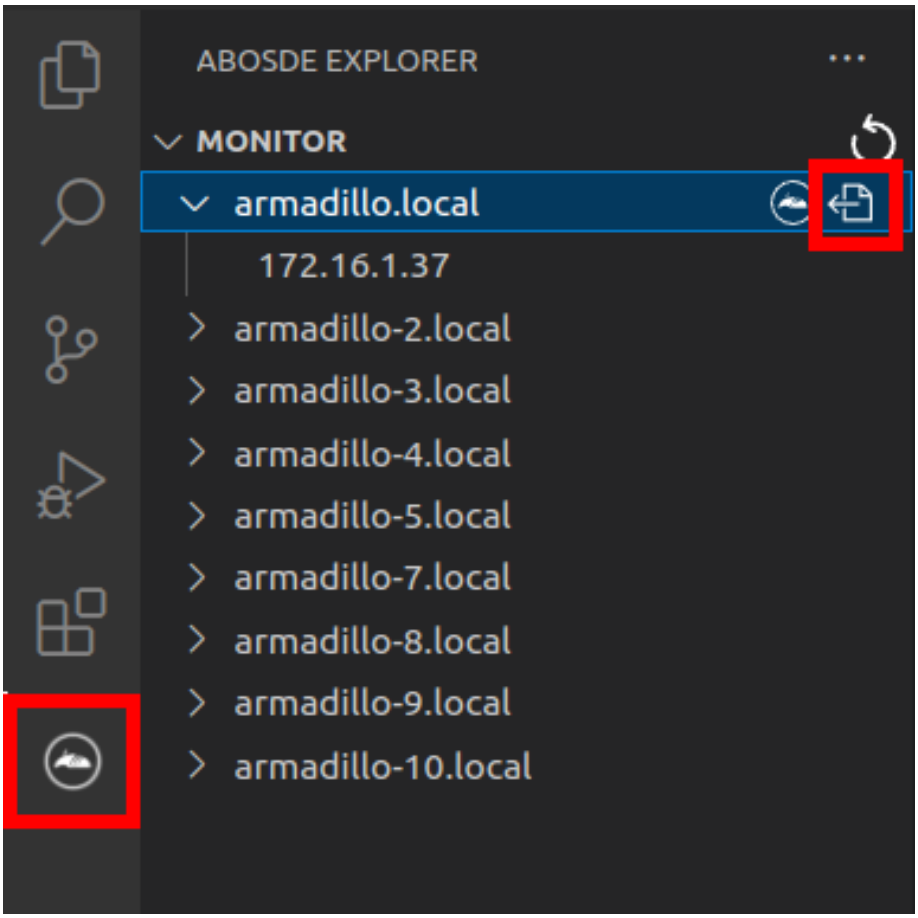


図 7.157 ABOSDE を使用して ssh 接続に使用する IP アドレスを設定する

「図 7.158. ABOSDE に表示されている Armadillo を更新する」 の赤枠で囲われているマークをクリックすることで、ABOSDE に表示されている Armadillo を更新することができます。

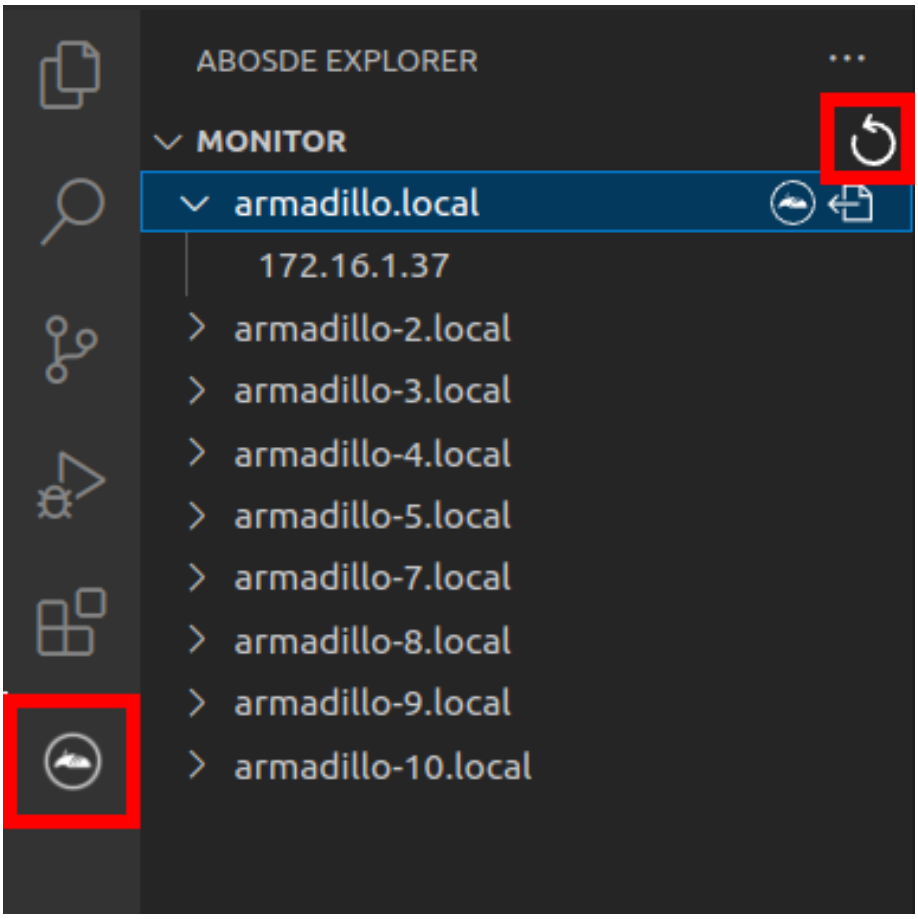


図 7.158 ABOSDE に表示されている Armadillo を更新する

ATDE のネットワークを NAT に設定している場合や、ABOS Web を起動していない場合等、ABOSDE のリストに Armadillo が表示されない場合は、プロジェクトディレクトリに入っている config/ssh_config ファイルを編集して IP アドレスを書き換えてください。

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

図 7.159 ssh_config を編集する

❶ Armadillo の IP アドレスに置き換えてください。

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

7.10.7.3. アプリケーションの実行

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

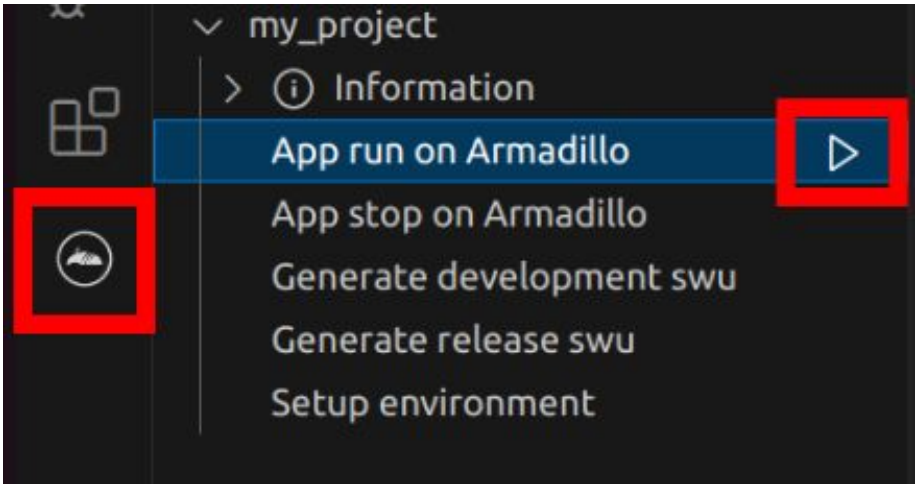


図 7.160 Armadillo 上でアプリケーションを実行する

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

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

図 7.161 実行時に表示されるメッセージ

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

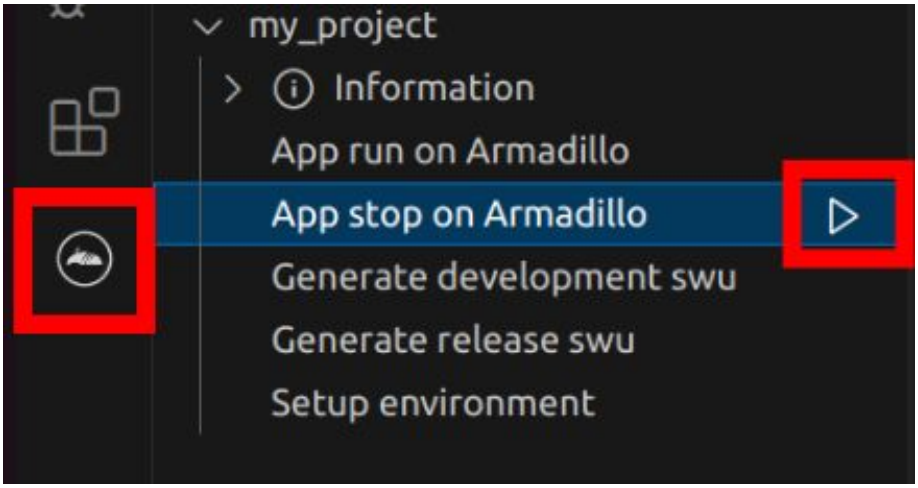


図 7.162 アプリケーションを終了する

7.10.8. SBOM 生成に関する設定

SWU イメージ作成時に、同時に SBOM を生成することができます。詳細は 「7.11. SBOM 生成に関わる設定を行う」 を参照してください。

7.10.9. リリース版のビルド

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

VS Code の左ペインの [my_project] から [Generate release swu] を実行すると、リリース版のアプリケーションを含んだ SWU イメージが作成されます。事前に 「9.3.1. SWU イメージの作成」 を参照して SWU の初期設定を行ってください。

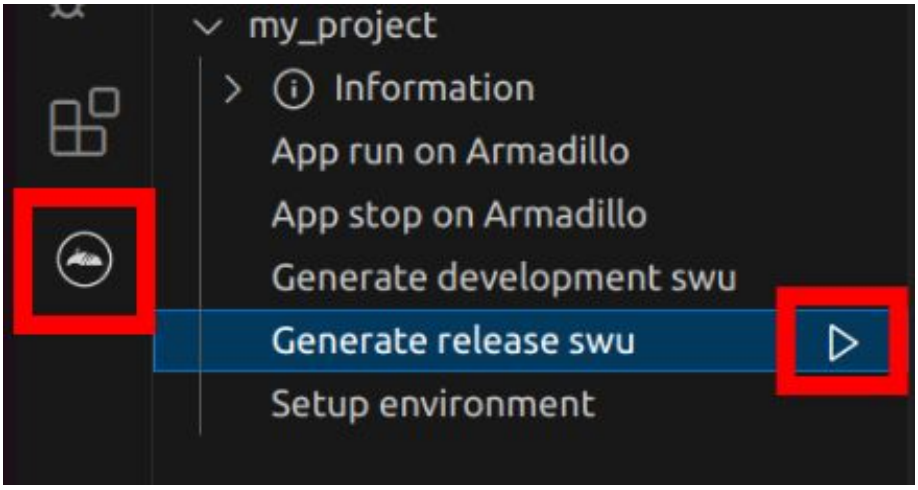


図 7.163 リリース版をビルドする

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

7.10.10. 製品への書き込み

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

この SWU イメージを 「7.3.3.6. SWU イメージのインストール」 を参照して Armadillo へインストールすると、Armadillo 起動時にアプリケーションも自動起動します。

7.10.11. Armadillo 上のコンテナイメージの削除

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

7.11. SBOM 生成に関わる設定を行う

ABOSDE では SWU イメージの生成と同時に SBOM が生成されます。生成される SBOM 名は `SWU イメージ.spdx.json` になります。json 形式で ISO/IEC5962 で国際標準となっている SPDX2.2 のフォーマットに準拠しています。SBOM についての詳細は 「10.20. SBOM の提供」 をご参照ください。

SBOM の生成には mkswu （6.4 以上）と、python3-make-sbom パッケージが必要です。python3-make-sbom パッケージがインストールされていない場合、SBOM は生成されません。「図 7.164. mkswu バージョン確認コマンド」 を実行するとインストール済のバージョンが確認できます。

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

図 7.164 mkswu バージョン確認コマンド

表示されない場合は mkswu がインストールされていませんので、「図 7.165. mkswu のインストール・アップデートコマンド」 を実行してインストールしてください。mkswu をアップデートする場合もこちらを実行して下さい。

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

図 7.165 mkswu のインストール・アップデートコマンド

python3-make-sbom パッケージがインストールされている場合、make_sbom.sh が実行可能です。「図 7.166. make_sbom.sh 実行確認コマンド」 を実行して、ヘルプが表示されるかご確認ください。

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

図 7.166 make_sbom.sh 実行確認コマンド

表示されない場合は python3-make-sbom がインストールされていませんので、「図 7.167. python3-make-sbom のインストールコマンド」 を実行してインストールしてください。

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

図 7.167 python3-make-sbom のインストールコマンド

7.11.1. SBOM 生成に必要なファイルを確認する

SBOM の生成には以下の二つのファイルが必要です。

- コンフィグファイル
- desc ファイル

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

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

SBOM に含めるコンテナイメージ等の情報については desc ファイルに記載されています。各項目の説明については 「10.20.4.2. desc ファイルを編集する」 をご覧ください。

7.12. 生成した SBOM をスキャンする

SBOM の利点のひとつに、スキャンツールに入力することでソフトウェアに含まれる脆弱性を検出することができる点が挙げられます。ここでは、 Google が提供しているオープンソース SBOM スキャンツール OSV-Scanner[2] を用いて、開発したソフトウェアに既知の脆弱性が含まれているかを確認する方法を紹介します。

7.12.1. OSV-Scanner のインストール

以下の手順はすべて ATDE 上で行います。

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

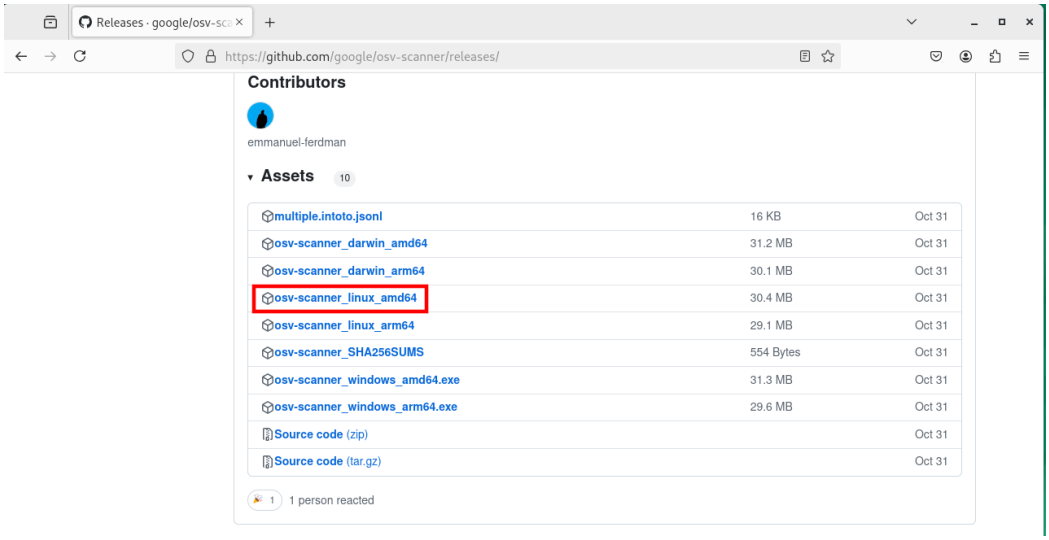


図 7.168 OSV-Scanner の実行ファイルをダウンロード

次に、「図 7.169. OSV-Scanner をインストールする」のコマンドを実行することで、 OSV-Scanner がインストールされます。

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

図 7.169 OSV-Scanner をインストールする

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

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

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

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

VERSION:
   1.9.1

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

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

図 7.170 OSV-Scanner がインストールされたことを確認する

これで OSV-Scanner のインストールが完了しました。

7.12.2. OSV-Scanner でソフトウェアの脆弱性を検査する

「図 7.171. OSV-Scanner を用いて SBOM をスキャンする」に示すコマンドを実行することで、ソフトウェアに含まれる既知の脆弱性を検出します。ここでは例として ABOSDE で開発したアプリケーションの SBOM をスキャンします。SBOM 生成については 「10.20.4. SWU イメージと同時に SBOM を作成する」 を参照してください。

```
[ATDE ~]$ osv-scanner scan --sbom ~/my_project/development.swu.spdx.json --format markdown ❶
Scanned /home/atmark/my_project/development.swu.spdx.json as SPDX SBOM and found 97 packages
5 unimportant vulnerabilities have been filtered out.
Filtered 5 vulnerabilities from output
| OSV URL | CVSS | Ecosystem | Package | Version | Source |
| --- | --- | --- | --- | --- | --- |
| https://osv.dev/CVE-2022-3715 | 7.8 | Debian | bash | 5.1-2+deb11u1 | development.swu.spdx.json |
| https://osv.dev/CVE-2016-2781 | 6.5 | Debian | coreutils | 8.32-4 | development.swu.spdx.json |
| https://osv.dev/CVE-2021-33560 | 7.5 | Debian | libgcrypt20 | 1.8.7-6 | development.swu.spdx.json |
| https://osv.dev/CVE-2024-2236 |  | Debian | libgcrypt20 | 1.8.7-6 | development.swu.spdx.json |
```

図 7.171 OSV-Scanner を用いて SBOM をスキャンする

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

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

7.13. システムのテストを行う

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

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

7.13.1. ランニングテスト

長期間のランニングテストは実施すべきです。

ランニングテストで発見できる現象としては、以下のようなものが挙げられます。

- 長期間稼働することでソフトウェアの動作が停止してしまう

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

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

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

- メモリリークが発生する

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

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

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

  メモリの空き容量は「図 7.172. メモリの空き容量の確認方法」に示すように free コマンドで確認できます。

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

図 7.172 メモリの空き容量の確認方法

7.13.2. 異常系における挙動のテスト

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

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

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

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

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

7.14. ユーザー設定とユーザーデータを一括削除する

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

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

ABOS Web から設定できるものであっても以下は削除されません。

- Rest API トークン
- UI カスタマイズの内容

ユーザーデータの削除では以下のデータを削除します。

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

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

```
[armadillo ~]# abos-ctrl reset-default ❶
Run with dry-run mode.
rm -f /etc/NetworkManager/system-connections/*
persist_file -r /etc/NetworkManager/system-connections
persist_file -r /etc/dnsmasq.d
rc-service dnsmasq restart
/etc/init.d/iptables save
sed -i -e '/NETAVARK/d' /etc/iptables/rules-save
persist_file /etc/iptables/rules-save
podman stop -a
find /var/app/volumes /var/log -mindepth 1 -delete
```

```
If you want to actually run the above commands,
add the -f/--force option.
```

図 7.173 削除されるユーザー設定とユーザーデータを確認

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

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

```
[armadillo ~]# abos-ctrl reset-default -f
rm -f /etc/NetworkManager/system-connections/*
persist_file -r /etc/NetworkManager/system-connections
persist_file -r /etc/dnsmasq.d
rc-service dnsmasq restart
/etc/init.d/iptables save
sed -i -e '/NETAVARK/d' /etc/iptables/rules-save
persist_file /etc/iptables/rules-save
podman stop -a
find /var/app/volumes /var/log -mindepth 1 -delete
Starting clone to /dev/mmcblk2p1
Cloning rootfs
Updating appfs snapshots
Reusing up-to-date bootloader
Rollback clone successful
WARNING: Rebooting!
```

図 7.174 実際にユーザー設定とユーザーデータを削除する

コマンド実行後は自動的に Armadillo が再起動します。

ABOS Web または Rest API から実行することもできます。ABOS Web から実行する場合は 「10.9.8. ユーザー設定とユーザーデータの削除」 を参照してください。Rest API から実行する場合は 「10.9.6.16. Rest API : ユーザー設定とユーザーデータの管理」 を参照してください。

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

8. 量産編

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

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

8.1. 概略

量産の進め方の概略図を「図 8.1. Armadillo 量産時の概略図」に示します。お客様の製品仕様や製造工程の要件によってはこの例とは違った工程順となる場合や、工程の追加・削除がある可能性があります。

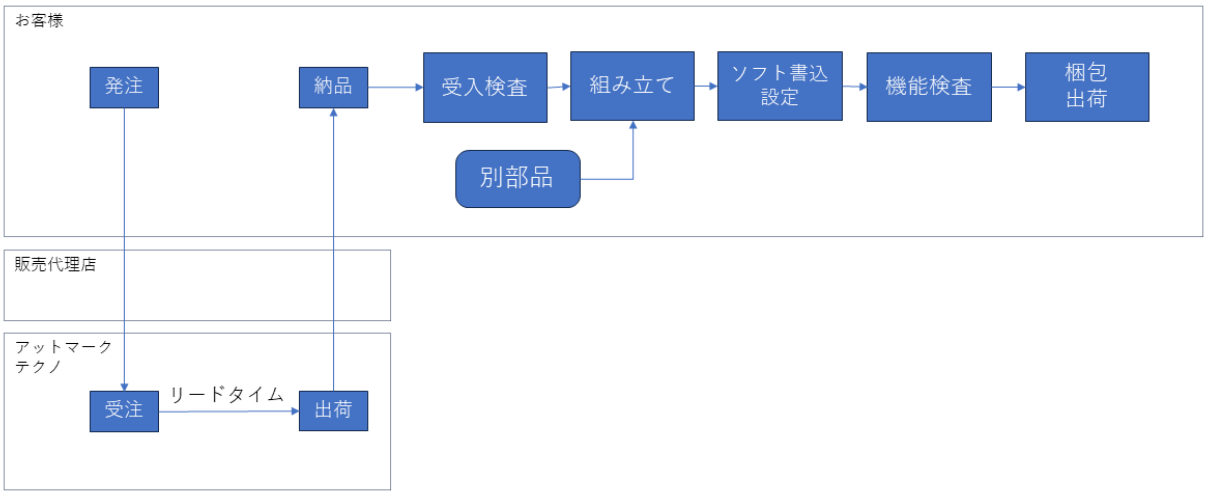


図 8.1 Armadillo 量産時の概略図

8.1.1. リードタイムと在庫

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

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

8.1.2. Armadillo 納品後の製造・量産作業

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

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

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

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

8.2. BTO サービスを使わない場合と使う場合の違い

図 8.2 BTO サービスで対応する範囲

8.2.1. BTO サービスを利用しない(標準ラインアップ品)

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

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

8.2.1.1. 標準ラインアップ品に書き込まれているソフトウェア

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

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

納品後、お客様の量産工程でソフトウェアの書き込み作業が必要となります。詳しくは「8.3. 量産時のイメージ書き込み手法」をご確認ください。

8.2.2. BTO サービスを利用する

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

販売代理店またはアットマークテクノの窓口からお申し込みいただけます。

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

8.3. 量産時のイメージ書き込み手法

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

- インストールディスクを用いてソフトウェアを書き込む
- SWUpdate を用いてソフトウェアを書き込む

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

それぞれの手法の特徴を「表 8.1. インストールディスクと SWUpdate によるソフトウェア書き込みの比較」にまとめます。ソフトウェア書き込み工程を決定する際の参考にしてください。

表 8.1 インストールディスクと SWUpdate によるソフトウェア書き込みの比較

手段	メリット	デメリット
インストールディスク	・インストールの前後処理を行なうシェルスクリプトのテンプレートが用意されている ・インストールの前後処理は、microSD カード内にシェルスクリプトを配置するだけなので製造担当者にも編集しやすい	・動いているシステムをそのままインストールディスクにするため、出荷時の標準イメージから手動で同じ環境を構築する手順が残らない
SWUpdate	・必ず必要となる初回アップデートを別途実行する必要がない	・SWU イメージの作成には、mkswu を使用できる環境と desc ファイルの記述方法を知る必要があるため、開発担当者以外に SWU イメージを更新させるハードルが少し高い ・ログの取得など、インストール前後の処理が必要な場合は自分で記述する必要がある

量産時のイメージ書き込みにインストールディスクを使用する場合は、「8.4. インストールディスクを用いてイメージ書き込みする」に進んでください。

量産時のイメージ書き込みに SWUpdate を使用する場合は、「8.5. SWUpdate を用いてイメージ書き込みする」に進んでください。

8.4. インストールディスクを用いてイメージ書き込みする

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

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

8.4.1. /etc/swupdate_preserve_file への追記

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

ここでは SWUpdate を使用して Armadillo Base OS のアップデートを行ないますが、このアップデートを行なうと、/etc/swupdate_preserve_files に記載の無いファイルは消えてしまいます。Armadillo Base OS のルートファイルシステム上に消えてほしくないファイルを開発中に配置していた場合は、「図 8.3. 任意のファイルパスを/etc/swupdate_preserve_files に追記する」に示すコマンドを実行することで /etc/swupdate_preserve_files にそのファイルが追記され、アップデート後も保持し続けるようになります。

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

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

図 8.3 任意のファイルパスを/etc/swupdate_preserve_files に追記する

8.4.2. Armadillo Base OS の更新

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

/etc/swupdate.watch に記載されている URL の SWU イメージでアップデートを行いますので、デフォルトでは 最新の Armadillo Base OS へアップデートします。

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

「図 8.4. Armadillo Base OS を最新にアップデートする」に示すコマンドを実行することで Armadillo Base OS を最新版にアップデートできます。

```
[armadillo /]# swupdate -d '-u https://armadillo.atmark-techno.com/
files/downloads/armadillo-900/image/baseos-900-latest.swu'
```

図 8.4 Armadillo Base OS を最新にアップデートする

正常に実行された場合は自動的に再起動します。

8.4.3. パスワードの確認と変更

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

```
[armadillo /]# passwd ❶
Changing password for root
New password: ❷
Retype password: ❸
passwd: password for root changed by root

[armadillo /]# passwd atmark ❹
Changing password for atmark
New password: ❺
Retype password: ❻
passwd: password for atmark changed by root

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

図 8.5 パスワードを変更する

❶ root ユーザのパスワードを変更します。

❷ 新しい root ユーザ用パスワードを入力します。

❸ 再度新しい root ユーザ用パスワードを入力します。

❹ atmark ユーザのパスワードを変更します。

❺ 新しい atmark ユーザ用パスワードを入力します。

❻ 再度新しい atmark ユーザ用パスワードを入力します。

❼ パスワードの変更を永続化させます。

8.4.4. 開発したシステムをインストールディスクにする

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

- VS Code を使用して生成

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

- コマンドラインから生成

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

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

8.4.5. VS Code を使用して生成する

ATDE と VS Code を使用して、開発したシステムのインストールディスクイメージを生成します。「7.1.2. VS Code のセットアップ」 を参考に、 ATDE に VS Code 開発用エクステンションをインストールしてください。VS Code を使用してインストールディスクを生成する場合は以下の手順になります。

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

この機能を使用するには、以下に示すバージョンのソフトウェアが必要です。

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

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

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

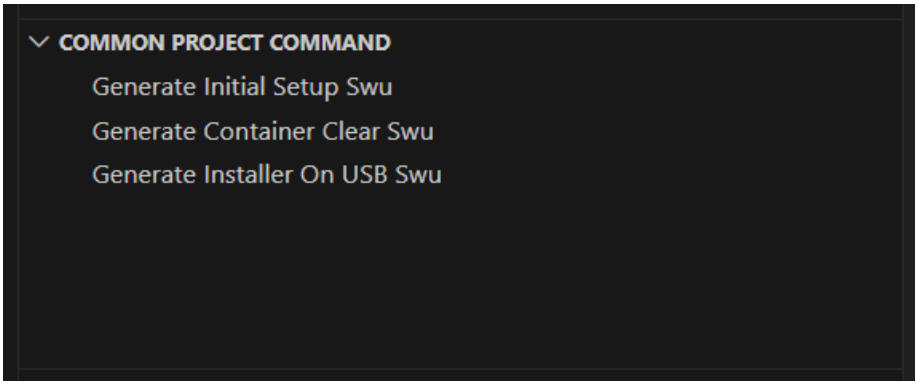


図 8.6 make-installer.swu を作成する

次に、対象製品を選択します。

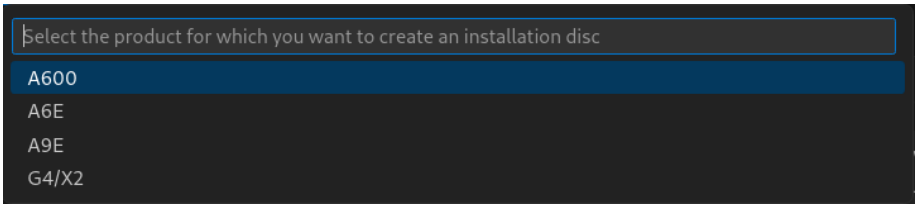


図 8.7 対象製品を選択する

無事に生成された場合、コンソールに以下のログが出力されます。

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

図 8.8 make-installer.swu 生成時のログ

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

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

8.4.5.2. Armadillo に USB メモリを挿入

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

USB メモリは vfat もしくは ext4 形式でフォーマットし、空き容量が 10GB 以上のものを使用してください。Armadillo-900 開発セット へのUSB メモリのマウントは不要です。

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

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

ABOS Web を使用して、生成した make-installer.swu をインストールします。「10.9.4. SWU インストール」を参考に make-installer.swu を Armadillo へインストールしてください。実行時は ABOS Web 上に「図 8.9. make-installer.swu インストール時のログ」ようなログが表示されます。

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

SWUpdate v2023.05_git20231025-r0

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

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

[INFO ] : SWUPDATE started : Software Update started !

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

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

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

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

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

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

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

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

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

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

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

[INFO ] : SWUPDATE running : [read_lines_notify] : Checking and growing installer main partition

[INFO ] : SWUPDATE running : [read_lines_notify] : Resize device id 1 (/dev/loop0p1) from 513.00MiB to max
```

```
[INFO ] : SWUPDATE running : [read_lines_notify] : Copying boot image

[INFO ] : SWUPDATE running : [read_lines_notify] : Copying rootfs

[INFO ] : SWUPDATE running : [read_lines_notify] : Copying appfs

[INFO ] : SWUPDATE running : [read_lines_notify] : At subvol app/snapshots/volumes

[INFO ] : SWUPDATE running : [read_lines_notify] : At subvol app/snapshots/boot_volumes

[INFO ] : SWUPDATE running : [read_lines_notify] : At subvol app/snapshots/boot_containers_storage

[INFO ] : SWUPDATE running : [read_lines_notify] : Trying to shrink the installer partition...

[INFO ] : SWUPDATE running : [read_lines_notify] : Shrinking the installer partition...

[INFO ] : SWUPDATE running : [read_lines_notify] : Cleaning up and syncing changes to disk...

[INFO ] : SWUPDATE running : [read_lines_notify] : Installer updated successfully!

[INFO ] : SWUPDATE running : [read_lines_notify] : -rwxr-xr-x 1 root root 687.0M Jan 23 15:12 /
target/mnt/installer.img

[INFO ] : SWUPDATE running : [install_single_image] : Installing post_script

[INFO ] : SWUPDATE running : [read_lines_notify] : Removing unused containers

[INFO ] : SWUPDATE running : [read_lines_notify] : Command 'command podman rm -a -f' output:

[INFO ] : SWUPDATE running : [read_lines_notify] :
9f4f64ec1926d17e75de4060dac4a448e66ca3d9535c408f632e4e2de4bafa4f

[INFO ] : SWUPDATE running : Installation in progress

[INFO ] : SWUPDATE successful ! SWUPDATE successful !

[INFO ] : No SWUPDATE running : Waiting for requests...

swupdate exited

インストールが成功しました。
```

図 8.9 make-installer.swu インストール時のログ

完了後、USB メモリを抜いてください。もし、エラーが出た場合は Armadillo-900 開発セットの電源を再投入してやり直してください。

8.4.5.4. USB メモリ上にインストールディスクイメージを生成

無事に生成が完了した場合、USB メモリ上に installer.img が保存されています。この installer.img を microSD カードに書き込むことでインストールディスクを作成することができます。動作確認については「8.4.6. インストールディスクの動作確認を行う」をご参照ください。これで、VS Code を使用してインストールディスクを生成する方法については終了です。

8.4.6. インストールディスクの動作確認を行う

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

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

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

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

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

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

8.4.7. コマンドラインから生成する

8.4.7.1. JTAG と SD ブートを無効化する

コマンドラインから生成する場合は、 Armadillo の JTAG と SD ブートを無効にするインストールディスクイメージを生成することができます。

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

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

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

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

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

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

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

図 8.10 JTAG と SD ブートを無効化する

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

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

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

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

図 8.11 JTAG と SD ブートの設定値を確認する

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

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

図 8.12 JTAG と SD ブートの設定値をリセットする

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

8.4.7.2. U-Boot のコマンドプロンプトを無効化する

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

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

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

図 8.13 U-Boot のコマンドプロンプトを無効化する

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

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

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

図 8.14 U-Boot のコマンドプロンプトの設定値を確認する

設定をリセットするには「図 8.12. JTAG と SD ブートの設定値をリセットする」と同様に、--reset オプションを付けて abos-ctrl installer-setting コマンドを実行してください。

8.4.7.3. インストールディスクイメージを生成する

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

```
[armadillo /]# abos-ctrl make-installer
Checking if /dev/mmcblk2 can be used safely...
It looks like your SD card does not contain an installer image
Download baseos-900-installer-latest.zip image from armadillo.atmark-techno.com (~170M) ? [y/N] ❶
WARNING: it will overwrite your SD card!!
y
Downloading and extracting image to SD card...
Finished writing baseos-900-installer-[VERSION].img, verifying written content...

Would you like to create a windows partition?
That partition would only be used for customization script at the end of
install, leave at 0 to skip creating it.
Custom partition size (MB, [0] or 16 - 29014): 500 ❷
Checking and growing installer main partition
Trying to install mkfs.exfat (exfatprogs) in memory from internet
fetch https://download.atmark-techno.com/alpine/v3.19/atmark/aarch64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.19/main/aarch64/APKINDEX.tar.gz
```

```
fetch https://dl-cdn.alpinelinux.org/alpine/v3.19/community/aarch64/APKINDEX.tar.gz
(1/1) Installing exfatprogs (1.2.2-r0)
Executing busybox-1.36.1-r15.trigger
OK: 148 MiB in 197 packages
exfatprogs version : 1.2.2
Creating exFAT filesystem(/dev/mmcblk2p2, cluster size=131072)

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

exFAT format complete!
Resize device id 1 (/dev/mmcblk2p1) from 520.00MiB to max
Installer will disable JTAG access
Installer will disable SD boot after installation
Installer will disable uboot prompt
Environment OK, copy 1
Copying boot image
Copying rootfs
Copying appfs
At subvol app/snapshots/volumes
At subvol app/snapshots/boot_volumes
At subvol app/snapshots/boot_containers_storage
Cleaning up and syncing changes to disk...
Installer updated successfully!
```

図 8.15 開発完了後のシステムをインストールディスクイメージにする

❶ y を入力し Enter を押下します。

❷ インストールディスク内にインストールログを保存したい場合など、自由に使用できる第 2 パーティションを指定したサイズで作成します。サイズを入力し Enter を押下します。詳細は「8.4.7.4. インストール時に任意のシェルスクリプトを実行する」を参照してください。

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

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

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

他のファイルを個体毎に変更したい場合は 「8.4.7.4. インストール時に任意のシェルスクリプトを実行する」 で対応してください。

8.4.7.4. インストール時に任意のシェルスクリプトを実行する

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

installer_overrides.sh に記載された「表 8.2. インストール中に実行される関数」に示す 3 つの名前の関数のみが、それぞれ特定のタイミングで実行されます。

表 8.2 インストール中に実行される関数

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

installer_overrides.sh を書くためのサンプルとして、 インストールディスクイメージの第 1 パーティション及び、「8.4.4. 開発したシステムをインストールディスクにする」で作成したのであれば第 2 パーティション直下に installer_overrides.sh.sample を用意してあります。このサンプルをコピーして編集するなどして、行ないたい処理を記述してください。

作成した installer_overrides.sh は、インストールディスクの第 1 パーティション(ラベル名は "rootfs_0")か、「8.4.4. 開発したシステムをインストールディスクにする」で作成したのであれば第 2 パーティション(ラベル名は"INST_DATA")の直下に配置することで実行されます。両方に配置した場合は、第 2 パーティションに配置した記述が適用されます。

