Armadillo-IoT ゲートウェイ G4
製品マニュアル

AGX4500-C00D0

Version 1.4.0
2022/03/29

株式会社アットマークテクノ [https://www.atmark-techno.com]

Armadillo サイト [https://armadillo.atmark-techno.com]

Armadillo-IoT ゲートウェイ G4 製品マニュアル

株式会社アットマークテクノ

Version 1.4.0
2022/03/29

Armadillo-IoT ゲートウェイ G4 製品マニュアル

目次

図目次

Armadillo-IoT ゲートウェイ G4 製品マニュアル

表目次

1. はじめに

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

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

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

Armadillo Base OS とユーザーアプリケーションを含むコンテナはどちらも、Armadillo Base OS のリモートアップデート機能で安全にアップデートすることができます。Armadillo Base OS はアップデートの状態を 2 面化しているので電源やネットワークの遮断でよって中断してもアップデート前の状態に復旧します。

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

1.1. 本書で扱うこと扱わないこと

1.1.1. 扱うこと

本書では、Armadillo-IoT ゲートウェイ G4 の使い方、製品仕様(ソフトウェアおよびハードウェア)、オリジナルの製品を開発するために必要となる情報、その他注意事項について記載しています。Linux あるいは組み込み機器に不慣れな方でも読み進められるよう、コマンドの実行例なども記載しています。

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

1.1.2. 扱わないこと

本書では、一般的な Linux のプログラミング、デバッグ方法やツールの扱い方、各種モジュールの詳細仕様など、一般的な情報や、他に詳しい情報があるものは扱いません。また、(Armadillo-IoT ゲートウェイ G4 を使用した)最終製品あるいはサービスに固有な情報や知識も含まれていません。

1.2. 本書で必要となる知識と想定する読者

本書は、読者として Armadillo-IoT ゲートウェイ G4 を使ってオリジナルのゲートウェイ機器を開発するエンジニアを想定して書かれています。また、「Armadillo-IoT ゲートウェイ G4 を使うと、どのよ

うなことが実現可能なのか」を知りたいと考えている設計者・企画者も対象としています。Armadillo-IoT ゲートウェイ G4 は組込みプラットフォームとして実績のある Armadillo をベースとしているため、標準で有効になっている機能以外にも様々な機能を実現することができます。

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

1.3. ユーザー限定コンテンツ

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

- 各種信頼性試験データ・納入仕様書等製造関連情報

限定コンテンツを取得するには、「6. ユーザー登録」を参照してください。

1.4. 本書および関連ファイルのバージョンについて

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

Armadillo サイト - Armadillo-IoT ゲートウェイ G4 ドキュメントダウンロード

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

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

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

1.5. 本書の構成

本書には、Armadillo-IoT ゲートウェイ G4 をベースに、オリジナルの製品を開発するために必要となる情報を記載しています。また、取扱いに注意が必要な事柄についても説明しています。

- **はじめにお読みください。**

  「1. はじめに」、「2. 注意事項」

- **Armadillo-IoT ゲートウェイ G4 の仕様を紹介します。**

  「3. 製品概要」

- **工場出荷状態のソフトウェアの使い方や、動作を確認する方法を紹介します。**

「4. Armadillo の電源を入れる前に」、「5. 起動と終了」、「7. 動作確認方法」

- **工場出荷状態のソフトウェア仕様について紹介します。**

  「10. 動作ログ」、「13. ソフトウェア仕様」

- **システム開発に必要な情報を紹介します。**

  「8. 開発の基本的な流れ」、「9. Howto」、「11. セキュリティ」、「12. 製品機能」

- **拡張基板の開発や、ハードウェアをカスタマイズする場合に必要な情報を紹介します。**

  「14. ハードウェア仕様」

- **ご購入ユーザーに限定して公開している情報の紹介やユーザー登録について紹介します。**

  「6. ユーザー登録」

1.6. 表記について

1.6.1. フォント

本書では以下のような意味でフォントを使いわけています。

表 1.1 使用しているフォント

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

1.6.2. コマンド入力例

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

表 1.2 表示プロンプトと実行環境の関係

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

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

表 1.3 コマンド入力例での省略表記

表記	説明
[VERSION]	ファイルのバージョン番号

1.6.3. アイコン

本書では以下のようにアイコンを使用しています。

注意事項を記載します。

役に立つ情報を記載します。

用語の説明や補足的な説明を記載します。

1.7. 謝辞

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

2. 注意事項

2.1. 安全に関する注意事項

本製品を安全にご使用いただくために、特に以下の点にご注意ください。

- ご使用の前に必ず製品マニュアルおよび関連資料をお読みになり、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。
- 無線 LAN 機能を搭載した製品は、心臓ペースメーカーや補聴器などの医療機器、火災報知器や自動ドアなどの自動制御器、電子レンジ、高度な電子機器やテレビ・ラジオに近接する場所、移動体識別用の構

内無線局および特定小電力無線局の近くで使用しないでください。製品が発生する電波によりこれらの機器の誤作動を招く恐れがあります。

2.2. 取扱い上の注意事項

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

破損しやすい箇所	LVDS、MIPI-CSI、HDMI、USB コンソールのコネクタは、破損しやすい部品になっています。無理に力を加えて破損することのないよう十分注意してください。
本製品の改造	本製品に改造[1]を行った場合は保証対象外となりますので十分ご注意ください。また、改造やコネクタ等の増設[2]を行う場合は、作業前に必ず動作確認を行ってください。
電源投入時のコネクタ着脱	本製品や周辺回路に電源が入っている状態で、活線挿抜対応インターフェース(LAN、USB、SD、HDMI) [3]以外へのコネクタ着脱は、絶対に行わないでください。
静電気	本製品には CMOS デバイスを使用しており、静電気により破壊されるおそれがあります。本製品を開封するときは、低湿度状態にならないよう注意し、静電防止用マットの使用、導電靴や人体アースなどによる作業者の帯電防止対策、備品の放電対策、静電気対策を施された環境下で行ってください。また、本製品を保管する際は、静電気を帯びやすいビニール袋やプラスチック容器などは避け、導電袋や導電性の容器・ラックなどに収納してください。
ラッチアップ	電源および入出力からの過大なノイズやサージ、電源電圧の急激な変動等により、使用している CMOS デバイスがラッチアップを起こす可能性があります。いったんラッチアップ状態となると、電源を切断しないかぎりこの状態が維持されるため、デバイスの破損につながることがあります。ノイズの影響を受けやすい入出力ラインには、保護回路を入れることや、ノイズ源となる装置と共通の電源を使用しない等の対策をとることをお勧めします。
衝撃	落下や衝撃などの強い振動を与えないでください。
使用場所の制限	無線機能を搭載した製品は、テレビ・ラジオに近接する場所で使用すると、受信障害を招く恐れがあります。
振動	振動が発生する環境では、Armadillo が動かないよう固定して使用してください。

2.3. 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

[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

お使いください。アットマークテクノは、当該ソフトウェアが特定の目的に適合すること、ソフトウェアの信頼性および正確性、ソフトウェアを含む本製品の使用による結果について、お客様に対し何らの保証も行いません。

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

以下のソフトウェアは、オープンソースソフトウェアではありません。

ボード情報取得ツール(get-board-info)

2.4. 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、家庭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

2.5. 保証について

本製品の本体基板は、製品に添付もしくは弊社 Web サイトに記載している「製品保証規定」に従い、ご購入から 1 年間の交換保証を行っています。添付品およびソフトウェアは保証対象外となりますのでご注意ください。

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

2.6. 輸出について

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

2.7. 商標について

- Armadillo は株式会社アットマークテクノの登録商標です。その他の記載の商品名および会社名は、各社・各団体の商標または登録商標です。™、®マークは省略しています。
- SD、SDHC、SDXC、microSD、microSDHC、microSDXC、SDIO ロゴは SD-3C, LLC の商標です。

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

3. 製品概要

3.1. 製品の特長

3.1.1. Armadillo とは

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

- Arm プロセッサ搭載・省電力設計

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

- 小型・手のひらサイズ

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

- 標準 OS として Linux をプリインストール

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

- 開発環境

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

3.1.2. Armadillo-IoT ゲートウェイ G4 とは

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

図 3.1 Armadillo-IoT ゲートウェイ G4 とは

Armadillo-IoT ゲートウェイ G4 には以下の特長があります。

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

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

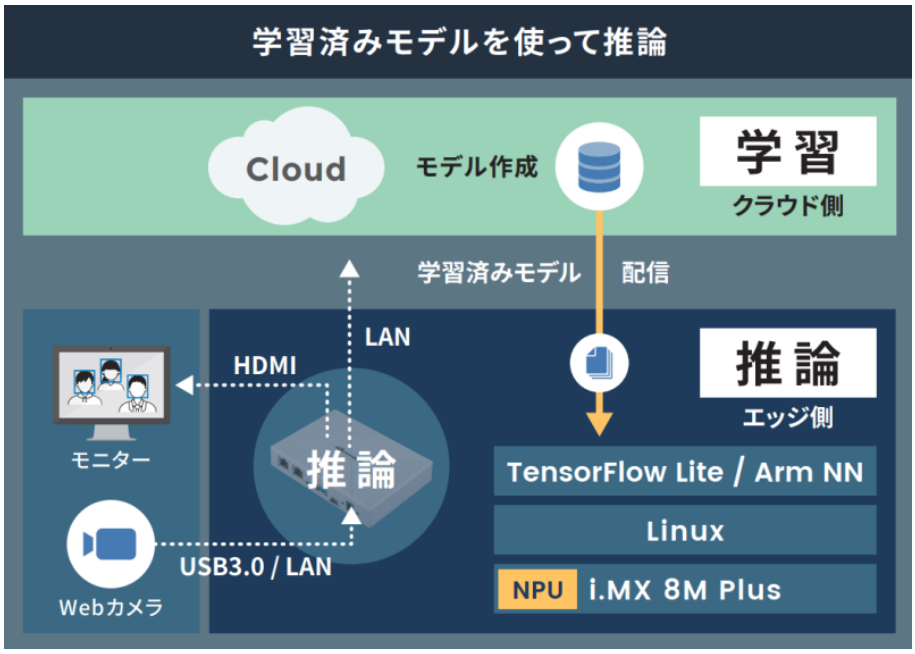


図 3.2 エッジ AI 処理、機械学習の例

- i.MX 8M Plus 搭載・動画を高速処理

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

- 動作温度範囲-20～+70℃の産業設計

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

- Armadillo Base OS 搭載

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

- セキュアエレメント搭載

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

3.1.3. Armadillo Base OS とは

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

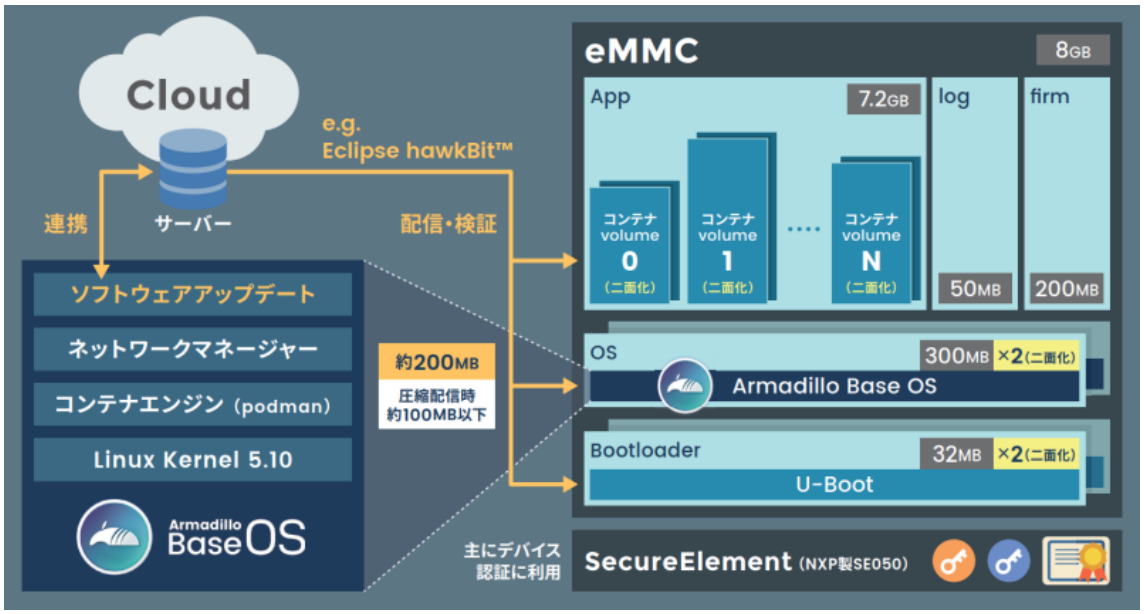


図 3.3 Armadillo Base OS とは

- OS のコンパクト化

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

- コンテナによるアプリケーション運用

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

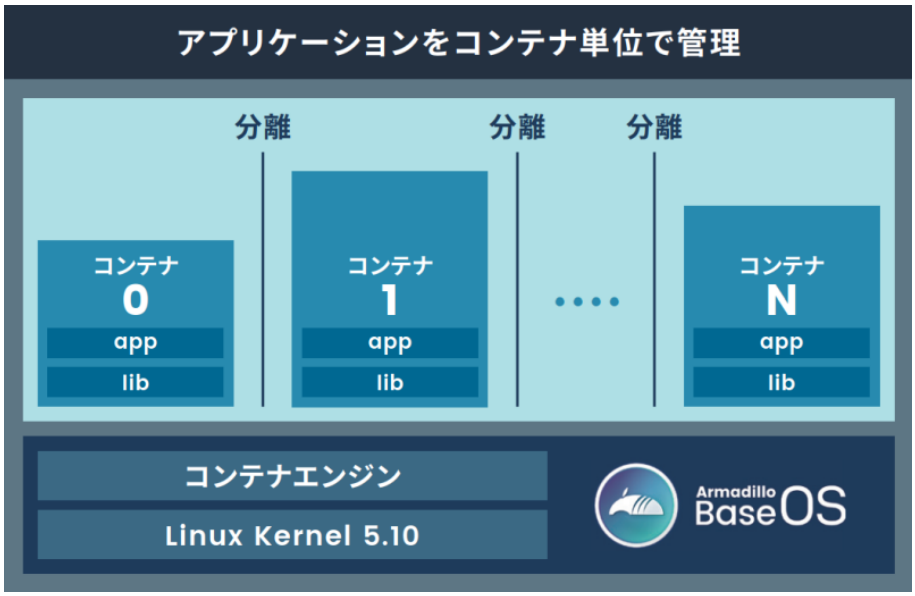


図 3.4 コンテナによるアプリケーションの運用

- アップデート機能を標準搭載

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

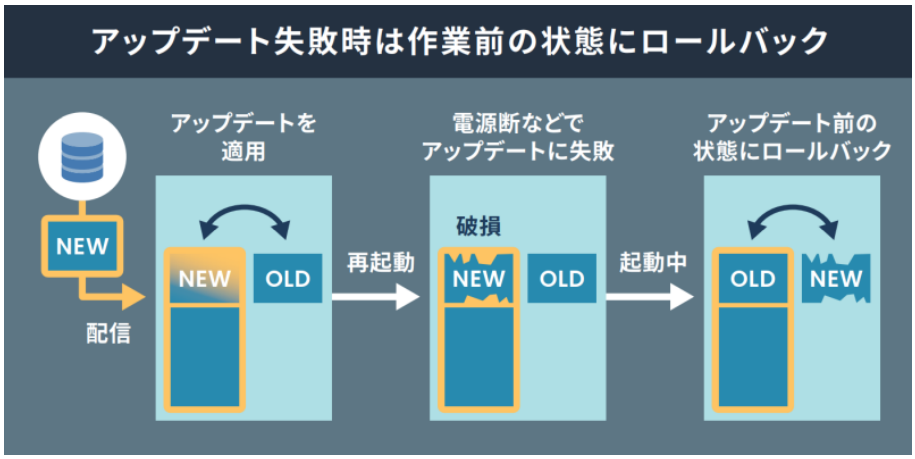


図 3.5 ロールバックの仕組み

- 堅牢性の向上

  安定性の高いファイルシステムで、ストレージへの書込みを減らして消耗を抑制するなど、高い堅牢性を有します。運用ログの記録機能も標準搭載しています。

- セキュリティ機能の向上

  コンテナにアクセス権限を設けて管理することができます。デバイス証明に利用できるセキュアエレメントを搭載するほか、セキュア環境「OP-TEE」を利用可能な状態で提供しています。

3.2. 製品ラインアップ

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

表 3.1 Armadillo-IoT ゲートウェイ G4 ラインアップ

名称	型番
Armadillo-IoT ゲートウェイ G4 LAN モデル開発セット(メモリ 2GB、ストレージ 10GB、WLAN コンボ非搭載)	AGX4500-C00D0
Armadillo-IoT ゲートウェイ G4 LAN モデル量産用(メモリ 2GB、ストレージ 10GB、WLAN コンボ非搭載)	AGX4500-C00Z
Armadillo-IoT ゲートウェイ G4 LAN モデル量産ボード(メモリ 2GB、ストレージ 10GB、WLAN コンボ非搭載)	AGX4500-U00Z

3.2.1. Armadillo-IoT ゲートウェイ G4 開発セット

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

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

- Armadillo-IoT ゲートウェイ G4

・ オプションケース(金属製)

・ USB(A オス-microB)ケーブル

・ AC アダプタ(12V/3.0A)

・ ジャンパソケット

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

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

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

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

3.3. 仕様

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

表 3.2 仕様

CPU	NXP Semiconductors i.MX 8M Plus
	Arm Cortex-A53 × 4 ・命令/データキャッシュ 32KByte/32KByte ・L2 キャッシュ 512KByte ・メディアプロセッシングエンジン(NEON)搭載 ・Thumb code(16bit 命令セット)サポート Arm Cortex-M7 × 1 ・命令/データキャッシュ 32KByte/32KByte ・TCM 256kByte
システムクロック	CPU コアクロック(Arm Cortex-A53): 1.6GHz CPU コアクロック(Arm Cortex-M7): 800MHz DDR クロック: 2GHz 源発振クロック: 32.768kHz、24MHz
NPU	2.3 TOPS
RAM	LPDDR4: 2GByte バス幅: 32bit
ROM	eMMC: 9.8GiB [a] HS400(最大転送速度: 400MB/s)
LAN(Ethernet)	1000BASE-T × 2 AUTO-MDIX 対応
USB	USB 3.0 Host × 1 (Type-A)
SD	microSD スロット × 1 UHS-I
ビデオ	HDMI 出力 × 1 (micro Type-D) LVDS 出力 (4 レーン) × 1 [b]
オーディオ	HDMI 出力 × 1 (micro Type-D)
カメラ	MIPI CSI-2 (2 レーン) × 1 [b]
拡張インターフェース [b] [c]	USB 2.0 × 1、GPIO × 34、SPI × 2、UART × 2、 PDM MIC × 4、I2S × 1、CAN x 2、I2C × 3、PWM × 4
カレンダー時計	リアルタイムクロック [d]
スイッチ	ユーザースイッチ × 1

LED	ユーザー LED × 1、電源 LED × 1 [b]
メンテナンスポート	USB micro-B シリアルコンソール
セキュアエレメント	NXP Semiconductors SE050
電源電圧	DC 12V±10%
消費電力(参考値)	2.5W(定常状態) [e]
動作温度範囲	-20〜+70℃(結露なきこと)
外形サイズ(基板)	135×95mm(突起部を除く)
外形サイズ(ケース)	143×100.5×26mm(突起部を除く)

[a]pSLC モードで動作します。

[b]ケース装着時はケース外部から利用できません。

[c]拡張インターフェース(CON11、CON12)の信号線において、i.MX 8M Plus のピンマルチプクレスの設定で、優先的に機能を割り当てた場合に拡張可能な最大数を記載しています。

[d]コイン電池によるバックアップが可能です。電池は付属していません。

[e]外部接続機器の消費分は含みません。

3.4. ブロック図

Armadillo-IoT ゲートウェイ G4 のブロック図は次のとおりです。

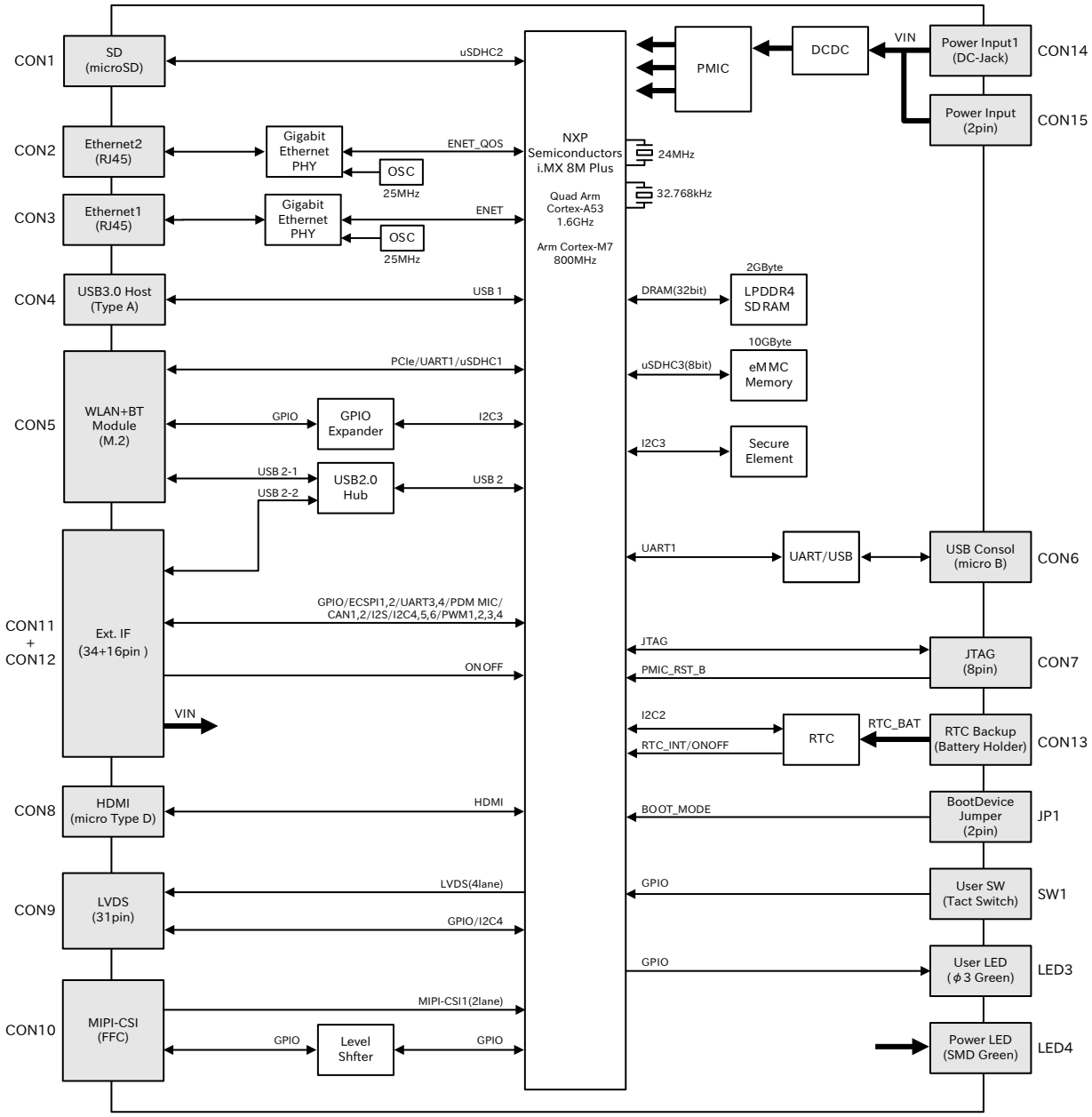


図 3.6 ブロック図

3.5. ストレージデバイスのパーティション構成

Armadillo-IoT ゲートウェイ G4 の eMMC のパーティション構成を「表 3.3. eMMC メモリマップ」に示します。

表 3.3 eMMC メモリマップ

パーティション	サイズ	ラベル	説明
1	300MiB	rootfs_0	A/B アップデートの A 面パーティション(Linux カーネルイメージ, Device Tree Blob, Alpine Linux rootfs を含む)

パーティション	サイズ	ラベル	説明
2	300MiB	rootfs_1	A/B アップデートの B 面パーティション(Linux カーネルイメージ, Device Tree Blob, Alpine Linux rootfs を含む)
3	50MiB	logs	ログ書き込み用パーティション
4	200MiB	firm	ファームウェア用パーティション
5	8.95GiB	app	アプリケーション用パーティション

Armadillo-IoT ゲートウェイ G4 の eMMC のブートパーティションの構成を「表 3.4. eMMC ブートパーティション構成」に示します。

表 3.4 eMMC ブートパーティション構成

ディスクデバイス	サイズ	説明
/dev/mmcblk0boot0	31.5 MiB	A/B アップデートの A 面
/dev/mmcblk0boot1	31.5 MiB	A/B アップデートの B 面

Armadillo-IoT ゲートウェイ G4 の eMMC の GPP(General Purpose Partition)の構成を「表 3.5. eMMC GPP 構成」に示します。

表 3.5 eMMC GPP 構成

ディスクデバイス	サイズ	説明
/dev/mmcblk0gp0	8 MiB	ライセンス情報等の保存
/dev/mmcblk0gp1	8 MiB	予約領域
/dev/mmcblk0gp2	8 MiB	予約領域
/dev/mmcblk0gp3	8 MiB	ユーザー領域

4. Armadillo の電源を入れる前に

4.1. 準備するもの

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

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

4.2. 開発/動作確認環境の構築

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

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

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

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

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

ATDE9 は Debian GNU/Linux 11 (コードネーム bullseye) をベースに、Armadillo-IoT ゲートウェイ G4 のソフトウェア開発を行うために必要なクロス開発ツールや、Armadillo-IoT ゲートウェイ G4 の動作確認を行うために必要なツールが事前にインストールされています。

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

4.2.1. ATDE のセットアップ

4.2.1.1. VMware のインストール

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

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

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

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

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

4.2.1.2. ATDE のアーカイブを取得

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

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

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

4.2.1.3. ATDE のアーカイブを展開

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

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

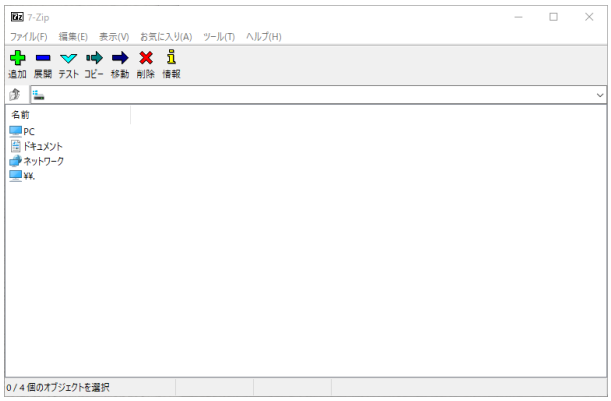
4.2.1.4. Windows で ATDE のアーカイブ展開する

1. 7-Zip のインストール

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

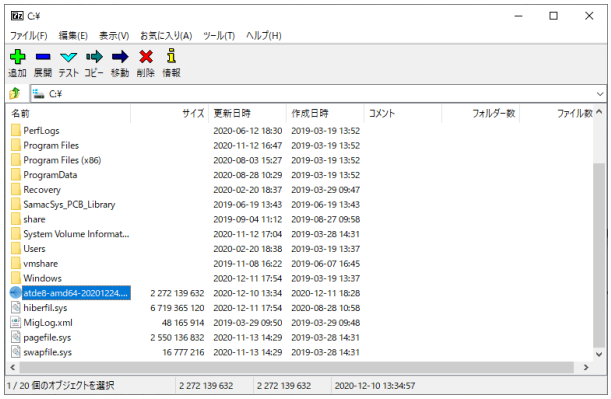
2. 7-Zip の起動

   7-Zip を起動します。



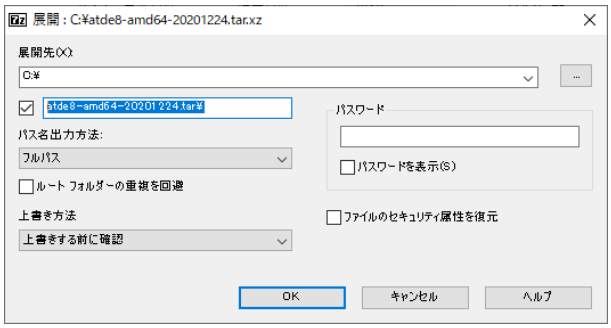
3. xz 圧縮ファイルの選択

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



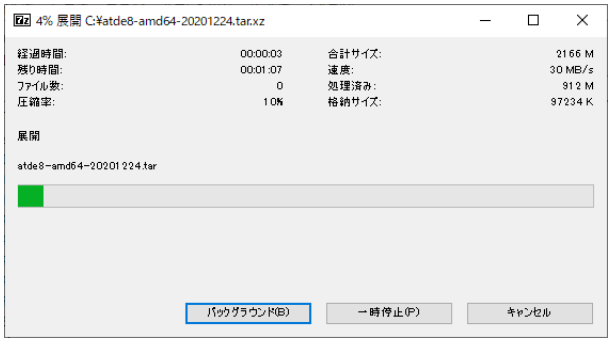
4. xz 圧縮ファイルの展開先の指定

   「展開先」を指定して、「OK」をクリックします。



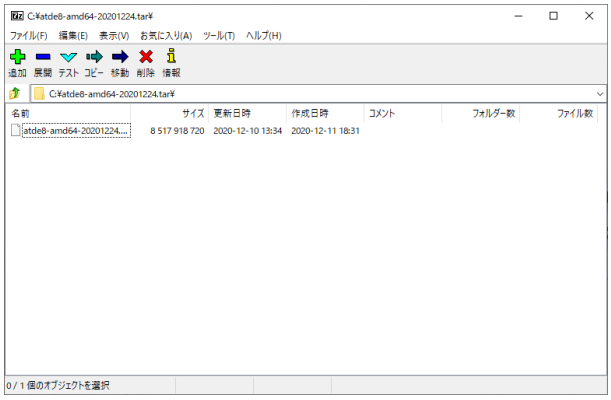
5. xz 圧縮ファイルの展開

   展開が始まります。



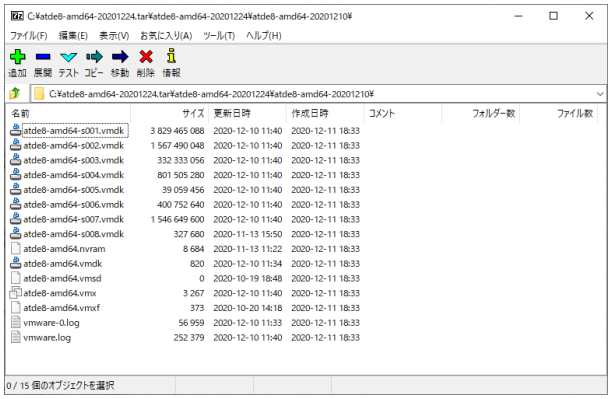
6. tar アーカイブファイルの選択

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



7. 展開の完了確認

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



4.2.1.5. Linux で tar.xz 形式のファイルを展開する

1. tar.xz 圧縮ファイルの展開

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

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

2. 展開の完了確認

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

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

4.2.1.6. ATDE の起動

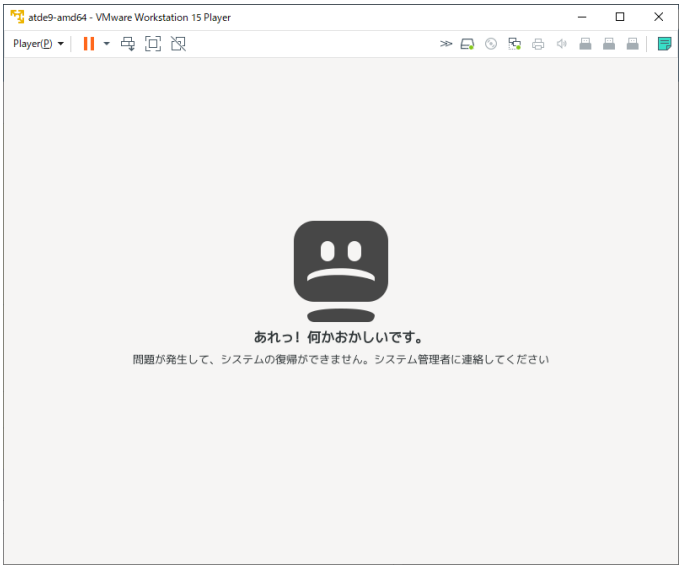
ATDE のアーカイブを展開したディレクトリに存在する仮想マシン構成(.vmx)ファイルを VMware 上で開くと、ATDE を起動することができます。ATDE9 にログイン可能なユーザーを、「表 4.1. ユーザー名とパスワード」に示します [2]。

表 4.1 ユーザー名とパスワード

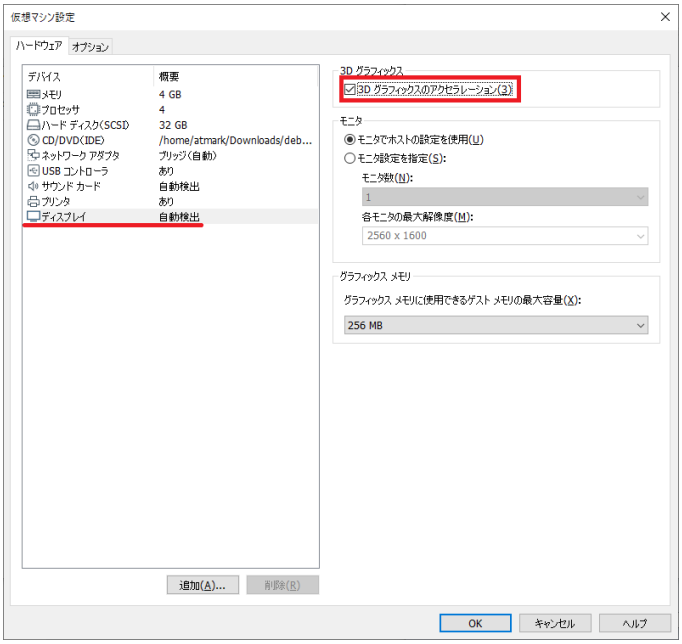
ユーザー名	パスワード	権限
atmark	atmark	一般ユーザー
root	root	特権ユーザー

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

[2]特権ユーザーで GUI ログインを行うことはできません



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



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

4.2.2. 取り外し可能デバイスの使用

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

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

Armadillo-IoT ゲートウェイ G4 の動作確認を行うためには、「表 4.2. 動作確認に使用する取り外し可能デバイス」に示すデバイスをゲスト OS に接続する必要があります。

表 4.2 動作確認に使用する取り外し可能デバイス

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

4.2.3. コマンドライン端末(GNOME 端末)の起動

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

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

図 4.1 GNOME 端末の起動

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

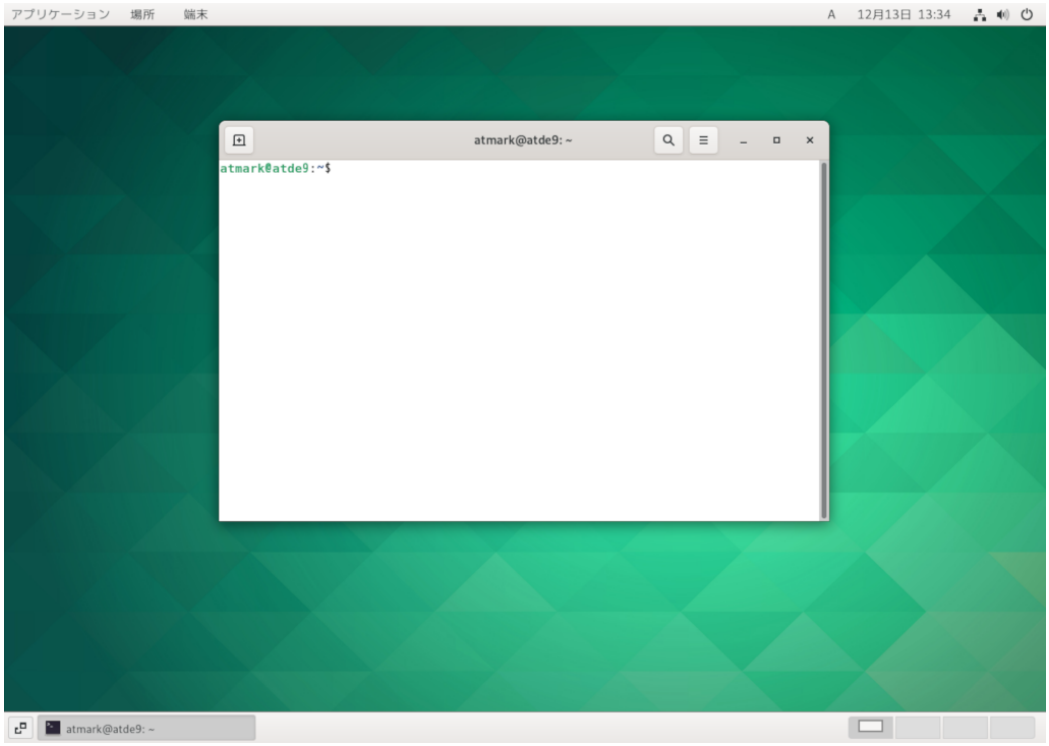


図 4.2 GNOME 端末のウィンドウ

4.2.4. シリアル通信ソフトウェア(minicom)の使用

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

表 4.3 シリアル通信設定

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

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

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

図 4.3 minicom の設定の起動

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

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

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

図 4.4 minicom の設定

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

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

図 4.5 minicom のシリアルポートの設定

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

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

Linux でシリアル通信用 USB ケーブル(A-microB)を接続した場合、コンソールに以下のようなログが表示されます。ログが表示されなくても、dmesg コマンドを実行することで、ログを確認することができます。

```
usb 2-2.1: new full-speed USB device number 4 using uhci_hcd
usb 2-2.1: New USB device found, idVendor=10c4, idProduct=ea60,
bcdDevice= 1.00
usb 2-2.1: New USB device strings: Mfr=1, Product=2,
SerialNumber=3
usb 2-2.1: Product: CP2102N USB to UART Bridge Controller
usb 2-2.1: Manufacturer: Silicon Labs
usb 2-2.1: SerialNumber: 6a9681f80272eb11abb4496e014bf449
usbcore: registered new interface driver usbserial_generic
usbserial: USB Serial support registered for generic
usbcore: registered new interface driver cp210x
```

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

図 4.6 例. シリアル通信用 USB ケーブル(A-microB)接続時のログ

上記のログからシリアル通信用 USB ケーブル(A-microB)が ttyUSB0 に割り当てられたことが分かります。

5. F キーを押して Hardware Flow Control を No に設定してください。
6. G キーを押して Software Flow Control を No に設定してください。
7. キーボードの E キーを押してください。「図 4.7. minicom のシリアルポートのパラメータの設定」が表示されます。

```
+---------[Comm Parameters]----------+
|                                    |
|     Current: 115200 8N1            |
| Speed            Parity      Data  |
| A: <next>        L: None     S: 5  |
| B: <prev>        M: Even     T: 6  |
| C:   9600        N: Odd      U: 7  |
| D:  38400        O: Mark     V: 8  |
| E: 115200        P: Space          |
|                                    |
| Stopbits                           |
| W: 1             Q: 8-N-1          |
| X: 2             R: 7-E-1          |
|                                    |
|                                    |
| Choice, or <Enter> to exit?        |
+------------------------------------+
```

図 4.7 minicom のシリアルポートのパラメータの設定

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

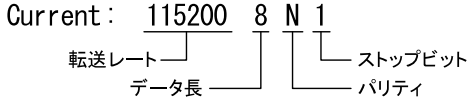


図 4.8 minicom シリアルポートの設定値

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

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

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

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

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

図 4.9 minicom 起動方法

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

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

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

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

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

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

図 4.10 minicom 終了確認

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

4.3. インターフェースレイアウト

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

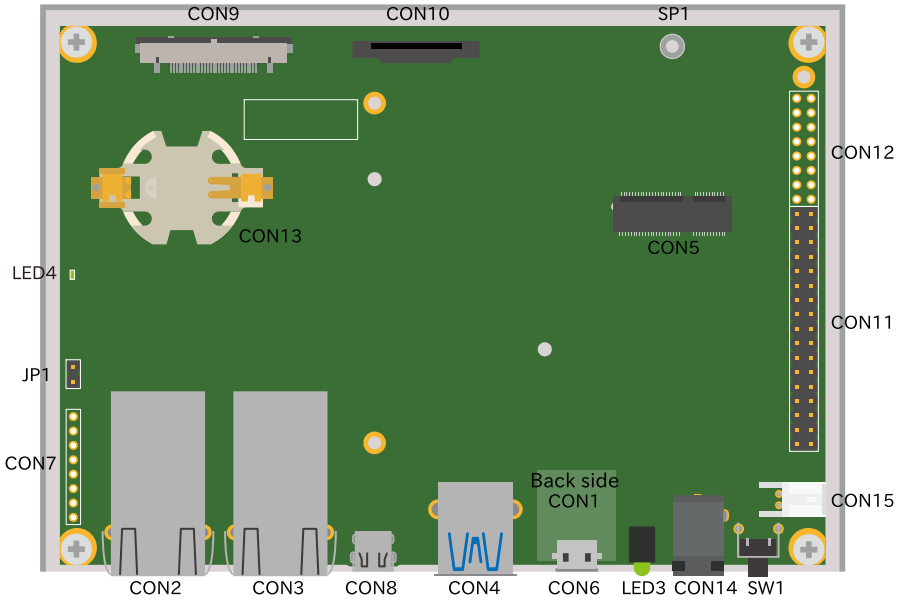


図 4.11 インターフェースレイアウト (ケース内部)

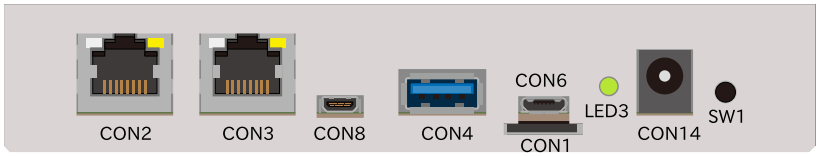


図 4.12 インターフェースレイアウト (ケース正面)

表 4.4 インターフェース内容

部品番号	インターフェース名	形状	備考
CON1	SD インターフェース	microSD スロット	-
CON2	LAN インターフェース 2	RJ-45 コネクタ	-
CON3	LAN インターフェース 1	RJ-45 コネクタ	-
CON4	USB インターフェース	USB 3.0 Type-A コネクタ	-
CON5	M.2 インターフェース	M.2 コネクタ(E-Key)	-
CON6	USB コンソールインターフェース	USB micro B コネクタ	-
CON7	JTAG インターフェース	ピンヘッダ 8 ピン(2.54mm ピッチ)	-
CON8	HDMI インターフェース	HDMI Type-D コネクタ	-
CON9	LVDS インターフェース	LVDS コネクタ 31 ピン	挿抜寿命: 50 回[a]
CON10	MIPI-CSI インターフェース	FFC コネクタ 15 ピン(1mm ピッチ)	挿抜寿命: 30 回[a]
CON11	拡張インターフェース 1	ピンヘッダ 34 ピン(2.54mm ピッチ)	-
CON12	拡張インターフェース 2	ピンヘッダ 16 ピン(2.54mm ピッチ)	-
CON13	RTC バックアップインターフェース	電池ボックス	対応電池: CR2032 等
CON14	電源入力インターフェース 1	DC ジャック	対応プラグ: 内径 2.1mm 外形 5.5mm
CON15	電源入力インターフェース 2	ピンヘッダ 2 ピン(2mm ピッチ)	-
JP1	起動デバイス設定ジャンパ	ピンヘッダ 2 ピン(2.54mm ピッチ)	-
SW1	ユーザースイッチ	タクトスイッチ	-
LED3	ユーザー LED	LED(緑色、φ3mm)	-
LED4	電源 LED	LED(緑色、面実装)	-
SP1	M.2 用スタッド	スペーサー(M2、L=2.45mm)	-

部品番号	インターフェース名	形状	備考
SW1	ユーザースイッチ	タクトスイッチ	-

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

4.4. 接続方法

Armadillo-IoT ゲートウェイ G4 と周辺装置の接続例を次に示します。

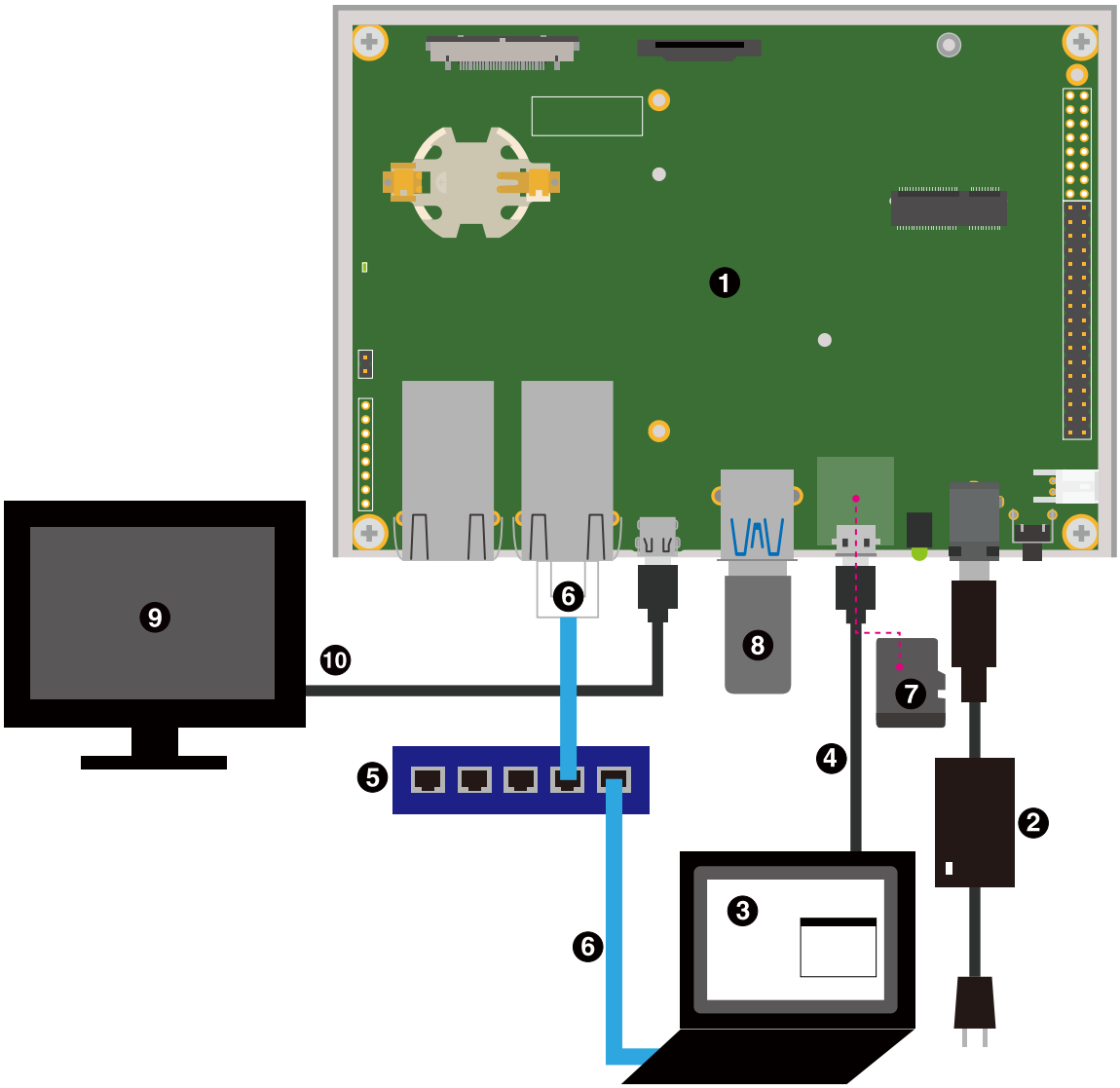


図 4.13 Armadillo-IoT ゲートウェイ G4 の接続例

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

❺ LAN HUB

❻ Ethernet ケーブル

❼ microSD カード

❽ USB メモリ

❾ ディスプレイ(HDMI 対応)

❿ HDMI ケーブル

4.5. ジャンパピンの設定について

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

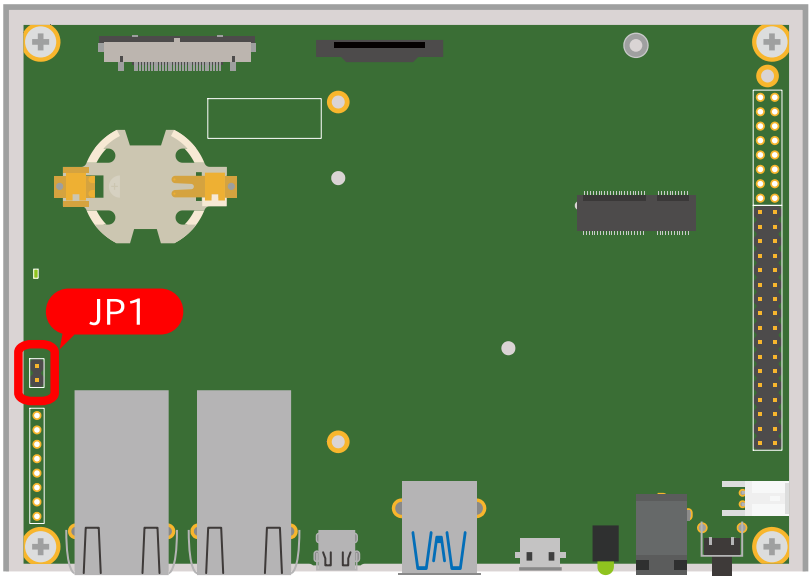


図 4.14 JP1 の位置

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

ジャンパのオープン、ショートとは

「オープン」とはジャンパピンにジャンパソケットを接続していない状態です。

「ショート」とはジャンパピンにジャンパソケットを接続している状態です。

4.6. vi エディタの使用方法

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

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

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

4.6.1. vi の起動

vi を起動するには、以下のコマンドを入力します。

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

図 4.15 vi の起動

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

4.6.2. 文字の入力

文字を入力するにはコマンドモードから入力モードへ移行する必要があります。コマンドモードから入力モードに移行するには、「表 4.5. 入力モードに移行するコマンド」に示すコマンドを入力します。入力モードへ移行後は、キーを入力すればそのまま文字が入力されます。

表 4.5 入力モードに移行するコマンド

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

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

日本語変換機能を OFF に

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

「i」、「a」それぞれのコマンドを入力した場合の文字入力の開始位置を「図 4.16. 入力モードに移行するコマンドの説明」に示します。

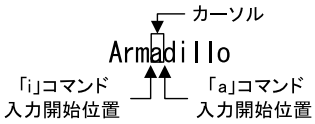


図 4.16 入力モードに移行するコマンドの説明

vi での文字削除

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

4.6.3. カーソルの移動

方向キーでカーソルの移動ができますが、コマンドモードで「表 4.6. カーソルの移動コマンド」に示すコマンドを入力することでもカーソルを移動することができます。

表 4.6 カーソルの移動コマンド

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

4.6.4. 文字の削除

文字を削除する場合は、コマンドモードで「表 4.7. 文字の削除コマンド」に示すコマンドを入力します。

表 4.7 文字の削除コマンド

コマンド	動作
x	カーソル上の文字を削除
dd	現在行を削除

「x」コマンド、「dd」コマンドを入力した場合に削除される文字を「図 4.17. 文字を削除するコマンドの説明」に示します。

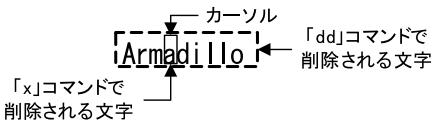


図 4.17 文字を削除するコマンドの説明

4.6.5. 保存と終了

ファイルの保存、終了を行うコマンドを「表 4.8. 保存・終了コマンド」に示します。

表 4.8 保存・終了コマンド

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

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

5. 起動と終了

5.1. 起動

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

Armadillo-IoT ゲートウェイ G4 の電源投入時点でのジャンパ JP1 の状態によって起動モードが変化します。 詳しくは 「14.2.14. JP1(起動デバイス設定ジャンパ)」 を参照してください。

以下に起動ログの例を示します。

```
U-Boot SPL 2020.04-at1 (Dec 02 2021 - 01:42:08 +0000)
DDRINFO: start DRAM init
DDRINFO: DRAM rate 4000MTS
DDRINFO:ddrphy calibration done
DDRINFO: ddrmix config done
Normal Boot
Trying to boot from BOOTROM
image offset 0x0, pagesize 0x200, ivt offset 0x0
NOTICE:  BL31: v2.4(release):lf-5.10.y-1.0.0-0-gba76d337e
NOTICE:  BL31: Built : 08:25:30, Jun  9 2021


U-Boot 2020.04-at1 (Dec 02 2021 - 01:42:08 +0000)

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

 BuildInfo:
  - ATF ba76d33
  - U-Boot 2020.04-at1

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

```
Normal Boot
Hit any key to stop autoboot:  0
switch to partitions #0, OK
mmc2(part 0) is current device
31051784 bytes read in 675 ms (43.9 MiB/s)
Booting from mmc ...

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

Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0000000000 [0x410fd034]
[    0.000000] Linux version 5.10.52-1-at (builder@98b643f88a21) (gcc (Alpine1
[    0.000000] Machine model: Atmark-Techno Armadillo-IoT Gateway G4 Board
[    0.000000] Reserved memory: created CMA memory pool at 0x0000000062000000B
[    0.000000] OF: reserved mem: initialized node linux,cma, compatible id shl
[    0.000000] Reserved memory: created DMA memory pool at 0x0000000094300000B
[    0.000000] OF: reserved mem: initialized node vdev0buffer@94300000, compal
[    0.000000] Zone ranges:
[    0.000000]   DMA      [mem 0x0000000040000000-0x00000000bfffffff]
[    0.000000]   DMA32    empty
[    0.000000]   Normal   empty
[    0.000000] Movable zone start for each node
[    0.000000] Early memory node ranges
[    0.000000]   node   0: [mem 0x0000000040000000-0x0000000055ffffff]
[    0.000000]   node   0: [mem 0x0000000058000000-0x00000000923fffff]
[    0.000000]   node   0: [mem 0x0000000092400000-0x00000000a43fffff]
[    0.000000]   node   0: [mem 0x00000000a4400000-0x00000000bfffffff]
[    0.000000] Initmem setup node 0 [mem 0x0000000040000000-0x00000000bfffffff]
[    0.000000] psci: probing for conduit method from DT.
[    0.000000] psci: PSCIv1.1 detected in firmware.
[    0.000000] psci: Using standard PSCI v0.2 function IDs
[    0.000000] psci: Trusted OS migration not required
[    0.000000] psci: SMC Calling Convention v1.2
[    0.000000] percpu: Embedded 22 pages/cpu s51728 r8192 d30192 u90112
[    0.000000] Detected VIPT I-cache on CPU0
[    0.000000] CPU features: detected: ARM erratum 845719
[    0.000000] CPU features: detected: GIC system register CPU interface
[    0.000000] Built 1 zonelists, mobility grouping on.  Total pages: 507904
[    0.000000] Kernel command line: console=ttymxc1,115200 root=/dev/mmcblk2po
[    0.000000] Dentry cache hash table entries: 262144 (order: 9, 2097152 byt)
[    0.000000] Inode-cache hash table entries: 131072 (order: 8, 1048576 byte)
[    0.000000] mem auto-init: stack:off, heap alloc:off, heap free:off
[    0.000000] Memory: 912664K/2064384K available (19326K kernel code, 1588K )
[    0.000000] random: get_random_u64 called from __kmem_cache_create+0x28/0x0
[    0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=4, Nodes=1
[    0.000000] rcu: Preemptible hierarchical RCU implementation.
[    0.000000] rcu:     RCU event tracing is enabled.
[    0.000000] rcu:     RCU restricting CPUs from NR_CPUS=256 to nr_cpu_ids=4.
[    0.000000]  Trampoline variant of Tasks RCU enabled.
[    0.000000] rcu: RCU calculated value of scheduler-enlistment delay is 25 .
[    0.000000] rcu: Adjusting geometry for rcu_fanout_leaf=16, nr_cpu_ids=4
[    0.000000] NR_IRQS: 64, nr_irqs: 64, preallocated irqs: 0
```

```
[    0.000000] GICv3: GIC: Using split EOI/Deactivate mode
[    0.000000] GICv3: 160 SPIs implemented
[    0.000000] GICv3: 0 Extended SPIs implemented
[    0.000000] GICv3: Distributor has no Range Selector support
[    0.000000] GICv3: 16 PPIs implemented
[    0.000000] GICv3: CPU0: found redistributor 0 region 0:0x0000000038880000
[    0.000000] ITS: No ITS available, not enabling LPIs
[    0.000000] arch_timer: cp15 timer(s) running at 8.00MHz (phys).
[    0.000000] clocksource: arch_sys_counter: mask: 0xffffffffffffff max_cycls
[    0.000003] sched_clock: 56 bits at 8MHz, resolution 125ns, wraps every 21s
[    0.000351] Console: colour dummy device 80x25
[    0.000382] Calibrating delay loop (skipped), value calculated using timer)
[    0.000397] pid_max: default: 32768 minimum: 301
[    0.000506] LSM: Security Framework initializing
[    0.000571] Mount-cache hash table entries: 4096 (order: 3, 32768 bytes, l)
[    0.000583] Mountpoint-cache hash table entries: 4096 (order: 3, 32768 byt)
[    0.001724] rcu: Hierarchical SRCU implementation.
[    0.002984] smp: Bringing up secondary CPUs ...
[    0.003331] Detected VIPT I-cache on CPU1
[    0.003356] GICv3: CPU1: found redistributor 1 region 0:0x00000000388a0000
[    0.003389] CPU1: Booted secondary processor 0x0000000001 [0x410fd034]
[    0.003778] Detected VIPT I-cache on CPU2
[    0.003797] GICv3: CPU2: found redistributor 2 region 0:0x00000000388c0000
[    0.003817] CPU2: Booted secondary processor 0x0000000002 [0x410fd034]
[    0.004235] Detected VIPT I-cache on CPU3
[    0.004253] GICv3: CPU3: found redistributor 3 region 0:0x00000000388e0000
[    0.004272] CPU3: Booted secondary processor 0x0000000003 [0x410fd034]
[    0.004328] smp: Brought up 1 node, 4 CPUs
[    0.004342] SMP: Total of 4 processors activated.
[    0.004349] CPU features: detected: 32-bit EL0 Support
[    0.004356] CPU features: detected: CRC32 instructions
[    0.004601] CPU: All CPU(s) started at EL2
[    0.004620] alternatives: patching kernel code
[    0.005296] devtmpfs: initialized
[    0.013072] KASLR disabled due to lack of seed
[    0.013221] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff,s
[    0.013234] futex hash table entries: 1024 (order: 4, 65536 bytes, linear)
[    0.031459] pinctrl core: initialized pinctrl subsystem
[    0.032122] NET: Registered protocol family 16
[    0.038565] DMA: preallocated 256 KiB GFP_KERNEL pool for atomic allocatios
[    0.039388] DMA: preallocated 256 KiB GFP_KERNEL|GFP_DMA pool for atomic as
[    0.040199] DMA: preallocated 256 KiB GFP_KERNEL|GFP_DMA32 pool for atomics
[    0.040256] audit: initializing netlink subsys (disabled)
[    0.040448] audit: type=2000 audit(0.040:1): state=initialized audit_enabl1
[    0.040844] thermal_sys: Registered thermal governor 'step_wise'
[    0.041137] cpuidle: using governor ladder
[    0.041157] cpuidle: using governor menu
[    0.041399] hw-breakpoint: found 6 breakpoint and 4 watchpoint registers.
[    0.041471] ASID allocator initialised with 65536 entries
[    0.042530] Serial: AMBA PL011 UART driver
[    0.042588] imx mu driver is registered.
[    0.042611] imx rpmsg driver is registered.
[    0.077103] imx8mp-pinctrl 30330000.pinctrl: initialized IMX pinctrl driver
[    0.094048] HugeTLB registered 1.00 GiB page size, pre-allocated 0 pages
[    0.094063] HugeTLB registered 32.0 MiB page size, pre-allocated 0 pages
[    0.094069] HugeTLB registered 2.00 MiB page size, pre-allocated 0 pages
[    0.094077] HugeTLB registered 64.0 KiB page size, pre-allocated 0 pages
[    0.095599] cryptd: max_cpu_qlen set to 1000
```

```
[    0.164236] raid6: neonx8   gen()  2150 MB/s
[    0.232284] raid6: neonx8   xor()  1604 MB/s
[    0.300362] raid6: neonx4   gen()  2200 MB/s
[    0.368398] raid6: neonx4   xor()  1567 MB/s
[    0.436455] raid6: neonx2   gen()  2098 MB/s
[    0.504512] raid6: neonx2   xor()  1443 MB/s
[    0.572561] raid6: neonx1   gen()  1798 MB/s
[    0.640615] raid6: neonx1   xor()  1222 MB/s
[    0.708677] raid6: int64x8  gen()  1438 MB/s
[    0.776717] raid6: int64x8  xor()   771 MB/s
[    0.844764] raid6: int64x4  gen()  1601 MB/s
[    0.912817] raid6: int64x4  xor()   822 MB/s
[    0.980863] raid6: int64x2  gen()  1399 MB/s
[    1.048914] raid6: int64x2  xor()   748 MB/s
[    1.116954] raid6: int64x1  gen()  1033 MB/s
[    1.185017] raid6: int64x1  xor()   517 MB/s
[    1.185024] raid6: using algorithm neonx4 gen() 2200 MB/s
[    1.185031] raid6: .... xor() 1567 MB/s, rmw enabled
[    1.185038] raid6: using neon recovery algorithm
[    1.187393] iommu: Default domain type: Translated
[    1.187536] vgaarb: loaded
[    1.187760] SCSI subsystem initialized
[    1.188012] usbcore: registered new interface driver usbfs
[    1.188047] usbcore: registered new interface driver hub
[    1.188073] usbcore: registered new device driver usb
[    1.189560] mc: Linux media interface: v0.10
[    1.189586] videodev: Linux video capture interface: v2.00
[    1.189653] pps_core: LinuxPPS API ver. 1 registered
[    1.189660] pps_core: Software ver. 5.3.6 - Copyright 2005-2007 Rodolfo Gi>
[    1.189674] PTP clock support registered
[    1.189706] EDAC MC: Ver: 3.0.0
[    1.190705] Advanced Linux Sound Architecture Driver Initialized.
[    1.191058] Bluetooth: Core ver 2.22
[    1.191079] NET: Registered protocol family 31
[    1.191086] Bluetooth: HCI device and connection manager initialized
[    1.191096] Bluetooth: HCI socket layer initialized
[    1.191105] Bluetooth: L2CAP socket layer initialized
[    1.191113] Bluetooth: SCO socket layer initialized
[    1.191319] nfc: nfc_init: NFC Core ver 0.1
[    1.191363] NET: Registered protocol family 39
[    1.192109] clocksource: Switched to clocksource arch_sys_counter
[    1.192260] VFS: Disk quotas dquot_6.6.0
[    1.192309] VFS: Dquot-cache hash table entries: 512 (order 0, 4096 bytes)
[    1.198319] NET: Registered protocol family 2
[    1.198427] IP idents hash table entries: 32768 (order: 6, 262144 bytes, l)
[    1.199407] tcp_listen_portaddr_hash hash table entries: 1024 (order: 2, 1)
[    1.199436] TCP established hash table entries: 16384 (order: 5, 131072 by)
[    1.199546] TCP bind hash table entries: 16384 (order: 6, 262144 bytes, li)
[    1.199762] TCP: Hash tables configured (established 16384 bind 16384)
[    1.199836] UDP hash table entries: 1024 (order: 3, 32768 bytes, linear)
[    1.199879] UDP-Lite hash table entries: 1024 (order: 3, 32768 bytes, line)
[    1.200027] NET: Registered protocol family 1
[    1.200330] RPC: Registered named UNIX socket transport module.
[    1.200336] RPC: Registered udp transport module.
[    1.200343] RPC: Registered tcp transport module.
[    1.200350] RPC: Registered tcp NFSv4.1 backchannel transport module.
[    1.201003] PCI: CLS 0 bytes, default 64
[    1.201724] hw perfevents: enabled with armv8_pmuv3 PMU driver, 7 counterse
```

```
[    1.206710] Initialise system trusted keyrings
[    1.206795] workingset: timestamp_bits=46 max_order=19 bucket_order=0
[    1.211625] DLM installed
[    1.212333] squashfs: version 4.0 (2009/01/31) Phillip Lougher
[    1.212845] NFS: Registering the id_resolver key type
[    1.212865] Key type id_resolver registered
[    1.212872] Key type id_legacy registered
[    1.212944] nfs4filelayout_init: NFSv4 File Layout Driver Registering...
[    1.212955] nfs4flexfilelayout_init: NFSv4 Flexfile Layout Driver Register.
[    1.212978] ntfs: driver 2.1.32 [Flags: R/W].
[    1.213113] jffs2: version 2.2. (NAND) (SUMMARY)  © 2001-2006 Red Hat, In.
[    1.213448] fuse: init (API version 7.32)
[    1.213771] SGI XFS with ACLs, security attributes, realtime, quota, no ded
[    1.252232] xor: measuring software checksum speed
[    1.256418]    8regs           :  2365 MB/sec
[    1.259935]    32regs          :  2802 MB/sec
[    1.264084]    arm64_neon      :  2380 MB/sec
[    1.264090] xor: using function: 32regs (2802 MB/sec)
[    1.264111] Key type asymmetric registered
[    1.264117] Asymmetric key parser 'x509' registered
[    1.264150] Block layer SCSI generic (bsg) driver version 0.4 loaded (majo)
[    1.264159] io scheduler mq-deadline registered
[    1.266122] samsung-hdmi-phy 32fdff00.hdmiphy: failed to get phy apb clk: 7
[    1.266262] imx8-pcie-phy 32f00000.pcie-phy: failed to get imx pcie phy clk
[    1.272711] i.MX clk 324: register failed with -2
[    1.272720] i.MX clk 325: register failed with -2
[    1.272727] i.MX clk 326: register failed with -2
[    1.272734] i.MX clk 328: register failed with -2
[    1.272742] i.MX clk 329: register failed with -2
[    1.272749] i.MX clk 330: register failed with -2
[    1.278254] imx-sdma 30bd0000.dma-controller: firmware found.
[    1.278409] imx-sdma 30bd0000.dma-controller: loaded firmware 4.5
[    1.280171] Bus freq driver module loaded
[    1.284867] Serial: 8250/16550 driver, 4 ports, IRQ sharing enabled
[    1.287139] 30860000.serial: ttymxc0 at MMIO 0x30860000 (irq = 30, base_baX
[    1.287798] 30890000.serial: ttymxc1 at MMIO 0x30890000 (irq = 31, base_baX
[    2.385364] printk: console [ttymxc1] enabled
[    2.392506] imx-lcdifv3 32e90000.lcd-controller: No pix clock get: -517
[    2.399421] imx-lcdifv3 32fc6000.lcd-controller: No irq get, ret=-517
[    2.409510] imx-hdmi-pavi 32fc4000.hdmi-pai-pvi: No pvi clock get
[    2.424356] loop: module loaded
[    2.427787] zram: Added device: zram0
[    2.431933] usbcore: registered new interface driver pn533_usb
[    2.438015] nfcsim 0.2 initialized
[    2.441452] usbcore: registered new interface driver port100
[    2.447934] imx ahci driver is registered.
[    2.454193] libphy: Fixed MDIO Bus: probed
[    2.458794] tun: Universal TUN/TAP device driver, 1.6
[    2.463983] CAN device driver interface
[    2.470522] imx-dwmac 30bf0000.ethernet: IRQ eth_lpi not found
[    2.476456] imx-dwmac 30bf0000.ethernet: no reset control found
[    2.482520] imx-dwmac 30bf0000.ethernet: User ID: 0x10, Synopsys ID: 0x51
[    2.489322] imx-dwmac 30bf0000.ethernet:     DWMAC4/5
[    2.494143] imx-dwmac 30bf0000.ethernet: DMA HW capability register suppord
[    2.501292] imx-dwmac 30bf0000.ethernet: RX Checksum Offload Engine suppord
[    2.508439] imx-dwmac 30bf0000.ethernet: TX Checksum insertion supported
[    2.515147] imx-dwmac 30bf0000.ethernet: Wake-Up On Lan supported
[    2.521298] imx-dwmac 30bf0000.ethernet: Enable RX Mitigation via HW Watchr
```

```
[    2.528965] imx-dwmac 30bf0000.ethernet: Enabled Flow TC (entries=8)
[    2.535329] imx-dwmac 30bf0000.ethernet: Enabling HW TC (entries=256, max_)
[    2.542907] imx-dwmac 30bf0000.ethernet: Using 34 bits DMA width
[    2.549637] imx-dwmac 30bf0000.ethernet: Cannot register the MDIO bus
[    2.556098] imx-dwmac 30bf0000.ethernet: stmmac_dvr_probe: MDIO bus (id: 1d
[    2.565202] pegasus: v0.9.3 (2013/04/25), Pegasus/Pegasus II USB Ethernet r
[    2.572658] usbcore: registered new interface driver pegasus
[    2.578352] usbcore: registered new interface driver rtl8150
[    2.584044] usbcore: registered new interface driver r8152
[    2.589573] usbcore: registered new interface driver lan78xx
[    2.595275] usbcore: registered new interface driver asix
[    2.600712] usbcore: registered new interface driver ax88179_178a
[    2.606840] usbcore: registered new interface driver cdc_ether
[    2.612702] usbcore: registered new interface driver dm9601
[    2.618309] usbcore: registered new interface driver CoreChips
[    2.624182] usbcore: registered new interface driver smsc75xx
[    2.629966] usbcore: registered new interface driver smsc95xx
[    2.635752] usbcore: registered new interface driver net1080
[    2.641444] usbcore: registered new interface driver plusb
[    2.646965] usbcore: registered new interface driver cdc_subset
[    2.652917] usbcore: registered new interface driver zaurus
[    2.658521] usbcore: registered new interface driver MOSCHIP usb-ethernet r
[    2.665953] usbcore: registered new interface driver cdc_ncm
[    2.671826] VFIO - User Level meta-driver version: 0.3
[    2.682647] ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver
[    2.689194] ehci-pci: EHCI PCI platform driver
[    2.693685] ehci-platform: EHCI generic platform driver
[    2.699118] ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
[    2.705309] ohci-pci: OHCI PCI platform driver
[    2.709785] ohci-platform: OHCI generic platform driver
[    2.715698] usbcore: registered new interface driver cdc_acm
[    2.721366] cdc_acm: USB Abstract Control Model driver for USB modems and s
[    2.729527] usbcore: registered new interface driver uas
[    2.734891] usbcore: registered new interface driver usb-storage
[    2.740961] usbcore: registered new interface driver usbserial_generic
[    2.747513] usbserial: USB Serial support registered for generic
[    2.753554] usbcore: registered new interface driver ftdi_sio
[    2.759325] usbserial: USB Serial support registered for FTDI USB Serial De
[    2.766666] usbcore: registered new interface driver usb_serial_simple
[    2.773216] usbserial: USB Serial support registered for carelink
[    2.779328] usbserial: USB Serial support registered for zio
[    2.785007] usbserial: USB Serial support registered for funsoft
[    2.791031] usbserial: USB Serial support registered for flashloader
[    2.797403] usbserial: USB Serial support registered for google
[    2.803343] usbserial: USB Serial support registered for libtransistor
[    2.809890] usbserial: USB Serial support registered for vivopay
[    2.815919] usbserial: USB Serial support registered for moto_modem
[    2.822206] usbserial: USB Serial support registered for motorola_tetra
[    2.828840] usbserial: USB Serial support registered for novatel_gps
[    2.835214] usbserial: USB Serial support registered for hp4x
[    2.840980] usbserial: USB Serial support registered for suunto
[    2.846917] usbserial: USB Serial support registered for siemens_mpi
[    2.853305] usbcore: registered new interface driver usb_ehset_test
[    2.861517] udc-core: couldn't find an available UDC - added [g_multi] to s
[    2.871861] snvs_rtc 30370000.snvs:snvs-rtc-lp: registered as rtc1
[    2.878267] i2c /dev entries driver
[    2.882133] IR NEC protocol handler initialized
[    2.886676] IR RC5(x/sz) protocol handler initialized
```

```
[    2.891733] IR RC6 protocol handler initialized
[    2.896268] IR JVC protocol handler initialized
[    2.900804] IR Sony protocol handler initialized
[    2.905429] IR SANYO protocol handler initialized
[    2.910138] IR Sharp protocol handler initialized
[    2.914846] IR MCE Keyboard/mouse protocol handler initialized
[    2.920684] IR XMP protocol handler initialized
[    2.925218] ir_imon_decoder: IR iMON protocol handler initialized
[    2.931316] IR RCMM protocol handler initialized
[    2.936718] usbcore: registered new interface driver uvcvideo
[    2.942472] USB Video Class driver (1.1.1)
[    2.949120] device-mapper: ioctl: 4.43.0-ioctl (2020-10-01) initialised: dm
[    2.957646] Bluetooth: HCI UART driver ver 2.3
[    2.962104] Bluetooth: HCI UART protocol H4 registered
[    2.967250] Bluetooth: HCI UART protocol BCSP registered
[    2.972588] Bluetooth: HCI UART protocol LL registered
[    2.977733] Bluetooth: HCI UART protocol ATH3K registered
[    2.983151] Bluetooth: HCI UART protocol Three-wire (H5) registered
[    2.989499] Bluetooth: HCI UART protocol Intel registered
[    2.994984] Bluetooth: HCI UART protocol Broadcom registered
[    3.000665] Bluetooth: HCI UART protocol QCA registered
[    3.005906] Bluetooth: HCI UART protocol AG6XX registered
[    3.011326] Bluetooth: HCI UART protocol Marvell registered
[    3.016946] usbcore: registered new interface driver bcm203x
[    3.022649] usbcore: registered new interface driver bpa10x
[    3.028264] usbcore: registered new interface driver bfusb
[    3.033786] usbcore: registered new interface driver btusb
[    3.039319] usbcore: registered new interface driver ath3k
[    3.045073] EDAC MC: ECC not enabled
[    3.049735] sdhci: Secure Digital Host Controller Interface driver
[    3.055926] sdhci: Copyright(c) Pierre Ossman
[    3.060418] Synopsys Designware Multimedia Card Interface Driver
[    3.067077] sdhci-pltfm: SDHCI platform and OF driver helper
[    3.074834] ledtrig-cpu: registered to indicate activity on CPUs
[    3.081336] SMCCC: SOC_ID: ARCH_SOC_ID not implemented, skipping ....
[    3.088544] caam-snvs 30370000.caam-snvs: violation handlers armed - init e
[    3.096958] usbcore: registered new interface driver usbhid
[    3.102542] usbhid: USB HID core driver
[    3.106306] mmc2: SDHCI controller on 30b60000.mmc [30b60000.mmc] using ADA
[    3.106838] mxc-mipi-csi2-sam 32e40000.csi: supply mipi-phy not found, usir
[    3.122897] mxc-md 32c00000.bus:camera: deferring cap_device registration
[    3.133168] optee: probing for conduit method.
[    3.137644] optee: revision 3.13 (2c1092df)
[    3.138454] optee: dynamic shared memory is enabled
[    3.147834] optee: initialized driver
[    3.154457] Galcore version 6.4.3.p2.336687
[    3.232442] mmc2: new HS400 MMC card at address 0001
[    3.238935] mmcblk2: mmc2:0001 S0J56X 7.30 GiB
[    3.243670] mmcblk2boot0: mmc2:0001 S0J56X partition 1 31.5 MiB
[    3.249753] mmcblk2boot1: mmc2:0001 S0J56X partition 2 31.5 MiB
[    3.255823] mmcblk2gp0: mmc2:0001 S0J56X partition 4 8.00 MiB
[    3.261721] mmcblk2gp1: mmc2:0001 S0J56X partition 5 8.00 MiB
[    3.267631] mmcblk2gp2: mmc2:0001 S0J56X partition 6 8.00 MiB
[    3.269745] [drm] Initialized vivante 1.0.0 20170808 for 40000000.mix_gpu_0
[    3.273559] mmcblk2gp3: mmc2:0001 S0J56X partition 7 8.00 MiB
[    3.282741] hantrodec 0 : module inserted. Major = 237
[    3.287841] mmcblk2rpmb: mmc2:0001 S0J56X partition 3 4.00 MiB, chardev (2)
[    3.292956] hantrodec 1 : module inserted. Major = 237
```

```
[    3.306236] hantroenc: HW at base < 0000000038320000 > with ID <0x80006200>
[    3.313214] hx280enc: module inserted. Major < 236 >
[    3.320369] usbcore: registered new interface driver snd-usb-audio
[    3.327391]  mmcblk2: p1 p2 p3 p4 p5
[    3.330540] fsl-aud2htx 30cb0000.aud2htx: failed to get mem clock
[    3.338973] imx-cdnhdmi sound-hdmi: snd_soc_register_card failed (-517)
[    3.345703] Mirror/redirect action on
[    3.349490] u32 classifier
[    3.352256]     Performance counters on
[    3.356111]     input device check on
[    3.359780]     Actions configured
[    3.363230]  mmcblk2gp1:
[    3.365602] xt_time: kernel timezone is -0000
[    3.370462] Initializing XFRM netlink socket
[    3.375232] NET: Registered protocol family 10
[    3.381554] Segment Routing with IPv6
[    3.385303] mip6: Mobile IPv6
[    3.388440] NET: Registered protocol family 17
[    3.392945] Bridge firewalling registered
[    3.396981] can: controller area network core
[    3.401387] NET: Registered protocol family 29
[    3.405838] can: raw protocol
[    3.408815] can: broadcast manager protocol
[    3.413011] can: netlink gateway - max_hops=1
[    3.417537] Bluetooth: RFCOMM TTY layer initialized
[    3.422433] Bluetooth: RFCOMM socket layer initialized
[    3.427614] Bluetooth: RFCOMM ver 1.11
[    3.431372] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[    3.436690] Bluetooth: BNEP filters: protocol multicast
[    3.441939] Bluetooth: BNEP socket layer initialized
[    3.446912] Bluetooth: HIDP (Human Interface Emulation) ver 1.2
[    3.452847] Bluetooth: HIDP socket layer initialized
[    3.459682] NET: Registered protocol family 33
[    3.464146] Key type rxrpc registered
[    3.467808] Key type rxrpc_s registered
[    3.471672] 8021q: 802.1Q VLAN Support v1.8
[    3.480632] DCCP: Activated CCID 2 (TCP-like)
[    3.485026] DCCP: Activated CCID 3 (TCP-Friendly Rate Control)
[    3.491248] sctp: Hash tables configured (bind 256/256)
[    3.497420] NET: Registered protocol family 21
[    3.501991] Registered RDS/tcp transport
[    3.505938] lib80211: common routines for IEEE802.11 drivers
[    3.511698] 9pnet: Installing 9P2000 support
[    3.516004] tsn generic netlink module v1 init...
[    3.520906] NET: Registered protocol family 36
[    3.525386] Key type dns_resolver registered
[    3.529669] Key type ceph registered
[    3.534057] libceph: loaded (mon/osd proto 15/24)
[    3.538794] mpls_gso: MPLS GSO support
[    3.542860] registered taskstats version 1
[    3.546967] Loading compiled-in X.509 certificates
[    3.557254] Loaded X.509 cert 'Build time autogenerated kernel key: f706f5'
[    3.567223] Key type ._fscrypt registered
[    3.571241] Key type .fscrypt registered
[    3.575172] Key type fscrypt-provisioning registered
[    3.581092] Btrfs loaded, crc32c=crc32c-generic
[    3.594254] M2_3P3V: supplied by VEXT_3P3V
[    3.608568] nxp-pca9450 0-0025: pca9450bc probed.
```

```
[    3.613440] i2c i2c-0: IMX I2C adapter registered
[    3.621476] rtc-rv8803 1-0032: Voltage low, temperature compensation stopp.
[    3.628635] rtc-rv8803 1-0032: Voltage low, data loss detected.
[    3.637310] rtc-rv8803 1-0032: Voltage low, data is invalid.
[    3.643687] rtc-rv8803 1-0032: registered as rtc0
[    3.649321] rtc-rv8803 1-0032: Voltage low, data is invalid.
[    3.655012] rtc-rv8803 1-0032: hctosys: unable to read the hardware clock
[    3.662507] usb251xb 1-002c: supply vdd not found, using dummy regulator
[    3.675466] random: fast init done
[    3.710279] usb251xb 1-002c: Hub configuration was successful.
[    3.716125] usb251xb 1-002c: Hub probed successfully
[    3.731942] i2c i2c-1: IMX I2C adapter registered
[    3.740570] i2c i2c-2: IMX I2C adapter registered
[    3.746184] i2c i2c-3: IMX I2C adapter registered
[    3.752265] imx8mq-usb-phy 381f0040.usb-phy: supply vbus not found, using r
[    3.760759] imx8mq-usb-phy 382f0040.usb-phy: supply vbus not found, using r
[    3.769162] samsung-hdmi-phy 32fdff00.hdmiphy: failed to get phy apb clk: 7
[    3.780992] imx6q-pcie 33800000.pcie: supply epdev_on not found, using dumr
[    3.791994] imx6q-pcie 33800000.pcie: PLL REF_CLK is used!.
[    3.792181] SoC: i.MX8MP revision 1.1
[    3.798166] imx6q-pcie 33800000.pcie: PCIe PHY PLL clock is locked.
[    3.801518] imx-cpufreq-dt imx-cpufreq-dt: cpu speed grade 7 mkt segment 24
[    3.818776] imx-lcdifv3 32fc6000.lcd-controller: No irq get, ret=-517
[    3.826599] imx-sdma 30e10000.dma-controller: firmware found.
[    3.846798] imx6q-pcie 33800000.pcie: PCIe PLL locked after 0 us.
[    3.852965] imx6q-pcie 33800000.pcie: host bridge /soc@0/pcie@33800000 ran:
[    3.860235] imx6q-pcie 33800000.pcie:   No bus range found for /soc@0/pcie]
[    3.869588] imx6q-pcie 33800000.pcie:       IO 0x001ff80000..0x001ff8ffff 0
[    3.877795] imx6q-pcie 33800000.pcie:      MEM 0x0018000000..0x001feffffff 0
[    3.886057] imx6q-pcie 33800000.pcie: invalid resource
[    4.109250] pps pps0: new PPS source ptp0
[    4.118961] libphy: fec_enet_mii_bus: probed
[    4.125458] fec 30be0000.ethernet eth0: registered PHC device 0
[    4.132187] imx-dwmac 30bf0000.ethernet: IRQ eth_lpi not found
[    4.138100] imx-dwmac 30bf0000.ethernet: no reset control found
[    4.144415] imx-dwmac 30bf0000.ethernet: User ID: 0x10, Synopsys ID: 0x51
[    4.151216] imx-dwmac 30bf0000.ethernet:     DWMAC4/5
[    4.156015] imx-dwmac 30bf0000.ethernet: DMA HW capability register supprd
[    4.163158] imx-dwmac 30bf0000.ethernet: RX Checksum Offload Engine supprd
[    4.170296] imx-dwmac 30bf0000.ethernet: TX Checksum insertion supported
[    4.177002] imx-dwmac 30bf0000.ethernet: Wake-Up On Lan supported
[    4.183098] imx-dwmac 30bf0000.ethernet: Enable RX Mitigation via HW Watchr
[    4.190759] imx-dwmac 30bf0000.ethernet: Enabled Flow TC (entries=8)
[    4.197135] imx-dwmac 30bf0000.ethernet: Enabling HW TC (entries=256, max_)
[    4.204713] imx-dwmac 30bf0000.ethernet: Using 34 bits DMA width
[    4.392439] libphy: stmmac: probed
[    4.399725] xhci-hcd xhci-hcd.1.auto: xHCI Host Controller
[    4.405256] xhci-hcd xhci-hcd.1.auto: new USB bus registered, assigned bus1
[    4.413275] xhci-hcd xhci-hcd.1.auto: hcc params 0x0220fe6c hci version 0x0
[    4.422703] xhci-hcd xhci-hcd.1.auto: irq 69, io mem 0x38100000
[    4.429295] hub 1-0:1.0: USB hub found
[    4.433064] hub 1-0:1.0: 1 port detected
[    4.437166] xhci-hcd xhci-hcd.1.auto: xHCI Host Controller
[    4.442663] xhci-hcd xhci-hcd.1.auto: new USB bus registered, assigned bus2
[    4.450332] xhci-hcd xhci-hcd.1.auto: Host supports USB 3.0 SuperSpeed
[    4.456895] usb usb2: We don't know the algorithms for LPM for this host, .
[    4.465909] hub 2-0:1.0: USB hub found
[    4.469685] hub 2-0:1.0: 1 port detected
```

```
[    4.474619] xhci-hcd xhci-hcd.2.auto: xHCI Host Controller
[    4.480127] xhci-hcd xhci-hcd.2.auto: new USB bus registered, assigned bus3
[    4.488119] xhci-hcd xhci-hcd.2.auto: hcc params 0x0220fe6c hci version 0x0
[    4.497550] xhci-hcd xhci-hcd.2.auto: irq 70, io mem 0x38200000
[    4.504474] hub 3-0:1.0: USB hub found
[    4.508245] hub 3-0:1.0: 1 port detected
[    4.512351] xhci-hcd xhci-hcd.2.auto: xHCI Host Controller
[    4.517853] xhci-hcd xhci-hcd.2.auto: new USB bus registered, assigned bus4
[    4.525520] xhci-hcd xhci-hcd.2.auto: Host supports USB 3.0 SuperSpeed
[    4.532089] usb usb4: We don't know the algorithms for LPM for this host, .
[    4.541033] hub 4-0:1.0: USB hub found
[    4.544821] hub 4-0:1.0: 1 port detected
[    4.551355] sdhci-esdhc-imx 30b50000.mmc: Got CD GPIO
[    4.551484] caam 30900000.crypto: device ID = 0x0a16040100000100 (Era 9)
[    4.563168] caam 30900000.crypto: job rings = 2, qi = 0
[    4.587229] caam algorithms registered in /proc/crypto
[    4.588548] mmc1: SDHCI controller on 30b50000.mmc [30b50000.mmc] using ADA
[    4.593246] caam 30900000.crypto: caam pkc algorithms registered in /proc/o
[    4.606926] caam 30900000.crypto: registering rng-caam
[    4.613314] Device caam-keygen registered
[    4.618810] mxc-mipi-csi2-sam 32e40000.csi: supply mipi-phy not found, usir
[    4.627543] mxc-mipi-csi2-sam 32e40000.csi: lanes: 2, hs_settle: 13, clk_s0
[    4.638726] isi-capture 32e00000.isi:cap_device: deferring 32e00000.isi:can
[    4.648753] mxc-isi 32e00000.isi: mxc_isi.0 registered successfully
[    4.655368] random: crng init done
[    4.655708] mxc-md 32c00000.bus:camera: deferring cap_device registration
[    4.666802] imx-cdnhdmi sound-hdmi: snd_soc_register_card failed (-517)
[    4.675665] imx-drm display-subsystem: bound imx-lcdifv3-crtc.0 (ops 0xfff)
[    4.683811] imx-drm display-subsystem: bound imx-lcdifv3-crtc.1 (ops 0xfff)
[    4.692062] imx-drm display-subsystem: bound 32c00000.bus:ldb@32ec005c (op)
[    4.701030] dwhdmi-imx 32fd8000.hdmi: Detected HDMI TX controller v2.13a w)
[    4.711333] dwhdmi-imx 32fd8000.hdmi: registered DesignWare HDMI I2C bus dr
[    4.720196] imx-drm display-subsystem: bound 32fd8000.hdmi (ops 0xffff8000)
[    4.728772] [drm] Initialized imx-drm 1.0.0 20120507 for display-subsystem1
[    4.780434] usb 3-1: new high-speed USB device number 2 using xhci-hcd
[    4.781256] Console: switching to colour frame buffer device 128x48
[    4.792591] mmc1: host does not support reading read-only switch, assuminge
[    4.814004] imx-drm display-subsystem: [drm] fb0: imx-drmdrmfb frame buffee
[    4.823157] mx8-img-md: Registered mxc_isi.0.capture as /dev/video2
[    4.829509] mx8-img-md: Registered sensor subdevice: imx219 1-0010 (1)
[    4.836081] mx8-img-md: created link [mxc_isi.0] => [mxc_isi.0.capture]
[    4.842700] mx8-img-md: created link [mxc-mipi-csi2.0] => [mxc_isi.0]
[    4.849146] mx8-img-md: created link [imx219 1-0010] => [mxc-mipi-csi2.0]
[    4.855937] mxc-md 32c00000.bus:camera: mxc_md_create_links
[    4.865370] input: audio-hdmi HDMI Jack as /devices/platform/sound-hdmi/so0
[    4.875042] input: gpio-keys as /devices/platform/gpio-keys/input/input1
[    4.878953] imx6q-pcie 33800000.pcie: Phy link never came up
[    4.882773] isi-m2m 32e00000.isi:m2m_device: Register m2m success for ISI.0
[    4.890186] imx6q-pcie 33800000.pcie: failed to initialize host
[    4.894476] cfg80211: Loading compiled-in X.509 certificates for regulatore
[    4.900315] imx6q-pcie 33800000.pcie: unable to add pcie port.
[    4.910491] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[    4.921546] ALSA device list:
[    4.924579]   #0: audio-hdmi
[    4.934273] EXT4-fs (mmcblk2p2): mounted filesystem with ordered data mode)
[    4.942461] VFS: Mounted root (ext4 filesystem) readonly on device 179:2.
[    4.950212] devtmpfs: mounted
[    4.953782] Freeing unused kernel memory: 2432K
```

```
[    4.972764] Run /sbin/init as init process
[    4.980292] hub 3-1:1.0: USB hub found
[    4.984090] mmc1: new ultra high speed SDR104 SDHC card at address 0007
[    4.984115] hub 3-1:1.0: 2 ports detected
[    4.995176] mmcblk1: mmc1:0007 SD16G 14.9 GiB

   OpenRC 0.43.3.bf57debcde is [    5.008424]  mmcblk1: p1 p2 p3 p4 p5
starting up Linux 5.10.52-1-at (aarch64)

 * Mounting /proc ... [ ok ]
 * Mounting /run ... * /run/openrc: creating directory
 * /run/lock: creating directory
 * /run/lock: correcting owner
 * Caching service dependencies ... [ ok ]
 * Clock skew detected with `(null)'
 * Adjusting mtime of `/run/openrc/deptree' to Thu Nov 25 19:26:50 2021

 * WARNING: clock skew detected!
 * Mounting /sys ... * Remounting devtmpfs on /dev ... [ ok ]
 [ ok ]
 * Mounting security filesystem ... [ ok ]
 * Mounting config filesystem ... [ ok ]
 * Mounting fuse control filesystem ... * Mounting /dev/mqueue ... [ ok ]
 [ ok ]
 * Mounting /dev/pts ... [ ok ]
 * Mounting /dev/shm ... [ ok ]
 * Mounting cgroup filesystem ... [ ok ]
udev             | * Starting udev ...overlayfs        | * Preparing overlay c
 [ ok ]
overlayfs        | * Preparing overlay for /var
overlayfs        | * Preparing overlay for /root
[    5.707199] udevd[706]: starting version 3.2.10
overlayfs        | * Preparing overlay for /home
fsck_atlog       |fsck.fat 4.2 (2021-01-31)
fsck_atlog       |/dev/mmcblk2gp1: 2 files, 1/4081 clusters
fsck             | * Checking local filesystems  ... [ ok ]
[    5.765935] udevd[706]: starting eudev-3.2.10
root             | * Remounting filesystems ... [ ok ]
localmount       | * Mounting local filesystems ...[    5.923330] EXT4-fs (mme
[    5.928821] EXT4-fs (mmcblk2p3): mounted filesystem with ordered data mode)
[    5.937874] BTRFS: device label app devid 1 transid 4752 /dev/mmcblk2p5 sc)
[    5.947458] BTRFS info (device mmcblk2p5): use zstd compression, level 3
[    5.954305] BTRFS info (device mmcblk2p5): turning on async discard
[    5.960585] BTRFS info (device mmcblk2p5): using free space tree
[    5.966595] BTRFS info (device mmcblk2p5): has skinny extents
[    5.981704] BTRFS info (device mmcblk2p5): enabling ssd optimizations
 [ ok ]
rngd             | * Starting rngd ... [ ok ]
 * WARNING: clock skew detected!
hostname         | * Setting hostname ...udev-trigger     | * Generating a ru]
 [ ok ]
udev-trigger     | * Populating /dev with existing devices through uevents ..]
 [ ok ]
bootmisc         | * Migrating /var/lock to /run/lock ... [ ok ]
bootmisc         | * / is not writable; unable to clean up underlying /run
bootmisc         | * Creating user login records ... [ ok ]
bootmisc         | * Wiping /var/tmp directory ... [ ok ]
syslog           | * Starting busybox syslog ... [ ok ]
```

```
dbus             | * Starting System Message Bus ... [ ok ]
[    7.098505] BTRFS: device label app devid 1 transid 62 /dev/mmcblk1p5 scan)
networkmanager   | * Starting networkmanager ... [ ok ]
 * WARNING: clock skew detected!
reset_bootcount  |Environment OK, copy 1
podman-atmark    | * Starting Start podman containers ...chronyd          | *]
[    7.758482] Microchip KSZ9131 Gigabit PHY 30be0000.ethernet-1:03: attached)
[    7.824606] imx-dwmac 30bf0000.ethernet eth1: PHY [stmmac-1:03] driver [Mi)
[    7.852130] imx-dwmac 30bf0000.ethernet eth1: No Safety Features support fd
[    7.859381] imx-dwmac 30bf0000.ethernet eth1: IEEE 1588-2008 Advanced Timed
[    7.872271] imx-dwmac 30bf0000.ethernet eth1: registered PTP clock
[    7.878773] imx-dwmac 30bf0000.ethernet eth1: configuring for phy/rgmii-ide
[    7.894988] 8021q: adding VLAN 0 to HW filter on device eth1
 [ ok ]

Welcome to Alpine Linux 3.14
Kernel 5.10.52-1-at on an aarch64 (/dev/ttymxc1)

armadillo login:
```

U-Boot プロンプト

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

```
: (省略)
 BuildInfo:
  - ATF ba76d33
  - U-Boot 2020.04-at1

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

5.2. ログイン

起動が完了するとログインプロンプトが表示されます。「表 5.1. シリアルコンソールログイン時のユーザ名とパスワード」に示すユーザでログインすることができます。

表 5.1 シリアルコンソールログイン時のユーザ名とパスワード

ユーザ名	パスワード	権限
root	root	root ユーザ
atmark	atmark	一般ユーザ

初めてログインしたときは、パスワードの変更を促されます。事前に新しいパスワードを用意してください。

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

1. root でログイン

   初期パスワードを変更します。

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

   ❶ 新しいパスワードを入力します

   ❷ 新しいパスワードを再入力します

2. atmark でログイン

   初期パスワードを変更します。root でログインと同様にパスワードを入力します。

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

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

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

persist_file コマンドに関する詳細は「9.8.2. overlayfs と persist_file について」を参照してください。

5.3. 終了方法

安全に終了させる場合は、次のようにコマンドを実行し、「reboot: Power down」と表示されたのを確認してから電源を切断します。

```
[armadillo ~]# poweroff
* WARNING: clock skew detected!
urandom          | * Saving random seed ...chronyd          | * Stopping chronyd ...syslog
| * Stopping busybox syslog ... [ ok ]
```

```
overlayfs        | * Unmounting /etc ...rngd             | * Stopping rngd ...podman-atmark    | *
Stopping Start podman containers ... [ ok ] [ ok ]

 * start-stop-daemon: no matching processes found
podman-atmark    | [ ok ]
 [ ok ]
udev             | * Stopping udev ...hwclock          | * Setting hardware clock using the system
clock [UTC] ... [ ok ]
 * in use but fuser finds nothing
overlayfs        | [ !! ]
overlayfs        | * Unmounting /dev/shm/overlay_etc_lower ... [ ok ]
networkmanager   | * Stopping networkmanager ...overlayfs        | * Unmounting /var ...nm-
dispatcher: req:1 'connectivity-change': find-scripts: Cannot execute '/etc/Netw.
 [ ok ]
dbus             | * Stopping System Message Bus ...nm-dispatcher: System bus stopped. Exiting
 * in use but fuser finds nothing
overlayfs        | [ !! ]
 [ ok ]
overlayfs        | * Unmounting /dev/shm/overlay_var_lower ... [ ok ]
overlayfs        | * Unmounting /root ... [ ok ]
overlayfs        | * Unmounting /dev/shm/overlay_root_lower ... [ ok ]
localmount       | * Unmounting loop devices
overlayfs        | * Unmounting /home ...localmount       | * Unmounting filesystems
localmount       | *   Unmounting /var/tmp ... [ ok ]
localmount       | *   Unmounting /var/app/volumes ... [ ok ]
localmount       | *   Unmounting /var/app/rollback/volumes ... [ ok ]
localmount       | *   Unmounting /var/lib/containers/storage_readonly ... [ ok ]
localmount       | *   Unmounting /var/log ... [ ok ]
localmount       | *   Unmounting /tmp ... [ ok ]
localmount       | *   Unmounting /home ...
 *   in use but fuser finds nothing
localmount       | [ !! ]
localmount       | *   Unmounting /var ... [ ok ]
localmount       | *   Unmounting /etc ... *   in use but fuser finds nothing
localmount       | [ !! ]
 [ ok ]
overlayfs        | * Unmounting /dev/shm/overlay_home_lower ... [ ok ]
killprocs        | * Terminating remaining processes ...mount-ro         | * Remounting remaining
filesystems read-only ... *   Remounting /etc read only ... [ ok ]
mount-ro         | *   Remounting / read only ... [ ok ]
mount-ro         | [ ok ]
The system is going down NOW!
Sent SIGTERM to all processes
Sent SIGKILL to all processes
Requesting system poweroff
[   62.855146] imx2-wdt 30280000.watchdog: Device shutdown: Expect reboot!
[   62.862470] reboot: Power down
```

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

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

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

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

6. ユーザー登録

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

ユーザー登録すると次のようなサービスを受けることができます。

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

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

アットマークテクノ Armadillo サイト

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

6.1. 購入製品登録

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

購入製品登録の詳しい手順は以下の URL をご参照ください。

Armadillo-IoT G4 購入製品登録

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

7. 動作確認方法

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

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

7.1. ネットワーク

ここでは、ネットワークの設定方法について説明します。

7.1.1. 接続可能なネットワーク

Armadillo-IoT ゲートウェイ G4 は、2 つの Ethernet ポートが搭載されています。Linux からは、それぞれ `eth0` 、 `eth1` に見えます。

表 7.1 ネットワークとネットワークデバイス

ネットワーク	ネットワークデバイス	出荷時の設定
Ethernet	`eth0`	DHCP
Ethernet	`eth1`	DHCP

eth1(LAN インターフェース 2)は 10Mbps(10BASE-T)に非対応です。10Mbps で通信を行う場合は、LAN インターフェース 0(LAN インターフェース 1)をご利用ください。

7.1.2. ネットワークの設定方法

Armadillo-IoT ゲートウェイ G4 では、通常の Linux システムと同様、ネットワークインターフェースの設定は NetworkManager を使用します。NetworkManager はすべてのネットワーク設定をコネクションとして管理します。コネクションには「どのようにネットワークへ接続するか」、「どのようにネットワークを作成するか」を記述し、 `/etc/NetworkManager/system-connections/` に保存します。また、1 つのデバイスに対して複数のコネクションを保存することは可能ですが、1 つのデバイスに対して有効化にできるコネクションは 1 つだけです。

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

7.1.2.1. nmcli について

`nmcli` は NetworkManager を操作するためのコマンドラインツールです。「図 7.1. nmcli のコマンド書式」に `nmcli` の書式を示します。このことから、 `nmcli` は「オブジェクト (OBJECT) というものが存在し、それぞれのオブジェクトに対してコマンド (COMMAND) を実行する。」という書式でコマンドを入力することがわかります。また、オブジェクトそれぞれに `help` が用意されていることもここから読み取れます。

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

図 7.1 nmcli のコマンド書式

7.1.3. nmcli の基本的な使い方

ここでは nmcli の、基本的な使い方を説明します。

7.1.3.1. コネクションの一覧

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

```
[armadillo ~]# nmcli connection
NAME             UUID                                  TYPE      DEVICE
Ifupdown (eth0)  681b428f-beaf-8932-dce4-687ed5bae28e  ethernet  eth0
Ifupdown (eth1)  7b635ed6-2640-7ad8-675d-744db12dd9fa  ethernet  --
```

図 7.2 コネクションの一覧

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

7.1.3.2. コネクションの有効化・無効化

コネクションを有効化するには、次のようにコマンドを実行します。

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

図 7.3 コネクションの有効化

コネクションを無効化するには、次のようにコマンドを実行します。

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

図 7.4 コネクションの無効化

7.1.3.3. コネクションの作成

コネクションを作成するには、次のようにコマンドを実行します。

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

図 7.5 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/

[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。

system-connections/ に[ID]の名前でコネクションファイルが作成されます。このファイルを vi などで編集し、コネクションを修正することも可能です。

Armadillo-IoT ゲートウェイ G4 を再起動したときにコネクションファイルが消えてしまわないように、persist_file コマンドで永続化する必要があります。persist_file コマンドに関する詳細は「9.8.2. overlayfs と persist_file について」を参照してください。

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

図 7.6 コネクションファイルの永続化

7.1.3.4. コネクションの削除

コネクションを削除するには、次のようにコマンドを実行します。

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

図 7.7 コネクションの削除

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

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

図 7.8 コネクションファイル削除時の永続化

7.1.3.5. 固定 IP アドレスに設定する

「表 7.2. 固定 IP アドレス設定例」の内容に設定する例を、「図 7.9. 固定 IP アドレス設定」に示します。

表 7.2 固定 IP アドレス設定例

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

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

図 7.9 固定 IP アドレス設定

7.1.3.6. DNS サーバーを指定する

DNS サーバーを指定する例を、「図 7.10. DNS サーバーの指定」に示します。

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

図 7.10 DNS サーバーの指定

7.1.3.7. DHCP に設定する

DHCP に設定する例を、「図 7.11. DNS サーバーの指定」に示します。

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

図 7.11 DNS サーバーの指定

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

7.1.3.8. コネクションの修正を反映する

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

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

図 7.12 コネクションの修正の反映

7.1.3.9. デバイスの一覧

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

```
[armadillo ~]# nmcli device
DEVICE  TYPE      STATE        CONNECTION
eth0    ethernet  connected    Ifupdown (eth0)
eth1    ethernet  unavailable  --
lo      loopback  unmanaged    --
```

図 7.13 デバイスの一覧

7.1.3.10. デバイスの接続

デバイスを接続するには、次のようにコマンドを実行します。

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

図 7.14 デバイスの接続

デバイスを接続するには、接続しようとしているデバイスの有効なコネクションが必要です。"Error: neither a valid connection nor devicegiven"

というメッセージが表示された場合には、 nmcli connection などで有効なコネクションがあるかを確認してください。

7.1.3.11. デバイスの切断

デバイスを切断するには、次のようにコマンドを実行します。

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

図 7.15 デバイスの切断

7.1.4. 有線 LAN の接続を確認する

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

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

```
[armadillo ~]# ping -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 7.16 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。設定を必ず確認してください。確実に有線 LAN の接続確認をする場合は、有線 LAN 以外のインターフェースを無効化してください。

7.2. ストレージ

Armadillo-IoT ゲートウェイ G4 でストレージとして使用可能なデバイスを次に示します。

表 7.3 ストレージデバイス

デバイス種類	ディスクデバイス	先頭パーティション	インターフェース
オンボード eMMC	/dev/mmcblk2	/dev/mmcblk2p1	オンボード
オンボード eMMC (GPP)	/dev/mmcblk2gp3	なし	オンボード
microSD/microSDHC/ microSDXC カード	/dev/mmcblk1	/dev/mmcblk1p1	SD インターフェース (CON1)

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
USB メモリ	/dev/sd* [a]	/dev/sd*1	USB ホストインターフェース (CON4)

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

GPP(General Purpose Partition)について

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

Armadillo-IoT ゲートウェイ G4 では、8 MiB の GPP を 4 つ作成しています。各領域の用途を「表 7.4. eMMC の GPP の用途」に示します。

表 7.4 eMMC の GPP の用途

ディスクデバイス	用途
/dev/mmcblk2gp0	ライセンス情報等の保存
/dev/mmcblk2gp1	予約領域
/dev/mmcblk2gp2	予約領域
/dev/mmcblk0gp3	ユーザー領域

7.2.1. ストレージの使用方法

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

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

mount コマンドの典型的なフォーマットは、次の通りです。

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

図 7.17 mount コマンド書式

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

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

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

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

SD インターフェース (CON1) に microSD カードを挿入し、以下に示すコマンドを実行すると、 /media ディレクトリに microSD カードのファイルシステムをマウントすることができます。 microSD カード内のファイルは、/mnt ディレクトリ以下に見えるようになります。

```
[armadillo ~]# mount -t vfat /dev/mmcblk1p1 /mnt
[armadillo ~]# ls /mnt
  :
  :
```

図 7.18 ストレージのマウント

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

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

図 7.19 ストレージのアンマウント

7.2.2. ストレージのパーティション変更とフォーマット

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

パーティション構成を変更したい場合、 fdisk コマンドを使用します。 fdisk コマンドの使用例として、一つのパーティションで構成されている microSD カードのパーティションを、2 つに分割する例を「図 7.20. fdisk コマンドによるパーティション変更」に示します。一度、既存のパーティションを削除してから、新たにプライマリパーティションを二つ作成しています。先頭のパーティションには 100MByte、二つめのパーティションに残りの容量を割り当てています。先頭のパーティションは /dev/mmcblk1p1 、二つめは /dev/mmcblk1p2 となります。

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

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


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

Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): p
```

```
Partition number (1-4, default 1): 1
First sector (2048-15138815, default 2048):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-15138815, default 15138815): +100M

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

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

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

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

図 7.20 fdisk コマンドによるパーティション変更

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

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

図 7.21 EXT4 ファイルシステムの構築

7.3. LED

Armadillo-IoT ゲートウェイ G4 の LED は GPIO で接続されているため、ソフトウェアで制御することができます。

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

表 7.5 LED クラスディレクトリと LED の対応

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

7.3.1. LED を点灯/消灯する

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

brightness に 0 以外の値を書き込むと LED が点灯します。

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

図 7.22 LED を点灯させる

Armadillo-IoT ゲートウェイ G4 の LED には輝度制御の機能がないため、0(消灯)、1〜255(点灯)の 2 つの状態のみ指定することができます。

brightness に 0 を書き込むと LED が消灯します。

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

図 7.23 LED を消灯させる

brightness を読み出すと LED の状態が取得できます。

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

図 7.24 LED の状態を表示する

7.3.2. トリガを使用する

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

表 7.6 LED トリガの種類

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

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

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

図 7.25 対応している LED トリガを表示

以下のコマンドを実行すると、心拍のように点灯/消灯を行います。

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

図 7.26 LED のトリガに heartbeat を指定する

7.4. ユーザースイッチ

Armadillo-IoT ゲートウェイ G4 のユーザースイッチのデバイスドライバは、インプットデバイスとして実装されています。インプットデバイスのデバイスファイルからボタンプッシュ/リリースイベントを取得することができます。

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

表 7.7 インプットデバイスファイルとイベントコード

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

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

7.4.1. イベントを確認する

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

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

図 7.27 evtest コマンドのインストール

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

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

図 7.28 ユーザースイッチ: イベントの確認

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

8. 開発の基本的な流れ

8.1. アプリケーション開発の流れ

8.1.1. Armadillo への接続

8.1.1.1. シリアルコンソール

Armadillo-IoT ゲートウェイ G4 では CON6 (USB コンソールインターフェース)をシリアルコンソールとして使用できます。シリアル通信設定等については、「4.2.4. シリアル通信ソフトウェア(minicom)の使用」を参照してください。

ログイン方法については、「5.2. ログイン」を参照してください。

8.1.1.2. ssh

Armadillo-IoT ゲートウェイ G4 には openssh がインストールされていますが、デフォルトでは SSH サーバーが起動していません。

SSH サーバーを自動的に起動するようにするためには、以下のコマンドを実行してください。

```
[armadillo:~]# rc-update add sshd
 * service sshd added to runlevel default
[armadillo ~]# persist_file /etc/runlevels/default/sshd
[ 2819.277066] EXT4-fs (mmcblk2p1): re-mounted. Opts: (null)
[armadillo ~]# reboot
```

上記の例では、再起動後も設定が反映されるように、 `persist_file` コマンドで eMMC に設定を保存しています。

8.1.2. overlayfs の扱い

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

その為、ファイルを変更した後 Armadillo の電源を切ると変更内容は保持されません。開発中などに rootfs の変更内容を保持するには、変更したファイルに対して `persist_file` コマンドを使用します。`persist_file` コマンドの詳細は 「9.8.2. overlayfs と persist_file について」 参照してください。

開発以外の時は安全のため、ソフトウェアアップデートによる更新を実行してください。アップデート手順に関しては 「9.7. Armadillo のソフトウェアをアップデートする」 を参照してください。

rootfs の内容を変更しても、ソフトウェアアップデートを実施した際に変更した内容が保持されない可能性があります。ソフトウェアアップデート実施後も変更内容を保持する手順に関しては 「9.7.5.1. swupdate_preserve_files について」 を参照してください。

8.1.3. Podman のデータを eMMC に保存する

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

eMMC への書き込みを最小限にする等の観点から、 Armadillo の運用時は、 Podman のデータは tmpfs に保存するのが適切です。Armadillo 開発時のみ、 eMMC に Podman のデータが保存されるようにすることを推奨します。

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

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

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

コンテナのデータをボリュームに入れたら消えません。詳しくは 「8.1.6. データを保存する」 を参照してください。

8.1.4. ベースとなるコンテナを取得する

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

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

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

8.1.5. デバイスのアクセス権を与える

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

`podman run` コマンドに `--device` オプションでデバイスファイルを指定します。

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

8.1.6. データを保存する

アプリケーションからデータを保存する場合は、eMMC に保存する必要があります。コンテナ自体のデータは基本的に RAM に保存されているか書き込み不可であるため、再起動すると消えてしまいます。

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/: 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. /var/app/rollback/volumes/: アップデートの際にコピーしてアップデートを行うので、アップデート中でも安全に使いつづけます。アプリケーションと一緒にアップデートするようなデータの保存に向いています。

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

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

8.1.7. アプリケーションを作成する

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

8.1.8. コンテナを保存する

コンテナが完成したら、 `podman commit` コマンドを実行して Podman 内のストレージに保存します。詳しい手順は「9.1.2.6. コンテナの変更を保存する」を参考にしてください。

8.2. アプリケーションコンテナの運用

「9.2. コンテナの運用」を参考にしてください。

8.2.1. アプリケーションの自動起動

podman_start 用の設定ファイル(/etc/atmark/containers/*.conf)を作成します。その後、`podman_start -a` コマンドを実行するか、armadillo を再起動してコンテナが自動起動することを確認してください。コンテナの自動起動に関する詳しい説明は「9.2.1. コンテナの自動起動」を参考にしてください。

8.2.2. アプリケーションの送信

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

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

手順の詳しい説明やオプションは「9.7. Armadillo のソフトウェアをアップデートする」を参考にしてください。

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

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

ここで作成した initial_setup_container.swu ファイルを USB メモリに配置して、Armadillo-IoT ゲートウェイ G4 に刺すとインストールされます。

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

8.2.3. インストール確認：初期化

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

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

8.2.4. アプリケーションのアップデート

アップデートを行う方法は以下の二通りです：

1. podman run コマンドで、差分アップデートを行う

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

   ```
   [armadillo ~]# podman run --name update myimage:1 sh -c "apk update && apk upgrade && apk ⏎
   cache --purge"
   [armadillo ~]# podman commit update myimage:2
   [armadillo ~]# podman rm update
   [armadillo ~]# podman_partial_image -b myimage:1 -o myimage2_update.tar myimage:2
   ```

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

   詳しいコンテナアップデートの手順は「9.1.2.7. コンテナの自動作成やアップデート」を参考にしてください。

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

2. コンテナを新しく構築する

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

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

自動的にイメージを作る方法は「9.1.2.7. コンテナの自動作成やアップデート」を参考にしてください。

8.3. VPU や NPU を使用する

VPU や NPU などを使うアプリケーションを ATDE 上で開発する場合や、Armadillo Base OS 上のコンテナ内で動作させる場合、ライブラリを ATDE 上でビルドする必要があります。ここではその手順について説明します。

8.3.1. ATDE にクロスコンパイル用ライブラリをインストールする

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

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

図 8.1 ビルドツール実行前の準備

その後、ビルドツールを実行します。

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

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

図 8.2 ビルドツールの実行

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

8.3.2. Armadillo へ書き込むためのライブラリイメージを作成する

以下に示す製品では、出荷状態でライブラリイメージが Armadillo に書き込まれています。このため、ここで説明する手順はライブラリをアップデートする場合や、「9.5.1. ブートディスクの作成」 または「9.6.1. インストールディスクの作成」 の手順に従ってディスクイメージを作成する場合に必要となります。

表 8.1 ライブラリイメージ書き込み済みの製品

名称	型番
Armadillo-IoT ゲートウェイ G4 LAN モデル開発セット	AGX4500-C00D0
Armadillo-IoT ゲートウェイ G4 LAN モデル量産用	AGX4500-C00Z
Armadillo-IoT ゲートウェイ G4 LAN モデル量産ボード	AGX4500-U00Z

Armadillo Base OS 上のコンテナ内から利用できるイメージを作成します。

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

図 8.3 ライブラリイメージ作成ツールの実行

VPU を使用しない場合は、--without-vpu オプションを付けてください。

```
[ATDE ~]$ cd at-imxlibpackage
[ATDE ~/at-imxlibpackage]$ make-imxlibimage --without-vpu
```

図 8.4 ライブラリイメージ作成ツールの実行 (VPU が不要の場合)

実行が完了すると imx_lib.img というファイルが生成されます。

8.3.3. Armadillo にライブラリイメージを書き込む

Armadillo Base OS 上で、「8.3.2. Armadillo へ書き込むためのライブラリイメージを作成する」で作成した imx_lib.img を eMMC の /dev/mmcblk2p4 パーティションに書き込みます。

次のコマンドは、 imx_lib.img が /tmp にある場合の実行例です。

```
[armadillo ~]$ umount /opt/firmware
[armadillo ~]$ dd if=/tmp/imx_lib.img of=/dev/mmcblk2p4 bs=1M conv=fsync
23+1 records in
23+1 records out
24965120 bytes (25 MB, 24 MiB) copied, 0.357741 s, 69.8 MB/s
```

図 8.5 ライブラリイメージを書き込む

書き込みが完了した後、/opt/firmware にマウントします。

```
[armadillo ~]$ mount /opt/firmware
```

図 8.6 ライブラリパーティションのマウント

8.3.4. コンテナ内からライブラリを使用するための準備

コンテナ内からライブラリを使用するためには、コンテナ作成時にライブラリの場所を明示する必要があります。

--volume オプションにファームウェアが書き込まれているディレクトリ(/opt/firmware)を、--env オプションにライブラリのパスを指定します。次の例では、コンテナイメージに Debian(bullseye) を利用しています。

```
[armadillo ~]$ podman run -it --name=container_name ¥
--volume=/opt/firmware:/opt/firmware ¥ ❶
--env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu ¥ ❷
docker.io/debian:bullseye /bin/bash
```

図 8.7 コンテナ作成時に /opt/firmware を渡す例

❶ --volume に /opt/firmware を指定します。

❷ --env に LD_LIBRARY_PATH を指定し、コンテナ内のアプリケーションからライブラリをリンクできるようにします。

次に、コンテナにログインし、/opt/firmware/usr/lib/aarch64-linux-gnu/imx-mm へのシンボリックリンクを/usr/lib/aarch64-linux-gnu/ に作成します。

```
[armadillo ~]$ podman exec -it container_name /bin/bash
[conteiner ~]# ln -s /opt/firmware/usr/lib/aarch64-linux-gnu/imx-mm /usr/lib/aarch64-linux-gnu
```

図 8.8 imx-mm へのシンボリックリンクを作成する

以上で、コンテナからライブラリを使用できるようになります。

9. Howto

9.1. アプリケーションをコンテナで実行する

9.1.1. Podman - コンテナ仮想化ソフトウェア

9.1.1.1. Podman - コンテナ仮想化ソフトウェアとは

コンテナとはホスト OS 上に展開される仮想的なユーザ空間のことです。コンテナを使用することで複数の Armadillo-IoT ゲートウェイ G4 でも同一の環境がすぐに再現できます。ゲスト OS を必要としない仮想化であるため、アプリケーションの起動が素早いという特徴があります。

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

9.1.2. コンテナを操作する

この章では、コンテナ仮想化ソフトウェアの 1 つである Podman の基本的な使い方について説明します。Armadillo-IoT ゲートウェイ G4 で実行させたいアプリケーションとその実行環境自体を 1 つの Podman イメージとして扱うことで、複数の Armadillo-IoT ゲートウェイ G4 がある場合でも、全てのボード上で同一の環境を再現させることが可能となります。

この章全体を通して、イメージの公開・共有サービスである Docker Hub [https://hub.docker.com] から取得した、Alpine Linux のイメージを使って説明します。

9.1.2.1. イメージからコンテナを作成する

イメージからコンテナを作成するためには、podman run コマンドを実行します。イメージは Docker Hub [https://hub.docker.com] から自動的に取得されます。ここでは、簡単な例として "ls /" コマンドを実行するコンテナを作成します。

```
[armadillo ~]# podman run -it --name=my_container docker.io/alpine ls /
Trying to pull docker.io/library/alpine:latest...
Getting image source signatures
...
...
Writing manifest to image destination
Storing signatures
[ 3023.533900] IPv6: ADDRCONF(NETDEV_CHANGE): veth57ddda14: link becomes ready
[ 3023.541031] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[ 3023.547637] cni-podman0: port 1(veth57ddda14) entered blocking state
[ 3023.554026] cni-podman0: port 1(veth57ddda14) entered disabled state
[ 3023.560542] device veth57ddda14 entered promiscuous mode
[ 3023.565958] cni-podman0: port 1(veth57ddda14) entered blocking state
[ 3023.572318] cni-podman0: port 1(veth57ddda14) entered forwarding state
[ 3023.788098] cgroup: podman (1758) created nested cgroup for controller ”memory” which has ⏎
incomplete hierarchy suppo.
[ 3023.803249] cgroup: ”memory” requires setting use_hierarchy to 1 on the root
[ 3023.812611] cgroup: cgroup: disabling cgroup2 socket matching due to net_prio or net_cls ⏎
activation
[ 3023.837443] kmem.limit_in_bytes is deprecated and will be removed. Please report your usecase ⏎
```

```
to linux-mm@kvack.org .
bin    etc    lib    mnt    proc   run    srv    tmp    var
dev    home   media  opt    root   sbin   sys    usr
armadillo:~# [ 3024.155747] cni-podman0: port 1(veth57ddda14) entered disabled state
[ 3024.162725] device veth57ddda14 left promiscuous mode
[ 3024.167819] cni-podman0: port 1(veth57ddda14) entered disabled state
```

図 9.1 コンテナを作成する実行例

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

podman run コマンドの詳細は --help オプションで確認できます。

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

図 9.2 podman run --help の実行例

9.1.2.2. イメージ一覧を表示する

コンテナを作成するためのイメージは、イメージ一覧を表示する podman images コマンドで確認できます。

```
[armadillo ~]# podman images
REPOSITORY                TAG     IMAGE ID      CREATED      SIZE
docker.io/library/alpine  latest  9c74a18b2325  2 weeks ago  4.09 MB
```

図 9.3 イメージ一覧の表示実行例

podman images コマンドの詳細は --help オプションで確認できます。

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

図 9.4 podman images --help の実行例

9.1.2.3. コンテナー一覧を表示する

作成済みコンテナー一覧を表示するためには podman ps コマンドを実行します。

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

図 9.5 コンテナー一覧の表示実行例

一覧表示により、コンテナ名やコンテナ ID を確認することができます。-a オプションを付けない場合は、動作中のコンテナのみ表示されます。podman ps コマンドの詳細は --help オプションで確認できます。

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

図 9.6 podman ps --help の実行例

9.1.2.4. コンテナを起動する

作成済みのコンテナを起動するためには podman start コマンドを実行します。

```
[armadillo ~]# podman start my_container
podman start my_container
[ 3119.081068] IPv6: ADDRCONF(NETDEV_CHANGE): vethe172e161: link becomes ready
[ 3119.088214] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[ 3119.094812] cni-podman0: port 1(vethe172e161) entered blocking state
[ 3119.101231] cni-podman0: port 1(vethe172e161) entered disabled state
[ 3119.107745] device vethe172e161 entered promiscuous mode
[ 3119.113185] cni-podman0: port 1(vethe172e161) entered blocking state
[ 3119.119546] cni-podman0: port 1(vethe172e161) entered forwarding state
my_container
[ 3119.620731] cni-podman0: port 1(vethe172e161) entered disabled state
[ 3119.627696] device vethe172e161 left promiscuous mode
[ 3119.632762] cni-podman0: port 1(vethe172e161) entered disabled state
```

図 9.7 コンテナを起動する実行例

-a オプションを与えると、コンテナ内で実行されたアプリケーションの出力を確認できます。

```
[armadillo ~]# podman start -a my_container
[ 3150.303962] IPv6: ADDRCONF(NETDEV_CHANGE): vetha9ef8f8e: link becomes ready
[ 3150.311106] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[ 3150.317703] cni-podman0: port 1(vetha9ef8f8e) entered blocking state
[ 3150.324139] cni-podman0: port 1(vetha9ef8f8e) entered disabled state
[ 3150.330687] device vetha9ef8f8e entered promiscuous mode
[ 3150.336085] cni-podman0: port 1(vetha9ef8f8e) entered blocking state
[ 3150.342443] cni-podman0: port 1(vetha9ef8f8e) entered forwarding state
bin    etc    lib    mnt    proc   run    srv    tmp    var
dev    home   media  opt    root   sbin   sys    usr
[ 3150.804164] cni-podman0: port 1(vetha9ef8f8e) entered disabled state
[ 3150.811249] device vetha9ef8f8e left promiscuous mode
[ 3150.816349] cni-podman0: port 1(vetha9ef8f8e) entered disabled state
```

図 9.8 コンテナを起動する実行例(a オプション付与)

ここで起動している my_container は、起動時に "ls /" を実行するようになっているので、その結果が出力されます。podman start コマンドの詳細は --help オプションで確認できます。

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

図 9.9 podman start --help 実行例

9.1.2.5. コンテナを停止する

動作中のコンテナを停止するためには podman stop コマンドを実行します。

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

図 9.10 コンテナを停止する実行例

podman stop コマンドの詳細は --help オプションで確認できます。

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

図 9.11 podman stop --help 実行例

9.1.2.6. コンテナの変更を保存する

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

変更を保存するには podman commit コマンドを実行してください。

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

図 9.12 my_container を保存する例

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

9.1.2.7. コンテナの自動作成やアップデート

podman run, podman commit でコンテナを作成できますが、定期的にアップデートをする際にはコンテナの作成やアップデートを自動化できると便利です。

これを実現するために、Dockerfile と podman build を使います。この手順は Armadillo で実行可能です。

1. イメージを docker.io のイメージから作りなおします

```
[armadillo ~/podman-build]# cat Dockerfile
FROM docker.io/arm64v8/alpine:latest

# update & install dependencies (example: usbutils)
RUN apk update && apk upgrade && apk add usbutils && rm -f /var/cache/apk/*

# copy our application and set it to run on start
COPY my_application /my_application
ENTRYPOINT /my_application

[armadillo ~/podman-build]# podman build -t my_image:1 -t my_image:latest .
STEP 1: FROM docker.io/arm64v8/alpine:latest
STEP 2: RUN apk update && apk upgrade && apk add usbutils && rm -f /var/cache/apk/*
--> 234bf79175e
STEP 3: COPY my_application /my_application
--> 05ab31bb278
STEP 4: ENTRYPOINT /my_application
STEP 5: COMMIT my_image:latest
--> 590e3ba6d55
Successfully tagged localhost/my_image:1
Successfully tagged localhost/my_image:latest
590e3ba6d55f3e29bdef158d7283e9c4f7515567b2d3f978cfab2510dc02376b

[armadillo ~/podman-build]# podman save my_image:latest -o my_image_1.tar
```

図 9.13 podman build の実行例

2. イメージを前のバージョンからアップデートします

```
[armadillo podman-build-update]# cat Dockerfile
FROM localhost/my_image:latest

# update OS packages
RUN apk update && apk upgrade && rm -f /var/cache/apk/*

# update application
COPY my_application /my_application
$ podman build -t my_image:2 -t my_image:latest .
STEP 1: FROM localhost/my_image:latest
STEP 2: RUN apk update && apk upgrade && rm -f /var/cache/apk/*
--> 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 9.14 podman build でのアップデートの実行例

この場合、 podman_partial_image コマンドを使って、差分だけをインストールすることもできます。

```
[armadillo podman-build-update]# podman_partial_image -b my_image:1 ¥
        -o my_image_2_partial.tar my_image:2

[armadillo podman-build-update]# ls -lh
-rw-r--r-- 1 root root   88 Dec 21 15:24 Dockerfile
-rw-r--r-- 1 root root 9.4M Dec 21 15:26 my_image_1.tar
-rw-r--r-- 1 root root 9.4M Dec 21 15:26 my_image_2.tar
-rw-r--r-- 1 root root  51K Dec 21 15:26 my_image_2_partial.tar
```

作成した .tar アーカイブは 「9.7.5. mkswu の desc ファイル」 の swdesc_embed_container と swdesc_usb_container で使えます。

9.1.2.8. コンテナを削除する

作成済みコンテナを削除する場合は podman rm コマンドを実行します。

```
[armadillo ~]# podman rm my_container
d6de5881b5fb973227b84d1d74abf269ac3183aad7e18b7a9d85208632641d94
[armadillo ~]# podman ps -a
CONTAINER ID  IMAGE                           COMMAND    CREATED         STATUS                     ⏎
PORTS    NAMES
```

図 9.15 コンテナを削除する実行例

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

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

図 9.16 $ podman rm --help 実行例

9.1.2.9. イメージを削除する

podman のイメージを削除するには podman rmi コマンドを実行します。イメージを削除するためには、そのイメージから作成したコンテナを先に削除しておく必要があります。podman rmi コマンドにはイメージ ID を指定する必要があるため、podman images コマンドで確認します。

```
[armadillo ~]# podman rm my_container
[armadillo ~]# podman images
REPOSITORY                TAG         IMAGE ID      CREATED      SIZE
docker.io/library/alpine  latest      02480aeb44d7  2 weeks ago  5.62 MB
[armadillo ~]# podman rmi 02480aeb44d7
Untagged: docker.io/library/alpine:latest
Deleted: 02480aeb44d78f1a44b8791af7edf7d6e1b18707397a1dfb3ff4f21c5ce4a44f
[armadillo ~]# podman images
REPOSITORY                TAG         IMAGE ID      CREATED      SIZE
```

図 9.17 イメージを削除する実行例

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

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

図 9.18 podman rmi --help 実行例

9.1.2.10. 実行中のコンテナに接続する

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

```
[armadillo ~]# podman run -itd --name=cat_container docker.io/alpine cat
[ 3293.447517] IPv6: ADDRCONF(NETDEV_CHANGE): veth2a539346: link becomes ready
[ 3293.454645] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[ 3293.461201] cni-podman0: port 1(veth2a539346) entered blocking state
[ 3293.467619] cni-podman0: port 1(veth2a539346) entered disabled state
[ 3293.474139] device veth2a539346 entered promiscuous mode
[ 3293.479554] cni-podman0: port 1(veth2a539346) entered blocking state
[ 3293.485913] cni-podman0: port 1(veth2a539346) entered forwarding state
f62e7a666d7156d261905c8406c72fc271534fa29e69771c76f4f6660a2da41a
[armadillo ~]# podman exec -it cat_container /bin/sh
[container ~]# ps
PID   USER     TIME  COMMAND
    1 root      0:00 cat
    2 root      0:00 /bin/sh
    3 root      0:00 ps
```

図 9.19 コンテナ内部のシェルを起動する実行例

podman run コマンドでコンテナを作成し、その後作成したコンテナ内で /bin/sh を実行しています。/bin/sh を実行すると、コンテナ内のプロンプトが表示されコンテナ内部を操作できるようになります。上記ではコンテナ内で、ps コマンドを実行しています。コンテナ作成時に実行した cat と podman exec で実行した/bin/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 9.20 コンテナ内部のシェルから抜ける実行例

podman exec コマンドの詳細は --help オプションで確認できます。

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

図 9.21 podman exec --help 実行例

9.1.2.11. コンテナ間で通信をする

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

コンテナには作成した時点でローカル IP アドレスが割り当てられるので、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 ~]# podman run -itd --name=my_container_1 docker.io/alpine /bin/sh
[ 3336.121730] IPv6: ADDRCONF(NETDEV_CHANGE): veth487e45d1: link becomes ready
[ 3336.128875] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[ 3336.135485] cni-podman0: port 2(veth487e45d1) entered blocking state
[ 3336.141877] cni-podman0: port 2(veth487e45d1) entered disabled state
[ 3336.148395] device veth487e45d1 entered promiscuous mode
[ 3336.153792] cni-podman0: port 2(veth487e45d1) entered blocking state
[ 3336.160151] cni-podman0: port 2(veth487e45d1) entered forwarding state
cbe0802f4e2d2fec88f4e300dabeba3b48865359dc02cbd99375b1b38c2c28eb
[armadillo ~]# podman run -itd --name=my_container_2 docker.io/alpine /bin/sh
podman run -itd --name=my_container_2 docker.io/alpine /bin/sh
[ 3351.593435] IPv6: ADDRCONF(NETDEV_CHANGE): vethad55f5c8: link becomes ready
[ 3351.600594] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[ 3351.607579] cni-podman0: port 3(vethad55f5c8) entered blocking state
[ 3351.613988] cni-podman0: port 3(vethad55f5c8) entered disabled state
[ 3351.620528] device vethad55f5c8 entered promiscuous mode
[ 3351.625896] cni-podman0: port 3(vethad55f5c8) entered blocking state
[ 3351.632257] cni-podman0: port 3(vethad55f5c8) entered forwarding state
5e645f5e40fc096ad0bea323a00bebebbda4bd825a5e8d12103f752d8868692e
```

図 9.22 コンテナを作成する実行例

コンテナに割り当てられた IP アドレスを確認するには podman inspect コマンドを実行します。

```
[armadillo ~]# podman inspect --format='{{.NetworkSettings.IPAddress}}' my_container_1
10.88.0.108
[armadillo ~]# podman inspect --format='{{.NetworkSettings.IPAddress}}' my_container_2
10.88.0.109
```

図 9.23 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 /bin/sh
[container ~]# ping -c 3 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
64 bytes from 10.88.0.109: seq=2 ttl=42 time=0.128 ms

--- 10.88.0.109 ping statistics ---
```

```
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.128/0.135/0.140 ms
```