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

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

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

これらを必要としない場合は「8.4.8. インストールの実行」に進んでください。

8.4.7.5. 個体ごとに異なる固定 IP アドレスを設定する

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

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

今回はそれぞれのファイルの内容は変更せず使用します。サンプルそのままですが、 ip_config.txt の内容を「図 8.16. ip_config.txt の内容」に示します。

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

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

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

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

# DNS servers to configure if present, semi-colon separated list
# not set if absent
DNS="1.1.1.1;8.8.8.8" ❺

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

図 8.16 ip_config.txt の内容

❶ このインストールディスクで割り振る IP アドレスの範囲の始まりを指定します。

❷ このインストールディスクで割り振る IP アドレスの範囲の終わりを指定します。

❸ ネットマスクを指定します。指定しない場合は 24 になります。デフォルトでコメントアウトされています。

❹ ゲートウェイアドレスを指定します。

❺ DNS アドレスを指定します。セミコロンで区切ることでセカンダリアドレスも指定できます。

❻ IP アドレスの設定を行なうインターフェースを指定します。指定しない場合は eth0 になります。デフォルトでコメントアウトされています。

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

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

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

図 8.17 IP アドレスの確認

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

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

図 8.18 allocated_ips.csv の内容

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

8.4.7.6. インストール実行時のログを保存する

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

「図 8.19. インストールログを保存する」は、インストールディスクの第 2 パーティションにインストールログを保存する場合の send_log 実装例です。

```
send_log() {
        : "This function is called after aggregating logs for archival"
        local LOG_FILE="$1"

        if [ -n "$USER_MOUNT" ]; then
                        mount /dev/mmcblk2p2 "$USER_MOUNT" ❶
                        cp $LOG_FILE $USER_MOUNT/${SN}_install.log ❷
                        umount "$USER_MOUNT" ❸
        fi
}
```

図 8.19 インストールログを保存する

❶ send_log 関数中では、SD カードの第 2 パーティション(/dev/mmcblk2p2)はマウントされていないのでマウントします。

❷ ログファイルを <シリアル番号>_install.log というファイル名で第 2 パーティションにコピーします。

❸ 第 2 パーティションをアンマウントします。

これらの変更を行なったインストールディスクでインストールを実行した後に、インストールディスクを PC などに接続して正しくログを保存できていることを確認してください。保存したログファイルの中身の例を「図 8.20. インストールログの中身」に示します。

```
RESULT:OK
abos-ctrl make-rootfs on Tue Jun 21 17:57:07 JST 2022 4194304 6b8250df711de66b
```

```
abos-ctrl make-rootfs on Tue Jun 21 17:57:24 JST 2022 314572800 58a9b6664158943e
firm 8e9d83d1ba4db65d
appfs 5108 1fa2cbaff09c2dbf
```

図 8.20 インストールログの中身

8.4.8. インストールの実行

前章までの手順で作成したインストールディスクを、開発に使用した Armadillo 以外の Armadillo に対して適用します。

クローン先の Armadillo の eMMC 内のデータは上書きされて消えるため、必要なデータは予めバックアップを取っておいてください。

「7.3.5.2. インストールディスクを使用する」を参照して、クローン先の Armadillo にインストールディスクを適用してください。

「8.4.7.1. JTAG と SD ブートを無効化する」 で JTAG と SD ブートを無効化した場合は、インストールを行った Armadillo の JTAG と SD ブートが無効化されています。

ここまで完了したら、「8.6. イメージ書き込み後の動作確認」に進んでください。

8.5. SWUpdate を用いてイメージ書き込みする

8.5.1. SWU イメージの準備

ここでは、sample-container という名称のコンテナの開発を終了したとします。コンテナアーカイブの作成方法は 「4.4.2.11. コンテナイメージを作成する」 を参照ください。

1. sample-container-v1.0.0.tar (動かしたいアプリケーションを含むコンテナイメージアーカイブ)
2. sample-container.conf (コンテナ自動実行用設定ファイル)

これらのファイルを /home/atmark/mkswu/sample-container ディレクトリを作成して配置した例を記載します。

```
[ATDE ~/mkswu/sample-container]$ ls
sample-container-v1.0.0.tar  sample-container.conf
```

図 8.21 Armadillo に書き込みたいソフトウェアを ATDE に配置

8.5.2. desc ファイルの記述

SWUpdate 実行時に、「8.5.1. SWU イメージの準備」で挙げたファイルを Armadillo に書き込むような SWU イメージを生成します。

SWU イメージを作成するためには、SWUpdate 時に実行する処理等を示した desc ファイルを記述し、その desc ファイルを入力として mkswu コマンドを実行することで、SWU イメージが出来上がります。

```
[ATDE ~/mkswu/sample-container]$ cat sample-container.desc
swdesc_option component=sample-container
```

```
swdesc_option version=1
swdesc_option POST_ACTION=poweroff ❶

swdesc_embed_container "sample-container-v1.0.0.tar" ❷
swdesc_files --extra-os --dest /etc/atmark/containers/ "sample-container.conf" ❸
```

図 8.22 desc ファイルの記述例

❶ SWUpdate 完了後に電源を切るように設定します。

❷ コンテナイメージファイルを SWU イメージに組み込み、Armadillo に転送します。

❸ コンテナ起動設定ファイルを Armadillo に転送します。

ここまで完了したら、「8.6. イメージ書き込み後の動作確認」に進んでください。desc ファイルの詳細な書式については、「10.3. mkswu の .desc ファイルを編集する」を参照してください。また、作成された SWU イメージの内容を確認したい場合は、「10.5. SWU イメージの内容の確認」を参照してください。

8.6. イメージ書き込み後の動作確認

「8.4. インストールディスクを用いてイメージ書き込みする」で作成したインストールディスクを使用してインストール、または「8.5. SWUpdate を用いてイメージ書き込みする」にて SWUpdate によってイメージ書き込みを行った後には、イメージが書き込まれた Armadillo が正しく動作するか、実際に動かして確認してみます。

再度電源を投入して、期待したアプリケーションが動作することを確認してください。

ここまで完了したならば、量産時のイメージ書き込みは完了です。

9. 運用編

9.1. Armadillo を設置する

Armadillo を組み込んだ製品を設置する際の注意点や参考情報を紹介します。

9.1.1. 設置場所

開発時と同様に、水・湿気・ほこり・油煙等の多い場所に設置しないでください。火災、故障、感電などの原因になる場合があります。

本製品に搭載されている部品の一部は、発熱により高温になる場合があります。周囲温度や取扱いによってはやけどの原因となる恐れがあります。本体の電源が入っている間、または電源切断後本体の温度が下がるまでの間は、基板上の電子部品、及びその周辺部分には触れないでください。

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

9.1.2. ケーブルの取り回し

一般的に以下の点を注意して設置してください。

- 設置時にケーブルを強く引っ張らないでください。
- ケーブルはゆるやかに曲げてください。 ケーブルを結線する場合、きつくせず緩く束ねてください。

9.1.3. WLAN/BT/TH 用外付けアンテナの指向性

WLAN/BT/TH 用外付けアンテナナは「図 9.1. Armadillo-900 開発セット WLAN/BT/TH 外付けアンテナの指向性」に示す指向性があります。

一般的な λ/2 ダイポールアンテナ、λ/4 モノポールアンテナの指向性イメージです。実際のアンテナの特性を正確に表しているものではありません

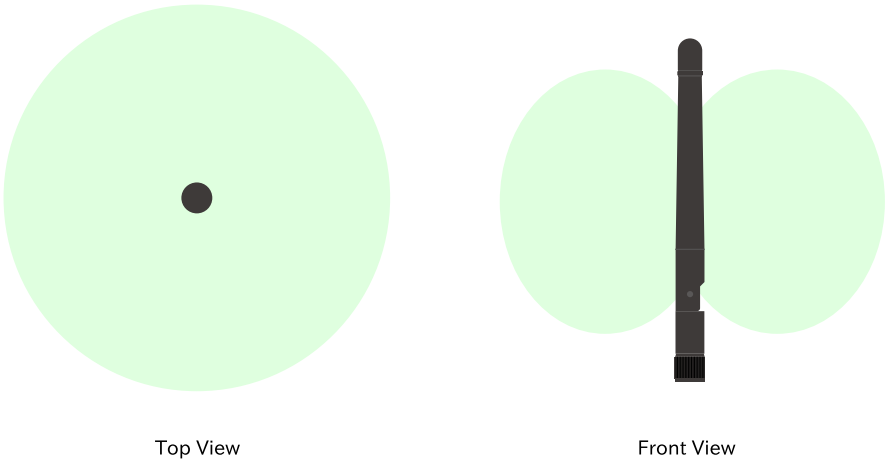


図 9.1 Armadillo-900 開発セット WLAN/BT/TH 外付けアンテナの指向性

9.1.4. LTE 用外付けアンテナの指向性

LTE 用外付けアンテナは「図 9.2. LTE 外付け用アンテナの指向性」に示す指向性があります。

一般的な λ/2 ダイポールアンテナ、λ/4 モノポールアンテナの指向性イメージです。実際のアンテナの特性を正確に表しているものではありません

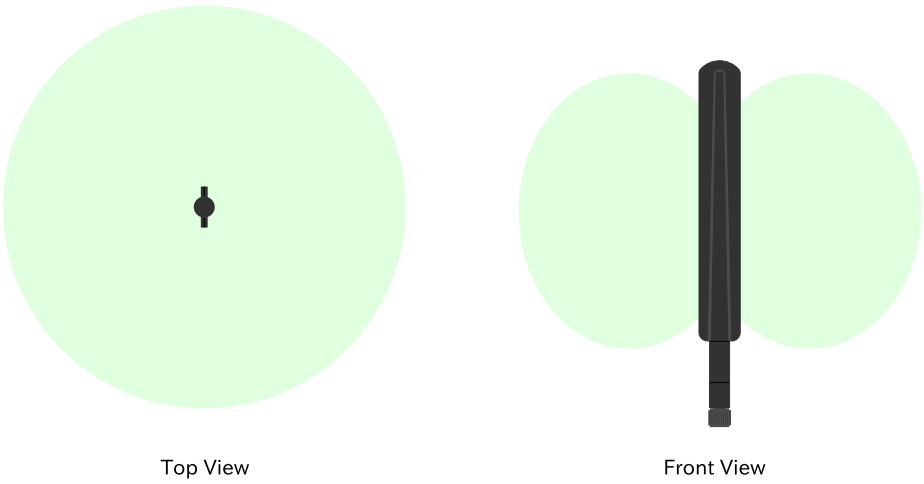


図 9.2 LTE 外付け用アンテナの指向性

9.1.5. LTE の電波品質に影響する事項

一般的には、周辺に障害物、特に鉄板や鉄筋コンクリート、電波シールドフィルムの貼られたガラスが存在する場合電波を大きく妨げます。また、周辺機器の電波出力、人通り(周辺のスマートフォン=機器が増える)、基地局との距離・方向など多くの要素によって変化します。

WWAN LED にて電波状況をチェックできますので、設置時に電波状況を確認の上運用ください。WWAN LED の状態と意味は「表 5.45. LED 状態と製品状態の対応について」を参照ください。

9.1.6. サージ対策

サージ対策については、以下の対策が効果的です。

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

また、接続されたケーブルが屋外に露出するような設置環境では、ケーブルに侵入した雷サージ等のストレスによりインターフェース回路が破壊される場合があります。ストレスへの耐性を向上させるために、以下の対策が効果的です。

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

9.1.7. Armadillo の状態を表すインジケータ

LED にて状態を表示しています。

有線 LAN の状態は「表 5.16. CON6 LAN LED の動作」を、Armadillo の状態を表示する LED に関しては「表 5.45. LED 状態と製品状態の対応について」を参照ください。

9.1.8. 個体識別情報の取得

設置時に Armadillo を個体ごとに識別したい場合、以下の情報を個体識別情報として利用できます。

- 個体番号
- MAC アドレス

Armadillo の設置前に個体識別情報を記録しておき、設置後の Armadillo を識別できるようにしておくことを推奨します。

これらの情報を取得する方法は以下のとおりです。状況に合わせて手段を選択してください。

- 本体シールから取得する
- コマンドによって取得する

9.1.8.1. 本体シールから取得

Armadillo の各種個体番号、 MAC アドレスなどの個体識別情報は、ケース裏や基板本体に貼付されているシールに記載されています。製品モデル毎に記載されている内容やシールの位置が異なるので、詳細は各種納入仕様書を参照してください。

9.1.8.2. コマンドによる取得

シールだけでなくコマンドを実行することによっても個体識別情報を取得することができます。以下に個体番号と MAC アドレスを取得する方法を説明します。

個体番号を取得する場合、「図 9.3. 個体番号の取得方法（device-info）」に示すコマンドを実行してください。device-info はバージョン v3.18.4-at.7 以降の ABOS に標準で組み込まれています。

```
[armadillo ~]# device-info -s
00C900010001 ❶
```

図 9.3 個体番号の取得方法（device-info）

❶ 使用している Armadillo の個体番号が表示されます。

device-info がインストールされていない場合は「図 9.4. device-info のインストール方法」に示すコマンドを実行することでインストールできます。

```
[armadillo ~]# persist_file -a update
[armadillo ~]# persist_file -a add device-info
```

図 9.4 device-info のインストール方法

上記の方法で device-info をインストールできない場合は最新のバージョンの ABOS にアップデートすることを強く推奨します。非推奨ですが、ABOS をアップデートせずに個体番号を取得したい場合は「図 9.5. 個体番号の取得方法（get-board-info）」に示すように get-board-info を実行することでも取得できます。

```
[armadillo ~]# persist_file -a add get-board-info
[armadillo ~]# get-board-info -s
00C900010001 ❶
```

図 9.5 個体番号の取得方法（get-board-info）

❶ 使用している Armadillo の個体番号が表示されます。

コンテナ上で個体番号を表示する場合は、個体番号を環境変数として設定することで可能となります。「図 9.6. 個体番号の環境変数を conf ファイルに追記」に示す内容を/etc/atmark/containers の下の conf ファイルに記入します。

```
add_args --env=SERIALNUM=$(device-info -s) ❶
```

図 9.6 個体番号の環境変数を conf ファイルに追記

❶ コンテナ起動毎に環境変数 SERIALNUM に値がセットされます。

「図 9.7. コンテナ上で個体番号を確認する方法」に示すコマンドを実行することでコンテナ上で個体番号を確認することができます。

```
[container ~]# echo $SERIALNUM
00C900010001
```

図 9.7 コンテナ上で個体番号を確認する方法

次に MAC アドレスを取得する方法を説明します。「図 9.8. MAC アドレスの確認方法」に示すコマンドを実行することで、各インターフェースの MAC アドレスを取得できます。

```
[armadillo ~]# ip addr
:（省略）
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
    link/ether 00:11:0c:12:34:56 brd ff:ff:ff:ff:ff:ff ❶
:（省略）
```

図 9.8 MAC アドレスの確認方法

❶ link/ether に続くアドレスが MAC アドレスです。

また、出荷時にアットマークテクノが書き込んだ Ethernet MAC アドレスは「図 9.9. 出荷時の Ethernet MAC アドレスの確認方法」に示すコマンドを実行することで取得することができます。

```
[armadillo ~]# device-info -m
eth0: 00:11:0C:12:34:56 ❶
```

図 9.9 出荷時の Ethernet MAC アドレスの確認方法

❶ 出荷時にアットマークテクノが書き込んだ Ethernet MAC アドレスが表示されます。

ただし、「図 9.9. 出荷時の Ethernet MAC アドレスの確認方法」で示すコマンドでは、お客様自身で設定した Ethernet MAC アドレスを取得することはできないのでご注意ください。お客様自身で設定した Ethernet MAC アドレスを取得したい場合は「図 9.8. MAC アドレスの確認方法」に示すコマンドを実行してください。

9.1.9. 電源を切る

Armadillo の電源を切る場合は、 poweroff コマンドを実行してから電源を切るのが理想的です。しかし、設置後はコマンドを実行できる環境にない場合が多いです。この場合、条件が整えば poweroff コマンドを実行せずに電源を切断しても安全に終了できる場合があります。

詳細は、「3.3.5. Armadillo の終了方法」を参照してください。

9.2. ABOSDE で開発したアプリケーションをアップデートする

ABOSDE で開発したアプリケーションのアップデートは、開発時と同様に ABOSDE を用いて行うことが出来ます。

「7.7. ABOSDE によるアプリケーションの開発」で示したように、開発時にはリリース版のアプリケーションを Armadillo にインストールするために、 VS Code の左ペインの [Generate release swu] を実行して release.swu を作成しました。

アップデート時にも、アップデートに必要なアプリケーションの編集をした後に [Generate release swu] を実行して、アップデート版のアプリケーションを含む release.swu を作成します。

具体的な ABOSDE を用いたアプリケーションのアップデートの流れは「9.2.1. アプリケーションのアップデート手順」に示します。

9.2.1. アプリケーションのアップデート手順

ここでは、プロジェクト名を my_project としています。

9.2.1.1. アップデートするアプリケーションのプロジェクトを VS Code で開く

「図 9.10. VS Code を起動」で示すように、アップデートするアプリケーションのプロジェクトを指定して VS Code を起動してください。

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

図 9.10 VS Code を起動

9.2.1.2. アップデート前のバージョンのプロジェクトを管理する

ABOSDE では、プロジェクトのバージョン管理は行っていません。必要な場合はユーザー自身でアップデート前のプロジェクトを管理してください。

アップデート前のプロジェクトの release.swu のバージョンを知りたい場合は「10.5. SWU イメージの内容の確認」を参照してください。

9.2.1.3. アプリケーションのソースコードを編集しテストする

既存のアプリケーションのソースコードを編集した後、「7.7. ABOSDE によるアプリケーションの開発」を参考に、アプリケーションが Armadillo 上で問題なく動作するかテストを行ってください。

9.2.1.4. アップデート用の swu を作成する

VS Code の左ペインの [Generate release swu] を実行してください。my_project ディレクトリ下に release.swu というファイル名で SWU ファイルが作成されます。

9.2.1.5. 運用中の Armadillo のアプリケーションをアップデートする

アプリケーションをアップデートするために、作成した release.swu を運用中の Armadillo にインストールしてください。SWU イメージファイルをインストールする方法は「7.3.3.6. SWU イメージのインストール」を参照してください。

9.3. Armadillo のソフトウェアをアップデートする

設置後の Armadillo のソフトウェアアップデートは SWUpdate を使用することで実現できます。

ここでは、ソフトウェアのアップデートとして以下のような処理を行うことを例として説明します。

- すでに Armadillo に sample_container_image というコンテナイメージがインストールされている
- sample_container_image のバージョンを 1.0.0 から 1.0.1 にアップデートする
- sample_container_image からコンテナを自動起動するための設定ファイル(sample_container.conf)もアップデートする

9.3.1. SWU イメージの作成

アップデートのために SWU イメージを作成します。SWU イメージの作成には、 mkswu というツールを使います。「7.1. 開発の準備」で作成した環境で作業してください。

9.3.1.1. SBOM の生成

SWU イメージ作成時に、同時に SBOM を生成することができます。詳細は 「10.20. SBOM の提供」を参照してください。

9.3.2. mkswu の desc ファイルを作成する

SWU イメージを生成するには、 desc ファイルを作成する必要があります。

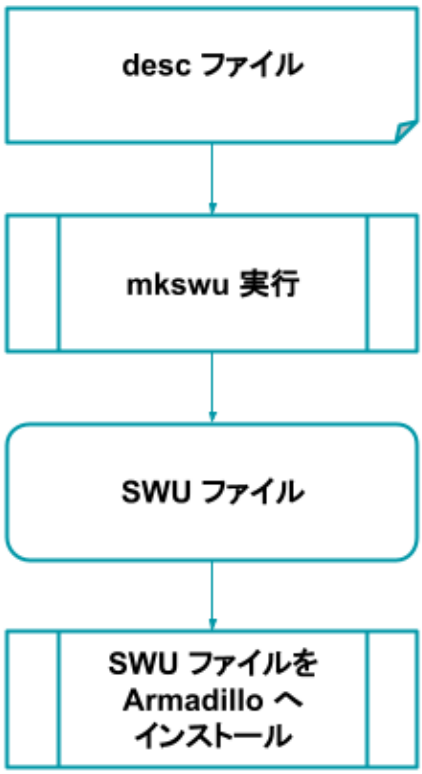


図 9.11 desc ファイルから Armadillo へ SWU イメージをインストールする流れ

desc ファイルとは、 SWU イメージを Armadillo にインストールする際に行われる命令を記述したものです。/usr/share/mkswu/examples/ ディレクトリ以下にサンプルを用意していますので、やりたいことに合わせて編集してお使いください。なお、 desc ファイルの詳細な書式については、「10.3. mkswu の .desc ファイルを編集する」を参照してください。

まず、以下のようなディレクトリ構成で、 sample_container.conf を作成しておきます。設定ファイルの内容については割愛します。

```
[ATDE ~/mkswu]$ tree container_start
container_start
└── etc
    └── atmark
        └── containers
            └── sample_container.conf
```

このような階層構造にしているのは、インストール先の Armadillo 上で sample_container.conf を /etc/atmark/containers/ の下に配置したいためです。

次に、アップデート先のコンテナイメージファイルである sample_container_image.tar を用意します。コンテナイメージを tar ファイルとして出力する方法を「図 9.12. コンテナイメージアーカイブ作成例」に示します。

```
[armadillo ~]# podman save sample_container:[VERSION] -o sample_container_image.tar
```

図 9.12 コンテナイメージアーカイブ作成例

次に、sample_container_update.desc という名前で desc ファイルを作成します。「図 9.13. sample_container_update.desc の内容」に、今回の例で使用する sample_container_update.desc ファイルの内容を示します。sample_container_image.tar と、コンテナ起動設定ファイルを Armadillo にインストールする処理が記述されています。

```
[ATDE ~/mkswu]$ cat sample_container_update.desc
swdesc_option version=1.0.1

swdesc_usb_container "sample_container_image.tar" ❶
swdesc_files --extra-os "container_start" ❷
```

図 9.13 sample_container_update.desc の内容

❶ sample_container_image.tar ファイルに保存されたコンテナをインストールします。

❷ container_start ディレクトリの中身を転送します。

コマンドは書かれた順番でインストールされます。

9.3.3. desc ファイルから SWU イメージを生成する

mkswu コマンドを実行することで、 desc ファイルから SWU イメージを生成できます。

```
[ATDE ~/mkswu]$ mkswu -o sample_container_update.swu sample_container_update.desc ❶
[ATDE ~/mkswu]$ ls sample_container_update.swu ❷
sample_container_update.swu
```

図 9.14 sample_container_update.desc の内容

❶ mkswu コマンドで desc ファイルから SWU イメージを生成

❷ sample_container_update.swu が生成されていることを確認

作成された SWU イメージの内容を確認したい場合は、「10.5. SWU イメージの内容の確認」を参照してください。

9.3.4. イメージのインストール

インストールの手順については、「7.3.3.6. SWU イメージのインストール」を参照してください。

9.4. eMMC の寿命を確認する

9.4.1. eMMC について

eMMC とは embedded Multi Media Card の頭文字を取った略称で NAND 型のフラッシュメモリを利用した内蔵ストレージです。当社で使用しているものは長期間運用を前提としている為、使用する容量を半分以下にして SLC モードで使用しています。(例えば 32GB 製品を 10GB で使用、残り 22GB は予備領域とする)。

eMMC は耐性に問題が発生した個所を内部コントローラがマスクし、予備領域を割り当てて調整しています。絶対ではありませんが、この予備領域がなくなると書き込みが出来なくなる可能性があります。

9.4.2. eMMC 予備領域の確認方法

Armadillo Base OS には emmc-utils というパッケージがインストールされています。

「図 9.15. eMMC の予備領域使用率を確認する」に示すコマンドを実行し、 EXT_CSD_PRE_EOL_INFO の内容を確認することで eMMC の予備領域の使用率がわかります。 EXT_CSD_PRE_EOL_INFO の値と意味の対応を「表 9.1. EXT_CSD_PRE_EOL_INFO の値の意味」に示します。

```
[armadillo ~]# mmc extcsd read /dev/mmcblk0 | grep EXT_CSD_PRE_EOL_INFO
eMMC Pre EOL information [EXT_CSD_PRE_EOL_INFO]: 0x01
```

図 9.15 eMMC の予備領域使用率を確認する

表 9.1 EXT_CSD_PRE_EOL_INFO の値の意味

値	意味
0x01	定常状態(問題無し)
0x02	予備領域 80% 以上使用
0x03	予備領域 90% 以上使用

また、Armadillo Twin からも eMMC の予備領域使用率を確認することができます。詳細は Armadillo Twin ユーザーマニュアル 「デバイス監視アラートを管理する」 [https://manual.armadillo-twin.com/management-device-monitoring-alert/] をご確認ください。

9.5. Armadillo の部品変更情報を知る

Armadillo に搭載されている部品が変更された場合や、製品が EOL となった場合には以下のページから確認できます。

Armadillo サイト - 変更通知(PCN)/EOL 通知

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

また、Armadillo サイトにユーザー登録していただくと、お知らせをメールで受信することが可能です。変更通知についても、メールで受け取ることが可能ですので、ユーザー登録をお願いいたします。

ユーザー登録については「4.6. ユーザー登録」を参照してください。

9.6. Armadillo を廃棄する

運用を終了し Armadillo を廃棄する際、セキュリティーの観点から以下のようなことを実施する必要があります。

- 設置場所に Armadillo を放置せず回収する
- Armadillo をネットワークから遮断する
  - SIM カードが挿入されているのであれば抜き、プロバイダーとの契約を終了する
  - 無線 LAN の設定を削除する
  - 接続しているクラウドのデバイス証明書を削除・無効にすることでクラウドに接続できなくする
- 「3.2. Armadillo の初期化と ABOS のアップデート」の手順にしたがって初期化を行う
  - インストールディスクは、 blkdiscard コマンドを用いて eMMC の GPP を含む全てのパーティション内のデータを消去しています
- 物理的に起動できなくする
- Armadillo Twin をご利用の場合は、Armadillo Twin 上で当該デバイスの廃棄手続きを行うか、Armadillo Twin を解約する際に廃棄する旨を申告する
  - 実際の手続きなど詳細は Armadillo Twin お問い合わせフォーム [https://apps.armadillo-twin.com/ja/inquiry]からお問い合わせください
- Armadillo を物理的に破壊する
  - SD ブートを無効化していてインストールディスクが適用できない場合や、情報吸い出しのリスクをより下げたい場合は、 eMMC やセキュアエレメント等の記憶媒体を物理的に破壊してください
  - 詳細な破壊箇所や破壊方法については、弊社までお問い合わせください

10. 応用編

本章では、ここまでの内容で紹介しきれなかった、より細かな Armadillo の設定方法や、開発に役立つヒントなどを紹介します。

各トピックを羅列していますので、目次の節タイトルからやりたいことを探して辞書的にご使用ください。

10.1. ログインできるユーザーについて

初期状態では「root」ユーザーと、一般ユーザーである「atmark」ユーザーが存在しますが、「atmark」ユーザーは初期状態ではロックされています。ロックを解除するためには、「atmark」ユーザーのパスワードを設定した initial_setup.swu をインストールするか、CUI からパスワードを直接設定します。

CUI からパスワードを直接設定する手順では、以下のように一度「root」ユーザーでログインしてから passwd atmark コマンドで「atmark」ユーザーのパスワードを設定します。