図 9.24 ping コマンドによるコンテナ間の疎通確認実行例

このように、my_container_1(10.88.0.108) から my_container_2(10.88.0.109) への通信が確認できます。

9.1.2.12. 開発時に有用な--privileged オプション

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

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

9.1.3. アットマークテクノが提供するイメージを使う

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

9.1.3.1. Docker ファイルからイメージをビルドする

Armadillo-IoT ゲートウェイ G4 コンテナ [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/container] から「Debian [VERSION] サンプル Dockerfile」 ファイル (at-debian-image-dockerfile-[VERSION].tar.gz) をダウンロードします。その後 podman build コマンドを実行します。

```
[armadillo ~]# tar xzf at-debian-image-dockerfile-[VERSION].tar.gz
[armadillo ~]# cd at-debian-image-dockerfile-[VERSION]
[armadillo ~]# podman_switch_storage --disk
[armadillo ~]# podman build -t at-debian-image:latest .
:
:（省略）
:
[armadillo ~]# podman images
REPOSITORY                  TAG         IMAGE ID      CREATED             SIZE
localhost/at-debian-image   latest      c8e8d2d55456  About a minute ago  233 MB
docker.io/library/debian    bullseye    723b4a01cd2a  18 hours ago        123 MB
```

図 9.25 Docker ファイルによるイメージのビルドの実行例

podman images コマンドにより at-debian-image がビルドされたことが確認できます。library/debian イメージはベースとなっている Debian イメージです。

9.1.3.2. ビルド済みのイメージを使用する

Armadillo-IoT ゲートウェイ G4 コンテナ [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/container] から「Debian [VERSION] サンプルコンテナイメージ」 ファイル (at-debian-image-[VERSION].tar) をダウンロードします。その後 podman load コマンドを実行します。

```
[armadillo ~]# podman load -i at-debian-image-[VERSION].tar
:
:（省略）
:
[armadillo ~]# podman images
REPOSITORY                 TAG         IMAGE ID      CREATED       SIZE
localhost/at-debian-image  [VERSION]   93a4ec873ac5  17 hours ago  233 MB
localhost/at-debian-image  latest      93a4ec873ac5  17 hours ago  233 MB
```

図 9.26 ビルド済みイメージを load する実行例

podman images コマンドにより at-debian-image がビルドされたことが確認できます。

9.1.4. 入出力デバイスを扱う

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

9.1.4.1. GPIO を扱う

コンテナ内で動作するアプリケーションから GPIO を扱うためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/gpiochip**N** を渡す必要があります。以下は、/dev/gpiochip2 を渡して alpine イメージからコンテナを作成する例です。/dev/gpiochip**N** を渡すと、GPIO**N+1** を操作することができます。

```
[armadillo ~]# podman run -itd --name=gpio_example --device=/dev/gpiochip2 docker.io/alpine /bin/sh
```

図 9.27 GPIO を扱うためのコンテナ作成例

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

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

図 9.28 コンテナ内からコマンドで GPIO を操作する例

❶ GPIO 番号 21 の値を取得します。

❷ 取得した値を表示します。

❸ GPIO 番号 21 に 1(High) を設定します。

他にも、gpiodetect コマンドで認識している gpiochip をリスト表示できます。以下の例では、コンテナを作成する際に渡した /dev/gpiochip2 が認識されていることが確認できます。

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

図 9.29 gpiodetect コマンドの実行

gpioinfo コマンドでは、指定した gpiochip の詳細な情報を表示することができます。

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

図 9.30 gpioinfo コマンドの実行

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

9.1.4.2. I2C を扱う

コンテナ内で動作するアプリケーションから I2C を扱うためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/i2c-N を渡す必要があります。以下は、/dev/i2c-1 を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# podman run -itd --name=i2c_example --device=/dev/i2c-1 docker.io/alpine /bin/sh
```

図 9.31 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 /bin/sh
[container ~]# apk update && apk upgrade
[container ~]# apk add i2c-tools
[container ~]# i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- UU -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- UU -- -- --
50: UU -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- 72 -- -- -- -- --
```

図 9.32 i2cdetect コマンドによる確認例

自動起動の場合は devices の設定で i2c-1 を渡すことができます。

```
[armadillo ~]# vi /etc/atmark/containers/i2c_example.conf
image=localhost/i2c_example_image
add_devices "/dev/i2c-1"
set_command /root/do_something_with_i2c.sh
```

9.1.4.3. SPI を扱う

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

```
[armadillo ~]# podman run -itd --name=spi_example --device=/dev/spidev1.0 docker.io/alpine /bin/sh
```

図 9.33 SPI を扱うためのコンテナ作成例

コンテナ内に入り、spi-tools に含まれる spi-config コマンドを使って現在の設定を確認することができます。

```
[armadillo ~]# podman exec -it spi_example /bin/sh
[container ~]# apk update && apk upgrade
[container ~]# apk add spi-tools
```

```
[container ~]# spi-config --device=/dev/spidev1.0 -q
/dev/spidev1.0: mode=0, lsb=0, bits=8, speed=500000, spiready=0
```

図 9.34 spi-config コマンドによる確認例

9.1.4.4. CAN を扱う

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

```
[armadillo ~]# podman run -itd --name=can_example --net=host --cap-add=NET_ADMIN docker.io/alpine /
bin/sh
```

図 9.35 CAN を扱うためのコンテナ作成例

コンテナ内に入り、ip コマンドで CAN を有効にすることができます。以下に、設定例を示します。

```
[armadillo ~]# podman exec -it can_example /bin/sh
[container ~]# apk update && apk upgrade
[container ~]# apk add iproute2 ❶
[container ~]# ip link set can0 type can bitrate 125000 ❷
[container ~]# ip link set can0 up ❸
[container ~]# ip -s link show can0 ❹
4: can0: <NOARP,UP,LOWER_UP,ECHO> mtu 16 qdisc pfifo_fast state UP mode DEFAULT
group default qlen 10
    link/can
    RX: bytes  packets  errors  dropped missed  mcast
    0          0        0       0       0       0
    TX: bytes  packets  errors  dropped carrier collsns
    0          0        0       0       0       0
```

図 9.36 CAN の設定例

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

❷ CAN の通信速度を 125000 kbps に設定します。

❸ can0 インターフェースを起動します。

❹ can0 インターフェースの現在の使用状況を表示します。

9.1.4.5. PWM を扱う

コンテナ内で動作するアプリケーションから PWM を扱うためには、Podman のイメージからコンテナを作成する際にホスト OS 側の /sys ディレクトリを渡す必要があります。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 run -itd --name=pwm_example --volume=/sys:/sys docker.io/alpine /bin/sh
```

図 9.37 PWM を扱うためのコンテナ作成例

コンテナ内に入り、/sys/class/pwm/pwmchipN ディレクトリ内の export ファイルに 0 を書き込むことで扱えるようになります。以下に、/sys/class/pwm/pwmchip2 を扱う場合の動作設定例を示します。

```
[armadillo ~]# podman exec -it pwm_example /bin/sh
[container ~]# echo 0 > /sys/class/pwm/pwmchip2/export ❶
[container ~]# echo 1000000000 > /sys/class/pwm/pwmchip2/pwm0/period ❷
[container ~]# echo 500000000 > /sys/class/pwm/pwmchip2/pwm0/duty_cycle ❸
[container ~]# echo 1 > /sys/class/pwm/pwmchip2/pwm0/enable ❹
```

図 9.38 PWM の動作設定例

❶ pwmchip2 を export します。

❷ 周期を 1 秒にします。単位はナノ秒です。

❸ PWM の ON 時間 を 0.5 秒にします。

❹ PWM 出力を有効にします。

9.1.4.6. シリアルインターフェースを扱う

コンテナ内で動作するアプリケーションから RS-232C や RS-485 などのシリアル通信を行うためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/ttymxcN を渡す必要があります。以下は、/dev/ttymxc0 を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# podman run -itd --name=serial_example --device=/dev/ttymxc0 docker.io/alpine /bin/sh
```

図 9.39 シリアルインターフェースを扱うためのコンテナ作成例

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

```
[armadillo ~]# podman exec -it serial_example /bin/sh
[container ~]# setserial -a /dev/ttymxc0
/dev/ttymxc0, Line 0, UART: undefined, Port: 0x0000, IRQ: 29
        Baud_base: 5000000, close_delay: 50, divisor: 0
        closing_wait: 3000
        Flags: spd_normal
```

図 9.40 setserial コマンドによるシリアルインターフェイス設定の確認例

9.1.4.7. USB を扱う

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

・USB シリアルデバイスを扱う

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

```
[armadillo ~]# podman run -itd --name=usb_example --device=/dev/ttyUSB0 docker.io/alpine /bin/sh
```

図 9.41 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 /bin/sh
[container ~]# setserial -a /dev/ttyUSB0
/dev/ttyUSB0, Line 0, UART: unknown, Port: 0x0000, IRQ: 0
        Baud_base: 24000000, close_delay: 0, divisor: 0
        closing_wait: infinite
        Flags: spd_normal
```

図 9.42 setserial コマンドによる USB シリアルデバイス設定の確認例

・USB カメラを扱う

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

```
[armadillo ~]# podman run -itd --name=usbcam_example --device=/dev/video3 docker.io/alpine /bin/sh
[armadillo ~]# podman exec -it usbcam_example /bin/sh
[container ~]# ls /dev/video3
/dev/video3
```

図 9.43 USB カメラを扱うためのコンテナ作成例

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

・USB メモリを扱う

ここでは、USB メモリを扱う方法について 2 つの例を示します。

・ホスト OS 側でマウントした USB メモリをコンテナから扱う

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

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

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

図 9.44 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 run -itd --name=usbmem_example --volume=/mnt:/mnt docker.io/alpine /bin/sh
```

図 9.45 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 /bin/sh
[container ~]# ls /mnt
sample.txt
[container ~]# cat /mnt/sample.txt
test
```

図 9.46 USB メモリに保存されているデータの確認例

・ USB メモリをコンテナ内からマウントする

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

```
[armadillo ~]# podman run -itd --name=usbmem_example --cap-add=SYS_RAWIO --cap-add=SYS_ADMIN --
device=/dev/sda --device=/dev/sda1 docker.io/alpine /bin/sh
```

図 9.47 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 /bin/sh
[container ~]# mount /dev/sda1 /mnt
[container ~]# ls /mnt
sample.txt
[container ~]# cat /mnt/sample.txt
test
```

図 9.48 コンテナ内から USB メモリをマウントする例

・ USB デバイスのホットプラグに対応する

通常、コンテナ内から USB デバイスを扱うためには、あらかじめ Armadillo-IoT ゲートウェイ G4 本体に USB 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。コンテナ起動後に USB デバイスを接続して認識させるためには、/dev を volume としてコンテナ内にマウントする必要があります。以下は、 volume として /dev を渡して alpine イメージからコンテナを作成する例です。イベントの通知のために --net=host も渡す必要があります。

```
[armadillo ~]# podman run -itd --name=usbhotplug_example --volume=/dev:/dev --net=host docker.io/
alpine /bin/sh
```

図 9.49 USB デバイスのホットプラグに対応する例

9.1.4.8. RTC を扱う

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

```
[armadillo ~]# podman run -itd --name=rtc_example --cap-add=SYS_TIME --device=/dev/rtc0 docker.io/
alpine /bin/sh
```

図 9.50 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 /bin/sh
[container ~]# hwclock ❶
Thu Feb 18 05:14:37 2021  0.000000 seconds
[container ~]# date --set "2021-04-01 09:00:00" ❷
Thu Apr  1 09:00:00 UTC 2021
[container ~]# hwclock --systohc ❸
[container ~]# hwclock ❹
Thu Apr  1 09:00:28 2021  0.000000 seconds
```

図 9.51 hwclock コマンドによる RTC の時刻表示と設定例

❶ RTC に設定されている現在時刻を表示します。

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

❸ システム時刻を RTC に反映させます。

❹ RTC に設定されている時刻が変更されていることを確認します。

9.1.4.9. 音声出力を行う

Armadillo-IoT ゲートウェイ G4 に接続したスピーカーなどの音声出力デバイスへコンテナ内から音声を出力するためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/snd を渡す必要があります。以下は、/dev/snd を渡して debian イメージからコンテナを作成する例です。

```
[armadillo ~]# podman run -itd --name=snd_example --device=/dev/snd docker.io/debian /bin/bash
```

図 9.52 音声出力を行うためのコンテナ作成例

コンテナ内に入り、alsa-utils などのソフトウェアで音声出力を行えます。

```
[armadillo ~]# podman exec -it snd_example /bin/bash
[container ~]# apt update && apt upgrade
[container ~]# apt install alsa-utils ❶
[container ~]# /etc/init.d/alsa-utils start ❷
[container ~]# aplay -D hw:N,M [ファイル名] ❸
```

図 9.53 alsa-utils による音声出力を行う例

❶ alsa-utils をインストールします。

❷ alsa-utils を起動します。

❸ 指定したファイル名の音声ファイルを再生します。

aplay の引数にある、M は音声を出力したい CARD 番号、N はデバイス番号を表しています。CARD 番号とデバイス番号は、aplay コマンドに -l オプションを与えることで確認できます。

9.1.4.10. ユーザースイッチのイベントを取得する

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

```
[armadillo ~]# podman run -itd --name=sw_example --device=/dev/input docker.io/alpine /bin/sh
```

図 9.54 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 /bin/sh
[container ~]# apk update && apk upgrade
[container ~]# apk add evtest
[container ~]# evtest /dev/input/event1
Input driver version is 1.0.1
Input device ID: bus 0x19 vendor 0x1 product 0x1 version 0x100
Input device name: "gpio-keys"
Supported events:
  Event type 0 (EV_SYN)
  Event type 1 (EV_KEY)
    Event code 28 (KEY_ENTER)
Properties:
Testing ... (interrupt to exit)
Event: time 1612849227.554456, type 1 (EV_KEY), code 28 (KEY_ENTER), value 1 ❶
```

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

図 9.55 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

9.1.4.11. LED を扱う

Armadillo-IoT ゲートウェイ G4 には LED が実装されています。これらの LED を扱うためには、Podman のイメージからコンテナを作成する際にホスト OS 側の /sys ディレクトリを渡す必要があります。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 run -itd --name=led_example --volume=/sys:/sys docker.io/alpine /bin/sh
```

図 9.56 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 /bin/sh
[container ~]# echo 0 > /sys/class/leds/led1/brightness
[container ~]# echo 1 > /sys/class/leds/led1/brightness
```

図 9.57 LED の点灯/消灯の実行例

9.1.5. 近距離通信を行う

この章では、コンテナ内から近距離通信デバイスを扱う方法について示します。

9.1.5.1. Bluetooth デバイスを扱う

コンテナ内から Bluetooth デバイスを扱うためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/ttymxcN を渡すと同時にネットワークとして host を、権限として NET_ADMIN を渡す必要があります。/dev/ttymxcN は Bluetooth 通信で使用するように設定したシリアルデバイスを指定してください。以下は、/dev/ttymxc0 を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# podman run -itd --name=bt_example --net=host --device=/dev/ttymxc0 --cap-
add=NET_ADMIN docker.io/alpine /bin/sh
```

図 9.58 Bluetooth デバイスを扱うためのコンテナ作成例

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

```
[armadillo ~]# podman exec -it bt_example /bin/sh
[container ~]# apk update && apk upgrade
[container ~]# apk add bluez dbus
[container ~]# /usr/bin/dbus-daemon --system
[container ~]# /usr/lib/bluetooth/bluetoothd &
[container ~]# btattach -B /dev/ttymxc0 -S 115200 &
```

図 9.59 Bluetooth を起動する実行例

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

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

図 9.60 bluetoothctl コマンドによるスキャンとペアリングの例

❶ コントローラを起動します。

❷ 周辺機器をスキャンします。

❸ ペアリングしたい機器の MAC アドレスを指定してペアリングします。

❹ exit で bluetoothctl のプロンプトを終了します。

9.1.5.2. Wi-SUN デバイスを扱う

ここでは、Wi-SUN デバイスが UART で接続されている場合の例を示します。この場合、コンテナ内で動作するアプリケーションから Wi-SUN デバイスで通信を行うためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/ttymxcN のうち、Wi-SUN と対応するものを渡す必要があります。以下は、/dev/ttymxc0 を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# podman run -itd --name=wisun_example --device=/dev/ttymxc0 docker.io/alpine /bin/sh
[armadillo ~]# podman exec -it wisun_example /bin/sh
```

```
[container ~]# ls /dev/ttymxc0
/dev/ttymxc0
```

図 9.61 Wi-SUN デバイスを扱うためのコンテナ作成例

コンテナ内から、/dev/ttymxc0 を使って Wi-SUN データの送受信ができるようになります。

9.1.5.3. EnOcean デバイスを扱う

ここでは、EnOcean デバイスが UART で接続されている場合の例を示します。この場合、コンテナ内で動作するアプリケーションから EnOcean デバイスで通信を行うためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/ttymxcN のうち、EnOcean と対応するものを渡す必要があります。以下は、/dev/ttymxc0 を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# podman run -itd --name=enocean_example --device=/dev/ttymxc0 docker.io/alpine /bin/ ⏎
sh
[armadillo ~]# podman exec -it enocean_example /bin/sh
[container ~]# ls /dev/ttymxc0
/dev/ttymxc0
```

図 9.62 EnOcean デバイスを扱うためのコンテナ作成例

コンテナ内から、/dev/ttymxc0 を使って EnOcean データの送受信ができるようになります。

9.1.6. ネットワークを扱う

この章では、コンテナ内のネットワークを扱う方法について示します。

9.1.6.1. 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 ~]# podman run -itd --name=net_example docker.io/alpine /bin/sh
[armadillo ~]# podman inspect --format '{{ .NetworkSettings.IPAddress }}' net_example
10.88.0.17
```

図 9.63 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 run -itd --name=net_example docker.io/alpine /bin/sh
[armadillo ~]# podman exec -it net_example /sbin/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 9.64 ip コマンドを用いたコンテナの IP アドレス確認例

9.1.6.2. コンテナに固定 IP アドレスを設定する

podman はデフォルトで 10.88.0.0/16 を使います。

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

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

```
[armadillo ~]# podman network create --subnet=192.168.1.0/24 my_network
```

図 9.65 ユーザ定義のネットワーク作成例

コンテナを作成する際に、上記で作成したネットワークと設定したい IP アドレスを渡すことで、コンテナの IP アドレスを固定することができます。以下の例では、IP アドレスを 192.168.1.10 に固定します。

```
[armadillo ~]# podman run -itd --name=network_example --net=my_network --ip=192.168.1.10 docker.io/
alpine /bin/sh
```

図 9.66 IP アドレス固定のコンテナ作成例

コンテナの IP アドレスが、192.168.1.10 に設定されていることが確認できます。

```
[armadillo ~]# podman inspect --format '{{ .NetworkSettings.Networks.my_network.IPAddress }}'
network_example
192.168.1.10
```

図 9.67 コンテナの IP アドレス確認例

Armadillo-IoT ゲートウェイ G4 を再起動したときにネットワークの設定ファイルが消えてしまわないように、 /etc/atmark/containers に設定する必要があります

```
[armadillo ~]# vi /etc/atmark/containers/my_network.conf
type=network
subnet=192.168.1.0/24
[armadillo ~]# persist_file /etc/atmark/containers/my_network.conf
```

> persist_file コマンドに関する詳細は「9.8.2. overlayfs と persist_file について」を参照してください。

9.1.7. サーバを構築する

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

9.1.7.1. 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 ~]# podman run -itd --name=apache_example --publish=8080:80 docker.io/alpine /bin/sh
[armadillo ~]# podman exec -it apache_example /bin/sh
[container ~]# apk update && apk upgrade && apk add apache2
[container ~]# httpd
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using ⏎
10.88.0.2. Set the 'ServerName' directive globally to suppress this message
```

図 9.68 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 ~]# podman run -itd --name=lighttpd_example --publish=8080:80 docker.io/alpine /bin/sh
[armadillo ~]# podman exec -it lighttpd_example /bin/sh
[container ~]# apk update && apk upgrade && apk add lighttpd
[container ~]# echo "<html><body>It works!</body></html>" > /var/www/localhost/htdocs/index.html
[container ~]# lighttpd -f /etc/lighttpd/lighttpd.conf
```

図 9.69 コンテナに lighttpd をインストールする例

lighttpd はデフォルトでは動作確認用ページが用意されていないため、上記の手順では簡単なページを /var/www/localhost/htdocs ディレクトリの下に配置しています。他の PC などの Web ブラウザから、ホスト OS の IP アドレスの 8080 番ポートに接続すると表示されます。lighttpd の詳細な設定は、/etc/lighttpd ディレクトリにある設定ファイルを編集することで変更可能です。

9.1.7.2. 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 ~]# podman run -itd --name=ftp_example --publish=21:21 --
publish=21100-21110:21100-21110 --env=PASV_ADDRESS=<ホストの IP アドレス> docker.io/alpine /bin/sh
[armadillo ~]# podman exec -it ftp_example /bin/sh
[container ~]# apk update && apk upgrade && apk add vsftpd
```

図 9.70 コンテナに vsftpd をインストールする例

コンテナ内にユーザアカウントを作成し、このユーザで ftp ログインできるようにします。

```
[container ~]# adduser atmark
Changing password for atmark
New password: （パスワードを入力）
Retype password: （パスワードを入力）
passwd: password for atmark changed by root
```

図 9.71 ユーザを追加する例

作成したユーザで ftp ログインできるように、vsftpd の設定ファイルを編集します。

```
[container ~]# sed -i -e 's/anonymous_enable=YES/#anonymous_enable=YES/g' /etc/vsftpd/vsftpd.conf
[container ~]# sed -i -e 's/#local_enable=YES/local_enable=YES/g' /etc/vsftpd/vsftpd.conf
[container ~]# sed -i -e 's/#write_enable=YES/write_enable=YES/g' /etc/vsftpd/vsftpd.conf
[container ~]# echo "pasv_enable=YES" >> /etc/vsftpd/vsftpd.conf
[container ~]# echo "pasv_min_port=21100" >> /etc/vsftpd/vsftpd.conf
[container ~]# echo "pasv_max_port=21110" >> /etc/vsftpd/vsftpd.conf
[container ~]# echo "pasv_address=$PASV_ADDRESS" >> /etc/vsftpd/vsftpd.conf
```

図 9.72 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 9.73 vsftpd の起動例

9.1.7.3. 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 ~]# podman run -itd --name=smb_example --publish=139:139 --publish=445:445 docker.io/
alpine /bin/sh
```

```
[armadillo ~]# podman exec -it smb_example /bin/sh
[container ~]# apk update && apk upgrade && apk add samba
```

図 9.74 コンテナに samba をインストールする例

コンテナ内にユーザアカウントを作成し、このユーザで samba にログインできるようにします。

```
[container ~]# adduser atmark
Changing password for atmark
New password: （パスワードを入力）
Retype password: （パスワードを入力）
passwd: password for atmark changed by root
[container ~]# pdbedit -a atmark
new password: （パスワードを入力）
retype new password: （パスワードを入力）
```

図 9.75 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 9.76 samba の起動例

共有するディレクトリの指定などの詳細設定は /etc/samba/smb.conf ファイルを編集することで変更可能です。

9.1.7.4. 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 ~]# podman run -itd --name=sqlite_example docker.io/alpine /bin/sh
[armadillo ~]# podman exec -it sqlite_example /bin/sh
[container ~]# apk update && apk upgrade && apk add sqlite
```

図 9.77 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 mydb.sqlite
SQLite version 3.34.1 2021-01-20 14:10:07
Enter ".help" for usage hints.
sqlite>
```

図 9.78 sqlite の実行例

9.1.8. セキュリティ

この章では、コンテナ内におけるセキュリティの確保の方法について示します。

9.1.8.1. iptables コマンドを使用する

コンテナ内から、iptables コマンドを使用してパケットフィルタリングを行うためには、コンテナを作成する際に、権限として NET_ADMIN と NET_RAW を渡す必要があります。以下は、権限を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# podman run -itd --name=iptables_example --cap-add=NET_ADMIN --cap-add=NET_RAW
docker.io/alpine /bin/sh
[armadillo ~]# podman exec -it iptables_example /bin/sh
[container ~]# apk update && apk upgrade && apk add iptables
```

図 9.79 iptables を使用するためのコンテナ作成例

以下に、iptables を使用した例を示します。

```
[container ~]# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
[container ~]# iptables -I INPUT -p tcp -m tcp --dport 8080 -j ACCEPT
[container ~]# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:http-alt

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

図 9.80 iptables の動作確認例

9.1.9. 画面表示を行う

この章では、コンテナ内で動作するアプリケーションから Armadillo-IoT ゲートウェイ G4 に接続されたディスプレイに出力を行う方法について示します。

9.1.9.1. Wayland を扱う

コンテナ内から、Wayland のコンポジタである weston を起動し画面表示を行う例を示します。ここではアットマークテクノが提供するイメージからコンテナを作成します。このイメージに関しては 「9.1.3. アットマークテクノが提供するイメージを使う」 を参照してください。また、「8.3. VPU や NPU を使用する」 を実施済みであるとします。

```
[armadillo ~]# podman run -itd --name=wayland_example ¥
--env=XDG_RUNTIME_DIR=/tmp ¥ ❶
--env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu ¥ ❷
--device=/dev/dri ¥ ❸
--device=/dev/galcore ¥ ❹
--device=/dev/input ¥ ❺
--device=/dev/tty1 ¥ ❻
--volume=/run/udev:/run/udev:ro ¥ ❼
--volume=/opt/firmware:/opt/firmware:ro ¥ ❽
--cap-add=SYS_TTY_CONFIG ¥ ❾
localhost/at-debian-image:latest /bin/bash
```

図 9.81 Wayland を扱うためのコンテナ作成例

❶ weston の実行に必要な環境変数を設定します。

❷ 必要なライブラリをロードするためのパスを設定します。

❸ 画面描画に必要なデバイスを設定します。

❹ 画面描画に必要なデバイスを設定します。

❺ キーボードやマウスなどを使用可能にするためのデバイスを設定します。

❻ weston に必要な tty を設定します。

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

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

❾ tty を操作するための権限を設定します。

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

```
[armadillo ~]# podman attach wayland_example
[container ~]# weston --tty=1
Date: 2021-11-21 UTC
[23:46:52.823] weston 9.0.0
               https://wayland.freedesktop.org
               Bug reports to: https://gitlab.freedesktop.org/wayland/weston/issues/
               Build: lf-5.10.35-2.0.0-rc2-0-g230e9bc+
[23:46:52.825] Command line: weston --tty=1
[23:46:52.825] OS: Linux, 5.10.52-1-at, #2-Alpine SMP PREEMPT Thu Nov 18 09:10:13 UTC 2021, aarch64
[23:46:52.826] Using config file '/etc/xdg/weston/weston.ini'
[23:46:52.829] Output repaint window is 16 ms maximum.
[23:46:52.831] Loading module '/usr/lib/aarch64-linux-gnu/libweston-9/drm-backend.so'
[23:46:52.897] initializing drm backend
[23:46:52.897] logind: not running in a systemd session
[23:46:52.897] logind: cannot setup systemd-logind helper (-2), using legacy fallback
[23:46:52.902] using /dev/dri/card1
[23:46:52.902] DRM: supports atomic modesetting
[23:46:52.902] DRM: does not support GBM modifiers
[23:46:52.902] DRM: supports picture aspect ratio
[23:46:52.903] Loading module '/usr/lib/aarch64-linux-gnu/libweston-9/g2d-renderer.so'
[23:46:52.982] event1  - gpio-keys: is tagged by udev as: Keyboard
```

```
[23:46:52.983] event1  - gpio-keys: device is a keyboard
[23:46:52.986] event0  - audio-hdmi HDMI Jack: is tagged by udev as: Switch
[23:46:53.027] event0  - not using input device '/dev/input/event0'
[23:46:53.066] libinput: configuring device "gpio-keys".
[23:46:53.067] DRM: head 'LVDS-1' found, connector 39 is connected, EDID make 'unknown', model
'unknown', serial 'unknown'
[23:46:53.067] DRM: head 'HDMI-A-1' found, connector 40 is disconnected.
[23:46:53.067] Registered plugin API 'weston_drm_output_api_v1' of size 24
[23:46:53.067] Compositor capabilities:
               arbitrary surface rotation: yes
               screen capture uses y-flip: yes
               presentation clock: CLOCK_MONOTONIC, id 1
               presentation clock resolution: 0.000000001 s
[23:46:53.070] Loading module '/usr/lib/aarch64-linux-gnu/weston/desktop-shell.so'
[23:46:53.073] launching '/usr/libexec/weston-keyboard'
[23:46:53.079] Loading module '/usr/lib/aarch64-linux-gnu/libweston-9/xwayland.so'
[23:46:53.210] Registered plugin API 'weston_xwayland_v1' of size 32
[23:46:53.210] Registered plugin API 'weston_xwayland_surface_v1' of size 16
[23:46:53.210] xserver listening on display :0
[23:46:53.211] launching '/usr/libexec/weston-desktop-shell'
```

図 9.82 コンテナ内で weston を起動する実行例

Armadillo-IoT ゲートウェイ G4 に接続しているディスプレイ上に、デスクトップ画面が表示されます。

・ weston の設定

アットマークテクノが提供するイメージでは、weston の設定ファイルは /etc/xdg/weston/weston.ini に配置してあります。

```
[container ~]# cat /etc/xdg/weston/weston.ini
[core]
idle-time=0
use-g2d=1
xwayland=true
repaint-window=16

[shell]
panel-position=none

[output]
name=HDMI-A-1
#mode=1920x1080 ❶

[output]
name=LVDS-1
mode=off
```

図 9.83 weston.ini

❶ 解像度指定を行う場合はこの行のコメントを外して指定して下さい。

設定ファイルを更新するにはコンテナイメージを新しく保存することもできますが、ボリュームを使ってこのファイルだけを更新することができます。

podman run --volume か /etc/atmark/containers/*.conf の volumes 変数で設定してください。

・ weston の運用

コンテナの管理として、一つのコンテナで一つのアプリケーションを動かす事を推奨します。

一つのコンテナで weston を起動して、XDG_RUNTIME_DIR を共有することで別のコンテナで weston を使用するアプリケーションを起動させることは以下のコンフィグで可能です。

```
[armadillo ~]# vi /etc/atmark/containers/weston.conf ❶
image=localhost/at-debian-image:latest
add_devices /dev/dri /dev/galcore /dev/input /dev/tty1
add_volumes /run/udev:/run/udev:ro /opt/firmware:/opt/firmware:ro
add_volumes /tmp/xdg_home:/run/xdg_home
add_args --env=XDG_RUNTIME_DIR=/run/xdg_home ❷
add_args --env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu
add_args --cap-add=SYS_TTY_CONFIG
set_command weston --tty 1
[armadillo ~]# vi /etc/atmark/containers/detect_object.conf ❸
image=localhost/at-debian-image:latest
add_devices /dev/galcore /dev/video3
add_volumes /opt/firmware:/opt/firmware:ro /tmp/xdg_home:/run/xdg_home
restart=always ❹
add_args --env=XDG_RUNTIME_DIR=/run/xdg_home
add_args --env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu
set_command /root/start_detect_object.sh
[armadillo ~]# podman_start weston ❺
[armadillo ~]# podman_start detect_object ❻
```

❶ weston の設定ファイルを作成します。

❷ XDG_RUNTIME_DIR を volume で共有して、同じディレクトリを使います。

❸ 例として detect_object という名前のクライアントの設定ファイルを作成します。ここでは任意の名前を設定できます。

❹ アプリケーションによっては、weston が異常終了した時にエラーを出力しない場合があるため、restart=always にします。

❺ 確認のためコンテナを手動で起動します。

❻

・ ユーザを指定して weston を起動する

アットマークテクノが提供するイメージ at-debian-image にはデフォルトで atmark ユーザが存在しています。at-weston-launch コマンドを使うと、 root ユーザではなく atmark ユーザで weston を起動することができます。

```
[armadillo ~]# vi /etc/atmark/containers/weston.conf ❶
image=localhost/at-debian-image:latest
add_devices /dev/dri /dev/galcore /dev/input /dev/tty7 ❷
add_volumes /run/udev:/run/udev:ro /opt/firmware:/opt/firmware:ro
add_volumes /tmp/xdg_home:/run/xdg_home
add_args --env=XDG_RUNTIME_DIR=/run/xdg_home
add_args --env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu
add_args --cap-add=SYS_TTY_CONFIG
set_command at-weston-launch --tty /dev/tty7 --user atmark ❸
[armadillo ~]# podman_start weston ❹
```

❶ weston の設定ファイルを作成します。

❷ 使用する tty として /dev/tty7 を追加します。

❸ at-weston-launch コマンドのオプションとして使用する tty とユーザ名を渡します。

❹ 確認のためコンテナを手動で起動します。

--tty と --user を指定しなかった場合は、デフォルトで /dev/tty7 と atmark ユーザが使われます。

・スクリーンショットを保存する

weston を起動する際に、--debug オプションを渡すと weston-screenshooter コマンドでスクリーンショットを保存することができます。

```
[armadillo ~]# vi /etc/atmark/containers/weston.conf ❶
image=localhost/at-debian-image:latest
add_devices /dev/dri /dev/galcore /dev/input /dev/tty1
add_volumes /run/udev:/run/udev:ro /opt/firmware:/opt/firmware:ro
add_volumes /tmp/xdg_home:/run/xdg_home
add_args --env=XDG_RUNTIME_DIR=/run/xdg_home
add_args --env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu
add_args --cap-add=SYS_TTY_CONFIG
set_command weston --tty 1 --debug ❷
[armadillo ~]# podman_start weston ❸
[armadillo ~]# podman exec -it weston /bin/bash ❹
[container ~]# weston-screenshooter ❺
[container ~]# ls
wayland-screenshot-[date].png ❻
```

❶ weston の設定ファイルを作成します。

❷ --debug オプションを渡します。

❸ 確認のためコンテナを手動で起動します。

❹ 起動した weston コンテナ内で /bin/bash を起動してログインします。

❺ weston-screenshooter コマンドを実行します。

❻ カレントディレクトリ内に wayland-screenshot-[date].png というファイル名で保存されます。

--debug オプションは開発時にのみ使用してください。正式運用時の使用は非推奨です。

Armadillo-IoT ゲートウェイ G4 にキーボードを接続している場合は、--debug オプションを渡さなくても Windows キー + s を押下することによりスクリーンショットを保存することができます。この場合、スクリーンショットはコンテナ内の /proc/[weston の PID]/cwd 下に保存されます。

9.1.9.2. X Window System を扱う

コンテナ内から、X Window System を起動し画面表示を行う例を示します。ここではアットマークテクノが提供するイメージからコンテナを作成します。このイメージに関しては 「9.1.3. アットマークテクノが提供するイメージを使う」 を参照してください。また、「8.3. VPU や NPU を使用する」 を実施済みであるとします。

```
[armadillo ~]# podman run -itd --name=x_example ¥
--env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu ¥ ❶
--device=/dev/tty7 ¥ ❷
--device=/dev/fb0 ¥ ❸
--device=/dev/input ¥ ❹
--volume=/run/udev:/run/udev:ro ¥ ❺
--cap-add=SYS_ADMIN ¥ ❻
localhost/at-debian-image:latest /bin/bash
```

図 9.84 X Window System を扱うためのコンテナ起動例

❶ 必要なライブラリをロードするためのパスを設定します。

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

❸ 画面描画先となるフレームバッファを設定します。

❹ キーボードやマウスなどを使用可能にするためのデバイスを設定します。

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

❻ X Window System の動作に必要な権限を設定します。

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

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

X.Org X Server 1.20.11
```

```
X Protocol Version 11, Revision 0
Build Operating System: linux Debian
Current Operating System: Linux 25297ceb226c 5.10.52-1-at #2-Alpine SMP PREEMPT Thu Nov 18 09:10:13 ⏎
UTC 2021 aarch64
Kernel command line: console=ttymxc1,115200 root=/dev/mmcblk2p1 rootwait ro
Build Date: 13 April 2021  04:07:31PM
xorg-server 2:1.20.11-1 (https://www.debian.org/support)
Current version of pixman: 0.40.0
        Before reporting problems, check http://wiki.x.org
        to make sure that you have the latest version.
Markers: (--) probed, (**) from config file, (==) default setting,
        (++) from command line, (!!) notice, (II) informational,
        (WW) warning, (EE) error, (NI) not implemented, (??) unknown.
(==) Log file: "/var/log/Xorg.0.log", Time: Sun Nov 21 23:51:18 2021
(==) Using system config directory "/usr/share/X11/xorg.conf.d"
```

図 9.85 コンテナ内で X Window System を起動する実行例

Armadillo-IoT ゲートウェイ G4 に接続しているディスプレイ上に、デスクトップ画面が表示されます。

9.1.9.3. フレームバッファに直接描画する

コンテナ内で動作するアプリケーションからフレームバッファに直接描画するためには、Podman のイメージからコンテナを作成する際にホスト OS 側のデバイスファイル /dev/fbN を渡す必要があります。以下は、/dev/fb0 を渡して alpine イメージからコンテナを作成する例です。

```
[armadillo ~]# podman run -itd --name=fb_example --device=/dev/fb0 docker.io/alpine /bin/sh
```

図 9.86 フレームバッファに直接描画するためのコンテナ作成例

コンテナ内に入って、ランダムデータをフレームバッファに描画する例を以下に示します。これにより、接続しているディスプレイ上の表示が変化します。

```
[armadillo ~]# podman exec -it fb_example /bin/sh
[container ~]# cat /dev/urandom > /dev/fb0
cat: write error: No space left on device
```

図 9.87 フレームバッファに直接描画する実行例

9.1.9.4. タッチパネルを扱う

タッチパネルが組み込まれているディスプレイを接続している環境で、コンテナ内からタッチイベントを取得するためには、Podman のイメージからコンテナを作成する際にホスト OS 側の /dev/input を渡す必要があります。

```
[armadillo ~]# podman run -itd --name=touch_example --device=/dev/input docker.io/alpine /bin/sh
```

図 9.88 タッチパネルを扱うためのコンテナ作成例

Wayland などの GUI 環境と組み合わせて使うことで、タッチパネルを利用した GUI アプリケーションの操作が可能となります。

9.1.9.5. VPU を扱う

Armadillo-IoT ゲートウェイ G4 で採用している i.MX 8M Plus には、動画のエンコード/デコード処理に特化した演算ユニットである VPU (Video Processing Unit) が搭載されています。VPU を活用することでシステム全体のパフォーマンスを落とすことなく、動画のエンコード/デコード処理を行うことができます。コンテナ内で動作するアプリケーションから VPU を扱うためには、コンテナ作成時にデバイスとして、/dev/mxc_hantro と /dev/mxc_hantro_vc8000e および /dev/ion を渡す必要があります。ここではアットマークテクノが提供するイメージからコンテナを作成します。このイメージに関しては「9.1.3. アットマークテクノが提供するイメージを使う」 を参照してください。また、「8.3. VPU や NPU を使用する」 を実施済みであるとします。

```
[armadillo ~]# podman run -itd --name=vpu_example ¥
--device=/dev/mxc_hantro ¥
--device=/dev/mxc_hantro_vc8000e ¥
--device=/dev/ion ¥
localhost/at-debian-image:latest /bin/bash
[armadillo ~]# podman exec -it vpu_example /bin/bash
[container ~]# ls /dev/mxc_hantro /dev/mxc_hantro_vc8000e /dev/ion
/dev/ion  /dev/mxc_hantro  /dev/mxc_hantro_vc8000e
```

図 9.89 VPU を扱うためのコンテナ作成例

weston と GStreamer がインストール済みのイメージと組み合わせて使うことで、VPU を使用して動画のエンコード/デコードを行うことができます。

```
[armadillo ~]# podman run -itd --name=gst_example ¥
--env=XDG_RUNTIME_DIR=/tmp ¥
--env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu ¥
--device=/dev/dri ¥
--device=/dev/galcore ¥
--device=/dev/mxc_hantro ¥
--device=/dev/mxc_hantro_vc8000e ¥
--device=/dev/ion ¥
--device=/dev/input ¥
--device=/dev/tty1 ¥
--device=device=/dev/video3 ¥
--volume=/run/udev:/run/udev:ro ¥
--volume=/opt/firmware:/opt/firmware:ro ¥
--cap-add=SYS_TTY_CONFIG ¥
localhost/at-debian-image:latest /bin/bash
```

図 9.90 weston と GStreamer を扱うためのコンテナ作成例

このようにして作成したコンテナにログインすると、GStreamer で VPU を使用した動画のエンコード/デコードが行なえます。

```
[armadillo ~]# podman attach gst_example
[container ~]# apt install gstreamer1.0-imx libgstreamer-imx gstreamer1.0-plugins-bad ¥
libgstreamer-plugins-bad1.0-0 gstreamer1.0-plugins-base libgstreamer-plugins-base1.0-0 ¥
gstreamer1.0-plugins-good libgstreamer1.0-0 gstreamer1.0-tools gstreamer1.0-imx-tools
[container ~]# weston --tty=1 &
```

```
[container ~]# gst-launch-1.0 filesrc location=<ファイル名> ! qtdemux ! h264parse ! vpudec ! queue ! ⏎
waylandsink
```

図 9.91 GStreamer によるデコード実行例

USB カメラも組み合わせると、カメラからの映像をエンコードしてファイルに保存することも可能です。

```
[container ~]# gst-launch-1.0 -e v4l2src device=/dev/video3 ! video/x- ⏎
raw,width=640,height=480,framerate=30/1 ! queue ! vpuenc_h264 ! h264parse ! queue ! qtmux ! ⏎
filesink location=./output.mp4
```

図 9.92 GStreamer によるエンコード実行例

上記を実行することで、USB カメラからの映像が H.264 にエンコードされてファイルに保存されます。この例ではカメラデバイスを /dev/video3 としていますが、環境によって異なりますので適切なものを設定してください。

9.1.10. パワーマネジメント機能を使う

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

9.1.10.1. サスペンド状態にする

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

```
[armadillo ~]# podman run -itd --name=pm_example --volume=/sys:/sys docker.io/alpine /bin/sh
```

図 9.93 パワーマネジメント機能を使うためのコンテナ作成例

コンテナ内から、/sys/power/state に次の文字列を書き込むことにより、サスペンド状態にすることができます。

表 9.1 対応するパワーマネジメント状態

パワーマネジメント状態	文字列	説明
Suspend-to-RAM	mem	最も消費電力を抑えることができる
Suspend-to-Idle	freeze	最も短時間で復帰することができる

```
[armadillo ~]# podman run -itd --name=pm_example --volume=/sys:/sys docker.io/alpine /bin/sh
[armadillo ~]# podman exec -it pm_example /bin/sh
[container ~]# echo mem > /sys/power/state
```

図 9.94 サスペンド状態にする実行例

9.1.10.2. 起床要因を有効化する

サスペンド状態から起床要因として、利用可能なデバイスを以下に示します。

UART2（console）

起床要因　データ受信

有効化

```
[container ~]# echo enabled > /sys/bus/platform/drivers/imx-
uart/30890000.serial/tty/ttymxc1/power/wakeup
```

USB

起床要因　USB デバイスの挿抜

有効化

```
[container ~]# echo enabled > /sys/bus/platform/devices/
32f10108.usb/power/wakeup
```

RTC(i.MX8MP)

起床要因　アラーム割り込み

実行例

```
[armadillo ~]# podman run -v /sys:/sys --device /dev/rtc0 -
ti docker.io/alpine sh
[container ~]# apk add util-linux
[container ~]# rtcwake -m mem -s 5
:（省略）
[  572.720300] printk: Suspending console(s) (use
no_console_suspend to debug)
<ここで 5 秒を待つ>
[  573.010663] Disabling non-boot CPUs ...
...
```

図 9.95 サスペンド状態にする実行例、rtc で起こす

ユーザースイッチ

起床要因　ユーザースイッチ押下

有効化

```
[armadillo ~]# vi /boot/overlays.txt ❶
fdt_overlays=armadillo_iotg_g4-sw1-wakeup.dtbo

[armadillo ~]# persist_file -vp /boot/overlays.txt ❷
'/boot/overlays.txt' -> '/mnt/boot/overlays.txt'
Added "/boot/overlays.txt" to /etc/swupdate_preserve_files

[armadillo ~]# reboot ❸
:（省略）
Applying fdt overlay: armadillo_iotg_g4-sw1-wakeup.dtbo ❹
:（省略）

[armadillo ~]# cat /sys/devices/platform/gpio-keys/power/
wakeup ❺
enabled
```

❶ /boot/overlays.txt ファイルに「armadillo_iotg_g4-sw1-wakeup.dtbo」を追加します。ファイルが存在しない場合は

新規に作成してください。このファイルの詳細については「9.9.4. DTS overlays によるカスタマイズ」 を参照してください。

❷ /boot/overlays.txt を保存し、アップデートの場合でも保存します。

❸ overlay の実行のために再起動します。

❹ シリアルコンソールの場合に、u-boot によるメッセージを確認できます。

❺ Linux からも確認できます。

9.1.11. コンテナからの poweroff か reboot

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

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

```
[armadillo ~]# podman run --pid=host -ti docker.io/alpine sh
[container ~]# kill -USR2 1  (poweroff)
[container ~]# kill -TERM 1  (reboot)
```

図 9.96 コンテナから shutdown を行う

9.1.12. 異常検知

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

9.1.12.1. ソフトウェアウォッチドッグタイマーを扱う

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

```
[armadillo ~]# podman run -itd --name=watchdog_example --device=/dev/watchdog0 docker.io/alpine /
bin/sh
```

図 9.97 ソフトフェアウォッチドッグタイマーを使うためのコンテナ作成例

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

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

図 9.98 コンテナ内からソフトウェアウォッチドッグタイマーを起動する実行例

ソフトウェアウォッチドックタイマーを起動した後、/dev/watchdog0 に任意の文字を書き込むことでソフトウェアウォッチドッグタイマーをリセットすることができます。60 秒間任意の文字の書き込みがない場合は、システムが再起動します。

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

図 9.99 ソフトウェアウォッチドッグタイマーをリセットする実行例

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

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

図 9.100 ソフトウェアウォッチドッグタイマーを停止する実行例

9.1.13. NPU を扱う

Armadillo-IoT ゲートウェイ G4 で採用している i.MX 8M Plus には、機械学習に特化した演算処理ユニットである NPU (Neural Processor Unit) が搭載されています。NPU を活用することで、顔認識や物体認識などの推論処理を高速に行うことができます。

コンテナ内で動作するアプリケーションから NPU を扱うためには、 アットマークテクノが提供するコンテナイメージである at-debian-image を使用する必要があります。また、コンテナ作成時にデバイスとして、/dev/galcore を渡す必要があります。以下は、/dev/galcore を渡して at-debian-image からコンテナを作成する例です。このイメージに関しては 「9.1.3. アットマークテクノが提供するイメージを使う」 を参照してください。

```
[armadillo ~]# podman run -itd --name=npu_example ¥
--env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu ¥
--volume=/opt/firmware:/opt/firmware ¥
--device=/dev/galcore ¥
localhost/at-debian-image:latest /bin/sh
[armadillo ~]# podman exec -it npu_example /bin/sh
[container ~]# ls /dev/galcore
/dev/galcore
```

図 9.101 NPU を扱うためのコンテナ作成例

i.MX 8M Plus に搭載されている NPU は INT8 で量子化された学習済みモデルを高速に推論するように設計されています。INT8 で量子化されていないモデルの場合、正常に推論できない、または推論実行速度の低下が発生する場合があります。

具体的な機械学習アプリケーションの開発方法については、NXP Semiconductors の公式サイト[https://www.nxp.com/design/software/development-software/eiq-ml-development-environment:EIQ]を参照してください。

アットマークテクノからも機械学習に関する開発ガイドを公開していますので、そちらも参照してください。Armadillo Base OS 開発ガイド [https://armadillo.atmark-techno.com/resources/documents/armadillo-iot-g4/manuals]。

9.1.13.1. ONNX Runtime を使う

ONNX Runtime は 学習済みの ONNX モデルを使って推論を行うためのソフトウェアです。[1]Armadillo-IoT ゲートウェイ G4 では、NPU を使って ONNX Runtime を実行することができます。

・ ONNX Runtime をインストールする

at-debian-image から作成したコンテナであれば、apt install でインストールすることができます。

```
[armadillo ~]# podman run -it --name=onnxruntime_example ¥
--env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu ¥
--volume=/opt/firmware:/opt/firmware ¥
--device=/dev/galcore ¥
localhost/at-debian-image:latest /bin/bash
[container ~]# apt install onnxruntime onnxruntime-dev onnxruntime-tools python3-onnxruntime
```

図 9.102 ONNX Runtime をインストールする例

・ python から ONNX Runtime を使う

python から ONNX Runtime を使うためには onnxruntime モジュールを import します。また、NPU を使うために InferenceSession オブジェクトを作成する際に、Providers として「VsiNpuExecutionProvider」を指定します。

```
[container ~]# python3
>>> import onnxruntime
:（省略）
>>> sess = onnxruntime.InferenceSession('model.onnx', providers=['VsiNpuExecutionProvider'])
```

図 9.103 python から ONNX Runtime を使う例

以上により、python から ONNX Runtime を使うことができます。

9.1.13.2. TensorFlow Lite を使う

TensorFlow Lite からも NPU を使って高速に推論を行うことができます。

・ TensorFlow Lite をインストールする

at-debian-image から作成したコンテナであれば、apt install でインストールすることができます。

```
[armadillo ~]# podman run -it --name=tflite_example ¥
--env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu ¥
--volume=/opt/firmware:/opt/firmware ¥
--device=/dev/galcore ¥
```

[1]推論実行用のソフトウェアであり、学習は行なえません。

```
localhost/at-debian-image:latest /bin/bash
[container ~]# apt install tensorflow-lite tensorflow-lite-dev python3-tflite-runtime
```

図 9.104 TensorFlow Lite をインストールする例

・ python から TensorFlow Lite を使う

python から TensorFlow Lite を使うためには Interpreter モジュールを import します。python から TensorFlow Lite を使う場合は特別な設定をしなくても、自動的に NPU が使われます。

```
[container ~]# python3
>>> from tflite_runtime.interpreter import Interpreter
: (省略)
>>> interpreter = Interpreter('model.tflite')
```

図 9.105 python から TensorFlow Lite を使う例

以上により、python から TensorFlow Lite を使うことができます。

9.1.13.3. Arm NN を使う

Arm NN とは TensorFlow Lite および ONNX のモデル形式をサポートしている推論用ソフトウェアです。Arm NN からも NPU を使って高速に推論を行うことができます。

・ Arm NN をインストールする

at-debian-image から作成したコンテナであれば、apt install でインストールすることができます。

```
[armadillo ~]# podman run -it --name=armnn_example ¥
--env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu ¥
--volume=/opt/firmware:/opt/firmware ¥
--device=/dev/galcore ¥
localhost/at-debian-image:latest /bin/bash
[container ~]# apt install libarmnn22 libarmnn-dev python3-pyarmnn armnn-examples
```

図 9.106 Arm NN をインストールする例

・ python から Arm NN を使う

python から TensorFlow Lite を使うためには pyarmnn モジュールを import します。また、NPU を使うために BackendId として「VsiNpu」を指定して、Optimize オブジェクトを作成します。

```
[container ~]# python3
>>> import pyarmnn as ann
: (省略)
>>> options = ann.CreationOptions()
>>> runtime = ann.IRuntime(options)
>>> parser = ann.ITfLiteParser()
>>> network = parser.CreateNetworkFromBinaryFile('model.tflite')
>>> preferred_backends = []
>>> preferred_backends.append(ann.BackendId('VsiNpu'))
>>> opt_network, _ = ann.Optimize(network, preferred_backends, runtime.GetDeviceSpec(),
```

```
ann.OptimizerOptions())
>>> net_id, _ = runtime.LoadNetwork(opt_network)
```

図 9.107 python から Arm NN を使う例

以上により、python から Arm NN を使うことができます。

9.2. コンテナの運用

9.2.1. コンテナの自動起動

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

```
[armadillo ~]# cat /etc/atmark/containers/nginx.conf
image=docker.io/library/nginx:alpine
readonly=no
ports=”80:80”

armadillo:~# podman ps
CONTAINER ID  IMAGE                           COMMAND               CREATED         STATUS                  ⏎
PORTS               NAMES
d5e67bcdd743  docker.io/library/nginx:alpine  nginx -g daemon o...  About a minute ago  Up About a  ⏎
minute ago  0.0.0.0:80->80/tcp  nginx
```

図 9.108 コンテナを自動起動するための設定例

.conf ファイルは以下のパラメータを設定できます。

- コンテナイメージの選択： **image=[イメージ名]**

  イメージの名前を設定できます。

  例: image=docker.io/debian:latest, image=localhost/myimage

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

  設定したポートで外部からコンテナへのアクセスが可能となります。

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

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

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

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

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

  コンテナでデバイスを作成して、使用可能となります。

  コンテナパスを設定しない場合はホストと同じパスを使います。

  複数のデバイスを作成したい場合はスペースで分けて設定してください。

  例: add_devices /dev/galcore /dev/v4l/by-id/usb-046d_HD_Pro_Webcam_C920_78DA8CAF-video-index0:/dev/video3

  ホストパスに「:」を含む場合は add_device "[ホストパス]" "[コンテナパス]" で追加できます。

  例: add_device "/dev/v4l/by-path/XXXXX" "/dev/video3"

  コンテナパスに「:」を含むようなパスは設定できません。

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

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

  ホストパスは以下のどちらかを指定してください。

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

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

    SWUpdate でアップデートするデータに向いています。

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

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

    ログやデータベースに向いています。

  - /tmp/<folder>: 複数のコンテナでメモリファイルシステムを共有したい場合に使ってください。

  - /opt/firmware: 学習能力に必要なファムウェアライブラリーのパス。

  コンテナパスを設定しない場合はホストパスと同じパスを使います。

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

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

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

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

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

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

図 9.109 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。

- pod の選択： **pod=[ポッド名]**

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

  例: pod=mypod

- ネットワークの選択： **network=[ネットワーク名]**

  この設定に「9.2.3. network の作成」で作成したネットワーク以外に none と host の特殊な設定も選べます。

  none の場合、コンテナに localhost しかない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: network=mynetwork

- IP アドレスの設定： **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: ip=10.88.0.100

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

- 読み取り専用設定： **readonly=yes**

  コンテナ内からのファイルシステムへの書き込み許可を設定します。

  デフォルトで書き込み可能となっています。

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

- イメージの自動ダウンロード設定： **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：pull=missing か pull=always

- コンテナのリスタート設定： **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: restart=always か restart=no

- 自動起動の無効化： **autostart=no**

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

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

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

- 実行コマンドの設定： **set_command [コマンド]**

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

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

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

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

  * 例 * : add_args --cap-add=SYS_TTY_CONFIG --env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu

9.2.2. pod の作成

podman_start で pod 機能を使うことができます。

pod を使うことで、複数のコンテナが同じネットワークネームスペースを共有することができます。同じ pod 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
type=pod
add_ports "80:80"
infra_image=k8s.gcr.io/pause:3.5

[armadillo ~]# cat /etc/atmark/containers/nginx.conf
image=docker.io/library/nginx:alpine
readonly=no
pod=mypod

armadillo:~# podman ps
CONTAINER ID  IMAGE                            COMMAND               CREATED      STATUS                  ⏎
PORTS               NAMES
0cdb0597b610  k8s.gcr.io/pause:3.5                                   2 hours ago  Up 2 hours ago          ⏎
0.0.0.0:80->80/tcp  5ba7d996f673-infra
3292e5e714a2  docker.io/library/nginx:alpine  nginx -g daemon o...  2 hours ago  Up 2 hours ago          ⏎
0.0.0.0:80->80/tcp  nginx
```

図 9.110 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、 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 pod=[NAME] の設定を追加します。

ネットワークネームスペースは pod を作成するときに必要なため、 ports, network と ip の設定は pod のコンフィグファイルに入れなければなりません。

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

必要であれば、他の podman pod create のオプションを add_args で設定することができます。

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

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

9.2.3. network の作成

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

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

```
[armadillo ~]# cat /etc/atmark/containers/mynetwork.conf
type=network
subnet=192.168.100.0/24

[armadillo ~]# cat /etc/atmark/containers/nginx.conf
image=docker.io/library/nginx:alpine
ports=80:80
ip=192.168.100.10
network=mynetwork

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

図 9.111 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、 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 network=[NAME] の設定をいれます。

ネットワークのサブネットは subnet=[SUBNET] で設定します。

他の podman network create のオプションが必要であれば、 add_args で設定することができます。

9.2.4. コンテナからのコンテナ管理

podman では REST API による管理アクセスも可能です。

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

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

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

9.2.5. コンテナの配布

コンテナの作成は「9.1. アプリケーションをコンテナで実行する」を参考にしてください。

コンテナのイメージを配布する方法は大きく分けて二つあります：

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

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

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

アップデート時にアップデート前のコンテナが使われているのでご注意ください。

9.2.5.1. リモートリポジトリにコンテナを送信する方法

1. イメージをリモートリポジトリに送信する：

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

2. 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「9.7. 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]$ cat pull_container_nginx.desc
version=1

swdesc_pull_container "docker.io/nginx:alpine"
swdesc_files --extra-os nginx_start
[ATDE ~/mkswu]$ cp -r /usr/share/mkswu/examples/nginx_start .
[ATDE ~/mkswu]$ mkswu pull_container_nginx.desc
Enter pass phrase for /home/atmark/mkswu/swupdate.key:
pull_container_nginx.swu を作成しました。
```

9.2.5.2. イメージをファイルで保存する方法

1. イメージをファイルに保存する：

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

2. ファイルを SWUpdate のイメージに入れる。

   二つのやり方があります：

   a. swu イメージ内に組み込む

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

swdesc_embed_container "nginx_alpine.tar"
swdesc_files --extra-os nginx_start
[ATDE ~/mkswu]$ cp -r /usr/share/mkswu/examples/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]$ cat usb_container_nginx.desc
version=1

swdesc_usb_container "nginx_alpine.tar"
swdesc_files --extra-os nginx_start
[ATDE ~/mkswu]$ cp -r /usr/share/mkswu/examples/nginx_start .
[ATDE ~/mkswu]$ podman pull --arch arm64 docker.io/nginx:alpine
[ATDE ~/mkswu]$ podman run --rm docker.io/nginx:alpine uname -m
aarch64
[ATDE ~/mkswu]$ podman save docker.io/nginx:alpine > nginx_alpine.tar
[ATDE ~/mkswu]$ mkswu -o usb_container_nginx.swu usb_container_nginx.desc
Enter pass phrase for /home/atmark/mkswu/swupdate.key:
以下のファイルを USB メモリにコピーしてください：
'/home/atmark/mkswu/usb_container_nginx.swu'
'/home/atmark/mkswu/nginx_alpine.tar'
'/home/atmark/mkswu/.usb_container_nginx/nginx_alpine.tar.sig'

usb_container_nginx.swu を作成しました。
```

9.3. マルチメディアデータを扱う

9.3.1. GStreamer - マルチメディアフレームワーク

9.3.1.1. GStreamer - マルチメディアフレームワークとは

GStreamer は、オープンソースのマルチメディアフレームワークです。小さなコアライブラリに様々な機能をプラグインとして追加できるようになっており、多彩な形式のデータを扱うことができます。GStreamer で扱うことができるデータフォーマットの一例を下記に示します。

- コンテナフォーマット: mp4, avi, mpeg-ps/ts, mkv/webm, ogg
- 動画コーデック: H.264/AVC, VP8, VP9
- 音声コーデック: AAC, MP3, Theora, wav
- 画像フォーマット: JPEG, PNG, BMP
- ストリーミング: http, rtp

GStreamer では、マルチメディアデータをストリームとして扱います。ストリームを流すパイプラインの中に、エレメントと呼ばれる処理単位を格納し、それらを繋ぎ合わせることで、デコードやエンコードなどの処理を行います。

9.3.2. GStreamer 実行用コンテナを作成する

この章における GStreamer の実行例はアットマークテクノが提供する debian イメージから作成したコンテナ内で実行することを想定しています。「8.3. VPU や NPU を使用する」 を実施済みの環境で、以下のようにコンテナを作成します。

```
[armadillo ~]# podman run -it --name=gst-example ¥
--env=XDG_RUNTIME_DIR=/tmp ¥
--env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu ¥
--volume=/sys:/sys ¥
--volume=/dev:/dev ¥
--volume=/run/udev:/run/udev ¥
--volume=/opt/firmware:/opt/firmware ¥
--privileged ¥
localhost/at-debian-image:latest /bin/bash
[container /]#
```

図 9.112 GStreamer を実行するためのコンテナ作成例

コンテナ内では最初に GStreamer をインストールします。

```
[container /]# apt install gstreamer1.0-imx libgstreamer-imx gstreamer1.0-plugins-bad ¥
libgstreamer-plugins-bad1.0-0 gstreamer1.0-plugins-base libgstreamer-plugins-base1.0-0 ¥
gstreamer1.0-plugins-good libgstreamer1.0-0 gstreamer1.0-tools gstreamer1.0-imx-tools
```

図 9.113 gstreamer のインストール

次に、コンテナ内で画面表示を行うためのデスクトップ環境を起動します。ここでは weston を起動します。

```
[container /]# weston --tty=1 &
```

図 9.114 weston の起動

--tty=1 のオプションは画面表示に使用する tty の値を設定してください。

次に、音声を出力するのに必要な pulseaudio を起動します。

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

図 9.115 pulseaudio の起動

以上により、GSreamer をコンテナ内で実行できるようになります。

9.3.3. GStreamer パイプラインの実行例

パイプラインの実行例を以下に示します。

```
[container ~]# gst-launch-1.0 filesrc location=<ファイルパス> ¥
! qtdemux name=demux0 demux0.video_0 ! h264parse ! queue ! vpudec ! queue ¥
! waylandsink window-width=1920 window-height=1080 demux0.audio_0 ! queue ! beepdec ! autoaudiosink
```

図 9.116 GStreamer の実行例

GStreamer のパイプラインは、シェルスクリプトのパイプ構文の構造に似ています。GStreamer の各エレメントとシェルスクリプト内のコマンドを対比することができます。構文的な違いとして、GStreamer のパイプラインは「!」を使って各エレメントを繋ぎますが、シェルスクリプトは「|」を使います。

上記例は、GStreamer のデバッグ/プロトタイピング用のコマンドラインツールである gst-launch-1.0 を使って説明しましたが、GStreamer はライブラリとして提供されているため、GStreamer を使ったマルチメディア機能を自作のアプリケーションプログラムに組み込むことができます。API やアプリケーション開発マニュアルは、gstreamer.freedesktop.org の Documentation ページ(http://gstreamer.freedesktop.org/documentation/)から参照することができます。

Armadillo-IoT ゲートウェイ G4 が採用している SoC である i.MX 8M Plus は、動画のデコード/エンコードを行うための Video Processing Unit(VPU) と呼ばれる専用プロセッサを搭載しています。Armadillo-IoT ゲートウェイ G4 には、この VPU を使用するための GStreamer エレメントがインストールされており、以下の動画コーデックではメイン CPU のパフォーマンスを落とすことなく動画のデコード/エンコードが行なえます。

- デコード可能なコーデック
  - H.264/AVC
  - VP8
  - VP9
- エンコード可能なコーデック

・ H.264/AVC

以降の章では、これらのコーデックに対する GStreamer の実行例を紹介します。

上記で挙げたコーデック以外のものであってもデコード/エンコードは可能ですが、その場合は CPU を使ったソフトウェア処理となってしまうため、システム全体のパフォーマンスは低下します。

9.3.4. 動画を再生する

GStreamer を使用して動画を再生するための実行例を、音声を含んでいる動画と含んでいない動画の 2 通りについて示します。VPU でハードウェアデコードを行う GStreamer エレメントとして vpudec を使うことができます。

9.3.4.1. H.264/AVC 動画を再生する

```
[container ~]# gst-launch-1.0 filesrc location=<ファイルパス> ¥
! qtdemux name=demux0 demux0.video_0 ! h264parse ! queue ! vpudec ! queue ¥
! waylandsink window-width=1920 window-height=1080 demux0.audio_0 ! queue ! beepdec ! autoaudiosink
```

図 9.117 H.264/AVC 動画の再生(音声あり)

```
[container ~]# gst-launch-1.0 filesrc location=<ファイルパス> ¥
! qtdemux ! h264parse ! vpudec ! queue ! waylandsink window-width=1920 window-height=1080
```

図 9.118 H.264/AVC 動画の再生(音声なし)

9.3.4.2. VP8 動画を再生する

```
[container ~]# gst-launch-1.0 filesrc location=<ファイルパス> ¥
! matroskademux name=demux0 demux0.video_0 ! queue ! vpudec ! queue ¥
! waylandsink window-width=1920 window-height=1080 demux0.audio_0 ! queue ! beepdec ! autoaudiosink
```

図 9.119 VP8 動画の再生(音声あり)

```
[container ~]# gst-launch-1.0 filesrc location=<ファイルパス> ¥
! matroskademux ! vpudec ! queue ! waylandsink window-width=1920 window-height=1080
```

図 9.120 VP8 動画の再生(音声なし)

9.3.4.3. VP9 動画を再生する

```
[container ~]# gst-launch-1.0 filesrc location=<ファイルパス> ¥
! matroskademux name=demux0 demux0.video_0 ! queue ! vpudec ! queue ¥
! waylandsink window-width=1920 window-height=1080 demux0.audio_0 ! queue ! beepdec ! autoaudiosink
```

図 9.121 VP9 動画の再生(音声あり)