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

:（省略）

armadillo login: atmark
Password: ❺
Welcome to Alpine!
```

❶ atmark ユーザーのパスワード変更コマンドです。

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

❸ 新しいパスワードを再入力します。

❹ パスワードファイルを永続化します。

❺ 設定したパスワードでログインすることができます。

10.2. swupdate を使用してアップデートする

10.2.1. swupdate で可能なアップデート

swupdate を実行する目的としては以下が考えられます。

a. コンテナをアップデートしたい

開発したコンテナのアップデートが可能です。

b. ユーザーデータディレクトリや Armadillo Base OS のファイルを差分アップデートしたい

ユーザーデータをアップデートする場合は、以下のディレクトリを更新します。

・ /var/app/rollback/volumes

ユーザーディレクトリについては「7.3.4.1. 電源を切っても保持されるディレクトリ(ユーザーデータディレクトリ)」を参照してください。

SWUpdate による /var/app/volumes の更新は推奨しません。

/var/app/volumes が二面化されていないため、書込みの途中で問題が発生した場合、不完全な書込みになる恐れがあります。

また、アップデート中にアプリケーションがそのデータにアクセスすると、書込み中のデータにアクセスすることになります。

/var/app/volumes のデータに対して更新が必要な場合、SWUpdate では /var/app/rollback/volumes に更新するデータを書き込んでください。その後、次回起動時にアプリケーション側から /var/app/rollback/volumes に書き込んだデータを /var/app/volumes に移動するようにしてください。

c. Armadillo Base OS を一括アップデートしたい

アットマークテクノがリリースする Armadillo Base OS の機能追加、更新、セキュリティパッチの追加が可能です。

d. ブートローダーをアップデートしたい

アットマークテクノがリリースするブートローダーのアップデートが可能です。

「2.1.4. Armadillo Base OS とは」で示すように、Armadillo Base OS は OS・ブートローダー・コンテナ部分の安全性を担保するため二面化しています。

それにより、万が一アップデートに失敗した場合でも起動中のシステムに影響ありません。

以降、それぞれの目的ごとに swupdate によるアップデートの流れを示します。

・ a, b の場合

「10.2.2. コンテナのアップデート、ユーザーデータディレクトリや Armadillo Base OS の差分アップデート」を参照してください。

・ c の場合

「10.2.3. Armadillo Base OS の一括アップデート」を参照してください。

・ d の場合

「10.2.4. ブートローダーのアップデート」を参照してください。

10.2.2. コンテナのアップデート、ユーザーデータディレクトリや Armadillo Base OS の差分アップデート

以下にアップデートの流れを示します。

ここでは、boot して起動中の面を A 面、アップデート先の面を B 面とします。

1. Armadillo Base OS を B 面へコピー

   Armadillo Base OS を B 面にコピーする流れを「図 10.1. Armadillo Base OS を B 面にコピー」に示します。

   A 面と B 面の Armadillo Base OS が同期しているか確認します。

   同期していない場合、 A 面の Armadillo Base OS を B 面にコピーします。

   同期している場合はコピーしません。

   `swdesc_option version` で指定するバージョンの書き方については「10.3.1. インストールバージョンを指定する」を参照してください。

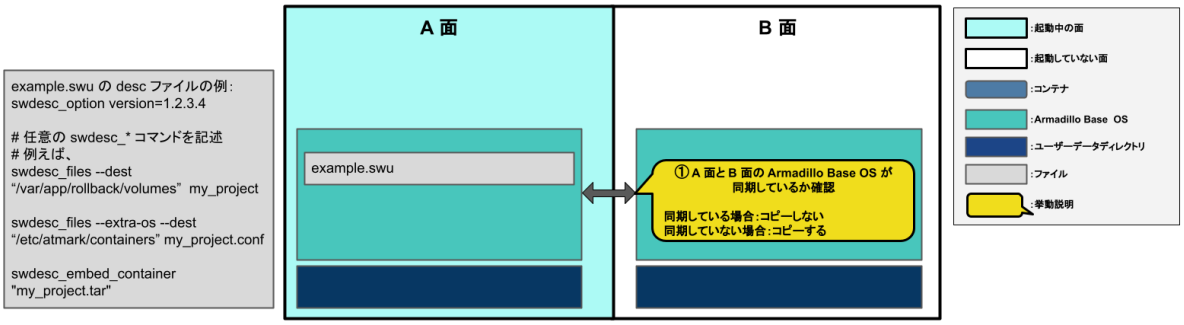


図 10.1 Armadillo Base OS を B 面にコピー

2. コマンドを順番に実行

   「図 10.2. desc ファイルに記述した swudesc_* コマンドを実行」に示すように、desc ファイルに記述した順番に従って swudesc_* コマンドを実行します。

   「10.3.1. インストールバージョンを指定する」に示すように、swdesc_* コマンドによって Armadillo Base OS に対して書き込みをする場合は --extra-os オプションをつけてください。

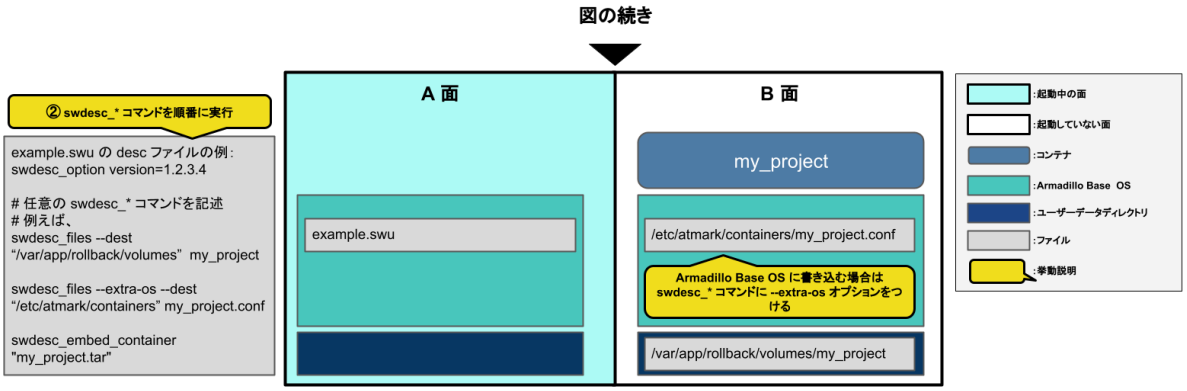


図 10.2 desc ファイルに記述した swudesc_* コマンドを実行

swudesc_* コマンドの種類を「表 10.1. swudesc_* コマンドの種類」に示します。

表 10.1 swudesc_* コマンドの種類

おおまかな機能	コマンド名	説明
ファイル転送 参照先：「10.3.2. Armadillo へファイルを転送する」	swdesc_files	指定したファイルをアップデート先の環境にコピー
	swdesc_tar	指定した tar アーカイブをアップデート先の環境に展開してコピー
コマンド実行 参照先：「10.3.3. Armadillo 上で任意のコマンドを実行する」	swdesc_command	指定したコマンドをアップデート先の環境で実行
	swdesc_script	指定したスクリプトをアップデート先の環境で実行
ファイル転送 + コマンド実行 参照先：「10.3.4. Armadillo にファイルを転送し、そのファイルをコマンド内で使用する」	swdesc_exec	指定したファイルをアップデート先の環境にコピーした後、そのファイル名を"$1"としてコマンドを実行
起動中の面に対してコマンド実行（非推奨） 参照先：「10.3.5. 動作中の環境でのコマンドの実行」	swdesc_command_nochroot	指定したコマンドを起動中の環境で実行
	swdesc_script_nochroot	指定したスクリプトを起動中の環境で実行
起動中の面に対してファイル転送 + コマンド実行（非推奨） 参照先：「10.3.5. 動作中の環境でのコマンドの実行」	swdesc_exec_nochroot	指定したファイルを起動中の環境にコピーした後、そのファイル名を"$1"としてコマンドを実行
コンテナイメージの転送 参照先：「10.3.6. Armadillo にコンテナイメージを転送する」	swdesc_embed_container	SWU ファイルに含まれるコンテナイメージの tar アーカイブをアップデート先の環境に展開
	swdesc_pull_container	アップデート先の環境でコンテナイメージをダウンロード
	swdesc_usb_container	SWU ファイルに含めない形で用意したコンテナイメージの tar アーカイブをアップデート先の環境に展開

3. アップデート完了後の挙動

   デフォルトではアップデート後に再起動（ POST_ACTION=reboot ）が行われます。

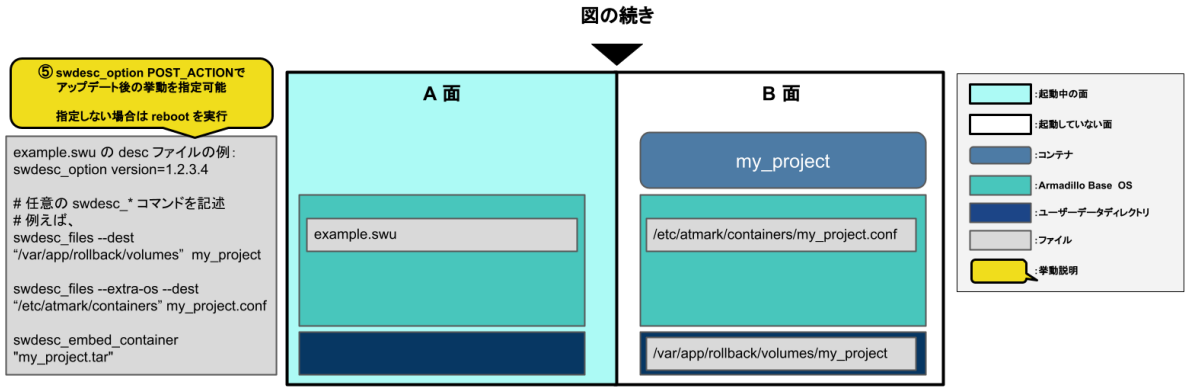


図 10.3 アップデート完了後の挙動

アップデート後の挙動を変更するには desc ファイルに swdesc_option POST_ACTION を追加してください。

swdesc_option POST_ACTION に指定できる挙動の種類を「表 10.2. アップデート完了後の挙動の種類」に示します。

表 10.2 アップデート完了後の挙動の種類

オプション名	説明
container	アップデート後にコンテナのみを再起動 (ただし、アップデート時に --extra_os オプションを指定したコマンドが実行された場合は reboot になる)
poweroff	アップデート後にシャットダウン
reboot	アップデートの後に再起動
wait	アップデート後に再起動は行われず、次回起動時に B 面に切り替わる

swdesc_option POST_ACTION の詳細は「10.3.10. SWUpdate 実行中/完了後の挙動を指定する」を参照してください。

4. B 面への切り替え

「図 10.4. B 面への切り替え」に示すように、正常にアップデートが行われると、次回起動時に B 面に切り替わります。

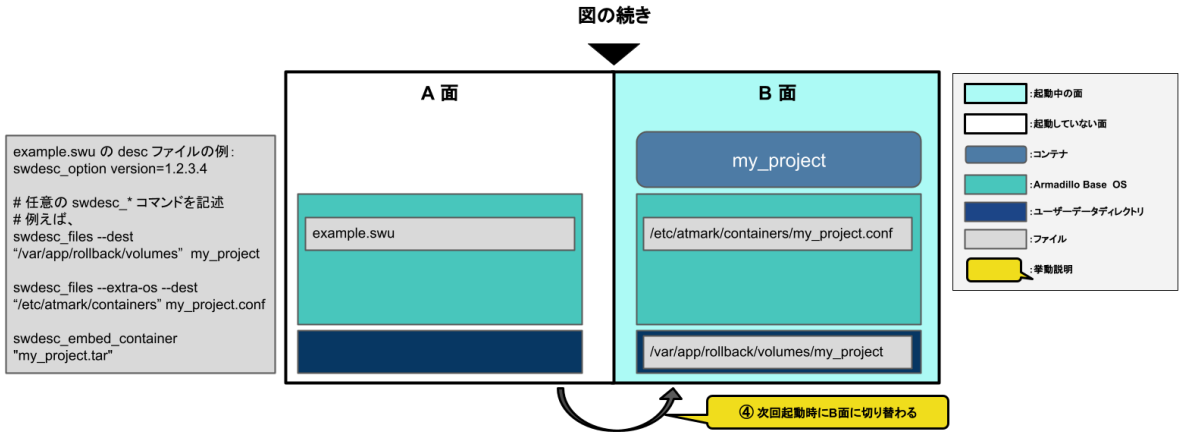


図 10.4 B 面への切り替え

5. desc ファイルの書き方の例

   下記に SWUpdate を用いたアップデートの例を示します。

   - コンテナイメージとコンテナ自動設定ファイルのアップデート：「10.8.1.10. コンテナイメージを SWUpdate で転送する」
   - sshd の設定：「10.3.11.1. 例: sshd を有効にする」

10.2.3. Armadillo Base OS の一括アップデート

アップデートの流れを示します。

ここでは、boot して起動中の面を A 面、アップデート先の面を B 面とします。

1. Armadillo Base OS とアップデート後に保持するファイルを B 面へコピー

   Armadillo Base OS とアップデート後に保持するファイルを B 面にコピーする流れを「図 10.5. Armadillo Base OS とファイルを B 面にコピー」に示します。

   「10.3.1. インストールバージョンを指定する」に示すように、Armadillo Base OS の tar アーカイブを展開する swdesc_tar コマンドに --base-os オプションをつけてください。

   `swdesc_option version` で指定するバージョンの書き方については「10.3.1. インストールバージョンを指定する」を参照してください。

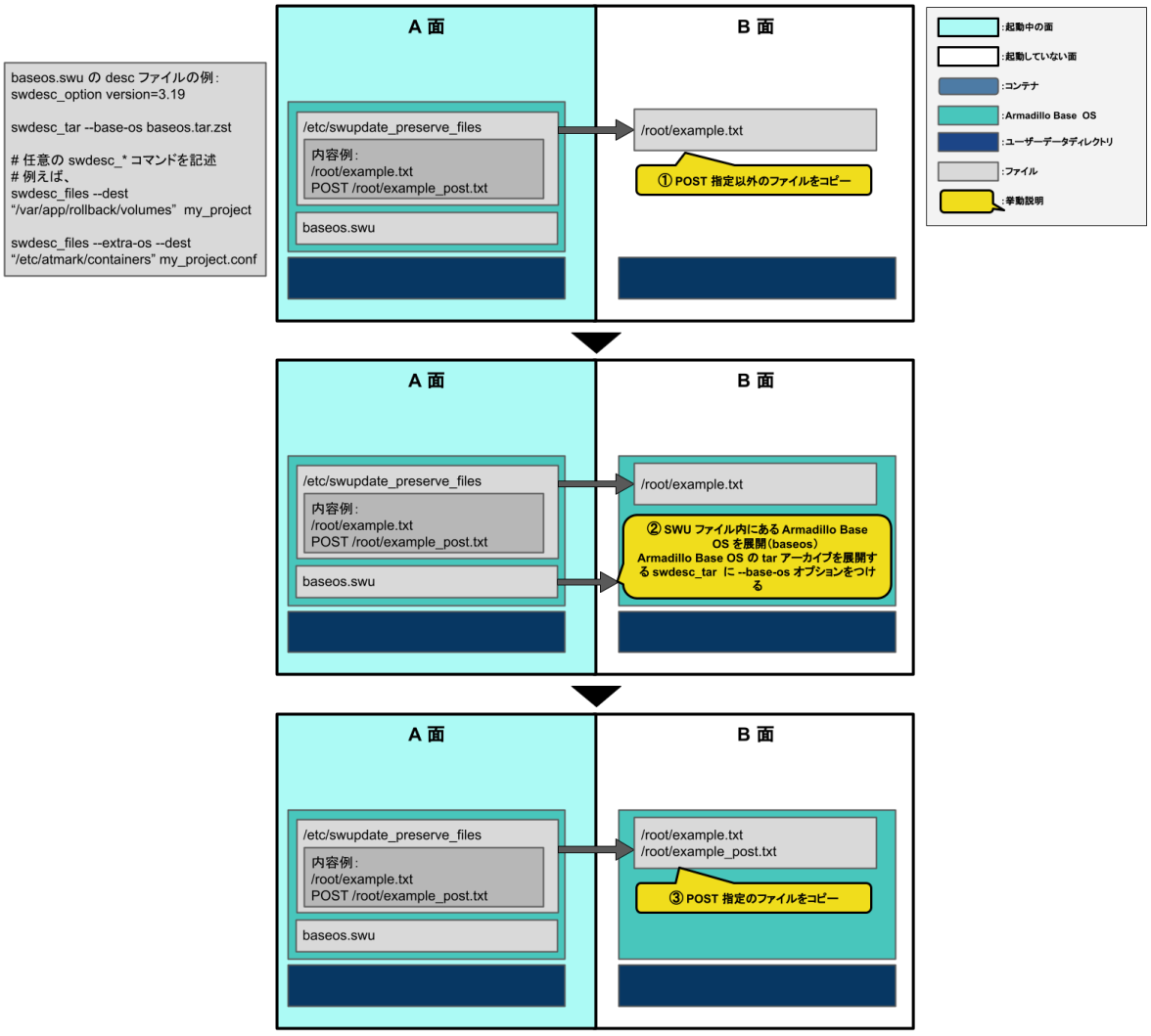


図 10.5 Armadillo Base OS とファイルを B 面にコピー

1. `/etc/swupdate_preserve` に記載された POST 指定以外のファイルを B 面にコピーします。
2. SWU ファイル内にある Armadillo Base OS を B 面に展開します。
3. `/etc/swupdate_preserve` に記載された POST 指定のファイルを B 面にコピーします。

`/etc/swupdate_preserve` への追記方法については「10.4. swupdate_preserve_files について」と「8.4.1. /etc/swupdate_preserve_file への追記」を参照してください。

2. コマンドを順番に実行

「図 10.6. desc ファイルに記述した swudesc_* コマンドを実行」に示すように、desc ファイルに記述した順番に従って swudesc_* コマンドを実行します。

「10.3.1. インストールバージョンを指定する」に示すように、swdesc_* コマンドによって Armadillo Base OS に対して書き込みをする場合は --extra-os オプションをつけてください。

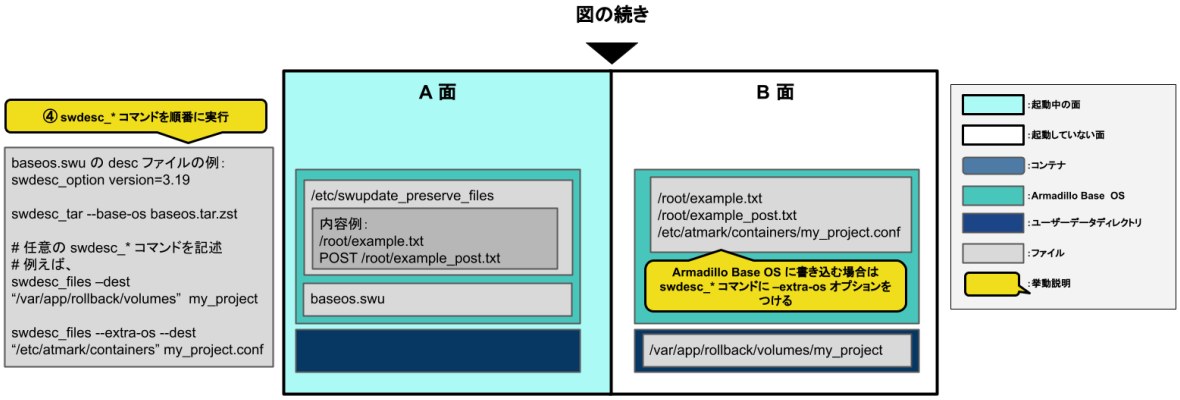


図 10.6 desc ファイルに記述した swudesc_* コマンドを実行

swudesc_* コマンドの種類を「表 10.3. swudesc_* コマンドの種類」に示します。

表 10.3 swudesc_* コマンドの種類

おおまかな機能	コマンド名	説明
ファイル転送 参照先：「10.3.2. Armadillo へファイルを転送する」	swdesc_files	指定したファイルをアップデート先の環境にコピー
	swdesc_tar	指定した tar アーカイブをアップデート先の環境に展開してコピー
コマンド実行 参照先：「10.3.3. Armadillo 上で任意のコマンドを実行する」	swdesc_command	指定したコマンドをアップデート先の環境で実行
	swdesc_script	指定したスクリプトをアップデート先の環境で実行
ファイル転送 + コマンド実行 参照先：「10.3.4. Armadillo にファイルを転送し、そのファイルをコマンド内で使用する」	swdesc_exec	指定したファイルをアップデート先の環境にコピーした後、そのファイル名を"$1"としてコマンドを実行
起動中の面に対してコマンド実行（非推奨） 参照先：「10.3.5. 動作中の環境でのコマンドの実行」	swdesc_command_nochroot	指定したコマンドを起動中の環境で実行
	swdesc_script_nochroot	指定したスクリプトを起動中の環境で実行
起動中の面に対してファイル転送 + コマンド実行（非推奨） 参照先：「10.3.5. 動作中の環境でのコマンドの実行」	swdesc_exec_nochroot	指定したファイルを起動中の環境にコピーした後、そのファイル名を"$1"としてコマンドを実行
コンテナイメージの転送 参照先：「10.3.6. Armadillo にコンテナイメージを転送する」	swdesc_embed_container	SWU ファイルに含まれるコンテナイメージの tar アーカイブをアップデート先の環境に展開
	swdesc_pull_container	アップデート先の環境でコンテナイメージをダウンロード
	swdesc_usb_container	SWU ファイルに含めない形で用意したコンテナイメージの tar アーカイブをアップデート先の環境に展開
ブートローダーの更新 参照先：「10.3.7. Armadillo のブートローダーを更新する」	swdesc_boot	SWU ファイルに含まれるブートローダーをアップデート先の環境に展開

3. アップデート完了後の挙動

   デフォルトではアップデート後に再起動（ POST_ACTION=reboot ）が行われます。

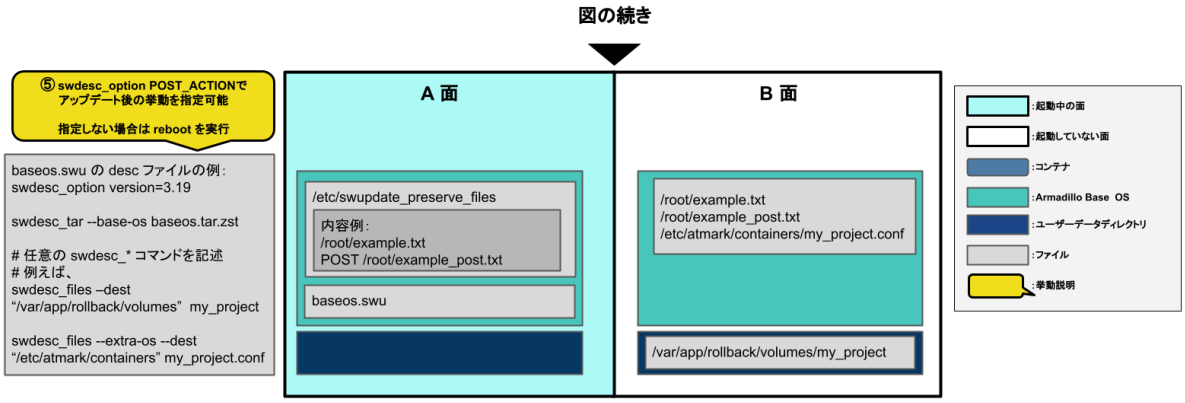


図 10.7 アップデート完了後の挙動

アップデート後の挙動を変更するには desc ファイルに `swdesc_option POST_ACTION` を追加してください。

`swdesc_option POST_ACTION` に指定できる挙動の種類を「表 10.4. アップデート完了後の挙動の種類」に示します。

表 10.4 アップデート完了後の挙動の種類

オプション名	説明
poweroff	アップデート後にシャットダウン
reboot	アップデートの後に再起動
wait	アップデート後に再起動は行われず、次回起動時に B 面に切り替わる

`swdesc_option POST_ACTION` の詳細は「10.3.10. SWUpdate 実行中/完了後の挙動を指定する」を参照してください。

4. B 面への切り替え

「図 10.8. B 面への切り替え（component=base_os)」に示すように、正常にアップデートが行われると、次回起動時に B 面に切り替わります。

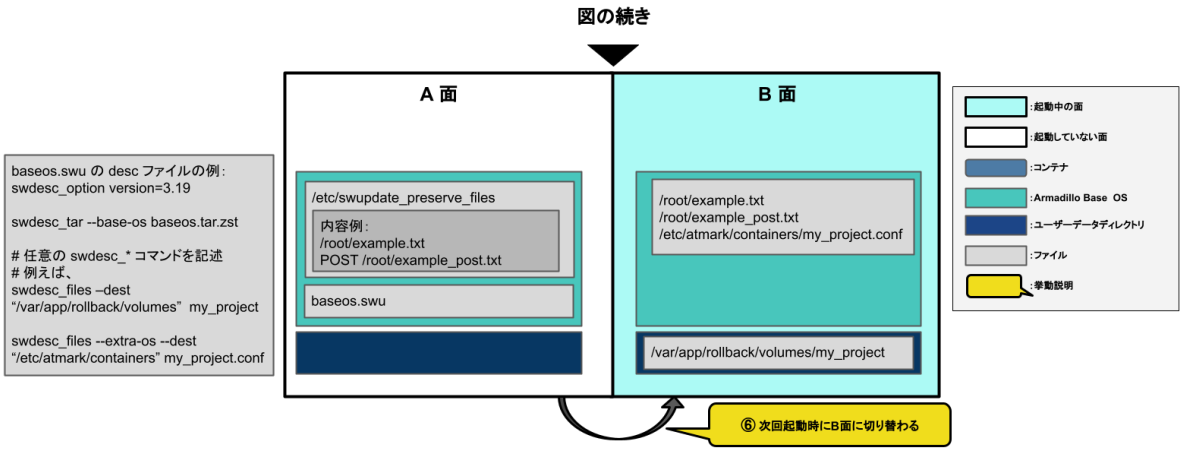


図 10.8 B 面への切り替え（component=base_os)

5. desc ファイルの書き方の例

下記に SWUpdate を用いたアップデートの例を示します。

- Armadillo Base OS のアップデート：「10.3.11.2. 例: Armadillo Base OS アップデート」
- Alpine Linux ルートファイルシステムのアップデート：「10.19.3. Alpine Linux ルートファイルシステムをビルドする」

10.2.4. ブートローダーのアップデート

swdesc_* コマンドには、swdesc_boot を指定してください。

swdesc_boot については「10.3.7. Armadillo のブートローダーを更新する」を参照してください。

ブートローダーのアップデートの流れは以下の通りです。

- desc ファイルに swdesc_boot がある場合

  SWU ファイルに含まれるブートローダーを B 面に書き込む

- desc ファイルに swdesc_boot がない場合

  A 面のブートローダーを B 面にコピーする

下記に SWUpdate を用いたアップデートの例を示します。

- ブートローダーのアップデート：「10.19.1. ブートローダーをビルドする」

10.2.5. swupdate がエラーする場合の対処

SWU イメージのインストール動作は、「7.3.3.2. SWU イメージとは」で述べたように swupdate が実行します。mkswu で作成した SWU イメージの内容が適切でなかったり、あるいは、ストレージの空き容量が不足していたりするなど、いくつかの理由で swupdate のインストール動作が失敗することがあります。インストールに失敗すると、swupdate は /var/log/messages にエラーメッセージのログを残しますので、エラーメッセージを見ると、エラーの内容・原因が分かります。

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

10.3. mkswu の .desc ファイルを編集する

mkswu で SWU イメージを生成するためには、 desc ファイルを正しく作成する必要があります。以下では、 desc ファイルの記法について紹介します。

10.3.1. インストールバージョンを指定する

```
swdesc_option component=<component>
swdesc_option version=<version>
か
swdesc_xxx --version <component> <version> [options]
```

- <component>は以下のどれかにしてください (デフォルトでは .desc ファイルのファイル名を使います)

1. base_os: rootfs (Armadillo Base OS)を最初から書き込む時に使います。現在のファイルシステムは保存されないです。

   この場合、/etc/swupdate_preserve_files に載ってるファイルのみをコピーして新しい base OS を展開します。

   この component がないと現在の rootfs のすべてがコピーされます。

   swdesc_tar コマンドで rootfs (Armadillo Base OS) の tar アーカイブを展開する時に、--base-os オプションをつけることで component に base_os を指定したときと同じ動作となります。

2. extra_os.<文字列>: rootfs の変更を行う時に使います。<文字列> には任意の文字列を指定します。

   rootfs を変更を行う時に使います。 swdesc_* コマンドに --extra-os オプションを追加すると、 component に自動的に extra_os. を足します。

3. <文字列> (コンテナの名前などの任意の文字列）: rootfs の変更がないときに使います。

   この component を使うと rootfs の変更ができませんのでご注意ください。

・ アップデートを行う際にこのバージョンと現在のバージョンを比べてアップデートの判断を行います。

<component> がまだインストールされてなかった時や <version> が上がる時にインストールします。

デフォルトではダウングレードはできませんが、 --install-if=different オプションを追加することで <version> が変わる際にインストール可能になります。

アップデートの一部をインストールすることもありますので、複数の component で管理し、いくつかの古いバージョンに対応するアップデートも作成可能です。

・ バージョンの指定方法

swdesc_option version で指定可能なバージョンのフォーマットは以下の 2 種類があります。

・ x[.y[.z[-t]]]

  x, y, z にはそれぞれ 0 〜 2147483647 の整数を適用してください。t には任意のアルファベットまたは 0 〜 147483647 の整数を適用してください。

  成功例は以下です：

  ・ 1

  ・ 1.2.3

  ・ 1.2.3-4

  ・ 1.2.3-abc.4

  ・ 1.2.3-a.b.c.4

  失敗例は以下です：

  ・ 2147483648

x には 0 〜 2147483647 の整数を適用してください。

- 1.2.3-a.2147483648

  t には 0 〜 2147483647 の整数を適用してください。

- 1.2.3-abc123

  t には 数字とアルファベットを混在しないでください。1.2.3-abc.123 ならば可能です。

- a.2.3

  x にはアルファベットではなく 0 〜 2147483647 の整数を適用してください。

- 1.1.1.1-a

  x[.y[.z[-t]]]の形式で書いてください。

- x.y.z.t

  x, y, z, t にはそれぞれ 0 〜 65535 の整数を適用してください。

  成功例は以下です：

  - 1.2.3.4
  - 65535.65535.65535.65535
  - 65535.2.3.4

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

  失敗例は以下です：

  - 65536.2.3.4

    x には 0 〜 65535 の整数を適用してください。

  - 1.2.3.a

    t にはアルファベットではなく 0 〜 65535 の整数を適用してください。

  - 1.2.3.4.5

    x.y.z.t の形式で書いてください。

10.3.2. Armadillo へファイルを転送する

- `swdesc_tar` と `swdesc_files` でファイルを転送します。

```
swdesc_tar [--dest <dest>] [--preserve-attributes] <tar_file>
swdesc_files [--dest <dest>] [--basedir <basedir>] [--preserve-attributes] ¥
             <file> [<more files>]
```

swdesc_tar の場合、予め用意されてある tar アーカイブをこのままデバイスで展開します。

--dest <dest> で展開先を選ぶことができます。デフォルトは / （--extra-os を含め、バージョンの component は base_os か extra_os.* の場合）か /var/app/rollback/volumes/ （それ以外の component)。後者の場合は /var/app/volumes と /var/app/rollback/volumes 以外は書けないので必要な場合に --extra-os を使ってください。

--preserve-attributes を指定しない場合はファイルのオーナー、モード、タイムスタンプ等が保存されませんので、必要な場合は設定してください。バージョンが base_os の場合は自動的に設定されます。

swdesc_files の場合、mkswu がアーカイブを作ってくれますが同じ仕組みです。

--basedir <basedir> でアーカイブ内のパスをどこで切るかを決めます。

- 例えば、swdesc_files --extra-os --basedir /dir /dir/subdir/file ではデバイスに /subdir/file を作成します。
- デフォルトは <file> から設定されます。ディレクトリであればそのまま basedir として使います。それ以外であれば親ディレクトリを使います。

10.3.3. Armadillo 上で任意のコマンドを実行する

- swdesc_command や swdesc_script でコマンドを実行します。

```
swdesc_command <command> [<more commands>]
swdesc_script <script>
```

アップデート先の環境でコマンドやスクリプトファイルを実行します。

バージョンの component は base_os と extra_os 以外の場合、 /var/app/volumes と /var/app/rollback/volumes 以外は変更できないのでご注意ください。

コマンドの実行が失敗した場合、アップデートも失敗します。

10.3.4. Armadillo にファイルを転送し、そのファイルをコマンド内で使用する

- swdesc_exec でファイルを配り、コマンド内でそのファイルを使用します。

```
swdesc_exec <file> <command>
```

swdesc_command と同じくコマンドを実行しますが、<file> を先に転送してコマンド内で転送したファイル名を"$1"として使えます。

10.3.5. 動作中の環境でのコマンドの実行

本節で紹介する swdesc_command_nochroot、swdesc_script_nochroot、swdesc_exec_nochroot は基本的に使用することはありません。

swdesc_command、swdesc_script、swdesc_exec をご使用ください。

- swdesc_command_nochroot, swdesc_script_nochroot, swdesc_exec_nochroot は アップデート先の環境ではなく動作中の環境でコマンドを実行します。

  使い方は「10.3.2. Armadillo へファイルを転送する」と「10.3.4. Armadillo にファイルを転送し、そのファイルをコマンド内で使用する」に示した nochroot なしのバージョンと同じです。

  アップデート先の環境は /target にマウントされるので、nochroot のコマンドを用いてアップデート先の環境に対してアクセスすることは可能です。

  しかし、その方法によるアップデート先の環境に対するコマンドの実行は nochroot なしのコマンドでは実現できない特殊な場合にのみ行ってください。

nochroot コマンドは確認を一切しないため、Armadillo が起動できない状態になる可能性もあります。充分にご注意ください。

例が必要な場合は /usr/share/mkswu/examples/firmware_update.desc を参考にしてください。

10.3.6. Armadillo にコンテナイメージを転送する

- swdesc_embed_container, swdesc_usb_container, swdesc_pull_container で予め作成したコンテナを転送します。

```
swdesc_embed_container <container_archive>
swdesc_usb_container <container_archive>
swdesc_pull_container <container_url>
```

  例は「10.8.1.9. リモートリポジトリにコンテナを送信する」、「10.8.1.10. コンテナイメージを SWUpdate で転送する」を参考にしてください。

10.3.7. Armadillo のブートローダーを更新する

- swdesc_boot でブートローダーを更新します。

```
swdesc_boot <boot image>
```

  このコマンドだけはバージョンは自動的に設定されます。

10.3.8. SWU イメージの設定関連

コマンドの他には、設定変数もあります。以下の設定は /home/atmark/mkswu/mkswu.conf に設定できます。

- DESCRIPTION="<text>": イメージの説明、ログに残ります。
- PRIVKEY=<path>, PUBKEY=<path>: 署名鍵と証明書
- PRIVKEY_PASS=<val>: 鍵のパスワード（自動用）

  openssl の Pass Phrase をそのまま使いますので、pass:password, env:var や file:pathname のどれかを使えます。pass や env の場合他のプロセスに見られる恐れがありますので file をおすすめします。
- ENCRYPT_KEYFILE=<path>: 暗号化の鍵

10.3.9. Armadillo 上のコンテナイメージと自動起動用 conf ファイルを削除する

以下のオプションも mkswu.conf に設定できますが、.desc ファイルにも設定可能です。swdesc_option で指定することで、誤った使い方をした場合 mkswu の段階でエラーを出力しますので、必要な場合は使用してください。

- swdesc_option CONTAINER_CLEAR: インストールされているコンテナと /etc/atmark/containers/*.conf をすべて削除します。

  このオプションは簡単な初期化と考えてください。通常の運用では、不要になったイメージは自動的に削除されますのでこのオプションを設定する必要はありません。

10.3.10. SWUpdate 実行中/完了後の挙動を指定する

以下のオプションは Armadillo 上の /etc/atmark/baseos.conf に、例えば MKSWU_POST_ACTION=xxx として設定することができます。

その場合に swu に設定されなければ /etc の設定で実行されますので、アットマークテクノが用意している Base OS のアップデートでも動作の変更は可能です。swu に特定のオプションが設定された場合は設定されたオプションが優先されますので、一時的な変更も可能です。

- swdesc_option POST_ACTION=container: コンテナのみのアップデート後に再起動を行いません。コンテナの中身だけをアップデートする場合、Armadillo-900 開発セットを再起動せずにコンテナだけを再起動させます。
- swdesc_option POST_ACTION=poweroff: アップデート後にシャットダウンを行います。
- swdesc_option POST_ACTION=wait: アップデート後に自動的に再起動は行われず、次回起動時にアップデートが適用されます。
- swdesc_option POST_ACTION=reboot: デフォルトの状態に戻します。アップデートの後に再起動します。
- swdesc_option NOTIFY_STARTING_CMD="command", swdesc_option NOTIFY_SUCCESS_CMD="command", swdesc_option NOTIFY_FAIL_CMD="command": アップデートをインストール中、成功した場合と失敗した場合に実行されるコマンドです。

コマンドを実行する事で、アプリケーションやユーザーにアップデートを知らせることができます。

LED で知らせる例を /usr/share/mkswu/examples/enable_notify_led.desc に用意してあります。

10.3.11. desc ファイル設定例

10.3.11.1. 例: sshd を有効にする

/usr/share/mkswu/examples/enable_sshd.desc を参考にします。

desc ファイルを編集する必要がありませんが自分の公開鍵を指定された場所に配置してください。

```
[ATDE ~/mkswu]$ cp -r /usr/share/mkswu/examples/enable_sshd* .
[ATDE ~/mkswu]$ cat enable_sshd.desc
swdesc_option component=extra_os.sshd version=1

# add your public key in enable_sshd/root/.ssh/authorized_keys
if [ -z "$SWDESC_TEST" ]; then
    grep -qE '^ssh-' enable_sshd/root/.ssh/authorized_keys ¥
        || error "Add your keys in enable_sshd/root/.ssh/authorized_keys"
fi
swdesc_files --dest /root enable_sshd/root ❶

swdesc_command "ssh-keygen -A" ¥ ❷
        "rc-update add sshd" ❸
[ATDE ~/mkswu]$ cp ~/.ssh/id_rsa.pub ¥
                        enable_sshd/root/.ssh/authorized_keys ❹
[ATDE ~/mkswu]$ mkswu enable_sshd.desc ❺
Enter pass phrase for /home/atmark/mkswu/swupdate.key:
enable_sshd.swu を作成しました。
```

❶ 自分の公開鍵を転送します。デフォルトのオプションなので enable_sshd/root ディレクトリの中身をこのまま /root に転送されます。

❷ 再起動する度に新しいサーバーの鍵が変わらないように、アップデートの時に一回作成します。

❸ サービスを有効にします。

❹ 自分の公開鍵を指定された場所に配置します。

❺ イメージを作成します。パスワードは証明鍵のパスワードです。

10.3.11.2. 例: Armadillo Base OS アップデート

ここでは、「10.19. 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。

10.3.11.3. 例: swupdate_preserve_files で Linux カーネル以外の Armadillo-900 開発セット向けのイメージをインストールする方法

Armadillo-900 開発セット 向けのアップデートイメージに Linux カーネルが含まれています。

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

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

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

❷ 変更した設定ファイルを保存します

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

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

❶ スクリプトの内容確認する場合は /usr/share/mkswu/examples/update_preserve_files.sh を参照してください。

Armadillo Base OS のカーネルを再び使用したい場合は同じスクリプトの --del オプションで行を削除することができます。

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

10.4. swupdate_preserve_files について

extra_os のアップデートで rootfs にファイルを配置することができますが、次の OS アップデートの際に削除される可能性があります。

デフォルトでは、 /etc/atmark と、 swupdate 、 sshd やネットワークの設定を保存しますがそれ以外はコピーされてません。

そうでないファイルを更新する際には /etc/swupdate_preserve_files に記載します。「10.3.11.3. 例: swupdate_preserve_files で Linux カーネル以外の Armadillo-900 開発セット 向けのイメージをインストールする方法」 を参考にしてください。

コピーのタイミングによって、以下のどれかを使用してください：

1. 単にファイルを記載する

   この場合、アップデートする前にファイルをコピーします。 baseos のイメージと同じ swu にアップデートしたいファイルを記載していても、このファイルが Armadillo Base OS に含まれないのであれば問題なくアップデートできます。

   例: echo "/root/.profile" >> /etc/swupdate_preserve_files

2. POST のキーワードの後に記載する

   この場合、アップデートの最後でコピーします。 Armadillo Base OS に含まれてるファイルであれば、インストール前にコピーしても保存されないのでコピーのタイミングをずらします。

   そのコピーが最後に行われるので、同じアップデートでファイルの変更ができません。アップデートを別けて、 baseos のイメージをインストールしてからこのファイルを更新することができます。

   例: echo "POST /etc/conf.d/podman-atmark" >> /etc/swupdate_preserve_files

10.5. SWU イメージの内容の確認

mkswu --show [file.swu] で SWU イメージの内容を確認することができます。

出力は desc ファイルに似ていますが、そのまま desc ファイルとして利用できませんので確認用としてお使いください。

```
[ATDE ~/mkswu]$ mkswu --show enable_sshd.swu
enable_sshd.swu

# built with mkswu 4.1
```

```
swdesc_files --dest /root enable_sshd/root
  --version extra_os.sshd 1
  (encrypted)

swdesc_command ssh-keygen -A && rc-update add sshd default
  --version extra_os.sshd 1
```

10.6. SWUpdate と暗号化について

mkswu --init の時に暗号化を有効にする場合は AES でファイルを暗号化します。

現在使われてる SWUpdate の暗号化はコマンドやメタデータを含む sw-description ファイルは暗号化されてません。そのため、通信の暗号化（HTTPS で送信するなど）を使うことを推奨します。

10.7. SWUpdate の署名鍵と証明書の更新

mkswu で SWU イメージを生成する際に SWU イメージ内の sw-description という命令ファイルを ATDE にある署名鍵と証明書を用いて署名します。swupdate を実行する際には、署名に使用した証明書が /etc/swupdate.pem に含まれているかを確認します。

その署名を確認できなかった場合、SWU イメージをインストールできないので、Armadillo にある証明書を管理しなければなりません。

また、暗号鍵管理のガイドラインとしては定期的に鍵を交換することが強く推奨されています。mkswu --init を実行した際に 1 つだけ署名鍵と証明書を生成しましたが、ここでは他の署名鍵および証明書の生成と Armadillo 側の管理の方法について説明します。

10.7.1. 署名鍵と証明書の追加

署名鍵と証明書の生成は以下の様に、 mkswu --genkey で行います。

```
[ATDE ~]$ mkswu --genkey
/home/atmark/mkswu/swupdate.key はすでに存在します。新しい鍵を作成しますか？ [Y/n] ❶
証明書のコモンネーム(一般名)を入力してください: [COMMON_NAME] ❷
署名鍵 /home/atmark/mkswu/swupdate-2.key と証明書 swupdate-2.pem を作成します。
Generating an EC private key
writing new private key to '/home/atmark/mkswu/swupdate-2.key.tmp'
Enter PEM pass phrase: ❸
Verifying - Enter PEM pass phrase:
 -----
/home/atmark/mkswu/swupdate-2.pem をコンフィグファイルに追加します。
/home/atmark/mkswu/swupdate-2.pem が次のアップデートをインストールするときに転送されます。
インストールされてから現在の鍵を /home/atmark/mkswu/mkswu.conf から外してください。
```

図 10.9 mkswu --genkey で署名鍵と証明書を追加する

❶ Enter キーを押下します。

❷ [COMMON_NAME] には会社や製品が分かる任意の名称を入力してください。任意ではありますが、一つ目の鍵と違う名前にすることを推奨します。

❸ 署名鍵を保護するパスフレーズを 2 回入力します。

このコマンドを実行すると以下の文字列が ~/mkswu/mkswu.conf に追加されます。

```
# extra swupdate certificate. Remove the old one and use new
# PRIVKEY after having installed an update with this first
PUBKEY="$PUBKEY,$CONFIG_DIR/swupdate-2.pem" ❶
# remove "NEW_" to use
NEW_PRIVKEY="$CONFIG_DIR/swupdate-2.key" ❷
# This controls if we should update certificates on device, and can be
# removed once all devices have been updated to only allow new certificate
UPDATE_CERTS=yes
```

図 10.10 mkswu --genkey により mkswu.conf に追加された内容

❶ PUBKEY の値に生成した証明書のパスが追加されます。UPDATE_CERTS=yes を設定して生成した SWU イメージは、PUBKEY の値にリストされている証明書を全て Armadillo にインストールします。PUBKEY にリストされてない証明書、またはアットマークテクノ側で管理している証明書以外は全て削除されます。

❷ 新しい署名鍵のパスです。この段階では未使用になります。

この状態で任意 (POST_ACTION=container 以外) の SWU イメージを生成し、インストールすると Armadillo の /etc/swupdate.pem に PUBKEY にリストされている両方の証明書がインストールされます。Armadillo にインストールされている鍵は、abos-ctrl certificates list で確認できます：

```
[armadillo ~]# abos-ctrl certificates list
- atmark-2
- atmark-3
- swupdate-2.pem: [mkswu --genkey で入力した COMMON NAME] ❶
- swupdate.pem: [mkswu --init で入力した COMMON NAME] ❷
```

図 10.11 新しい証明書が Armadillo に追加されていることを確認する

❶ 追加した証明書のコモンネーム

❷ mkswu --init 時に生成した証明証のコモンネーム。当時の mkswu のバージョンによっては swupdate.pem が記載されてない可能性があります。

この状態で新しい鍵を使えるようになります。

10.7.2. 署名鍵と証明書の削除

上記のように Armadillo に複数の署名鍵を使用できる状態になった場合は証明に使う署名鍵と証明書を切り替えることができます。

mkswu.conf の PUBKEY の値から一つ目の値を削除し、 PRIVKEY に新しい値を設定し、必要であれば UPDATE_CERTS=yes を記述します。