```
[container ~]# gst-launch-1.0 filesrc location=<ファイルパス> ¥
! matroskademux ! vpudec ! queue ! waylandsink window-width=1920 window-height=1080
```

図 9.122 VP9 動画の再生(音声なし)

9.3.5. ストリーミングデータを再生する

GStreamer を使用してネットワーク上にある動画ファイルを HTTP 及び RTSP でストリーミング再生する実行例を示します。VPU でハードウェアデコードを行う GStreamer エレメントとして vpudec を使うことができます。

9.3.5.1. HTTP ストリーミング

```
[container ~]# gst-launch-1.0 souphttpsrc location=<動画ファイルの URI> ¥
! qtdemux name=demux demux. ! queue ! vpudec ! queue ¥
! waylandsink demux. ! queue ! beepdec ! autoaudiosink
```

図 9.123 HTTP ストリーミングの再生(音声あり)

```
[container ~]# gst-launch-1.0 souphttpsrc location=<動画ファイルの URI> ¥
! qtdemux ! queue ! vpudec ! queue ! waylandsink
```

図 9.124 HTTP ストリーミングの再生(音声なし)

9.3.5.2. RTSP ストリーミング

```
[container ~]# gst-launch-1.0 rtspsrc location=<動画ファイルの URI> name=source ¥
! queue ! rtph264depay ! vpudec ! queue ! waylandsink source. ! queue ¥
! rtpmp4gdepay ! aacparse ! beepdec ! autoaudiosink
```

図 9.125 RTSP ストリーミングの再生(音声あり)

```
[container ~]# gst-launch-1.0 rtspsrc location=<動画ファイルの URI> ¥
! queue ! rtph264depay ! vpudec ! queue ! waylandsink
```

図 9.126 RTSP ストリーミングの再生(音声なし)

9.3.6. USB カメラからの映像を表示する

GStreamer の v4l2src エレメントを使うことで、V4L2(Video for Linux 2) デバイスとして実装されているカメラデバイスから映像を取得できます。どのデバイスから映像を取得するかは、v4l2src エレメントの device プロパティにデバイスファイル名を指定することで変更できます。UVC 対応 USB カメラなども同様に v4l2src で扱うことができるので、ここでは USB カメラからの映像を表示する実行例を示します。

加えて、カメラの他にマイクも接続していて、同時にマイクからの音声も出力する場合の例も示しています。実行例中のデバイスファイル /dev/video1 の部分や、縦横サイズである width や height の値

は実行する環境によって異なる可能性がありますので、適宜変更してください。また、/dev/v4l/by-id ディレクトリの下に、接続しているカメラ名の付いた /dev/videoN へのシンボリックリンクがありますので、デバイスとしてそれを指定することも可能です。

```
[container ~]# gst-launch-1.0 v4l2src device=/dev/video1 ¥
! video/x-raw,width=640,height=480,framerate=30/1 ¥
! waylandsink window-width=640 window-height=480 pulsesrc ¥
! audio/x-raw,rate=44100,channels=2 ! autoaudiosink
```

図 9.127 USB カメラからの映像表示(音声あり)

```
[container ~]# gst-launch-1.0 v4l2src device=/dev/video1 ¥
! video/x-raw,width=640,height=480,framerate=30/1 ¥
! waylandsink window-width=640 window-height=480
```

図 9.128 USB カメラからの映像表示(音声なし)

9.3.7. USB カメラからの映像を録画する

GStreamer の v4l2src エレメントを使うことで、V4L2(Video for Linux 2) デバイスとして実装されているカメラデバイスから映像を取得できます。どのデバイスから映像を取得するかは、v4l2src エレメントの device プロパティにデバイスファイル名を指定することで変更できます。UVC 対応 USB カメラなども同様に v4l2src で扱うことができるので、ここでは USB カメラからの映像をファイルへ保存する実行例と、映像を表示しながら同時にファイルへ保存する実行例を示します。

加えて、カメラの他にマイクも接続していて、映像の保存と同時にマイクからの音声も MP3 へエンコードして保存する場合の例も示しています。実行例中のデバイスファイル /dev/video1 の部分や、縦横サイズである width や height の値は実行する環境によって異なる可能性がありますので、適宜変更してください。また、/dev/v4l/by-id ディレクトリの下に、接続しているカメラ名の付いた /dev/videoN へのシンボリックリンクがありますので、デバイスとしてそれを指定することも可能です。

パイプライン停止時に EOS イベントを発行するように、gst-launch-1.0 コマンドに-e オプションを付けています。エンコードを終了するには、Ctrl-C で gst-launch-1.0 コマンドを停止してください。

9.3.7.1. H.264/AVC で録画する

VPU でハードウェアエンコードを行う GStreamer エレメントとして vpuenc_h264 を使うことができます。

```
[container ~]# gst-launch-1.0 -e v4l2src device=/dev/video1 ¥
! video/x-raw,width=640,height=480,framerate=30/1 ¥
! queue ! vpuenc_h264 ! h264parse ! queue ! mux. pulsesrc ¥
! audio/x-raw,rate=44100,channels=2 ! lamemp3enc ! queue ¥
! mux. qtmux name=mux ! filesink location=./output.mp4
```

図 9.129 USB カメラからの映像を H.264 で録画(音声あり)

```
[container ~]# gst-launch-1.0 -e v4l2src device=/dev/video1 ¥
! video/x-raw,width=640,height=480,framerate=30/1 ¥
```

```
! queue ! vpuenc_h264 ! h264parse ! queue ¥
! filesink location=./output.mp4
```

図 9.130 USB カメラからの映像を H.264 で録画(音声なし)

・ 表示と録画を同時に行う

```
[container ~]# gst-launch-1.0 -e v4l2src device=/dev/video1 ¥
! video/x-raw,width=640,height=480,framerate=30/1 ¥
! tee name=t1 ! queue ! vpuenc_h264 ! h264parse ! queue ! mux. pulsesrc ¥
! tee name=t2 ! audio/x-raw,rate=44100,channels=2 ! lamemp3enc ! queue ¥
! mux. qtmux name=mux ! filesink location=./output.mp4 t1. ! queue ¥
! waylandsink window-width=640 window-height=480 t2. ! queue ! autoaudiosink
```

図 9.131 USB カメラからの映像を表示しながら H.264 で録画(音声あり)

```
[container ~]# gst-launch-1.0 -e v4l2src device=/dev/video1 ¥
! video/x-raw,width=640,height=480,framerate=30/1 ¥
! tee name=t1 ! queue ! vpuenc_h264 ! h264parse ! queue ¥
! qtmux ! filesink location=./output.mp4 t1. ! queue ¥
! waylandsink window-width=640 window-height=480
```

図 9.132 USB カメラからの映像を表示しながら H.264 で録画(音声なし)

9.3.8. Video Processing Unit(VPU)

9.3.8.1. Video Processing Unit とは

Video Processing Unit(以下、VPU) とは i.MX 8M Plus に搭載されている、動画のエンコード/デコード処理専用のプロセッサです。動画のエンコード/デコード処理は、システムに負荷をかけることが多く、メイン CPU で処理を行うとシステム全体のパフォーマンスが低下します。VPU を利用することでシステム全体のパフォーマンスを落とすことなく、動画のエンコード/デコード処理を行うことができます。

VPU が対応しているフォーマットは以下の通りです。

- デコーダーが対応しているフォーマット
  - H.264/AVC
  - VP8
  - VP9
- エンコーダが対応しているフォーマット
  - H.264/AVC

9.3.8.2. VPU の仕様

- H.264/AVC デコーダー

表 9.2 H.264/AVC デコーダー仕様

Profile	High、Main、Baseline
Min resolution	48x48
Max resolution	1920x1080
Frame rate	60 fps
Bitrate	60 Mbps

・ VP8 デコーダー

表 9.3 VP8 デコーダー仕様

Profile	-
Min resolution	48x48
Max resolution	1920x1080
Frame rate	60 fps
Bitrate	60 Mbps

・ VP9 デコーダー

表 9.4 VP9 デコーダー仕様

Profile	Profile 0, 2
Min resolution	72x72
Max resolution	1920x1080
Frame rate	60 fps
Bitrate	100 Mbps

・ H.264/AVC エンコーダー

表 9.5 H.264/AVC エンコーダー仕様

Profiles	Baseline、Main、High、High 10
Maximum Luma pixel sample rate	1920x1080 @ 60 fps
Slices	I, P and B slices
Frame Types	Progressive
Entropy encoding	CABAC、CAVLC
Error resilience	Slices
Maximum MV range	Horizontal (P slice) in pixels: +/-139 Horizontal (B slice) in pixels: +/-75 Vertical (P or B slice) in pixels: ・ Config1: +/-13 (planned) ・ Config2: +/-21 ・ Config3: +/-29 (planned) ・ Config4: +/-45 (planned) ・ Config5: +/-61 (planned) (= Search Window Size -3 pixels)
MV accuracy	1/4 pixel
Supported block sizes	Macroblock and sub-macroblock partitions: ・ Intra PU: 16x16 / 8x8 / 4x4 ・ Inter PU: 16x16 / 8x16 / 16x8 ・ TU: 4x4 and 8x8 transforms

Intra-prediction modes	16x16: 4 modes 8x8: 9 modes 4x4: 9 modes
Maximum number of reference frames	2
Encoding picture type	Only progressive frame
IPCM encoding	Supported
Temporal scalable video coding	Up to 5 layers including the base layer
IPCM	IPCM rectangle mode
ROI / ROI_map	Absolute QP and qpoffset mode (-32 〜 31) User controllable CU coded as IPCM CU or skip CU

9.4. Armadillo のソフトウェアをビルドする

ここでは、Armadillo-IoT ゲートウェイ G4 で使用するソフトウェアのビルド方法を説明します。

9.4.1. ブートローダーをビルドする

ここでは、Armadillo-IoT ゲートウェイ G4 向けのブートローダーイメージをビルドする方法を説明します。

1. ブートローダーのビルドに必要なパッケージのインストール

   次のコマンドを実行します。

```
[PC ~]$ sudo apt install build-essential git wget gcc-aarch64-linux-gnu libgcc-*-dev-arm64-cross bison flex zlib1g-dev bash python3-pycryptodome python3-pyelftools device-tree-compiler
```

2. ソースコードの取得

   Armadillo-IoT ゲートウェイ G4 ブートローダー [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/boot-loader] から「ブートローダー ソース」ファイル (imx-boot-[VERSION].tar.gz) をダウンロードして、次のコマンドを実行します。

```
[PC ~]$ tar xf imx-boot-[VERSION].tar.gz
[PC ~]$ cd imx-boot-[VERSION]
```

3. ビルド

   次のコマンドを実行します。

```
[PC ~/imx-boot-[VERSION]]$ make imx-boot_armadillo_x2
:
:（省略）
:
Second Loader IMAGE:
 sld_header_off         0x58000
 sld_csf_off            0x59020
 sld hab block:         0x401fcdc0 0x58000 0x1020
make[1]: ディレクトリ '/home/atmark/imx-boot-[VERSION]/imx-mkimage' から出ます
cp imx-mkimage/iMX8M/flash.bin imx-boot_armadillo_x2
```

初めてのビルドの場合、i.MX 8M Plus に必要なファームウェアの EULA への同意を求められます。内容を確認の上、同意してご利用ください。[2]

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

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

LA_OPT_NXP_Software_License v19 February 2021
:
: (省略)
:
Do you accept the EULA you just read? (y/N)
```

4. ビルド結果の確認

   次のコマンドを実行します。

```
[PC ~/imx-boot-[VERSION]]$ ls imx-boot_armadillo_x2
imx-boot_armadillo_x2
```

9.4.2. Linux カーネルをビルドする

ここでは、Armadillo-IoT ゲートウェイ G4 向けの Linux カーネルイメージをビルドする方法を説明します。

Armadillo-IoT ゲートウェイ G4 では、基本的には Linux カーネルイメージをビルドする必要はありません。「9.4.3. 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には、「9.4.3. Alpine Linux ルートファイルシステムをビルドする」の手順の中で、ax2/packages から linux-at を削除し、ax2/resources/boot/ にイメージを配置する必要があります。

1. Linux カーネルのビルドに必要なパッケージのインストール

   次のコマンドを実行します。

```
[PC ~]$ sudo apt install crossbuild-essential-arm64 bison flex python3-pycryptodome python3-
pyelftools zlib1g-dev libssl-dev bc firmware-misc-nonfree firmware-libertas firmware-
atheros wireless-regdb
```

2. ソースコードの取得

[2] スペースキーでページを送ると、最終ページに同意するかどうかの入力プロンプトが表示されます。

Armadillo-IoT ゲートウェイ G4 Linux カーネル [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/linux-kernel] から「Linux カーネル」ファイル (linux-at-[VERSION].tar) をダウンロードして、次のコマンドを実行します。

```
[PC ~]$ tar xf linux-at-[VERSION].tar
[PC ~]$ tar xf linux-at-[VERSION]/linux-[VERSION].tar.gz
[PC ~]$ cd linux-[VERSION]
```

3. デフォルトコンフィギュレーションの適用

   次のコマンドを実行します。

```
[PC ~/linux-[VERSION]]$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- x2_defconfig
```

4. カーネルコンフィギュレーションの変更

   次のコマンドを実行します。カーネルコンフィギュレーションの変更を行わない場合はこの手順は不要です。

```
[PC ~]$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- menuconfig
```

   コマンドを実行するとカーネルコンフィギュレーション設定画面が表示されます。カーネルコンフィギュレーションを変更後、"Exit"を選択して「Do you wish to save your new kernel configuration? (Press <ESC><ESC> to continue kernel configuration.)」で"Yes"とし、カーネルコンフィギュレーションを確定します。

```
.config - Linux/arm64 5.10.86 Kernel Configuration
──────────────────────────────────────────────────────────────────────────────
  ┌──────────────────── Linux/arm64 5.10.86 Kernel Configuration ─────────────────────┐
  │  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus  │
  │  ----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M>│
  │  modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.   │
  │  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable               │
  │  ┌──────────────────────────────────────────────────────────────────────────────┐ │
  │  │          General setup  --->                                                 │ │
  │  │      [*] Support DMA zone                                                    │ │
  │  │      [*] Support DMA32 zone                                                  │ │
  │  │          Platform selection  --->                                            │ │
  │  │          Kernel Features  --->                                               │ │
  │  │          Boot options  --->                                                  │ │
  │  │          Power management options  --->                                      │ │
  │  │          CPU Power Management  --->                                          │ │
  │  │          Firmware Drivers  --->                                              │ │
  │  │      [ ] Virtualization  ----                                                │ │
  │  │      -*- ARM64 Accelerated Cryptographic Algorithms  --->                    │ │
  │  │          General architecture-dependent options  --->                        │ │
  │  │      [*] Enable loadable module support  --->                                │ │
  │  │      [*] Enable the block layer  --->                                        │ │
  │  │          IO Schedulers  --->                                                 │ │
  │  │          Executable file formats  --->                                       │ │
  │  │          Memory Management options  --->                                     │ │
  │  │      [*] Networking support  --->                                            │ │
```

```
|  |          Device Drivers  --->                                          |  |
|  |          File systems  --->                                            |  |
|  |          Security options  --->                                        |  |
|  |      -*- Cryptographic API  --->                                       |  |
|  |          Library routines  --->                                        |  |
|  |          Kernel hacking  --->                                          |  |
|  |                                                                        |  |
|  └────────────────────────────────────────────────────────────────────────┘  |
├──────────────────────────────────────────────────────────────────────────────┤
|           <Select>    < Exit >    < Help >    < Save >    < Load >          |
└──────────────────────────────────────────────────────────────────────────────┘
```

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

5. ビルド

   次のコマンドを実行します。

```
[PC ~/linux-[VERSION]]$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- -j5
```

6. ビルド結果の確認

   次のコマンドを実行します。

```
[PC ~/linux-[VERSION]]$ ls arch/arm64/boot/Image
arch/arm64/boot/Image
[PC ~/linux-[VERSION]]$ ls arch/arm64/boot/dts/freescale/armadillo_*.dtb
arch/arm64/boot/dts/freescale/armadillo_iotg_g4-nousb.dtb
arch/arm64/boot/dts/freescale/armadillo_iotg_g4.dtb
```

9.4.3. Alpine Linux ルートファイルシステムをビルドする

ここでは、alpine/build-rootfs を使って、Alpine Linux ルートファイルシステムを構築する方法を説明します。

alpine/build-rootfs は PC で動作している Linux 上で Armadillo-IoT ゲートウェイ G4 用の aarch64 アーキテクチャに対応した Alpine Linux ルートファイルシステムを構築することができるツールです。

9.4.3.1. デフォルトの Alpine Linux ルートファイルシステムを構築する

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。

```
[PC ~]$ sudo apt install podman
```

2. alpine/build-rootfs の入手

   Armadillo-IoT ゲートウェイ G4 開発用ツール [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/tools] から「Alpine Linux ルートファイルシステムビルドツール」 ファイル (build-rootfs-[VERSION].tar.gz) をダウンロードして、次のコマンドを実行します。

```
[PC ~/]$ wget https://download.atmark-techno.com/armadillo-iot-g4/tool/build-rootfs-
latest.tar.gz
[PC ~/]$ tar xf build-rootfs-latest.tar.gz
[PC ~/]$ cd build-rootfs-[VERSION]
```

3. 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。

```
[PC ~/build-rootfs-[VERSION]]$ sudo ./build_rootfs.sh
use default(board=ax2)
use default(arch=aarch64)
use default(outdir=/home/xxxx/at-optee-build/build-rootfs)
use default(output=baseos-x2-ATVERSION.tar.zst)
'repositories' -> '/etc/apk/repositories'
:
: (略)
:
> Creating rootfs archive
-rw-r--r--    1 root     root     231700480 Nov 26 07:18 rootfs.tar
ERROR: No such package: .make-alpine-make-rootfs
================================================
footprint[byte]  tarball[byte]  packages
      229904000       74942331  alpine-base coreutils chrony ...(省略)
================================================
done.
```

ビルド時のログにエラー"ERROR: No such package: .make-alpine-make-rootfs"が出ていますが、正常時でも出力されるメッセージのため、問題はありません。

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。

```
[PC ~/build-rootfs-[VERSION]]$ ./build_rootfs.sh ~/alpine.tar.gz
:
: (略)
:
[PC ~/build-rootfs-[VERSION]]$ ls ~/alpine.tar.gz
~/alpine.tar.gz
```

4. ビルド結果の確認

   次のコマンドを実行します。

```
[PC ~/build-rootfs-[VERSION]]$ ls *tar.zst
baseos-x2-[VERSION].tar.zst
```

9.4.3.2. Alpine Linux ルートファイルシステムをカスタマイズする

alpine/build-rootfs ディレクトリ直下にある ax2 ディレクトリ以下のファイルを変更し、build.sh を実行することで、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。

- install
  - resources/ ディレクトリ内のファイルを、 ルートファイルシステムにインストールするためのスクリプト
- resources/
  - ルートファイルシステムにインストールするファイルを含んだディレクトリ
- packages
  - ルートファイルシステムにインストールするパッケージのリスト
- fixup
  - パッケージのインストールや上記 install スクリプトの後に実行されるスクリプト
- ファイル/ディレクトリを追加する

ax2/resources/ 以下に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の直下にコピーされます。デフォルトでは、UID と GID は共に root、パーミッションは 0744(ディレクトリの場合は 0755)となります。

ax2/install を修正することで、ファイルの UID、GID、パーミッションを変更することができます。UID、GID を変更する場合は chown、パーミッションを変更する場合は chmod を利用してください。

- パッケージを変更する

ax2/packages を変更することで、ルートファイルシステムにインストールするパッケージをカスタマイズすることができます。

パッケージ名は 1 行に 1 つ書くことができます。パッケージ名は Armadillo 上で"apk add"の引数に与えることのできる正しい名前で記載してください。誤ったパッケージ名を指定した場合は、ルートファイルシステムのビルドに失敗します。

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 Packageshttps://pkgs.alpinelinux.org/packages

Alpine Linux ルートファイルシステム使用した Armadillo で検索することもできます。

```
[armadillo ~]# apk list *ruby*
ruby-rmagick-4.1.2-r1 armhf {ruby-rmagick} (MIT)
ruby-concurrent-ruby-ext-1.1.6-r1 armhf {ruby-concurrent-ruby} (MIT)
ruby-net-telnet-2.7.2-r3 armhf {ruby} (Ruby AND BSD-2-Clause AND MIT)
:
:（省略）
:
ruby-mustache-1.1.1-r3 armhf {ruby-mustache} (MIT)
ruby-nokogiri-1.10.10-r0 armhf {ruby-nokogiri} (MIT)
```

9.5. SD ブートの活用

本章では、microSD カードから直接起動(以降「SD ブート」と表記します)する手順を示します。SD ブートを活用すると、microSD カードを取り替えることでシステムイメージを変更することができます。本章に示す手順を実行するためには、容量が 8Gbyte 以上の microSD カードを必要とします。

SD ブートを行った場合、ブートローダーの設定は **microSD カード** に保存されます。

9.5.1. ブートディスクの作成

1. ブートディスクイメージのビルドします

   「9.4.3. Alpine Linux ルートファイルシステムをビルドする」 で説明されているソースツリー alpine/build-rootfs にあるスクリプト `build_image` と 「9.4.1. ブートローダーをビルドする」でビルドした `imx-boot_armadillo_x2` を利用します。

   VPU や NPU も使用する場合は、 「8.3. VPU や NPU を使用する」 で用意した imx_lib.img も組み込めます。

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

2. ATDE に microSD カードを接続します。詳しくは「4.2.2. 取り外し可能デバイスの使用」を参考にしてください。

3. microSD カードのデバイス名を確認します

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

4. microSD カードがマウントされている場合、アンマウントします。

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

図 9.133 自動マウントされた microSD カードのアンマウント

5. ブートディスクイメージの書き込み

```
[PC ~]$ sudo dd if=~/build-rootfs-[VERSION]/baseos-x2-[VERSION].img ¥
                of=/dev/sdb bs=1M oflag=direct status=progress
```

microSD カードの性能にもよりますが、書き込みには 5 分程度かかります。

microSD カードのパーティション構成は次のようになっています。

表 9.6 microSD カードのパーティション構成

パーティション	オフセット	サイズ	説明
-	0	10MiB	ブートローダー
1	10MiB	300MiB	A/B アップデートの A 面パーティション
2	310MiB	300MiB	A/B アップデートの B 面パーティション
3	610MiB	50MiB	ログ用パーティション
4	660MiB	200MiB	ファームウェア
5	860MiB	残り	アプリケーション用パーティション

gdisk で確認すると次のようになります。

```
[PC ~]$ sudo gdisk -l /dev/mmcblk0
GPT fdisk (gdisk) version 1.0.8
```

```
Partition table scan:
  MBR: protective
  BSD: not present
  APM: not present
  GPT: present

Found valid GPT with protective MBR; using GPT.
Disk /dev/mmcblk0: 15319040 sectors, 7.3 GiB
Sector size (logical/physical): 512/512 bytes
Disk identifier (GUID): 309AD967-470D-4FB2-835E-7963578102A4
Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
First usable sector is 34, last usable sector is 15319006
Partitions will be aligned on 2048-sector boundaries
Total free space is 20446 sectors (10.0 MiB)

Number  Start (sector)    End (sector)  Size       Code  Name
   1           20480          634879   300.0 MiB   8300  rootfs_0
   2          634880         1249279   300.0 MiB   8300  rootfs_1
   3         1249280         1351679   50.0 MiB    8300  logs
   4         1351680         1761279   200.0 MiB   8300  firm
   5         1761280        15319006   6.5 GiB     8300  app
```

9.5.2. SD ブートの実行

「9.5.1. ブートディスクの作成」で作成したブートディスクから起動する方法を説明します。

1. Armadillo-IoT ゲートウェイ G4 に電源を投入する前に、ブートディスクを CON1(SD インターフェース)に挿入します。また、JP1 ジャンパーをショート(SD ブートに設定)します。
2. 電源を投入します。

```
U-Boot SPL 2020.04-at1 (Dec 09 2021 - 11:17:22 +0900)
rv8803 rtc woken by interrupt
DDRINFO: start DRAM init
DDRINFO: DRAM rate 4000MTS
DDRINFO:ddrphy calibration done
DDRINFO: ddrmix config done
Normal Boot
Trying to boot from BOOTROM
image offset 0x8000, pagesize 0x200, ivt offset 0x0
NOTICE:  BL31: v2.4(release):
NOTICE:  BL31: Built : 11:24:17, Dec  9 2021


U-Boot 2020.04-at1 (Dec 09 2021 - 11:17:22 +0900)

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

```
Loading Environment from MMC... OK
In:    serial
Out:   serial
Err:   serial

 BuildInfo:
  - ATF
  - U-Boot 2020.04-at1

first boot since power on
switch to partitions #0, OK
mmc1 is current device
flash target is MMC:1
Net:   eth0: ethernet@30be0000 [PRIME], eth1: ethernet@30bf0000
Fastboot: Normal
Saving Environment to MMC... Writing to MMC(1)... OK
Normal Boot
Warning: Bootlimit (3) exceeded. Using altbootcmd.
Hit any key to stop autoboot:  0
u-boot=>
```

3. ブートディスク上の Linux カーネルを起動します。

```
u-boot=> boot
```

9.6. Armadillo のソフトウェアの初期化

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

初期化を行っても、ファームウェアパーティション(mmcblk2p4)は変更されません。故障が疑われる場合など、ファームウェアも初期化したい場合、初期化してから 「8.3. VPU や NPU を使用する」 を参考にしてもう一度書き込みしてください。

9.6.1. インストールディスクの作成

1. 512 MB 以上の microSD カードを用意してください。

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

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

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

```
baseos-x2-[VERSION].img
[ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_image.sh ¥
        --boot ~/imx-boot-[VERSION]/imx-boot_armadillo_x2 ¥
        --installer ./baseos-x2-[VERSION].img
```

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

3. ATDE に microSD カードを接続します。詳しくは「4.2.2. 取り外し可能デバイスの使用」を参考にしてください。

4. microSD カードのデバイス名を確認します

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

5. microSD カードがマウントされている場合、アンマウントします。

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

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

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

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

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

9.6.2. インストールディスクを使用した初期化

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

9.7. Armadillo のソフトウェアをアップデートする

Armadillo-IoT ゲートウェイ G4 では、開発・製造・運用それぞれに適した複数のソフトウェアアップデート方法を用意しています。本章では、それぞれのソフトウェアアップデート方法について説明します。

ソフトウェアアップデートを実現するソフトウェアの概要や仕様、用語については「13. ソフトウェア仕様」を参照してください。

9.7.1. SWU イメージとは？

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

SWU イメージは `swupdate` (https://sbabic.github.io/swupdate/swupdate.html) によって Armadillo Base OS 上で検証とインストールが実行されます。SWU イメージを Armadillo に転送するための方法は、用途や状況に合わせて様々な方法を用意しています。例えば、USB メモリから読み取る、ウェブサーバーからダウンロードする、hawkBit という Web アプリケーションを使うなどです。

9.7.2. SWU イメージの作成

SWU イメージの作成には、mkswu というツールを使います。

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

詳しくは「9.7.5. mkswu の desc ファイル」を参考にしてください。

1. mkswu の取得

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

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

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

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

```
[ATDE ~/swupdate-mkimage]$ mkswu --import
コンフィグファイルを更新しました：/home/atmark/swupdate-mkimage/
mkswu.conf
/home/atmark/swupdate-mkimage/mkswu.conf のコンフィグファイルとその鍵を
/home/atmark/mkswu にコピーします。
mkdir: ディレクトリ '/home/atmark/mkswu' を作成しました
'/home/atmark/swupdate-mkimage/swupdate.key' -> '/home/atmark/mkswu/
swupdate.key'
'/home/atmark/swupdate-mkimage/swupdate.pem' -> '/home/atmark/mkswu/
swupdate.pem'
/home/atmark/swupdate-mkimage/mkswu.conf のコンフィグファイルを
```

```
/home/atmark/mkswu/mkswu.conf にコピーしました。
mkswu でイメージ作成を試してから前のディレクトリを消してください。
```

1. 最初に行う設定

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

   使用している Armadillo に、過去に一度でもこの初回アップデート作業を行っている場合は二度同じ作業を行うことはできず、する必要もありません。その際に Armadillo に配置した公開鍵に対応する秘密鍵でアップデートを行いますので、 「9.7.5. mkswu の desc ファイル」 を参考にしてそのままにお使い下さい。

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

"/home/atmark/mkswu/initial_setup.swu" をそのまま使えますが、追加の desc を入れたい場合
次のコマンドで作成してください： mkswu "/home/atmark/mkswu/initial_setup.swu" ⏎
other_desc_files

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

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

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

❷ 証明鍵を保護するパスフレーズを 2 回入力します。

❸ swu イメージ自体を暗号化する場合に「y」を入力します。詳細は 「9.7.6. 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予定でしたら「n」を入力します。

❺ root のパスワードを 2 回入力します。

❻ atmark ユーザーのパスワードも 2 回入力します。何も入力しない場合は root と同じパスワードを使います。

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

❽ 作成したファイルを確認します。「swupdate.aes-key」は暗号化の場合のみに作成されます。

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

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

このイメージに他の変更も入れれます。他の /usr/share/mkswu/examples/ ディレクトリーにある.desc ファイルや「9.7.5. mkswu の desc ファイル」を参考にして、以下の例のように同じ swu にいくつかの.desc を組み込めます。

例えば、openssh を有効にします。

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

2. イメージのインストール

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

3. 次回以降のアップデート

   次回以降のアップデートは作成した証明鍵を使用して Armadillo-IoT ゲートウェイ G4 の SWU イメージを作成します。

   .desc ファイルの内容は /usr/share/mkswu/examples/ のディレクトリや「9.7.5. mkswu の desc ファイル」を参考にしてください。

9.7.3. イメージのインストール

イメージをインストールする方法として下記に示すような方法があります。

・ USB メモリからの自動インストール

Armadillo-IoT ゲートウェイ G4 に USB メモリを接続すると自動的にアップデートが始まります。アップデート終了後に Armadillo-IoT ゲートウェイ G4 は自動で再起動します。

USB メモリは 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 メモリのマウントされている場所を確認します。

❷ ファイルをコピーします。

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

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

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

❶ 証明が間違ったメッセージ。

・外部記憶装置からイメージのインストール（手動）

USB メモリ（ルート以外）や microSD カード等の外部記憶装置に 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。以下は外部記憶装置が/dev/mmcblk1p1(microSD カード）として認識された場合に、イメージのインストールを行う例です。

```
[armadillo ~]# mount /dev/mmcblk1p1 /mnt
[armadillo ~]# swupdate -i /mnt/initial_setup.swu
[INFO ] : SWUPDATE started :  Software Update started !
[INFO ] : SWUPDATE running :  Installation in progress
:（省略）
```

```
[ERROR] : SWUPDATE failed [0] ERROR : swupdate triggering reboot! ❶
[INFO ] : SWUPDATE successful ! SWUPDATE successful !
```

・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'
[INFO ] : SWUPDATE started :  Software Update started !
[INFO ] : SWUPDATE running :  Installation in progress
: (省略)
[ERROR] : SWUPDATE failed [0] ERROR : swupdate triggering reboot! ❶
[INFO ] : SWUPDATE successful ! SWUPDATE successful !
```

❶ はエラーとして赤で表示されますが、見やすくするためでエラーではありません。
❶

・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-iot-g4/image/baseos-x2-latest.swu ¥
                        > /etc/swupdate.watch ❸
[armadillo ~]# echo 'schedule="0 tomorrow"' > /etc/conf.d/swupdate-url
[armadillo ~]# echo 'rdelay="21600"' >> /etc/conf.d/swupdate-url ❹
[armadillo ~]# persist_file /etc/swupdate.watch /etc/conf.d/swupdate-url ❺
```

❶ swupdate-url サービスを有効します。

❷ サービスの有効化を保存します。

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

❹ チェックやインストールのスケジュールを設定します。

❺ 変更した設定ファイルを保存します。

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

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

・ hawkBit を使用した自動インストール

hawkBit で Armadillo-IoT ゲートウェイ G4 を複数台管理してアップデートすることができます。以下の「9.7.4. hawkBit サーバーから複数の Armadillo に配信する」を参考にしてください。

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

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

手順は次のとおりです。

1. コンテナ環境の準備

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

   Docker の準備ができたら、要件に合わせてコンテナの設定を行います。

   ・ ATDE の場合

   ・ `apt update && apt install mkswu` で最新のバージョンを確認してください。

   ・ ポート転送も必要です。一番シンプルな、プロキシを使用しない場合は 8080、TLS を使う場合は 443 を転送してください。

   vmware を使う場合は vmware の NAT モードのネットワークを使用している仮想マシン上で Web サーバを構成する [https://kb.vmware.com/s/article/2006955?lang=ja]ページを参考にしてください。

   ・ ホスト PC の IP アドレスを控えておいてください。

   ・ ATDE 以外の場合

   ・ Armadillo-IoT ゲートウェイ G4 開発用ツール [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/tools] から「Hawkbit docker-compose コンテナ」 をダウンロードして展開してください。この場合、以下に `/usr/share/mkswu/hawkbit-compose` を使う際に展開先のディレクトリとして扱ってください。

   ・ docker がアクセスできるホストネームやアドレスを控えておいてください。

2. hawkBit サーバーの準備

   `/usr/share/mkswu/hawkbit-compose/setup_container.sh` を実行して、質問に答えてください。

   以下に簡単な（TLS を有効にしない）テスト用の場合と、TLS を有効にした場合の例を参考にしてください。

   `setup_container.sh` を一度実行した場合はデータのディレクトリーにある `setup_container.sh` のリンクを実行して、ユーザーの追加等のオプション変更を行うこともできます。`--help` を参考にしてください。

```
[ATDE ~]$ /usr/share/mkswu/hawkbit-compose/setup_container.sh
docker-compose の設定ファイルと hawkBit のデータをどこに保存しますか？ [/home/atmark/hawkbit-compose] ❶
```

```
setup_container.sh へのリンクを /home/atmark/hawkbit-compose に作ります。
docker サービスに接続できませんでした。sudo でもう一度試します。
[sudo] atmark のパスワード: ❷
OK!
Hawkbit admin user name [admin] ❸
admin ユーザーのパスワード: ❹
パスワードを再入力してください:
追加の管理人アカウントのユーザーネーム（空にすると追加しません） ❺
hawkBit の「device」ユーザーを登録しますか？（自動登録用） [Y/n] ❻
device ユーザーのパスワード:
パスワードを再入力してください:
hawkBit の「mkswu」ユーザーを登録しますか？（swu のアップロード用） [Y/n] ❼
ユーザーにロールアウトの権限を与えますか？（インストール要求を出すこと） [Y/n] ❽
mkswu ユーザーのパスワード:
パスワードを再入力してください:
Setup TLS reverse proxy? [y/N] ❾

コンテナの設定が完了しました。docker-compose コマンドでコンテナの管理が可能です。
/home/atmark/hawkbit-compose/setup_container.sh を再び実行すると設定の変更が可能です。
hawkBit コンテナを起動しますか？ [Y/n] ❿
Creating network "hawkbit-compose_default" with the default driver
Pulling mysql (mysql:5.7)...
: (省略)
Creating hawkbit-compose_hawkbit_1 ... done
Creating hawkbit-compose_mysql_1   ... done
```

図 9.134 hawkBit コンテナの一番簡単な設定（テスト用）の実行例

❶ コンテナのコンフィグレーションとデータベースの場所を設定します。

❷ docker の設定によって sudo が必要な場合もあります

❸ hawkBit の Web UI のログインを admin とデフォルトにします。

❹ そのパスワードを二回入力します。

❺ 追加のユーザーが必要な場合に追加できます。

❻ examples/hawkbit_register.desc で armadillo を登録する場合に作っておいてください。詳細は 「9.7.4.2. SWU で hawkBit を登録する」 を参考にしてください。

❼ hawkbit_push_update でアップデートを CLI で扱う場合に作っておいてくダサい。詳細は <sct.hawkbit_push_update>> を参照してください。

❽ さらに、hawkbit_push_update でアップデートを実行までする場合は、この権限を許可してください。

❾ ここでは http でテストのコンテナを作成するので、「N」のままで進みます。

❿ コンテナを起動します。初期化が終わったら <IP>:8080 でアクセス可能になります。

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

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

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

LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUJlekNDQVNHZ0F3SUJBZ0lVQTByZ0cwcTJF
SFNnampmb0tUZWg3aGlaSVVVd0NnWUlLb1pJemowRUF3SXcKRXpFUk1BOEdBMVVFQXd3SU1UQXVN
UzR4TGpFd0hoY05Nakl3TXpJMU1EVXhOVFU0V2hjTk16SXdNekl5TURVeApOVFU0V2pBVE1SRXdE
d1lEVlFRRERBZ3hNQzR4TGpFdU1UQlpNQk1HQnlxR1NNNDlBZ0VHQ0NxR1NNNDlBd0VICkEwSUFC
SDFFREhBN3NOTlFJUDlTdlhlUnNmWjl2dVVFWkRkMVE2TzViRlV2RTh4UjUwUlBCLzNlajMzd0VI
NEoKYmZqb296bEpXaExlSG5SbGZsaHExVDlKdm5TalV6QlJNQjBHQTFVZERnUVdCQlFBUmYvSkdT
dkVJek5xZ2JMNQpQamY2VGRpSk1EQWZCZ05WSFNNRUdEQVdnQlFBUmYvSkdTdkVJek5xZ2JMNVBq
ZjZUZGlKTURBUEJnTlZIUk1CCkFmOEVCVEFEQVFIL01Bb0dDQ3FHU0000OUJBTUNBMGdBTUVVQ0lD
Nis3ZzJlZk1SRXl0RVk5WDhDNC8vUEw1U1kKWUlGZHUxVFZiUEZrSlV0SUFpRUE4bm1VSnVQSFlz
SHg2N2ErZFRwSXZ1QmJUSG1KbWd6dUl3bTJ2RXppRnZRPQotLS0tLUVORCBDRVJUSUZJQ0FURS0t
LS0tCg== ❼
```

```
Let's encrypt の設定は後で足したい場合に setup_container.sh を--letsencrypt で実行してください。

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

図 9.135 hawkBit コンテナの TLS ありの場合の実行例

❶ 今回は TLS を有効にしますので、「y」を答えます。

❷ lighttpd サービスが起動している場合に聞かれます。不要なので、停止します。

❸ 証明書の common name を入力してください。ATDE の場合、ポート転送によってホストの IP アドレスで接続しますのでそのアドレスを入力します。Let's encrypt を使用する場合には外部からアクセス可能な DNS を入力してください。

❹ 証明書の有効期間を設定します。デフォルトでは 10 年になっています。Let's encrypt を使用する場合には使われていません。

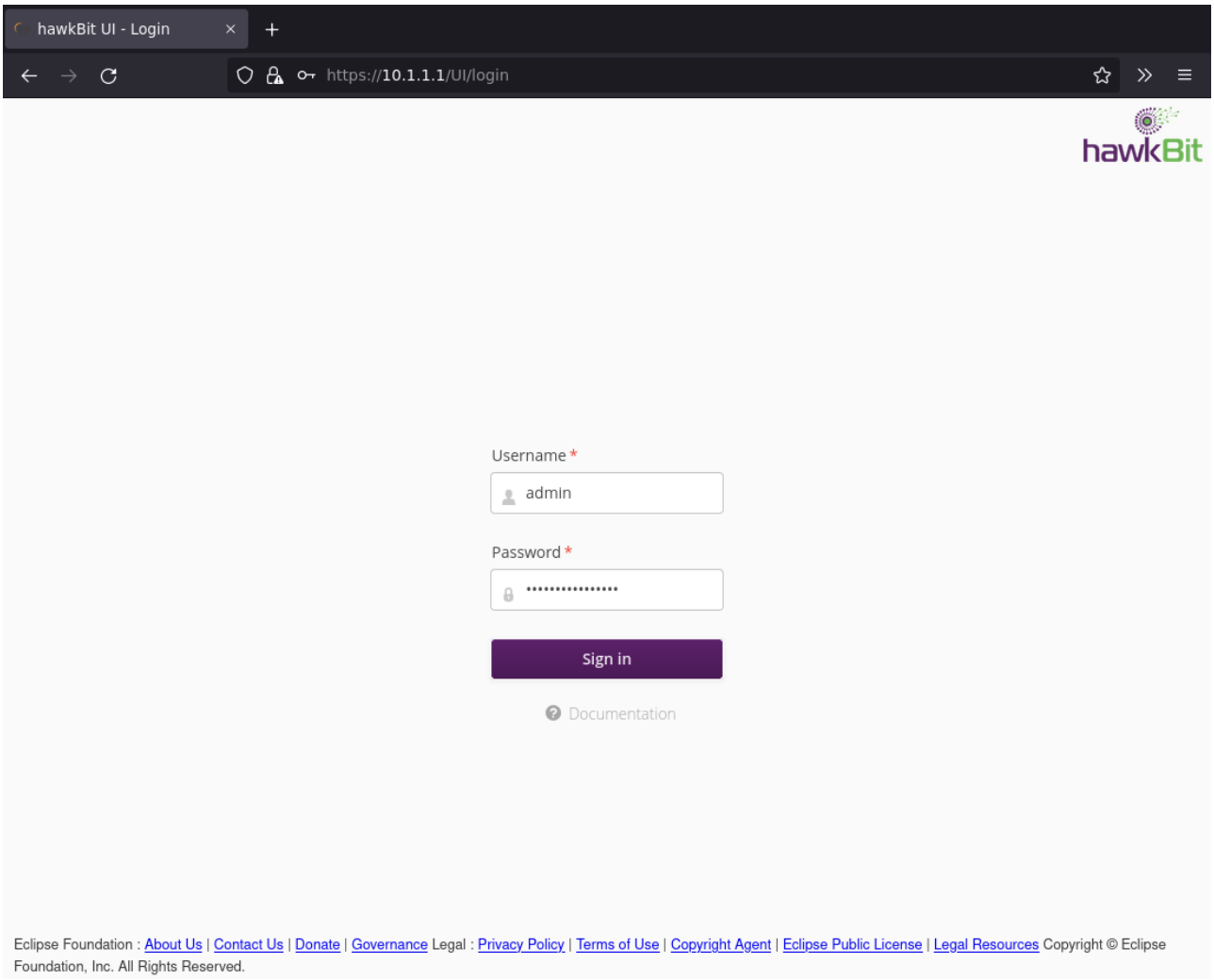
❺ クライアント側では x509 証明書で認証をとることができますが、この例では使用しません。

❻ Let's encrypt による証明書を作成できます。ATDE の場合は外部からのアクセスが難しいので、この例では使用しません。

❼ 自己署名証明書を作成したので、 Armadillo に設置する必要があります。この証明書の取扱いには 「9.7.4.2. SWU で hawkBit を登録する」 を参照してください。

3. hawkBit へのログイン

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

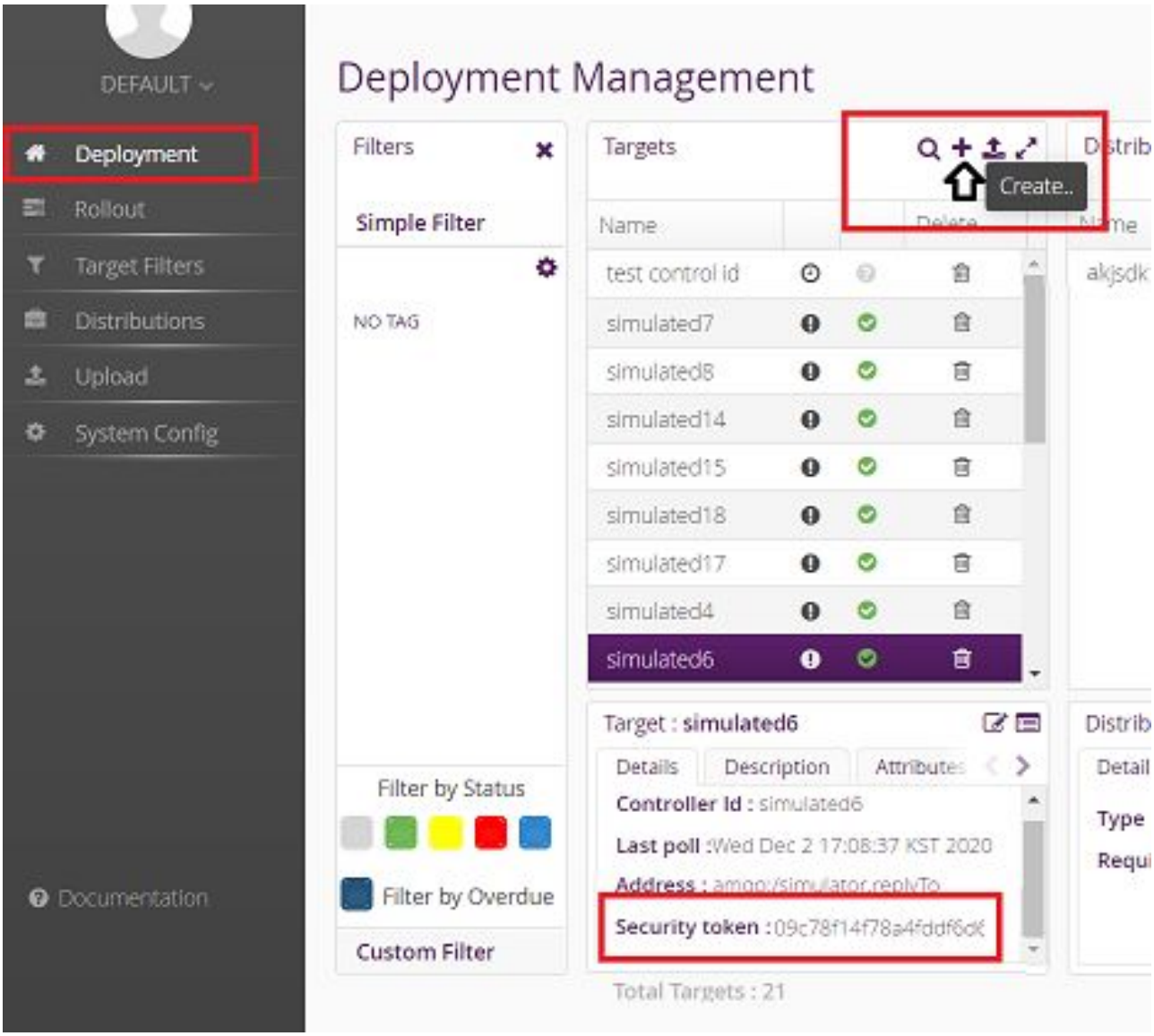


デフォルトでは次のアカウントでログインできます。

ユーザー	admin
パスワード	admin

4. Armadillo を Target に登録する

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



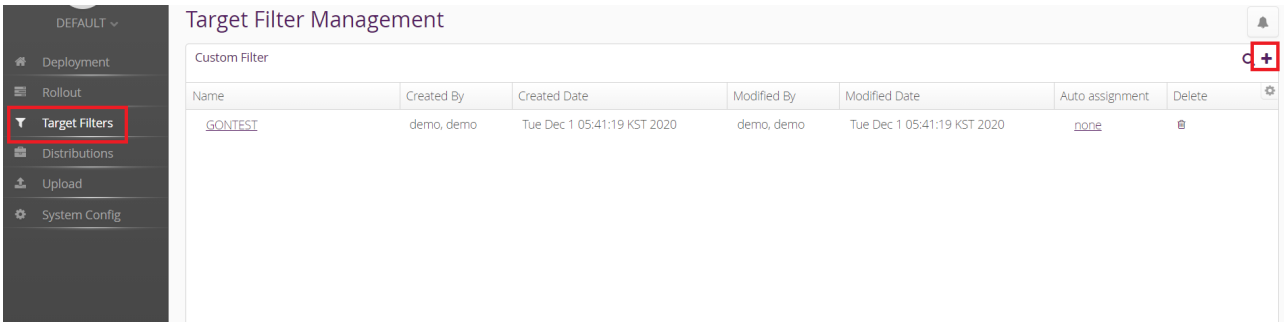
"+"をクリックして Target を作成します。

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

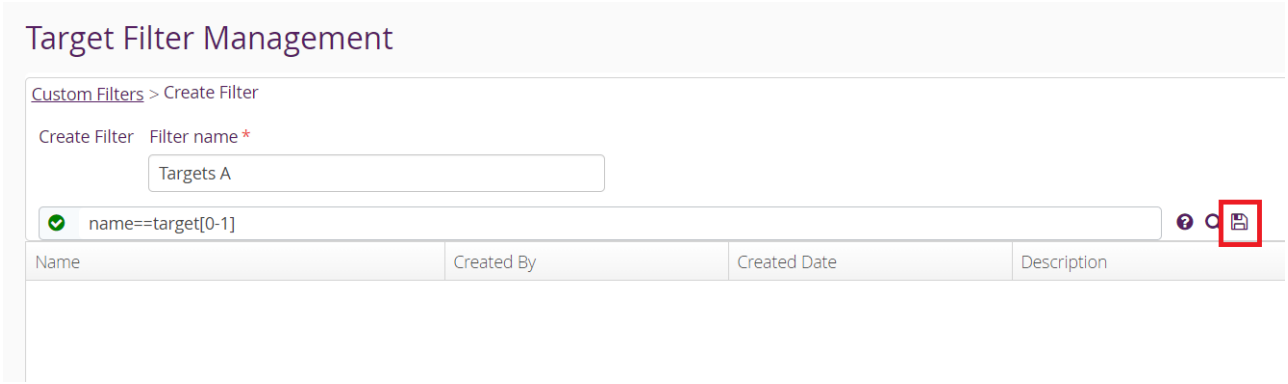
メモした<文字列>を Armadillo の /etc/swupdate.cfg に設定すると Hawkbit への接続認証が通るようになります。

5. Target Filter を作成する

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



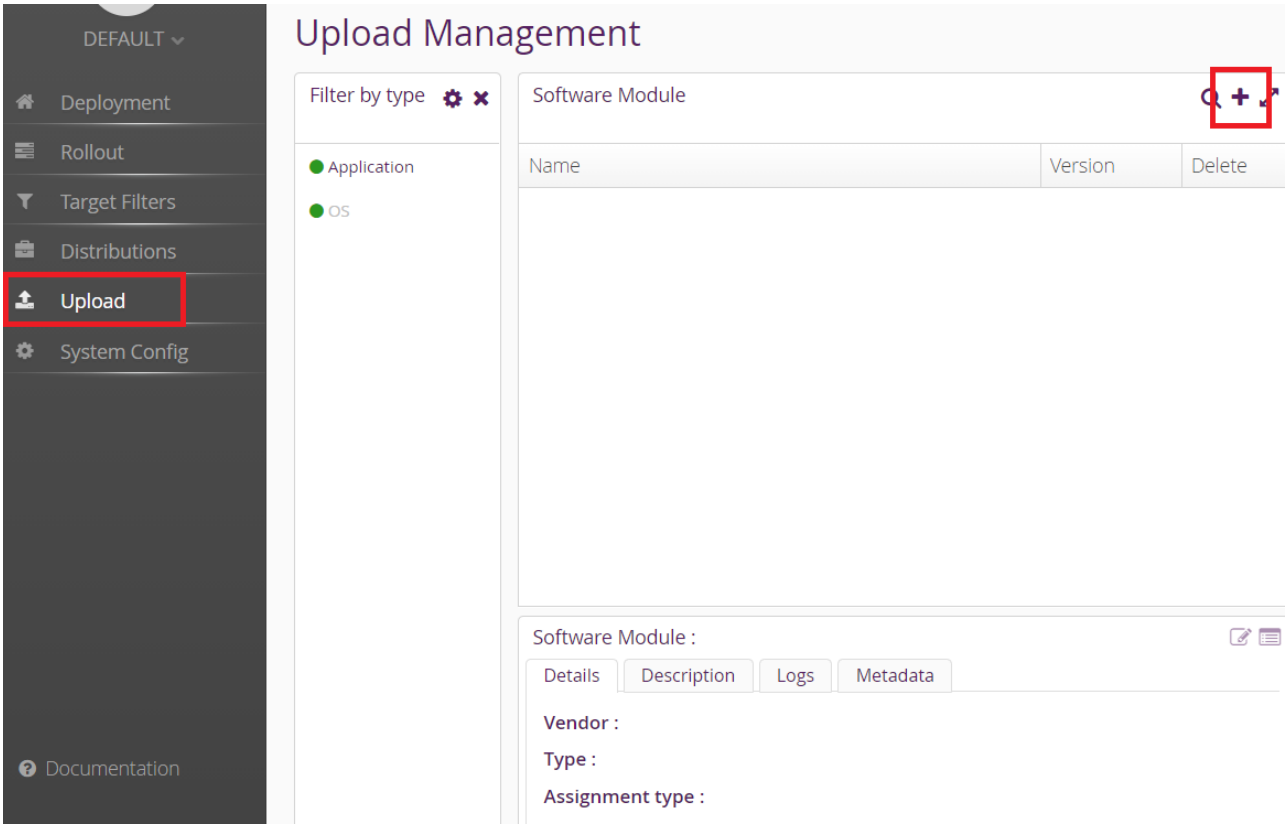
"+" をクリックして新規に Target Filter を作成します。



Filter name と 検索式を入力して保存します。

6. Software module を作成する

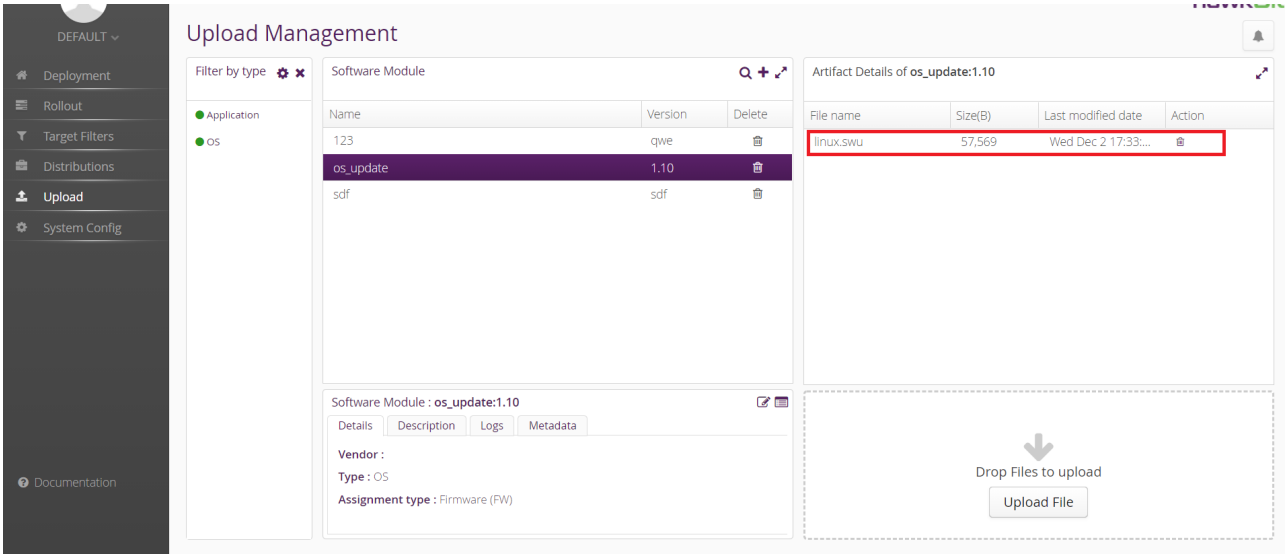
左側のメニューから"Upload"をクリックして、Upload Management の画面に移ります。



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

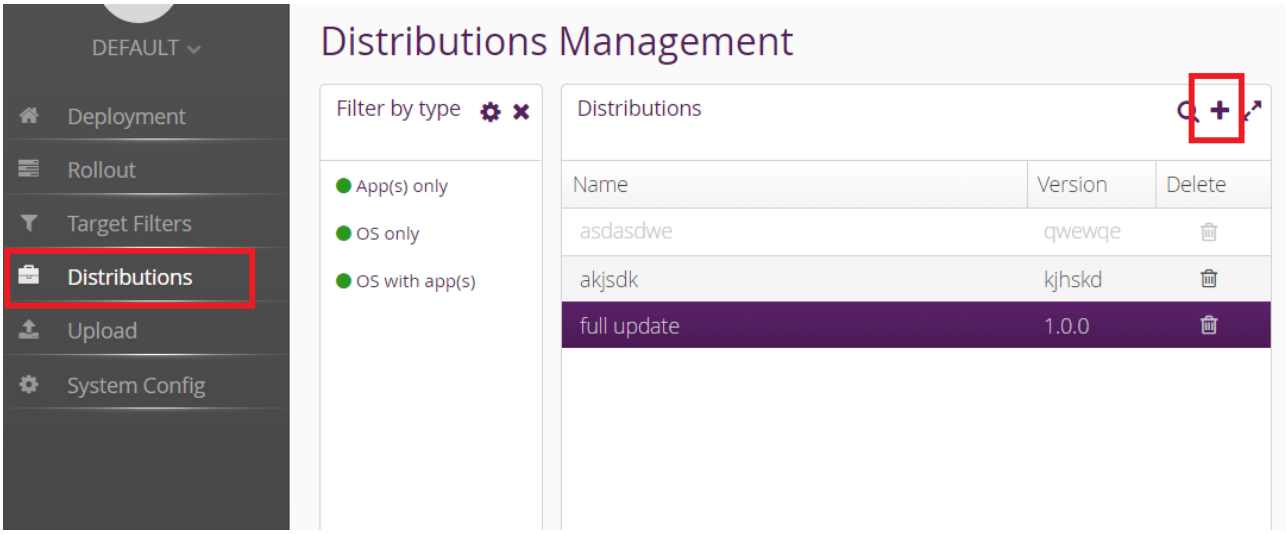
7. swu パッケージをアップロードして Software module に関連付ける

   先程作成した Software module を選択して、ハイライトされた状態で、"Upload File"ボタン、もしくは、ファイルをドラッグアンドドロップします。

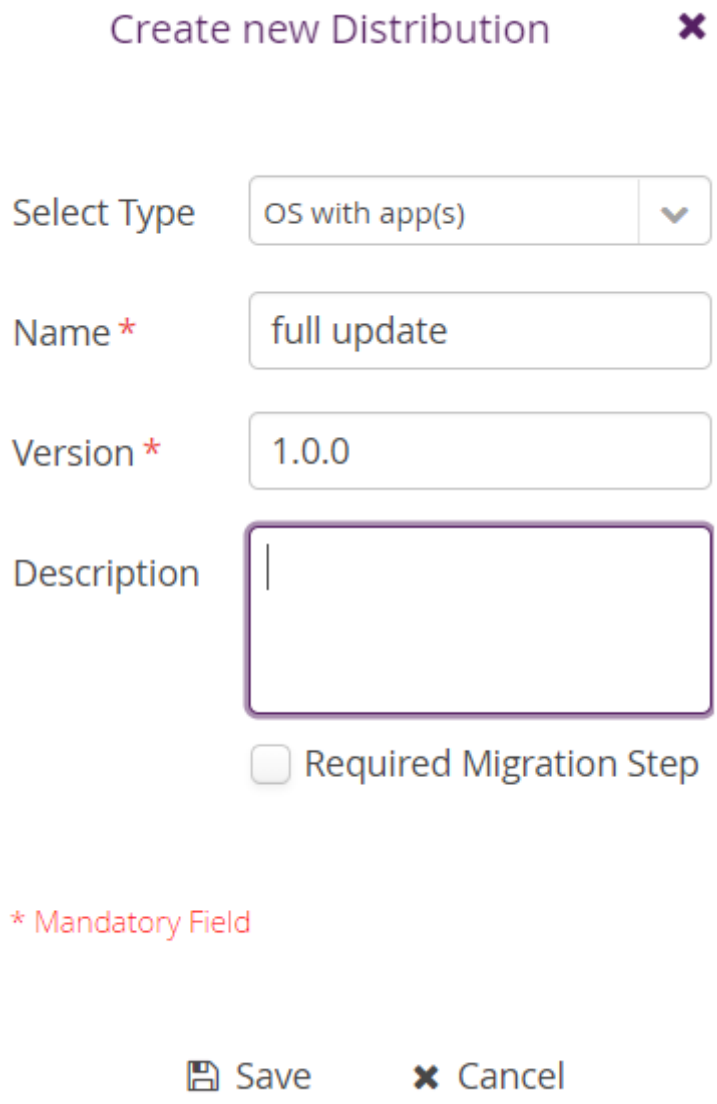


8. Distribution を作成して Software module を関連付ける

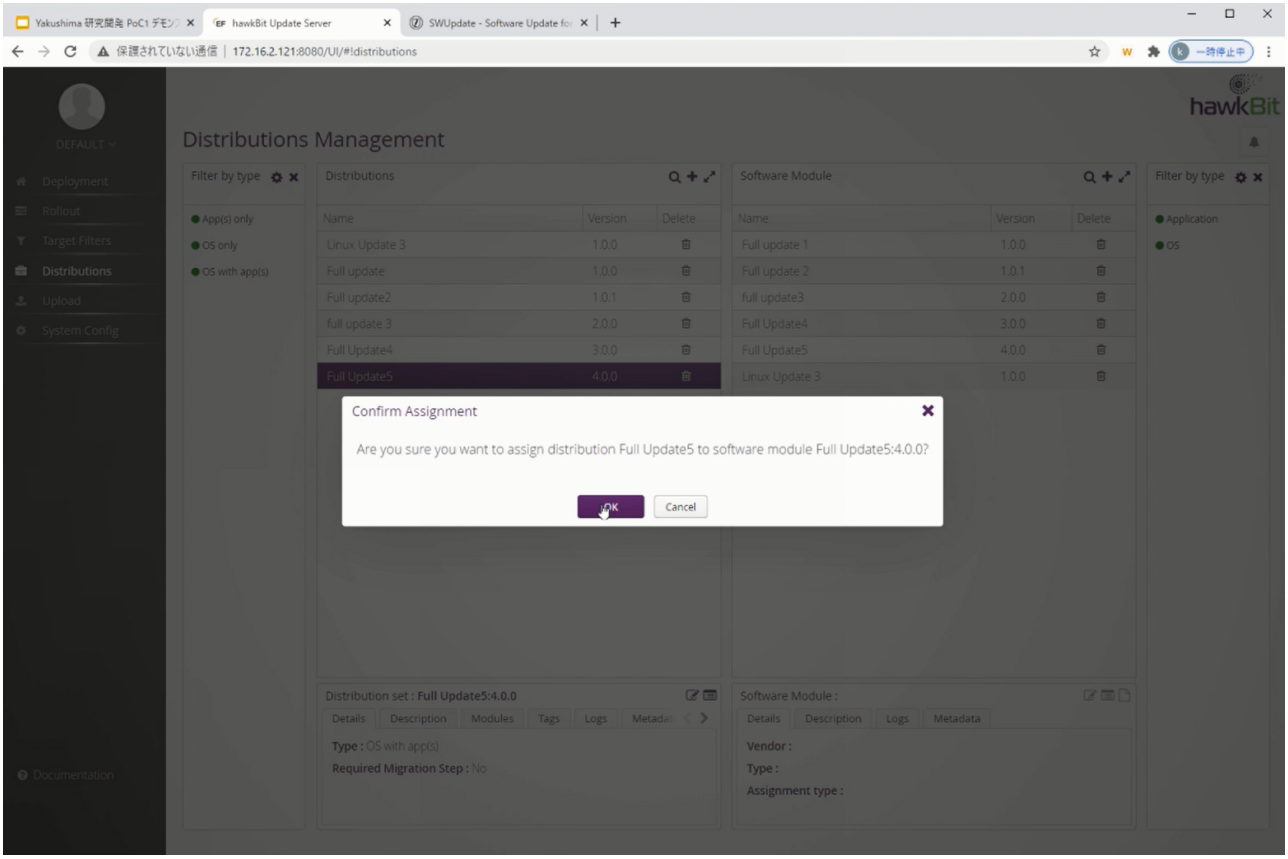
左側のメニューから"Distribution"をクリックして、Distribution Management の画面に移ります。



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

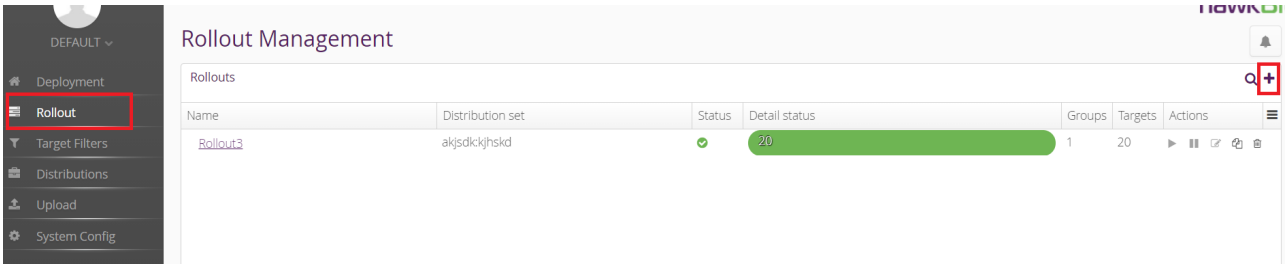


"Software module"のペインから先程作成した Software をドラッグして、作成した Distribution の上にドロップします。

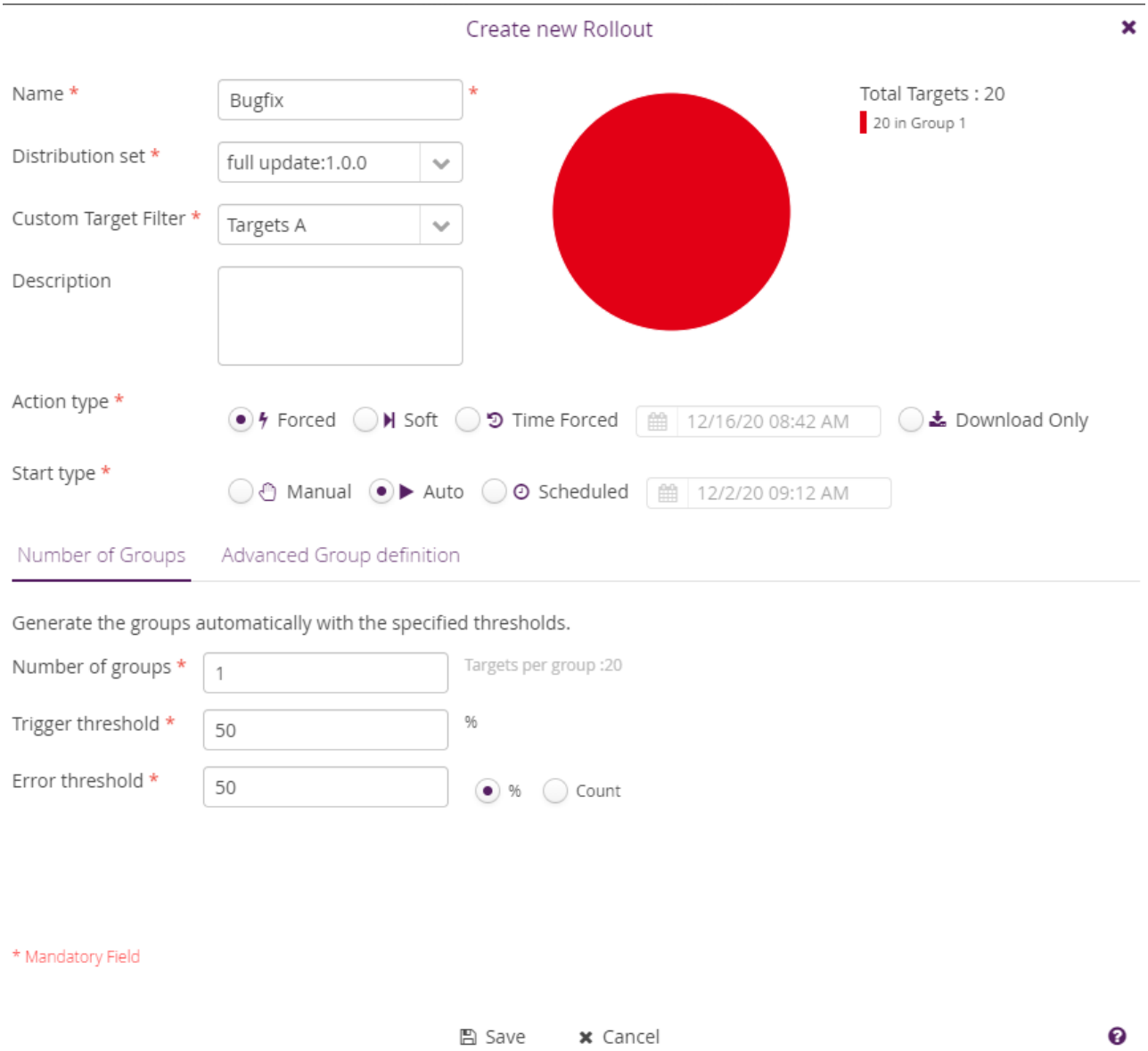


9. Rollout を作成してアップデートを開始する

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



   "+"をクリックして Rollout を作成します。



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

10. アップデートの状態を確認する。

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

    アップデート中は黄色、アップデートが正常に完了すると緑色になります。

9.7.4.1. hawkBit のアップデート管理を CLI で行う

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

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

1. hawkbit_push_update の実行例です

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

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

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

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

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

❶ この例ではあらかじめ作成されてる enable_sshd.swu を hawkBit に登録します。

❷ --no-rollout を使う場合に SWU を「distribution」として登録します。デフォルトでは rollout も作成します。テストする際、デバイスがまだ登録されていなければ rollout の段階で失敗します。

❸ 同じ SWU で rollout を二回作成した場合にエラーが出ます。もう一度作りたい場合は `--new` を使ってください。

❹ 一度 rollout をスタートして、 Armadillo で失敗した場合には失敗したデバイスだけに対応した rollout を作れます。

❺ 作成した rollout をすぐ実行します。このオプションには追加の権限を許可する必要があります。

❻ スタートまで行う実行例です。実行結果は Web UI で表示されます。

9.7.4.2. SWU で hawkBit を登録する

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

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

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

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

```
[ATDE ~/mkswu]$ cp /usr/share/mkswu/examples/hawkbit_register.* . ❶

[ATDE ~/mkswu]$ vi hawkbit_register.sh ❷
# Script configuration: edit this if required!
# user given here must have CREATE_TARGET,READ_TARGET_SECURITY_TOKEN permissions
HAWKBIT_USER=device
HAWKBIT_PASSWORD="CS=wC,zJmrQeeKT.3" ❸
HAWKBIT_URL=https://10.1.1.1 ❹
HAWKBIT_TENANT=default
# set custom options for suricatta block or in general in the config
CUSTOM_SWUPDATE_SURICATTA_CFG="" # e.g. "polldelay = 86400;"
CUSTOM_SWUPDATE_CFG=""
# set to non-empty if server certificate is invalid
SSL_NO_CHECK_CERT=
# or set to cafile that must have been updated first
SSL_CAFILE=
# ... or paste here base64 encoded crt content
SSL_CA_BASE64="
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUJlakNDQVNHZ0F3SUJBZ0lVYTMvYXpNSHZ0
bFFnaFZnZDhIZWhMaEwxNm5Bd0NnWUlLb1pJemowRUF3SXcKRXpFUk1BOEdBMVVFQXd3SU1UQXVN
UzR4TGpFd0hoY05Nakl3TWpFNE1EVTFNakV6V2hjTk16SXdNakUyTURVMQpNakV6V2pBVE1SRXdE
d1lEVlFRRERBZ3hNQzR4TGpFdU1UQlpNQk1HQnlxR1NNNDlBZ0VHQ0NxR1NNNDlBd0VICkEwSUFC
RFJGcnJVV3hHNnBHdWVoejRkRzVqYkVWTm5scHUwYXBHT1c3UVBPYUF4cWp1ZzJWYjk2UHNScWJY
Sk8KbEFDVVo2OStaMHk3clBqeDJHYnhDNms0czFHalV6QlJNQjBHQTFVZERnUVdCQlJtZzhxL2FV
OURRc3EvTGE1TgpaWFdkTHROUmNEQWZCZ05WSFNNRUdEQVdnQlJtZzhxL2FVOURRc3EvTGE1TlpY
V2RMdE5SY0RBUEJnTlZIUk1CCkFmOEVCVEFEQVFIL01Bb0dDQ3FHU000OUJBTUNBMGNBTUVRQ0lB
ZTRCQ0xKREpWZnFTQVdRcVBqNTFmMjJvQkYKRmVBbVlGY2VBMU45dE8rN0FpQXVvUEV1VGFxWjhH
UFYyRUg1UWdOMFRKS05SckJDOEtpNkZWcFlkRUowYWc9PQotLS0tLUVORCBDRVJUSUZJQ0FURS0t
LS0tCg== ❺
"
# ... or add your own options if required
CURLOPT=-s

:（省略）

[ATDE ~/mkswu]$ cat hawkbit_register.desc ❻
:（省略）
swdesc_script hawkbit_register.sh --version extra_os.hawkbit 1

[ATDE ~/mkswu]$ mkswu hawkbit_register.desc ❼
hawkbit_register.swu を作成しました。

[ATDE ~/mkswu]$ mkswu initial_setup.desc hawkbit_register.desc ❽
hawkbit_register.desc を組み込みました。
initial_setup.swu を作成しました。
```

❶ hawkbit_register.sh と .desc ファイルをカレントディレクトリにコピーします。

❷ hawkbit_register.sh を編集して、設定を記載します。

❸ hawkBit の設定の時に選んだ「device」ユーザーのパスワードを入力します。この例のパスワードは使用しないでください。

❹ hawkBit サーバーの URL を入力します。

❺ TLS を使用の場合に、コンテナ作成の時の証明書を base64 で入力します。

❻ hawkbit_register.desc の中身を確認します。hawkbit_register.sh を実行するだけです。

❼ SWU を作成して、initial_setup がすでにインストール済みの Armadillo にインストールできます。

❽ または、initial_setup.desc と合わせて hawkbit_register を含んだ initial_setup.swu を作成します。

9.7.5. mkswu の desc ファイル

.desc ファイルを編集して、いくつかのコマンドが使えます。

例

```
[ATDE ~/mkswu]$ cat /usr/share/mkswu/examples/usb_container_nginx.desc
version=1

swdesc_usb_container "nginx_alpine.tar" ❶
swdesc_files --extra-os nginx_start ❷
```

❶ nginx_alpine.tar にあるコンテナをインストールします。

❷ nginx_start にあるファイルを転送します。

コマンドは書かれた順番でインストールします。インストールするかどうかの判断はバージョンで行います:

```
component=<component>
version=<version>
か
swdesc_xxx --version <component> <version> [options]
```

- <component>は以下のどれかにしてください (デフォルトでは .desc ファイルのファイル名を使います)

  1. base_os: rootfs (Armadillo Base OS)を最初から書き込む時に使います。現在のファイルシステムは保存されていない。

     この場合、/etc/swupdate_preserve_files に載ってるファイルのみをコピーして新しい base OS を展開します。

     この component がないと現在の rootfs のすべてがコピーされます。

  2. extra_os.<文字列>: rootfs の変更を行う時に使います。<文字列> には任意の文字列を指定します。

     rootfs を変更を行う時に使います。 swdesc_* コマンドに --extra-os オプションを追加すると、 component に自動的に extra_os. を足します。

  3. <文字列> (コンテナの名前などの任意の文字列) : rootfs の変更がないときに使います。

     この component を使うと rootfs の変更ができませんのでご注意ください。

- アップデートをインストールする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します。

<component> がまだインストールされてなかった時か <version> が上がる時にインストールします。

デフォルトではダウングレードはできませんが、 --install-if=different オプションを追加することで <version> が変わる際にインストール可能になります。

アップデートの一部をインストールすることもありますので、複数の component で管理し、いくつかの古いバージョンに対応するアップデートも作成可能です。

以下のコマンドから使ってください

- swdesc_tar と swdesc_files でファイルを転送します。

```
swdesc_tar [--dest <dest>] <tar_file>
swdesc_files [--dest <dest>] [--basedir <basedir>] ¥
             <file> [<more files>]
```

swdesc_tar の場合、予め用意されてある tar アーカイブをこのままデバイスで展開します。

--dest <dest> で展開先を選ぶことができます。デフォルトは / （--extra-os を含め、バージョンの component は base_os か extra_os.* の場合）か /var/app/rollback/volumes/ （それ以外の component)。後者の場合は /var/app/volumes と /var/app/rollback/volumes 以外は書けないので必要な場合に --extra-os を使ってください。

swdesc_files の場合、mkswu がアーカイブを作ってくれますが同じ仕組みです。

--basedir <basedir> でアーカイブ内のパスをどこで切るかを決めます。

- 例えば、swdesc_files --extra-os --basedir /dir /dir/subdir/file ではデバイスに /subdir/file を作成します。
- デフォルトは <file> から設定されます。ディレクトリであればそのまま basedir として使います。それ以外であれば親ディレクトリを使います。

- swdesc_command や swdesc_script でコマンドを実行する

```
swdesc_command <command> [<more commands>]
swdesc_script <script>
```

アップデート先の環境でコマンドやスクリプトファイルを走らせます。

バージョンの component は base_os と extra_os 以外の場合、 /var/app/volumes と /var/app/rollback/volumes 以外何も変更できないのでご注意ください。

コマンドが成功しないとアップデートが失敗します。

- swdesc_exec でファイルを配ってコマンドでそのファイルを使う

```
swdesc_exec <file> <command>
```

swdesc_command と同じくコマンドを走らせますが、<file> を先に転送してコマンド内で"$1"として使えます。

- swdesc_command_nochroot, swdesc_script_nochroot, swdesc_exec_nochroot で起動中のシステム上でコマンドを実行します。

  このコマンドは nochroot なしのバージョンと同じ使い方で、現在起動中のシステムに変更や確認が必要な場合にのみ使用してください。

  nochroot コマンドは確認を一切しないため、Armadillo が起動できない状態になる可能性もあります：充分にご注意ください。

  例が必要な場合は /usr/share/mkswu/examples/firmware_update.desc を参考にしてください。

- swdesc_embed_container, swdesc_usb_container, swdesc_pull_container で予め作成したコンテナを転送します。

  ```
  swdesc_embed_container <container_archive>
  swdesc_usb_container <container_archive>
  swdesc_pull_container <container_url>
  ```

  例は「9.2.5. コンテナの配布」を参考にしてください。

- swdesc_boot で imx-boot を更新します。

  ```
  swdesc_boot <boot image>
  ```

  このコマンドだけにバージョンは自動的に設定されます。

コマンドの他には、設定変数もあります

- DESCRIPTION: 自由なイメージの説明、ログに残ります。
- PRIVKEY, PUBKEY: 署名鍵と証明書
- PRIVKEY_PASS: 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: 暗号化の鍵
- POST_ACTION=container: コンテナのみのアップデート後に再起動を行いません。

  コンテナの中身だけをアップデートする場合、Armadillo-IoT ゲートウェイ G4 を再起動せずにコンテナだけを再起動させます。

- POST_ACTION=poweroff: アップデート後にシャットダウンを行います。
- 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。

9.7.5.1. 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。「9.7.5.4. 例: swupdate_preserve_files で Linux カーネル以外の Armadillo-IoT ゲートウェイ G4 向けのイメージをインストールする方法」 を参考にしてください。

コピーのタイミングによって、以下のどれかを使ってください：

1. 単にファイルを記載する。

   この場合、アップデートする前にファイルをコピーします。 baseos のイメージと同じ swu にアップデートしたいファイルを記載していても、このファイルが Armadillo Base OS に含まれないのであれば問題なくアップデートできます。

   例: echo ”/root/.profile” >> /etc/swupdate_preserve_files

2. POST のキーワードの後に記載する。

   この場合、アップデートの最後でコピーします。 Armadillo Base OS に含まれてるファイルであれば、インストール前にコピーしても保存されないのでコピーのタイミングをずらします。

   そのコピーが最後に行われるので、同じアップデートでファイルの変更ができません。アップデートを別けて、 baseos のイメージをインストールしてからこのファイルを更新することができます。

   例: echo ”POST /etc/conf.d/podman-atmark” >> /etc/swupdate_preserve_files

9.7.5.2. 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
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。

9.7.5.3. 例: Armadillo Base OS アップデート

ここでは、「9.4. 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 バージョンを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 を見て上がるように設定してください。

❹ イメージを作成します。パスワードは証明鍵の時のパスワードです。

9.7.5.4. 例: swupdate_preserve_files で Linux カーネル以外の Armadillo-IoT ゲートウェイ G4 向けのイメージをインストールする方法

Armadillo-IoT ゲートウェイ G4 向けのアップデートイメージに Linux カーネルが含まれています。

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

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

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

❷ 変更した設定ファイルを保存します

/usr/share/mkswu/examples/update_kernel*.desc を使うと、このパスを自動的に /etc/swupdate_preserve_files に追加します:

```
swdesc_script update_preserve_files.sh -- ¥
        "POST /boot" ¥
        "POST /lib/modules"
```

9.7.6. 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。

9.8. Armadillo Base OS の操作

Armadillo Base OS は Alpine OS をベースとして作られています。

このセクションでは Armadillo Base OS の機能を紹介します。

9.8.1. アップデート

Armadillo Base OS は swupdate によってアップデートすることができます。

アップデートする際には、rootfs ファイルシステムにインストールされたファイルをすべて消して、アップデートの中身と /etc/swupdate_preserve_files に記載されているファイルで新しい rootfs を作ります。「9.7.5.1. swupdate_preserve_files について」 を参照してください。

アップデートでファイルを削除してしまった場合に abos-ctrl mount-old で前のシステムを read-only でマウントして、削除されたファイルをコピーすることもできます。

9.8.2. overlayfs と persist_file について

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

そのため、ファイルを変更した後 Armadillo の電源を切ると変更内容は保持されません。開発中などに rootfs の変更内容を保持するには、変更したファイルに対して persist_file コマンドを使用します。

開発以外の時は安全のため、ソフトウェアアップデートによる更新を実行してください。アップデート手順に関しては 「9.7. Armadillo のソフトウェアをアップデートする」 を参照してください。

rootfs の内容を変更しても、ソフトウェアアップデートを実施した際に変更した内容が保持されない可能性があります。ソフトウェアアップデート実施後も変更内容を保持する手順に関しては 「9.7.5.1. swupdate_preserve_files について」 を参照してください。

persist_file コマンドの概要を 「図 9.136. 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 9.136 persist_file のヘルプ

1. ファイルの保存・削除手順例

```
[armadillo ~]# echo test > test
[armadillo ~]# persist_file -rv /root
'/root/test' -> '/mnt/root/test' ❶
'/root/.ash_history' -> '/mnt/root/.ash_history'
[armadillo ~]# rm -f test
[armadillo ~]# persist_file -rv /root
removed '/mnt/root/test' ❷
removed '/mnt/root/.ash_history' ❸
'/root/.ash_history' -> '/mnt/root/.ash_history'
```

図 9.137 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 9.138 persist_file ソフトウェアアップデート後も変更を維持する手順例

❶ 何らかのファイルの内容を変更します。

❷ -P オプションを付与して persist_file を実行します。

❸ swupdate_preserve_files に追加されたことを確認します。

3. 変更ファイルの一覧表示例

```
[armadillo ~]# mkdir dir
[armadillo ~]# persist_file -l
directory          /
directory          /root
opaque directory   /root/dir ❶
whiteout           /root/test ❷
regular file       /root/.ash_history
directory          /etc
regular file       /etc/resolv.conf
directory          /var
symbolic link      /var/lock
:（省略）
```

図 9.139 persist_file 変更ファイルの一覧表示例

❶ rootfs のファイルを見せないディレクトリは opaque directory と表示されます。

❷ 削除したファイルは whiteout と表示されます。

4. パッケージをインストールする時は apk コマンドを使用してメモリ上にインストールできますが、persist_file コマンドで rootfs に直接インストールすることも可能です。

```
[armadillo ~]# persist_file -a add strace
(1/3) Installing fts (1.2.7-r1)
(2/3) Installing libelf (0.185-r0)
(3/3) Installing strace (5.14-r0)
Executing busybox-1.34.1-r3.trigger
OK: 251 MiB in 188 packages
Install succeeded, but might not work in the running system
Please reboot if installed program does not work ❶
[armadillo ~]# strace ls
:（省略）
```

```
exit_group(0)                           = ?
+++ exited with 0 +++
```

図 9.140 persist_file でのパッケージインストール手順例

❶ この例では Armadillo を再起動せずにインストールしたコマンドを使用できましたが、Armadillo の再起動が必要となるパッケージもありますので、その場合は Armadillo を再起動してください。

9.8.3. ロールバック状態の確認

Armadillo Base OS の ルートファイルシステムが壊れて起動できなくなった場合に自動的に前のバージョンで再起動します。

自分で確認する必要がある場合に abos-ctrl status でロールバックされてるかどうかの確認ができます。

必要な場合（例えば、自分のアプリケーションがアップデート直後に問題があった場合)、 abos-ctrl rollback で手動のロールバックも可能です。ロールバックにエラーがなかったら、再起動してロールバックを完了します。

なお、/var/at-log/atlog に切り替えの際に必ずログを書きますので、調査の時に使ってください。

```
[armadillo ~]# cat /var/at-log/atlog
Mar 17 14:51:35 armadillo NOTICE swupdate: Installed update to /dev/mmcblk2p2: ¥
extra_os.sshd: unset -> 1, extra_os.initial_setup: unset -> 1
Mar 17 16:48:52 armadillo NOTICE swupdate: Installed update to /dev/mmcblk2p1: ¥
boot: 2020.04-at5 -> 2020.04-at6, base_os: 3.15.0-at.3 -> 3.15.0-at.4
Mar 17 17:42:15 armadillo NOTICE swupdate: Installed update to /dev/mmcblk2p2: ¥
other_boot: 2020.04-at5 -> 2020.04-at6, container: unset -> 1, extra_os.container: unset -> 1
```

図 9.141 /var/at-log/atlog の内容の例

9.8.4. ボタンやキーを扱う

自分のアプリケーションで直接入力の処理ができない場合に Base OS から簡単な処理ができます。

buttond サービスで指定されたイベントでコマンドを実行します。

デフォルトでは 「表 14.25. CON12 信号配列」 にある PWR_OFF と REBOOT を 3 秒押す（pull down) 場合に poweroff か reboot を実行します。

/etc/atmark/buttond.conf に BUTTOND_ARGS を上書きすればその対応を無効にすることもできますし、別のキー（SW1 など）の対応も追加できます:

- -s <key> -a "command" : 早押しの設定。キーを 1 秒以内に離せば早押しと認識されてコマンドを実行します。その 1 秒のタイミングは -t <time_ms> でチューニング可能です。
- -l <key> -s "command" : 長押しの設定。キーを 5 秒押しつづけたらその時にコマンドを実行します。その 5 秒のタイミングは -t <time_ms> でチューニング可能です。
- 一つのキーを違うタイミングで何回か設定できます。その場合、長押しの設定あればその一番長いタイミングで実行されますが、他のアクションはキーを放す時に合っているコマンドを実行します。

(例：-s 1 秒、-l 2 秒、-l 10 秒では、1秒以内に離したら一番目、2 秒以上で 10 秒以内に離したら二番目、10 秒を越えたら三番目のコマンドを実行します)

以下にデフォルトを維持したままで SW1 の早押しと長押しにそれぞれの場合にコマンドを実行させます。

```
[armadillo ~]# vi /etc/atmark/buttond.conf ❶
BUTTOND_ARGS="$BUTTOND_ARGS -s prog1 -a 'date >> /tmp/shortpress'"
BUTTOND_ARGS="$BUTTOND_ARGS -l prog1 -t 5000 -a 'date >> /tmp/longpress'"
[armadillo ~]# persist_file /etc/atmark/buttond.conf ❷
[armadillo ~]# rc-service buttond restart ❸
buttond         | * Stopping button watching daemon ...                                  [ ok ]
buttond         | * Starting button watching daemon ...                                  [ ok ]
[armadillo ~]# cat /tmp/shortpress ❹
Tue Mar 22 17:16:42 JST 2022
Tue Mar 22 17:16:43 JST 2022
[armadillo ~]# cat /tmp/longpress
Tue Mar 22 17:16:48 JST 2022
```

図 9.142 buttond で SW1 を扱う

❶ カスタマイズ用のコンフィグファイルを編集します。早押しで shortpress, 長押し（5 秒）で longpress に日付を出力します。

❷ コンフィグファイルを保存します。

❸ buttond サービスを再起動させます。ここで早押し二回、長押し一回行います。

❹ 押された回数を確認します。

USB キーボードや他の入力デバイスにも対応できます。

1. デバイスを接続してから、 buttond でデバイス名とキーコードを確認します。

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

❶ buttond を -vvv で冗長出力にして、すべてのデバイスを指定します。

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

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

```
[armadillo ~]# vi /etc/atmark/buttond.conf
BUTTOND_ARGS="$BUTTOND_ARGS -i /dev/input/by-id/usb-0566_3029-event-kbd"
BUTTOND_ARGS="$BUTTOND_ARGS -s LEFTCTRL -a 'podman_start button_pressed_container'"
[armadillo ~]# persist_file /etc/atmark/buttond.conf
[armadillo ~]# rc-service buttond restart
```

9.8.5. Armadillo Base OS 側の起動スクリプト

起動時に何かスクリプトを走らせるためにはコンテナとして実行することを推奨します。 「9.2.1. コンテナの自動起動」 を参照してください。

コンテナで実行不可能な場合に、「local」サービスを使うことができます: /etc/local.d ディレクトリに.start ファイルを置いておくと起動時に実行されて、 .stop ファイルは終了時に実行されます。

```
[armadillo ~]# vi /etc/local.d/date_test.start ❶
#!/bin/sh

date > /tmp/boottest
[armadillo ~]# chmod +x /etc/local.d/date_test.start ❷
[armadillo ~]# persist_file /etc/local.d/date_test.start ❸
[armadillo ~]# reboot
: (省略)
[armadillo ~]# cat /tmp/boottest ❹
Tue Mar 22 16:36:12 JST 2022
```

図 9.143 local サービスの実行例

❶ スクリプトを作ります。

❷ スクリプトを実行可能にします。

❸ スクリプトを保存して、再起動します。

❹ 実行されたことを確認します。

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

Armadillo Base OS では chronyd を使っています。

デフォルトの設定（使用するサーバーなど）は /etc/chrony/conf.d/ にあり、変更用に /etc/atmark/chrony.conf.d/ のファイルも読み込みます。/etc/atmark/chrony.conf.d ディレクトリに /etc/chrony/conf.d/ と同じファイル名の設定ファイルを置いておくことで、デフォルトのファイルを読まないようになります。

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

```
[armadillo ~]# vi /etc/atmark/chrony.conf.d/servers.conf ❶
pool my.ntp.server iburst
[armadillo ~]# persist_file /etc/atmark/chrony.conf.d/servers.conf ❷
[armadillo ~]# rc-service chronyd restart ❸
chronyd          | * Stopping chronyd ... [ ok ]
chronyd          | * Starting chronyd ... [ ok ]
armadillo:~# chronyc sources ❹
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
^? my.ntp.server                 1   6     3     2    +88ms[  +88ms] +/-  173ms
```

図 9.144 chronyd のコンフィグの変更例

❶ コンフィグファイルを作ります。

❷ ファイルを保存します

❸ chronyd サービスを再起動します。

❹ chronyc で新しいサーバーが使用されていることを確認します。

9.9. Device Tree をカスタマイズする

at-dtweb を利用して Device Tree をカスタマイズする方法を説明します。at-dtweb では、Web ブラウザ上のマウス操作で DTS および DTB を生成することができます。カスタマイズの対象は拡張インターフェース(CON11、CON12)です。

9.9.1. at-dtweb のインストール

ATDE9 に at-dtweb パッケージをインストールします。

```
[ATDE ~]$ sudo apt update
[ATDE ~]$ sudo apt install at-dtweb
```

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

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

9.9.2. at-dtweb の起動

1. at-dtweb の起動開始

   at-dtweb の起動を開始するには、デスクトップ左上のアプリケーションの「システムツール」から「at-dtweb」を選択してください。

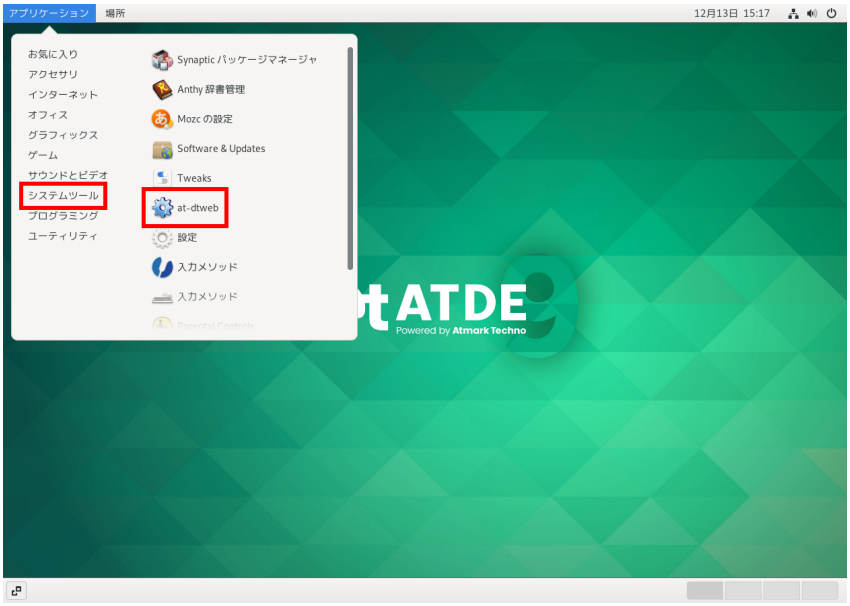


図 9.145 at-dtweb の起動開始

コマンドライン上からでも、at-dtweb コマンドで起動できます。

```
[ATDE ~]$ at-dtweb
```

1. ボードの選択

   ボードを選択します。Armadillo-IoT_G4 を選択して、「OK」をクリックします。

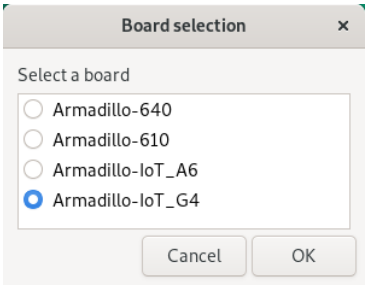


図 9.146 ボード選択画面

2. Linux カーネルディレクトリの選択

   Linux カーネルディレクトリを選択します。コンフィギュレーション済みの Linux カーネルディレクトリを選択して、「OK」をクリックします。

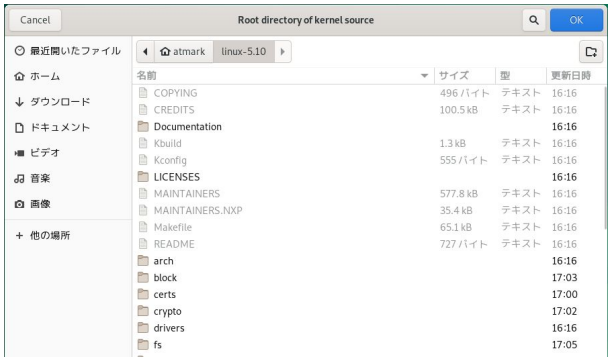


図 9.147 Linux カーネルディレクトリ選択画面

3. at-dtweb の起動完了

   at-dtweb が起動し、次のように画面が表示されます。

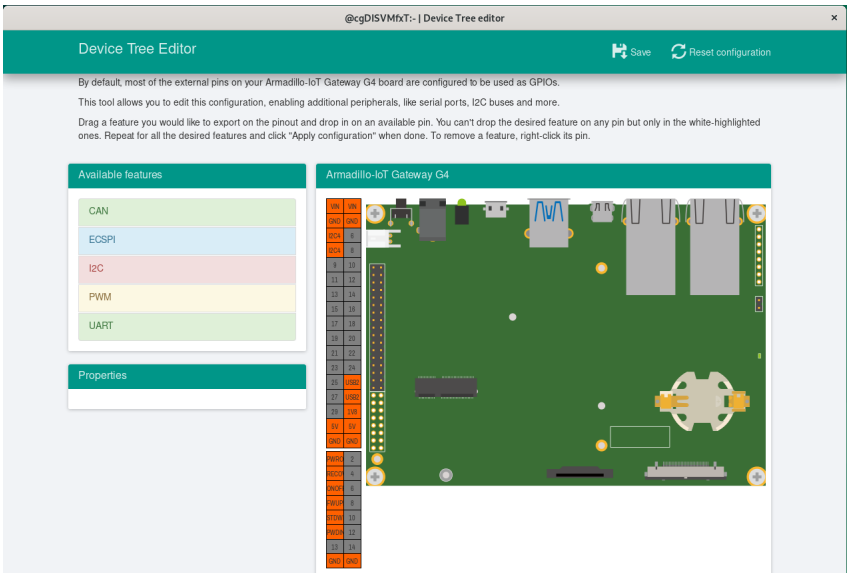


図 9.148 at-dtweb 起動画面

Linux カーネルは、事前にコンフィギュレーションされている必要があります。コンフィギュレーションの手順については「9.4. Armadillo のソフトウェアをビルドする」を参照してください。

9.9.3. Device Tree をカスタマイズ

9.9.3.1. 機能の選択

機能の選択は、ドラッグ&ドロップで行います。画面左上の「Available features」から有効にしたい機能をドラッグし、画面右側の「Armadillo-IoT Gateway G4」の白色に変化したピンにドロップします。例として CON11 8/10 ピンを UART3(RXD/TXD) に設定します。

何も機能が選択されていないピンには GPIO の機能が割り当てられます。

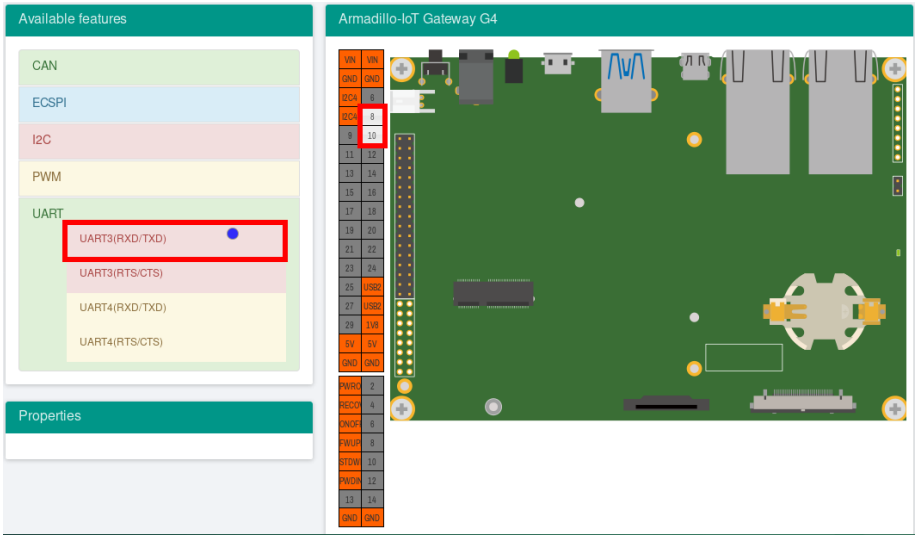


図 9.149 UART3(RXD/TXD) のドラッグ

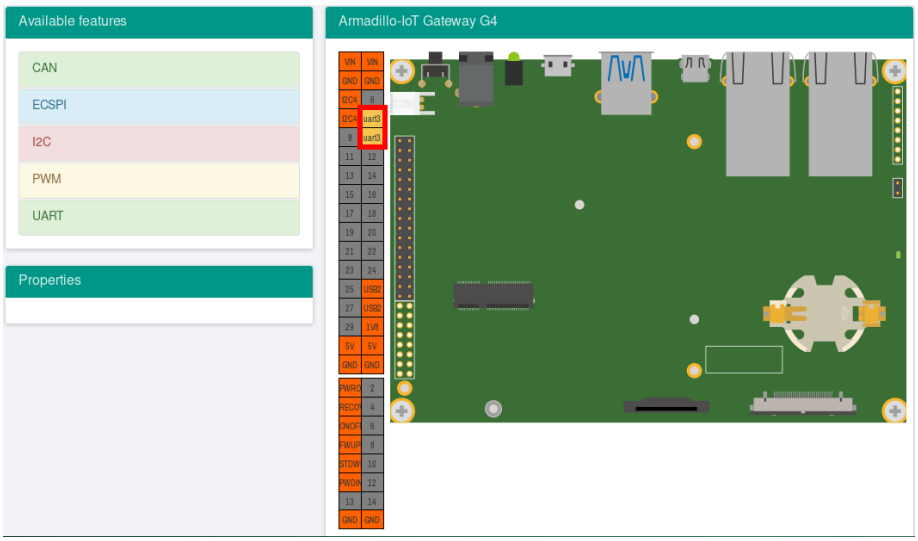


図 9.150 CON11 8/10 ピンへのドロップ

9.9.3.2. 信号名の確認

画面右側の「Armadillo-IoT Gateway G4」にドロップして設定したピンを左クリックすると信号名が表示されます。どのピンがどの信号に対応しているのかを確認することができます。

例として UART3(RXD/TXD) の信号名を確認します。

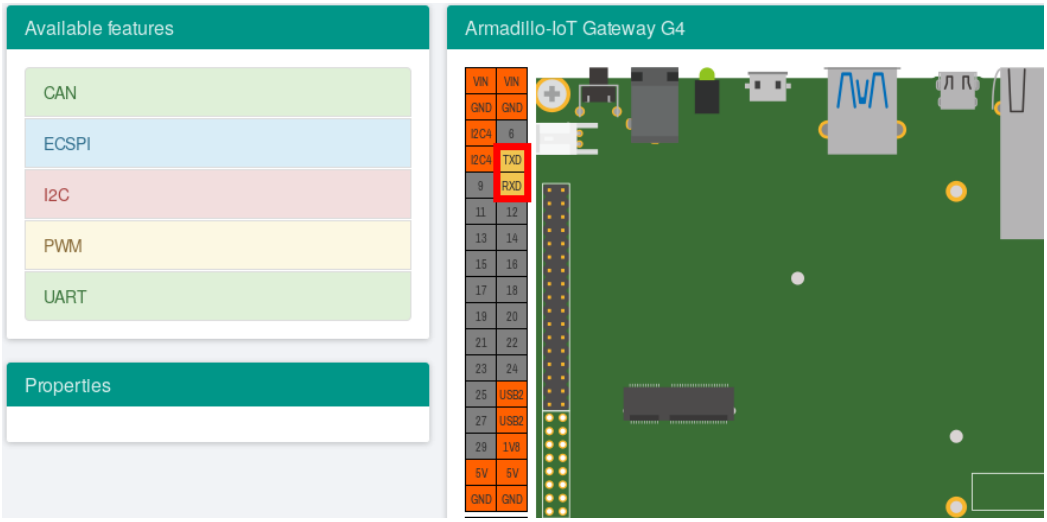


図 9.151 信号名の確認

再度ピンを左クリックすると機能名の表示に戻ります。

9.9.3.3. プロパティの設定

いくつかの機能にプロパティを設定することができます。画面右側の「Armadillo-IoT Gateway G4」に選択した機能を左クリックすると、画面左下の「Properties」からプロパティを選択することができます。

例として CON11 19/27 ピンの I2C5(SCL/SDA) の clock_frequency プロパティを設定します。

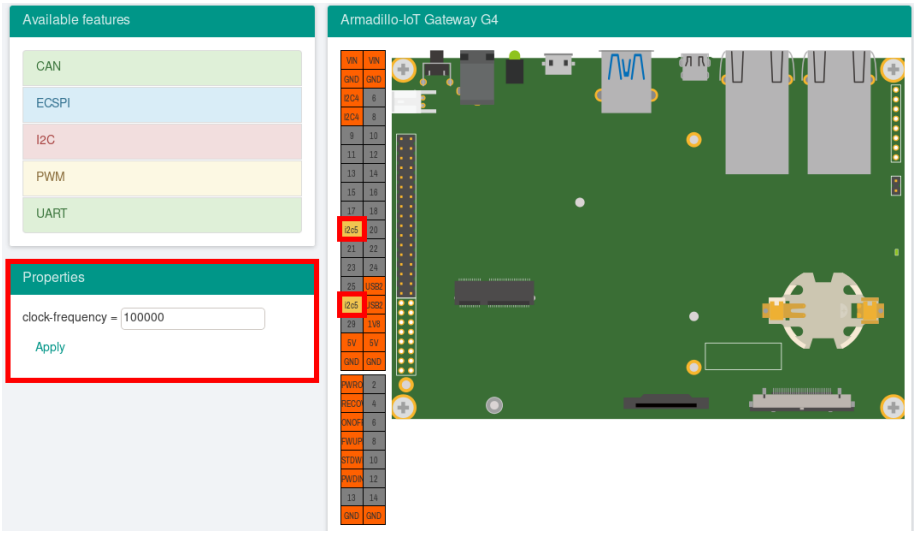


図 9.152 プロパティの設定

設定したプロパティを確定させるには「Apply」をクリックします。

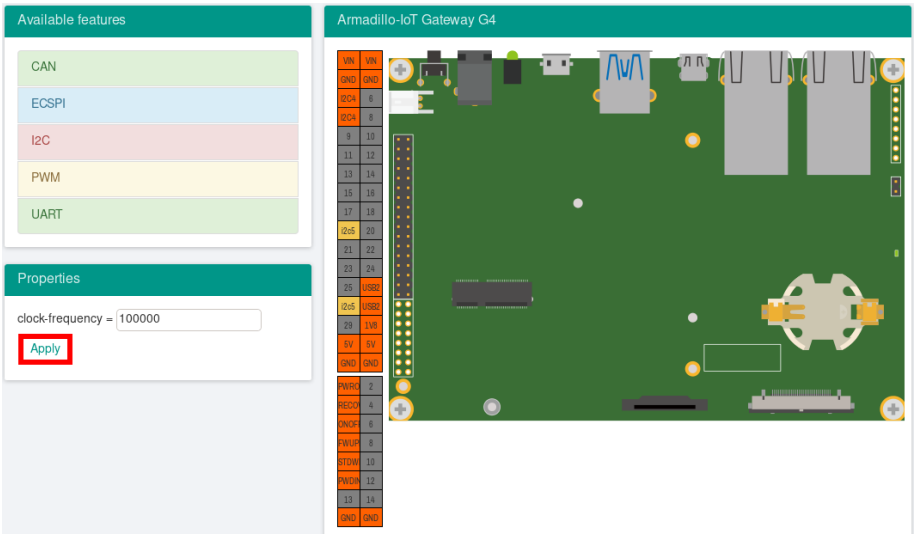


図 9.153 プロパティの保存

9.9.3.4. 機能の削除

全ての機能を削除する場合は、画面右上の「Reset configuration」をクリックします。機能ごとに削除する場合は、画面右側の「Armadillo-IoT Gateway G4」のピンを右クリックして「Remove」をクリックします。

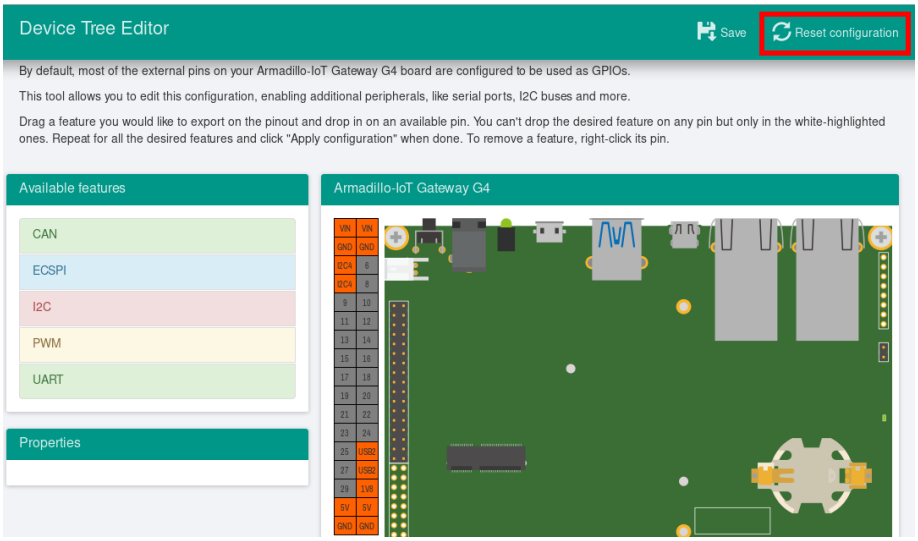


図 9.154 全ての機能の削除

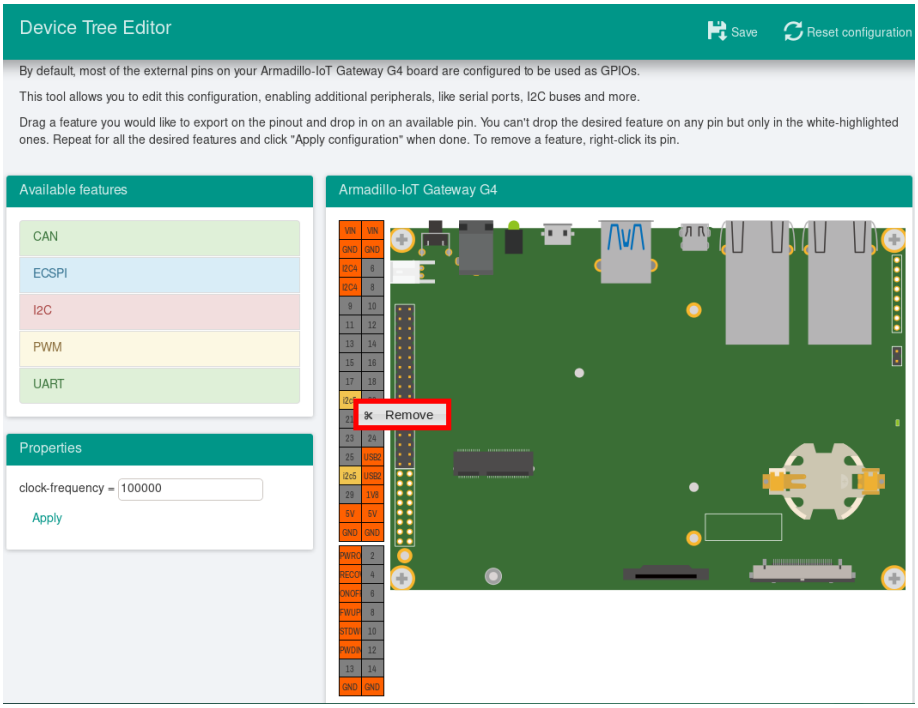


図 9.155 I2C5(SCL/SDA) の削除

9.9.3.5. DTS/DTB の生成

DTS および DTB を生成するには、画面右上の「Save」をクリックします。

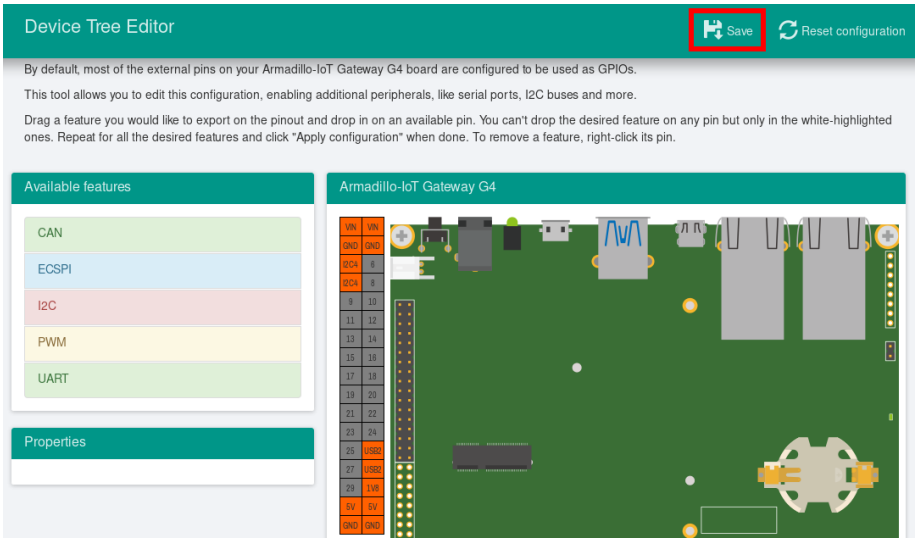


図 9.156 DTS/DTB の生成

「Device tree built!」と表示されると、DTS および DTB の生成は完了です。

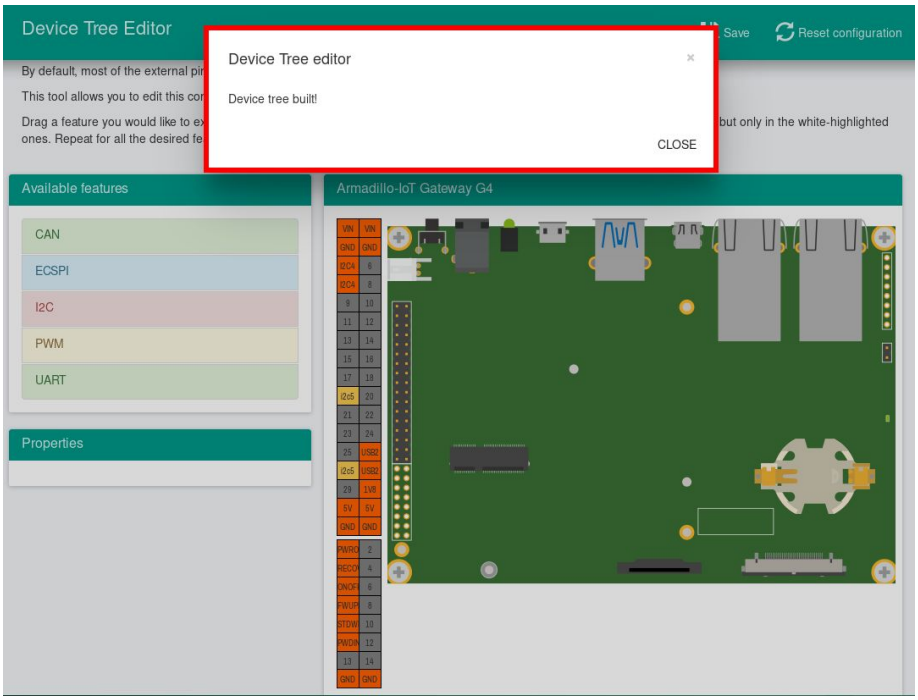


図 9.157 DTS/DTB の生成完了

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

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

9.9.4. DTS overlays によるカスタマイズ

Device Tree は「DTS overlay」(dtbo) を使用することでも変更できます。

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 ~]# vi /boot/overlays.txt ❶
fdt_overlays=armadillo_iotg_g4-nousb.dtbo armadillo_iotg_g4-sw1-wakeup.dtbo

[armadillo ~]# persist_file -vp /boot/overlays.txt ❷
'/boot/overlays.txt' -> '/mnt/boot/overlays.txt'
Added "/boot/overlays.txt" to /etc/swupdate_preserve_files

[armadillo ~]# reboot ❸
```

```
:（省略）
Applying fdt overlay: armadillo_iotg_g4-nousb.dtbo ❹
Applying fdt overlay: armadillo_iotg_g4-sw1-wakeup.dtbo
:（省略）

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

図 9.158 /boot/overlays.txt の変更例

❶ /boot/overlays.txt ファイルに「armadillo_iotg_g4-sw1-wakeup.dtbo」を追加します。ファイルが存在しない場合は新規に作成してください。このファイルの詳細については 「9.9.4. DTS overlays によるカスタマイズ」 を参照してください。

❷ /boot/overlays.txt を保存し、アップデートの場合でも保存します。

❸ overlay の実行のために再起動します。

❹ シリアルコンソールの場合に、u-boot によるメッセージを確認できます。

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

❻ sw1-wakeup も有効になっていることを確認できます。

9.9.4.1. 提供している DTS overlay

以下の DTS overlay を用意しています：

- armadillo_iotg_g4-nousb.dtbo: USB の電源を切ります。
- armadillo_iotg_g4-sw1-wakeup.dtbo: SW1 の起床要因を有効にします。
- armadillo_iotg_g4-con10-arducam.dtbo: arducam カメラを MIPI-DSI で接続する場合にご使用ください。
- armadillo_iotg_g4-con10-imx219.dtbo: Raspberry Pi 向けの imx219 カメラを MIPI-DSI で接続する場合にご使用ください。
- armadillo_iotg_g4-con10-ox01f10.dtbo: OMNIVISION の OX01F10 カメラを MIPI-DSI で接続する場合にご使用ください。
- armadillo_iotg_g4-lte-ext-board.dtbo: LTE 拡張ボードで自動的に使います。

9.9.4.2. カスタマイズした DTS overlay の作成

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

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

1. 「9.4.2. Linux カーネルをビルドする」 を参照して、カーネルのソースコードを取得します。
2. ソースディレクトリの arch/arm64/boot/dts/freescale/armadillo_iotg_g4-customize.dts を編集します。

3. make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- dtbs で DTS overlay をビルドします。
4. arch/arm64/boot/dts/freescale/armadillo_iotg_g4-customize.dtbo ファイルを Armadillo の /boot に配置し、/boot/overlays.txt に記載します。

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

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

#include "imx8mp-pinfunc.h"

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

&iomuxc {
        pinctrl_pwm2: pwm2grp {
                fsl,pins = <
                        MX8MP_IOMUXC_SPDIF_RX__PWM2_OUT  0x186
                >;
        };
};
[PC ~/linux-[VERSION]]$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- dtbs ❸
: (省略)
  DTC     arch/arm64/boot/dts/freescale/armadillo_iotg_g4-customize.dtbo
: (省略)
[PC ~/linux-[VERSION]]$ scp ¥
    arch/arm64/boot/dts/freescale/armadillo_iotg_g4-customize.dtbo ¥
    armadillo:/boot ❹
armadillo_iotg_g4-customize.dtbo          100%  551   207.5KB/s   00:00
[armadillo ~]# cd /boot
[armadillo /boot]# vi /boot/overlays.txt ❺
fdt_overlays=armadillo_iotg_g4-customize.dtbo
[armadillo /boot]# persist_file -vp overlays.txt ¥
                                  armadillo_iotg_g4-customize.dtbo ❻
'/boot/overlays.txt' -> '/mnt/boot/overlays.txt'
'/boot/armadillo_iotg_g4-customize.dtbo' -> '/mnt/boot/armadillo_iotg_g4-customize.dtbo'
Added "/boot/armadillo_iotg_g4-customize.dtbo" to /etc/swupdate_preserve_files
[armadillo /boot]# reboot ❼
: (省略)
Applying fdt overlay: armadillo_iotg_g4-customize.dtbo
```

図 9.159 DTS overlay を作成する例

❶ 取得したカーネルのソースディレクトリに入ります。

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

❸ DTS overlay をビルドします。

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

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

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

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

9.10. eMMC のデータリテンション

eMMC は主に NAND Flash メモリから構成されるデバイスです。NAND Flash メモリには書き込みしてから 1 年から 3 年程度の長期間データが読み出されないと電荷が抜けてしまう可能性があります。その際、電荷が抜けて正しくデータが読めない場合は、eMMC 内部で ECC (Error Correcting Code) を利用してデータを訂正します。しかし、訂正ができないほどにデータが化けてしまう場合もあります。そのため、一度書いてから長期間利用しない、高温の環境で利用するなどのケースでは、データ保持期間内に電荷の補充が必要になります。電荷の補充にはデータの読み出し処理を実行し、このデータの読み出し処理をデータリテンションと呼びます。

Armadillo-IoT ゲートウェイ G4 に搭載の eMMC には長期間データが読み出されない状態であっても、データリテンションを自動的に行う機能を搭載しています。

詳しい仕様については 「9.10.3. 実装仕様に関する技術情報」 を参照してください。

9.10.1. データリテンションの設定

データリテンションは /etc/conf.d/micron_emmc_reten というファイルに書かれた設定、use_system_time によって以下の２通りの挙動を示します。

表 9.7 データリテンションの挙動

/etc/conf.d/micron_emmc_reten	initiating condition
use_system_time=yes	Linux 起動した時に前回のリテンションから 1 日以上経過していたら開始する
use_system_time=no (default)	Linux 起動した時に毎回開始する

これで設定は完了しました。

以下は挙動ごとのシステム概略図です。

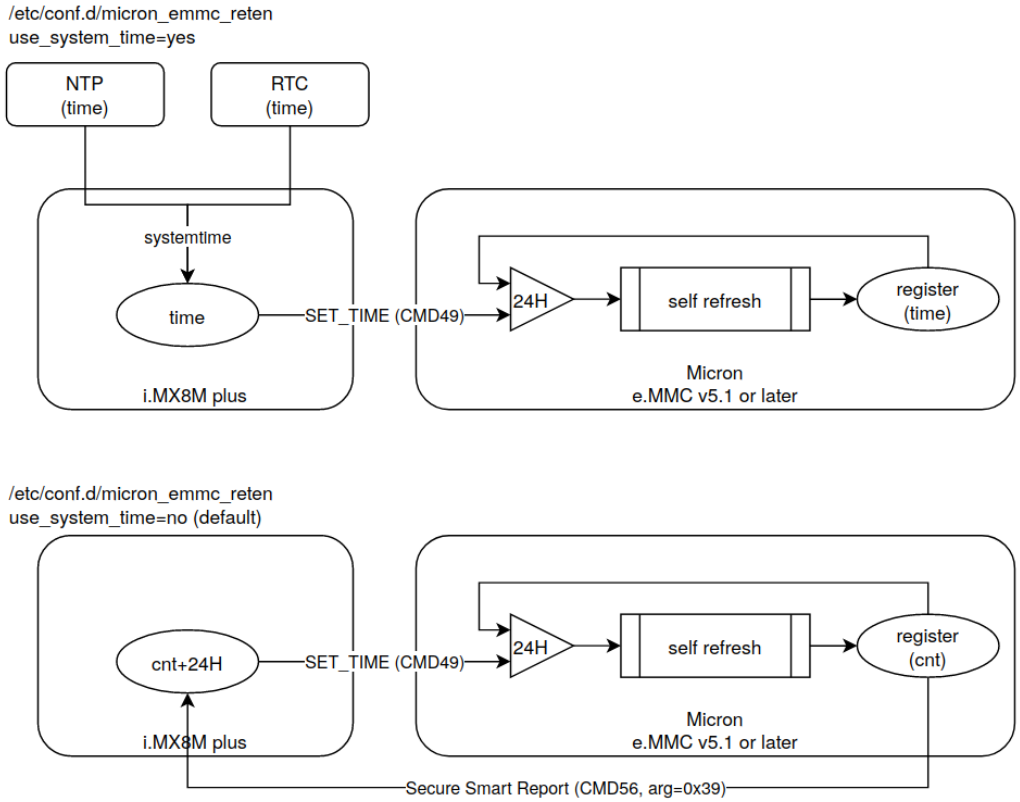


図 9.160 データリテンション開始トリガーの方式

use_system_time を有効にした場合のデータリテンションの動作例を以下に示します。

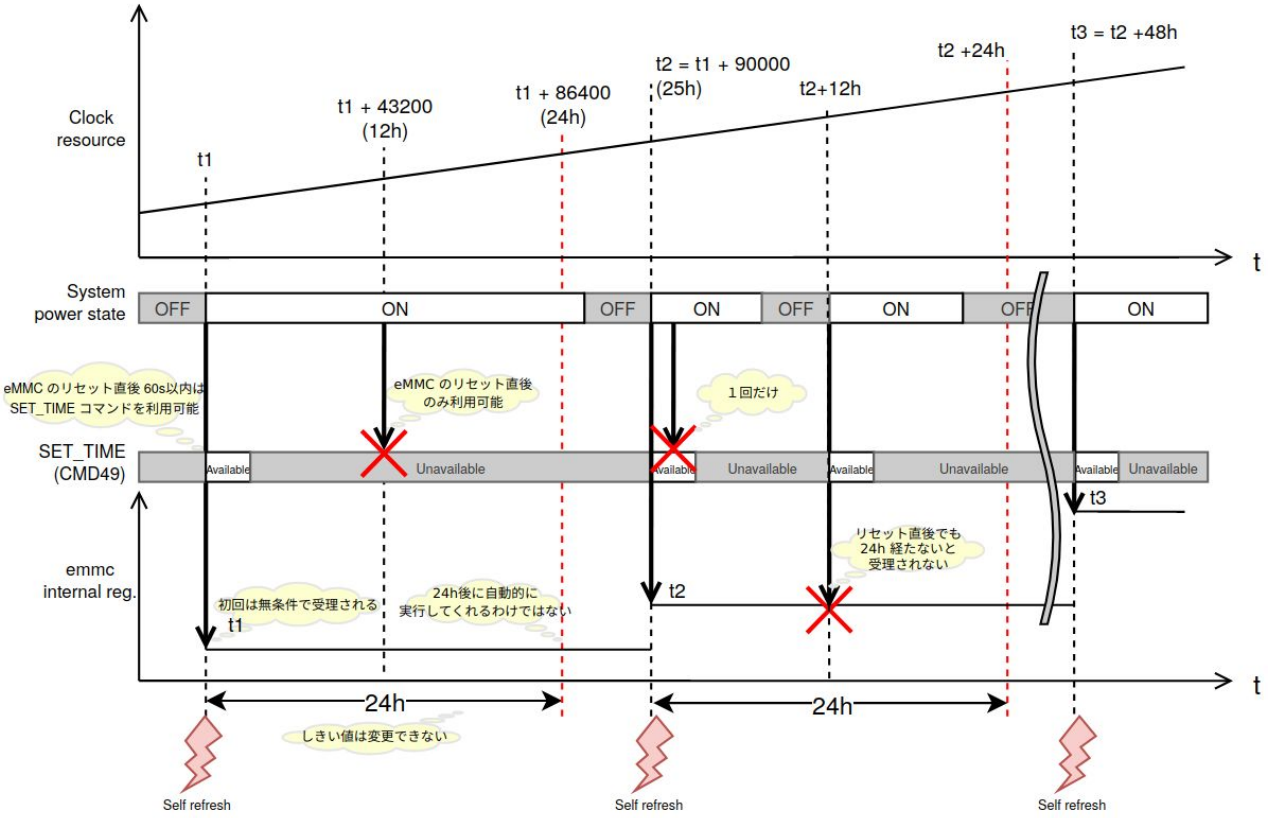


図 9.161 データリテンションの開始トリガーの動作例

9.10.2. より詳しくデータリテンションの統計情報を確認するには

Micron Technology が提供する emmcparm というツールを使うことで、データリテンションの統計情報を確認することができます。統計情報として eMMC 内部に保存されているのは実行回数、最終実行完了時のカウンター値、現在のデータリテンション処理の進捗があります。次の手順で、emmcparm を使って eMMC の情報を確認することができます。このツールではデータリテンション処理のことを「セルフリフレッシュ」と呼びます。

1. emmcparm をダウンロードする

   以下の検索結果から最新の emmcparm をダウンロードする。ユーザー登録が必要になります。

   emmcparm https://jp.micron.com/search-results?searchRequest=%7B%22term%22%3A%22emmcparm%20%22%7D

   マニュアル作成時点では 5.0.0 を利用しました

2. パッケージを展開する

```
[armadillo ~]# unzip emmc_emmcparm_c_code_derived_from_TN¥ FC¥ 25_v5.0.0 _binary.zip
```

3. SSR を取得する

```
[armadillo ~]# emmcparm/bin/emmcparm_arm_64bit -r /dev/mmcblk2
```

```
:（省略）
=========================================================================
|                          Secure Smart Report                          |
=========================================================================
  Self Refresh progress of scan[215-212]:              0x00000000 (0) ❶
  Power Loss Counter[195-192]:                         0x00000005 (5)
  Current total ON time[131-128]:                      0x00001b28 (6952)
  Number of Blocks in Refresh Queue[99-96]:            0x00000000 (0)
  Self Refresh Completion date [95-88]:                0xfffffffffd8148931 ❷
                                                       (-669742799)
  Self Refresh Loop Count[81-80]:                      0x00000002 (2)  ❸
  Written Data 100MB Size Count (from NAND)[79-76]:    0x0004889d (297117)
  Cumulative Initialization Count (from NAND)[75-72]:  0x00005300 (21248)
  Written Data 100MB Size Count (from RAM)[71-68]:     0x0004889d (297117)
  Refresh Count[55-52]:                                0x00000004 (4)
:（省略）
```

❶ 現在のセルフリフレッシュ処理の進捗。0 ということは実行中ではない

❷ 最後に行ったセルフリフレッシュのカウンター値

❸ セルフリフレッシュを行った回数

9.10.3. 実装仕様に関する技術情報

ここではデータリテンションを自動的におこなう機能の仕様について詳細に説明します。Armadillo で採用している eMMC には、データリテンションを自動的に実行することができる「セルフリフレッシュ」と呼ばれる機能が搭載されます。実行トリガーは 2 種類のうちどちらかを選択できます。OTP のため一度設定すると変更できません。この設定は出荷時に「eMMC 内部レジスタ値とコマンドに入力された値を比較して 1 日以上経過していると実行する」を設定しています。

1. リセット後に毎回実行する
2. eMMC 内部レジスタ値とコマンドに入力された値を比較して 1 日以上経過していると実行する

2 の設定の場合、セルリフレッシュ機能が実行されるまでの流れは以下のとおりです。

1. ホストによって eMMC がハードウェアもしくはソフトウェアリセットされる
2. 一定時間 (delay 1) 以内に、ホストから SET_TIME (CMD49)と呼ばれるコマンドが eMMC に発行される
3. eMMC コントローラは、バスの稼動状態を監視する
4. eMMC コントローラは、アイドルになってから一定時間 (delay 2) 経過した後にセルフリフレッシュを実行する

・ ECC エラーなどのエラーがしきい値 (2) を越えたセルに対してのみセルフリフレッシュを実行する

Armadillo でのセルフリフレッシュ機能搭載 eMMC への設定は以下のとおりです。

表 9.8 Armadillo のデータリテンションの設定

setting	value	description
RTC	ON	eMMC 内部レジスタの値と SET_TIME の値を比較してセルフリフレッシュを実行する
Delay 1	60s	リセット後の SET_TIME 有効期間
Delay 2	100ms	アイドル確認後のセルフリフレッシュ実行までの遅れ時間

詳しい情報は以下を参照してください。

Refresh Features for Micron e.MMC Automotive 5.1 Devices
https://jp.micron.com/search-results?searchRequest=%7B%22term%22%3A%22TN-FC-60%22%7D

マイクロンのサイトの会員登録が必要になります。

9.11. デモアプリケーションを実行する

この章では、アットマークテクノが提供するデモアプリケーションについて説明します。デモアプリケーションは GUI アプリケーションであるため、ディスプレイを接続している必要があります。デモアプリケーションを実行するためのコンテナイメージとして、アットマークテクノが提供するコンテナイメージを想定しています。このイメージに関しては 「9.1.3. アットマークテクノが提供するイメージを使う」 を参照してください。

9.11.1. コンテナを作成する

デモアプリケーションを実行するためのコンテナを以下のように作成します。ここでは 「8.3. VPU や NPU を使用する」 をすでに実行済みであるとします。

```
[armadillo ~]# podman run -it --name=demo-app ¥
--env=XDG_RUNTIME_DIR=/tmp ¥
--env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu ¥
--env=QT_QPA_PLATFORM=wayland ¥
--volume=/sys:/sys ¥
--volume=/dev:/dev ¥
--volume=/run/udev:/run/udev ¥
--volume=/opt/firmware:/opt/firmware ¥
--privileged ¥
localhost/at-debian-image:latest /bin/bash
[container /]#
```