```
[ATDE ~] tail -n 3 ~/mkswu/mkswu.conf
PUBKEY="$CONFIG_DIR/swupdate-2.pem"
```

```
PRIVKEY="$CONFIG_DIR/swupdate-2.key"
UPDATE_CERTS=yes
```

図 10.12 署名鍵と証明書を削除する設定

署名鍵の追加と同じく、この状態で SWU イメージを生成し Armadillo にインストールすると前の証明書が削除されます。

こちらも abos-ctrl certificates list コマンドで確認可能です。

```
[armadillo ~]# abos-ctrl certificates list
- atmark-2
- atmark-3
- swupdate-2.pem: [mkswu --genkey で入力した COMMON NAME]
```

図 10.13 証明書がインストールされていることを確認する

10.8. コンテナについて

ここでは、Podman やコンテナのより応用的な使用方法について説明します。

10.8.1. コンテナの応用的な操作

10.8.1.1. ユーザーデータディレクトリを使用する

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

保存するデータの性質によって、保存先を選択してください。

1. /var/app/volumes/myvolume: アップデートした場合はコピーされません。ログやデータベースなど、アプリケーションが作成し続けるようなデータの保存に向いています。
2. myvolume か /var/app/rollback/volumes/myvolume: アップデートの際にコピーしてアップデートを行うので、アップデート中でも安全に使いつづけます。アプリケーションと一緒にアップデートするようなデータの保存に向いています。

ここで紹介する内容はコンテナ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 「4.4.2.12. コンテナをコンテナイメージとして保存する」 にあるボリュームの説明を参照してください。

10.8.1.2. コンテナイメージをアーカイブにする

podman save コマンドで、コンテナイメージを .tar 形式で保存することができます。作成した .tar アーカイブは 「10.3. mkswu の .desc ファイルを編集する」 の swdesc_embed_container と swdesc_usb_container で使えます。

```
[armadillo ~]# podman save my_image:latest -o my_image_1.tar
Writing manifest to image destination
```

```
[armadillo ~]# ls
my_image_1.tar
```

図 10.14 コンテナイメージをアーカイブにする

10.8.1.3. コンテナイメージをアップデートする

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]# vi Dockerfile
FROM localhost/my_image:latest

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

# update application
COPY my_application /my_application
[armadillo ~/podman-build-update]# podman build -t my_image:2 -t my_image:latest .
STEP 1: FROM localhost/my_image:latest
STEP 2: RUN apk upgrade --no-cache
--> cf1dc0d7296
STEP 3: COPY my_application /my_application
STEP 4: COMMIT my_image:latest
--> 9e9d9366072
Successfully tagged localhost/my_image:2
Successfully tagged localhost/my_image:latest
9e9d9366072751007b2e70544d76c46b95a7a5a02df658ef0fa3f7dcccf8850a

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

図 10.15 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
```

10.8.1.4. コンテナイメージを eMMC に保存する

ここまでで、コンテナイメージのダウンロード・作成（ podman pull, podman build ）とコンフィグファイルの作成を行いました。ですが、 podman pull や podman build で用意したコンテナイメージや、作成・変更したコンフィグファイルはデフォルト状態ではメモリ上にしか保存されません。これは、Armadillo Base OS の仕様により Podman のデータは tmpfs に保存されるためです。

そのため、このまま Armadillo を終了すると、これらのデータは消えてしまいます。Armadillo を終了してもデータが消えないようにするためには、ファイルは persist_file で保存し、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 に保存します。

また、起動時にコンテナを起動する場合にもイメージを eMMC に書き込む必要があります。

開発が終わって運用の場合は 「10.8.1.10. コンテナイメージを SWUpdate で転送する」 でコンテナのイメージを転送します。この場合は読み取り専用の app パーティションのサブボリュームに展開します。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。

ここで紹介する内容はコンテナ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 「4.4.2.12. コンテナをコンテナイメージとして保存する」 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 images
REPOSITORY                 TAG         IMAGE ID      CREATED      SIZE        R/O
localhost/at-debian-image  latest      b5a30f8581cc  2 hours ago  233 MB      true
```

図 10.16 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 images ❺
REPOSITORY                TAG         IMAGE ID      CREATED     SIZE        R/O
docker.io/library/alpine  latest      3d81c46cd875  3 days ago  5.56 MB     true
```

図 10.17 abos-ctrl podman-storage のイメージコピー例

❶ イメージを書き込み可能ストレージに取得します。

❷ abos-ctrl podman-storage をオプション無しで実行します。

❸ 書き込み可能ストレージにイメージがある場合に対応を聞かれます。今回はコピー（copy）します。

❹ abos-ctrl podman-storage にオプションを指定しなかったので、ストレージが tmpfs のままになります。すでに --disk で切り替えた場合にディスクのままでも可能です。

❺ コピーされたイメージを確認します。イメージが読み取り専用（R/O, Read only）になりました。

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

開発時には「abos-ctrl podman-storage --disk」の状態で作業を行い、運用時には「abos-ctrl podman-storage --tmpfs」に戻してください。戻る際に「copy」を選択する場合は一時的なストレージをそのまま使いつづけますので、すべての変更が残ります。

SWUpdate でアップデートをインストールする際には、/var/lib/containers/storage_readonly ディレクトリの不要になったイメージを自動的に削除します。

自動起動させる予定がなくても、「10.8.3. コンテナ起動設定ファイルを作成する」 を参考にして、 /etc/atmark/containers/*.conf を使ってください。 set_autostart no を設定することで自動実行されません。

10.8.1.5. コンテナ間で通信をする

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

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

準備として、2 つのコンテナを作成します。

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

図 10.18 コンテナを作成する実行例

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

```
[armadillo ~]# podman inspect --format='{{.NetworkSettings.IPAddress}}' my_container_1
10.88.0.108
[armadillo ~]# podman inspect --format='{{.NetworkSettings.IPAddress}}' my_container_2
10.88.0.109
```

図 10.19 コンテナの IP アドレスを確認する実行例

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

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

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

--- 10.88.0.109 ping statistics ---
```

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

図 10.20 ping コマンドによるコンテナ間の疎通確認実行例

このように、my_container_1(10.88.0.108) から my_container_2(10.88.0.109) への通信が確認できます。

10.8.1.6. pod でコンテナのネットワークネームスペースを共有する

podman_start で pod 機能を使うことができます。

pod を使うことで、複数のコンテナが同じネットワークネームスペースを共有することができます。同じ pod の中のコンテナが IP の場合 localhost で、 unix socket の場合 abstract path で相互に接続することができます。

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

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

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

図 10.21 pod を使うコンテナを自動起動するための設定例

コンテナと同じく、 /etc/atmark/containers/[NAME].conf ファイルを作って、 set_type pod を設定することで pod を作成します。

pod を使う時にコンテナの設定ファイルに set_pod [NAME] の設定を追加します。

ネットワークネームスペースは pod を作成するときに必要なため、 ports, network と ip の設定は pod のコンフィグファイルに入れなければなりません。

必要であれば、他の podman pod create のオプションを add_args で設定することができます。

.conf ファイルで使用できる各種パラメータについては、「10.8.3. コンテナ起動設定ファイルを作成する」を参照してください。

10.8.1.7. network の作成

podman_start で podman の network も作成できます。

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

```
[armadillo ~]# cat /etc/atmark/containers/mynetwork.conf
set_type network
set_subnet 198.51.100.0/24

[armadillo ~]# cat /etc/atmark/containers/nginx.conf
set_image docker.io/library/nginx:alpine
add_ports 80:80
set_ip 198.51.100.10
set_network mynetwork

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

図 10.22 network を使うコンテナを自動起動するための設定例

コンテナと同じく、 /etc/atmark/containers/[NAME].conf ファイルを作って、 set_type network を設定することで network を作成します。

そのネットワークを使う時にコンテナの設定ファイルに set_network [NAME] の設定をいれます。

ネットワークのサブネットは set_subnet [SUBNET] で設定します。この設定は set_type network の後しか使えませんので、set_type はファイルの最初のところに使ってください

他の podman network create のオプションが必要であれば、 add_args で設定することができます。

.conf ファイルで使用できる各種パラメータについては、「10.8.3. コンテナ起動設定ファイルを作成する」を参照してください。

10.8.1.8. コンテナからのコンテナ管理

podman では REST API による管理アクセスも可能です。

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

コンテナの設定によって podman の socket へのパスが自動設定されない場合もあります。podman --remote でエラーが発生した場合に CONTAINER_HOST=unix:/path/to/podman.sock で socket へのパスを設定してください。

Armadillo のホスト側の udev rules からコンテナを起動する場合は podman_start 等を直接実行すると udev の子プロセス管理によってコンテナが停止されますので、その場合はサービスを有効にし、podman_start --create <container> コマンドまたは set_autostart create の設定でコンテナを生成した上 podman --remote start <container> で起動してください。

10.8.1.9. リモートリポジトリにコンテナを送信する

1. イメージをリモートリポジトリに送信する：

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

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

   (mkswu については「9.3. Armadillo のソフトウェアをアップデートする」を参考にしてください)

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

swdesc_pull_container "docker.io/nginx:alpine"
swdesc_files --extra-os nginx_start
[ATDE ~/mkswu]$ mkswu pull_container_nginx.desc
Enter pass phrase for /home/atmark/mkswu/swupdate.key:
pull_container_nginx.swu を作成しました。
```

10.8.1.10. コンテナイメージを SWUpdate で転送する

1. イメージをファイルに保存する：

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

2. ファイルを SWUpdate のイメージに入れる。

   二つのやり方があります：

   a. SWU イメージ内に組み込む

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

swdesc_embed_container "nginx_alpine.tar"
swdesc_files --extra-os nginx_start
[ATDE ~/mkswu]$ podman pull --arch arm64 docker.io/nginx:alpine
[ATDE ~/mkswu]$ podman run --rm docker.io/nginx:alpine uname -m
aarch64
[ATDE ~/mkswu]$ podman save docker.io/nginx:alpine > nginx_alpine.tar
[ATDE ~/mkswu]$ mkswu embed_container_nginx.desc
Enter pass phrase for /home/atmark/mkswu/swupdate.key:
embed_container_nginx.swu を作成しました
```

   b. USB ドライブに保存する

```
[ATDE ~/mkswu]$ cp /usr/share/mkswu/examples/usb_container_nginx.desc .
[ATDE ~/mkswu]$ cp -r /usr/share/mkswu/examples/nginx_start .
```

```
[ATDE ~/mkswu]$ cat usb_container_nginx.desc
swdesc_option version=1

swdesc_usb_container "nginx_alpine.tar"
swdesc_files --extra-os nginx_start
[ATDE ~/mkswu]$ podman pull --arch arm64 docker.io/nginx:alpine
[ATDE ~/mkswu]$ podman run --rm docker.io/nginx:alpine uname -m
aarch64
[ATDE ~/mkswu]$ podman save docker.io/nginx:alpine > nginx_alpine.tar
[ATDE ~/mkswu]$ mkswu -o usb_container_nginx.swu usb_container_nginx.desc
Enter pass phrase for /home/atmark/mkswu/swupdate.key:
以下のファイルを USB メモリにコピーしてください：
'/home/atmark/mkswu/usb_container_nginx.swu'
'/home/atmark/mkswu/nginx_alpine.tar'
'/home/atmark/mkswu/.usb_container_nginx/nginx_alpine.tar.sig'

usb_container_nginx.swu を作成しました。
```

10.8.2. コンテナとコンテナに関連するデータを削除する

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

10.8.2.1. VS Code から実行する

VS Code 上で ABOSDE(Armadillo Base OS Development Environment) から、Armadillo のコンテナイメージを全て削除する SWU イメージを作成することができます。

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

この SWU イメージを 「7.3.3.6. SWU イメージのインストール」 を参照して Armadillo へインストールしてください。

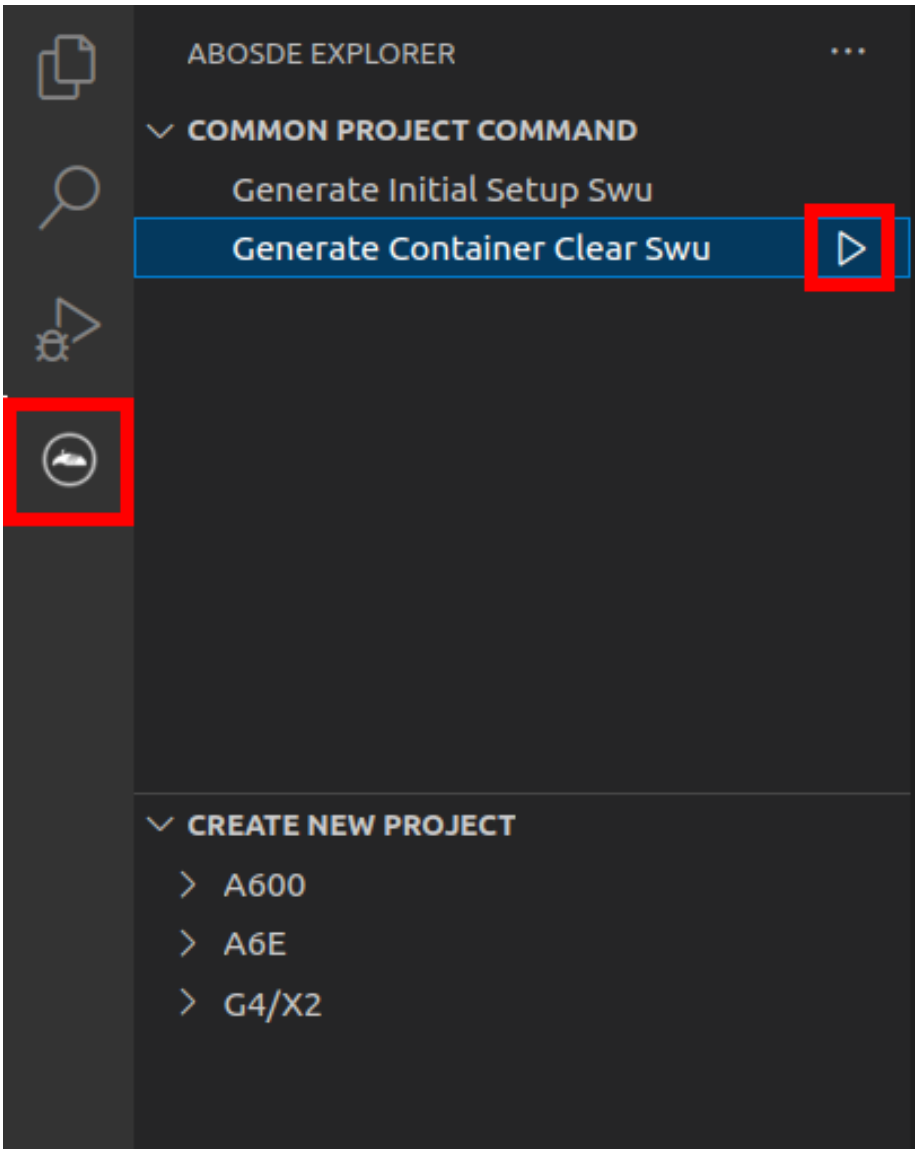


図 10.23 Armadillo 上のコンテナイメージを削除する

10.8.2.2. コマンドラインから実行する

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

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

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

・ /etc/swupdate.watch

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

図 10.24 abos-ctrl container-clear 実行例

10.8.3. コンテナ起動設定ファイルを作成する

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

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

図 10.25 コンテナを自動起動するための設定例

.conf ファイルに設定可能なパラメーターの説明を以下に記載します。podman_start --long-help コマンドでも詳細を確認できます。

10.8.3.1. コンテナイメージの選択

set_image [イメージ名]

イメージの名前を設定できます。

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

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

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

10.8.3.2. ポート転送

add_ports [ホストポート]:[コンテナポート]

設定したポートで外部からコンテナへのアクセスが可能となります。

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

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

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

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

10.8.3.3. デバイスファイル作成

add_devices [ホストパス]:[コンテナパス]

コンテナでデバイスを作成して、使用可能となります。

コンテナパスを設定しない場合はホストと同じパスを使います。

複数のデバイスを作成したい場合はスペースで分けて設定してください。

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

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

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

コンテナパスに「:」を含むようなパスは設定できません。

10.8.3.4. ボリュームマウント

add_volumes [ホストパス]:[コンテナパス]:[オプション]

指定するパスをコンテナ内でマウントして、データの保存や共有ができます。

ホストパスは以下のどれかを指定してください。

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

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

  SWUpdate でアップデートするデータに向いています。

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

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

  ログやデータベースに向いています。

- /tmp/<folder>: 複数のコンテナでメモリファイルシステムを共有したい場合に使ってください。

・ /opt/firmware: 学習能力に必要なファムウェアライブラリーのパス。

コンテナパスを設定しない場合はホストパスと同じパスを使います。

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

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

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

「:」はホスト側のパスとコンテナの側のパスを分割する意味があるため、ファイル名に「:」を使用することはできません。ホスト側のパスにのみ「:」が含まれてる場合は「 add_volumes "[ホストパス]" "[コンテナパス]" "[オプション]" 」と指定することで設定できます。

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

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

図 10.26 ボリュームを shared でサブマウントを共有する例

❶ マウントを行うコンテナに shared の設定とマウント権限 (SYS_ADMIN) を与えます。

❷ マウントを使うコンテナに slave だけを設定すれば一方にしか共有されません。

❸ USB デバイスをマウントします。

❹ マウントされたことを確認します。

10.8.3.5. ホットプラグデバイスの追加

add_hotplugs [デバイスタイプ]

コンテナ起動後に挿抜を行なっても認識される(ホットプラグ)デバイスを設定できます。

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

例: add_hotplugs input

add_hotplugs に指定できる主要な文字列とデバイスファイルの対応について、「表 10.5. 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 10.5 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 10.5. add_hotplugs オプションに指定できる主要な文字列」に示した文字列の他にも、/proc/devices の数字から始まる行に記載されている文字列を指定することができます。「図 10.27. /proc/devices の内容例」に示す状態の場合、デバイスタイプを示す文字列としては、各行の先頭の数字を除いた mem や pty などを指定できることがわかります。

```
[armadillo ~]# cat /proc/devices
Character devices:
  1 mem
  2 pty
  3 ttyp
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  7 vcs
 10 misc
 13 input
 29 fb
 81 video4linux
: (省略)
```

図 10.27 /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

10.8.3.6. 個体識別情報の環境変数の追加

add_armadillo_env

アットマークテクノが設定した個体識別情報をコンテナの環境変数として追加することができます。

例: add_armadillo_env

add_armadillo_env を設定することで追加されるコンテナの環境変数について、「表 10.6. add_armadillo_env で追加される環境変数」に示します。

表 10.6 add_armadillo_env で追加される環境変数

環境変数	環境変数の説明	表示例
AT_ABOS_VERSION	ABOS のバージョン	3.18.4-at.5
AT_LAN_MAC1	アットマークテクノが設定した LAN1（eth0）の MAC アドレス	00:11:0C:12:34:56
AT_PRODUCT_NAME	製品名	Armadillo-900 開発セット
AT_SERIAL_NUMBER	個体番号	00C900010001

「表 10.6. add_armadillo_env で追加される環境変数」に示した環境変数をコンテナ上で確認する場合、「図 10.28. add_armadillo_env で設定した環境変数の確認方法」に示すコマンドを実行してください。ここでは、個体番号の環境変数を例に示します。

```
[container ~]# echo $AT_SERIAL_NUMBER
00C900010001
```

図 10.28 add_armadillo_env で設定した環境変数の確認方法

お客様が独自の環境変数をコンテナに追加する場合は「図 9.6. 個体番号の環境変数を conf ファイルに追記」を参考に conf ファイルを編集してください。

10.8.3.7. pod の選択

set_pod［ポッド名］

「10.8.1.6. pod でコンテナのネットワークネームスペースを共有する」で作成した pod の名前を入れてコンテナを pod 内で起動します。

例: set_pod mypod

10.8.3.8. ネットワークの選択

set_network［ネットワーク名］

この設定に「10.8.1.7. network の作成」で作成したネットワーク以外に none と host の特殊な設定も選べます。

none の場合、コンテナに localhost しかないネームスペースに入ります。

host の場合は OS のネームスペースをそのまま使います。

例: set_network mynetwork

10.8.3.9. IP アドレスの設定

set_ip［アドレス］

コンテナの IP アドレスを設定することができます。

例: set_ip 10.88.0.100

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

10.8.3.10. 読み取り専用設定

set_readonly yes

コンテナ内からのファイルシステムへの書き込み許可を設定します。

デフォルトで書き込み可能となっています。

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

10.8.3.11. イメージの自動ダウンロード設定

set_pull［設定］

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

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

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

例：set_pull missing か set_pull always

10.8.3.12. コンテナのリスタート設定

set_restart［設定］

コンテナが停止した時にリスタートさせます。

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

デフォルトで on-failure になっています。

例: set_restart always か set_restart no

10.8.3.13. 信号を受信するサービスの無効化

set_init no

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

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

例: set_init no

10.8.3.14. podman logs 用のログサイズ設定

set_log_max_size 〈サイズ〉

podman logs でログを表示するために /run にログファイルを保存しています。そのログのサイズが設定したサイズを越えるとクリアされます。デフォルトは「1MB」です。

10.8.3.15. podman のフックの仕組み

add_hook --stage 〈ステージ〉[--] コマンド [コマンド引数]

コンテナが起動されるなど、動作ステージの変化をフックとしてコマンドを実行します。複数のステージで実行したい場合は --stage オプションを複数設定してください。

指定可能なステージは precreate, prestart, createRuntime, createContainer, startContainer, poststart, と poststop です。ステージの意味や使用方法の詳細は podman のドキュメンテーションを参照してください。

Armadillo Base OS 3.19.1-at.4 現在では set_restart によるコンテナの再起動でも 1 度目の停止時のみ poststop フックが実行されます。2 度目以降の停止では実行されませんのでご注意ください。

10.8.3.16. ヘルスチェック機能の設定

set_healthcheck [引数] [--] コマンド [コマンド引数]

定期的にコマンドを実行して、コンテナの正常性を確認します。指定可能な引数は以下のとおりです：

- **--retries 〈リトライ数〉**: エラーを検知するまでのリトライ回数。(デフォルト: 3)
- **--action 〈none|restart|kill|stop|reboot|rollback〉**: 指定したリトライ回数分連続でチェックが失敗したときのアクション (デフォルト: restart)：
  - none: set_healthcheck_fail_command に指定した処理を実行する以外何もしません。
  - restart: コンテナを再起動します。 set_restart オプションと異なり、コンテナを起動しなおし初期状態で再起動します。
  - kill/stop: コンテナを停止します。
  - reboot: Armadillo を再起動します。
  - rollback: ロールバック可能の場合はロールバックして Armadillo を再起動します。ロールバック不可能な場合はそのまま Armadillo を再起動します。
- **--interval 〈時間〉**: チェックする時間間隔です。(デフォルト: 1 min)
- **--start-period 〈時間〉**: 最初のチェックを実行する前の待ち時間です。(デフォルト: interval 設定の値)
- **--timeout 〈秒数〉**: 設定された時間以内にヘルスチェックが終了しなかった場合は失敗となります。(デフォルト: 無し)

また、いくつかのタイミングでコマンドを実行させることができます：

- **set_healthcheck_start_command コマンド［コマンド引数］**: コンテナ起動後にヘルスチェックが初めて成功した際に実行されるコマンドです。
- **set_healthcheck_fail_command コマンド［コマンド引数］**: ヘルスチェックが retries 回失敗した後に実行されるコマンドです。このコマンドは set_healthcheck の --action 設定の前に実行されますので、コマンドだけを実行したい場合は --action none で無効化してください。
- **set_healthcheck_recovery_command コマンド［コマンド引数］**: ヘルスチェックが retries 回失敗した後に再び成功した際に実行されるコマンドです。コンテナを起動する際に成功せずに失敗した場合は、その 1 回目の成功の際に set_healthcheck_start_command で設定されたコマンドのみが実行されます。

例: set_healtcheck -- curl -s --fail http://localhost:8080/status
例: set_healthcheck_start_command abos-ctrl rollback-clone

```
armadillo:~# grep podman_atmark /var/log/messages
Jun 20 11:33:21 armadillo user.notice podman_atmark: my_container healthcheck is now healthy (was starting)
Jun 20 11:33:21 armadillo user.notice podman_atmark: my_container first healthy check: running abos-ctrl rollback-clone
Jun 20 11:40:21 armadillo user.notice podman_atmark: my_container healthcheck failed (from healthy, 1 / 3)
Jun 20 11:41:21 armadillo user.notice podman_atmark: my_container healthcheck failed (from healthy, 2 / 3)
Jun 20 11:42:21 armadillo user.notice podman_atmark: my_container healthcheck failed (from healthy, 3 / 3)
Jun 20 11:42:21 armadillo user.notice podman_atmark: my_container is unhealthy, restarting container
Jun 20 11:43:21 armadillo user.notice podman_atmark: my_container healthcheck is now healthy (was failed)
```

図 10.29 上記の例でエラーを発生させた際の起動ログ

10.8.3.17. 自動起動の無効化

set_autostart no または **set_autostart create**

Armadillo の起動時にコンテナを自動起動しないように設定できます。

create を指定した場合はコンテナは生成されており、podman start <name> で起動させることができます。

no を指定した場合は podman_start <name> で起動させることができます。

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

10.8.3.18. 実行コマンドの設定

set_command［コマンド］

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

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

10.8.3.19. コンテナ起動前にコマンドを実行する

add_pre_command [コマンド]

コンテナを起動する直前に設定したコマンドを実行します。

Armadillo Base OS の環境で実行されてますので、ハードウェアの設定等に適切です。

また、複数のコマンドを実行する場合は順番に実行されます。設定したコマンドが 1 つでも失敗した場合は、コンテナは起動されません。

例: add_pre_command gpioset --daemonize CONx_y=1

10.8.3.20. podman run に引数を渡す設定

add_args [引数]

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

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

10.8.4. アットマークテクノが提供するイメージを使う

アットマークテクノは、動作確認環境として使用できる Debian ベースのイメージを提供しています。ここでは以下の 3 つの手順について説明します。

- ABOSDE からインストールする方法
- Docker ファイルからイメージをビルドする方法
- すでにビルド済みのイメージを使う方法

10.8.4.1. ABOSDE からインストールする

1. インストール用のプロジェクトを作成する

   VS Code の左ペインの [A9E] から [Atmark Container New Project] を実行し、表示されるディレクトリ選択画面からプロジェクトを保存するディレクトリを選択してください。保存先を選択すると、プロジェクト名を入力するダイアログが表示されるので、任意のプロジェクト名を入力してエンターキーを押してください。この操作により、選択した保存先に、入力したプロジェクト名と同名のディレクトリが作成されます。

   また、ここでは次のように設定しています。

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

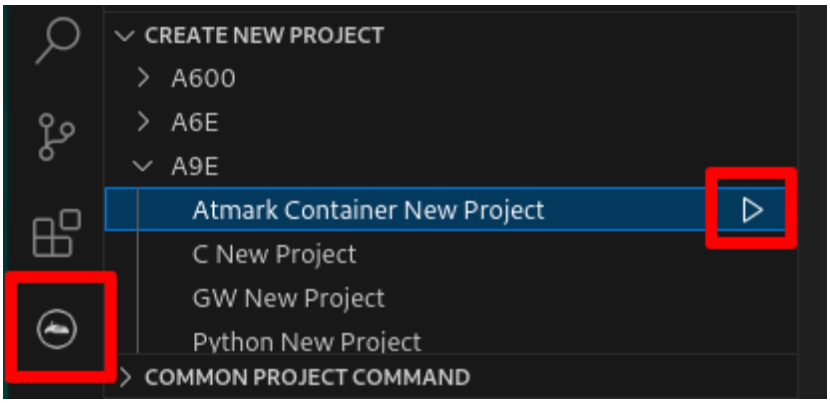


図 10.30 インストール用のプロジェクトを作成する

2. SWU イメージを作成する

   VS Code の左ペインの [my_project] から [Generate at-debian-image container setup swu] を実行してください。

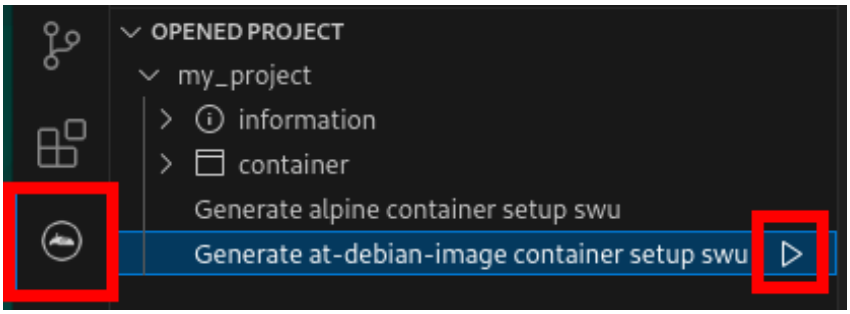


図 10.31 at-debian-image のコンテナイメージをインストールする SWU ファイルを作成する

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

3. SBOM 生成に関わる設定を行う

   ABOSDE から作成した場合は SBOM が同時に生成されます。詳細は 「7.11. SBOM 生成に関わる設定を行う」 をご確認ください。SBOM の生成には以下の二つのファイルが必要です。

   - コンフィグファイル
   - desc ファイル

     SBOM の生成にはライセンス情報を示したコンフィグファイルを使用します。コンフィグファイルは `container_setup/at-debian-image.sbom_config.yaml.tmpl` になります。SWU イメージ作成時にこのコンフィグファイルからバージョン番号をアップデートした `container_setup/at-debian-image.sbom_config.yaml` が生成されます。

     リリース時にはコンフィグファイルの内容を確認し、正しい内容に変更してください。各項目の詳細な説明については SPDX specification v2.2.2 (https://spdx.github.io/spdx-spec/v2.2.2/) をご覧ください。SBOM に含めるコンテナイメージ等の情報については desc ファ

イルに記載されています。各項目の説明については 「10.20.4.2. desc ファイルを編集する」をご覧ください。

10.8.4.2. Docker ファイルからイメージをビルドする

Armadillo-900 開発セット コンテナ [https://armadillo.atmark-techno.com/resources/software/armadillo-900/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 10.32 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。

10.8.4.3. ビルド済みのイメージを使用する

Armadillo-900 開発セット コンテナ [https://armadillo.atmark-techno.com/resources/software/armadillo-900/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 10.33 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。

10.8.5. alpine のコンテナイメージをインストールする

alpine のコンテナイメージは、 ABOSDE を用いてインストールすることが可能です。「10.8.4.1. ABOSDE からインストールする」 を参照して、 インストール用のプロジェクトを作成しておいてください。

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

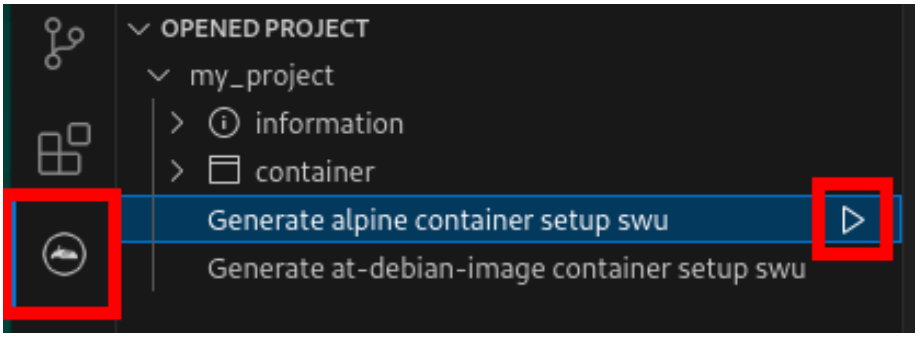


図 10.34 alpine のコンテナイメージをインストールする SWU ファイルを作成する

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

10.8.5.1. SBOM 生成に関わる設定を行う

ABOSDE から作成した場合は SBOM が同時に生成されます。詳細は 「7.11. SBOM 生成に関わる設定を行う」 をご確認ください。SBOM の生成には以下の二つのファイルが必要です。

- コンフィグファイル
- desc ファイル

SBOM の生成にはライセンス情報を示したコンフィグファイルを使用します。コンフィグファイルは container_setup/alpine.sbom_config.yaml.tmpl になります。SWU イメージ作成時にこのコンフィグファイルからバージョン番号をアップデートした container_setup/alpine.sbom_config.yaml が生成されます。

リリース時にはコンフィグファイルの内容を確認し、正しい内容に変更してください。各項目の詳細な説明については SPDX specification v2.2.2 (https://spdx.github.io/spdx-spec/v2.2.2/) をご覧ください。SBOM に含めるコンテナイメージ等の情報については desc ファイルに記載されています。各項目の説明については 「10.20.4.2. desc ファイルを編集する」 をご覧ください。

10.8.6. コンテナのネットワークを扱う

この章では、コンテナ内のネットワークを扱う方法について示します。

10.8.6.1. コンテナの IP アドレスを確認する

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

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

```
48ae479af65445674323567c17c5418dd4624292351e061bd2bd8a0add4cf150
[armadillo ~]# podman inspect --format '{{ .NetworkSettings.IPAddress }}' net_example
10.88.0.17
```

図 10.35 コンテナの IP アドレス確認例

コンテナ内の ip コマンドを用いて確認することもできます。

```
[armadillo ~]# podman exec net_example ip addr show eth0
3: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether xx:xx:xx:xx:xx:xx brd ff:ff:ff:ff:ff:ff
    inet 10.88.0.17/16 brd 10.88.255.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::40e5:98ff:feec:4b17/64 scope link
       valid_lft forever preferred_lft forever
```

図 10.36 ip コマンドを用いたコンテナの IP アドレス確認例

10.8.6.2. 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 198.51.100.0/24 にユーザ定義のネットワークを作成する例を示します。

```
[armadillo ~]# vi /etc/atmark/containers/my_network.conf
set_type network
set_subnet 198.51.100.0/24
[armadillo ~]# podman_start my_network
Creating network 'my_network'
my_network
```

図 10.37 ユーザ定義のネットワーク作成例

コンテナを作成する際に、上記で作成したネットワークと設定したい IP アドレスを渡すことで、コンテナの IP アドレスを固定することができます。以下の例では、IP アドレスを 198.51.100.10 に固定します。

```
[armadillo ~]# vi /etc/atmark/containers/network_example.conf
set_image docker.io/alpine
set_command sleep infinity
set_network my_network
set_ip 198.51.100.10
[armadillo ~]# podman_start network_example
```

```
Starting 'network_example'
3ea8c9031bf833228908bd73d8929b1d543b189b436c218e0634e0d39409e100
```

図 10.38 IP アドレス固定のコンテナ作成例

コンテナの IP アドレスが、198.51.100.10 に設定されていることが確認できます。

```
[armadillo ~]# podman inspect --format '{{ .NetworkSettings.Networks.my_network.IPAddress }}'
network_example
198.51.100.10
```

図 10.39 コンテナの IP アドレス確認例

10.8.7. コンテナ内にサーバを構築する

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

10.8.7.1. HTTP サーバを構築する

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

・ Apache を使用する

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

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

図 10.40 コンテナに Apache をインストールする例

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

・ lighttpd を使用する

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

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

図 10.41 コンテナに lighttpd をインストールする例

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

10.8.7.2. FTP サーバを構築する

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

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

図 10.42 コンテナに vsftpd をインストールする例

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

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

図 10.43 ユーザを追加する例

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

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

図 10.44 設定ファイルの編集例

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

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

図 10.45 vsftpd の起動例

10.8.7.3. Samba サーバを構築する

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

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

図 10.46 コンテナに samba をインストールする例

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

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

図 10.47 ユーザを追加する例

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

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

図 10.48 samba の起動例

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

10.8.7.4. SQL サーバを構築する

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

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

図 10.49 コンテナに sqlite をインストールする例

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

```
[container ~]# sqlite3 /db/mydb.sqlite
SQLite version 3.34.1 2021-01-20 14:10:07
Enter ".help" for usage hints.
sqlite>
```

図 10.50 sqlite の実行例

10.8.8. コンテナからの poweroff 及び reboot

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

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

```
[armadillo ~]# vi /etc/atmark/containers/shutdown_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_args --pid=host
[armadillo ~]# podman_start shutdown_example
Starting 'shutdown_example'
c8e3b9b418fc72395db9f3c22b1eb69eb41eaaf790d3b7151047ef066cc4c8ff
[armadillo ~]# podman exec -ti shutdown_example sh
```

```
[container ~]# kill -USR2 1  (poweroff)
[container ~]# kill -TERM 1  (reboot)
```

図 10.51 コンテナから shutdown を行う

10.8.9. 異常検知

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

10.8.9.1. ソフトウェアウォッチドッグタイマーを扱う

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

```
[armadillo ~]# vi /etc/atmark/containers/watchdog_example.conf
set_image docker.io/alpine
set_command sleep infinity
add_devices /dev/watchdog0
[armadillo ~]# podman_start watchdog_example
Starting 'watchdog_example'
a5d329cca49d60423ce4155d72a119b8049a03dbd1d0277817a253e96dce7bc7
```

図 10.52 ソフトフェアウォッチドッグタイマーを使うためのコンテナ作成例

ソフトウェアウォッチドッグタイマーは、プログラム内からデバイスファイル /dev/watchdog0 を open した時点で起動します。コンテナ内に入ってソフトウェアウォッチドッグタイマーを echo コマンドで起動する例を以下に示します。

```
[armadillo ~]# podman exec -it watchdog_example sh
[container ~]# echo > /dev/watchdog0
```

図 10.53 コンテナ内からソフトウェアウォッチドッグタイマーを起動する実行例

ソフトウェアウォッチドッグタイマーを起動した後、/dev/watchdog0 に（ V 以外の）任意の文字を書き込むことでソフトウェアウォッチドッグタイマーをリセットすることができます。60 秒間（ V 以外の）任意の文字の書き込みがない場合は、システムが再起動します。

```
[armadillo ~]# podman exec -it watchdog_example sh
[container ~]# echo a > /dev/watchdog0
```

図 10.54 ソフトウェアウォッチドッグタイマーをリセットする実行例

ソフトウェアウォッチドッグタイマーを停止したい場合は、/dev/watchdog0 に V を書き込みます。

```
[armadillo ~]# podman exec -it watchdog_example sh
[container ~]# echo V > /dev/watchdog0
```

図 10.55 ソフトウェアウォッチドッグタイマーを停止する実行例

10.9. Web UI から Armadillo をセットアップする (ABOS Web)

ABOS Web は、Web ブラウザから Armadillo の動作設定を行う機能で、ABOS (Armadillo Base OS) を搭載する全ての Armadillo に対応しています。

詳細は、「5.1.1. ABOS Web とは」を参照してください。

10.9.1. ABOS Web ではできないこと

ABOS Web は、ABOS の詳細や Linux のコマンドシェルの操作に詳しくない方でも、簡単に Armadillo のセットアップを行なえることを目的にしています。そのための、Armadillo の動作設定を行う機能ですから、動作設定以外のこと、たとえば、Armadillo の動作状態を監視したりすることは、できません。さらに、Armadillo をインターネットから設定操作する、リモート操作もできません。セキュリティの観点から、ABOS Web は、同じ LAN 内からの接続しか受け付けないように実装しています。

ABOS Web でできる Armadillo の設定については、「10.9.2. ABOS Web の設定機能一覧と設定手順」を参照してください。なお、ABOS Web は OSS で提供していますので、現在の ABOS Web に無い設定機能を、ご自分で実装して機能追加することも可能です。

10.9.2. ABOS Web の設定機能一覧と設定手順

現在、ネットワークに関して ABOS Web で設定できるのは以下のものです。

- WWAN 設定
- WLAN 設定
- 各接続設定（各ネットワークインターフェースの設定）
- DHCP サーバー設定
- NAT 設定
- VPN 設定

これらについては、「5.1. ABOS Web を用いたネットワーク設定方法」で紹介していますので、そちらを参照してください。

ネットワーク以外にも ABOS Web は以下の機能を持っています。

- コンテナ管理
- SWU インストール
- 時刻設定
- アプリケーション向けのインターフェース (Rest API)
- カスタマイズ

本章では、これらのネットワーク以外の設定項目について紹介します。

10.9.3. コンテナ管理

ABOS Web から Armadillo 上のコンテナを一覧表示して、コンテナごとに起動・停止を行うことができます。

ABOS Web のトップページから、"コンテナ管理"をクリックすると、「図 10.56. コンテナ管理」の画面に遷移します。

図 10.56 コンテナ管理

この画面では、ABOS 上にあるコンテナ全てについて、イメージ名やコンテナ名、現在状態を一覧表示します。コンテナの一覧表示欄で選択したコンテナに対し、起動と停止、および、コンテナから出力されたログの表示を行うことができます。

「5.1.12. VPN 設定」に記載のとおり、VPN 接続を設定すると、abos_web_openvpn のコンテナが作成されます。VPN 接続中は、このコンテナが動作状態になっており、このコンテナをコンテナ管理画面で停止すると、VPN 接続が切断されます。

10.9.4. SWU インストール

ABOS Web から PC 上の SWU イメージや HTTP サーバー上の SWU イメージを Armadillo にインストールすることができます。

SWU イメージについては、「7.3.3.2. SWU イメージとは」を参照してください。

ABOS Web のトップページから、"SWU インストール"をクリックすると、「図 10.57. SWU インストール」の画面に遷移します。

図 10.57 SWU インストール

この画面では、PC 上の SWU イメージファイルまたは、HTTP サーバー上の SWU イメージファイルの URL を指定して、Armadillo にインストールすることができます。Armadillo のソフトウェアのアップデート用に最初に行う設定で作成する initial_setup.swu が、まだ Armadillo にインストールされていなければ、"mkswu --init で作成した initial_setup.swu をインストールしてください。" というメッセージを画面上部に表示します。

SWU イメージのインストール動作を実行する時には、進行状況を示すログを表示します。"現在の SWU で管理されているバージョン" 欄には、ABOS の各ソフトウェアコンポーネントの名前とバージョン情報を一覧表示します。

10.9.5. 時刻設定

SWU 管理対象ソフトウェアコンポーネントの一覧表示. ABOS Web から時刻に関する設定を行うことができます。

ABOS Web のトップページから "時刻設定" をクリックすると、以下の内容が表示されます。

「図 10.58. ネットワークタイムサーバーと同期されている場合の状況確認画面」 では Armadillo の現在時刻と、同期中のサーバーとの時間差を確認することができます。

図 10.58 ネットワークタイムサーバーと同期されている場合の状況確認画面

時刻が同期されてない状態では 「図 10.59. ネットワークタイムサーバーと同期されていない場合の状況確認画面」 の様に 「PC と同期する」ボタンを押すことで、 Armadillo の時刻を PC と同期することができます。

図 10.59 ネットワークタイムサーバーと同期されていない場合の状況確認画面

「図 10.60. ネットワークタイムサーバーの設定項目」 では NTP (ネットワークからの時刻同期）サーバーと Armadillo 起動時に同期するサーバーを設定することができます。

図 10.60 ネットワークタイムサーバーの設定項目

最後に、「図 10.61. タイムゾーンの設定項目」 では Armadillo Base OS で使用するタイムゾーンの変更ができます。コンテナには影響ありませんのでご注意ください。

図 10.61 タイムゾーンの設定項目

10.9.6. アプリケーション向けのインターフェース (Rest API)

コンテナやスクリプトから ABOS Web の一部の機能を使用できます。

10.9.6.1. Rest API へのアクセス権の管理

Rest API は ABOS Web のパスワードと Rest API 用のトークンで認証されます。

また、接続可能なネットワークにも制限をかけております。初期状態では、同一サブネットからのアクセスのみ許容しています。同一サブネット外の IP アドレスからアクセスしたい場合は設定が必要です。設定方法は 「5.1.2. ABOS Web へのアクセス」 を参照してください。

各リクエストは以下のどちらかの Authorization ヘッダーで認証されます：

- Basic (パスワード認証) : curl の -u :<password> 等で認証可能です。<password> の文字列は ABOS Web で設定したパスワードです。
- Bearer (トークン認証) : curl の -H "Authorization: Bearer <token> 等で認証可能です。<token> は /api/tokens であらかじめ生成した文字列です。

また、トークンには権限も設定できます。Admin で生成されたトークンはすべてのインターフェースにアクセスできますが、一部のインターフェースしか使用しない場合はそのインターフェースに必要な権限だけを持つトークンを生成してください。

トークンの管理は ABOS Web の「設定管理」ページで行えます:

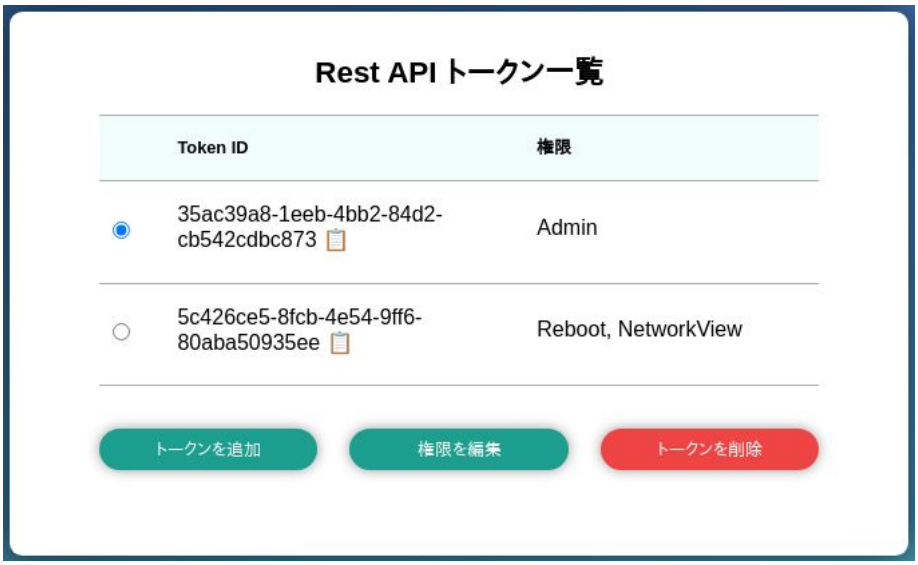


図 10.62 設定管理の Rest API トークン一覧表示

ABOS Web の バージョン 1.2.3 以降では、Token ID の横にあるクリップボードアイコンをクリックするとクリップボードにコピーすることができます。

10.9.6.2. Rest API 使用例の前提条件

各 Rest API の使用例を説明します。使用例では以下を前提としています。:

- ABOS Web に `https://armadillo.local:58080` でアクセスします。
- 「 `AUTH` 」環境変数に ABOS Web で生成したトークンを設定します。例： `AUTH="Authorization: Bearer 35ac39a8-1eeb-4bb2-84d2-cb542cdbc873"`
- `curl` コマンドを省略するため、以下のように alias を使用します：

```
[ATDE ~]$ alias curl_rest='curl -k -H "$AUTH" -w "¥nhttp code: %{http_code}¥n" '
```

コンテナから ABOS Web には「https://host.containers.internal:58080」でアクセスできます。

この章で説明する例では、`curl` のオプションに `-k` を指定して証明書を無視するようにしています。もし、証明書を使用したい場合は以下のように設定してください。

```
[ATDE ~]$ openssl s_client -showcerts -connect armadillo.local:58080 </
dev/null 2>/dev/null | openssl x509 -outform PEM > abosweb.pem
[ATDE ~]$ CERT="$PWD/abosweb.pem"
[ATDE ~]$ alias curl_rest='curl -H "$AUTH" --cacert "$CERT" -w "¥nhttp
code: %{http_code}¥n" '
```

10.9.6.3. Rest API の入力と出力

インターフェースの一部にはパラメータを取るものがあります。パラメータがある場合は json (Content-Type を application/json に設定する）と form（デフォルトの application/x-www-form-urlencoded でのパラメータ）のどちらでも使用可能です。

インターフェースの出力がある場合は json object で出力されます。今後のバージョンアップで json object のキーが増える可能性があるため、出力された値を処理する場合はその点に留意してください。

エラーの場合は json object の「error」キーに文字列のエラーが記載されています。http のステータスコードも 50x になります。

エラーの例：

```
[ATDE ~]$ curl_rest -X DELETE https://armadillo.local:58080/api/tokens/
3b2d830d-2f64-4e76-9e59-316da82eefc4
{"error":"No such token"}
http code: 500
```

10.9.6.4. Rest API : トークン管理

トークン管理のためのインターフェースは以下のとおりです：

- **トークン一覧**
  GET "/api/tokens"
  必要権限: Admin
  パラメータ: 無し
  出力: トークンリスト

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/tokens
{"tokens":[{"token":"35ac39a8-1eeb-4bb2-84d2-cb542cdbc873","permissions":["Admin"]},
{"token":"5c426ce5-8fcb-4e54-9ff6-80aba50935ee","permissions":["Reboot","NetworkView"]}]}
http code: 200
```

- **トークン取得**
  GET "/api/tokens/<token>"
  必要権限: Admin
  パラメータ: 無し
  出力: トークン情報

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/tokens/35ac39a8-1eeb-4bb2-84d2-
cb542cdbc873
{"token":"35ac39a8-1eeb-4bb2-84d2-cb542cdbc873","permissions":["Admin"]}
http code: 200
```

- **トークン生成**
  POST "/api/tokens"
  必要権限: Admin
  パラメータ: 付与したい permissions 権限リスト(ない場合は「Admin」で生成されます)
  出力: 生成されたトークン情報

```
[ATDE ~]$ curl_rest -H "Content-type: application/json" -d '{"permissions": ["SwuInstall",
"ContainerView"]}' https://armadillo.local:58080/api/tokens
{"token":"3b2d830d-2f64-4e76-9e59-316da82eefc4","permissions":
["SwuInstall","ContainerView"]}
http code: 200
```

- **トークン編集 (存在しない場合は指定のトークンで生成されます)**
  POST "/api/tokens/{token_id}"
  必要権限: Admin
  パラメータ: 付与したい permissions 権限リスト(ない場合は編集しません)
  出力: 編集か生成されたトークン情報

```
[ATDE ~]$ curl_rest -X POST -d permissions=Poweroff -d permissions=ContainerAdmin https://
armadillo.local:58080/api/tokens/3b2d830d-2f64-4e76-9e59-316da82eefc4
{"token":"3b2d830d-2f64-4e76-9e59-316da82eefc4","permissions":["Poweroff","ContainerAdmin"]}
```

- **トークン削除**
  DELETE "/api/tokens/{token_id}"
  必要権限: Admin
  パラメータ: 無し
  出力: 無し

```
[ATDE ~]$ curl_rest -X DELETE https://armadillo.local:58080/api/tokens/
3b2d830d-2f64-4e76-9e59-316da82eefc4
http code: 200
```

- **abos-web パスワード変更**
  POST "/api/password"
  必要権限: Admin
  パラメータ: password でハッシュ済みのパスワード文字列か hashed=false が設定されている場合は平文の文字列
  出力: 無し

```
[ATDE ~]$ PWD_HASH=$(openssl passwd -6)
Password:
Verifying - Password:
[ATDE ~]$ echo $PWD_HASH
$6$LuXQduN7L3PwbMaZ$txrw8vLJqEVUreQnZhM0CYMQ5U5B9b58L0mpVRULDiVCh2O46GKscq/
xsDPskjxg.x8ym0ri1/8NqFBu..IZE0
[ATDE ~]$ curl_rest --data-urlencode "password=$PWD_HASH" -X POST https://armadillo.local:
58080/api/password
http code: 200
```

10.9.6.5. Rest API : SWU

- **インストール済み SWU のバージョン情報取得**
  GET "/api/swu/versions"
  必要権限: SwuView
  パラメータ: 無し
  出力: Swupdate の各バージョン情報

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/swu/versions
{"extra_os.custom":"54","extra_os.container":"1","custom":"54","extra_os.initial_setup":"4",
"boot":"2020.4-at19","base_os":"3.18.4-at.6","extra_os.sshd":"1"}
http code: 200
```

- **アップデートステータス取得**
  GET "/api/swu/status"
  必要権限: SwuView
  パラメータ: 無し
  出力: rollback_ok: ロールバック状態 (false の場合は rollback されています)、last_update_timestamp: UTC の unix epoch (数字での日付)、 last_update_versions: 最新のアップデートで更新されたバージョン情報 (コンポーネント → [更新前のバージョン, 更新後のバージョン]。 更新前に存在しなかったコンポーネントの場合は null で記載されています)

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/swu/status
{"rollback_ok":true,"last_update_timestamp":1703208559,"last_update_versions":{"custom":
[null,"54"],"extra_os.custom":["53","54"]}}
http code: 200
```

- **SWU をファイルアップロードでインストール**
  POST "/api/swu/install/upload"
  必要権限: SwuInstall
  パラメータ: multipart/form-data で **swu** の転送
  出力: swupdate プロセスの出力 (stdout または stderr)、またはアップデートプロセスの出力ステータス (exit_code または exit_signal)

```
[ATDE ~]$ curl_rest -F swu=@"$HOME/mkswu/file.swu" https://armadillo.local:58080/api/swu/
install/upload
{"stdout":"SWUpdate v2023.05_git20231025-r0¥n"}
{"stdout":"¥n"}
{"stdout":"Licensed under GPLv2. See source distribution for detailed copyright notices.¥n"}
{"stdout":"¥n"}
{"stdout":"[INFO ] : SWUPDATE running :  [main] : Running on AGX4500 Revision at1¥n"}
{"stdout":"[INFO ] : SWUPDATE started :  Software Update started !¥n"}
{"stdout":"[INFO ] : SWUPDATE running :  [install_single_image] : Installing pre_script¥n"}
{"stdout":"[INFO ] : SWUPDATE running :  [read_lines_notify] : No base os update: copying
current os over¥n"}
: (省略)
{"stdout":"[INFO ] : SWUPDATE running :  [install_single_image] : Installing post_script¥n"}
{"stdout":"[INFO ] : SWUPDATE running :  [read_lines_notify] : Removing unused containers¥n"}
{"stdout":"[INFO ] : SWUPDATE running :  [read_lines_notify] : swupdate triggering reboot!¥n"}
{"stderr":"Killed¥n"}
{"exit_code":0}
```

```
http code: 200
```

- **SWU を URL でインストール**
  POST "/api/swu/install/url"
  必要権限: SwuInstall
  パラメータ: url=<SWU をダウンロードできる URL>
  出力: swupdate プロセスの出力 (stdout または stderr)、またはアップデートプロセスの出力ステータス (exit_code または exit_signal)

```
[ATDE ~]$ curl_rest -d url=https://url/to/file.swu https://armadillo.local:58080/api/swu/
install/url
{"stdout":"Downloading https://url/to/file.swu...¥n"}
{"stdout":"SWUpdate v2023.05_git20231025-r0¥n"}
{"stdout":"¥n"}
{"stdout":"Licensed under GPLv2. See source distribution for detailed copyright notices.¥n"}
{"stdout":"¥n"}
{"stdout":"[INFO ] : SWUPDATE running :  [main] : Running on AGX4500 Revision at1¥n"}
{"stdout":"[INFO ] : SWUPDATE started :  Software Update started !¥n"}
{"stdout":"[INFO ] : SWUPDATE running :  [install_single_image] : Installing pre_script¥n"}
{"stdout":"[INFO ] : SWUPDATE running :  [read_lines_notify] : No base os update: copying
current os over¥n"}
: (省略)
{"stdout":"[INFO ] : SWUPDATE running :  [install_single_image] : Installing post_script¥n"}
{"stdout":"[INFO ] : SWUPDATE running :  [read_lines_notify] : Removing unused containers¥n"}
{"stdout":"[INFO ] : SWUPDATE running :  [read_lines_notify] : swupdate triggering reboot!¥n"}
{"stderr":"Killed¥n"}
{"exit_code":0}

http code: 200
```

10.9.6.6. Rest API : コンテナ操作

- **コンテナー覧**
  GET "/api/containers"
  必要権限: ContainerView
  パラメータ: 無し
  出力: 各コンテナの id, name, state, command, image 情報

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/containers
{"containers":
[{"id":"02616122dcea5bd75c551b29b2ef54f54e09f59c50ce3282684773bc6bfb86a8","name":"python_app
","state":"running","command":["python3","/vol_app/src/main.py"],"image":"localhost/
python_arm64_app_image:latest"}]}
http code: 200
```

- **コンテナログ取得**
  GET "/api/containers/{container}/logs"
  必要権限: ContainerView
  パラメータ: **follow=true** (podman logs -f と同様の効果)
  出力: podman logs プロセスの出力 (stdout または stderr)、またはアップデートプロセスの出力ステータス (exit_code または exit_signal)

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/containers/python_app/logs
{"stdout":"Some message¥n"}
{"exit_code":0}

http code: 200
```

follow=true を付与する例

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/containers/python_app/logs?follow=true
{"stdout":"Some message¥n"}
Ctrl-C で終了
```

- **コンテナ起動**
  POST "/api/containers/{container}/start"
  必要権限: ContainerAdmin
  パラメータ: 無し
  出力: 無し

```
[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/containers/python_app/start

http code: 200
```

- **コンテナ停止**
  POST "/api/containers/{container}/stop"
  必要権限: ContainerAdmin
  パラメータ: 無し
  出力: 無し

```
[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/containers/python_app/stop

http code: 200
```

10.9.6.7. Rest API : ネットワーク設定

- **ネットワーク設定一覧**
  GET "/api/connections"
  必要権限: NetworkView
  パラメータ: 無し
  出力: ネットワーク設定一覧と各接続の uuid, name, state, ctype, 存在すれば device 情報

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/connections
{"connections":[{"name":"Wired connection
1","state":"activated","uuid":"18d241f1-946c-3325-974f-65cda3e6eea5","ctype":"802-3-
ethernet","device":"eth0"},{"name":"lo","state":"activated","uuid":"529ec241-f122-4cb2-843f-
ec9787b2aee7","ctype":"loopback","device":"lo"},
{"name":"podman0","state":"activated","uuid":"be4583bc-3498-4df2-
a31c-773d781433aa","ctype":"bridge","device":"podman0"},
{"name":"veth0","state":"activated","uuid":"03446b77-b1ab-47d0-98fc-
f167c3f3778a","ctype":"802-3-ethernet","device":"veth0"},{"name":"Wired connection
```

```
2","state":"","uuid":"181f44df-850e-36c1-a5a4-6e461c768acb","ctype":"802-3-ethernet"},
{"name":"Wired connection 3","state":"","uuid":"e4381368-6351-3985-
ba6e-2625c62b8d39","ctype":"802-3-ethernet"}]}

http code: 200
```

- **ネットワーク設定詳細取得**
  GET "/api/connections/{connection}"
  必要権限: NetworkView
  パラメータ: 無し（URL の connection は UUID または接続名で使用可能）
  出力: 接続の詳細情報（Network Manager のプロパティ）

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/connections/Wired%20connection%201
{"name":"Wired connection
1","state":"activated","uuid":"18d241f1-946c-3325-974f-65cda3e6eea5","ctype":"802-3-
ethernet","device":"eth0","props":{"802-3-ethernet.accept-all-mac-addresses":"-1","802-3-
ethernet.auto-negotiate":"no","802-3-ethernet.cloned-mac-address":"","802-3-
ethernet.duplex":"","802-3-ethernet.generate-mac-address-mask":"","802-3-ethernet.mac-
address":"","802-3-ethernet.mac-address-blacklist":"","802-3-ethernet.mtu":"auto","802-3-
ethernet.port":"","802-3-ethernet.s390-nettype":"","802-3-ethernet.s390-options":"","802-3-
ethernet.s390-subchannels":"","802-3-ethernet.speed":"0","802-3-ethernet.wake-on-
lan":"default","802-3-ethernet.wake-on-lan-password":"","GENERAL.CON-PATH":"/org/
freedesktop/NetworkManager/Settings/1","GENERAL.DBUS-PATH":"/org/freedesktop/NetworkManager/
ActiveConnection/
6","GENERAL.DEFAULT":"yes","GENERAL.DEFAULT6":"no","GENERAL.DEVICES":"eth0","GENERAL.IP-
IFACE":"eth0","GENERAL.MASTER-PATH":"","GENERAL.NAME":"Wired connection 1","GENERAL.SPEC-
OBJECT":"","GENERAL.STATE":"activated","GENERAL.UUID":"18d241f1-946c-3325-974f-65cda3e6eea5"
,"GENERAL.VPN":"no","GENERAL.ZONE":"","IP4.ADDRESS[1]":"198.51.100.123/16","IP4.DNS[1]":"192
.0.2.1","IP4.DNS[2]":"192.0.2.2","IP4.GATEWAY":"198.51.100.1","IP4.ROUTE[1]":"dst =
198.51.100.0/16, nh = 0.0.0.0, mt = 100","IP4.ROUTE[2]":"dst = 0.0.0.0/0, nh = 198.51.100.1,
mt = 100","IP6.ADDRESS[1]":"fe80::211:cff:fe00:b13/64","IP6.GATEWAY":"","IP6.ROUTE[1]":"dst
= fe80::/64, nh = ::, mt = 1024","connection.auth-
retries":"-1","connection.autoconnect":"yes","connection.autoconnect-
priority":"-999","connection.autoconnect-retries":"-1","connection.autoconnect-
slaves":"-1","connection.dns-over-tls":"-1","connection.gateway-ping-
timeout":"0","connection.id":"Wired connection 1","connection.interface-
name":"eth0","connection.lldp":"default","connection.llmnr":"-1","connection.master":"","con
nection.mdns":"-1","connection.metered":"unknown","connection.mptcp-
flags":"0x0","connection.multi-connect":"0","connection.permissions":"","connection.read-
only":"no","connection.secondaries":"","connection.slave-type":"","connection.stable-
id":"","connection.timestamp":"1703208824","connection.type":"802-3-
ethernet","connection.uuid":"18d241f1-946c-3325-974f-65cda3e6eea5","connection.wait-
activation-delay":"-1","connection.wait-device-
timeout":"-1","connection.zone":"","ipv4.addresses":"198.51.100.123/16","ipv4.auto-route-
ext-gw":"-1","ipv4.dad-timeout":"-1","ipv4.dhcp-client-id":"","ipv4.dhcp-
fqdn":"","ipv4.dhcp-hostname":"","ipv4.dhcp-hostname-flags":"0x0","ipv4.dhcp-
iaid":"","ipv4.dhcp-reject-servers":"","ipv4.dhcp-send-hostname":"yes","ipv4.dhcp-
timeout":"0","ipv4.dhcp-vendor-class-
identifier":"","ipv4.dns":"192.0.2.1,192.0.2.2","ipv4.dns-options":"","ipv4.dns-
priority":"0","ipv4.dns-search":"","ipv4.gateway":"198.51.100.1","ipv4.ignore-auto-
dns":"no","ipv4.ignore-auto-routes":"no","ipv4.link-local":"0","ipv4.may-
fail":"yes","ipv4.method":"manual","ipv4.never-default":"no","ipv4.replace-local-
rule":"-1","ipv4.required-timeout":"-1","ipv4.route-metric":"-1","ipv4.route-
table":"0","ipv4.routes":"","ipv4.routing-rules":"","ipv6.addr-gen-
mode":"eui64","ipv6.addresses":"","ipv6.auto-route-ext-gw":"-1","ipv6.dhcp-
duid":"","ipv6.dhcp-hostname":"","ipv6.dhcp-hostname-flags":"0x0","ipv6.dhcp-
```

```
iaid":"","ipv6.dhcp-send-hostname":"yes","ipv6.dhcp-timeout":"0","ipv6.dns":"","ipv6.dns-
options":"","ipv6.dns-priority":"0","ipv6.dns-search":"","ipv6.gateway":"","ipv6.ignore-
auto-dns":"no","ipv6.ignore-auto-routes":"no","ipv6.ip6-privacy":"-1","ipv6.may-
fail":"yes","ipv6.method":"auto","ipv6.mtu":"auto","ipv6.never-default":"no","ipv6.ra-
timeout":"0","ipv6.replace-local-rule":"-1","ipv6.required-timeout":"-1","ipv6.route-
metric":"-1","ipv6.route-table":"0","ipv6.routes":"","ipv6.routing-
rules":"","ipv6.token":"","proxy.browser-only":"no","proxy.method":"none","proxy.pac-
script":"","proxy.pac-url":""}}
http code: 200
```

- **ネットワーク設定の変更**
  PATCH "/api/connections/{connection}"
  必要権限: NetworkAdmin
  パラメータ: Network Manager で編集可能な値
  出力: 無し

```
[ATDE ~]$ curl_rest -X PATCH -d ipv4.method=manual -d ipv4.addresses=198.51.100.123/16
https://armadillo.local:58080/api/connections/Wired%20connection%201

http code: 200
```

- **ネットワークの接続**
  POST "/api/connections/{connection}/up"
  必要権限: NetworkAdmin
  パラメータ: 無し
  出力: 無し

```
[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/connections/Wired%20connection
%201/up

http code: 200
```

- **ネットワークの切断**
  POST "/api/connections/{connection}/down"
  必要権限: NetworkAdmin
  パラメータ: 無し
  出力: 無し

```
[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/connections/Wired%20connection
%201/down

http code: 200
```

- **ネットワーク設定の削除**
  DELETE "/api/connections/{connection}"
  必要権限: NetworkAdmin
  パラメータ: 無し出力: 無し

```
[ATDE ~]$ curl_rest -X DELETE https://armadillo.local:58080/api/connections/178b8c95-
fcad-4bb1-8040-5a02b9ad046f
```

```
http code: 200
```

通信に使用しているネットワークの設定を削除した場合は Armadillo へアクセスできなくなりますので、ご注意ください。

10.9.6.8. Rest API : WLAN

- **無線ネットワークのリスト取得**
  GET "/api/wlan/scan"
  必要権限: NetworkView
  パラメータ: (任意)rescan=true/false, false を指定するとキャッシュされているスキャン結果を出力します。
  出力: リスト

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/wlan/scan
[{"id":"my_ap","signal":74,"bssid":"04:42:1A:E4:78:0C","chan":44,"rate":"540 Mbit/
s","security":"WPA2 WPA3"},{"id":"other_ap","signal":65,"bssid":"AC:44:F2:56:22:38","chan":
1,"rate":"130 Mbit/s","security":"WPA2"}]
http code: 200
```

- *無線ネットワークの接続
  POST "/api/wlan/connect"
  必要権限: NetworkAdmin
  パラメータ： ssid, passphrase, ifname, bssid, hidden. ssid 以外は任意です。
  出力: 生成した接続の uuid

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/wlan/connect -d ssid=my_ap -d
passphrase=my_passphrase
{"uuid":"178b8c95-fcad-4bb1-8040-5a02b9ad046f"}
http code: 200
```

- **無線ネットワーク アクセスポイントの設定**
  POST "/api/wlan/ap"
  必要権限: NetworkAdmin
  パラメータ: ssid, passphrase, bridge_addr, hw_mode/channel, interface.
  interface は任意です。hw_mode:2.4GHz を使用する場合は "g"、5GHz を使用する場合は "a" を設定します。
  channel: 2.4GHz の場合は 1 ～ 13、5GHz の場合は 36、40、44、48 を設定します。
  hw_mode/channel を設定しない場合は自動的に選択されますが、両方を未設定にすることはできません。
  出力: 無し

```
[ATDE ~]$ curl_rest -d ssid=my_ap -d passphrase=my_passphrase -d bridge_addr=198.51.100.1/24
-d channel=3 https://armadillo.local:58080/api/wlan/ap

http code: 200
```

アクセスポイントを設定するとクライアントの接続が無効になります。

クライアントの接続の削除は DELETE "/api/connections/{connection}" で行えます。

- **無線ネットワーク アクセスポイントの削除**
  DELETE "/api/wlan/ap"
  必要権限: NetworkAdmin
  パラメータ: interface (任意)
  出力: 無し

```
[ATDE ~]$ curl_rest -X DELETE https://armadillo.local:58080/api/wlan/ap

http code: 200
```

10.9.6.9. Rest API : WWAN の設定

- **WWAN の設定追加**
  POST "/api/wwan"
  必要権限: NetworkAdmin
  パラメータ: apn, user, password, auth_type (CHAP/PAP, デフォルト CHAP), mccmnc, ipv6 (bool、デフォルト true)
  apn 以外は任意です。
  出力: 追加された接続の uuid

```
[ATDE ~]$ curl_rest -d apn=provider.tld -d user=provider -d password=provider https://
armadillo.local:58080/api/wwan
{"uuid":"ce603d3e-838b-4ac8-b7fd-6a3f1abe4003"}
http code: 200
```

- **WWAN の設定削除**
  DELETE "/api/wwan"
  必要権限: NetworkAdmin
  パラメータ: 無し
  出力: 無し

```
[ATDE ~]$ curl_rest -X DELETE https://armadillo.local:58080/api/wwan

http code: 200
```

WWAN の設定確認または一時的な切断は connection の API で行ってください。

- **IMEI の取得**
  GET "/api/wwan/imei"
  必要権限: NetworkView
  パラメータ: 無し
  出力: LTE モジュールの IMEI

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/wwan/imei
{"imei":"XXXXXXXXXXXXXX"}
http code: 200
```

- **電話番号の取得**
  GET "/api/wwan/phone_numbers"
  必要権限: NetworkView
  パラメータ: 無し
  出力: SIM カードに設定されている電話番号

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/wwan/phone_numbers
{"phone_numbers":["XXXXXXXXXX"]}
http code: 200
```

- **電波品質の取得**
  GET "/api/wwan/signal_quality"
  必要権限: NetworkView
  パラメータ: 無し
  出力: 電波品質(mmcli コマンドで出力される signal quality 相当の値)

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/wwan/signal_quality
{"signal_quality":"54"}
http code: 200
```

- **SMS 一覧の取得**
  GET "/api/wwan/sms"
  必要権限: SmsView
  パラメータ: 無し
  出力: SMS メッセージ ID 一覧

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/wwan/sms
{"message_ids":["0", "1"]}
http code: 200
```

- **SMS の取得**
  GET "/api/wwan/sms/{message_id}"
  必要権限: SmsView

パラメータ: 無し
出力: SMS の内容

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/wwan/sms/0
{"sms":{"content":{"data":"--","number":"XXXXXXXXXXX","text":"message"},"dbus_path":"/org/
freedesktop/ModemManager1/SMS/0","properties":
{"pdu_type":"deliver","state":"received","storage":"me","timestamp":"20YY-MM-DDThh:mm:ss
+ZZ"}}}

http code: 200
```

message_id は SMS 一覧の取得で表示された値を使用します。

- **SMS の作成・送信**
  POST "/api/wwan/sms"
  必要権限: SmsSend
  パラメータ: phone_number: 送信先電話番号, message: 送信するメッセージ
  出力: 無し

```
[ATDE ~]$ curl_rest -d phone_number=XXXXXXXXXXX -d message="message" https://armadillo.local:
58080/api/wwan/sms

http code: 200
```

- **SMS の削除**
  DELETE "/api/wwan/sms/{message_id}"
  必要権限: SmsView
  パラメータ: 無し
  出力: 無し

```
[ATDE ~]$ curl_rest -X DELETE https://armadillo.local:58080/api/sms/0

http code: 200
```

message_id は SMS 一覧の取得で表示された値を使用します。

- **SMS の全削除**
  DELETE "/api/wwan/sms"
  必要権限: SmsView
  パラメータ: 無し
  出力: 無し

```
[ATDE ~]$ curl_rest -X DELETE https://armadillo.local:58080/api/sms

http code: 200
```

10.9.6.10. Rest API : DHCP の設定

- **DHCP の設定確認**
  GET "/api/dhcp"
  必要権限: NetworkView
  パラメータ: 無し
  出力: interface, ip_addr, start_addr, end_addr, lease_time のリスト

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/dhcp
[{"interface":"br_ap","ip_addr":"198.51.100.1/24","start_addr":"198.51.100.10","end_addr":"1 ⏎
98.51.100.20","lease_time":"3600"}]
http code: 200
```

- **DHCP の設定**
  POST "/api/dhcp/{interface}"
  必要権限: NetworkAdmin
  パラメータ: start_addr, end_addr, lease_time
  lease_time を設定しなかった場合は 3600 (秒) とする
  出力: 無し

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/dhcp/br_ap -d start_addr=198.51.100.10 ⏎
-d end_addr=198.51.100.20

http code: 200
```