図 9.162 デモアプリケーションを実行するためのコンテナ作成例

9.11.2. weston を起動する

デモアプリケーションは GUI アプリケーションであるため、コンテナにログイン後、まずデスクトップ環境を起動する必要があります。ここでは weston を起動します。

```
[container /]# weston --tty=1 &
```

図 9.163 weston の起動

--tty=1 のオプションは画面表示に使用する tty の値を設定してください。

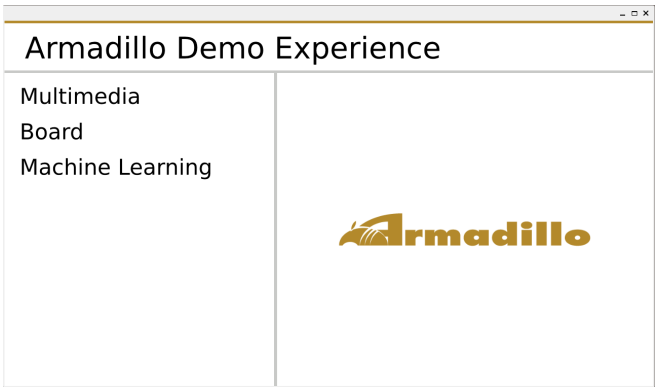
9.11.3. デモアプリケーションランチャを起動する

デモアプリケーションランチャを起動します。個々のデモアプリケーションはこのデモアプリケーションランチャから起動できます。このデモアプリケーションランチャは GUI フレームワークとして Qt を使用しています。デモアプリケーションランチャのソースコードは、apt source で取得することができます。

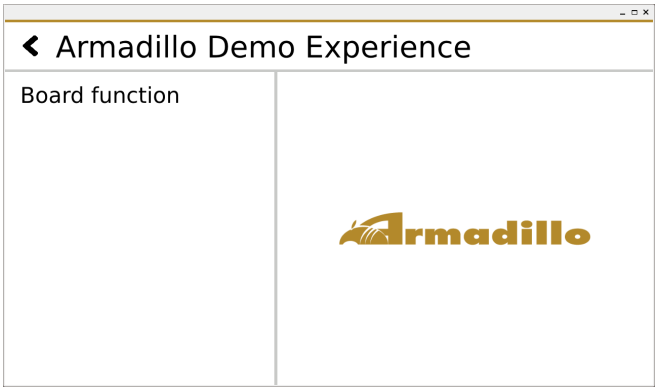
```
[container /]# apt install armadillo-demo-experience
[container /]# demoexperience
```

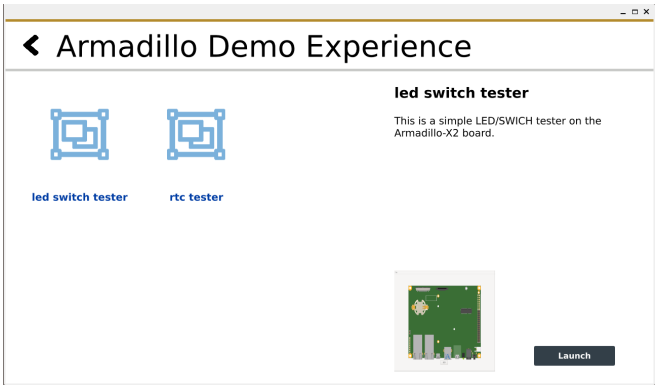
図 9.164 デモアプリケーションランチャの起動

以下のようなアプケーションが起動します。



左側のカテゴリから起動したいデモアプリケーションを選びます。

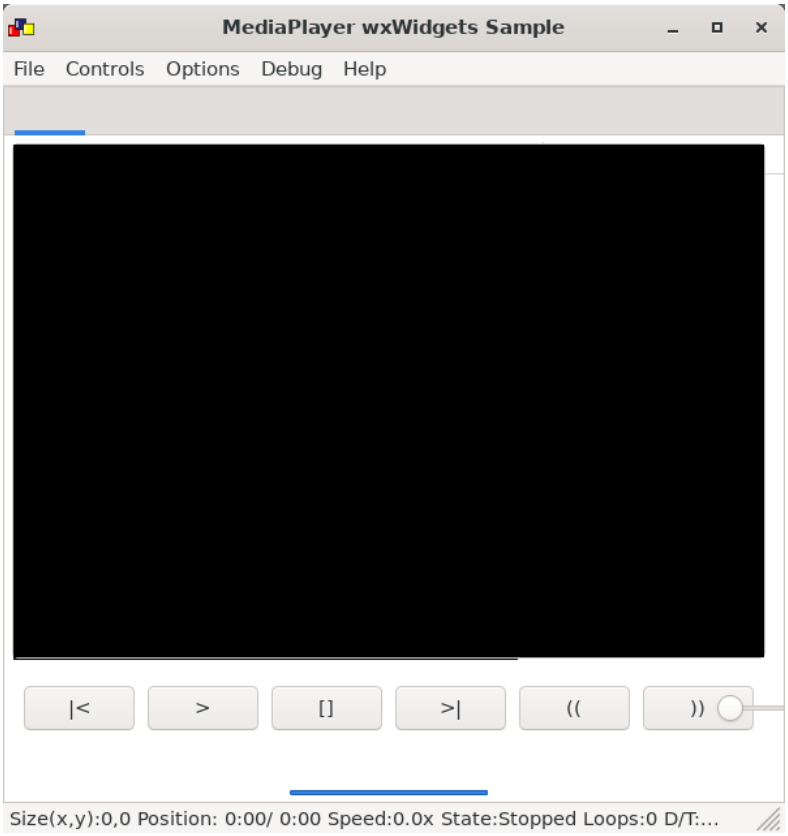




選んだアプリケーションは、右下の Launch ボタンで起動することができます。

9.11.4. mediaplayer

mediaplayer は動画を再生するアプリケーションです。H.264, VP8, VP9 でエンコードされた動画ファイルであれば、動画のデコードに VPU が使われます。File メニューから、再生したい動画ファイルを選択することができます。このアプリケーションは、GUI フレームワークとして wxWidgets を使用しています。



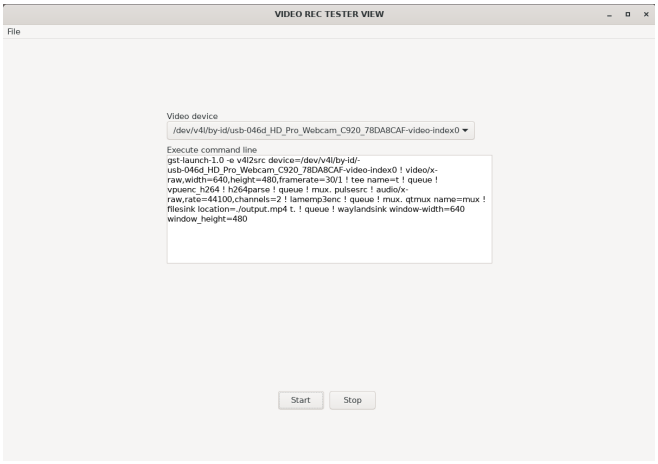
音声も出力したい場合は、pulseaudio をインストールして起動する必要があります。

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

図 9.165 pulseaudio のインストールと起動

9.11.5. video recoder

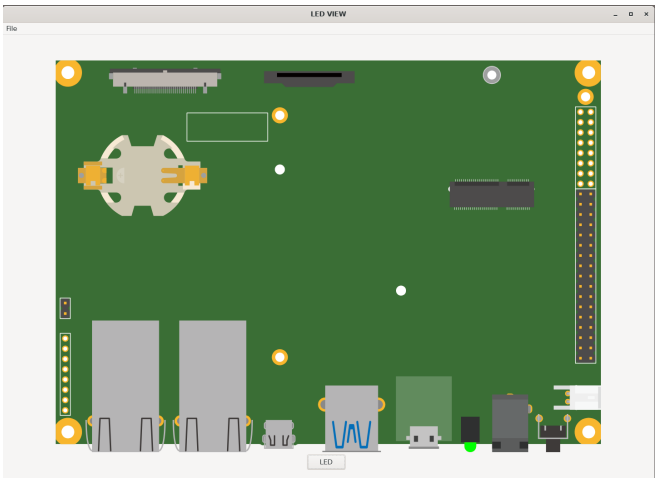
video recoder は gstreamer を使用してカメラからの映像を録画することができます。そのため、このアプリケーションを使用するためには、Armadillo 本体にカメラを接続する必要があります。カメラが接続されていると Video device の項目でカメラを選択できるようになります。カメラを選択し、Start ボタンを押すと別ウィンドウが表示され録画が開始されます。アプリケーション上のテキストボックスには、Start ボタンを押したときに起動する gstreamer のコマンドを表示しています。テキストボックスの内容はキーボードで編集可能です。このアプリケーションは、GUI フレームワークとして wxWidgets を使用しています。



マイク付きのカメラなどで同時に音声も録音したい場合は、「9.11.4. mediaplayer」 を参照して pulseaudio を起動してください。

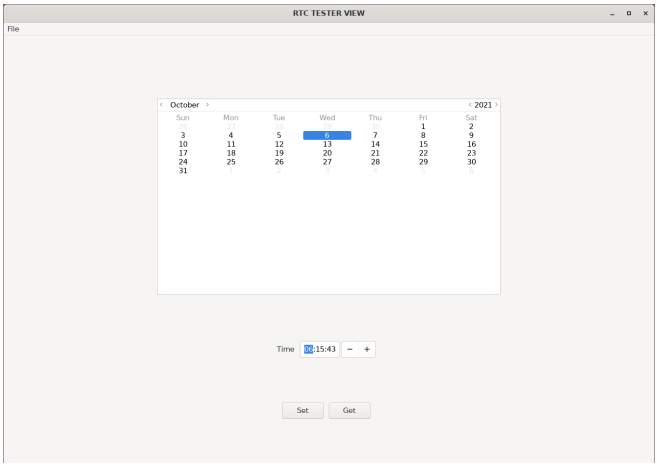
9.11.6. led switch tester

led switch tester は Armadillo 本体上の LED と SW1 を扱うアプリケーションです。LED ボタンを押すことで Armadillo 本体上の LED の 点灯・消灯を確認することができます。Armadillo 本体上の SW1 を押すとアプリケーションの SW1 部分の表示が変化することを確認できます。このアプリケーションは、GUI フレームワークとして wxWidgets を使用しています。



9.11.7. rtc tester

rtc tester は Armadillo 本体上の RTC に対して日時の設定および取得が行えるアプリケーションです。カレンダー上から日付を選び、Time に設定したい時刻を入力した後、Set ボタンを押すと RTC にその日時が設定されます。Get ボタンを押すと、現在の日時を RTC から読み込みアプリケーション上に反映されます。このアプリケーションは、GUI フレームワークとして wxWidgets を使用しています。



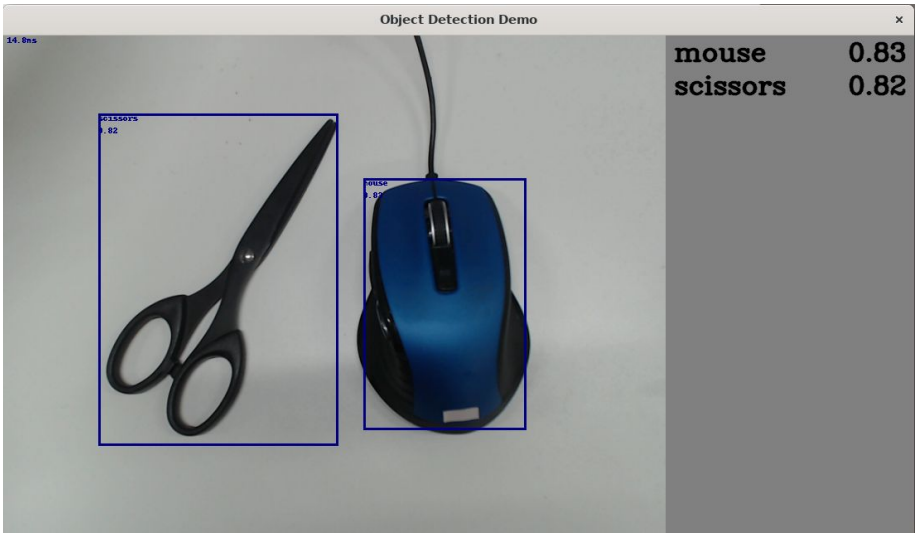
9.11.8. object detection demo

object detection demo はカメラからの映像に対して物体認識を行うアプリケーションです。NPU を使用しているため高速に物体認識を行えます。画面の左側には認識した物体を囲む四角形が表示され、右側には認識した物体のラベルとスコアが表示されます。このアプリケーションは機械学習のライブラリとして TensorFlow Lite を使用しています。

起動する前に、必要な Python ライブラリをインストールする必要があります。

```
[container /]# pip3 install pillow
```

図 9.166 pillow のインストールと起動



このアプリケーションはカメラデバイスとしてデフォルトで /dev/video2 を使用します。お使いの環境によって別のカメラデバイスに設定したい場合は、以下のファイルを変更してください。

```
[container /]# vi /usr/share/armadillo-demo-experience/resources/demos.json
:
: (省略)
:
{"Machine Learning":[{
        "Tensorflow Lite":[{
            "name": "object detection demo",
            "executable": "python3 /usr/share/armadillo-demo-experience/AI-demo/object_detection/
detect_usbcamera.py --model /usr/share/armadillo-demo-experience/AI-demo/object_detection/
detect.tflite --labels /usr/share/armadillo-demo-experience/AI-demo/object_detection/
coco_labels.txt --camera_id 2", ❶
            "compatible": "armadillo-x2",
          "description": "This is a simple object detection aplication that used NPU and TensorFlow
Lite on the Armadillo-X2 board."
        }]
    }]
```

図 9.167 ビデオデバイスの変更

❶ --camera_id の値を環境に合わせて変更します。

9.11.9. pose estimation demo

pose estimation demo はカメラに映った人物の姿勢を推定して表示するアプリケーションです。NPU を使用しているため高速に姿勢推定を行えます。推定した姿勢は人物の上に重ねて表示されます。このアプリケーションは機械学習のライブラリとして TensorFlow Lite を使用しています。

このアプリケーションは起動してから画面に映像が表示されるまで約 1 分ほどかかります。また、カメラデバイスとしてデフォルトで /dev/video2 を使用します。お使いの環境によって別のカメラデバイスに設定したい場合は、以下のファイルを変更してください。

```
[container /]# vi /usr/share/armadillo-demo-experience/resources/demos.json
:
: (省略)
:
{"Machine Learning":[{
:
: (省略)
:
        {
            "name": "pose estimation",
            "executable": "python3 /usr/share/armadillo-demo-experience/AI-demo/pose_estimation/
pose_estimation.py --model /usr/share/armadillo-demo-experience/AI-demo/pose_estimation/
posenet.tflite --camera_id 2", ❶
            "source": "",
            "screenshot": "pose_estimation_demo.png",
            "compatible": "armadillo-x2",
          "description": "This is a simple pose estimation aplication that uses NPU on the Armadillo-
X2 board."
        }]
    }]
```

図 9.168 ビデオデバイスの変更

❶ --camera_id の値を環境に合わせて変更します。

9.11.10. image segmentation demo

image segmentation demo はカメラに映った人物の「人物として認識された領域(セグメント)」を推定して表示するアプリケーションです。NPU を使用しているため高速に領域推定を行えます。推定した領域は人物の上に青の透過色で重ねて表示されます。このアプリケーションは機械学習のライブラリとして TensorFlow Lite を使用しています。

このアプリケーションはカメラデバイスとしてデフォルトで /dev/video2 を使用します。お使いの環境によって別のカメラデバイスに設定したい場合は、以下のファイルを変更してください。

```
[container /]# vi /usr/share/armadillo-demo-experience/resources/demos.json
:
: (省略)
:
{"Machine Learning":[{
:
: (省略)
:
        {
            "name": "image segmentation",
           "executable": "python3 /usr/share/armadillo-demo-experience/AI-demo/image_segmentation/
image_segmentation.py --model /usr/share/armadillo-demo-experience/AI-demo/image_segmentation/
human_segmentation.tflite --camera_id 2", ❶
            "source": "",
            "screenshot": "image_segmentation_demo.png",
            "compatible": "armadillo-x2",
            "description": "This is a simple image segmentation aplication that uses NPU on the
Armadillo-X2 board."
        }]
    }]
```

図 9.169 ビデオデバイスの変更

❶ --camera_id の値を環境に合わせて変更します。

10. 動作ログ

10.1. 動作ログについて

Armadillo-IoT ゲートウェイ G4 ではシステムが出力するログの一部は、一般的な /var/log ディレクトリではなく、/var/at-log ディレクトリに出力されます。/var/at-log は、ルートファイルシステムとは別のパーティションになっているので、ルートファイルシステムに障害が発生した場合でも、/var/at-log のパーティションが無事であれば、ログファイルを取り出して、不具合等の解析に利用することができます。

10.2. 動作ログを取り出す

ログファイルは /var/at-log ディレクトリ内に atlog というファイル名で作成されているので、これを任意のディレクトリにコピーすることで取り出せます。もし、eMMC 上のルートファイルシステムが壊れてしまい起動できない場合は、microSD カードから起動することでログファイルを取り出すことができます。

10.3. ログファイルのフォーマット

ログファイルの内容はテキストデータであり、以下のようなフォーマットになっています。

```
日時 armadillo ログレベル 機能: メッセージ
```

図 10.1 動作ログのフォーマット

10.4. ログ用パーティションについて

ログ出力先である /var/at-log ディレクトリには、GPP である /dev/mmcblk2gp1 パーティションがマウントされています。このパーティションに論理的な障害が発生した場合は、/dev/mmcblk2gp1 のデータを /dev/mmcblk2gp2 にコピーし、/dev/mmcblk2gp1 は FAT ファイルシステムでフォーマットされます。このパーティションの障害チェックはシステム起動時に自動的に実行されます。

11. セキュリティ

この章では、Armadillo を利用したシステム上に存在する情報資産を守るための方法について説明する。

11.1. セキュリティの費用対効果

製品開発に費やすコストには様々なものがありますが、セキュリティのような機能性につながらないコストは軽視されがちです。しかし、一旦セキュリティ問題が発生してしまうと、業務停止など直接的な影響だけでなく、社会的な信用の低下、損害賠償など様々な被害につながってしまう可能性があります。守るべき情報資産が明確に存在するのであれば、セキュリティは重要視されるべきです。しかし、コスト度外視でありったけの対策を盛り込むのではコストが膨らみます。適切かつ適度なセキュリティ対策を講じることが大切です。製品開発の早い段階で守るべき情報資産を認識し、それら情報資産に対する脅威とリスクを分析して、リスクを評価する作業を行うことをお勧めします。そのうえで、費用対効果に見合ったセキュリティ技術を選択するとよいでしょう。

11.2. Armadillo Base OS のセキュリティ

幅広い利用者のユースケースに沿った製品セキュリティ実現のため、Armadillo Base OS に様々なセキュリティ技術を組み込むことが可能です。

- ソフトウェア実行環境の保護
  - TEE (Trusted Execution Environment)
- セキュアブート、チェーンオブトラスト
  - HAB (High Assurance Boot)
  - SE050 secure storage
- ストレージの秘匿化
  - パーティション/ファイシステムの暗号化
- キーストレージ
  - SE050
- メモリの完全性チェック
  - RTIC (Run-Time Integrity Checker)
- 暗号処理のアクセラレータ
  - CAAM (Cryptographic Acceleration and Assurance Module)

次に各セキュリティ技術のカバーする範囲を示します。セキュリティ技術を採用する際には、採用する技術によってどこまでの範囲が守られるのかを把握することが大切です。設定次第でそれぞれの技術は高いセキュリティ性能を実現するものになり得ますが、単独で利用するだけではその効果は限定的で回避可能なものになってしまいます。そのため、いくつかの技術を適切に組み合わせて利用することで、

より広範囲をカバーする回避困難なセキュリティを実現できます。そして、カバーする範囲が重なりあうことで突破困難なセキュリティを実現します。

技術 / 要件	OP-TEE	ストレージ暗号化	セキュアFWアップデート(SWUpdate)	アプリケーションの難読化	SE050セキュアエレメント	i.MX8M Plus セキュアブート(HAB)	i.MX8M Plus セキュリティ機能(CAAM, SNVS)	JTAG ポートの無効化と利用認証
ファームウェアのハッキング対策								
正規ファームウエア以外を起動させない						✓		✓
ファームウエアを改竄させない		✓		✓		✓		✓
アプリケーションの実行環境を守る	✓							✓
不正なファームウェアを書き込ませない			✓					✓
データのハッキング対策								
ストレージから抜かせない、盗聴させない		✓			✓		✓	✓
RAMから抜かせない、盗聴させない	✓				✓		✓ *1	✓
FWアップデートから抜かせない、盗聴させない	✓		✓					✓
証明書や鍵のハッキング対策								
ストレージから抜かせない、盗聴させない					✓		✓	✓
事後対策								
インシデント発生時に鍵をリボークできる						✓		
インシデント発生時に鍵を変更する	✓	✓	✓		✓	✓		

図 11.1 セキュリティ技術のカバーする範囲

*1 i.MX 8M Plus には RTIC (Run-time Integrity Checker) が含まれます。この機能を利用することでブート時、ランタイム時の周辺メモリの完全性をチェックすることが可能です

11.2.1. モデルユースケース

Armadillo は様々な製品、様々な環境で利用されることが想定されます。それぞれのケースによって考慮すべき脅威が存在します。それらを正しく把握してセキュリティ技術を選択、組み合わせることが大切です。 組み合わせの例としていくつかのモデルユースケースを示します。

- デフォルト
  - 方針
    - ネットワークからの侵入の保護
  - 技術
    - 「9.1. アプリケーションをコンテナで実行する」 を参考にしてください
- クラウドサービスのデバイス認証が可能なレベル
  - 方針
    - ネットワークを突破されても情報資産は守る
    - 部分的な情報資産(暗号処理や鍵)の保護
    - デバイスへの直接攻撃は想定しない
  - 技術

    - ＋ OP-TEE
    - ＋鍵のストレージは SE050
    - ＋ファイルシステムの暗号化
- 決済処理が可能なレベル
  - 方針
    - ネットワークを突破されても情報資産は守る
    - スクリプトキディによるデバイスへの直接攻撃は守るが、ラボレベルの攻撃は想定しない
  - 技術
    - ＋セキュアブート
    - ＋チェーンオブトラストを使った鍵のストレージ
    - ＋パーティションの暗号化
    - ＋メモリの完全性チェック

11.3. データの保護

この章では情報資産を保護するための方法を説明します。説明するセキュリティ技術は以下のとおり。

- SE050 セキュアストレージ

11.3.1. データの保護

情報資産はデータとして永続的にストレージや一次的にＲＡＭ上に保存されます。セキュリティ対策が講じられていないデバイスでは情報資産が露出していることになり、改竄や盗聴につながる可能性があります。 鍵や証明書をストレージに保存する、暗号処理のために一次的に鍵や証明書をＲＡＭ上に展開するケースでデータ保護を採用することが考えられます。

11.3.2. SE050 とは

NXP Semiconductors の EdgeLock SE050 は IoT アプリケーション向けのセキュアエレメントです。様々なアルゴリズムに対応した暗号エンジン、セキュアストレージを搭載します。GlobalPlatform が規定する Secure Channel Protocol 03 に準拠し、バスレベル暗号化 (AES)、ホストとカードの相互認証 (CMAC ベース) を行います。

SE050 の詳細については以下の NXP Semiconductors のページから検索して、ご確認ください。

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

11.3.3. ビルド環境を構築する

1. Plug and trust ミドルウェアをダウンロードします

   以下のサイトからダウンロードします。ここではユーザーが NXP Semiconductors のサイトで最新版を検索してダウンロードすることを想定しています。

   plug and trust middleware https://www.nxp.com/search?keyword=EdgeLock%20SE05x%20Plug%20&%20Trust%20Middleware

   コンテナ上から参照できる位置に配置しておいてください。

   - ダウンロードには NXP Semiconductors サイトへのユーザー登録が必要になります
   - 本マニュアル作成時点では 04.00.00 を利用しました

2. コンテナを立ち上げる

   NXP Semiconductors のビルド環境はターゲットボード上のコンテナになります。ここでは alpine を利用します。

```
[armadillo ~]# podman run -it --name=dev_se050 --device=/dev/i2c-2 ¥
        -v "$(pwd)":/mnt docker.io/alpine /bin/sh
```

3. ビルド環境を構築する

   plug and trust をビルドするために必要なパッケージをインストールします。

```
[container ~]# apk add musl-dev gcc make g++ file ¥
        linux-headers openssl-dev cmake python3
```

11.3.3.1. SE050 を有効にする

Armadillo は消費電力の削減のため SE050 を Deep Power-down モードに設定してパワーゲーティングしている。Deep Power-down モードを解除して SE050 を利用するためには、i.MX 8M Plus に接続されている SE050 の ENA ピンをアサートする必要があります。

表 11.1 SE050 ena pin

SE050 PIN	i.MX8MP port	initial port status
ENA	GPIO1_IO12	GPIO input

gpioset コマンドを利用して GPIO1_IO12 を出力ポートに変更して high にする。

```
[armadillo ~]# persist_file -a add libgpiod
[armadillo ~]# gpioset gpiochip0 12=1
```

11.3.3.2. Plug and trust ミドルウェアをビルドする

1. Plug and trust ミドルウェアを展開します

```
[container /mnt]# unzip SE-PLUG-TRUST-MW.zip
```

2. ビルドスクリプトを変更する

コンテナ上でビルドスクリプトを実行すると正しくプラットフォームを認識できないために、異なるプラットフォーム向けのバイナリを作ってしまいます。imx_native_se050_t1oi2c_openssl_el2go 向けを強制的に True になるように変更する必要があります。以下が変更箇所です。i.MX6UL/ULL, i.MX8M Mini 向けの条件文が真になるように変更します。

```
[container /mnt]# cd simw-top
[container /mnt/simw-top]# vi scripts/create_cmake_projects.py
: (省略)
    # i.MX6UL/ULL and i.MX8M Mini EVK
    if True: #gc.imx_native_compilation() and gc.is_with_el2go(): ❶
        e = gc.generate_native("imx_native_se050_t1oi2c_openssl_el2go", {
            "PTMW_Applet": "SE05X_C",
            "PTMW_SE05X_Auth": "None",
: (省略)
```

❶ if 文を強制的に True にする。元の行はコメントアウトした

3. ビルドする

プロジェクトをつくるスクリプトを実行する。

```
[container /mnt]# cd simw-top
[container /mnt/simw-top]# python3 scripts/create_cmake_projects.py
### Native compilation on iMX Linux for SE05X using T=1 Over I2C for EdgeLock 2GO
#cmake -DPTMW_Applet=SE05X_C -DPTMW_SE05X_Auth=None (省略)...
-- The C compiler identification is GNU 10.3.1
-- The CXX compiler identification is GNU 10.3.1
: (省略)
-- Generating done
 Build files have been written to: /simw-top_build/(省略)...
```

simw-top_build に移動してビルドする。

```
[container /mnt]# cd simw-top_build/imx_native_se050_t1oi2c_openssl_el2go
[container /mnt/simw-top_build/imx_native_(省略)...]# cmake --build .
: (省略)
[  1%] Linking C shared library libmwlog.so
[  1%] Built target mwlog
[100%] Linking C executable ../../../bin/nxp_iot_agent_demo
[100%] Built target nxp_iot_agent_demo
```

4. ビルド結果の確認

以下の２つのディレクトリができる。

```
[container /mnt/simw-top_build]# ls
imx_native_se050_t1oi2c_openssl_el2go  simw-top-eclipse_jrcpv1
```

以下のデモアプリケーションがビルドされる。

```
[container /mnt/simw-top_build/imx_native_se050_t1oi2c_openssl_el2go]# ls bin
MQTTVersion                     se05x_ConcurrentEcc
accessManager                   se05x_ConcurrentSymm
apdu_player_demo                se05x_Delete_and_test_provision
claimcode_inject                se05x_GetInfo
ex_attest_ecc                   se05x_I2cMaster
ex_attest_mont                  se05x_I2cMasterWithAttestation
ex_ecc                          se05x_InjectCertificate
ex_ecdaa                        se05x_InvokeGarbageCollection
ex_ecdh                         se05x_MandatePlatformSCP
ex_hkdf                         se05x_Minimal
ex_hmac                         se05x_MultiThread
ex_md                           se05x_MultipleDigestCryptoObj
ex_policy                       se05x_PCR
ex_rsa                          se05x_ReadWithAttestation
ex_se05x_WiFiKDF_derive         se05x_TimeStamp
ex_se05x_WiFiKDF_inject         se05x_TransportLock
ex_symmetric                    se05x_TransportUnLock
generate_certificate            se05x_ex_export_se_to_host
generate_certificate_key        se05x_ex_import_host_to_se
jrcpv1_server                   seTool
nxp_iot_agent_demo              test_Crypto
remote_provisioning_client
```

5. デモアプリケーションで動作を確認する

se05x_GetInfo を実行する。SE050 にアクセスできると以下のようなログが出力される。

```
[container /mnt/simw-top_build/imx_native_(省略)/bin]# ./se05x_GetInfo ¥
 /dev/i2c-2:0x48
App   :INFO :PlugAndTrust_v03.03.00_20210528
App   :INFO :Running ./se05x_GetInfo
App   :INFO :Using PortName='/dev/i2c-2:0x48' (CLI)
: (省略)
App   :WARN :#####################################################
App   :INFO :Applet Major = 3
App   :INFO :Applet Minor = 1
App   :INFO :Applet patch = 1
App   :INFO :AppletConfig = 6FFF
App   :INFO :With    ECDAA
App   :INFO :With    ECDSA_ECDH_ECDHE
App   :INFO :With    EDDSA
App   :INFO :With    DH_MONT
App   :INFO :With    HMAC
App   :INFO :With    RSA_PLAIN
App   :INFO :With    RSA_CRT
App   :INFO :With    AES
```

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

11.3.4. アプリケーションの作成

plug and trust ミドルウェアにはでもアプリケーションが含まれます。そちらを参考にしてアプリケーションを作成することができます。

詳しくはミドルウェア内の doc ディレクトリ以下を参考にしてください。

SE05X Examples simw-top/doc/demos/index.html#se05x-examples

11.4. ソフトウェア実行環境の保護

この章ではソフトウェア実行環境を守るセキュリティ技術を適用する方法を説明します。 説明するセキュリティ技術は以下のとおりです。

- OP-TEE
  - CAAM を利用した OP-TEE
  - SE050 を利用した OP-TEE

11.4.1. OP-TEE

11.4.1.1. Arm TrustZone と TEE の活用

Linux を利用するシステムでは、自社開発したソフトウェアだけでなく複数の OSS が導入されるケースがあります。このような状況下では、自社開発したタスクだけでなく、同時に OSS のタスクが実行されることになり、システムのどこかに悪意のあるコードが含まれるのか、その可能性を排除することは困難です。仮にすべての OSS が信頼できるものであったとしても、不具合がないことを保証することはほぼ不可能であり、結局のところどのソフトウェアが脆弱性のきっかけになるかは分からないのです。

そういった背景から Arm は TrustZone 技術を導入しました。リソースアクセス制限によって敵対的なソフトウェアからソフトウェア実行環境を隔離することができます。TrustZone の導入によって、自社開発以外のソフトウェアが多数動作する状況においても、通常はセキュアワールドにその影響が及びません。TrustZone を利用したものとして、GlobalPlatform の TEE (Trusted Execution Environment) を実現するソフトウェアがいくつか存在します。TEE を活用すると secure world において信頼できるソフトウェアだけを実行させることが可能です。

では、どういったケースで TEE を採用するべきでしょうか。一概には言えませんが、情報資産とその資産を処理するライブラリなどをまとめて保護するケースで利用することが考えられます。具体的には、電子決済処理、証明書の処理、有料コンテンツの処理、個人情報の処理などで利用するケースが考えられます。

11.4.1.2. OP-TEE とは

商用、OSS の TEE などいくつかの TEE 実装が存在します。Armadillo Base OS では OP-TEE を採用します。OSS である OP-TEE は Arm コア向け TEE 実装の 1 つです。Arm TrustZone テクノロジー

によって情報資産とその処理をセキュアワールドに隔離して攻撃から保護します。OP-TEE は GlobalPlatform によって定義される TEE Client API や TEE Core API といった GlobalPlatformAPI に準拠したライブラリを提供しています。これらの API を利用することでユーザーはカスタムアプリケーションを開発することが可能です。imx-optee-xxx は NXP による i.MX ボートのサポート対応が含まれる OP-TEE の派生プロジェクトです。

OP-TEE の詳しい情報は公式のドキュメントを参照してください。

OP-TEE documentation	https://optee.readthedocs.io/en/latest/

11.4.2. OP-TEE の構成

OP-TEE を構成する主要なリポジトリは以下のとおりです。

- imx-optee-os
  - セキュアワールドで動作する TEE。optee-os の派生
- imx-optee-client
  - TEE を呼び出すためのノンセキュア、セキュアワールド向けの API ライブラリ。optee-client の派生
- imx-optee-test
  - OP-TEE の基本動作のテスト、パフォーマンス測定を行う。optee-test の派生。NXP 独自のテストが追加される
- optee_examples
  - サンプルアプリケーション

このうち、imx- というプレフィックスが付加されるリポジトリは、アップストリームのリポジトリに対して NXP による i.MX シリーズ向けの対応が入ったリポジトリになります。OP-TEE を利用するためには imx-optee-os をブートローダーに配置するだけでなく、Linux 上で動作するコンパニオン環境 (imx-optee-client) が必要です。また、TEE を利用するために CA (Client Application), TA (Trusted Application) を、imx-optee-os と imx-optee-client を用いてビルドします。必要に応じてテスト環境 (imx-optee-test) も追加してください。

詳しい OP-TEE のシステム構成については 「11.4.7.1. ソフトウェア全体像」 を参照してください。

OP-TEE git に関する詳しい情報は公式のドキュメントを参照してください。

OP-TEE gits	https://optee.readthedocs.io/en/latest/building/gits/

11.4.2.1. Armadillo Base OS への組み込み

前節で説明した OP-TEE の主要なリポジトリを実際に Armdillo BaseOS に適用する場合、Armadillo Base OS とどのように関係するのか全体像を説明します。

以下が Armadillo Base OS との関係を表した全体像。

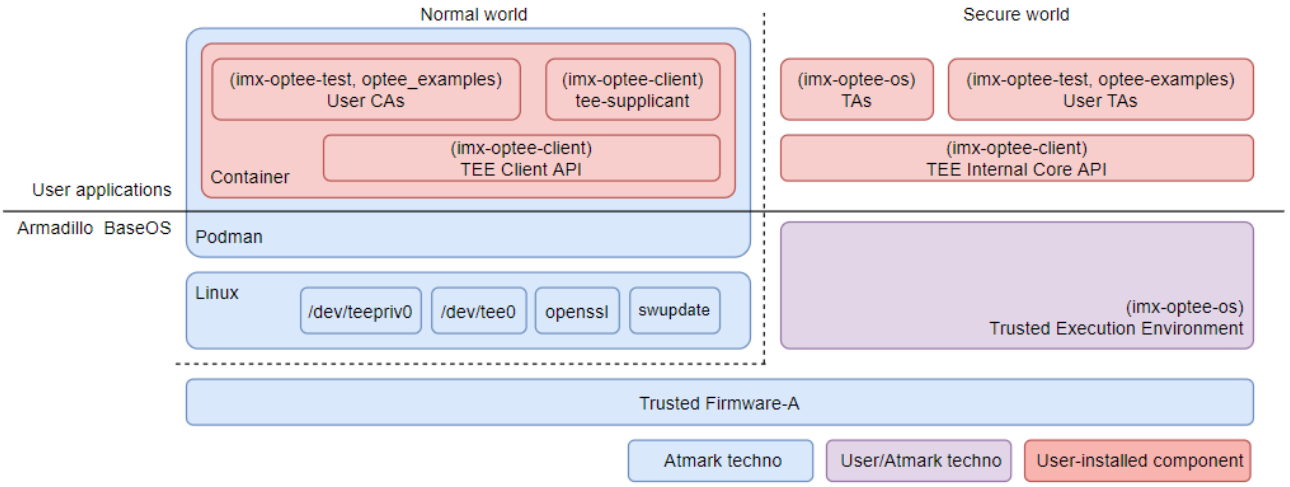


図 11.2 Armadillo Base OS と OP-TEE の担当範囲

- 青色の部分は Armadillo Base OS によってカバーされる範囲
- 赤色の部分は OP-TEE を組み込むために用意しなければならない部分
- 紫色の部分は デフォルトで Armadillo Base OS に組み込まれているが更新が必要な部分

imx-optee-os を組み込む作業は煩雑です。ユーザーの手間を省くために Armadillo Base OS には常に imx-optee-os バイナリがブートローダーに組み込むように実装されています。工場出荷状態イメージや製品アップデート向け Armadillo Base OS イメージにも含まれます。しかし、セキュリティ上の懸念から OP-TEE が利用できる状態で組み込まれていません。詳しくは 「11.4.3.1. 鍵の更新」 で説明します。

11.4.3. OP-TEE を利用する前に

OP-TEE を組み込むための準備を行います。

11.4.3.1. 鍵の更新

imx-optee-os リポジトリには TA を署名するためのデフォルトの秘密鍵が配置されています。その秘密鍵はあくまでテストや試行のためのものです。そのまま同じ秘密鍵を使い続けると、攻撃者がデフォルトの秘密鍵で署名した TA が動作する環境になってしまいます。各ユーザーが新しい鍵を用意する必要があります。

1. ソースコードを取得する「9.4. Armadillo のソフトウェアをビルドする」 の章を参考にしてビルド環境の構築と、ブートローダーのソースコードの取得を行ってください。

2. RSA 鍵ペアを生成します。

   次のコマンドを実行します。RSA/ECC を選択できます。

```
[PC ~]$ cd imx-boot/imx-optee-os/keys
[PC ~/imx-boot/imx-optee-os/keys]$ openssl genrsa -out rsa4096.pem 4096
[PC ~/imx-boot/imx-optee-os/keys]$ openssl rsa -in rsa4096.pem -pubout -out rsa4096_pub.pem
```

生成した秘密鍵 (上記の rsa4096.pem) は TA を更新していくために、今後も利用するものです。壊れにくい、セキュアなストレージにコピーしておくことをお勧めします。

鍵の更新は計画性を持って行ってください。たとえば開発時のみ利用する鍵、運用時に利用する鍵を使い分ける。また、鍵は定期的に更新が必要です。以下を参考にしてください。

BlueKrypt Cryptographic Key Length Recommendation
https://www.keylength.com/

鍵の更新に関する詳しい情報は公式のドキュメントを参照してください。

Offline Signing of TAs https://optee.readthedocs.io/en/latest/building/trusted_applications.html#offline-signing-of-tas

11.4.4. CAAM を活用した TEE を構築する

i.MX 8M Plus には CAAM (Cryptographic Acceleration and Assurance Module) と呼ばれる高機能暗号アクセラレータが搭載されています。CAAM は SoC 内部にあるためセキュアかつ高速に暗号処理を行うことが可能です。

11.4.4.1. ビルドの流れ

OP-TEE が含まれるブートローダーをビルドしていきます。Armadillo Base OS を利用した OP-TEE のビルドの流れは以下のとおりです。

1. ブートローダーをビルドする

2. imx-optee-client をビルドする

3. TA, CA をビルドする

4. ビルド結果を集める

本マニュアルで説明するビルド環境でのディレクトリ構成の概略は以下のとおりです。

optee_examples はユーザーアプリケーションを配置する場所です。本マニュアルでは Linaro が提供するアプリケーションのサンプルプログラムである optee_examples としました。本来は各ユーザーのアプリケーションを配置する場所になります。

```
├──── imx-boot
│     ├──── imx-atf
│     ├──── imx-mkimage
│     ├──── imx-optee-os
│     └──── uboot-imx
├──── imx-optee-client
├──── imx-optee-test
├──── optee_examples
└──── out
```

11.4.4.2. ビルド環境を構築する

「9.4. Armadillo のソフトウェアをビルドする」 の章を参考にしてビルド環境の構築からブートローダーのビルドまでを事前に行ってください。

imx-optee-test をビルドするために必要になります。

```
[PC ~]$ sudo apt install g++-aarch64-linux-gnu
```

11.4.4.3. ブートローダーを再ビルドする

新しい鍵を取り込むためにブートローダーを再ビルドする必要があります。

1. uboot-imx のバージョンを変更する

   この後の作業で swupdate を利用してアップデートを行いますが、ターゲットボード上の uboot-imx と同じバージョンではアップデートが実行されないので、imx-boot を更新するたびに uboot-imx のバージョンも更新する必要があります。

   uboot-imx ディレクトリに localversion ファイルを作ります。

   ```
   [PC ~]$ cd imx-boot/uboot-imx
   [PC ~/imx-boot/uboot-imx]$ echo "-localv1.0.0" > localversion
   ```

localversion の仕様はユーザーによって定義されます。

2. ブートローダーをビルドする

次のコマンドを実行します。デフォルトの設定ではデフォルトの鍵を利用してしまうので make 引数で鍵のパスを渡す必要があります。「11.4.3.1. 鍵の更新」 で生成した鍵のパスを変数 TA_SIGN_KEY で渡します。ここでは鍵名を rsa4096 としましたが、各ユーザーの環境に合わせた鍵名と読み替えてください。

```
[PC ~]$ make CFG_CC_OPT_LEVEL=2 ¥
        TA_SIGN_KEY="${PWD}/imx-boot/imx-optee-os/keys/rsa4096.pem" ¥
        -C imx-boot imx-boot_armadillo_x2
```

引数 TA_SIGN_KEY で秘密鍵をわたすことで、ビルド時に TA の署名に利用されます。また、実行時の署名確認のためには公開鍵を取り出して実行バイナリに取り込みます。

ビルド結果をコピーしておきます。

```
[PC ~]$ install -D -t ${PWD}/out/lib/optee_armtz ¥
        -m644 ${PWD}/imx-boot/imx-optee-os/out/export-ta_arm64/ta/*
```

11.4.4.4. imx-optee-client をビルドする

imx-optee-client は TA, CA が利用するライブラリ。アプリケーションをビルドする前にビルド必要がある。

imx-optee-client は imx-optee-os のビルド結果を参照しているので、事前にブートローダーをビルドする必要があります。

1. imx-optee-client をクローンする

imx-boot のディレクトリと並列に配置されるように imx-optee-client をクローンして、必要に応じて適切なブランチなどをチェックアウトしてください。

```
[PC ~]$ git clone https://source.codeaurora.org/external/imx/imx-optee-client.git ¥
        -b lf-5.10.y_2.0.0
```

2. imx-optee-client をビルドする

基本的な情報を参照するために TA_DEV_KIT_DIR を渡します。フォルトのターゲットでインストールも合わせて行うので DESTDIR を渡します。アプリケーションをビルドする際に API ライブラリとして利用されます。

```
[PC ~]$ make -C imx-optee-client ¥
        DESTDIR="${PWD}/out" ¥
```

```
        CROSS_COMPILE="aarch64-linux-gnu-" ¥
        TA_DEV_KIT_DIR="${PWD}/imx-boot/imx-optee-os/out/export-ta_arm64"
```

11.4.4.5. アプリケーションをビルドする

本来はユーザーが独自に作ったアプリケーションをビルドしますが、ここでは参考までにサンプルアプリケーション optee_examples をビルドします。アプリケーション開発の参考にしてください。

- アプリケーションをビルドするためには imx-optee-os, imx-optee-client のビルド結果を参照しているので、一度は imx-boot, imx-optee-client をビルドする必要があります
- optee_examples にはインストールする make ターゲットがないので、ビルド後に手動で集める作業が必要があります

1. optee_examples をクローンする

   imx-boot や imx-optee-client ディレクトリと並列に配置されるように optee_examples をクローンして、必要に応じて適切なブランチなどをチェックアウトしてください。

```
[PC ~]$ git clone https://github.com/linaro-swg/optee_examples.git ¥
        -b 3.13.0
```

2. optee_examples をビルドする

   基本的な情報を参照するために TA_DEV_KIT_DIR、TA を署名するために TA_SIGN_KEY、ライブラリ参照のために TEEC_EXPORT を渡します。インストールターゲットがないので後ほど手動でビルド結果を収集する必要があります。

```
[PC ~]$ make -C optee_examples ¥
        TA_CROSS_COMPILE="aarch64-linux-gnu-" ¥
        HOST_CROSS_COMPILE="aarch64-linux-gnu-" ¥
        TA_DEV_KIT_DIR="${PWD}/imx-boot/imx-optee-os/out/export-ta_arm64" ¥
        TEEC_EXPORT="${PWD}/out/usr" ¥
        TA_SIGN_KEY="$PWD/imx-boot/imx-optee-os/keys/rsa4096.pem"
```

   ビルド結果をコピーしておきます。

```
[PC ~]$ install -D -t ${PWD}/out/lib/optee_armtz -m644 ${PWD}/optee_examples/out/ta/*
[PC ~]$ install -D -t ${PWD}/out/usr/bin -m755 ${PWD}/optee_examples/out/ca/*
```

11.4.4.6. imx-optee-test をビルドする

imx-optee-test は必ずしも必要ではありません。利用機会は限られますが、OP-TEE の基本動作を確認するため、パフォーマンスを計測するために imx-optee-test を利用することができます。組み込むかどうかの判断はお任せします。

アプリケーションをビルドするためには imx-optee-os, imx-optee-client のビルド結果を参照しているので、一度は imx-boot, imx-optee-client をビルドする必要があります。

1. imx-optee-test をクローンする

   imx-boot や imx-optee-client ディレクトリと並列に配置されるように imx-optee-client をクローンして、必要に応じて適切なブランチなどをチェックアウトしてください。

```
[PC ~]$ git clone https://source.codeaurora.org/external/imx/imx-optee-test.git ¥
        -b lf-5.10.y_2.0.0
```

2. imx-optee-test をビルドする

   基本的な情報を参照するために TA_DEV_KIT_DIR、TA を署名するために TA_SIGN_KEY、ライブラリ参照のために TEEC_EXPORT を渡します。ビルドとインストールは別のターゲットなのでそれぞれ make を実行する必要があります。

```
[PC ~]$ CFLAGS=-O2 make -R -C imx-optee-test ¥
        CROSS_COMPILE="aarch64-linux-gnu-" ¥
        TA_DEV_KIT_DIR="${PWD}/imx-boot/imx-optee-os/out/export-ta_arm64" ¥
        OPTEE_CLIENT_EXPORT="${PWD}/out/usr" ¥
        TA_SIGN_KEY="${PWD}/imx-boot/imx-optee-os/keys/rsa4096.pem"
```

   インストールします。

```
[PC ~]$ make -C imx-optee-test install ¥
        DESTDIR="${PWD}/out" ¥
        TA_DEV_KIT_DIR="${PWD}/imx-boot/imx-optee-os/out/export-ta_arm64" ¥
        OPTEE_CLIENT_EXPORT="${PWD}/out/usr"
```

本リリース時点では xtest 1014 にてエラーになる問題があります。そのため環境変数で CFLAGS=-O2 を渡しています。make 引数で渡すとヘッダファイルの include 設定がなくなります。ブートローダーのビルド時に O2 としているのも同様の理由です。

NXP LS Platforms: pkcs_1014 test is failing #4909 https://github.com/OP-TEE/optee_os/issues/4909

11.4.4.7. ビルド結果の確認とまとめる

imx-optee-os, imx-optee-client の最小構成では以下のとおりです。

```
out/
|-- lib
|   `-- optee_armtz ❶
|       |-- 023f8f1a-292a-432b-8fc4-de8471358067.ta
|       |-- f04a0fe7-1f5d-4b9b-abf7-619b85b4ce8c.ta
|       `-- fd02c9da-306c-48c7-a49c-bbd827ae86ee.ta
`-- usr
    |-- include
    |   |-- ck_debug.h
    |   |-- optee_client_config.mk
    |   |-- pkcs11.h
    |   |-- pkcs11_ta.h
    |   |-- tee_bench.h
    |   |-- tee_client_api.h
    |   |-- tee_client_api_extensions.h
    |   |-- tee_plugin_method.h
    |   `-- teec_trace.h
    |-- lib ❷
    |   |-- libckteec.a
    |   |-- libckteec.so -> libckteec.so.0
    |   |-- libckteec.so.0 -> libckteec.so.0.1
    |   |-- libckteec.so.0.1 -> libckteec.so.0.1.0
    |   |-- libckteec.so.0.1.0
    |   |-- libteec.a
    |   |-- libteec.so -> libteec.so.1
    |   |-- libteec.so.1 -> libteec.so.1.0.0
    |   |-- libteec.so.1.0 -> libteec.so.1.0.0
    |   `-- libteec.so.1.0.0
    `-- sbin ❸
        `-- tee-supplicant
```

❶　Dynamic TA。Linux のファイルシステムに保存される

❷　TEE client API, TEE, internal core API

❸　Linux 上で動作する OP-TEE の補助的な機能をもつ

imx-optee-test, optee_examples を含めたビルド結果は以下のとおりです。

```
out
|-- bin
|   `-- xtest
|-- lib
|   `-- optee_armtz
|       |-- 023f8f1a-292a-432b-8fc4-de8471358067.ta
|       |-- 2a287631-de1b-4fdd-a55c-b9312e40769a.ta
|       |-- 380231ac-fb99-47ad-a689-9e017eb6e78a.ta
|       |-- 484d4143-2d53-4841-3120-4a6f636b6542.ta
|       |-- 528938ce-fc59-11e8-8eb2-f2801f1b9fd1.ta
|       |-- 5b9e0e40-2636-11e1-ad9e-0002a5d5c51b.ta
|       |-- 5ce0c432-0ab0-40e5-a056-782ca0e6aba2.ta
|       |-- 5dbac793-f574-4871-8ad3-04331ec17f24.ta
|       |-- 614789f2-39c0-4ebf-b235-92b32ac107ed.ta
|       |-- 690d2100-dbe5-11e6-bf26-cec0c932ce01.ta
|       |-- 731e279e-aafb-4575-a771-38caa6f0cca6.ta
|       |-- 873bcd08-c2c3-11e6-a937-d0bf9c45c61c.ta
```

```
|       |-- 8aaaf200-2450-11e4-abe2-0002a5d5c51b.ta
|       |-- a4c04d50-f180-11e8-8eb2-f2801f1b9fd1.ta
|       |-- a734eed9-d6a1-4244-aa50-7c99719e7b7b.ta
|       |-- b3091a65-9751-4784-abf7-0298a7cc35ba.ta
|       |-- b689f2a7-8adf-477a-9f99-32e90c0ad0a2.ta
|       |-- b6c53aba-9669-4668-a7f2-205629d00f86.ta
|       |-- c3f6e2c0-3548-11e1-b86c-0800200c9a66.ta
|       |-- cb3e5ba0-adf1-11e0-998b-0002a5d5c51b.ta
|       |-- d17f73a0-36ef-11e1-984a-0002a5d5c51b.ta
|       |-- e13010e0-2ae1-11e5-896a-0002a5d5c51b.ta
|       |-- e626662e-c0e2-485c-b8c8-09fbce6edf3d.ta
|       |-- e6a33ed4-562b-463a-bb7e-ff5e15a493c8.ta
|       |-- f04a0fe7-1f5d-4b9b-abf7-619b85b4ce8c.ta
|       |-- f157cda0-550c-11e5-a6fa-0002a5d5c51b.ta
|       |-- f4e750bb-1437-4fbf-8785-8d3580c34994.ta
|       |-- fd02c9da-306c-48c7-a49c-bbd827ae86ee.ta
|       `-- ffd2bded-ab7d-4988-95ee-e4962fff7154.ta
`-- usr
    |-- bin
    |   |-- optee_example_acipher
    |   |-- optee_example_aes
    |   |-- optee_example_hello_world
    |   |-- optee_example_hotp
    |   |-- optee_example_plugins
    |   |-- optee_example_random
    |   `-- optee_example_secure_storage
    |-- include
    |   |-- ck_debug.h
    |   |-- optee_client_config.mk
    |   |-- pkcs11.h
    |   |-- pkcs11_ta.h
    |   |-- tee_bench.h
    |   |-- tee_client_api.h
    |   |-- tee_client_api_extensions.h
    |   |-- tee_plugin_method.h
    |   `-- teec_trace.h
    |-- lib
    |   |-- libckteec.a
    |   |-- libckteec.so -> libckteec.so.0
    |   |-- libckteec.so.0 -> libckteec.so.0.1
    |   |-- libckteec.so.0.1 -> libckteec.so.0.1.0
    |   |-- libckteec.so.0.1.0
    |   |-- libteec.a
    |   |-- libteec.so -> libteec.so.1
    |   |-- libteec.so.1 -> libteec.so.1.0.0
    |   |-- libteec.so.1.0 -> libteec.so.1.0.0
    |   |-- libteec.so.1.0.0
    |   `-- tee-supplicant
    |       `-- plugins
    |           `-- f07bfc66-958c-4a15-99c0-260e4e7375dd.plugin
    `-- sbin
        `-- tee-supplicant
```

ターゲット上のコンテナに展開するために、tarball で固めます。

```
[PC ~]$ tar -caf optee.tar.gz -C out .
```

「9.1. アプリケーションをコンテナで実行する」 を参考にしてコンテナ作成の時に組み込むことをお勧めします

11.4.4.8. OP-TEE を組み込む

ターゲットデバイスで debian コンテナを起動して、その上で OP-TEE を動作させます。

1. swu でブートローダーをアップデートする「9.7. Armadillo のソフトウェアをアップデートする」を参考にアップデートしてください。
2. ビルド結果が置かれているパスでコンテナを立ち上げます

   ここでは debian を利用しています。

```
[armadillo ~]# podman run -it --name=dev_optee --device=/dev/tee0 ¥
        --device=/dev/teepriv0 -v "$(pwd)":/mnt docker.io/debian /bin/bash
```

3. ビルド結果を展開する

   tarball を展開します。

```
[container ~]# tar -xaf /mnt/optee.tar.gz -C /
```

4. tee-supplicant を起動する

```
[container ~]# tee-supplicant -d
```

5. xtest で動作を確認する

   xtest で OP-TEE の基本動作を確認します。以下のログは全テストをパスしたログです。

```
[container ~]# xtest
Run test suite with level=0

TEE test application started over default TEE instance
##########################################################
#
# regression+pkcs11+regression_nxp
#
##########################################################

* regression_1001 Core self tests
  regression_1001 OK
: (省略)
+-----------------------------------------------------
33939 subtests of which 0 failed
114 test cases of which 0 failed
```

```
0 test cases were skipped
TEE test application done!
```

xtest の全テストをパスできない場合は環境構築から見直していただくことをお勧めします。問題が解決できないようであればサポートにご連絡ください。

1. アプリケーションを起動する

   ビルド結果を展開したことで CA も TA も配置されました。目的の CA を起動してください。ここでは optee_examples の optee_example_hello_world を実行します。

```
[container ~]# optee_example_hello_world
D/TA:  TA_CreateEntryPoint:39 has been called
D/TA:  TA_OpenSessionEntryPoint:68 has been called
I/TA: Hello World!
D/TA:  inc_value:105 has been called
I/TA: Got value: 42 from NW
I/TA: Increase value to: 43
I/TA: Goodbye!
Invoking TA to increment 42
TA incremented value to 43
D/TA:  TA_DestroyEntryPoint:50 has been called
```

tee-supplicant は OP-TEE の linux 環境のコンパニオンプロセスです。OP-TEE を利用するためにはなくてはならないものです。自動起動することをお勧めします。詳しくは、「9.1. アプリケーションをコンテナで実行する」 を参考にしてください。

11.4.5. パフォーマンスを測定する

xtest を利用することで AES, SHA アルゴリズムの OP-TEE OS のパフォーマンスを測定することができます。

AES のパフォーマンスを計測するために次のコマンドを実行します。この結果は例になります。

```
[container ~]# xtest --aes-perf
min=113.753us max=191.881us mean=116.426us stddev=4.10202us (cv 3.5233%) (8.38786MiB/s)
```

SHA のパフォーマンスを計測するためのコマンドを実行します。この結果も例になります。

```
[container ~]# xtest --sha-perf
min=50.876us max=123.003us mean=52.8036us stddev=2.4365us (cv 4.61427%) (18.494
```

11.4.6. SE050 を活用した TEE を構築する

NXP Semiconductors の EdgeLock SE050 は IoT アプリケーション向けのセキュアエレメントです。様々なアルゴリズムに対応した暗号エンジン、セキュアストレージを搭載します。GlobalPlatform によって標準化されている Secure Channel Protocol 03 に準拠し、バスレベル暗号化 (AES)、ホストとカードの相互認証 (CMAC ベース) を行います。

OP-TEE で SE050 を用いるユースケースとしては、IoT アプリケーション向けに特化された豊富な機能を活用した上で、ホスト側の処理を守りたい場合に利用することが考えられます。OP-TEE を組み合わせることで SE050 へアクセスする部分、保存された情報資産を取り出して実際に処理する部分を守ることができます。

ユーザーが SE050 にアクセスする場合は、OP-TEE の TEE Client API や TEE Core API といった GlobalPlatformAPI を呼び出すことになります。デバイスの変更などの状況で比較的容易に移植が可能になります。

SE050 の詳細については以下の NXP Semiconductors のページから検索して、ご確認ください。

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

11.4.6.1. OP-TEE 向け plug-and-trust ライブラリ

NXP Semiconductors が開発するライブラリ plug-and-trust を利用して SE050 にアクセスします。OP-TEE への移植は Foundries.io によって行われ、Github にて公開されています。現状、SE050 の全ての機能を使えるわけではありません。主に暗号強度が弱い鍵長が無効化されています。

現状で対応する処理:

- RSA 2048, 4096 encrypt/decrypt/sign/verify
- ECC sign/verify
- AES CTR
- RNG
- SCP03 (i2c communications between the processor and the device are encrypted)
- DieID generation
- cryptoki integration

OP-TEE 向け plug-and-trust の詳細の情報は以下を参照してください。

OP-TEE Enabled Plug and Trust Library https://github.com/foundriesio/plug-and-trust

11.4.6.2. ビルドの流れ

基本的には CAAM の場合と同様の流れになる。

1. ブートローダーをビルドする
2. imx-optee-client をビルドする
3. TA, CA をビルドする
4. ビルド結果を集める

ディレクトリ構成の概略は以下のとおりです。

```
├──── imx-boot
│     ├──── imx-atf
│     ├──── imx-mkimage
│     ├──── imx-optee-os
│     └──── uboot-imx
├──── imx-optee-client
├──── imx-optee-test
├──── optee_examples
├──── plug-and-trust
└──── out
```

11.4.6.3. ビルド環境を構築する

OP-TEE 向け plug-and-trust をビルドするために必要なパッケージをインストールします。

```
[PC ~]$ sudo apt install cmake
```

11.4.6.4. OP-TEE 向け plug-and-trust をビルドする

1. OP-TEE 向け plug-and-trust をクローンする

   imx-boot や imx-optee-client ディレクトリと並列に配置されるように OP-TEE 向け plug-and-trust をクローンして、必要に応じて適切なブランチなどをチェックアウトしてください。

   ```
   [PC ~]$ git clone https://github.com/foundriesio/plug-and-trust.git -b optee_lib
   ```

2. OP-TEE 向け plug-and-trust をビルドする

   ```
   [PC ~]$ mkdir -p plug-and-trust/optee_lib/build
   [PC ~]$ cd plug-and-trust/optee_lib/build
   [PC ~/plug-and-trust/optee_lib/build]$ cmake ¥
   ```

```
        -DCMAKE_C_FLAGS="-mstrict-align -mgeneral-regs-only" ¥
        -DCMAKE_C_COMPILER=aarch64-linux-gnu-gcc ¥
        -DOPTEE_TREE="${PWD}/../../../imx-boot/imx-optee-os" ..

-- The C compiler identification is GNU 10.2.1
-- The CXX compiler identification is GNU 10.2.1
: (省略)
-- Generating done
-- Build files have been written to: /path/plug-and-trust/optee_lib/build
```

```
[PC ~/plug-and-trust/optee_lib/build]$ make
make
Consolidate compiler generated dependencies of target se050
[  4%] Building C object CMakeFiles/se050.dir/path/plug-and-trust/hostlib/hostLib/ ⏎
libCommon/infra/global_platf.c.o
[  8%] Building C object CMakeFiles/se050.dir/path/plug-and-trust/hostlib/hostLib/ ⏎
libCommon/infra/sm_apdu.c.o
: (省略)
[100%] Linking C static library libse050.a
[100%] Built target se050
```

11.4.6.5. imx-optee-os のコンフィグの修正

SE050 を crypto driver として利用するためにコンフィグを修正する。

SE050 向けにビルドするために imx-boot/Makefile を追加する。

```
$(OPTEE)/out/tee.bin: $(OPTEE)/.git FORCE
        $(MAKE) -C $(OPTEE) O=out ARCH=arm PLATFORM=imx CFG_WERROR=y ¥
        PLATFORM_FLAVOR=mx8mpevk ¥
        CFG_NXP_SE05X=y ¥ ❶
        CFG_IMX_I2C=y ¥ ❷
        CFG_CORE_SE05X_I2C_BUS=2 ¥
        CFG_CORE_SE05X_BAUDRATE=400000 ¥
        CFG_CORE_SE05X_OEFID=0xA200 ¥
        CFG_IMX_CAAM=n ¥ ❸
        CFG_NXP_CAAM=n ¥
        CFG_CRYPTO_WITH_CE=y ¥ ❹
        CFG_STACK_THREAD_EXTRA=8192 ¥ ❺
        CFG_STACK_TMP_EXTRA=8192 ¥
        CFG_NUM_THREADS=1 ¥ ❻
        CFG_WITH_SOFTWARE_PRNG=n ¥ ❼
        CFG_NXP_SE05X_PLUG_AND_TRUST_LIB=~/plug-and-trust/optee_lib/build/libse050.a ¥ ❽
        CFG_NXP_SE05X_PLUG_AND_TRUST=~/plug-and-trust/
```

コンフィグの修正に関する詳細

❶　SE050 を利用するために有効にする

❷　imx-i2c ドライバ を有効にする

❸　CAAM は無効化する

❹ AES や SHA は高速な Arm CE を利用する

❺ スタックを通常よりも多く消費するためにスタックを増量する

❻ スレッドによる複数のコンテキストに対応していないためスレッドを 1 つとする

❼ ハードウェア乱数発生器を利用するために無効にする

❽ SE050 のドライバの実装は OP-TEE 向け plug-and-trust ライブラリ内に存在する

- SE050 の host 接続用 I2C は最大 3.2 MHz (high speed)ですが、i.MX 8M Plus の i2c の最大周波数は 400kHz のため、遅い通信速度で実装されています
- CAAM と SE050 の共存は、SE050 を有効にすることによって CAAM の個別のドライバの依存関係が不正になるため、実行時にエラーになる問題があります

11.4.6.6. uboot-imx の修正

Armadillo は消費電力の削減のため SE050 を Deep Power-down モードに設定してパワーゲーティングしている。Deep Power-down モードを解除して SE050 を利用するためには、i.MX 8M Plus に接続されている SE050 の ENA ピンをアサートする必要があります。ENA ピンをアサートすると SE050 は一定時間の後に起動するので SE050 を利用すためには若干の待ち時間が必要となります。OP-TEE OS は起動時にドライバの初期化等を行う実装になっている。そのため、OP-TEE OS が起動する前に生存している SPL (Secondary Program Loader) で Deep Power-down を解除することで待ち時間を稼いでいる。

以下はシステムの起動と SE050 の関係を示したシーケンス図。

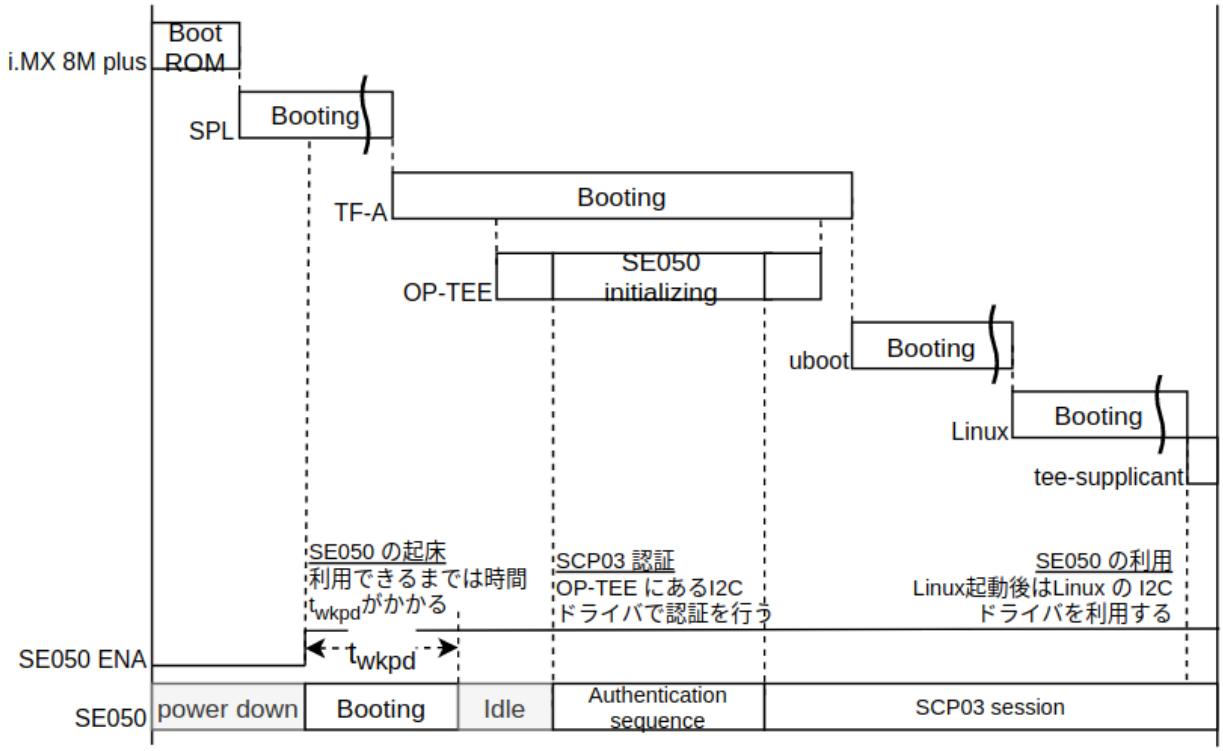


図 11.3 SE050 向け OP-TEE の起動シーケンス図

ENA をアサートするために以下のように変更する

```
diff --git a/board/atmark-techno/armadillo_x2/spl.c b/board/atmark-techno/armadillo_x2/spl.c
index a26bc85633..88f5b8560c 100644
--- a/board/atmark-techno/armadillo_x2/spl.c
+++ b/board/atmark-techno/armadillo_x2/spl.c
@@ -111,6 +111,11 @@ static struct fsl_esdhc_cfg usdhc_cfg[2] = {
        {USDHC3_BASE_ADDR, 0, 8},
 };

+#define SE_RST_N IMX_GPIO_NR(1, 12)
+static iomux_v3_cfg_t const se_rst_n_pads[] = {
+       MX8MP_PAD_GPIO1_IO12__GPIO1_IO12 | MUX_PAD_CTRL(NO_PAD_CTRL),
+};
+
 int board_mmc_init(bd_t *bis)
 {
        int i, ret;
@@ -228,6 +233,12 @@ void spl_board_init(void)
        clock_enable(CCGR_GIC, 1);
 #endif

+       imx_iomux_v3_setup_multiple_pads(se_rst_n_pads,
+                                        ARRAY_SIZE(se_rst_n_pads));
+
+       gpio_request(SE_RST_N, "se_rst_n");
+       gpio_direction_output(SE_RST_N, 1);
+
```

```
        puts("Normal Boot¥n");
 }
```

11.4.6.7. imx-optee-os の imx-i2c ドライバの修正

imx-optee-os の imx-i2c ドライバには i.MX 8M Plus の対応が入っていないため、レジスタ等の定義を追加する必要がある。imx-optee-os には lf-5.10.y_2.0.0 から SE050 ドライバが取り込まれている。

以下のように修正する。

```
diff --git a/core/arch/arm/plat-imx/conf.mk b/core/arch/arm/plat-imx/conf.mk
index b4fbfed5..3f6388f3 100644
--- a/core/arch/arm/plat-imx/conf.mk
+++ b/core/arch/arm/plat-imx/conf.mk
@@ -555,7 +555,7 @@ endif

 else

-$(call force,CFG_CRYPTO_DRIVER,n)
-$(call force,CFG_WITH_SOFTWARE_PRNG,y)
+#$(call force,CFG_CRYPTO_DRIVER,n) ❶
+#$(call force,CFG_WITH_SOFTWARE_PRNG,y) ❷

 ifneq (,$(filter y, $(CFG_MX6) $(CFG_MX7) $(CFG_MX7ULP)))
diff --git a/core/arch/arm/plat-imx/registers/imx8m.h b/core/arch/arm/plat-imx/registers/imx8m.h
index 9b6a50ee..59fcea88 100644
--- a/core/arch/arm/plat-imx/registers/imx8m.h
+++ b/core/arch/arm/plat-imx/registers/imx8m.h
@@ -42,6 +42,17 @@
 #define IOMUXC_I2C1_SDA_CFG_OFF        0x480
 #define IOMUXC_I2C1_SCL_MUX_OFF        0x214
 #define IOMUXC_I2C1_SDA_MUX_OFF        0x218
+#elif defined(CFG_MX8MP)
+#define I2C1_BASE              0x30a20000
+#define I2C2_BASE              0x30a30000
+#define I2C3_BASE              0x30a40000
+
+#define IOMUXC_I2C1_SCL_CFG_OFF        0x460
+#define IOMUXC_I2C1_SDA_CFG_OFF        0x464
+#define IOMUXC_I2C1_SCL_MUX_OFF        0x200
+#define IOMUXC_I2C1_SDA_MUX_OFF        0x204
+#define IOMUXC_I2C1_SCL_INP_OFF        0x5A4
+#define IOMUXC_I2C1_SDA_INP_OFF        0x5A8
 #endif

 #endif /* __IMX8M_H__ */
diff --git a/core/drivers/imx_i2c.c b/core/drivers/imx_i2c.c
index a318c32c..a9dab31c 100644
--- a/core/drivers/imx_i2c.c
+++ b/core/drivers/imx_i2c.c
@@ -34,6 +34,16 @@
 /* Clock */
 #define I2C_CLK_CGRBM(__x)     0 /* Not implemented */
 #define I2C_CLK_CGR(__x)       CCM_CCRG_I2C##__x
+#elif defined(CFG_MX8MP)
+/* IOMUX */
```

```
+#define I2C_INP_SCL(__x)       (IOMUXC_I2C1_SCL_INP_OFF + ((__x) - 1) * 0x8)
+#define I2C_INP_SDA(__x)       (IOMUXC_I2C1_SDA_INP_OFF + ((__x) - 1) * 0x8)
+#define I2C_INP_VAL(__x)       (((__x) == 1 || (__x) == 2) ? 0x2 : 0x4)
+#define I2C_MUX_VAL(__x)       0x010
+#define I2C_CFG_VAL(__x)       0x1c6
+/* Clock */
+#define I2C_CLK_CGRBM(__x)     0 /* Not implemented */
+#define I2C_CLK_CGR(__x)       CCM_CCRG_I2C##__x
 #elif defined(CFG_MX6ULL)
 /* IOMUX */
 #define I2C_INP_SCL(__x)       (IOMUXC_I2C1_SCL_INP_OFF + ((__x) - 1) * 0x8)
@@ -182,7 +192,7 @@ static void i2c_set_bus_speed(uint8_t bid, int bps)
        vaddr_t addr = i2c_clk.base.va;
        uint32_t val = 0;

-#if defined(CFG_MX8MM)
+#if defined(CFG_MX8MM) || defined(CFG_MX8MP)
        addr += CCM_CCGRx_SET(i2c_clk.i2c[bid]);
        val = CCM_CCGRx_ALWAYS_ON(0);
 #elif defined(CFG_MX6ULL)
```

以下は imx プラットフォームの makefile の問題です。回避するためにコメントアウトします。

❶ imx プラットフォームで CAAM 以外の crypto driver を利用することを想定していない

❷ CAAM の HWRNG を利用しないということは PRNG を使うことしか想定しない

11.4.6.8. ビルドとターゲットボードへの組み込み

修正した後は、ビルドからターゲットボードへの組み込みまで CAAM 向けの OP-TEE と同様の手順で作業することが可能です。 「11.4.4.3. ブートローダーを再ビルドする」 の作業から開始して組み込みしてください。

11.4.6.9. xtest の制限

「11.4.6.1. OP-TEE 向け plug-and-trust ライブラリ」 で説明したように一部のアルゴリズムに制限があります。そのため xtest の全てのテスト項目をパスするわけではありません。以下に失敗するテスト項目を列挙する。

- 1009 TEE Wait cancel
  - キャンセル処理をするために OP-TEE はマルチスレッドが有効な構成でなくてはならない。SE050 へのアクセスをシリアライズする利用するために imx-optee-os の make 時にスレッドを 1 つにしているため
- regression_4007_rsa.1 Generate RSA-256 key
  - RSA 256 は対応していない鍵長
- regression_4006 Test TEE Internal API Asymmetric Cipher operations
  - RSA 1024 は対応していない鍵長
- regression_4009 Test TEE Internal API Derive key ECDH
  - E/TC:? 0 shared_secret:333 private key must be stored in SE050 flash

    - provisioning が必要
- regression_4011 Test TEE Internal API Bleichenbacher attack
    - RSA 512 は対応していない鍵長
- regression_6018 Large object
    - 原因が不明。ヒープが不足している可能性がある

以下は特に問題はないが時間がかかるためにフリーズしているかのように見える。

- 1006 Secure time source
    - 大変時間がかかるため固まったかのように見えます
- regression_nxp_0001, regression_nxp_0003
    - 大変時間がかかるため固まったかのように見えます

xtest を行う際にはデフォルトでテストに失敗しても先に進む設定となっています。ただ、時間のかかるテストは無効にすることも可能です。

```
[container ~]# xtest -x 1006 -x regression_nxp_0003
```

11.4.7. imx-optee-os 技術情報

11.4.7.1. ソフトウェア全体像

OP-TEE のアーキテクチャの概要を説明する。ここでは i.MX 8M Plus に搭載される Cortex-A53 コアのアーキテクチャである aarch64 を前提に話を進める。

以下にのシステム図を示す。

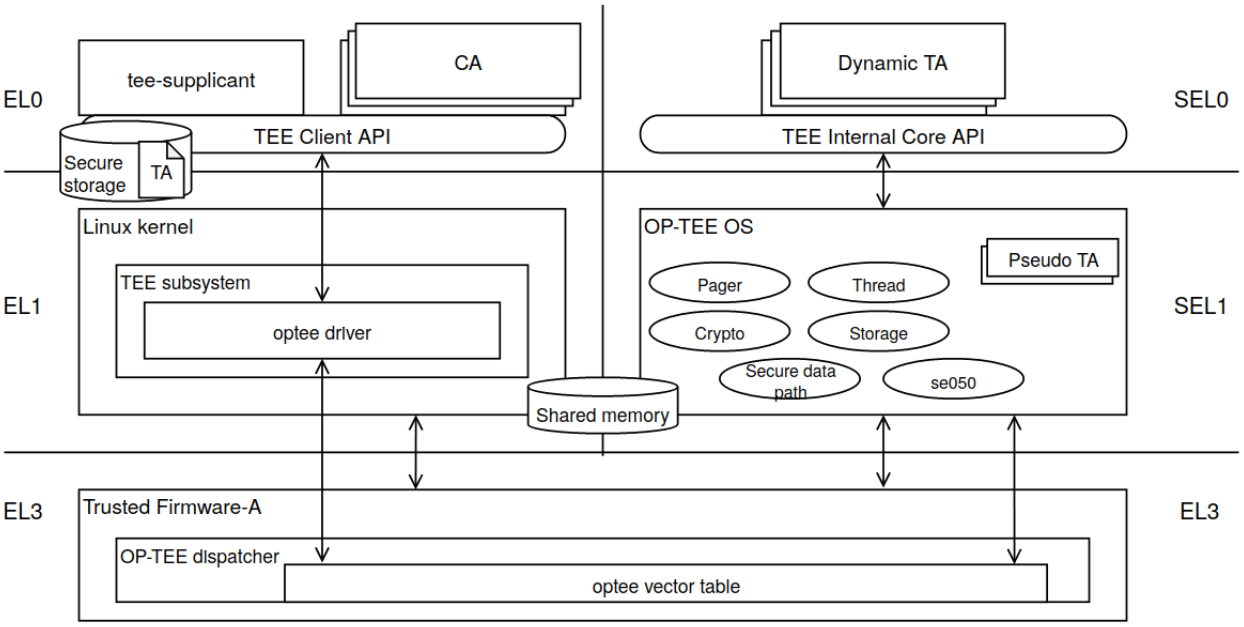


図 11.4 OPTEE のシステム図

以下のコンポーネントによってシステムが構成される。概要と主な責務について説明する。

- OP-TEE
  - GlobalPlatform の TEE 実装。secure EL1 に配置される
- Trusted Firmware-A
  - Linaro によって開発される Secure monitor の実装
  - secure state と non-secure state の遷移管理、PSCI に準拠した電源管理などを担当する
  - optee dispatcher と呼ばれる OP-TEE の呼び出しモジュールを内部に持つ
- Client Application (CA)
  - TA を呼び出すアプリケーション
- Trusted Application (TA)
  - TEE 上で CA からの呼び出しを処理
  - Dynamic TA と Pesudo TA がある
    - Pesudo は TA ではありません。通常は Dynamic TA を利用してください。Pesudo TA は OP-TEE OS に直接リンクされるため TEE Internal Core API は呼べません
  - Dynamic TA は Linux のファイルシステム上に配置される。tee-supplicant によって OP-TEE OS に引き渡される
- tee-supplicant

- Linux user 空間で動作する OP-TEE を補うプロセス。目的は Linux のリソースを OP-TEE OS が利用するため

11.4.7.2. フロー

TEE を呼び出すフローについて説明する。

CA が OP-TEE 上の TA とのセッションを確立する流れ

1. Linux 上の CA が、セッションを開くために uuid を指定して TEE Client API を呼び出す
   - システムコールで tee driver が呼ばれる
2. tee driver は セキュアモニタコールで Trusted Firmware-A (ATF) 上の OP-TEE dispacher を呼び出す
3. OP-TEE dispatcher は optee vector table に登録されている OP-TEE のハンドラを呼び出す
   - この段階ではまだ EL3 の状態
4. OP-TEE OS は自ら SEL1 に落ちて、内部処理をしてから、ここまでの逆順で tee-supplicant を呼び出す
5. tee-supplicant は uuid を基に TA をロードして共有メモリに配置して OP-TEE OS を呼び出す
6. OP-TEE は TA をロードする
7. セッションができる

CA が TEE Client API を通して TA 上である処理を実行する流れ

1. Linux 上の CA が TEE Client API を呼び出す
   - システムコールで tee driver が呼ばれる
2. tee driver は セキュアモニタコールで Trusted Firmware-A (ATF) 上の OP-TEE dispacher を呼び出す
3. OP-TEE dispatcher は optee vector table に登録されている OP-TEE のハンドラを呼び出す
4. ハンドラ (OP-TEE OS) は自ら SEL1 に落ちる。内部処理をしてから、SEL0 に落ちて TA を呼び出す
5. TA は API の引数を基にある処理を実行する
6. ここまでの逆順で CA まで戻る

より詳しい内容については公式ドキュメントをご覧ください。

Normal World invokes OP-TEE OS using SMC https://optee.readthedocs.io/en/latest/architecture/core.html#normal-world-invokes-op-tee-os-using-smc

11.4.7.3. メモリマップ

セキュリティ関連の領域を含めた i.MX 8M Plus の物理メモリマップを次に示します。

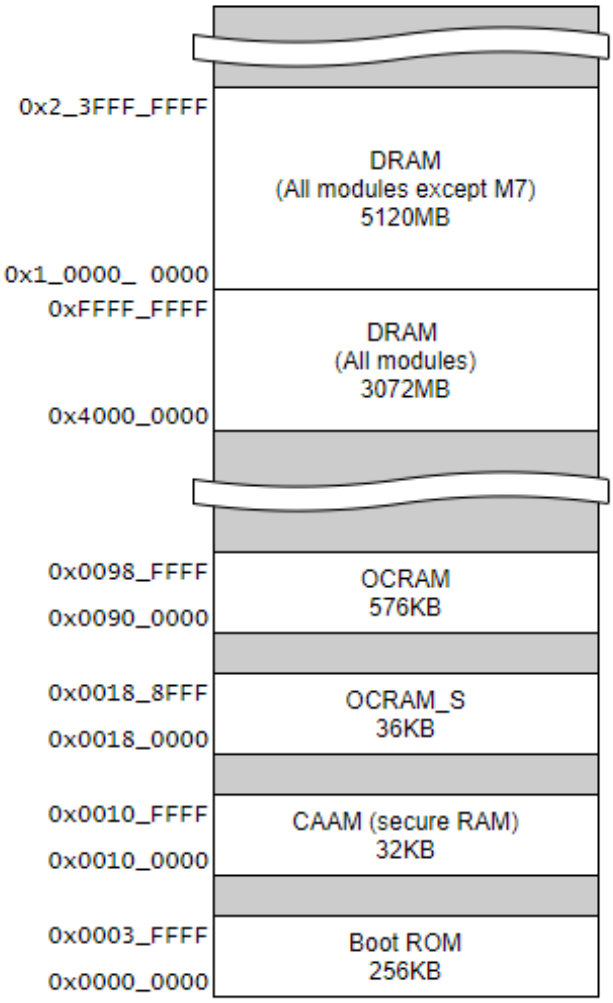


図 11.5 i.MX 8M Plus の物理メモリマップ

imx-optee-os のメモリマップを次に示します。デバッグ等にお役立てください。

表 11.2 OP-TEE メモリマップ

type	virtual address	physical address	size	description
TEE_RAM_RX/RW	0x5600_0000.. 0x561f_ffff	0x5600_0000.. 0x561f_ffff	0x0020_0000 (smallpg)	OP-TEE text + data セクション
IO_SEC	0x5620_0000.. 0x5620_ffff	0x32f8_0000.. 0x32f8_ffff	0x0001_0000 (smallpg)	TZASC
SHM_VASPACE	0x5640_0000.. 0x583f_ffff	0x0000_0000.. 0x01ff_ffff	0x0200_0000 (pgdir)	OP-TEE dynamic shared memory area, va の確保のみ
RES_VASPACE	0x5840_0000.. 0x58df_ffff	0x0000_0000.. 0x009f_ffff	0x00a0_0000 (pgdir)	OP-TEE 予約領域 (late mapping), va の確保のみ
IO_SEC	0x58e0_0000.. 0x591f_ffff	0x3020_0000.. 0x305f_ffff	0x0040_0000 (pgdir)	AIPS1(GPIO1, GPT, IOMUXC など)

type	virtual address	physical address	size	description
IO_NSEC	0x5920_0000.. 0x595f_ffff	0x3080_0000.. 0x30bf_ffff	0x0040_0000 (pgdir)	AIPS3(UART2, CAAM, I2C など)
IO_SEC	0x5960_0000.. 0x597f_ffff	0x3880_0000.. 0x389f_ffff	0x0020_0000 (pgdir)	GICv3
TA_RAM	0x5980_0000.. 0x5b1f_ffff	0x5620_0000.. 0x57bf_ffff	0x01a0_0000 (pgdir)	TA ロード、実行領域
NSEC_SHM	0x5b20_0000.. 0x5b5f_ffff	0x57c0_0000.. 0x57ff_ffff	0x0040_0000 (pgdir)	OP-TEE contignuous shared memory area

11.5. セキュアブート

この章では起動ソフトウェアを認証するセキュリティ技術を適用する方法を説明します。

11.5.1. セキュアブートとチェーンオブトラスト

組み込みデバイスへの攻撃は様々な方向から行われます。ある方向のセキュリティ対策が強固な場合、攻撃者は回避可能な別の方向がないのか模索します。攻撃者のコードを何らかの方法でデバイスに組み込んで対策を回避するのが、単純ですが有効な方法でしょう。IoT デバイスはネットワーク上のサービスとデータのやり取りを行います。通信路の暗号化、サーバーとデバイスの相互認証などの対策を講じたとしても、IoT デバイス上にあるソフトウェアに攻撃者のコードを組み込むことで対策を回避してシステムに侵入される可能性があるのです。

セキュアブートは、起動ソフトウェアのディジタル署名を用いて正規ソフトウェアであることを確認してから起動する処理のことです。攻撃者によって作られた不正なコードを実行前に検出することができます。セキュアブートはチェーンオブトラスト (chain of trust) と表裏一体に実装されます。チェーンオブトラストとは、その名の通り、信頼を繋いでいく形態のことを指します。ルートオブトラストと呼ばれる基礎となる情報から枝葉のように繋がれた情報を認証していくことで、繋がれた個々のコンポーネントだけでなくシステムを信頼できるものにしてくれます。セキュアブートは、起動時にソフトウェアを順番に認証することで信頼を次に繋いでいるのです。セキュアブートの範囲をどこまでにするかによりますが、起動時に認証されたソフトウェアで別の情報を認証すれば、チェーンオブトラストを繋げていくことができます。また、IoT デバイスとクラウドサービスから構成されるような広範囲に及ぶシステムは特に信頼が必要になります。信頼できるセキュリティ基盤を構築するためには、構成するソフトウェアが正規のリリース物であることを確認することが重要になります。チェーンオブトラストを採用することでより信頼できる IoT システムとなり得るのです。

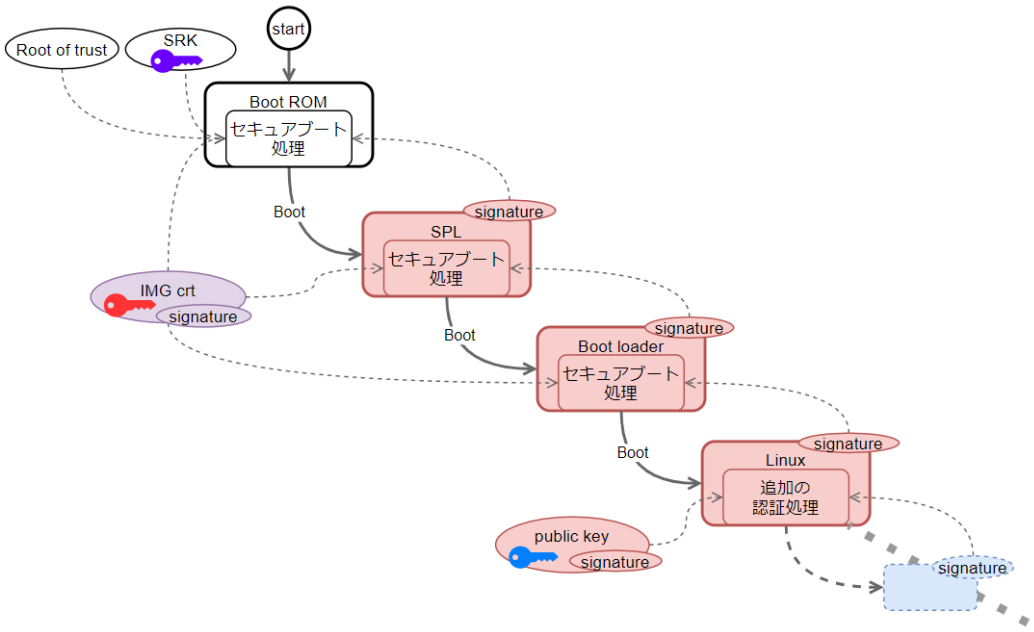


図 11.6 チェーンオブトラスト

では、どういったケースでセキュアブートを採用するべきでしょうか。セキュアブートはセキュアな組み込みデバイスを実現する上で最初のステップにするべき技術です。しかし、導入するのであればコスト面にも配慮することが必要でしょう。まず、製造時の追加コストが必要です。鍵等を書き込む工程が必要になります。ただ書き込めばよいわけではなく、漏洩があってはいけないので物理的な隔離などセキュアに書き込む必要があります。メンテナンスにも追加のコストが必要です。ソフトウェアのリリース時にはソフトウェアの署名が必要になります。こちらも、物理的な隔離などの漏洩、汚染対策が必要になります。また、どこまでやるかによりますが、定期的な鍵の更新、インシデントや製品寿命による鍵のリボーケーションなどのメンテナンスコストが必要になってきます。費用対効果を検討してからの導入をお勧めします。

11.5.2. HAB とは

HAB (High Assurance Booting) は、NXP Semiconductors が提供するセキュアブートの実装です。i.MX 8M Plus の BootROM には HABv4 が組み込まれます。デフォルトでは無効な状態になっていますが、一度、eFuse に情報を書き込むことでセキュアブートが有効になり、それ以降、有効な状態のまま変更不可能になります。HABv4 で規定する仕様では次の情報をブートローダーイメージに追加することで BootROM が起動時にブートローダーの認証を行います。

- CSF (Command Sequence File)、IVT (Image Vector Table)
- SRK (Super Root Key)、CSF、IMG 署名確認鍵
- イメージの署名

NXP Semiconductors は署名ツールとして CST (Code Signing Tool) をリリースしています。本来、署名範囲などは環境によって様々な実装がなされるべきなので署名ツール自体にはその辺りの仕様が含まれません。ブートローダーの実装仕様に依存します。Armadillo Base OS で採用される uboot-imx のセキュアブート処理では、Trusted Firmware-A (ATF)、OP-TEE OS、Linux カーネルイメージまでが認証の対象になります。署名に関する概要は以下のとおりです。

- 署名確認用の鍵は X.509 証明書に対応する

- 署名は RSA に対応する
  - 1024, 2048, 3072, 4096 bits
- 署名のダイジェストは SHA256 のみ

HAB の詳しい仕様は以下を参照してください。

i.MX Secure Boot on HABv4 Supported Devices https://www.nxp.com/search?keyword=AN4581

Encrypted Boot on HABv4 and CAAM Enabled Devices https://www.nxp.com/search?keyword=AN12056

11.5.3. 署名環境を構築する

NXP Semiconductors からリリースされる署名ツール (cts)、設定ファイルを準備してから、鍵の証明書を生成します。

11.5.3.1. 署名環境の構成

ディレクトリ構成の一部抜粋は以下のとおり。

```
├── cst_3.3.1 ❶
│   ├── crts ❷
│   ├── keys ❸
│   └── linux64/bin ❹
├── imx-boot
├── linux-[VERSION]
├── build-rootfs-[VERSION]
├── csf ❺
└── out ❻
```

❶ 本マニュアルでは v3.3.1 を利用した

❷ ブートローダーに組み込む自己証明書群が配置される

❸ 鍵の生成ツール、生成された鍵が配置される

❹ ビルド済み署名ツールが配置される

❺ CSF が配置される

❻ 署名済みイメージが配置される

11.5.3.2. 署名ツールを準備する

1. NXP Semiconductors から署名ツールをダウンロードします

   以下のサイトからダウンロードします。ここではユーザーが NXP Semiconductors のサイトで最新版を検索してダウンロードすることを想定しています。

CST の最新版 https://www.nxp.com/search?keyword=IMX_CST_TOOL_NEW

- ダウンロードには NXP Semiconductors サイトへのユーザー登録が必要になります
- 本マニュアル作成時点では v3.3.1 を利用しました

2. ツールを展開します

```
[PC ~]$ tar -xaf cst-3.3.1.tgz
```

11.5.3.3. シリアル番号、パスワードを設定する

PKI 証明書を作るために、証明書のシリアル番号、パスワードを事前に作っておく必要があります。シリアル番号ファイルとパスワードファイルを作ります。

シリアル番号とパスワードは任意の値を設定してください。

```
[PC ~]$ cd cst-3.1.1/keys
[PC ~/cst-3.1.1/keys]$ echo "12345678" > serial
[PC ~/cst-3.1.1/keys]$ echo "pass_phrase" > key_pass.txt
[PC ~/cst-3.1.1/keys]$ echo "pass_phrase" >> key_pass.txt
```

11.5.3.4. 鍵を生成する

セキュアブート用の PKI tree を作ります。生成する鍵とその証明書は以下のとおりです。

表 11.3 セキュアブート用の鍵と証明書

name	description	file name
CA 鍵ペア	ルート CA	CA1_sha256_[alg]_v3_ca_crt
SRK 鍵ペア	Super Root Key 用の鍵ペア	SRK[n]_sha256_[alg]_v3_ca_key
CSF 鍵ペア	CSF 用の鍵ペア	CSF[n]_sha256_[alg]_v3_usr_key
IMG 鍵ペア	イメージ用の鍵ペア	IMG[n]_sha256_[alg]_v3_usr_key
SRK 証明書	CA 鍵によって署名された SRK 公開鍵を含んだ証明書	SRK[n]_sha256_[alg]_v3_ca_crt
CSF 証明書	SRK によって署名された CSF 公開鍵を含んだ証明書	CSF[n]_sha256_[alg]_v3_usr_crt
IMG 証明書	SRK によって署名された IMG 公開鍵を含んだ証明書	IMG[n]_sha256_[alg]_v3_usr_crt

- [alg]: アルゴリズム
- [n]: n = 1,2,3,4

1. 鍵を生成する

   既存の CA 証明書を利用せずに、例として期限５年の４本の RSA-2048 の鍵を生成する。

```
[PC ~/cst-3.1.1/keys]$ ./hab4_pki_tree.sh -existing-ca n ¥ ❶
        -use-ecc n -kl 2048 ¥ ❷
        -duration 5 ¥ ❸
        -num-srk 4 ¥ ❹
        -srk-ca y ❺
```

❶ 既存の CA 証明書を利用するかどうか

❷ -use-ecc は ECC を利用するかどうか。-kl は鍵長。-use-ecc が n の場合は RSA になる

❸ 期限は５年

❹ SRK の数は４本

❺ 標準的な PKI ツリー

生成した鍵は今後も利用するものです。壊れにくい、セキュアなストレージにコピーしておくことをお勧めします。

鍵の更新は計画性を持って行ってください。たとえば開発時のみ利用する鍵、運用時に利用する鍵を使い分ける。また、鍵は定期的に更新が必要です。以下を参考にしてください。

BlueKrypt Cryptographic Key Length Recommendation https://www.keylength.com/

- 詳しくは cst 内の docs ディレクトリにある CST_UG.pdf をご参照ください
- 自己署名証明書の場合は、hab4_pki_tree.sh の実装として、openssl-req の subj オプションで CN=CA1_sha256_[alg]_v3_ca のみを設定している

2. 生成された鍵の確認

以下のファイルが生成されたことを確認してください。

```
cst-3.3.1
    ├── crts
    │   ├── CA1_sha256_2048_65537_v3_ca_crt.der
    │   ├── CA1_sha256_2048_65537_v3_ca_crt.pem
```

```
│   ├── CSF1_1_sha256_2048_65537_v3_usr_crt.der
│   ├── CSF1_1_sha256_2048_65537_v3_usr_crt.pem
│   ├── CSF2_1_sha256_2048_65537_v3_usr_crt.der
│   ├── CSF2_1_sha256_2048_65537_v3_usr_crt.pem
│   ├── CSF3_1_sha256_2048_65537_v3_usr_crt.der
│   ├── CSF3_1_sha256_2048_65537_v3_usr_crt.pem
│   ├── CSF4_1_sha256_2048_65537_v3_usr_crt.der
│   ├── CSF4_1_sha256_2048_65537_v3_usr_crt.pem
│   ├── IMG1_1_sha256_2048_65537_v3_usr_crt.der
│   ├── IMG1_1_sha256_2048_65537_v3_usr_crt.pem
│   ├── IMG2_1_sha256_2048_65537_v3_usr_crt.der
│   ├── IMG2_1_sha256_2048_65537_v3_usr_crt.pem
│   ├── IMG3_1_sha256_2048_65537_v3_usr_crt.der
│   ├── IMG3_1_sha256_2048_65537_v3_usr_crt.pem
│   ├── IMG4_1_sha256_2048_65537_v3_usr_crt.der
│   ├── IMG4_1_sha256_2048_65537_v3_usr_crt.pem
│   ├── SRK1_sha256_2048_65537_v3_ca_crt.der
│   ├── SRK1_sha256_2048_65537_v3_ca_crt.pem
│   ├── SRK2_sha256_2048_65537_v3_ca_crt.der
│   ├── SRK2_sha256_2048_65537_v3_ca_crt.pem
│   ├── SRK3_sha256_2048_65537_v3_ca_crt.der
│   ├── SRK3_sha256_2048_65537_v3_ca_crt.pem
│   ├── SRK4_sha256_2048_65537_v3_ca_crt.der
│   └── SRK4_sha256_2048_65537_v3_ca_crt.pem
└── keys
    ├── CA1_sha256_2048_65537_v3_ca_key.der
    ├── CA1_sha256_2048_65537_v3_ca_key.pem
    ├── CSF1_1_sha256_2048_65537_v3_usr_key.der
    ├── CSF1_1_sha256_2048_65537_v3_usr_key.pem
    ├── CSF2_1_sha256_2048_65537_v3_usr_key.der
    ├── CSF2_1_sha256_2048_65537_v3_usr_key.pem
    ├── CSF3_1_sha256_2048_65537_v3_usr_key.der
    ├── CSF3_1_sha256_2048_65537_v3_usr_key.pem
    ├── CSF4_1_sha256_2048_65537_v3_usr_key.der
    ├── CSF4_1_sha256_2048_65537_v3_usr_key.pem
    ├── IMG1_1_sha256_2048_65537_v3_usr_key.der
    ├── IMG1_1_sha256_2048_65537_v3_usr_key.pem
    ├── IMG2_1_sha256_2048_65537_v3_usr_key.der
    ├── IMG2_1_sha256_2048_65537_v3_usr_key.pem
    ├── IMG3_1_sha256_2048_65537_v3_usr_key.der
    ├── IMG3_1_sha256_2048_65537_v3_usr_key.pem
    ├── IMG4_1_sha256_2048_65537_v3_usr_key.der
    ├── IMG4_1_sha256_2048_65537_v3_usr_key.pem
    ├── SRK1_sha256_2048_65537_v3_ca_key.der
    ├── SRK1_sha256_2048_65537_v3_ca_key.pem
    ├── SRK2_sha256_2048_65537_v3_ca_key.der
    ├── SRK2_sha256_2048_65537_v3_ca_key.pem
    ├── SRK3_sha256_2048_65537_v3_ca_key.der
    ├── SRK3_sha256_2048_65537_v3_ca_key.pem
    ├── SRK4_sha256_2048_65537_v3_ca_key.der
    └── SRK4_sha256_2048_65537_v3_ca_key.pem
```

3. SRK テーブルと SRK ハッシュテーブルを生成する

   以下のコマンドを実行してください。

```
[PC ~/]$ cd cst-3.3.1/crts
[PC ~/cst-3.3.1/crts]$ ../linux64/bin/srktool --hab_ver 4 --digest sha256 ¥
        --table SRK_1_2_3_4_table.bin --efuses SRK_1_2_3_4_fuse.bin --fuse_format 1 ¥
        --certs "./SRK1_sha256_2048_65537_v3_ca_crt.pem,./
SRK2_sha256_2048_65537_v3_ca_crt.pem,./SRK3_sha256_2048_65537_v3_ca_crt.pem,./
SRK4_sha256_2048_65537_v3_ca_crt.pem"
```

以下の２つのファイルが生成されていれば成功です。

```
[PC ~/cst-3.3.1/crts]$ ls
:（省略）
SRK_1_2_3_4_fuse.bin
SRK_1_2_3_4_table.bin
:（省略）
```

11.5.4. セキュアブートを有効にする

生成した鍵と証明書を用いてセキュアブートを有効にしていきます。

セキュアブート有効後は、基本的に起動毎に署名確認が実行されるようになります。既に eMMC に書かれている署名されていないファームウェアは起動に失敗するようになります。そのため、署名済みイメージを書き込む作業が必要になります。セキュアブートを有効にする前に、一度「11.5.5. セキュアブート有効後のファームウェアの書き込みについて」 に目を通すことをお勧めします。

11.5.4.1. eFuse に SRK ハッシュを書き込む

セキュアブートを有効にするためには、i.MX 8M Plus の eFuse に SRK のハッシュ値を書く必要があります。i.MX 8M Plus の OCOTP (On-chip One-Time Programmable Element Controller) のレジスタ経由で書き込むことになります。uboot-imx には OCOTPA へのアクセスを行う fuse コマンドがあります。

1. 生成されたハッシュ値を確認する

   PC 上の cst ディレクトリで、以下のコマンドでハッシュ値を表示する

   ```
   [PC ~]$ cd cst/crts
   [PC ~/cst/crts]$ hexdump -e '1/4 "0x"' -e '1/4 "%X""¥n"' SRK_1_2_3_4_fuse.bin
   ```

   結果の例 (ハッシュ値は生成された鍵毎に異なります)

   ```
   0x72DBC22F
   0xDCAB0F6E
   0xBEBA9104
   0x35E61298
   0x768FA4B5
   0x2179343B
   ```

```
0x92BF13D4
0x461BAE7C
```

2. ハッシュ値を書き込む

eFuse は OTP なので一度作業を行うと変更はできません。注意して作業してください。

ハッシュ値は生成された鍵毎にことなります。例の値をそのまま書かないで下さい。

書き込みコマンドのヘルプです。

```
fuse prog [-y] <bank> <word> <hexval> [<hexval>...]
```

u-boot コマンドを利用して、確認したハッシュ値を eFuse に書いていきます。u-boot プロンプトを立ち上げてください。

以下に例を示します (あくまで例なのでそのまま書かないでください)。

```
u-boot=> fuse prog 6 0 0x72DBC22F
u-boot=> fuse prog 6 1 0xDCAB0F6E
u-boot=> fuse prog 6 2 0xBEBA9104
u-boot=> fuse prog 6 3 0x35E61298
u-boot=> fuse prog 7 0 0x768FA4B5
u-boot=> fuse prog 7 1 0x2179343B
u-boot=> fuse prog 7 2 0x92BF13D4
u-boot=> fuse prog 7 3 0x461BAE7C
```

１つのコマンドのログは以下のように出力されます。

```
u-boot=> fuse prog 6 0 0x72DBC22F
Programming bank 6 word 0x00000000 to 0x72dbc22f...
Warning: Programming fuses is an irreversible operation!
        This may brick your system.
        Use this command only if you are sure of what you are doing!

Really perform this fuse programming? <y/N>
y
```

3. SRK 領域をロックする

このままでは SRK 領域はビットの状態によっては書き込み可能な状態なのでロックする必要があります。

```
u-boot=> fuse prog 0 0 0x200
```

11.5.5. セキュアブート有効後のファームウェアの書き込みについて

セキュアブートを有効にした直後の初回書き込みは、再起動による再試行が可能な SD ブートを利用します。SD ブートを利用して署名済みイメージの書き込みが成功したのちには、Armadillo Base OS のファームウェアアップデート機能の利用が再び可能になります。

以下は初回とそれ以降のファームウェア書き込みの流れです。

セキュアブート有効直後の初回書き込み

1. 「11.5.6. ブートローダーイメージを署名する」
2. 「11.5.7. Linux カーネルイメージを署名する」
3. 「11.5.8. セキュアブート有効後の初回ファームウェアアップデート」 (特別な対応)

二回目以降の書き込み

1. 「11.5.6. ブートローダーイメージを署名する」
2. 「11.5.7. Linux カーネルイメージを署名する」
3. 「9.7. Armadillo のソフトウェアをアップデートする」

11.5.6. ブートローダーイメージを署名する

セキュアブートで利用するブートローダーイメージを署名していきます。Armadillo Base OS を利用した署名済みイメージを生成するまでの流れは以下のとおりです。

1. ブートローダーをビルドする
2. イメージ情報を抽出する
3. CSF 設定ファイルを書く
4. CST を実行する
5. ブートローダーイメージに CSF ファイルを書き込む

11.5.6.1. ブートローダーをビルドする

ソースコードの取得を行うために 「9.4. Armadillo のソフトウェアをビルドする」 の章を参考にしてビルド環境の構築からブートローダーのビルドまでを事前に行ってください。

1. uboot-imx のコンフィグを変更する

   x2_defconfig に CONFIG_IMX_HAB=y を追加する

```
[PC ~]$ cd imx-boot/uboot-imx/
[PC ~/imx-boot/uboot-imx]$ vi configs/x2_defconfig
CONFIG_IMX_HAB=y
```

2. ブートローダーを再ビルドする

   「11.4.4.3. ブートローダーを再ビルドする」 を参考にして、ブートローダーのバージョンを変更してから、ブートローダーのビルドを行ってください。

11.5.6.2. ブートローダーイメージの情報を取得する

NXP Semiconductors が提供する cst (code signing tool) はイメージを署名をする機能、CSF をつくる機能を持っているだけで、実際にどの部分を署名するのかは実装依存となっています。Armadillo Base OS では uboot-imx を採用しているので、uboot-imx のガイドに従うことになります。uboot-imx のガイドによると、ビルドログに含まれている情報を抜き出して CSF ファイルを作成するように書かれています。イメージの詳しい情報については 「図 11.7. ブートローダーの署名済みイメージ」 を参照してください。

取得する情報は以下のとおりです。

表 11.4 署名済みイメージ向けの情報

name	description	source	example
csf_off	SPL CSF の情報	ビルドログ OFFSET dump	csf_off 0x22e00
sld_csf_off	2nd loader CSF の情報	ビルドログ 中の OFFSET dump	sld_csf_off 0x59020
spl hab block	SPL IVT, SPL イメージの情報	ビルドログ 中の OFFSET dump	spl hab block: 0x91ffc0 0x0 0x22e00
sld hab block	2nd loader IVT (uboot, atf, op-tee, dtb が記載される) の情報	ビルドログ 中の OFFSET dump	sld hab block: 0x401fcdc0 0x58000 0x1020
fit_hab	2nd loader (uboot, atf, op-tee, dtb) イメージの情報	make print_fit_hab	0x40200000 0x5B000 0xCC1E8 …

uboot-imx のガイドラインは以下を参照してください。ソースコードにドキュメントがあります。**i.MX8M, i.MX8MM Secure Boot guide using HABv4** imx-boot/uboot-imx/doc/imx/habv4/guides/mx8m_secure_boot.txt

1. ビルドログから情報を抽出する

   ブートローダーのビルドログの一番最後のログに情報があります。抜き出してください。以下の結果は例となります。そのまま利用しないでください。

```
========= OFFSET dump =========
Loader IMAGE:
 header_image_off       0x0
 dcd_off                0x0
 image_off              0x40
 csf_off                0x22e00 ❶
 spl hab block:         0x91ffc0 0x0 0x22e00 ❷

Second Loader IMAGE:
 sld_header_off         0x58000
 sld_csf_off            0x59020 ❸
 sld hab block:         0x401fcdc0 0x58000 0x1020 ❹
```

❶ SPL CSF の情報

❷ SPL IVT と SPL image の情報

❸ 2nd loader CSF の情報

❹ 2nd loader IVT の情報

2. print_fit_hab から情報を抽出する

以下コマンドを実行して <1> の部分をメモする。以下の結果は例となります。そのまま利用しないでください。

```
[PC ~]$ cd imx-boot
[PC ~/imx-boot]$ make -C imx-mkimage/armadillo_x2 -f soc.mak SOC=iMX8MP print_fit_hab
make: Entering directory '/home/xxx/imx-boot/imx-mkimage/armadillo_x2'
./../scripts/dtb_check.sh imx8mp-evk.dtb evk.dtb
Use u-boot DTB: imx8mp-evk.dtb
./../scripts/pad_image.sh tee.bin
./../scripts/pad_image.sh bl31.bin
./../scripts/pad_image.sh u-boot-nodtb.bin evk.dtb
u-boot-nodtb.bin + evk.dtb are padded to 866464
TEE_LOAD_ADDR=0x56000000 ATF_LOAD_ADDR=0x00970000 VERSION=v2 ./print_fit_hab.sh 0x60000 ⏎
evk.dtb
0x40200000 0x5B000 0xCC1E8 ❶
0x402CC1E8 0x1271E8 0x76B8
0x970000 0x12E8A0 0xA150
0x56000000 0x1389F0 0x206600
make: Leaving directory '/home/xxx/imx-boot/imx-mkimage/armadillo_x2'
```

❶ 2nd loader (u-boot, atf, op-tee, dtb) のイメージ情報

11.5.6.3. CSF 設定ファイルを書く

1. csf_spl.txt を作る

以下のコマンドを実行して下さい。

```
[PC ~]$ mkdir -p csf
[PC ~]$ cd csf
[PC ~/csf]$ vi csf_spl.txt
```

以下は csf_spl.txt のサンプルです。注釈のある行以外はそのまま利用してください。

```
[Header]
Version = 4.3
Hash Algorithm = sha256
Engine = CAAM
Engine Configuration = 0
Certificate Format = X509
Signature Format = CMS

[Install SRK]
File = "cst-3.3.1/crts/SRK_1_2_3_4_table.bin" ❶
Source index = 0 ❷
```

```
[Install CSFK]
File = "cst-3.3.1/crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem" ❸

[Authenticate CSF]

[Install Key]
Verification index = 0
Target index = 2
File = "cst-3.3.1/crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem" ❹

[Authenticate Data]
Verification index = 2
Blocks = 0x91ffc0 0x0 0x22e00 "imx-boot/imx-boot_armadillo_x2" ❺

[Unlock]
Engine = CAAM
Features = MID
```

❶ SRK テーブルファイルを指定する

❷ SRK1 (0 はじまり) を指定する

❸ CSF1 ファイルを指定する

❹ IMG1 ファイルを指定する

❺ spl hab block の情報を書き込む

2. csf_fit.txt をつくる

以下のコマンドを実行して下さい。

```
[PC ~]$ cd csf
[PC ~/csf]$ vi csf_fit.txt
```

以下は csf_fit.txt のサンプルです。ここでは csf_spl と鍵は同一です。注釈のある行以外はそのまま利用してください。

```
[Header]
Version = 4.3
Hash Algorithm = sha256
Engine = CAAM
Engine Configuration = 0
Certificate Format = X509
Signature Format = CMS

[Install SRK]
File = "cst-3.3.1/crts/SRK_1_2_3_4_table.bin"
Source index = 0

[Install CSFK]
File = "cst-3.3.1/crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"

[Authenticate CSF]
```

```
[Install Key]
Verification index = 0
Target index = 2
File = "cst-3.3.1/crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"

[Authenticate Data]
Verification index = 2
Blocks = 0x401fcdc0 0x58000 0x1020 "imx-boot/imx-boot_armadillo_x2", ¥ ❶
0x40200000 0x5B000 0xCC1E8 "imx-boot/imx-boot_armadillo_x2", ¥ ❷
0x402CC1E8 0x1271E8 0x76B8 "imx-boot/imx-boot_armadillo_x2", ¥
0x970000 0x12E8A0 0xA150 "imx-boot/imx-boot_armadillo_x2", ¥
0x56000000 0x1389F0 0x206600 "imx-boot/imx-boot_armadillo_x2"
```

❶ sld hab block の情報を 1 行目

❷ それ以降の署名対象について fit_hab の情報を書き込む

11.5.6.4. CST を実行する

CSF 設定ファイルから CSF ファイルを作ります。

csf_spl.bin を作ります。

```
[PC ~]$ mkdir -p out
[PC ~]$ ./cst-3.3.1/linux64/bin/cst -i csf/csf_spl.txt -o out/csf_spl.bin
Install SRK
Install CSFK
Authenticate CSF
Install key
Authenticate data
CSF Processed successfully and signed data available in out/csf_spl.bin
```

csf_fit.bin を作ります。

```
[PC ~]$ ./cst-3.3.1/linux64/bin/cst -i csf/csf_fit.txt -o out/csf_fit.bin
Install SRK
Install CSFK
Authenticate CSF
Install key
Authenticate data
CSF Processed successfully and signed data available in out/csf_fit.bin
```

以下のファイルができていれば成功です。

```
[PC ~]$ ls out
csf_fit.bin  csf_spl.bin
```

11.5.6.5. ブートローダーイメージに CSF ファイルを書き込む

CSF をブートローダーイメージ内に予約された領域に書き込みます。seek で書き込み位置を合わせます。ここでのアドレス 0x22e0 と 0x59020 は例です。そのまま実行しないでください。

以下のコマンドを実行する。

```
[PC ~]$ cp imx-boot/imx-boot_armadillo_x2 out/signed_imx-boot_armadillo_x2
[PC ~]$ dd if=out/csf_spl.bin of=out/signed_imx-boot_armadillo_x2 seek=$((0x22e00)) ¥
        oflag=seek_bytes bs=4K conv=notrunc ❶
[PC ~]$ dd if=out/csf_fit.bin of=out/signed_imx-boot_armadillo_x2 seek=$((0x59020)) ¥
        oflag=seek_bytes bs=4K conv=notrunc ❷
```

❶ csf_off の情報を引数とする

❷ sld_csf_off の情報を引数とする

11.5.7. Linux カーネルイメージを署名する

セキュアブートで利用する Linux カーネルイメージを署名していきます。Armadillo Base OS を利用した署名済みイメージを生成するまでの流れは以下のとおりです。

1. イメージ情報を抽出して加工する
2. IVT を生成する
3. CSF 設定ファイルを書く
4. CST を実行する
5. Linux カーネルイメージに CSF を追加する

11.5.7.1. イメージ情報を抽出して加工する

署名確認対象となる Linux カーネルイメージが実際にメモリに展開されるサイズを取得します。ビルドされた Linux カーネルイメージ (Image) は bss 領域などの領域が省略されます。そのため、ファイルサイズではなく、イメージの内部に組み込まれている値を取得します。

イメージ情報を抽出するために Linux カーネルイメージが必要です。ブートローダーのようにセキュアブート処理を有効にするために再ビルドが必要なわけではないので、ビルド済みイメージを用意するか、「9.4. Armadillo のソフトウェアをビルドする」 を参考に Linux カーネルをビルドするかしてください。

1. イメージ情報を抽出する

   ファイルから１６バイト目を３２ビット分取得する。

   ```
   [PC ~]$ hexdump -s 16 -n 4 -e '"0x""%X""¥n"' linux-[VERSION]/arch/arm64/boot/Image
   ```

   以下は結果の例です。

   ```
   0x1E80000
   ```

2. イメージにパディングを追加する

   省略される領域をパディングとして追加します。--pad-to に取得したサイズを入力します。ここでは 0x1E80000 を入力します。

```
[PC ~]$ objcopy -I binary -O binary --pad-to 0x1E80000 --gap-fill=0x00 ¥
        linux-[VERSION]/arch/arm64/boot/Image out/Image_pad.bin
```

11.5.7.2. IVT を生成する

1. IVT 生成スクリプト (genIVT.pl) を編集する

コピーしてから編集します。

```
[PC ~]$ cp imx-boot/uboot-imx/doc/imx/habv4/script_examples/genIVT.pl out/
[PC ~]$ cd out
[PC ~/out]$ vi genIVT.pl
```

イメージの編集箇所は以下のとおりです。Self Pointer と CSF Pointer を抽出した実際のサイズを基に変更します。イメージの構成についての詳細は 「図 11.8. Linux カーネルの署名済みイメージ」 を参考にしてください。それ以外の行はそのまま利用してください。

```
#! /usr/bin/perl -w
use strict;
open(my $out, '>:raw', 'ivt.bin') or die "Unable to open: $!";
print $out pack("V", 0x412000D1); # Signature
print $out pack("V", 0x40480000); # Load Address (*load_address)
print $out pack("V", 0x0); # Reserved
print $out pack("V", 0x0); # DCD pointer
print $out pack("V", 0x0); # Boot Data
print $out pack("V", 0x42300000); # Self Pointer (*ivt) ❶
print $out pack("V", 0x42300020); # CSF Pointer (*csf) ❷
print $out pack("V", 0x0); # Reserved
close($out);
```

❶ 0x42300000 = [load_address] + [actual_image_size] = 0x40480000 + 0x1E80000

❷ 0x42300020 = [ivt] + 0x20 = 0x42300000 + 0x20

1. IVT を生成する

   以下のコマンドを実行してください。

   ```
   [PC ~/out]$ perl genIVT.pl
   ```

2. IVT を追加する

   パッドされた Linux カーネルイメージに IVT を追加します。以下のコマンドを実行してください。

   ```
   [PC ~/out]$ cat Image_pad.bin ivt.bin > Image_pad_ivt.bin
   ```

11.5.7.3. CSF 設定ファイルを書く

1. デバイスツリーブロブのサイズを調べる

以下のコマンドを実行してください。

```
[PC ~]$ printf '0x%X¥n' $(stat -c '%s' ¥
        linux-[VERSION]/arch/arm64/boot/dts/freescale/armadillo_iotg_g4.dtb)
```

以下は結果の例です。

```
0xD112
```

2. csf_linux.txt を作る

以下のコマンドを実行して下さい。

```
[PC ~]$ cd csf
[PC ~/csf]$ vi csf_linux.txt
```

以下は csf_linux.txt のサンプルです。注釈のある行以外はそのまま利用してください。

```
[Header]
Version = 4.5
Hash Algorithm = sha256
Engine = CAAM
Engine Configuration = 0
Certificate Format = X509
Signature Format = CMS

[Install SRK]
File = "cst-3.3.1/crts/SRK_1_2_3_4_table.bin"
Source index = 0

[Install CSFK]
File = "cst-3.3.1/crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"

[Authenticate CSF]

[Install Key]
Verification index = 0
Target index = 2
File = "cst-3.3.1/crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"

[Authenticate Data]
Verification index = 2
Blocks = 0x40480000 0x00000000 0x01E80020 "out/Image_pad_ivt.bin", ¥ ❶
0x45000000 0x00000000 0xD112 "linux-[VERSION]/arch/arm64/boot/dts/freescale/ ⏎
armadillo_iotg_g4.dtb" ❷
```

❶ Image_pad_ivt.bin のサイズを第３引数 (length) に入力する。ここでは 0x01E80020 の箇所。

❷ dtb のサイズを第３引数に入力する。ここでは 0xD112 の箇所。

11.5.7.4. CST を実行する

CSF 設定ファイルから CSF ファイルを作ります。

csf_linux.bin を作ります。

```
[PC ~]$ ./cst-3.3.1/linux64/bin/cst -i csf/csf_linux.txt -o out/csf_linux.bin
Install SRK
Install CSFK
Authenticate CSF
Install key
Authenticate data
CSF Processed successfully and signed data available in out/csf_linux.bin
```

11.5.7.5. Linux カーネルイメージに CSF を追加する

Linux カーネルイメージに CSF を追加することで署名が付加されることになります。

以下のコマンドを実行します。

```
[PC ~]$ cat out/Image_pad_ivt.bin out/csf_linux.bin > out/signed_Image
```

11.5.8. セキュアブート有効後の初回ファームウェアアップデート

セキュアブートを有効にすると、既に書かれている署名されていないファームウェアでは基本的に起動ができなくなります。そのためファームウェアアップデートによって署名済みイメージを書き込む作業が必要になります。ただし、ファームウェアアップデートには注意が必要です。アップデート途中の電源断などが原因で Armadillo を再起動してしまうと、既に書かれていたファームウェアを起動することになるので起動に失敗してしまいます。そのため、セキュアブートを有効にした直後の書き込みは署名済みイメージが書かれた SD ブートを利用して eMMC を書き込みます。起動メディアを壊すことがないので再起動による再試行が可能です。

流れは以下の通りです。

1. 署名イメージをつくる
   - 「11.5.6. ブートローダーイメージを署名する」 と 「11.5.7. Linux カーネルイメージを署名する」 を参考にブートローダーイメージと Linux kernel イメージを署名する
2. リポジトリ alpine/build-rootfs を取得する
   - 「9.4.3. Alpine Linux ルートファイルシステムをビルドする」 を参考にする
3. イメージを作る
4. イメージを microSD カードに書き込む
5. SD ブートでデバイスの eMMC を書き換える

11.5.8.1. イメージの作成と書き込み

PC 上で SD ブート用 microSD カードを作り、Armadillo に挿入して署名済みイメージを書いていきます。ここでは 「9.4.3. 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 を利用します。

1. Linux カーネルイメージを参照するように変更する

   以下のファイルを開きます。

```
[PC ~]$ vi ~/build-rootfs-[VERSION]/ax2/packages
```

   linux-at の行を消します。

```
:（省略）
dosfstools
atmark-x2-base

linux-at

crun
podman
:（省略）
```

2. Linux カーネルイメージとデバイスツリーを配置する

```
[PC ~]$ mkdir -p build-rootfs-[VERSION]/ax2/resources/boot
[PC ~]$ cp out/signed_Image build-rootfs-[VERSION]/ax2/resources/boot/Image
[PC ~]$ cp linux-[VERSION]/arch/arm64/boot/dts/freescale/armadillo_iotg_g4.dtb ¥
        build-rootfs-[VERSION]/ax2/resources/boot/
```

3. ルートファイルシステムをビルドする

   以下のコマンドを実行します。

```
[PC ~]$ cd build-rootfs-[VERSION]
[PC ~/build-rootfs-[VERSION]]$ sudo ./build_image.sh ¥
        -B ../out/signed_imx-boot_armadillo_x2
```

   成功すると以下の２つのファイルが生成されます。ファイル名に含まれる日付やバージョンはあくまで例です。

```
baseos-x2-3.14.3-at.1.20211124.img baseos-x2-3.14.3-at.1.20211124.tar.gz
```

4. イメージを作る

   以下のコマンドを実行します。

```
[PC ~/build-rootfs-[VERSION]]$ sudo ./build_image.sh ¥
        -B ../out/signed_imx-boot_armadillo_x2 ¥
        --installer ./baseos-x2-[VERSION].img
```

   成功すると以下のファイルが生成されます。ファイル名に含まれる日付やバージョンはあくまで例です。

```
baseos-x2-3.14.3-at.1.20211124-installer.img
```

5. SD ブート用 microSD カードをつくり、SD ブートする

   「9.5. SD ブートの活用」 を参考に baseos-x2-installer.img を microSD カードに書いて起動してください。このを microSD カードを利用すると SD ブートイメージのコピー (自分自身) を eMMC に書き込むように作られているので、セキュアブート用の署名付きイメージを書き込むことができます。最後にブートモードを戻して再起動すると署名済みイメージから起動するはずです。

11.5.8.2. セキュアブートの確認

Linux カーネルが立ち上がることを確認してください。Linux カーネルまで立ち上がらない場合、uboot-imx のコマンドで HAB の状態を確認することができます。

Linux 起動まで正常な起動ログ

```
:（省略）
Booting from mmc ...

## Checking Image at 40480000 ...
Unknown image format!
53522 bytes read in 22 ms (2.3 MiB/s)

Authenticate image from DDR location 0x40480000...

Secure boot enabled ❶

HAB Configuration: 0xcc, HAB State: 0x99
No HAB Events Found! ❷

## Flattened Device Tree blob at 45000000
   Booting using the fdt blob at 0x45000000
   Using Device Tree in place at 0000000045000000, end 0000000045010111

Starting kernel ...
:（省略）
```

❶ セキュアブートが有効な場合に表示されます

❷ 問題がない場合はイベントが表示されません

ブートローダーに問題がある場合の起動ログ

```
:（省略）
spl: ERROR:  image authentication unsuccessful
### ERROR ### Please RESET the board ###
:（省略）
```

Linux カーネルイメージに問題がある場合の起動ログ

```
:（省略）
Authenticate image from DDR location 0x40480000...
```

```
bad magic magic=0x14 length=0xa1 version=0x0
bad length magic=0x14 length=0xa1 version=0x0
bad version magic=0x14 length=0xa1 version=0x0
Error: Invalid IVT structure

Allowed IVT structure:
IVT HDR      = 0x4X2000D1
IVT ENTRY    = 0xXXXXXXXX
IVT RSV1      = 0x0
IVT DCD      = 0x0
IVT BOOT_DATA = 0xXXXXXXXX
IVT SELF      = 0xXXXXXXXX
IVT CSF      = 0xXXXXXXXX
IVT RSV2      = 0x0
Authenticate Image Fail, Please check ❶
:（省略）
```

❶　認証に失敗しています

u-boot コマンドの hab_status

問題がない場合

```
u-boot=> hab_status

Secure boot enabled

HAB Configuration: 0xcc, HAB State: 0x99
No HAB Events Found!
```

Linux カーネルイメージの署名確認で問題がある場合

```
u-boot=> hab_status

Secure boot disabled

HAB Configuration: 0xf0, HAB State: 0x66

--------- HAB Event 1 -----------------
event data:
        0xdb 0x00 0x14 0x45 0x33 0x0c 0xa0 0x00
        0x00 0x00 0x00 0x00 0x40 0x1f 0xdd 0xc0
        0x00 0x00 0x00 0x20

STS = HAB_FAILURE (0x33)
RSN = HAB_INV_ASSERTION (0x0C)
CTX = HAB_CTX_ASSERT (0xA0)
ENG = HAB_ENG_ANY (0x00)
```

11.5.9. SRK の無効化と切り替え

何らかのインシデント対応による鍵更新、また、鍵の定期更新などが必要な場合、その時点で利用している鍵を無効化して、別の鍵に切り替えることが可能です。ただし、その場合は複数 (i.MX 8M Plus の場合、最大４つ) の SRK が書かれていることが前提となります。

11.5.9.1. SRK の無効化 (revocation)

ここでは SRK1 から SRK2 に変更する例を説明します。

1. csf_spl.txt に revocation のロックを解除するブロックを追加します

   デフォルトでは eFuse の revoke レジスタはロックされているので書き込みできません。csf ファイルでロックを解除することができます。常にロックを解除すると攻撃者に悪用される可能性があるので通常時のセキュアブートではロックされるべきです。

   csf_spl.txt を開いて、最後に Unlock ブロックを追加します。

   ```
   [PC ~]$ vi csf/csf_spl.txt
   : (省略)
   [Unlock]
   Engine = OCOTP
   Features = SRK REVOKE
   ```

2. 署名済みイメージを書き込む

   「11.5.6. ブートローダーイメージを署名する」 を参考に署名済みイメージを生成して、イメージを書き込んでください。

3. 再起動

4. Unlock を確認する

   再起動時の uboot-imx のプロンプトを立ち上げてレジスタ値を確認します。以下のコマンドを実行してください。bit1 (SRK_REVOKE_LOCK) が落ちていると Unlock 状態です。

   ```
   u-boot=> md 0x30350050 1
   30350050: 00007dbc ❶
   ```

   ❶ 7dbc の bit 1 が経っていないので unlock 状態

5. SRK を無効化する

   ビットフィールドはビットは 0 はじまりで、鍵の番号は 1 はじまり (1,2,3,4) になります。bit0 が SRK1、bit1 が SRK2、bit2 が SRK3、bit3 が SRK4 です。

   以下のコマンドはあくまで例なので、そのまま実行しないで下さい。

   SRK1 を無効化する場合は以下のコマンドを実行してください。最終引数が無効化する鍵の設定値です。

   ```
   u-boot=> fuse prog 9 3 1
   ```

11.5.9.2. SRK の切り替え

ここでは SRK1 から SRK2 に変更する例を説明します。

1. SRK の変更

   csf_spl.txt を開いて、[Install SRK] ブロックのインデックス (0,1,2,3) と [Install CSFK], [Install Key] のファイル番号(1,2,3,4) を目的の SRK へ変更する。

```
[PC ~]$ vi csf/csf_spl.txt
: (省略)
[Install SRK]
File = "~/cst-3.3.1/crts/SRK_1_2_3_4_table.bin"
Source index = 1 ❶

[Install CSFK]
File = "~/cst-3.3.1/crts/CSF2_1_sha256_2048_65537_v3_usr_crt.pem" ❷

[Install Key]
File = "~/cst-3.3.1/crts/IMG2_1_sha256_2048_65537_v3_usr_crt.pem" ❸
```

   ❶ SRK のインデックス (0,1,2,3) を変更先のインデックスに切り替える

   ❷ CSF ファイルの番号 (1,2,3,4) を変更先の番号に切り替える

   ❸ IMG ファイルの番号 (1,2,3,4) を変更先の番号に切り替える

2. 再署名する

   「11.5.6. ブートローダーイメージを署名する」 を参考に署名済みイメージを作成してください。

11.5.10. 技術情報

11.5.10.1. 署名済みイメージと展開先

以下にブートローダーの署名済みイメージと展開先についての例を示します。緑色の部分が BootROM によって署名検証される部分、橙色の部分は SPL によって署名検証される部分になります。