- **DHCP の設定削除**
  DELETE "/api/dhcp/{interface}"
  必要権限: NetworkAdmin
  パラメータ: 無し
  出力: 無し

```
[ATDE ~]$ curl_rest -X DELETE https://armadillo.local:58080/api/dhcp/br_ap

http code: 200
```

10.9.6.11. Rest API : NAT の設定

- **NAT (masquerade) の設定確認**
  GET "/api/nat"
  必要権限: NetworkView
  パラメータ: 無し
  出力: NAT されている interface のリスト

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/nat
[{"interface":"eth0"}]
http code: 200
```

- NAT の設定
  POST "/api/nat/{interface}"
  必要権限: NetworkAdmin
  パラメータ: 無し
  出力: 無し

```
[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/nat/eth0

http code: 200
```

- NAT の削除
  DELETE "/api/nat/{interface}"
  必要権限: NetworkAdmin
  パラメータ: 無し
  出力: 無し

```
[ATDE ~]$ curl_rest -X DELETE https://armadillo.local:58080/api/nat/eth0

http code: 200
```

- ポートフォワードの設定確認
  GET "/api/port_forwarding"
  必要権限: NetworkView
  パラメータ: 無し
  出力: フォワードされてるポート

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/port_forwarding
[{"interface":"eth0","protocol":"tcp","dport":"22","destination":"127.0.0.1","destination_po ⏎
rt":"2222"}]
http code: 200
```

- ポートフォワードの設定
  POST "/api/port_forwarding"
  必要権限: NetworkAdmin
  パラメータ: interface, protocol (デフォルト tcp), dport, destination, destination_port
  出力: 無し

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/port_forwarding -d interface=eth0  -d ⏎
dport=22 -d de
stination=127.0.0.1 -d destination_port=2222

http code: 200
```

- ポートフォワードの削除

DELETE "/api/port_forwarding"
必要権限: NetworkAdmin
パラメータ: interface, protocol (デフォルト tcp), dport, destination, destination_port
出力: 無し

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/port_forwarding -X DELETE -H "Content-Type: application/json" -d
'{"interface":"eth0","protocol":"tcp","dport":"22","destination":"127.0.0.1","destination_port":"2222"}'

http code: 200
```

10.9.6.12. Rest API : VPN の設定

VPN クライアントは、現在 OpenVPN [https://openvpn.net/community/] をサポートしています。

- **VPN の設定**
  POST "/api/vpn/openvpn"
  必要権限: VpnAdmin
  パラメータ: setting_name, conf, auth_type, username, password, cert, key, key_pass
  setting_name: 設定名です。任意の文字列を設定してください。
  conf: OpenVPN の設定ファイルです。
  auth_type: 認証方式です。userpass(ユーザ名とパスワード) または cert(証明書) を設定してください。
  username: auth_type が userpass の場合、ユーザ名を設定します。
  password: auth_type が userpass の場合、パスワードを設定します。
  cert: auth_type が cert の場合、証明書ファイルを設定します。OpenVPN の設定ファイルに含まれている場合は不要です。
  key: auth_type が cert の場合、鍵ファイルを設定します。OpenVPN の設定ファイルに含まれている場合は不要です。
  key_pass: 鍵がパスワードで保護されている場合、そのパスワードを設定します。
  出力: 無し

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/vpn/openvpn -F setting_name=myvpn -F conf=@"$HOME/conf.ovpn" -F auth_type=userpass -F username=myname -F password=mypass
http code: 200
```

図 10.63 ユーザ名とパスワード認証の例

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/vpn/openvpn  -F setting_name=myvpn -F conf=@"$HOME/conf.ovpn" -F auth_type=cert -F cert=@"$HOME/cert.crt" -F key=@"$HOME/key.key" -F key_pass=mykeypass
http code: 200
```

図 10.64 証明書認証の例

コンテナ内から VPN 設定の Rest API を使う場合、 Armadillo 上に alpine_openvpn コンテナイメージが存在していないと、正しく設定されません。このコンテナイメージが存在していない場合、先に ABOS Web

> の Web UI やコンテナ外からの Rest API で設定を行ってください。一度 alpine_openvpn コンテナイメージがインストールされれば、コンテナ内からも Rest API で設定できます。alpine_openvpn コンテナイメージは VPN 設定を削除しても残り続けますが、VPN 設定を削除した後に swupdate でアップデートを行うと削除されますので、その場合は再度 alpine_openvpn コンテナイメージのインストールを行う必要があります。

- **VPN の接続**
  POST "/api/vpn/up"
  必要権限: VpnAdmin
  パラメータ: 無し
  出力: 無し

```
[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/vpn/up
http code: 200
```

- **VPN の切断**
  POST "/api/vpn/down"
  必要権限: VpnAdmin
  パラメータ: 無し
  出力: 無し

```
[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/vpn/down
http code: 200
```

- **VPN 設定の削除**
  DELETE "/api/vpn/openvpn"
  必要権限: VpnAdmin
  パラメータ: 無し
  出力: 無し

```
[ATDE ~]$ curl_rest -X DELETE https://armadillo.local:58080/api/vpn/openvpn
http code: 200
```

10.9.6.13. Rest API : 時刻の設定

- **時刻の状況確認**
  GET "/api/time/ntp_info"
  必要権限: TimeView
  パラメータ: 無し
  出力: time_now: epoch 形式の現在時刻、ntp_server_ip: 現在同期中のサーバーアドレス。同期されていない場合は「null」となります。ntp_server_offset: 現在同期中のサーバーとの時刻の遅れ（マイナスの場合は Armadillo がサーバーより早いです）

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/time/ntp_info
{"ntp_server_ip":"203.0.113.10","ntp_server_offset":"-0.000015824","time_now":1710139558}
http code: 200
```

・**NTP の設定確認**
GET "/api/time/ntp_config"
必要権限: TimeView
パラメータ: 無し
出力: servers: 同期する対象、 initstepslew: Armadillo 起動時に同期するかどうかの設定

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/time/ntp_config
{"servers":["pool pool.ntp.org iburst"],"initstepslew":"10 pool.ntp.org"}
http code: 200
```

・**NTP の設定**
POST "/api/time/ntp_config"
必要権限: TimeAdmin
パラメータ: servers: 同期する対象、 initstepslew: Armadillo 起動時に同期するかどうかの設定。パラメータを送信しない場合は設定されません。値が空の場合は設定が削除されて、「 default 」の場合は Armadillo Base OS のデフォルトに戻ります。
出力: 取得時と同じ

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/time/ntp_config -X POST -d "servers=server 203.0.113.10 iburst" -d "servers=server 203.0.113.11 iburst" -d "initstepslew="
{"servers":["server 203.0.113.10 iburst","server 203.0.113.11 iburst"],"initstepslew":null}
http code: 200
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/time/ntp_config -X POST -d "servers=default&initstepslew=default"
{"servers":["pool pool.ntp.org iburst"],"initstepslew":"10 pool.ntp.org"}
http code: 200
```

・**タイムゾーンの確認**
GET "/api/time/timezone"
必要権限: TimeView
パラメータ: 無し
出力: timezone: 使用されているタイムゾーン

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/time/timezone
{"timezone":"Asia/Tokyo"}
http code: 200
```

・**タイムゾーンの設定**
POST "/api/time/timezone"
必要権限: TimeAdmin
パラメータ: timezone: 設定するタイムゾーン
出力: 無し

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/time/timezone -X POST -d "timezone=Asia/Tokyo"
http code: 200
```

・**時刻を強制的に設定する**
POST "/api/time/set"

必要権限: TimeAdmin
パラメータ: timestamp: epoch 形式の時刻
出力: 無し

```
[ATDE ~]$ curl_rest https://armadillo.local:58080/api/time/set -X POST -d "timestamp=$(date +%s)"
http code: 200
```

10.9.6.14. Rest API : 電源制御

- **再起動**
  POST "/api/reboot"
  必要権限: Reboot
  パラメータ: 無し
  出力: 無し

  ```
  [ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/reboot

  http code: 200
  ```

- **停止**
  POST "/api/poweroff"
  必要権限: Poweroff
  パラメータ: 無し
  出力: 無し

  ```
  [ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/poweroff

  http code: 200
  ```

10.9.6.15. Rest API : ABOS Web 制御

- **リスタート**
  POST "/api/abosweb/restart"
  必要権限: AbosWebRestart
  パラメータ: 無し
  出力: コネクションリセット。ABOS Web はリスタートする前に一度終了するためコネクションリセットが発生します。

  ```
  [ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/abosweb/restart

  http code: 000
  curl: (52) Empty reply from server
  ```

10.9.6.16. Rest API : ユーザー設定とユーザーデータの管理

- **ユーザー設定とユーザーデータの削除**
  POST "/api/reset_default"
  必要権限: ResetDefault
  パラメータ: 無し

出力: abos-ctrl reset-default の出力 (stdout または stderr)、および出力ステータス (exit_code または exit_signal)

```
[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/reset_default
{"stdout":"rm -f /etc/NetworkManager/system-connections/*¥n"}
{"stdout":"persist_file -r /etc/NetworkManager/system-connections¥n"}
{"stdout":"persist_file -r /etc/dnsmasq.d¥n"}
{"stdout":"rc-service dnsmasq restart¥n"}
{"stdout":"/etc/init.d/iptables save¥n"}
{"stdout":"sed -i -e '/NETAVARK/d' /etc/iptables/rules-save¥n"}
{"stdout":"persist_file /etc/iptables/rules-save¥n"}
{"stdout":"podman stop -a¥n"}
{"stdout":"find /var/app/volumes /var/log -mindepth 1 -delete¥n"}
{"stdout":"Starting clone to /dev/mmcblk0p1¥n"}
{"stdout":"Cloning rootfs¥n"}
{"stdout":"Updating appfs snapshots¥n"}
{"stdout":"Reusing up-to-date bootloader¥n"}
{"stdout":"Rollback clone successful¥n"}
{"stderr":"WARNING: Rebooting!¥n"}
{"exit_code":0}

http code: 200
```

10.9.6.17. Rest API : カスタムスクリプトの実行

ユーザが Armadillo に追加したスクリプトを Rest API を使用して実行することができます。実行したいスクリプトに実行権限を付与し、Armadillo の /etc/atmark/abos_web/customize_rest ディレクトリ下に置いてください。

実行に root 権限が必要なスクリプトの場合は、 以下のように /etc/doas.d/abos_web_customize.conf にスクリプトを追加してください。

```
[armadillo ~]# cat /etc/doas.d/abos_web_customize.conf
permit nopass abos-web-admin as root cmd /etc/atmark/abos_web/customize_rest/root_command.sh
```

- **任意のスクリプト実行**
  POST "/api/custom/{script}"
  必要権限: Custom
  パラメータ: **args** でスクリプトの引数を順番に指定できます。
  root を true に設定すると root 権限でスクリプトを実行します。
  出力: /etc/atmark/abos_web/customize_rest/{script} {args} {args...} を実行して、そのスクリプトの出力を stdout/stderr で返します。スクリプトが終了した際の出力ステータスは exit_code または exit_signal (どちらも int) です。

```
[armadillo ~]# cat /etc/atmark/abos_web/customize_rest/print_args.sh
#!/bin/sh

printf "arg: %s¥n" "$@"
[ATDE ~]$ curl_rest -X POST https://armadillo.local:58080/api/custom/print_args.sh ¥
      -H 'Content-type: application/json' -d '{"args": ["param", "second arg"], "root":false}'
{"stdout":"arg: param¥n"}
{"stdout":"arg: second arg¥n"}
{"exit_code":0}
```

標準の ABOS Web には最小限の権限しか与えていません。
root 権限でスクリプトを実行する場合、 Armadillo の故障やセキュリティにも関わりますので、十分注意して追加してください。

10.9.7. カスタマイズ

ABOS Web をお客様の最終製品へ組み込む場合に、ロゴ画像や背景色、メニューの文言などをカスタマイズすることができます。詳細は 「7.5. ABOS Web をカスタマイズする」 を参照してください。

10.9.8. ユーザー設定とユーザーデータの削除

カスタマイズと Rest API トークン以外の設定内容と、ユーザーデータを一括削除することができます。

ユーザーデータの削除では以下のデータを削除します。

- /var/app/volumes ディレクトリ下のファイルを全て
- /var/log ディレクトリ下のファイルを全て

ABOS Web のトップページから、「設定管理」ページへ移動し「ユーザー設定とユーザーデータの削除」にある「削除」ボタンを押すと削除できます。削除後は Armadillo が再起動するので引き続き ABOS Web を使用する場合は、再起動が完了してからアクセスしてください。

10.9.9. ABOS Web を停止する

「図 10.65. ABOS Web を停止する」に ABOS Web のサービスを停止する方法を示します。

```
[armadillo ~]# rc-update | grep abos-web ❶
             abos-web |      default
[armadillo ~]# rc-service abos-web status ❷
 * status: started
[armadillo ~]# rc-service abos-web stop ❸
abos-web                 | * Stopping abos-web ... [ ok ]
[armadillo ~]# rc-update del abos-web ❹
 * service abos-web deleted from runlevel default
[armadillo ~]# persist_file -d /etc/runlevels/default/abos-web ❺
```

図 10.65 ABOS Web を停止する

❶ OpenRC に ABOS Web のサービスが登録されていることを確認します。

❷ ABOS Web のサービスが起動していることを確認します。

❸ ABOS Web のサービスを停止します。

❹ サービスを管理している OpenRC から ABOS Web のサービスの登録を解除します。

❺ サービス設定ファイルの削除を永続化します。

ABOS Web を停止すると ABOS Web の Rest API も使用できなくなります。

10.9.10. ABOS Web を起動する

「図 10.66. ABOS Web を起動する」に ABOS Web のサービスを起動する方法を示します。

```
[armadillo ~]# rc-update | grep abos-web ❶
[armadillo ~]# rc-update add abos-web ❷
* service abos-web added to runlevel default
[armadillo ~]# rc-service abos-web start ❸
abos-web                | * Starting abos-web ... [ ok ]
[armadillo ~]# persist_file /etc/runlevels/default/abos-web ❹
```

図 10.66 ABOS Web を起動する

❶ OpenRC に ABOS Web のサービスが登録されていないことを確認します。何も出力されなければ登録されていません。

❷ サービスを管理している OpenRC に ABOS Web のサービスを登録します。

❸ ABOS Web のサービスを起動します。

❹ サービス設定ファイルを永続化します。

10.9.11. ABOS Web のセキュリティ対策

ABOS Web は開発時には便利ですが、運用時にはできることの多さ故に外部からの攻撃面になり得ます。ここでは、 ABOS Web が具備しているセキュリティ対策機能について紹介します。

10.9.11.1. 同一 LAN 以外からのアクセス禁止

「10.9.1. ABOS Web ではできないこと」でも紹介している通り、 ABOS Web は同一 LAN 内からのみ接続でき、それ以外の接続は拒否します。これによって ABOS Web はインターネット側からの攻撃面となり得ず、攻撃者は同一の LAN 内に侵入してから ABOS Web にアクセスする必要があります。

10.9.11.2. ABOS Web のユーザ認証

ABOS Web はユーザ認証としてパスワードによる認証を行います。初回起動時には必ずパスワードの設定を求められます。このとき設定するパスワードは 8 文字以上のものに強制しています。

また、ユーザ認証のブルートフォースアタック対策として、パスワードを間違える等でユーザ認証に失敗した場合に、次回再認証まで 2 秒間時間を空ける実装になっています。これによって、機械的に総当りでパスワードを解析するまでの時間が大幅に増大することが見込めます。

10.9.11.3. 通信の暗号化

ABOS Web の通信は TLS によって暗号化されています。使用する TLS 証明書は ECDSA (secp384r1) の暗号技術を用いて生成されています。

10.10. ABOSDE から ABOS Web の機能を使用する

ABOSDE は以下に示す ABOS Web の情報取得や動作を行うことができます。

- Armadillo の SWU バージョンを取得する
- Armadillo のコンテナの情報を取得する

- Armadillo のコンテナを起動・停止する
- Armadillo のコンテナのログを取得する
- Armadillo に SWU をインストールする

ABOSDE は ABOS Web の Rest API を用いて通信を行っていますので、ABOS Web にパスワードでログインができる状態である必要があります。ABOS Web へのログインを行っていない場合は「5.1.1. ABOS Web とは」を参考にしてください。

ABOSDE から ABOS Web の機能を使用するには通信を行う対象の Armadillo を選択する必要があります。「図 10.67. ABOSDE で ローカルネットワーク上の Armadillo をスキャンする」 の赤枠で囲まれているボタンをクリックすることで、ローカルネットワーク上で ABOS Web が動作している Armadillo をスキャンすることができます。ただし、ATDE のネットワークを NAT に設定している場合は Armadillo がリストに表示されません。

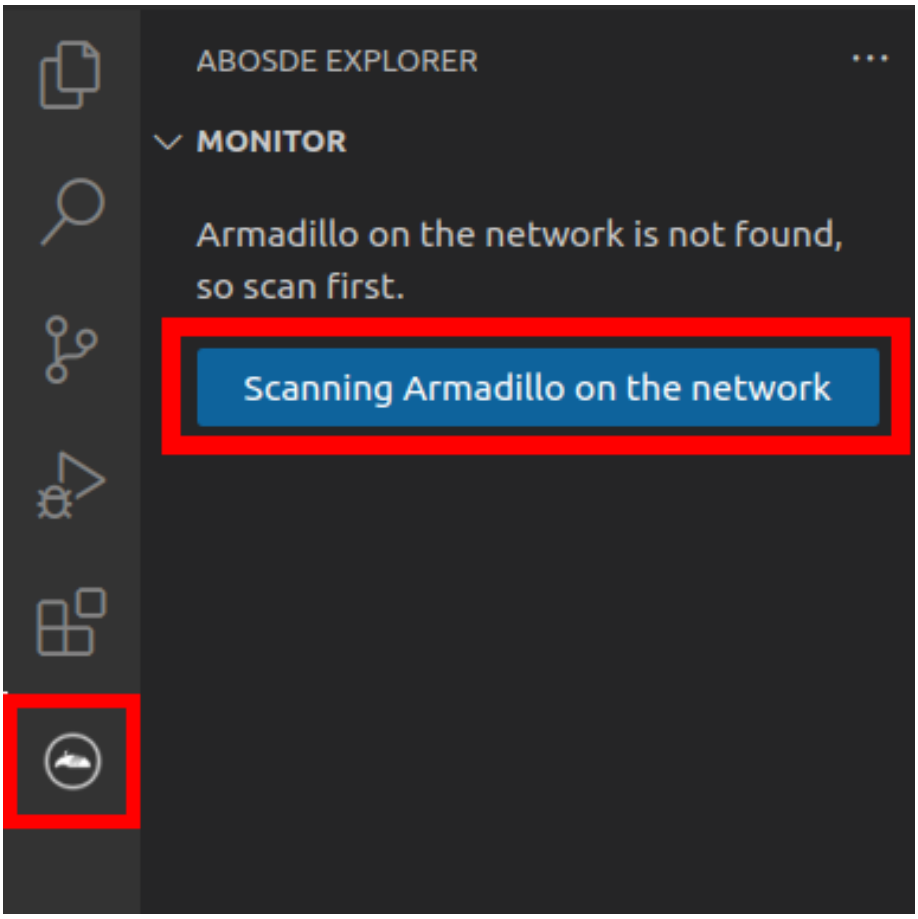


図 10.67 ABOSDE で ローカルネットワーク上の Armadillo をスキャンする

ABOSDE から ABOS Web に初めて通信を行う時、ABOS Web は通信に使用するためのトークンを発行します。そのため、ABOSDE では 「図 10.68. ABOSDE の ABOS Web パスワード入力画面」 のように ABOS Web のパスワードを求められますので、設定したパスワードを入力してください。

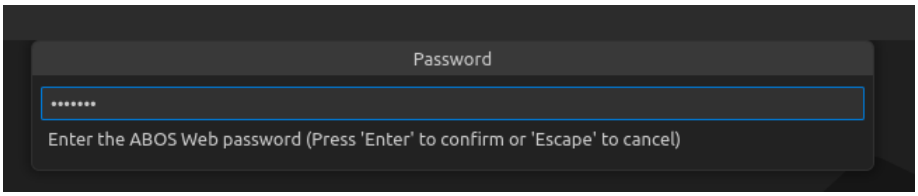


図 10.68 ABOSDE の ABOS Web パスワード入力画面

10.10.1. Armadillo の SWU バージョンを取得する

ローカルネットワーク上の Armadillo をスキャンした後に、「図 10.69. ABOSDE で Armadillo の SWU バージョンを取得」 の赤枠で囲まれているボタンをクリックすることで、選択した Armadillo の SWU バージョンを取得することができます。

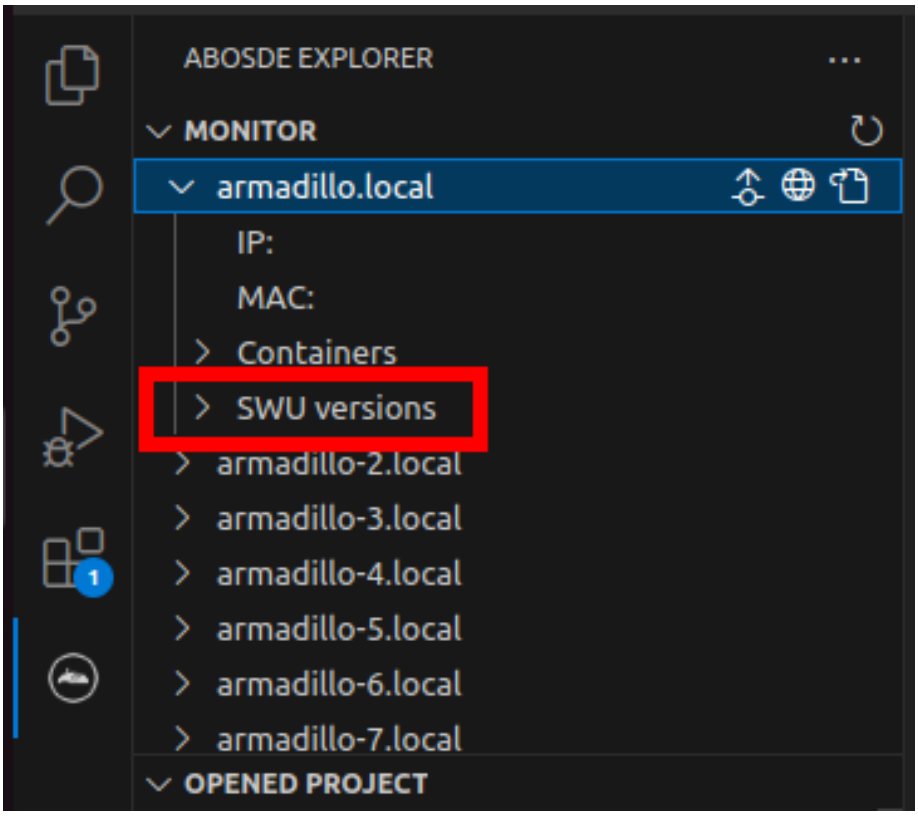


図 10.69 ABOSDE で Armadillo の SWU バージョンを取得

10.10.2. Armadillo のコンテナの情報を取得する

ローカルネットワーク上の Armadillo をスキャンした後に、「図 10.70. ABOSDE で Armadillo のコンテナ情報を取得」 の赤枠で囲まれているボタンをクリックすることで、選択した Armadillo のコンテナの情報を取得できます。表示されるコンテナの情報は以下の通りとなります。

- **state** : コンテナが起動中の場合は running、コンテナが停止中の場合は exited
- **image** : コンテナのイメージ名
- **command** : コンテナ起動時に実行しているコマンド

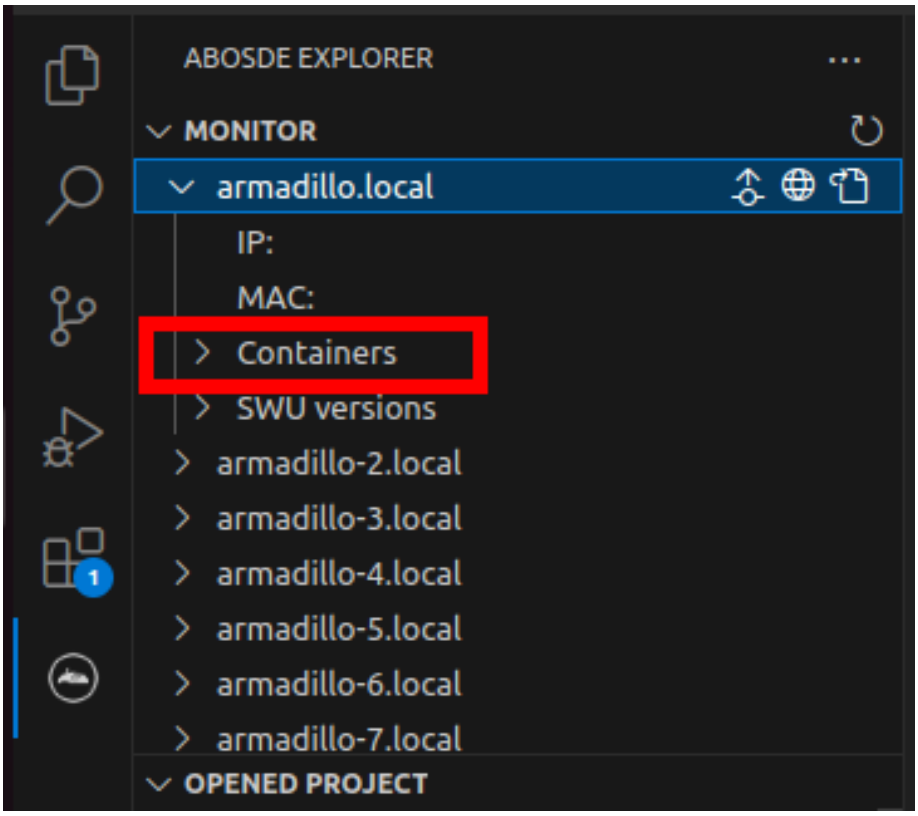


図 10.70 ABOSDE で Armadillo のコンテナ情報を取得

10.10.3. Armadillo のコンテナを起動・停止する

ローカルネットワーク上の Armadillo をスキャンした後に、「図 10.71. ABOSDE で Armadillo のコンテナを起動」 の赤枠で囲まれているボタンをクリックすることで、選択したコンテナを起動することができます。コンテナを起動できた場合はコンテナの status が running に変化します。また、「図 10.72. ABOSDE で Armadillo のコンテナを停止」 の赤枠で囲まれているボタンをクリックすることで、選択したコンテナを停止することができます。コンテナを停止できた場合はコンテナの status が exited に変化します。

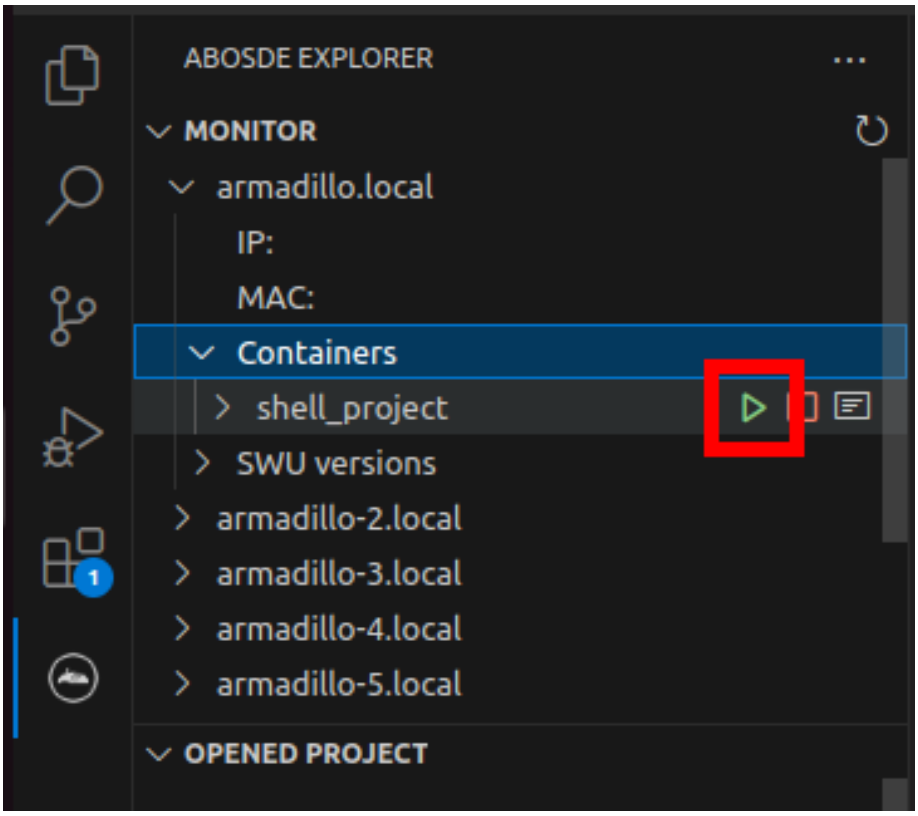


図 10.71 ABOSDE で Armadillo のコンテナを起動

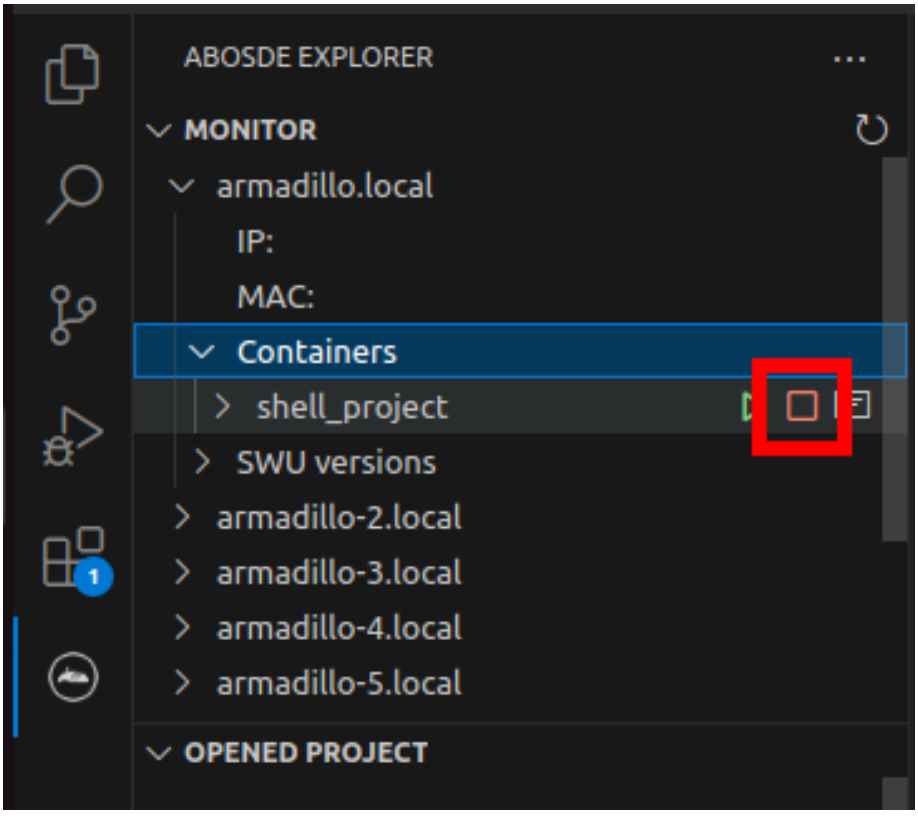


図 10.72 ABOSDE で Armadillo のコンテナを停止

10.10.4. Armadillo のコンテナのログを取得する

「図 10.73. ABOSDE で Armadillo のコンテナのログを取得」 の赤枠で囲まれているボタンをクリックすることで、コンテナが出力したログを取得することができます。ログは VS Code のテキストエディタに開かれます。コンテナが何もログを出力していない場合は表示されません。

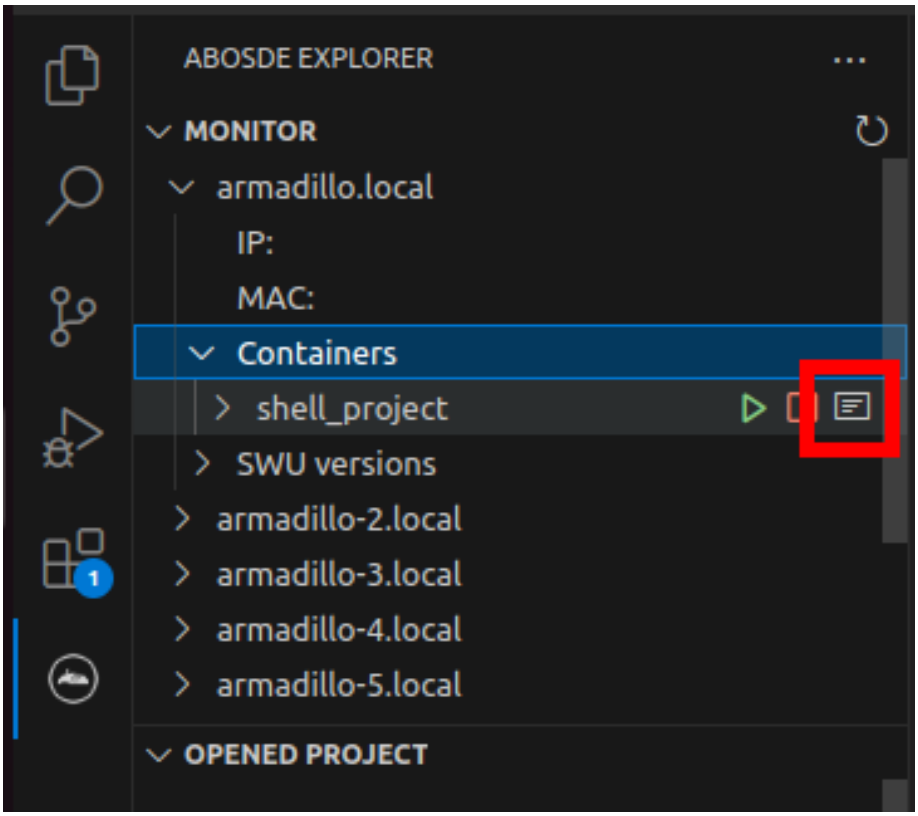


図 10.73 ABOSDE で Armadillo のコンテナのログを取得

10.10.5. Armadillo に SWU をインストールする

ローカルネットワーク上の Armadillo をスキャンした後に、「図 10.74. ABOSDE で Armadillo に SWU をインストール」 の赤枠で囲まれているボタンをクリックすることで、選択した Armadillo に SWU をインストールすることができます。SWU インストールのログは VS Code 画面下部の OUTPUT に表示されます。

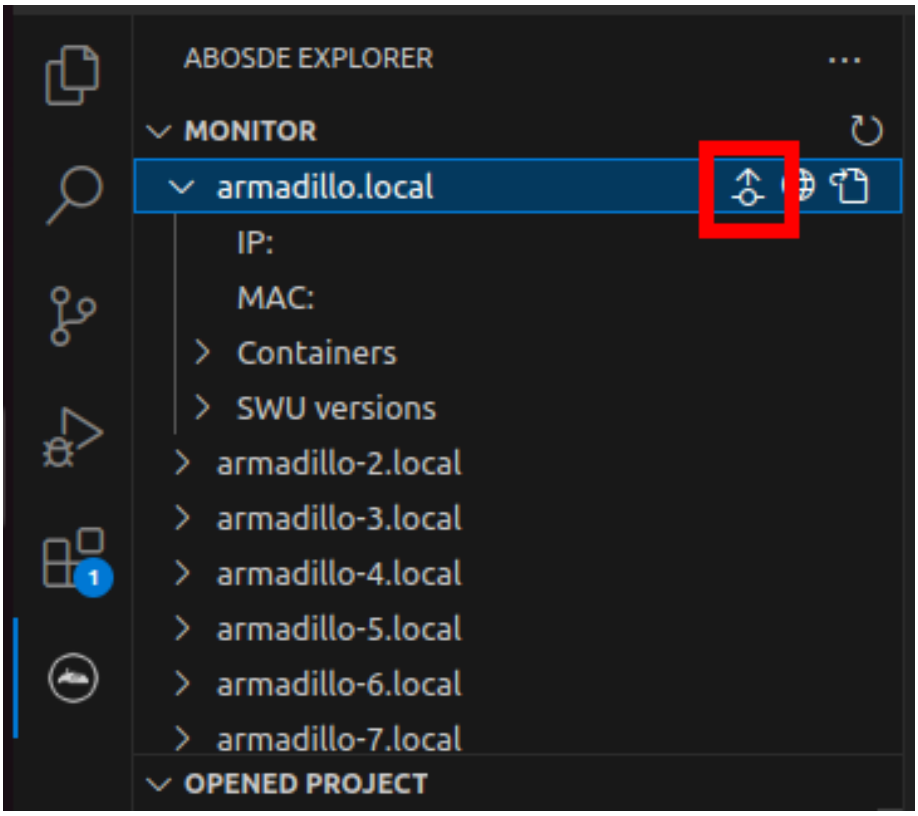


図 10.74 ABOSDE で Armadillo に SWU をインストール

10.11. ssh 経由で Armadillo Base OS にアクセスする

Armadillo-900 開発セット 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
[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。

10.12. 電源を安全に切るタイミングを通知する

Armadillo Base OS には、シャットダウン中に電源を切っても安全なタイミングで通知する機能があります。通知は GPIO 出力を用いて行います。どの GPIO 出力ピンを使うのかを Device Tree で設定します。Device Tree で通知用に設定された GPIO 出力ピンの出力レベルを変化させる動作は、シャットダウン中に実行される signal_indicator が行います。

Device Tree の設定手順を順に述べます。Device Tree の設定は、DTS overlays を使用して行います。

10.12.1. DTS overlays の設定

あらかじめ、CON16(拡張インターフェース)の 26 番ピンを使用する場合の Device Tree を用意してあります。

その場合は以下の、「10.12.1.1. CON16(拡張インターフェース)の 26 番ピンを使用する」 の設定を行ってください。これ以外のピンを使用する場合はデバイスツリーを記載してビルドする必要があります。

10.12.1.1. CON16(拡張インターフェース)の 26 番ピンを使用する

「10.18.1. DTS overlays によるカスタマイズ」 を参考にして /boot/overlays.txt に armadillo_iotg_a9e-stdwn-ind-con10-pin26.dtbo を追加してください。

10.12.2. 動作確認

ここまで述べた設定を行うと、シャットダウン動作中に通知が行われるようになります。シャットダウン動作の最後の方で、以下のメッセージを出力するのと同じタイミングで通知を行います。つまり、通知用に割り当てた GPIO 出力ピンの出力レベルが、0/Low から 1/High に変わります。シャットダウンが完了して SoC (CPU) への給電がオフすると、出力レベルが 0/Low に戻ります。出力レベルが 0/Low から 1/High に変化した時点以降であれば、Armadillo の電源を切っても安全です。

```
indicator_signals        | * Signaling external devices we are shutting down ...
```

図 10.75 indicator_signals のコンソール出力

10.13. Armadillo Base OS をアップデートする

Armadillo Base OS は SWUpdate によってアップデートすることができます。

アップデートする際には、rootfs ファイルシステムにインストールされたファイルをすべて消して、アップデートの中身と /etc/swupdate_preserve_files に記載されているファイルで新しい rootfs を作ります。「10.4. swupdate_preserve_files について」 を参照してください。

アップデートでファイルを削除してしまった場合に abos-ctrl mount-old で前のシステムを read-only でマウントして、削除されたファイルをコピーすることもできます。

10.14. ロールバック状態を確認する

Armadillo Base OS のルートファイルシステムが破損し起動できなくなった場合、自動的に以前のバージョンで再起動します。

abos-ctrl status コマンドでロールバックされてるかどうかを確認できます。

```
[armadillo ~]# abos-ctrl status
Currently booted on /dev/mmcblk0p1
Last update on Fri Jun  7 16:03:37 JST 2024, updated: ❶
  boot: 2020.4-at23-00001-g01508f65b8 -> 2020.4-at23
  base_os: 3.19.1-at.3.20240523.pc.gtr -> 3.19.1-at.4
rollback-status: OK: available, no auto-rollback ❷
```

図 10.76 abos-ctrl status の例

❶ 最新のアップデートの日付と内容が出力されています。

❷ 「表 10.7. rollback-status の出力と意味」「表 10.8. rollback-status 追加情報の出力と意味」 に示す状態と追加情報が出力されています。

表 10.7 rollback-status の出力と意味

出力	説明
OK	ロールバックされていません。
rolled back	ロールバックされています。

表 10.8 rollback-status 追加情報の出力と意味

出力	説明
no fallback (fresh install)	初期化状態。
no fallback	何かの理由で B 面が起動できない状態になっています（アップデート失敗後等)。
auto-rollback enabled (post-update)	アップデート直後でまだ再起動していない状態です。再起動して失敗した場合にロールバックが発生します。
auto-rollback enabled (cloned)	abos-ctrl rollback-clone コマンドを実行した後の状態です。同じくロールバック可能です。
available, no auto-rollback	アップデートの後に正常に起動できたので、自動ロールバックが無効になっていますが abos-ctrl rollback --allow-downgrade コマンドで手動ロールバック可能です。

Armadillo Base OS 3.19.1-at.4 以下のバージョンでは「アップデート直後」の概念がなかったため、ステータスは「no fallback」(B 面がない状態)、「optimal」(ロールバック可能)、と「rolled back」の 3 択だけでした。

必要な場合（例えば、自分のアプリケーションがアップデート直後に問題があった場合)、 abos-ctrl rollback で手動のロールバックも可能です。ロールバックにエラーがなければ、再起動してロールバックを完了します。

なお、/var/at-log/atlog に切り替えの際に必ずログを書きますので、調査の時に使ってください。以下の例では、Armadillo Base OS を更新した後に起動できないカーネルをインストールして、起動できなかったためにロールバックされました。

```
[armadillo ~]# cat /var/at-log/atlog
Jun  7 16:03:37 armadillo NOTICE swupdate: Installed update to /dev/mmcblk0p2: ¥
boot: 2020.4-at22 -> 2020.4-at23, base_os: 3.19.1-at.3 -> 3.19.1-at.4
Jun  7 16:11:39 armadillo NOTICE swupdate: Installed update to /dev/mmcblk0p1: ¥
extra_os.kernel: unset -> 5.10.218-1
Jun  7 16:12:18 armadillo WARNING uboot: reset by wdt
Jun  7 16:12:42 armadillo WARNING uboot: reset by wdt
Jun  7 16:13:06 armadillo WARNING uboot: reset by wdt
Jun  7 16:13:09 armadillo WARNING uboot: Counted 3 consecutive unfinished boots
Jun  7 16:13:09 armadillo WARNING uboot: Rolling back to mmcblk0p2
```

図 10.77 /var/at-log/atlog の内容の例

10.15. Armadillo 起動時にコンテナの外でスクリプトを実行する

起動時に何かスクリプトを走らせるためにはコンテナとして実行することを推奨します。 「10.8.3. コンテナ起動設定ファイルを作成する」 を参照してください。

コンテナで実行不可能な場合に、「local」サービスを使うことができます: /etc/local.d ディレクトリに.start ファイルを置いておくと起動時に実行されて、 .stop ファイルは終了時に実行されます。

```
[armadillo ~]# vi /etc/local.d/date_test.start ❶
#!/bin/sh

date > /tmp/boottest
[armadillo ~]# chmod +x /etc/local.d/date_test.start ❷
[armadillo ~]# persist_file /etc/local.d/date_test.start ❸
[armadillo ~]# reboot
: (省略)
[armadillo ~]# cat /tmp/boottest ❹
Tue Mar 22 16:36:12 JST 2022
```

図 10.78 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。

10.16. U-Boot

USB コンソールインターフェース 上で "t" キー以外のキーを押したまま電源を接続すると U-Boot のプロンプトが表示されます。何もキーを押していなければ自動的に Linux カーネルが起動します。

```
U-Boot SPL 2023.04-at3 (Apr 14 2025 - 05:58:59 +0000)
Normal Boot
ELE firmware version 1.0.0-344acb84

: (省略)

U-Boot 2023.04-at3 (Apr 14 2025 - 05:58:59 +0000)

M33 Sync: OK
CPU:   i.MX8ULP(Dual 5) rev1.2 at 800MHz
CPU current temperature: 26
Reset cause: POR
Boot mode: Single boot
Model: Atmark-Techno Armadillo-900
DRAM:    Hold key pressed for tests: t (fast) / T (slow)
992 MiB
Disabled RTC alarm
Core:  51 devices, 23 uclasses, devicetree: separate
MMC:   FSL_SDHC: 0, FSL_SDHC: 2
Loading Environment from MMC... OK
```

```
In:    serial
Out:   serial
Err:   serial
SEC0:  RNG instantiated
switch to partitions #0, OK
mmc0(part 0) is current device
flash target is MMC:0
Net:   eth0: ethernet@29950000
Fastboot: Normal
Normal Boot
Hit any key to stop autoboot:  0
=>
```

10.16.1. 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 10.79 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。

「10.19.1. ブートローダーをビルドする」 の際に u-boot のデフォルトを変更した場合や、u-boot のプロンプトで「setenv」や「saveenv」を実行しても、 /boot/uboot_env.d/00_defaults によって変更がアップデートの際にリセットされます。

00_defaults のファイルは Base OS の一部で更新されることもありますので、変更を望む場合は別のファイルを作って設定してください。ファイルはアルファベット順で処理されます。 00_defaults にある変数を後のファイルにも設定した場合はそのファイルの値だけが残ります。

主要な u-boot の環境変数を以下に示します。

表 10.9 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。	ttyLP0,115200
bootcount	起動回数を示します。初回起動時に 1 となり、起動に失敗する度にインクリメントされます。ユーザーランドの reset_bootcount サービスが起動されると、この値はクリアされます。この値が"bootlimit"を越えた場合はロールバックします。ロールバックの詳細については、「7.3.3.5. ロールバック(リカバリー)」を参照してください。	1
bootlimit	"bootcount"のロールバックを行うしきい値を指定します。	3
upgrade_available	1 以上の場合は bootcount を管理してロールバック可能になります。0 か空の場合はロールバックできません。値を abos-ctrl status で確認できます。	状況による
bootdelay	保守モードに遷移するためのキー入力を待つ時間を指定します(単位:秒)。次の値は特別な意味を持ちます。 ・ -1: キー入力の有無に関らず保守モードに遷移します。 ・ -2: キー入力の有無に関らず保守モードに遷移しません。	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、「10.18.1. DTS overlays 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

環境変数	説明	デフォルト値
mmcdev	"image"や"fdt_file"で指定されたファイルが配置してある mmc デバイスのインデックスを指定します。インデックスと mmc デバイスの対応は次の通りです。 ・ 0: eMMC ・ 1: microSD/microSDHC/microSDXC カード "mmcautodetect"に yes が指定されている場合は、u-boot の起動時に上書きされます。	0
mmcpart	"image"や"fdt_file"で指定されたファイルが配置してある、"mmcdev"で指定された mmc デバイスのパーティション番号を指定します。"mmcautodetect"に yes が指定されている場合は、u-boot の起動時に上書きされます。	1
mmcroot	ルートファイルシステムが配置されているデバイスノードと、マウントオプションを指定します。"mmcautodetect"に yes が指定されている場合は、u-boot の起動時に上書きされます。overlayfs が正しく機能しなくなる場合があるので、ro の指定は変更しないでください。	/dev/mmcblk0p1 rootwait ro
optargs	Linux カーネル起動時パラメータを指定します。"quiet"を削除すると、コンソールに起動ログが出力されるようになりますが、起動時間が長くなります。	quiet
loadaddr	Linux カーネルが RAM にロードされる物理アドレスを指定します。	0x80400000
fdt_addr	DTB が RAM にロードされる物理アドレスを指定します。	0x83000000
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。	0x83040000

10.16.1.1. u-boot の環境変数の変更を制限する

CONFIG_ENV_WRITEABLE_LIST=y を追加すると、変更可能と明示したもの以外の環境変数を変更不可にすることができます。CFG_ENV_FLAGS_LIST_STATIC で設定します。

デフォルトのコンフィグでは、以下の環境変数が変更可能です：

- `upgrade_available` と `bootcount`: ロールバック機能に必要な変数です。ロールバック機能を無効にする場合は必ず upgrade_available のデフォルト値も空にしてください。

u-boot のソースの取得方法、ビルド方法およびインストール方法については「10.19.1. ブートローダーをビルドする」を参照してください。ビルドしたものをインストールすると CFG_ENV_FLAGS_LIST_STATIC で設定した環境変数以外は変更できなくなります。

10.17. 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。

10.17.1. ブートディスクの作成

1. ブートディスクイメージをビルドします

「10.19.3. Alpine Linux ルートファイルシステムをビルドする」 で説明されているソースツリー alpine/build-rootfs にあるスクリプト build_image と 「10.19.1. ブートローダーをビルドする」 でビルドした u-boot-dtb.imx を利用します。

```
[ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_image.sh --board a900 ¥
          --boot ~/imx-boot-[VERSION]/imx-boot_armadillo-900
:（省略）
[ATDE ~/build-rootfs-[VERSION]]$ ls baseos-900*img
baseos-900-[VERSION].img
```

1. ブートディスクイメージの書き込みブートディスクイメージの書き込みは「3.2.1. 初期化インストールディスクの作成」を参照してください。

   参照先では初期化インストールディスクの場合の手順を示していますが、ここでビルドしたイメージについても同じ手順になります。

microSD カードのパーティション構成は次のようになっています。

表 10.10 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	100MiB	ファームウェア
10	710MiB	50MiB	セキュアブート用の A 面パーティション[a]
11	760MiB	50MiB	セキュアブート用の B 面パーティション[a]
5	860MiB	残り	アプリケーション用パーティション

[a]セキュアブートを有効にした場合に署名済み Linux カーネルイメージが格納されます。

gdisk で確認すると次のようになります。

```
[ATDE ~]$ sudo gdisk -l /dev/sdb
GPT fdisk (gdisk) version 1.0.6

Partition table scan:
  MBR: protective
  BSD: not present
  APM: not present
  GPT: present

Found valid GPT with protective MBR; using GPT.
Disk /dev/sdb: 60506112 sectors, 28.9 GiB
Model:
Sector size (logical/physical): 512/512 bytes
Disk identifier (GUID): 44B816AC-8E38-4B71-8A96-308F503238E3
Partition table holds up to 128 entries
```

```
Main partition table begins at sector 20448 and ends at sector 20479
First usable sector is 20480, last usable sector is 60485632
Partitions will be aligned on 2048-sector boundaries
Total free space is 0 sectors (0 bytes)

Number  Start (sector)    End (sector)  Size       Code  Name
   1           20480          634879   300.0 MiB   8300  rootfs_0
   2          634880         1249279   300.0 MiB   8300  rootfs_1
   3         1249280         1351679   50.0 MiB    8300  logs
   4         1351680         1761279   200.0 MiB   8300  firm
   5         1761280        60485632   28.0 GiB    8300  app
```

10.17.2. SD ブートの実行

「10.17.1. 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-900 開発セット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、SW4 を 起動デバイスは microSD 側設定します。SW4 に関しては、「図 5.166. スイッチの状態と起動デバイス」 を参照ください。

2. 電源を投入します。

```
U-Boot 2023.04-at1 (Apr 01 2025 - 03:59:06 +0000)

M33 Sync: OK
CPU:   i.MX8ULP(Dual 5) rev1.2 at 800MHz
CPU current temperature: 29
Reset cause: POR
Boot mode: Single boot
Model: Atmark-Techno Armadillo-900
DRAM:    Hold key pressed for tests: t (fast) / T (slow)
992 MiB
Core:  51 devices, 23 uclasses, devicetree: separate
MMC:   FSL_SDHC: 0, FSL_SDHC: 2
Loading Environment from MMC... OK
In:    serial
Out:   serial
Err:   serial
SEC0:  RNG instantiated
switch to partitions #0, OK
mmc2 is current device
flash target is MMC:2
Net:   eth0: ethernet@29950000
Fastboot: Normal
Normal Boot
Hit any key to stop autoboot:  0
switch to partitions #0, OK
mmc2 is current device
Failed to load 'boot/boot.scr'
23122432 bytes read in 2942 ms (7.5 MiB/s)
Booting from mmc ...
40248 bytes read in 45 ms (873 KiB/s)
Loading fdt boot/armadillo.dtb
Working FDT set to 83000000
```

```
...中略...

Welcome to Alpine Linux 3.21
Kernel 5.10.235-1-at on an aarch64 (/dev/ttyLP0)

armadillo login:
```

10.18. Device Tree をカスタマイズする

10.18.1. DTS overlays によるカスタマイズ

Device Tree は「DTS overlay」(dtbo) を使用することで変更できます。

DTS 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 DTS overlay を通常の dtb と結合して起動します。

複数の DTS 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 ~]# ls /boot/ ❶
armadillo_900-evaboard-con10-waveshare4-3-inch-dsi.dtbo
armadillo_900-evaboard-con9-ox01f10.dtbo
armadillo_900-evaboard.dtbo
armadillo_900-usdhc2-alwayson.dtbo
armadillo_900.dtb
armadillo_iotg_a9e-lbes5pl2el.dtbo
armadillo_iotg_a9e-nousb.dtbo
armadillo_iotg_a9e-sim7672.dtbo
armadillo_iotg_a9e-stdwn-ind-con10-pin26.dtbo
armadillo_iotg_a9e-stdwn-ind-do1.dtbo
armadillo_iotg_a9e.dtbo
Image
overlays.txt
uboot_env.d

[armadillo ~]# vi /boot/overlays.txt ❷
fdt_overlays=armadillo_900-evaboard.dtbo armadillo_iotg_a9e-lbes5pl2el.dtbo armadillo_iotg_a9e- ⏎
sim7672.dtbo armadillo_900-evaboard-con9-ox01f10.dtbo

[armadillo ~]# persist_file -vp /boot/overlays.txt ❸
'/boot/overlays.txt' -> '/mnt/boot/overlays.txt'
Added "/boot/overlays.txt" to /etc/swupdate_preserve_files

[armadillo ~]# reboot ❹
: (省略)
Applying fdt overlay: armadillo_900-evaboard.dtbo ❺
Applying fdt overlay: armadillo_iotg_a9e-lbes5pl2el.dtbo
Applying fdt overlay: armadillo_iotg_a9e-sim7672.dtbo
```

```
Applying fdt overlay: armadillo_900-evaboard-con9-ox01f10.dtb
:（省略）
```

図 10.80 /boot/overlays.txt の変更例

❶ /boot ディレクトリに保存されている dtbo ファイルを確認します。

❷ /boot/overlays.txt ファイルに使用したい dtbo ファイルを追加します。ファイルが存在しない場合は新規に作成してください。このファイルの詳細については 「10.18.1. DTS overlays によるカスタマイズ」 を参照してください。

❸ /boot/overlays.txt を保存し、アップデートの場合でも保存します。

❹ overlay の実行のために再起動します。

❺ シリアルコンソールの場合に、u-boot によるメッセージを確認できます。

10.18.1.1. 提供している DTS overlay

以下の DTS overlay を用意しています：

- armadillo_900-evaboard.dtbo: 自動的に使用します。
- armadillo_iotg_a9e-sim7672.dtbo: LTE Cat.1 bis モジュールに関する dtbo です。自動的に使用します。
- armadillo_iotg_a9e-lbes5pl2el.dtbo: WLAN+BT+TH コンボモジュールに関する dtbo です。自動的に使用します。
- armadillo_iotg_a9e-stdwn-ind-con10-pin26.dtbo: /boot/overlays.txt に記載することで使用できます。使用方法は「10.12. 電源を安全に切るタイミングを通知する」を参照ください。
- armadillo_900-evaboard-con10-waveshare4-3-inch-dsi.dtbo: Raspberry Pi 用 4.3 インチタッチスクリーン液晶用の dtbo です。/boot/overlays.txt に記載することで使用できます。
- armadillo_900-evaboard-con9-ox01f10.dtbo: KBCR-S08MM 用の dtbo です。/boot/overlays.txt に記載することで使用できます。

10.18.2. 独自の DTS overlay を追加する

標準イメージで提供している DTS overlay では不十分な場合に、独自の DTS overlay を作成し Armadillo へ適用する手順を示します。

1. 「10.19.2. Linux カーネルをビルドする」 を参照の上、最新版カーネルのビルドまで実施してください。

   以下、ATDE のホームディレクトリに linux-[VERSION] ディレクトリができている前提で進めます。

2. カスタマイズ用に用意している arch/arm64/boot/dts/freescale/armadillo_900-customize.dts を編集します。

```
[ATDE ~/linux-[VERSION]]$ vi arch/arm64/boot/dts/freescale/armadillo_900-customize.dts
```

図 10.81 armadillo_900-customize.dts の編集

3. 編集したファイルをビルドします。

```
[ATDE ~/linux-[VERSION]]$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- dtbs
  DTC     arch/arm64/boot/dts/freescale/armadillo_900-customize.dtbo
```

図 10.82 編集した dts ファイルのビルド

4. ビルドしてできた armadillo_900-customize.dtbo を Armadillo の /boot/ に配置します。

```
[armadillo ~]# ls /boot/armadillo_900-customize.dtbo
/boot/armadillo_900-customize.dtbo
```

図 10.83 ビルドした DTS overlay ファイルを Armadillo に配置

5. 配置した dtbo を永続化します。このとき、配置した dtbo が SWUpdate 時に消去されてしまわないように、 -p オプションを付与して dtbo を swupdate_preserve_files に追記させます。

```
[armadillo ~]# persist_file -vp /boot/armadillo_900-customize.dtbo
Added "/boot/armadillo_900-customize.dtbo" to /etc/swupdate_preserve_files
'/mnt/boot/armadillo_900-customize.dtbo' -> '/target/boot/armadillo_900-customize.dtbo'
```

図 10.84 ビルドした DTS overlay ファイルを永続化

6. /boot/overlays.txt に armadillo_900-customize.dtbo を追記し、/boot/overlays.txt を永続化します。

```
[armadillo ~]# vi /boot/overlays.txt
fdt_overlays=armadillo_900-evaboard.dtbo armadillo_iotg_a9e-lbes5pl2el.dtbo
armadillo_iotg_a9e-sim7672.dtbo armadillo_900-customize.dtbo ❶
[armadillo ~]# persist_file /boot/overlays.txt
```

図 10.85 /boot/overlays.txt の編集と永続化

❶ すでに別の dtbo ファイルが記載されている場合、スペースを挿入して後ろに追加してください。

7. Armadillo を再起動し、動作確認をします。

10.19. Armadillo のソフトウェアをビルドする

ここでは、Armadillo-900 開発セット で使用するソフトウェアのビルド方法を説明します。

10.19.1. ブートローダーをビルドする

ここでは、ATDE 上で Armadillo-900 開発セット 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 crossbuild-essental-arm64 bison flex ⏎
zlib1g-dev python3-pycryptodome python3-pyelftools python3-cryptography device-tree- ⏎
compiler
```

2. ソースコードの取得

Armadillo-900 開発セット ブートローダー [https://armadillo.atmark-techno.com/resources/software/armadillo-900/boot-loader] から「ブートローダー ソース」ファイル (imx-boot-[VERSION].tar.zst) を次のようにダウンロードします。

```
[ATDE ~]$ wget https://download.atmark-techno.com/armadillo-900/bootloader/imx-boot- ⏎
[VERSION].tar.zst
[ATDE ~]$ tar xf imx-boot-[VERSION].tar.zst
[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-900
:
: (省略)
:
Note: Please copy image to offset: IVT_OFFSET + IMAGE_OFFSET
cp flash.bin boot-spl-container.img
append u-boot-atf-container.img at 289 KB
3079+0 レコード入力
3079+0 レコード出力
3152896 bytes (3.2 MB, 3.0 MiB) copied, 0.00455921 s, 692 MB/s
make[1]: ディレクトリ '/home/atmark/imx-boot-[VERSION]/imx-mkimage/armadillo-900' から出ます
cp imx-mkimage/armadillo-900/flash.bin imx-boot_armadillo-900
```

初めてのビルドの場合、i.MX 8ULP 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。[1]

```
Welcome to NXP firmware-upower-1.3.0.bin

You need to read and accept the EULA before you can continue.

LA_OPT_NXP_Software_License v45 May 2023
:
: (省略)
:
Do you accept the EULA you just read? (y/N)
```

[1]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。

m33-firmware-at ビルド状態の確認

imx-boot のソースアーカイブにはビルド済みの M33 ファームウェアは含まれていますので、変更しない場合はビルド不要です。ビルドが必要な場合は gcc-arm-none-eabi パッケージをインストールすると自動的にビルドされます。詳細は 「7.6. RTOS ファームウェアの開発」 を参照ください。

```
[ATDE ~/imx-boot-[VERSION]]$ make
:（省略）
Not rebuilding m33-firmware-at.bin without arm-none-eabi-gcc ❶
:（省略）
[ATDE ~/imx-boot-[VERSION]]$ sudo apt install gcc-arm-none-eabi ⏎
❷
[ATDE ~/imx-boot-[VERSION]]$ make
:（省略）
[  3%] Linking C executable release/m33-firmware-at.elf
Memory region         Used Size  Region Size  %age Used
    m_interrupts:          792 B        800 B     99.00%
          m_text:        95608 B     253152 B     37.77%
m_m33_suspend_ram:           0 GB         16 KB      0.00%
m_a35_suspend_ram:           0 GB         16 KB      0.00%
          m_data:        61336 B       160 KB     37.44%
        m_ncache:           0 GB         32 KB      0.00%
make[3]: Leaving directory '/home/atmark/imx-boot-[VERSION]/ ⏎
m33_firmware_at/armgcc'
[100%] Built target m33-firmware-at.elf
make[2]: Leaving directory '/home/atmark/imx-boot-[VERSION]/ ⏎
m33_firmware_at/armgcc'
make[1]: Leaving directory '/home/atmark/imx-boot-[VERSION]/ ⏎
m33_firmware_at/armgcc'
cp m33_firmware_at/m33-firmware-at.bin imx-mkimage/ ⏎
armadillo-900/m33_image.bin ❸
:（省略）
```

❶ ビルドされなかった場合の出力

❷ パッケージのインストールコマンド

❸ ビルドされた場合の出力

4. インストール

ビルドしたブートローダーは、以下に示すどちらかの方法でインストールしてください。

・ swupdate でインストールする

mkswu の初期化を行った後に 提供されているスクリプトを使って SWU イメージを作成してください。

```
[ATDE ~/imx-boot-[VERSION]]$ echo 'swdesc_boot imx-boot_armadillo-900' > 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 「7.3.3.6. SWU イメージのインストール」を参照ください。

- 「10.17.1. 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-{product-image-name} を使ってください。

10.19.2. Linux カーネルをビルドする

ここでは、Armadillo-900 開発セット 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-900 開発セットでは、基本的には Linux カーネルイメージをビルドする必要はありません。「10.19.3. Alpine Linux ルートファイルシステムをビルドする」の手順を実施することで、標準の Linux カーネルイメージがルートファイルシステムに組み込まれます。

標準の Linux カーネルイメージは、アットマークテクノが提供する linux-at という Alpine Linux 用のパッケージに含まれています。

カスタマイズした Linux カーネルイメージを利用する場合は、以下に示す手順を参照してください。

1. ソースコードの取得

   Armadillo-900 開発セット Linux カーネル [https://armadillo.atmark-techno.com/resources/software/armadillo-900/linux-kernel] から「Linux カーネル」ファイル (linux-at-a9-[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-a9-[VERSION].tar
   [ATDE ~]$ tar xf linux-at-a9-[VERSION]/linux-[VERSION].tar.gz
   [ATDE ~]$ cd linux-[VERSION]
   ```

2. デフォルトコンフィギュレーションの適用

   次のコマンドを実行します。

   ```
   [ATDE ~/linux-[VERSION]]$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu-
   armadillo_a9_defconfig
   ```

3. Linux カーネルコンフィギュレーションの変更

   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。変更する際は、「図 10.86. Linux カーネルコンフィギュレーションの変更」に示すコマンドを実行します。

```
[ATDE ~/linux-[VERSION]]$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- menuconfig
```

図 10.86 Linux カーネルコンフィギュレーションの変更

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" を選択し、 カーネルコンフィギュレーションを確定します。

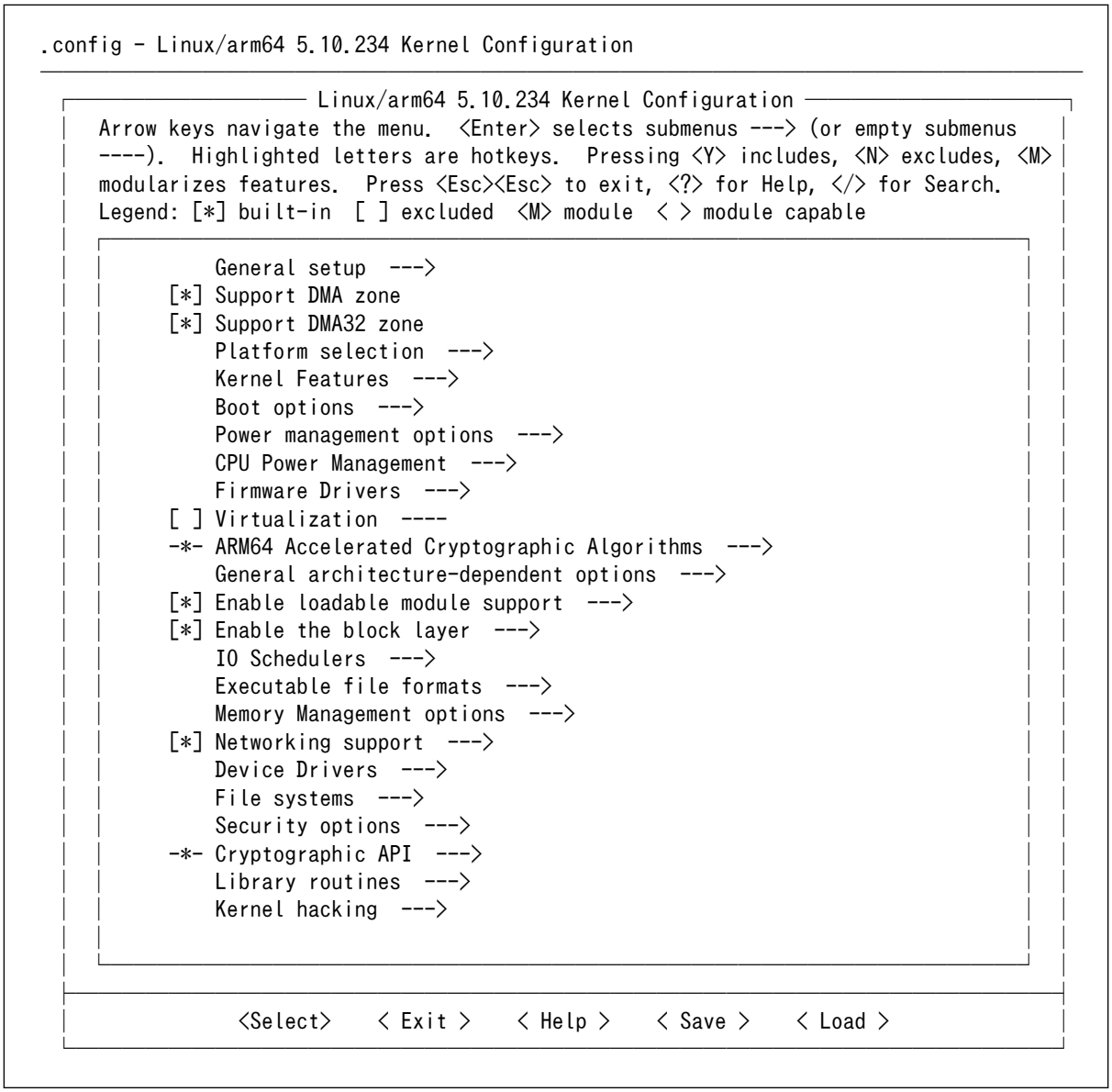


図 10.87 Linux カーネルコンフィギュレーション設定画面

Linux Kernel Configuration メニューで"/"キーを押下すると、カーネルコンフィギュレーションの検索を行うことができます。カーネルコンフィギュレーションのシンボル名(の一部)を入力し

て"Ok"を選択すると、部分一致するシンボル名を持つカーネルコンフィギュレーションの情報が一覧されます。

4. ビルド

次のコマンドを実行します。

```
[ATDE ~/linux-[VERSION]]$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- -j5
```

5. インストール

ビルドしたカーネルは、以下に示すどちらかの方法でインストールしてください。

・ swupdate でインストールする

mkswu の初期化を行った後に 提供されているスクリプトを使って SWU イメージを作成してください。

```
[ATDE ~/linux-[VERSION]]$ /usr/share/mkswu/examples/kernel_update_plain.install.sh ~/
mkswu/kernel.desc
Installing kernel in /home/atmark/mkswu/kernel ...
'arch/arm64/boot/Image' -> '/home/atmark/mkswu/kernel/Image'
'arch/arm64/boot/dts/freescale/armadillo_900.dtb' -> '/home/atmark/mkswu/kernel/
armadillo_900.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 10.88 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 「7.3.3.6. SWU イメージのインストール」を参照ください。

この kernel.swu をインストールする際は /etc/swupdate_preserve_files の更新例 の様に /boot と /lib/modules を維持するように追加します。カーネルをインストールした後に Armadillo Base OS を更新しても、この kernel.swu のカーネルが維持されます。

標準のカーネルに戻りたいか、以下の 「図 10.89. 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' "$BROOTFS/a900/packages" ❷
[ATDE ~/linux-[VERSION]]$ cp -v arch/arm64/boot/Image "$BROOTFS/a900/resources/boot/"
'arch/arm64/boot/Image' -> '/home/atmark/build-rootfs-v3.21-at.1/a900/resources/boot/
Image'
[ATDE ~/linux-[VERSION]]$ cp -v arch/arm64/boot/dts/freescale/armadillo_*.{dtb,dtbo}
"$BROOTFS/a900/resources/boot/"
'arch/arm64/boot/dts/freescale/armadillo_900.dtb' -> '/home/atmark/build-rootfs-v3.21-
at.1/a900/resources/boot/armadillo_900.dtb'
: (省略)
[ATDE ~/linux-[VERSION]]$ rm -rfv "$BROOTFS/a900/resources/lib/modules" ❸
[ATDE ~/linux-[VERSION]]$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu-
INSTALL_MOD_PATH="$BROOTFS/a900/resources" -j5 modules_install
  INSTALL arch/arm64/crypto/poly1305-neon.ko
  INSTALL drivers/block/loop.ko
: (省略)
  DEPMOD  [VERSION]
```

図 10.89 Linux カーネルを build_rootfs でインストールする方法

❶ build_rootfs のディレクトリ名を設定します。これによって、長いディレクトリ名を何度も入力する必要が無くなります。

❷ アットマークテクノが提供するカーネルをインストールしない様に、 linux-at-a9@atmark と記載された行を削除します。

❸ 別のカーネルをすでにインストールしている場合は、新しいモジュールをインストールする前に古いモジュールを削除する必要があります。

10.19.3. Alpine Linux ルートファイルシステムをビルドする

ここでは、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。

build-rootfs は、ATDE 上で Armadillo-900 開発セット 用の Alpine Linux ルートファイルシステムを構築することができるツールです。

1. ルートファイルシステムのビルドに必要な Podman のインストール

   次のコマンドを実行します。

   ```
   [ATDE ~]$ sudo apt install podman btrfs-progs xxhash
   ```

2. build-rootfs の入手

   {Armadillo-900 開発用ツール [https://armadillo.atmark-techno.com/resources/software/armadillo-900/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-900/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

a900 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 a900 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 a900 内のサブディレクトリにある同名ファイルは、a900 のファイルが利用されます。

build-rootfs に含まれるファイルの説明は次の通りです。

表 10.11 build-rootfs のファイル説明

ファイル	説明
a900/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。
a900/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。
a900/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。
a900/image_firstboot/*	配置したファイルやディレクトリは、「10.17.1. ブートディスクの作成」や「7.3.5.1. インストールディスクの作成」の手順のようにブートディスクイメージを作成する際、そのままルートファイルシステム直下にコピーされます。
a900/image_installer/*	配置したファイルやディレクトリは、「7.3.5.1. インストールディスクの作成」の手順のようにインストールディスクイメージを作成する際、そのままインストーラーにコピーされます。ルートファイルシステムに影響はありません。
a900/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
grpc-plugins-1.62.1-r2
jruby-9.3.13.0-r0
jruby-irb-9.3.13.0-r0
```

```
:
: (省略)
:
unit-ruby-1.34.1-r0
xapian-bindings-ruby-1.4.26-r0
```

4. ビルド

次のコマンドを実行します。

パッケージをインターネット上から取得するため回線速度に依存しますが、ビルドには数分かかります。

```
[ATDE ~/build-rootfs-[VERSION]]$ ./build_rootfs.sh -b a900
use default(outdir=/home/atmark/work/build-rootfs-v3.21-at.1)
use default(output=baseos-900-ATVERSION.tar.zst)
:
: (略)
:
INFO:root:Building SBOM...
INFO:root:created baseos-900-3.21.3-at.1.20250226.tar.zst.spdx.json

Successfully built /home/atmark/work/build-rootfs-v3.21-at.1/baseos-900-3.21.3-at.
1.20250226.tar.zst
```

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。

任意のパス、ファイル名で結果を出力することもできます。

```
[ATDE ~/build-rootfs-[VERSION]]$ ./build_rootfs.sh -b a900 ~/
alpine.tar.zst
:
: (略)
:
[ATDE ~/build-rootfs-[VERSION]]$ ls ~/alpine.tar.zst
~/alpine.tar.zst
```

「Alpine Linux ルートファイルシステムビルドツール」のバージョンが 3.18-at.7 以降を使用している場合は、ビルドが終わると SBOM も [output].spdx.json として出力されます。ライセンス情報等を記載するためのコンフィグファイルはデフォルトは baseos_sbom.yaml となっています。コンフィグファイルを変更する場合は --sbom-config <config> に引数を入れてください。SBOM が不要な場合は --nosbom を引数に追加してください。

SBOM のライセンス情報やコンフィグファイルの設定方法については 「10.20.3. ビルドしたルートファイルシステムの SBOM を作成する」 をご覧ください。

5. インストール

   ビルドしたルートファイルシステムは、以下に示すどちらかの方法でインストールしてください。

   - 「7.3.3.6. SWU イメージのインストール」 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-900-[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 「7.3.3.6. SWU イメージのインストール」 を参照ください。

   - 「10.17.1. 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-{product-image-name}-[VERSION].tar.zst が自動的に利用されます。

     アットマークテクノがリリースするファームウェアはバージョンのサフィックスとして"-at.[数字]"を含めています。オリジナルのサフィックスをつける場合は、"-at.[数字]"を使用しないことを強く推奨します。

10.20. SBOM の提供

アットマークテクノでは ABOS 及び ABOS 上で動作する標準ソフトウェアの SBOM を提供しています。また、開発したソフトウェアの SWU イメージを作成するタイミングで SBOM を生成することができます。SBOM 生成手順は 「10.20.3. ビルドしたルートファイルシステムの SBOM を作成する」 もしくは 「10.20.4. SWU イメージと同時に SBOM を作成する」 を参照ください。

10.20.1. SBOM について

SBOM（Software Bill of Materials: ソフトウェア部品表) は、ソフトウェアを構成するコンポーネントやソフトウェア間の依存関係、ライセンス情報を記したリストです。経済産業省は、ソフトウェアサプライチェーンが複雑化する中で、急激に脅威が増しているソフトウェアのセキュリティを確保するための管理手法の一つとして SBOM の導入を推進しています。SBOM の導入はソフトウェアのトレーサビリティを確保し、脆弱性残留リスクの低減、脆弱性対応期間の低減に繋がります。アットマークテクノが提供する SBOM は ISO/IEC5962 で国際標準となっている SPDX2.2 のフォーマットに準拠しています。

SPDX2.2 の詳細については以下のドキュメントをご参照ください。

The Software Package Data Exchange® (SPDX®) Specification Version 2.2.2 [https://spdx.github.io/spdx-spec/v2.2.2/]

アットマークテクノの提供する mkswu コマンドでは SWU を作成するタイミングで SBOM を生成することができます。

10.20.2. SBOM の利点

SBOM の利点はソフトウェアのサプライチェーン攻撃への対応です。ソフトウェアのセキュリティ対策は日々見直されており、トレーサビリティが明らかになることで、ソフトウェアに含まれる脆弱性に速やかに対処することが可能になります。SBOM はトレーサビリティを辿るのに優れており、加えて、脆弱性スキャンツールを用いることで、表面化していない脆弱性の発見に利用できます。脆弱性スキャンツールには例として、Google が提供する osv-scanner が挙げられます。脆弱性に関する詳細なリンクや、脆弱性の深刻度を示す CVSS(Common Vulnerability Scoring System) を出力します。アットマークテクノが提供する SBOM は osv-scanner のスキャンに対応しています。

osv-scanner を用いた SBOM のスキャンについては「7.12. 生成した SBOM をスキャンする」をご参照ください。

アットマークテクノが提供している ABOS は GPLv3（GNU General Public License 第 3 版）のソフトウェアを含まない構成で提供しています。OSS（オープンソース・ソフトウェア）利用者に広く普及している GPLv3 は、インストール用情報の開示義務、関連する特許ライセンスの許諾について定める条項が含まれ、組み込み機器に適用する際の妨げになる場合があります。SBOM にはパッケージのライセンス情報が含まれているため、GPLv3 ライセンスが含まれているかどうかの検出を可能にします。

10.20.3. ビルドしたルートファイルシステムの SBOM を作成する

「10.19.3. Alpine Linux ルートファイルシステムをビルドする」 を実行すると、OS_update.swu と同じ場所に SBOM を作成します。SBOM を作成するには、作成する対象のファイルとライセンス情報等を記載するためのコンフィグファイルが必要となります。また、baseos-900-[VERSION].tar.zst から、アーカイブに含まれるパッケージ情報やファイル情報を SBOM に記載します。

ライセンス情報等を記載するためのコンフィグファイルの例は以下のコマンドで確認することができます。各項目に関する説明はコメントに記載しておりますので、必要に応じて値を変更してください。各項目の詳細な説明については SPDX specification v2.2.2 (https://spdx.github.io/spdx-spec/v2.2.2/) をご覧ください。

```
[ATDE ~/build-rootfs-[VERSION]]$ cat submodules/make-sbom/config.yaml
```

作成したコンフィグファイルと、baseos-900-[VERSION].tar.zst から OS_update.swu の SBOM を作成します。

```
[ATDE ~/build-rootfs-[VERSION]]$ ./build_sbom.sh -i OS_update.swu -c <コンフィグファイル> -f ⏎
baseos-900-[VERSION].tar.zst
INFO:root:created OS_update.swu.spdx.json
```

作成される SBOM は OS_update.swu.spdx.json になります。json 形式で ISO/IEC5962 で国際標準となっている SPDX2.2 のフォーマットに準拠しています。

アットマークテクノが提供しているソフトウェアの SBOM はソフトウェアダウンロード [https://armadillo.atmark-techno.com/armadillo-900/resources/software]の各ソフトウェアダウンロードページからダウンロードすることができます。

10.20.4. SWU イメージと同時に SBOM を作成する

「9.3.1. SWU イメージの作成」 の実行時に SBOM を作成する方法について説明します。SWU イメージは desc ファイルから作成されます。この desc ファイルに SBOM 作成に必要な情報についても記載します。

10.20.4.1. コンフィグファイルを作成する

SBOM を作成するには、作成する対象のファイルとライセンス情報等を記載するためのコンフィグファイルが必要となります。コンフィグファイルについて指定がない場合はデフォルトのコンフィグファイルで SBOM を作成します。デフォルトのコンフィグファイルは /usr/share/make-sbom/config/config.yaml にあります。このファイルは SBOM 作成ツールによって配置されます。コンフィグファイルを編集するために、例としてカレントディレクトリにコピーします。リリース時には正しいコンフィグファイルの内容を記載してください。

```
[ATDE ~]$ cp /usr/share/make-sbom/config/config.yaml .
[ATDE ~]$ vi config.yaml
```

ライセンス情報等を記載するためのコンフィグファイルの例は以下のコマンドで確認することができます。各項目に関する説明はコメントに記載しておりますので、必要に応じて値を変更してください。各項目の詳細な説明については SPDX specification v2.2.2 (https://spdx.github.io/spdx-spec/v2.2.2/) をご覧ください。

「10.20.4.2. desc ファイルを編集する」 で desc ファイルに編集したコンフィグファイルのパスを指定します。

10.20.4.2. desc ファイルを編集する

SBOM 作成のために、desc ファイルに記載する項目を以下に示します。

表 10.12 desc ファイルの設定項目

項目	設定値	説明
swdesc_option BUILD_SBOM=<mode>	auto(デフォルト): SBOM 作成ツールがある場合作成する	SBOM を作成するかどうか。記載がない場合は auto が選択される
	yes: SBOM を作成する。SBOM 作成ツールがない場合はエラーする	
	no: SBOM を作成しない	
swdesc_option sbom_config_yaml=<path>	ファイルパス	コンフィグファイルのパスを指定する。記載がない場合はデフォルトのコンフィグファイルを使用する
swdesc_sbom_source_file <path>	ファイルパス	SBOM に含めるファイルを指定する。記載がない場合は SBOM に含まれない

以下に desc ファイルの記載例について示します。

```
swdesc_option component=make_sbom
swdesc_option version=1
swdesc_option BUILD_SBOM=yes❶
swdesc_option sbom_config_yaml=config.yaml❷

swdesc_sbom_source_file manifest.json❸
```

図 10.90 desc ファイルの追加例

❶ SBOM を作成するように設定します。例として必ず作成するように "yes" を指定します。

❷ コンフィグファイルのパスを設定します。例としてカレントディレクトリにある config.yaml を指定します。

❸ SBOM に含めたいファイルがある場合に指定します。例として manifest.json を指定します。

desc ファイルの作成が出来たら 「9.3.1. SWU イメージの作成」 を実行すると、SWU イメージと同じ場所に SBOM が作成されます。desc ファイルの内容によっては SBOM 作成に数分かかります。作成される SBOM のファイル名は <SWU イメージ名>.spdx.json になります。json 形式で ISO/IEC5962 で国際標準となっている SPDX2.2 のフォーマットに準拠しています。

10.21. 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-900 開発セットに搭載の eMMC は、eMMC 自身にデータリテンション機能が備わっており、Armadillo に電源が接続されて eMMC に電源供給されている状態で、eMMC 内部でデータリテンション処理が自動実行されます。

10.22. 動作ログ

10.22.1. 動作ログについて

Armadillo-900 開発セット ではシステムが出力するログの一部は、一般的な /var/log ディレクトリではなく、/var/at-log ディレクトリに出力されます。/var/at-log は、ルートファイルシステムとは別のパーティションになっているので、ルートファイルシステムに障害が発生した場合でも、/var/at-log のパーティションが無事であれば、ログファイルを取り出して、不具合等の解析に利用することができます。

Armadillo-900 開発セット で /var/log 配下に出力するログに関しては 「10.22.5. /var/log/ 配下のログに関して」 を参照ください。

10.22.2. 動作ログを取り出す

ログファイルは /var/at-log ディレクトリ内に atlog というファイル名で作成されているので、これを任意のディレクトリにコピーすることで取り出せます。もし、eMMC 上のルートファイルシステムが壊れてしまい起動できない場合は、microSD カードから起動することでログファイルを取り出すことができます。

/var/at-log/atlog はファイルサイズが 3MiB になるとローテートされ /var/at-log/atlog.1 に移動されます。

/var/at-log/atlog.1 が存在する状態で、更に /var/at-log/atlog のファイルサイズが 3MiB になった場合は、 /var/at-log/atlog の内容が /var/

at-log/atlog.1 に上書きされます。 /var/at-log/atlog.2 は生成されません。

10.22.3. 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 10.91 動作ログのフォーマット

atlog には以下の内容が保存されています。

- インストール状態のバージョン情報
- swupdate によるアップデートの日付とバージョン変更
- abos-ctrl / uboot の rollback 日付
- uboot で wdt による再起動があった場合にその日付

10.22.4. ログ用パーティションについて

ログ出力先である /var/at-log ディレクトリには、GPP である /dev/mmcblk0gp1 パーティションがマウントされています。このパーティションに論理的な障害が発生した場合は、/dev/mmcblk0gp1 のデータを /dev/mmcblk0gp2 にコピーし、/dev/mmcblk0gp1 は FAT ファイルシステムでフォーマットされます。このパーティションの障害チェックはシステム起動時に自動的に実行されます。

10.22.5. /var/log/ 配下のログに関して

「表 10.13. /var/log/ 配下のログ」 に Armadillo-900 開発セット で /var/log/ 配下に出力するログを示します。

最大ファイルサイズを超えると 「表 10.13. /var/log/ 配下のログ」 の「ファイル名」の 2 行目に記載されたファイル名にコピーします。

その状態から更に最大ファイルサイズを超えた場合、 「表 10.13. /var/log/ 配下のログ」 の「ファイル名」の 2 行目に記載されたファイル名に上書きします。

表 10.13 /var/log/ 配下のログ

ファイル名	説明	最大ファイルサイズ	最大ファイル数
/var/log/messages /var/log/messages.0	通常のログです。	4MiB	2
/var/log/armadillo-twin-agent/agent_log /var/log/armadillo-twin-agent/agent_log.1	Armadillo Twin Agent の動作ログです。	1MiB	2

10.23. ATDE・Linux でインストールディスクを作成する

ATDE や Linux でインストールディスクイメージからインストールディスクを作成する方法を紹介します。(Windows でインストールディスクを作成する方法については「3.2.1. 初期化インストールディ

スクの作成」を参照してください。参照先は初期化インストールディスクイメージを使用した方法ですが、それ以外のインストールディスクイメージでも同様の手順で行うことができます。）

ここでは、例として ATDE で初期化インストールディスクを作成する 2 種類の方法（CUI・GUI）を記載しています。初期化インストールディスクイメージ以外のインストールディスクイメージでも同様の手順で行うことができます。

いずれの方法でも、まずは次の共通の手順を実施してください。

1. 1 GB 以上の microSD カードを用意してください。

2. 標準のインストールディスクイメージをダウンロードします。Armadillo-900 開発セット インストールディスクイメージ [https://armadillo.atmark-techno.com/resources/software/armadillo-900/disc-image] から「Armadillo Base OS インストールディスクイメージ」をダウンロードしてください。

10.23.1. GUI でインストールディスクを作成する

1. ダウンロードした zip ファイルを展開します。図のとおり、zip ファイルを選択して右クリックし、「ここで展開」をしてください。

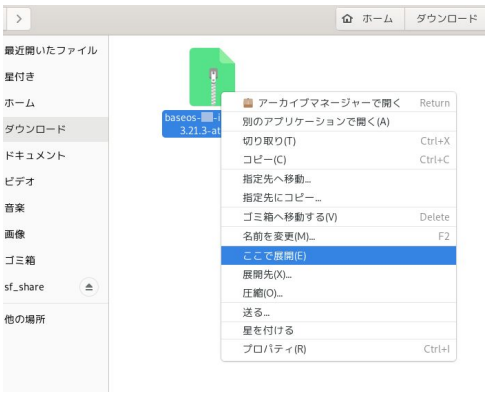


図 10.92 zip ファイルを展開

2. ATDE に microSD カードを接続します。詳しくは「7.1.1.7. 取り外し可能デバイスの使用」を参考にしてください。

3. 展開したフォルダ内にあるインストールディスクイメージ（.img）をダブルクリックして、「ディスクイメージをリストア」のウィンドウを表示します。

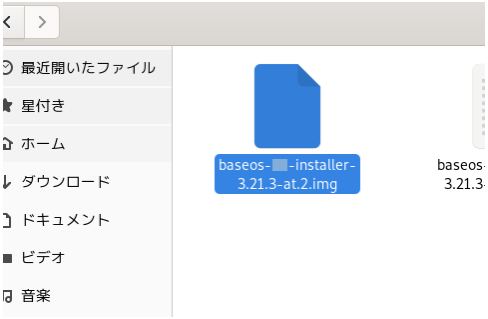


図 10.93 展開したフォルダ内にある img ファイルをダブルクリック

図 10.94 ディスクイメージをリストア

4. microSD カードを指定します。[転送先]から接続した microSD カードに対応するものを選びます。

[転送先]は細心の注意を払って選んでください。[転送先]には ATDE のシステムディスク（右側に VBOX HARDDISK や /dev/atde9-vg* と記載されているもの）も表示されていますが、**これらは決して選択しないでください。**誤って選択して書き込んでしまった場合、ATDE が破壊される可能性があります。

また、他に接続されているデバイスに意図せず書き込んでしまった場合、そのデバイスのデータが上書きされてしまいます。選んだ[転送先]が目的の microSD カードなのかどうか、しっかりと確認してください。microSD カードやカードリーダーの挿抜による[転送先]の表示の変化で確かめることもできます。

図 10.95 microSD カードを指定

5. [リストアを開始]ボタンをクリックして、書き込みを開始します。
6. 確認のウィンドウが現れるので、今一度確認した上で[リストア]をクリックします。

図 10.96 確認のウィンドウ

7. パスワードが要求されるので入力します。

図 10.97 パスワードの要求

8. 書き込みが完了したら、 microSD カードを取り外します。

10.23.2. CUI でインストールディスクを作成する

1. ATDE に microSD カードを接続します。詳しくは「7.1.1.7. 取り外し可能デバイスの使用」を参考にしてください。

2. microSD カードのデバイス名を確認します

```
[ATDE ~]$ ls /dev/sd?
/dev/sda  /dev/sdb
[ATDE ~]$ sudo fdisk -l /dev/sdb
Disk /dev/sdb: 7.22 GiB, 7751073792 bytes, 15138816 sectors
Disk model: SD/MMC
:(省略)
```

3. microSD カードがマウントされている場合、アンマウントします。

```
[ATDE ~]$ mount
:(省略)
/dev/sdb1 on /media/52E6-5897 type ext2
(rw,nosuid,nodev,relatime,uid=1000,gid=1000,fmask=0022,dmask=0077,codepage=cp437,iocharset
```

```
=utf8,shortname=mixed,showexec,utf8,flush,errors=remount-ro,uhelper=udisks)
[ATDE ~]$ sudo umount /dev/sdb1
```

4. ダウンロードしたファイルを展開し、以下の dd コマンドで img ファイルを microSD カードに書き込んでください。

```
[ATDE ~]$ unzip baseos-900-installer-[VERSION].zip
[ATDE ~]$ sudo dd if=baseos-900-installer-[VERSION].img ¥
                 of=/dev/sdb bs=1M oflag=direct status=progress
```

5. 書き込みが完了したら、 microSD カードを取り外します。

10.24. シリアル通信ソフトウェア(minicom)

Linux や ATDE で使用できるシリアル通信ソフトウェアとして minicom の使用方法を紹介します。

10.24.1. シリアル通信ソフトウェア(minicom)のセットアップ

ATDE9 v20240925 以降の ATDE では以下の設定を実施した状態のイメージを配布しています。これより前のバージョンの場合は、次の手順に沿って minicom のシリアル通信設定を実施してください。

minicom を使用して Armadillo とシリアルコンソール経由で通信を行うためには、「表 10.14. シリアル通信設定」 のとおりにあらかじめ設定しておく必要があります。ここでは、その設定手順について説明します。また、minicom を起動する端末の横幅を 80 文字以上にしてください。横幅が 80 文字より小さい場合、コマンド入力中に表示が乱れることがあります。

表 10.14 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 10.98. 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 10.98 minicom の設定の起動

2. 「図 10.99. 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 10.99 minicom の設定

3. 「図 10.100. 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 10.100 minicom のシリアルポートの設定

4. Serial Device に使用するデバイスファイル名として /dev/ttyUSB0 を入力して Enter キーを押してください。

デバイスファイル名の確認方法

デバイスファイル名は、環境によって /dev/ttyS0 や /dev/ttyUSB1 など、本書の実行例とは異なる場合があります。

その場合は以下の方法でデバイスファイル名を確認してください。

Linux で PC と Armadillo 側のシリアルポートを接続した場合、コンソールに以下のようなログが表示されます。ログが表示されなくても、dmesg コマンドを実行することで、ログを確認することができます。

例. シリアルポート接続時のログ. 上記の例では Armadillo 側のシリアルポートが ttyUSB0 に割り当てられたことが分かります。

5. F キーを押して Hardware Flow Control を No に設定してください。

6. G キーを押して Software Flow Control を No に設定してください。

7. キーボードの E キーを押してください。「図 10.101. 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 10.101 minicom のシリアルポートのパラメータの設定

8. 「図 10.101. 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 10.102. minicom シリアルポートの設定値」を参照してください。

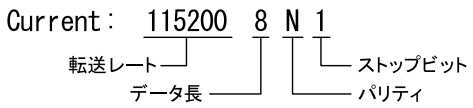


図 10.102 minicom シリアルポートの設定値

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 10.99. minicom の設定」に戻ってください。

13. 「図 10.99. minicom の設定」から、「Save setup as dfl」を選択し、設定を保存してください。

14. 「Exit from Minicom」を選択し、minicom の設定を終了してください。

Ctrl-a に続いて z キーを入力すると、minicom のコマンドヘルプが表示されます。

10.24.2. minicom の起動

ATDE の場合は、minicom を起動する前に シリアルコンソールとして使用する取り外し可能デバイスに示すデバイスを ATDE に接続してください。

シリアルコンソールとして使用する取り外し可能デバイス. 「図 10.103. minicom 起動方法」のようにして、minicom を起動してください。また、minicom を起動する端末の横幅を 80 文字以上にしてください。横幅が 80 文字より小さい場合、コマンド入力中に表示が乱れることがあります。

```
[ATDE ~]$ sudo LANG=C minicom --wrap --device /dev/ttyUSB0
```

図 10.103 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
```

10.24.3. minicom の終了

minicom を終了させるには、まず Ctrl-a に続いて q キーを入力します。その後、以下のように表示されたら「Yes」にカーソルを合わせて Enter キーを入力すると minicom が終了します。

```
+-----------------------+
| Leave without reset? |
|     Yes       No     |
+-----------------------+
```

図 10.104 minicom 終了確認

10.25. 不正な USB デバイスの接続を拒否する

USB 接続制御機能は、 ABOS バージョン 3.20.3-at.8 以降で対応しています。

この機能に関して、ABOS バージョン 3.21.3-at.12 以降であれば ABOS Web で設定することも可能です。

「5.4. USB デバイスの接続を許可する」をご参照ください。

```
[armadillo ~]# abos-ctrl usb-filter help
Usage: abos-ctrl usb-filter [action [arguments]]

Possible actions:
  enable: enabling the USB filter function
  disable: disabling the USB filter
  list-devices [--verbose]: list currently connected USB devices
  list-rules [--verbose]: list currently USB device allowlist
  reset-rules [--force]: reset USB device allowlist
                         with --force option, do not prompt
  allow-device [--vendor-id VID] [--product-id PID]
               [--model MODEL] [--usb-interfaces INTERFACES]
               [--serial SERIAL]: add device to the allowlist
                                  Omitted parameters are stored as wildcard
                                  parameters can be checked with list-devices and
                                  list-rules --verbose flag
  allow-device {DEVID}: allow connected USB devices specified by ID to connect,
                        and add device's information to the allowlist
                        ID can be checked with `abos-ctrl usb-filter list-devices`
  block-device {DEVID}: block alllowed USB devices specified by ID to connect,
                        and delete device's information from the allowlist
                        ID can be checked with `abos-ctrl usb-filter list-devices`
  allow-class [CLASS]...: create allow rule for each USB device class
                          the string for 'CLASS' can be found by running
                          `abos-ctrl usb-filter allow-class`
  remove-rule {RULEID}...: remove the allow rule specified by ID
                           ID can be checked with `abos-ctrl usb-filter list-rules`
  help: show this message
```

図 10.105 USB 接続制御機能を管理するコマンド

SWUpdate から USB 接続制御機能の設定を変更する手順を Armadillo サイトの Howto で公開していますので、必要な場合はご参照ください。

Armadillo サイト Howto 「SWUpdate を用いて USB 接続制御機能の設定を行う」

https://armadillo.atmark-techno.com/howto/setting-usb-filter-with-swupdate

10.25.1. USB 接続制御機能を有効/無効化する

現在 USB 接続制御機能が有効か無効かは「図 10.106. USB 接続制御機能の状態を確認する」に示すコマンドを実行することで確認できます。

```
[armadillo ~]# abos-ctrl usb-filter
Currently USB filter is disabled.
```

図 10.106 USB 接続制御機能の状態を確認する

「図 10.107. USB 接続制御機能を有効化する」に示すコマンドを実行することで、 USB 接続制御機能を有効化できます。

```
[armadillo ~]# abos-ctrl usb-filter enable
USB filter enabled.
please reboot to apply rules to currently connected devices
```

図 10.107 USB 接続制御機能を有効化する

有効化したあとに接続された USB デバイスは、設定した許可ルールにしたがって許可/拒否されます。デフォルトでは全てのデバイスは拒否されます。

「図 10.108. USB 接続制御機能を無効化する」に示すコマンドを実行することで、 USB 接続制御機能を無効化できます。

```
[armadillo ~]# abos-ctrl usb-filter disable
USB filter disabled.
please reboot to apply rules to currently connected devices
```

図 10.108 USB 接続制御機能を無効化する

10.25.2. 接続済みの USB デバイスの一覧を表示する

「図 10.109. 接続されている USB デバイスをリストする」に示すコマンドを実行することで、Armadillo に接続されている全ての USB デバイスのリストを表示します。

```
[armadillo ~]# abos-ctrl usb-filter list-devices
     1  block "001:003" "046d" "08e5" "HD_Pro_Webcam_C920" ":0e0100:0e0200:010100:010200:" ⏎
"046d_HD_Pro_Webcam_C920" "/devices/platform/soc@0/32f10100.usb/38100000.dwc3/xhci-hcd.1.auto/ ⏎
usb1/1-1"
```

図 10.109 接続されている USB デバイスをリストする

1 行につき 1 つのデバイスの情報をスペース区切りで示しています。各列が何を示しているかは「表 10.15. デバイスリストの各列の意味」を参照してください。

表 10.15 デバイスリストの各列の意味

1 列目	そのデバイスに割り当たっている ID です。USB デバイスを許可/拒否する際に識別子として使用されます
2 列目	そのデバイスが現在許可(allow)されているか。拒否(block)されているかを示します
3 列目	そのデバイスに割り当たっている USB バス番号とデバイス番号のペアです
4 列目	そのデバイスのベンダー ID です
5 列目	そのデバイスのモデル ID です
6 列目	そのデバイスのモデル名です

7 列目	そのデバイスの USB クラスコードです。デバイスによっては複数存在するものもあり、":"(コロン)区切りで表示されます
8 列目	そのデバイスのシリアル番号です。一般に同じ製品であっても個体ごとに一意な値です
9 列目	そのデバイスに割り当たっている/sys 以下のディレクトリパスです

10.25.3. USB デバイスの接続を許可する

「図 10.109. 接続されている USB デバイスをリストする」のコマンドを実行して、2 列目が「block」となっているデバイスは拒否されているデバイスです。このデバイスに対して、「図 10.110. 指定した USB デバイスを許可する」に示すコマンドを実行することで接続を許可し、今後再接続されたとしても許可します。

```
[armadillo ~]# abos-ctrl usb-filter allow-device 1 ❶
allowed the following device:
    "001:003" "046d" "08e5" "HD_Pro_Webcam_C920" ":0e0100:0e0200:010100:010200:"
"046d_HD_Pro_Webcam_C920" "/devices/platform/soc@0/32f10100.usb/38100000.dwc3/xhci-hcd.1.auto/
usb1/1-1" ❷
```

図 10.110 指定した USB デバイスを許可する

❶ この例では、「図 10.109. 接続されている USB デバイスをリストする」のコマンドを実行して ID が 1 のデバイスを許可しています

❷ 許可されたデバイスの情報が表示されます

また、ベンダー/モデル ID やシリアル番号などの情報を前もって知っている場合は、それらの情報を用いて USB デバイスの接続を許可することも可能です。

```
[armadillo ~]# abos-ctrl usb-filter allow-device ¥
--vendor-id "046d" ¥
--product-id "08e5" ¥
--model "HD_Pro_Webcam_C920" ¥
--usb-interfaces ":0e0100:0e0200:010100:010200:" ¥
--serial "046d_HD_Pro_Webcam_C920"
allowed the following device:
    "046d" "08e5" "HD_Pro_Webcam_C920" ":0e0100:0e0200:010100:010200:" "046d_HD_Pro_Webcam_C920"
```

図 10.111 パラメータで指定した USB デバイスを許可する

「図 10.111. パラメータで指定した USB デバイスを許可する」において、1 つ以上のパラメータが指定されていない場合、その指定しなかったパラメータはワイルドカードとして扱われます。この場合、デバイスを接続したときに、指定したパラメータさえ完全に一致していれば、ワイルドカードとして扱われているパラメータが何であってもデバイスの接続が許可されます。

Armadillo に接続されている USB デバイスであれば 「図 10.112. Armadillo に接続している USB デバイスのパラメータを調べる」に示すコマンドを実行することで、「図 10.111. パラメータで指定した USB デバイスを許可する」に指定すべきパラメータを調べることができます。

```
[armadillo ~]# abos-ctrl usb-filter list-devices -v
     1  block "001:003" --vendor-id "046d" --product-id "08e5" --model
```

```
”HD_Pro_Webcam_C920” --usb-interfaces ”:0e0100:0e0200:010100:010200:” --
serial ”046d_HD_Pro_Webcam_C920” ”/devices/platform/soc@0/32f10100.usb/
38100000.dwc3/xhci-hcd.1.auto/usb1/1-1”
```

図 10.112 Armadillo に接続している USB デバイスのパラメータを調べる

10.25.4. USB デバイスの接続を拒否する

「図 10.109. 接続されている USB デバイスをリストする」のコマンドを実行して、2 列目が「allow」となっているデバイスは許可されているデバイスです。このデバイスに対して、「図 10.113. 指定した USB デバイスを拒否する」に示すコマンドを実行することで接続を拒否し、今後再接続されたとしても拒否します。

```
[armadillo ~]# abos-ctrl usb-filter block-device 1 ❶
blocked the following device:
    ”001:003” ”046d” ”08e5” ”HD_Pro_Webcam_C920” ”:0e0100:0e0200:010100:010200:”
”046d_HD_Pro_Webcam_C920” ”/devices/platform/soc@0/32f10100.usb/38100000.dwc3/xhci-hcd.1.auto/
usb1/1-1” ❷
```

図 10.113 指定した USB デバイスを拒否する

❶ この例では、「図 10.109. 接続されている USB デバイスをリストする」のコマンドを実行して ID が 1 のデバイスを拒否しています

❷ 拒否されたデバイスの情報が表示されます

10.25.5. USB デバイスクラス単位で USB デバイスの接続を許可する

「10.25.3. USB デバイスの接続を許可する」の手順では、 USB デバイスのベンダー ID やプロダクト ID、シリアル番号などが完全一致したデバイスのみを許可します。そのため同じメーカーの同じデバイスであっても、シリアル番号が一致しないと許可されません。ここでは、「USB メモリの接続はどのメーカーのどの製品であっても全て許可するが、 USB カメラなどの接続は拒否する」といったルールを手軽に作成する機能を紹介します。

abos-ctrl usb-filter allow-class コマンドを用いて、指定した USB デバイスクラスのみを許可することができます。例として、「図 10.114. 指定した USB デバイスクラスを許可する」では、 USB MassStorage クラスのデバイス(USB メモリなど)を許可します。

```
[armadillo ~]# abos-ctrl usb-filter allow-class MassStorage
```

図 10.114 指定した USB デバイスクラスを許可する

「図 10.114. 指定した USB デバイスクラスを許可する」の例では、引数として"MassStorage"を指定していますが、他にも指定できるデバイスクラスがあります。「図 10.115. 指定可能な USB デバイスクラスを確認する」に示すコマンドを実行することで、引数として指定できる文字列を確認できます。

```
[armadillo ~]# abos-ctrl usb-filter allow-class
supported USB device classes:
```

```
        Audio
        CDC
        HID
        Physical
        Image
        Printer
        MassStorage
        Hub
        CDCdata
        SmartCard
        ContentSecurity
        Video
        PersonalHealthCare
```

図 10.115 指定可能な USB デバイスクラスを確認する

各 USB デバイスクラスについては、 Defined Class Codes [https://www.usb.org/defined-class-codes] を参照してください。

1 つのデバイスで複数のデバイスクラスを持っている場合、そのデバイスの全てのデバイスクラスが許可されていなければ認識されません。例えば、「図 10.109. 接続されている USB デバイスをリストする」の 1 番目に認識されている USB カメラは Video と Audio の 2 つのデバイスクラスを持っています。このデバイスは、 Video と Audio 両方のデバイスクラスを許可している場合に認識されます。

10.25.6. 定義済みの USB デバイス許可ルールを表示する

「図 10.110. 指定した USB デバイスを許可する」や「図 10.113. 指定した USB デバイスを拒否する」、「図 10.114. 指定した USB デバイスクラスを許可する」を実行すると、 USB デバイスの許可ルールが更新されます。許可ルールに記載されているデバイスは、接続されたときに認識され、使用できます。

「図 10.116. 定義済みの USB デバイス許可ルールを表示する」に示すコマンドを実行することで現在の許可ルールの一覧を表示できます。

```
[armadillo ~]# abos-ctrl usb-filter list-rules
     1  class MassStorage
     2  device "046d" "08e5" "HD_Pro_Webcam_C920" ":0e0100:0e0200:010100:010200:"
"046d_HD_Pro_Webcam_C920"
```

図 10.116 定義済みの USB デバイス許可ルールを表示する

各行の最初の数値は、そのルールに割り当たった ID です。この ID は ルールを個別に削除する際に使用されます。詳細は「10.25.7. 定義済みの USB デバイス許可ルールを削除する」を参照してください。

2 列目の文字列は、そのルールがデバイスクラス単位の許可ルールであるか、デバイス固有の情報に基づいた許可ルールであるかを示します。
この列が class であれば、当該のルールはデバイスクラス単位の許可ルール(「10.25.5. USB デバイスクラス単位で USB デバイスの接続を許可する」によって追加されたルール)であり、3 列目の文字列は許可する USB デバイスクラス名です。
この列が device であれば、当該のルールはデバイス固有の情報に基づいた許可ルール(「10.25.3. USB デバイスの接続を許可する」によって追加されたルール)であり、以降の文字列はその条件に一致したデバイスが接続されると接続が許可されるルールです。この場合の 3 列目以降の文字列の意味を「表 10.16. 2 列目が device のときの許可ルールリストの各列の意味」に示します。

表 10.16 2 列目が device のときの許可ルールリストの各列の意味

3 列目	このルールによって許可されるベンダー ID です
4 列目	このルールによって許可されるモデル ID です
5 列目	このルールによって許可されるモデル名です
6 列目	このルールによって許可される USB クラスコードです。複数ある場合も全てが完全一致した場合のみ許可されます
7 列目	このルールによって許可されるシリアル番号です

1 つのルールにつき 1 行で表記され、接続したデバイスが全てのルールのうちどれか 1 つでも満たしていれば許可されます。

10.25.7. 定義済みの USB デバイス許可ルールを削除する

定義した許可ルールは削除することができます。削除することでそのデバイスは再接続時に接続が拒否され使用できなくなります。

「図 10.116. 定義済みの USB デバイス許可ルールを表示する」に示す状況のときに、"MassStorage" の許可を削除する例を「図 10.117. 定義済みの USB デバイス許可ルールを削除する」に示します。

```
[armadillo ~]# abos-ctrl usb-filter remove-rule 1
[armadillo ~]# abos-ctrl usb-filter list-rules
     1  device "046d" "08e5" "HD_Pro_Webcam_C920" ":0e0100:0e0200:010100:010200:"
"046d_HD_Pro_Webcam_C920"
```

図 10.117 定義済みの USB デバイス許可ルールを削除する

10.26. オプション品

本章では、Armadillo-900 開発セットのオプション品について説明します。

表 10.17 Armadillo-900 開発セット 関連のオプション品

名称	型番
AC アダプタ（12V/2.0A φ2.1mm）温度拡張品 効率レベル VI 品	OP-AC12V4-00

Armadillo-900 開発セット 製品マニュアル

改訂履歴

バージョン	年月日	改訂内容
1.0.0	2025/04/30	・初版発行
1.1.0	2025/05/28	・「5.5.8. GNSS を使用する」 内に 「5.5.8.2. ソフトウェア仕様」 を追加 ・「5.4. USB デバイスの接続を許可する」 を追加 ・「7.1.3. Armadillo に初期設定をインストールする」 内に、Armadillo をインターネットに接続できない環境についての記述を追加 ・「10.25. 不正な USB デバイスの接続を拒否する」内の、各種コマンドの使用方法を新しいバージョン向けに修正 ・「5.2.6. LTE」 内に SIM7672G のファームウェアに関する説明を追加 ・誤記などの軽微な修正

Armadillo-900 開発セット 製品マニュアル
Version 1.1.0
2025/05/28