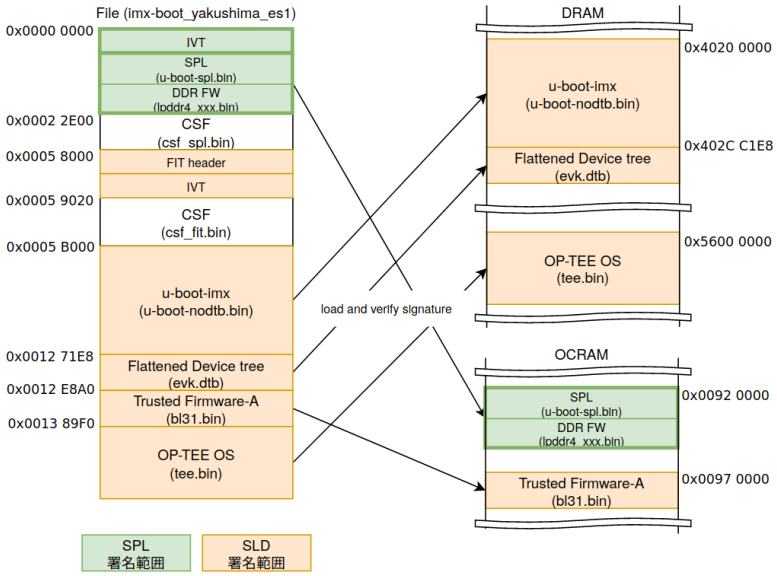


図 11.7 ブートローダーの署名済みイメージ

Linux の署名済みイメージと展開先の例は以下のとおりです。水色の部分は u-boot によって署名検証される部分になります。

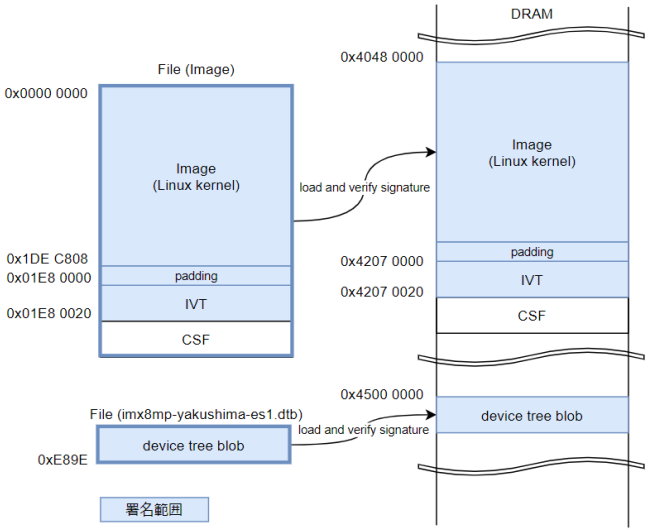


図 11.8 Linux カーネルの署名済みイメージ

11.5.10.2. セキュアブートのフロー

以下に SPL (Secondary Program Loader) のブートフローの概要を示します。点線で囲っている部分はセキュアブートで有効になる処理です。

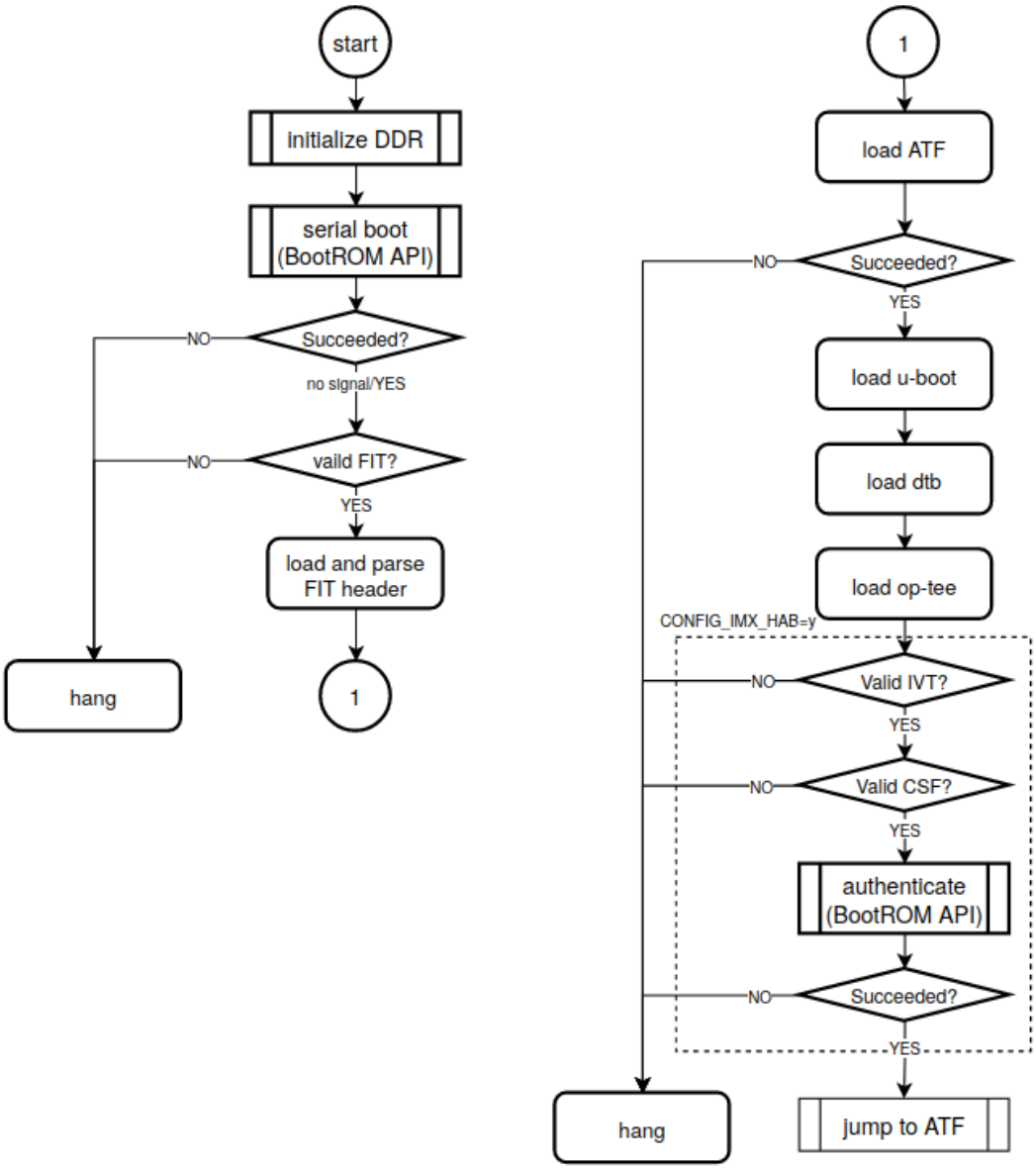


図 11.9 SPL セキュアブートのフロー

u-boot のブートフローの概要は以下のとおりです。SPL と同様に点線で囲っている部分はセキュアブートで有効になる処理です。

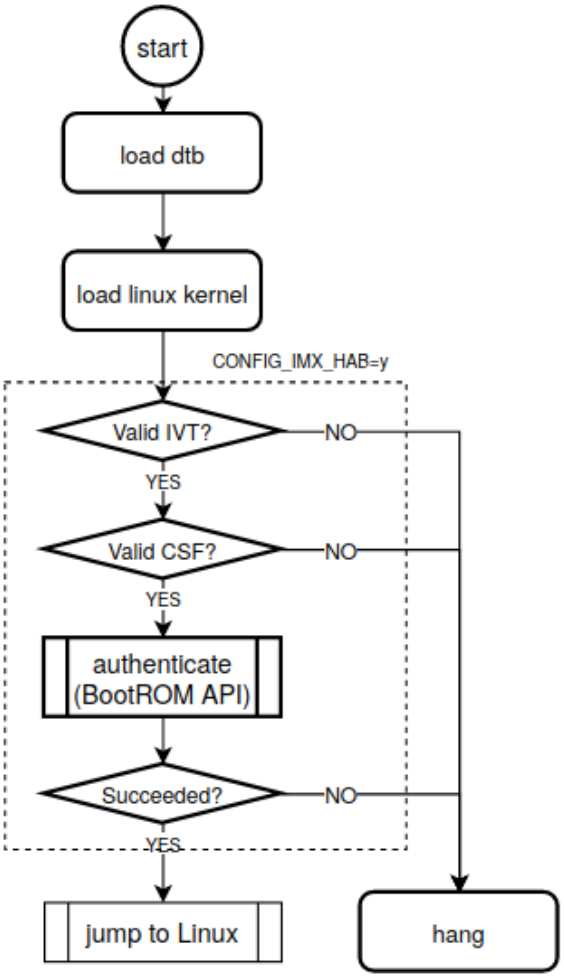


図 11.10 u-boot セキュアブートのフロー

12. 製品機能

本章では、Armadillo-IoT ゲートウェイ G4 で利用できる各種機能の仕様について説明します。

12.1. SD ホスト

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

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

機能	・カードタイプ: microSD/microSDHC/microSDXC/microSDIO ・バス幅: 1bit or 4bit ・スピードモード: Default Speed(26MHz), High Speed(52MHz), UHS-I(208MHz) ・カードディテクトサポート
デバイスファイル	・/dev/mmcblk1
関連するソースコード	・drivers/mmc/host/sdhci-esdhc-imx.c ・drivers/mmc/host/sdhci-of-esdhc.c
Device Tree ドキュメント	・Documentation/devicetree/bindings/mmc/fsl-imx-esdhc.yaml ・Documentation/devicetree/bindings/mmc/mmc-controller.yaml

カーネルコンフィギュレーション

```
Device Drivers  --->
  <*> MMC/SD/SDIO card support  --->                          <MMC>
    <*>   MMC block device driver                          <MMC_BLOCK>
    (32)    Number of minors per block device    <MMC_BLOCK_MINORS>
          *** MMC/SD/SDIO Host Controller Drivers ***
    <*>   Secure Digital Host Controller Interface support
                                                           <MMC_SDHCI>
    <*>     SDHCI platform and OF driver helper   <MMC_SDHCI_PLTFM>
    <*>       SDHCI OF support for the Freescale eSDHC controller
                                                   <MMC_SDHCI_OF_ESDHC>
    <*>       SDHCI support for the Freescale eSDHC/uSDHC i.MX
controller support
                                                <MMC_SDHCI_ESDHC_IMX>
```

12.2. Ethernet

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

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

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

機能

- 通信速度: 1000Mbps(1000BASE-T), 100Mbps(100BASE-TX), 10Mbps(10BASE-T)
- 通信モード: Full-Duplex(全二重), Half-Duplex(半二重) [1]
- Auto Negotiation サポート
- キャリア検知サポート
- リンク検出サポート

関連するソースコード

- drivers/net/ethernet/freescale/fec_main.c
- drivers/net/ethernet/stmicro/stmmac/dwmac-imx.c
- drivers/net/phy/micrel.c
- drivers/net/mdio/of_mdio.c

Device Tree ドキュメント

- Documentation/devicetree/bindings/net/fsl-fec.txt
- Documentation/devicetree/bindings/net/imx-dwmac.txt
- Documentation/devicetree/bindings/net/micrel-ksz90x1.txt
- Documentation/devicetree/bindings/net/net/ethernet-phy.yaml

ネットワークデバイス

- eth0 (LAN インターフェース 1)
- eth1 (LAN インターフェース 2)

カーネルコンフィギュレーション

```
Device Drivers  --->
  [*] Network device support  --->                    <NETDEVICES>
    [*]   Ethernet driver support  --->                 <ETHERNET>
      [*]   Freescale devices                <NET_VENDOR_FREESCALE>
     <*>     FEC ethernet controller (of ColdFire and some i.MX CPUs)
                                                              <FEC>
      [*]   STMicroelectronics devices         <NET_VENDOR_STMICRO>
      <*>     STMicroelectronics Multi-Gigabit Ethernet driver
                                                          <STMMAC_ETH>
      <*>       STMMAC Platform bus support       <STMMAC_PLATFORM>
      <*>         Generic driver for DWMAC          <DWMAC_GENERIC>
      <*>         NXP IMX8 DWMAC support               <DWMAC_IMX8>

    -*-   PHY Device support and infrastructure  --->      <PHYLIB>
      [*]   Support LED triggers for tracking link state
```

[1] 1000Mbps(1000BASE-T)は Half-Duplex に非対応です。

```
                                                          <LED_TRIGGER_PHY>
      <*>   Micrel PHYs                                        <MICREL_PHY>
```

12.3. USB ホスト

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

Armadillo-IoT ゲートウェイ G4 では、USB インターフェース(CON4)が USB1 を利用しています。USB2 には「12.4. USB ハブ」に示す USB2422 が接続されています。

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

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

関連するソースコード
- drivers/usb/dwc3/dwc3-imx8mp.c
- drivers/phy/freescale/phy-fsl-imx8mq-usb.c

Device Tree ドキュメント
- Documentation/devicetree/bindings/usb/dwc3-imx8mp.txt
- Documentation/devicetree/bindings/phy/fsl,imx8mq-usb-phy.txt

カーネルコンフィギュレーション

```
Device Drivers  --->
  [*] USB support  --->                                  <USB_SUPPORT>
    <*>   Support for Host-side USB                            <USB>
          *** USB Host Controller Drivers ***
    <*>   xHCI HCD (USB 3.0) support                   <USB_XHCI_HCD>
    -*-     Generic xHCI driver for a platform device
                                                    <USB_XHCI_PLATFORM>
    <*>   DesignWare USB3 DRD Core Support                 <USB_DWC3>
            DWC3 Mode Selection (Dual Role mode)  --->
                                                   <USB_DWC3_DUAL_ROLE>
    <*>     NXP iMX8MP Platform                      <USB_DWC3_IMX8MP>
  PHY Subsystem  --->
    -*- PHY Core                                           <GENERIC_PHY>
    <*> Freescale i.MX8M USB3 PHY                <PHY_FSL_IMX8MQ_USB>
```

12.4. USB ハブ

Armadillo-IoT ゲートウェイ G4 には、Microchip 製 USB2422 が搭載されています。USB2422 は、「14.2.11. CON11、CON12 (拡張インターフェース)」に接続されています。

機能
- USB specification rev 2.0 準拠

・転送レート: high-speed(480 Mbps), full-speed(12 Mbps), low-speed(1.5 Mbps)

関連するソースコード
・drivers/usb/misc/usb251xb.c

Device Tree ドキュメント
・Documentation/devicetree/bindings/usb/usb251xb.txt

カーネルコンフィギュレーション

```
Device Drivers  --->
  [*] USB support  --->                                    <USB_SUPPORT>
          *** USB Miscellaneous drivers ***
    <*>   USB251XB Hub Controller Configuration Driver
                                                           <USB_HUB_USB251XB>
```

12.5. UART

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

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

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

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

関連するソースコード
・drivers/tty/serial/imx.c
・drivers/tty/serial/imx_earlycon.c

Device Tree ドキュメント
・Documentation/devicetree/bindings/serial/fsl-imx-uart.yaml

デバイスファイル

シリアルインターフェース	デバイスファイル
UART1	/dev/ttymxc0
UART2	/dev/ttymxc1

カーネルコンフィギュレーション

```
Device Drivers  --->
  Character devices  --->
    [*] Enable TTY                                         <TTY>
        Serial drivers  --->
```

```
          <*> IMX serial port support                 <SERIAL_IMX>
          <*>   Console on IMX serial port     <SERIAL_IMX_CONSOLE>
          [*] Earlycon on IMX serial port     <SERIAL_IMX_EARLYCON>
```

12.6. HDMI

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

Armadillo-IoT ゲートウェイ G4 は、HDMI 対応ディスプレイへの画像出力及び、音声出力をサポートしています。 Linux では、それぞれ DRM(Direct Rendering Manager)デバイス [2]、ALSA(Advanced Linux Sound Architecture)デバイスとして利用することができます。

機能(画像出力)
- 最大解像度: 4096x2160 ピクセル
- 最大ドットクロック: 297MHz
- カラーフォーマット: RGB888(24bit)
- 走査方式: プログレッシブ

上記を満していても、画像出力できない場合があります。次の VIC [3] は非対応である為、画像出力できません。

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

機能(音声出力)
- サンプリング周波数: 32kHz, 44.1kHz, 48kHz, 88.2kHz, 96kHz, 176.4kHz, 192kHz
- チャンネル数: 2
- フォーマット: Signed 24/32 bit, Little-endian

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

フレームバッファデバイスは「12.7. LVDS」と共通です。

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

[2]フレームバッファデバイスとして利用することもできます。
[3]CEA-861 規格の VIC コード。

関連するソースコード

- drivers/gpu/drm/imx/dw_hdmi-imx.c
- drivers/phy/freescale/phy-fsl-samsung-hdmi.c
- drivers/clk/imx/clk-blk-ctrl.c
- drivers/gpu/drm/imx/imx8mp-hdmi-pavi.c
- drivers/gpu/imx/lcdifv3/lcdifv3-common.c
- sound/soc/fsl/imx-cdnhdmi.c
- sound/soc/fsl/fsl_aud2htx.c

Device Tree ドキュメント

- Documentation/devicetree/bindings/phy/fsl,samsung-hdmi-phy.yaml
- Documentation/devicetree/bindings/sound/imx-audio-cdnhdmi.txt
- Documentation/devicetree/bindings/sound/fsl,aud2htx.txt

カーネルコンフィギュレーション

```
Device Drivers  --->
  Graphics support  --->
    <*> i.MX LCDIFV3 core support                <IMX_LCDIFV3_CORE>
    -*- NXP i.MX8MP HDMI Audio Video (PVI/PAI)   <IMX8MP_HDMI_PAVI>
    <*> Freescale i.MX DRM HDMI                      <DRM_IMX_HDMI>
  -*- Common Clock Framework  --->                     <COMMON_CLK>
    <*>   IMX8MP CCM Clock Driver                      <CLK_IMX8MP>
  <*> Sound card support  --->                              <SOUND>
    <*>   Advanced Linux Sound Architecture  --->             <SND>
      <*>   ALSA for SoC audio support  --->              <SND_SOC>
        SoC Audio for Freescale CPUs  --->
          <*> SoC Audio support for i.MX boards with CDN HDMI port
                                                  <SND_SOC_IMX_CDNHDMI>
  PHY Subsystem  --->
    -*- AUDIO TO HDMI TX module support       <SND_SOC_FSL_AUD2HTX>
    <*> Samsung HDMI PHY support            <PHY_SAMSUNG_HDMI_PHY>
```

以下のコマンドを実行することで映像出力の信号を停止することができます。

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

映像出力を行いたい場合は以下のコマンドを実行します。

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

12.7. LVDS

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

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

Linux では、画像出力を DRM(Direct Rendering Manager)デバイス [4]として利用することができます。

機能	・最大解像度: 1366x768p60 ・最大ピクセルクロック: 80MHz ・レーン数: 4(データ), 1(クロック) ・カラーフォーマット: RGB666(18bit), RGB888(24bit) ・カラーマッピング: JEIDA, VESA
デバイスファイル	・/dev/dri/card1 (DRM) ・/dev/fb0 (フレームバッファ)

フレームバッファデバイスは「12.6. HDMI」と共通です。

sysfs DRM クラスディレクトリ	・/sys/class/drm/card1-LVDS-1
関連するソースコード	・drivers/gpu/drm/imx/imx8mp-ldb.c ・drivers/phy/freescale/phy-fsl-imx8mp-lvds.c ・drivers/gpu/imx/lcdifv3/lcdifv3-common.c
Device Tree ドキュメント	・Documentation/devicetree/bindings/display/imx/ldb.txt ・Documentation/devicetree/bindings/phy/fsl,imx8mp-lvds-phy.yaml ・Documentation/devicetree/bindings/display/panel/lvds.yaml

カーネルコンフィギュレーション

```
Device Drivers  --->
  Graphics support  --->
    <*> i.MX LCDIFV3 core support                <IMX_LCDIFV3_CORE>
    <*> DRM Support for Freescale i.MX                    <DRM_IMX>
    <*>   Support for i.MX8mp LVDS displays        <DRM_IMX8MP_LDB>
```

[4]フレームバッファデバイスとして利用することもできます。

```
  PHY Subsystem  --->
    <*> Freescale i.MX8MP LVDS PHY            <PHY_FSL_IMX8MP_LVDS>
```

12.8. MIPI CSI-2

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

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

Linux では、カメラ [5] からの画像入力を V4L2(Video4Linux2)デバイスとして利用することができます。

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

デバイスファイル
- /dev/video2 [6]

関連するソースコード
- drivers/staging/media/imx/imx8-mipi-csi2-sam.c
- drivers/staging/media/imx/imx8-media-dev.c
- drivers/staging/media/imx/imx8-isi-core.c
- drivers/staging/media/imx/imx8-isi-cap.c

Device Tree ドキュメント
- Documentation/devicetree/bindings/media/imx8-media-dev.txt

カーネルコンフィギュレーション

```
Device Drivers  --->
  [*] Staging drivers  --->                                   <STAGING>
    [*]   Media staging drivers  --->                  <STAGING_MEDIA>
      <*>   i.MX V4L2 media core driver           <VIDEO_IMX_CAPTURE>
              i.MX8QXP/QM Camera ISI/MIPI Features support  --->
        <*> IMX8 MIPI CSI2 SAMSUNG Controller  <IMX8_MIPI_CSI2_SAM>
        <*> IMX8 Image Sensor Interface Core Driver <IMX8_ISI_CORE>
```

[5]Armadillo-IoT ゲートウェイ G4 にカメラは付属していません。
[6]UVC カメラなどを接続して V4L2 デバイスを追加している場合は、番号が異なる可能性があります。

```
        <*> IMX8 Image Sensor Interface Capture Device Driver
                                                <IMX8_ISI_CAPTURE>
        <*> IMX8 Media Device Driver            <IMX8_MEDIA_DEVICE>
```

12.9. PCI Express

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

機能

- PCI Express Base Specification, Revision 4.0, Version 0.7 準拠
- PCI Local Bus Specification, Revision 3.0 準拠
- PCI Bus Power Management Specification, Revision 1.2 3.0 準拠
- PCI Express Card Electromechanical Specification, Revision 1.1 準拠
- リンク幅: x1
- 転送レート: 8.0GT/s
- 割り込み通知方式: MSI

関連するソースコード

- drivers/pci/controller/dwc/pci-imx6.c
- drivers/phy/freescale/phy-fsl-imx8-pcie.c

Device Tree ドキュメント

- Documentation/devicetree/bindings/pci/fsl,imx6q-pcie.txt
- Documentation/devicetree/bindings/phy/fsl,imx-pcie-phy.txt

カーネルコンフィギュレーション

```
Device Drivers  --->
  [*] PCI support  --->                                          <PCI>
    PCI controller drivers  --->
      DesignWare PCI Core Support  --->
        -*- Freescale i.MX6/7/8 PCIe driver                <PCI_IMX6>
  PHY Subsystem  --->
    <*> Freescale i.MX PCIE PHY                      <PHY_FSL_IMX_PCIE>
```

12.10. リアルタイムクロック

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

機能

- アラーム割り込みサポート

デバイスファイル

- /dev/rtc (/dev/rtc0 へのシンボリックリンク)
- /dev/rtc0 (RV-8803-C7)
- /dev/rtc1 (SNVS_HP Real Time Counter)

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

関連するソースコード
- drivers/rtc/rtc-rv8803.c
- drivers/rtc/rtc-snvs.c

Device Tree ドキュメント
- Documentation/devicetree/bindings/rtc/epson,rx8900.txt
- Documentation/devicetree/bindings/crypto/fsl-sec4.txt

カーネルコンフィギュレーション

```
Device Drivers --->
  [*] Real Time Clock  --->                                  <RTC_CLASS>
    (rtc0)  RTC used to synchronize NTP adjustment
                                                  <RTC_SYSTOHC_DEVICE>
    <*>   Micro Crystal RV8803, Epson RX8900       <RTC_DRV_RV8803>
    <*>   Freescale SNVS RTC support                <RTC_DRV_SNVS>
```

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

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

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。

12.11. ユーザースイッチとイベント信号

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

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 12.1 キーコード

ユーザースイッチ	キーコード	イベントコード	X11 キーコード
SW1	KEY_PROG1	148	XF86Launch1
予約	KEY_PROG2	149	XF86Launch2

ユーザースイッチ	キーコード	イベントコード	X11 キーコード
予約	KEY_PROG3	202	XF86Launch3
予約	KEY_PROG4	203	XF86Launch4
PWR_OFF	KEY_POWER	116	XF86PowerOff
REBOOT	KEY_RESET	408	なし

デバイスファイル
- /dev/input/by-path/platform-gpio-keys-event [7]

関連するソースコード
- drivers/input/keyboard/gpio_keys.c

Device Tree ドキュメント
- Documentation/devicetree/bindings/input/gpio-keys.yaml

カーネルコンフィギュレーション

```
Device Drivers  --->
  Input device support  --->
    -*- Generic input layer (needed for keyboard, mouse, ...)
                                                          <INPUT>
      [*]   Keyboards  --->                          <INPUT_KEYBOARD>
      <*>   GPIO Buttons                              <KEYBOARD_GPIO>
```

12.12. LED

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

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

sysfs LED クラスディレクトリ
- /sys/class/leds/led1

関連するソースコード
- drivers/leds/leds-gpio.c

Device Tree ドキュメント
- Documentation/devicetree/bindings/leds/leds-gpio.yaml

カーネルコンフィギュレーション

```
Device Drivers  --->
  [*] LED Support  --->                                  <NEW_LEDS>
    <*>   LED Support for GPIO connected LEDs           <LEDS_GPIO>
```

12.13. I2C

Armadillo-IoT ゲートウェイ G4 の I2C インターフェースは、i.MX 8M Plus の I2C(I2C Controller)を利用しています。また、i2c-gpio を利用することで、I2C バスを追加することができます。

Armadillo-IoT ゲートウェイ G4 で利用している I2C バスと、接続される I2C デバイスを次に示します。

表 12.2 I2C デバイス

I2C バス	I2C デバイス	
	アドレス	デバイス名
0(I2C1)	0x25	PCA9450(PMIC)

[7]USB キーボードなどを接続してインプットデバイスを追加している場合は、番号が異なる可能性があります

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

Armadillo-IoT ゲートウェイ G4 の標準状態では、CONFIG_I2C_CHARDEV が有効となっているためユーザードライバで I2C デバイスを制御することができます。ユーザードライバを利用する場合は、Linux カーネルで I2C デバイスに対応するデバイスドライバを無効にする必要があります。

機能
- 最大クロック: 384kHz

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

/dev/i2c-6 は、HDMI DDC です。

関連するソースコード
- drivers/i2c/busses/i2c-imx.c

Device Tree ドキュメント
- Documentation/devicetree/bindings/i2c/i2c-imx.yaml

カーネルコンフィギュレーション

```
Device Drivers  --->
  I2C support  --->
    -*- I2C support                                         <I2C>
      <*>   I2C device interface                      <I2C_CHARDEV>
      I2C Hardware Bus support  --->
        <*> IMX I2C interface                           <I2C_IMX>
```

12.14. GPIO

Armadillo-IoT ゲートウェイ G4 の GPIO は、i.MX 8M Plus の GPIO(General Purpose Input/Output)および、ON Semiconductor 製 FXL6408(GPIO エキスパンダー)を利用しています。

関連するソースコード
- drivers/gpio/gpio-mxc.c
- drivers/gpio/gpio-fxl6408.c

Device Tree ドキュメント
- Documentation/devicetree/bindings/gpio/fsl-imx-gpio.yaml
- Documentation/devicetree/bindings/gpio/gpio-fxl6408.txt

デバイスファイル

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

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

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

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

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

カーネルコンフィギュレーション

```
Device Drivers  --->
  [*] GPIO Support  --->                                    <GPIOLIB>
    (512) Maximum number of GPIOs for fast path
                                              <GPIOLIB_FASTPATH_LIMIT>
    I2C GPIO expanders  --->
       <*> FXL6408 I2C GPIO expander                  <GPIO_FXL6408>
    Memory mapped GPIO drivers  --->
      -*- i.MX GPIO support                                <GPIO_MXC>
```

12.15. 温度センサー

Armadillo-IoT ゲートウェイ G4 の温度センサーは、i.MX 8M Plus の TMU(Thermal Monitoring Unit)を利用しています。CPU(Arm Cortex-A53)周辺温度と、SoC(ANAMIX 内部)温度を測定することができます。

起動直後の設定では、ARM または SoC の測定温度が 105°C 以上になった場合、Linux カーネルはシステムを停止します。

機能
- 測定温度範囲: -40～+105°C

sysfs thermal クラスディレクトリ
- /sys/class/thermal/thermal_zone0 (CPU)
- /sys/class/thermal/thermal_zone1 (SoC)

関連するソースコード
- drivers/thermal/imx8mm_thermal.c

Device Tree ドキュメント
- Documentation/devicetree/bindings/thermal/imx8mm-thermal.yaml

カーネルコンフィギュレーション

```
Device Drivers  --->
  -*- Thermal drivers  --->                                 <THERMAL>
```

```
    <*>   Temperature sensor driver for Freescale i.MX8MM SoC
                                                    <IMX8MM_THERMAL>
```

12.16. ウォッチドッグタイマー

Armadillo-IoT ゲートウェイ G4 のウォッチドッグタイマーは、i.MX 8M Plus の WDOG(Watchdog Timer)を利用しています。

ウォッチドッグタイマーは、U-Boot によって有効化されます。標準状態でタイムアウト時間は 10 秒に設定されます。Linux カーネルは、ウォッチドッグタイマードライバの初期化時にタイムアウト時間を 60 秒に再設定します。

何らかの要因でウォッチドッグタイマーのキックができなくなりタイムアウトすると、システムリセットが発生します。

関連するソースコード
- drivers/watchdog/imx2_wdt.c

Device Tree ドキュメント
- Documentation/devicetree/bindings/watchdog/fsl-imx-wdt.yaml

カーネルコンフィギュレーション

```
Device Drivers --->
  [*] Watchdog Timer Support  --->                          <WATCHDOG>
    <*>   IMX2+ Watchdog                                    <IMX2_WDT>
```

ウォッチドッグタイマーの設定変更は、ioctl システムコール経由で行うことができます。詳細な情報については、Linux カーネルのソースコードに含まれているドキュメント(Documentation/watchdog/watchdog-api.rst)を参照してください。

ウォッチドッグタイマーを停止することはできません。

12.17. パワーマネジメント

Armadillo-IoT ゲートウェイ G4 のパワーマネジメント機能は、Linux の SPM(System Power Management)および DPM(Device Power Management)を利用しています。パワーマネジメント状態を省電力モードに遷移させることにより、Armadillo-IoT ゲートウェイ G4 の消費電力を抑えることができます。

パワーマネジメント状態を省電力モードに遷移させると、アプリケーションの実行は一時停止し、Linux カーネルはサスペンド状態となります。起床要因が発生すると、Linux カーネルのリジューム処理が行われた後、アプリケーションの実行を再開します。

sysfs ファイル
- /sys/power/state

関連するソースコード
- kernel/power/

カーネルコンフィギュレーション

```
Power management options  --->
  [*] Suspend to RAM and standby                              <SUSPEND>
  -*- Device power management core functionality                   <PM>
```

Armadillo-IoT ゲートウェイ G4 が対応するパワーマネジメント状態と、/sys/power/state に書き込む文字列の対応を次に示します。

表 12.3 対応するパワーマネジメント状態

パワーマネジメント状態	文字列	説明
Suspend-to-RAM	mem	Suspend-to-Idle よりも消費電力を抑えることができる
Suspend-to-Idle	freeze	suspend-to-ram よりも短時間で復帰することができる

起床要因として利用可能なデバイスは次の通りです。

USB コンソールインターフェース(CON6)

起床要因: データ受信

有効化:

```
[armadillo ~]# echo enabled > /sys/class/tty/
ttymxc1/power/wakeup
```

USB インターフェース(CON4)

起床要因: USB デバイスの挿抜

有効化:

```
[armadillo ~]# echo enabled > /sys/devices/
platform/soc@0/32f10100.usb/power/wakeup
[armadillo ~]# echo enabled > /sys/bus/usb/devices/
usb1/power/wakeup
```

RTC(SNVS_HP Real Time Counter)

起床要因: アラーム割り込み

有効化:

```
デフォルトで有効化されています
```

RTC(RV-8803-C7)

起床要因: アラーム割り込み

有効化:

```
デフォルトで有効化されています
```

13. ソフトウェア仕様

13.1. SWUpdate

13.1.1. SWUpdate とは

SWUpdate は、デバイス上で実行されるプログラムで、ネットワーク/ストレージ経由でデバイスのソフトウェアを更新することができます。Stefano Babic, DENX software engineering, Germany によってオープンソースで開発が進められています。

Armadillo-IoT ゲートウェイ G4 では、SWUpdate を利用することで次のような機能を実現しています。

- A/B アップデート(アップデートの 2 面化)
- リカバリーモード
- ソフトウェアの圧縮、暗号化、署名付与
- Web サーバー機能
- hawkBit への対応
- ダウングレードの禁止

13.1.2. swu パッケージ

swu パッケージは、SWUpdate 独自のソフトウェアの配布フォーマットです。SWUpdate では、1 回のアップデートは 1 つの swu パッケージで行われます。

swu パッケージには、次のような様々なものを含めることができます。

- アップデート対象のイメージファイル
- アップデート対象のイメージファイルのチェックサム
- アップデート前後に実行するスクリプト
- 書き込み先ストレージの情報
- U-Boot 環境変数の書き換え情報
- ソフトウェアのバージョン情報
- etc…

13.1.3. A/B アップデート(アップデートの 2 面化)

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

常に使用していない方のパーティションを書き換えるため次の特徴を持ちます。

- ○ アップデートによって動作中のソフトウェアは破壊されない
- ○ 書き込みが電源断などで中断後しても、すぐに復帰出来る
- ○ 機器が動作中に書き込みが出来る
- × 使用 Flash メモリ量が増える

13.1.4. リカバリーモード

リカバリーモードは、不測の事態によりシステムが起動できなくなった際に、A/B アップデートの仕組みとは別のリカバリー用のシステムが起動するモードです。

万が一、システムが起動できなくなったとしても、ネットワーク経由でのシステム復旧が可能です。

13.2. hawkBit

13.2.1. hawkBit とは

hawkBit は、サーバー上で実行されるプログラムで、ネットワーク経由でデバイスのソフトウェアを更新(配信)することができます。

hawkBit は次のような機能を持っています。

- ソフトウェアの管理
- デバイスの管理
  - デバイス認証 (セキュリティトークン、証明書)
  - デバイスのグループ化
- アップデート処理の管理
  - 進捗のモニタリング
  - スケジューリング、強制アップデート
- RESTful API での直接操作

13.2.2. データ構造

hawkBit は、配信するソフトウェアを次のデータ構造で管理します。

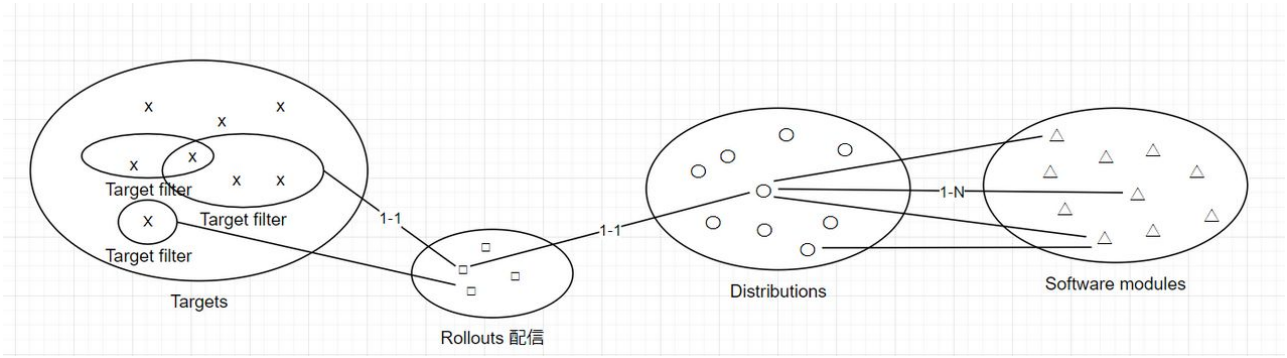


図 13.1 hawkBit が扱うソフトウェアのデータ構造

14. ハードウェア仕様

14.1. 電気的仕様

14.1.1. 絶対最大定格

表 14.1 絶対最大定格

項目	記号	Min.	Max.	単位	備考
電源電圧	VIN	-0.3	13.2	V	-
入出力電圧	VI,VO (VDD_3V3, VDD_1V8, NVCC_SNVS_1V8)	-0.3	OVDD+0.3	V	OVDD=VDD_3V3, VDD_1V8, NVCC_SNVS_1V8
	VI,VO(M2_3V3)	-0.5	4.6	V	-
	VI,VO(VEXT_3V3)	-0.5	7.0	V	-
USB コンソール電源電圧	VBUS_CNSL	-0.3	5.8	V	-
RTC バックアップ電源電圧	RTC_BAT	-0.3	5.5	V	-
動作温度範囲	Topr	-20	70	℃	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。

14.1.2. 推奨動作条件

表 14.2 推奨動作条件

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

14.1.3. 電源出力仕様

表 14.3 電源出力仕様

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

14.1.4. 入出力インターフェースの電気的仕様

表 14.4 拡張入出力ピンの電気的仕様(OVDD=VDD_3V3, VDD_1V8)

項目	記号	Min.	Typ.	Max.	単位	備考
ハイレベル出力電圧	VOH (VDD_1V8)	0.8xOVDD	-	OVDD	V	IOH = 1.6/3.2/6.4/9.6mA
	VOH (VDD_3V3)	0.8xOVDD	-	OVDD	V	IOH = 2/4/8/12mA
ローレベル出力電圧	VOL (VDD_1V8)	0	-	0.2xOVDD	V	IOL = 1.6/3.2/6.4/9.6mA
	VOL (VDD_3V3)	0	-	0.2xOVDD	V	IOL = 2/4/8/12mA
ハイレベル入力電圧	VIH	0.7×OVDD	-	OVDD+0.3	V	-
ローレベル入力電圧	VIL	-0.3	-	0.3×OVDD	V	-
Pull-up 抵抗 (VDD_1V8)	-	12	22	49	kΩ	-
Pull-down 抵抗 (VDD_1V8)	-	13	23	48	kΩ	-
Pull-up 抵抗 (VDD_3V3)	-	18	37	72	kΩ	-
Pull-down 抵抗 (VDD_3V3)	-	24	43	87	kΩ	-

表 14.5 拡張入出力ピンの電気的仕様(OVDD=VEXT_3V3)

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

14.1.5. 電源回路の構成

Armadillo-IoT ゲートウェイ G4 の電源回路の構成は「図 14.1. 電源回路の構成」のとおりです。

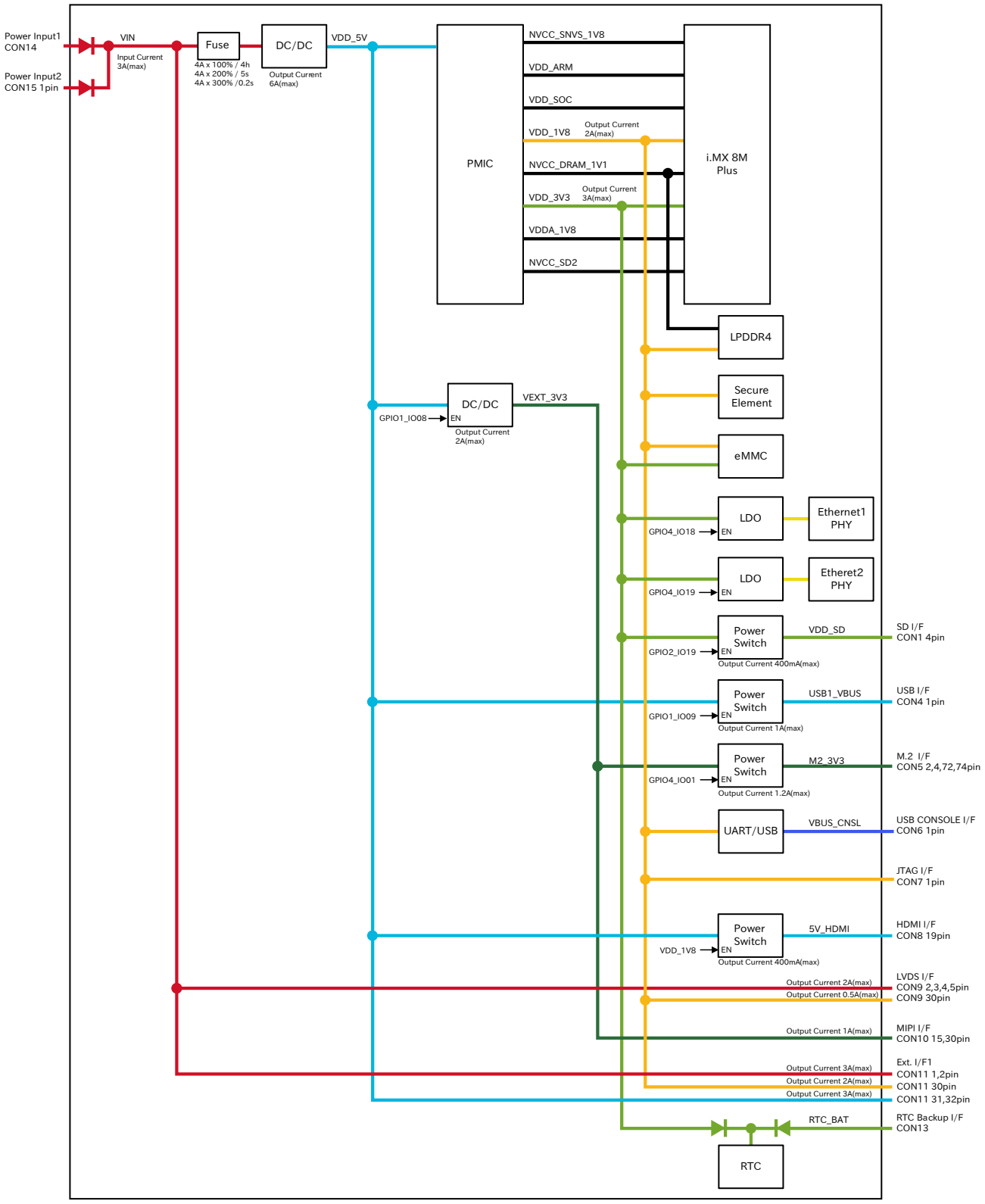


図 14.1 電源回路の構成

入力電圧(VIN)を電源 IC で各電圧に変換し、内部回路および各インターフェースに供給しています。各インターフェースやスイッチング・レギュレータの最大出力電流値を超えないように、外部機器の接続、供給電源の設計を行なってください。

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

14.1.6. 電源シーケンス

電源シーケンスは「図 14.2. 電源シーケンス」のとおりです。

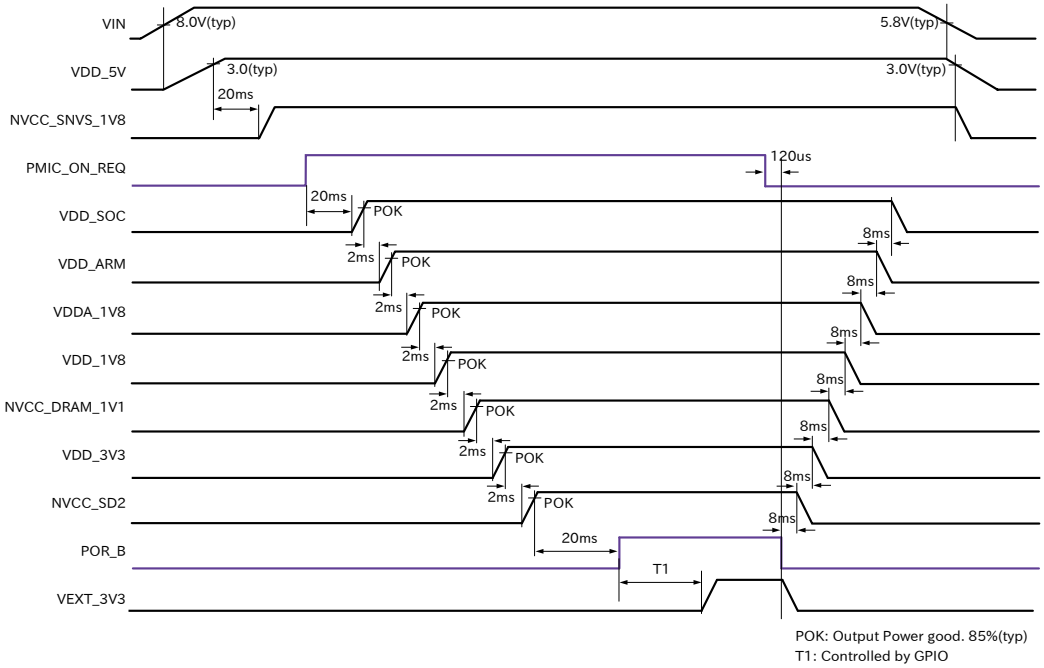


図 14.2 電源シーケンス

14.1.6.1. 電源オン時

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

14.1.6.2. 電源オフ時

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

14.1.7. リセット回路の構成

Armadillo-IoT ゲートウェイ G4 のリセット回路の構成は「図 14.3. リセット回路の構成」のとおりです。

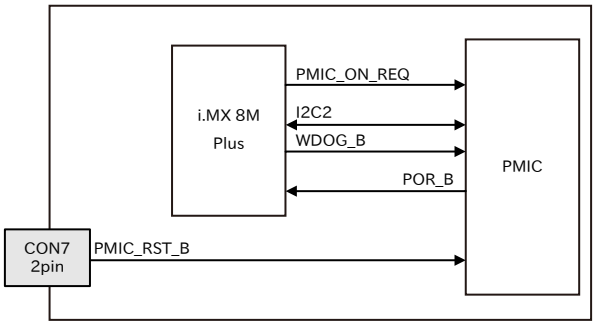


図 14.3 リセット回路の構成

14.1.8. リセットシーケンス

リセットシーケンスは「図 14.4. リセットシーケンス」のとおりです。

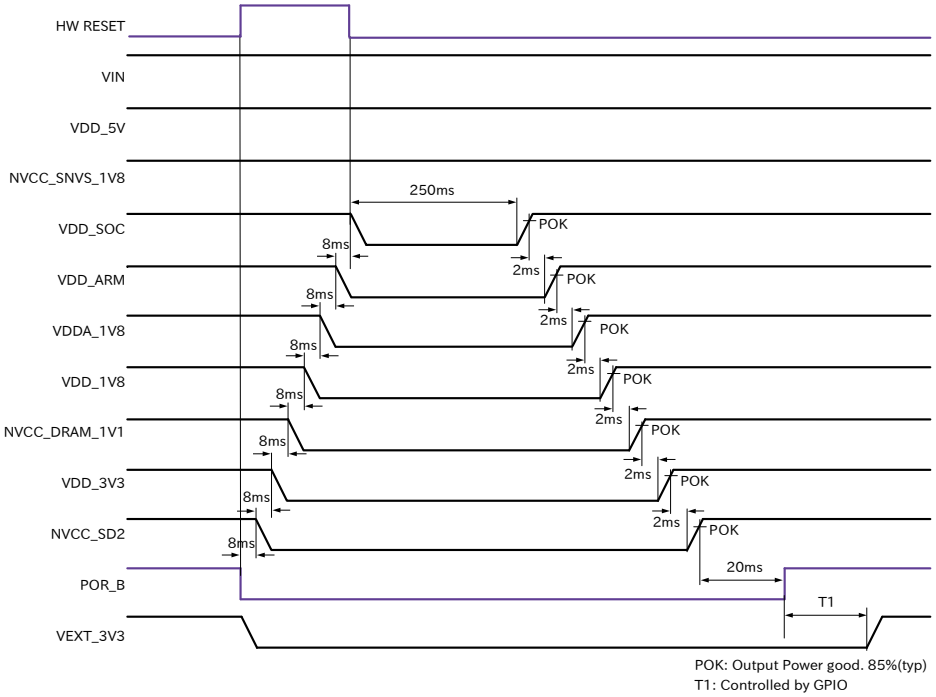


図 14.4 リセットシーケンス

Armadillo-IoT ゲートウェイ G4 のハードウェアリセットには、I2C によるリセット[1]、ウォッチドックタイマーによるリセット、JTAG インターフェースの PMIC_RST_B 信号(CON7 2 ピン)によるリセットの 3 つがあります。

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

[1]reboot コマンド時は、I2C によりリセットされます。

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

14.1.9. 外部からの電源制御

14.1.9.1. ONOFF ピンからの電源制御

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

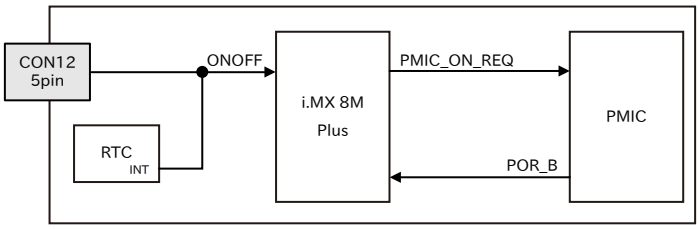


図 14.5 ONOFF 回路の構成

ONOFF 信号を一定時間以上、Low レベルとすることで、i.MX 8M Plus のオン状態、オフ状態を切り替えることができます。オン状態になると、PMIC_ON_REQ 信号がアサートされ、ソフトウェアからの制御で電源切断しているものを除いて、すべての電源が供給されます。オフ状態になると、PMIC_ON_REQ 信号がディアサートされ、VIN、VDD_5V、NVCC_SNVS_1V8 以外の電源が切断されます。オン状態からオフ状態に切り替える場合は 5 秒以上、オフ状態からオン状態に切り替える場合は 500 ミリ秒以上、Low レベルを保持する必要があります。

表 14.6 オン状態、オフ状態を切り替えする際の Low レベル保持時間

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

ONOFF 信号を制御する場合は、オープンドレイン出力等で GND とショートする回路を接続してください。オン状態およびオフ状態は、NVCC_SNVS_1V8 が供給されている限り、保持されます。

オフ状態にして Armadillo-IoT ゲートウェイ G4 の電源(VIN)を切断した場合、コンデンサに蓄えられた電荷が抜けるまではオフ状態であることが保持されます。オフ状態を保持した状態で電源を投入したくない場合は、一定時間以上空けて電源を投入する必要があります。開発セット付属の AC アダプタの場合に必要な時間は以下のとおりです。

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

14.2. インターフェース仕様

Armadillo-IoT ゲートウェイ G4 のインターフェース仕様について説明します。

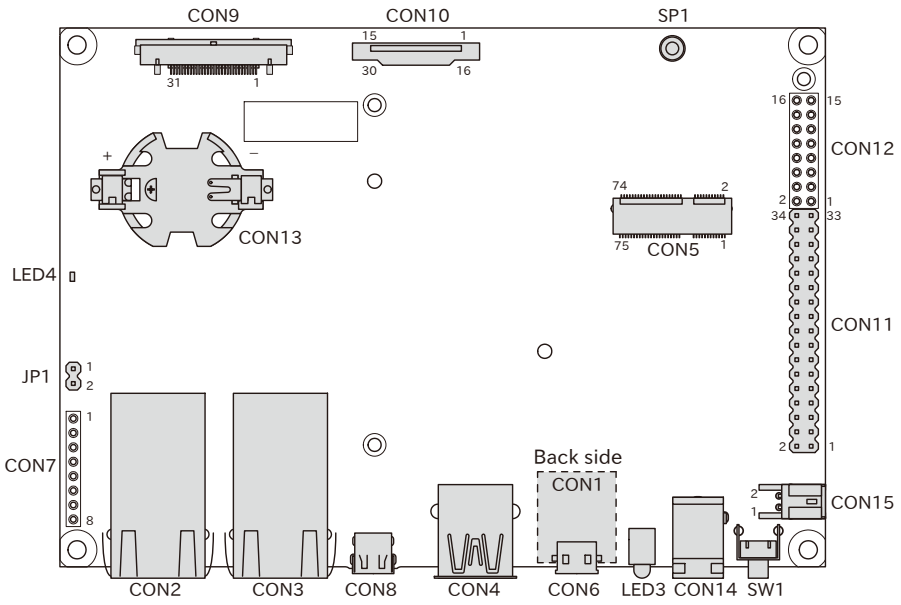


図 14.6 Armadillo-IoT ゲートウェイ G4 のインターフェース

表 14.7 Armadillo-IoT ゲートウェイ G4 インターフェース一覧

部品番号	インターフェース名	型番	メーカー
CON1	SD インターフェース	DM3BT-DSF-PEJS	HIROSE ELECTRIC
CON2	LAN インターフェース 2	56F-1304DYDZ2NL	YUAN DEAN SCIENTIFIC
CON3	LAN インターフェース 1	56F-1304DYDZ2NL	YUAN DEAN SCIENTIFIC
CON4	USB インターフェース	UJ3-AH-4-TH	CUI
CON5	M.2 インターフェース	SM3ZS067U410AER1000	Japan Aviation Electronics Industry
CON6	USB コンソールインターフェース	UB-MC5BR3-SD204-4S-1-TB NMP	J.S.T.Mfg.
CON7	JTAG インターフェース	A2-8PA-2.54DSA(71)	HIROSE ELECTRIC
CON8	HDMI インターフェース	DC3RX19JA2R1700	Japan Aviation Electronics Industry
CON9	LVDS インターフェース	FX15S-31S-0.5SH	HIROSE ELECTRIC
CON10	MIPI-CSI インターフェース	1-1734248-5	TE Connectivity
CON11	拡張インターフェース 1	61303421121	Wurth Elektronik
CON12	拡張インターフェース 2	61301621121	Wurth Elektronik
CON13	RTC バックアップインターフェース	BU2032SM-FH-GTR	Memory Protection Devices
CON14	電源入力インターフェース 1	PJ-102AH	CUI
CON15	電源入力インターフェース 2	S02B-PASK-2(LF)(SN)	J.S.T.Mfg.
JP1	起動デバイス設定ジャンパ	61300211121	Wurth Elektronik
SW1	ユーザースイッチ	SKHHLUA010	ALPS ELECTRIC
LED3	ユーザー LED	L-710A8CB/1GD	Kingbright Electronic
LED4	電源 LED	SML-D12M8WT86	ROHM
SP1	M.2 用スタッド	SM3ZS067U410-NUT1-R1200	Japan Aviation Electronics Industry

「表 14.7. Armadillo-IoT ゲートウェイ G4 インターフェース一覧」には部品の実装、未実装を問わず、搭載可能な代表型番を記載しています。お手元の製品に搭載されている実際の部品情報につきましては、「アットマー

クテクノ Armadillo サイト」 [https://armadillo.atmark-techno.com/] からダウンロードできる納入仕様書および変更履歴表をご確認ください。

14.2.1. CON1 (SD インターフェース)

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

表 14.8 CON1 信号配列

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

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

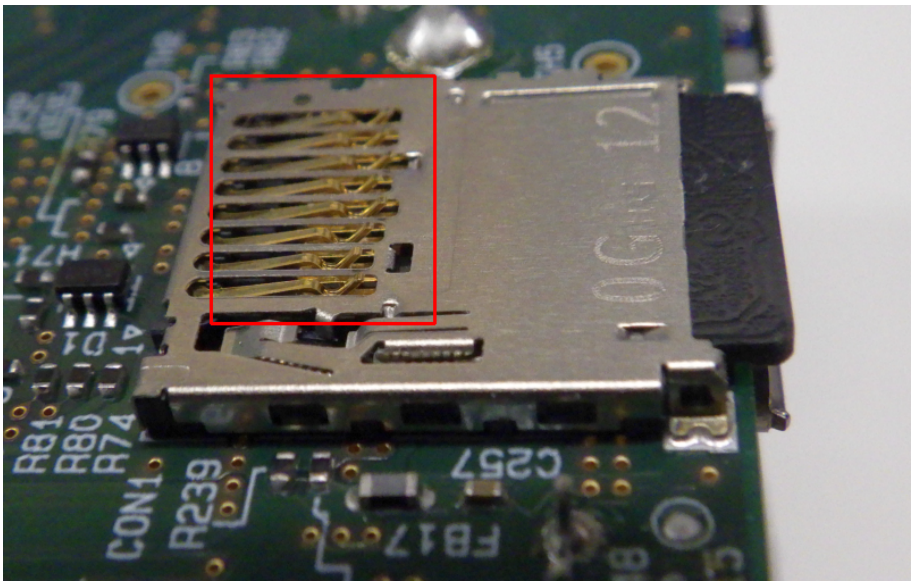


図 14.7 CON1 microSD スロット 取り扱い上の注意事項

14.2.2. CON2 (LAN インターフェース 2)

CON2 は 100BASE-TX/1000BASE-T に対応した LAN インターフェースです。カテゴリ 5e 以上のイーサネットケーブルを接続することができます。AUTO-MDIX 機能を搭載しており、ストレートケーブルまたはクロスケーブルを自動認識して送受信端子を切り替えます。

信号線は Ethenet PHY(KSZ9131RNXI-TR/Microchip Technology)を経由して i.MX 8M Plus の Ethernet Quality Of Service(ENET_QOS)に接続されています。N

CON2 (LAN インターフェース 2)は 10BASE-T に対応していません。

表 14.9 CON2 信号配列 (10BASE-T/100BASE-TX)

ピン番号	ピン名	I/O	説明
1	LAN2_TX+	In/Out	送信データ(+)
2	LAN2_TX-	In/Out	送信データ(-)
3	LAN2_RX+	In/Out	受信データ(+)
4	-	-	-
5	-	-	-
6	LAN2_RX-	In/Out	受信データ(-)
7	-	-	-
8	-	-	-

表 14.10 CON2 信号配列 (1000BASE-T)

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

表 14.11 CON2 LAN LED の動作

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

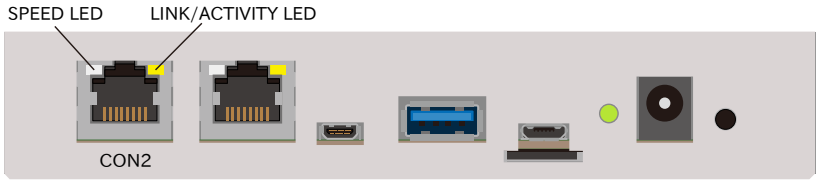


図 14.8 CON2 LAN LED 配置

14.2.3. CON3 (LAN インターフェース 1)

CON3 は 10BASE-T/100BASE-TX/1000BASE-T に対応した LAN インターフェースです。カテゴリ 5e 以上のイーサネットケーブルを接続することができます。AUTO-MDIX 機能を搭載しており、ストレートケーブルまたはクロスケーブルを自動認識して送受信端子を切り替えます。

信号線は Ethenet PHY(KSZ9131RNXI-TR/Microchip Technology)を経由して i.MX 8M Plus の Ethernet MAC(ENET)に接続されています。

表 14.12 CON3 信号配列 (10BASE-T/100BASE-TX)

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

表 14.13 CON3 信号配列 (1000BASE-T)

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

表 14.14 CON3 LAN LED の動作

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

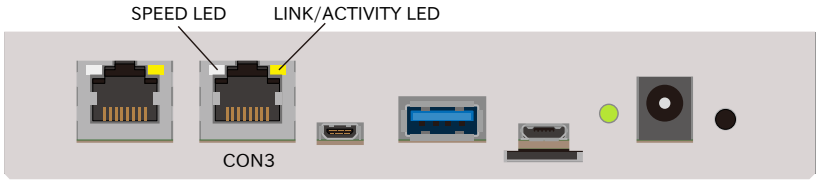


図 14.9 CON3 LAN LED 配置

14.2.4. CON4 (USB インターフェース)

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

USB デバイスに供給される電源(USB1_VBUS)は、i.MX 8M Plus の GPIO1_IO09 ピンで制御しており、High レベル出力で電源が供給され、Low レベル出力で電源が切断されます。

- データ転送モード
  - Super Speed(5Gbps)
  - High Speed(480Mbps)
  - Full Speed(12Mbps)
  - Low Speed(1.5Mbps)

表 14.15 CON4 信号配列

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

14.2.5. CON5 (M.2 インターフェース)

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

M.2 インターフェースに供給される電源(M2_3V3)は、i.MX 8M Plus の SAI1_RXC ピン(GPIO4_IO01)で制御しており、High レベル出力で電源が供給され、Low レベル出力で電源が切断されます。

CON5 の信号配列につきましては、公開していません。

14.2.6. CON6 (USB コンソールインターフェース)

CON6 は USB コンソール用インターフェースです。

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

表 14.16 CON6 信号配列

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

14.2.7. CON7 (JTAG インターフェース)

CON7 は JTAG デバッガを接続することのできる JTAG インターフェースです。

JTAG の詳細につきましては、NXP Semiconductors のホームページからダウンロード可能な『i.MX 8M Plus Applications Processor Reference Manual』をご参照ください。

表 14.17 CON7 信号配列

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

[a]PMIC リセット信号の詳細につきましては、「14.1.8. リセットシーケンス」をご確認ください。

14.2.8. CON8 (HDMI インターフェース)

CON8 は HDMI 出力インターフェースです。

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

表 14.18 CON7 信号配列

ピン番号	ピン名	I/O	説明
1	HDMI_HPD	In	ホットプラグ検出、HEAC(-)、i.MX 8M Plus の EARC_N_HPD ピン、HDMI_HPD ピンに接続
2	HDMI_Utility	In/Out	Utility、HEAC(+)、i.MX 8M Plus の EARC_P_UTIL ピンに接続
3	HDMI_TX2+	Out	TMDS データ 2(+)、i.MX 8M Plus の HDMI_TX2_P ピンに接続
4	HDMI_TX2_Shield	-	TMDS データ 2 シールド
5	HDMI_TX2-	Out	TMDS データ 2(-)、i.MX 8M Plus の HDMI_TX2_N ピンに接続
6	HDMI_TX1+	Out	TMDS データ 1(+)、i.MX 8M Plus の HDMI_TX1_P ピンに接続
7	HDMI_TX1_Shield	-	TMDS データ 1 シールド

ピン番号	ピン名	I/O	説明
8	HDMI_TX1-	Out	TMDS データ 1(-)、i.MX 8M Plus の HDMI_TX1_N ピンに接続
9	HDMI_TX0+	Out	TMDS データ 0(+)、i.MX 8M Plus の HDMI_TX0_P ピンに接続
10	HDMI_TX0_Shield	-	TMDS データ 0 シールド
11	HDMI_TX0-	Out	TMDS データ 0(-)、i.MX 8M Plus の HDMI_TX0_N ピンに接続
12	HDMI_TXC+	Out	TMDS クロック(+)、i.MX 8M Plus の HDMI_TXC_P ピンに接続
13	HDMI_TXC_Shield	-	TMDS クロックシールド
14	HDMI_TXC-	Out	TMDS クロック(-)、i.MX 8M Plus の HDMI_TXC_N ピンに接続
15	HDMI_CEC	In/Out	CEC 信号、i.MX 8M Plus の HDMI_CEC ピンに接続
16	HDMI_GND	Power	電源(GND)
17	HDMI_SCL	In/Out	DDC クロック、i.MX 8M Plus の HDMI_DDC_SCL ピンに接続
18	HDMI_SDA	In/Out	DDC データ、i.MX 8M Plus の HDMI_DDC_SDA ピンに接続
19	5V_HDMI	Power	電源(5V_HDMI)

14.2.9. CON9 (LVDS インターフェース)

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

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

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

表 14.19 CON9 搭載コネクタと対向コネクタ例

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

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

表 14.20 CON9 信号配列

ピン番号	ピン名	I/O	説明	電圧グループ
1	GND	Power	電源(GND)	-
2	VIN	Power	電源(VIN)	-
3	VIN	Power	電源(VIN)	-
4	VIN	Power	電源(VIN)	-
5	VIN	Power	電源(VIN)	-
6	GND	Power	電源(GND)	-

ピン番号	ピン名	I/O	説明	電圧グループ
7	I2C4_SCL	In/Out	I2C クロック、i.MX 8M Plus の I2C4_SCL ピン、CON11 5 ピンに接続、 基板上で 4.7k プルアップ (VDD_1V8)	VDD_1V8
8	I2C4_SDA	In/Out	I2C データ、i.MX 8M Plus の I2C4_SDA ピン、CON11 7 ピンに接続、 基板上で 4.7k プルアップ (VDD_1V8)	VDD_1V8
9	GND	Power	電源(GND)	-
10	LVDS0_TX0N	Out	LVDS データ 0(-)、i.MX 8M Plus の LVDS0_D0_N ピンに接続	-
11	LVDS0_TX0P	Out	LVDS データ 0(+)、i.MX 8M Plus の LVDS0_D0_P ピンに接続	-
12	GND	Power	電源(GND)	-
13	LVDS0_TX1N	Out	LVDS データ 1(-)、i.MX 8M Plus の LVDS0_D1_N ピンに接続	-
14	LVDS0_TX1P	Out	LVDS データ 1(+)、i.MX 8M Plus の LVDS0_D1_P ピンに接続	-
15	GND	Power	電源(GND)	-
16	LVDS0_CLKN	Out	LVDS クロック(-)、i.MX 8M Plus の LVDS0_CLK_N ピンに接続	-
17	LVDS0_CLKP	Out	LVDS クロック(+)、i.MX 8M Plus の LVDS0_CLK_P ピンに接続	-
18	GND	Power	電源(GND)	-
19	LVDS0_TX2N	Out	LVDS データ 2(-)、i.MX 8M Plus の LVDS0_D2_N ピンに接続	-
20	LVDS0_TX2P	Out	LVDS データ 2(+)、i.MX 8M Plus の LVDS0_D2_P ピンに接続	-
21	GND	Power	電源(GND)	-
22	LVDS0_TX3N	Out	LVDS データ 3(-)、i.MX 8M Plus の LVDS0_D3_N ピンに接続	-
23	LVDS0_TX3P	Out	LVDS データ 3(+)、i.MX 8M Plus の LVDS0_D3_P ピンに接続	-
24	GND	Power	電源(GND)	-
25	GPIO3_IO00	In/Out	拡張入出力、i.MX 8M Plus の NAND_ALE ピンに接続	VDD_1V8
26	GPIO3_IO01	In/Out	拡張入出力、i.MX 8M Plus の NAND_CE0_B ピンに接続	VDD_1V8
27	GPIO5_IO03	In/Out	拡張入出力、i.MX 8M Plus の SPDIF_TX ピンに接続	VDD_1V8
28	GPIO5_IO04	In/Out	拡張入出力、i.MX 8M Plus の SPDIF_RX ピンに接続	VDD_1V8
29	GND	Power	電源(GND)	-
30	VDD_1V8	Power	電源(VDD_1V8)	-
31	GND	Power	電源(GND)	-

14.2.10. CON10 (MIPI-CSI インターフェース)

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

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

表 14.21 CON10 搭載コネクタとフレキシブルフラットケーブル例

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

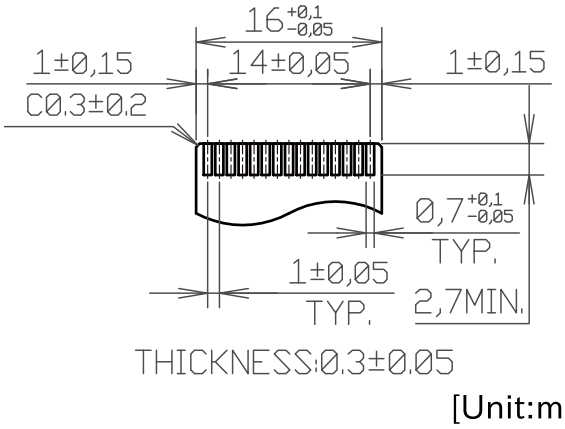


図 14.10 CON10 接続可能なフレキシブルフラットケーブルの形状

表 14.22 CON10 信号配列

ピン番号	ピン名	I/O	説明	電圧グループ
1	GND	Power	電源(GND)	-
2	CSI1_DN_0	In	MIPI データ 0(-)、i.MX 8M Plus の MIPI_CSI1_D0_N ピンに接続、 17 ピンと共通	-
3	CSI1_DP_0	In	MIPI データ 0(+)、i.MX 8M Plus の MIPI_CSI1_D0_P ピンに接続、 18 ピンと共通	-
4	GND	Power	電源(GND)	-
5	CSI1_DN_1	In	MIPI データ 1(-)、i.MX 8M Plus の MIPI_CSI1_D1_N ピンに接続、 20 ピンと共通	-
6	CSI1_DP_1	In	MIPI データ 1(+)、i.MX 8M Plus の MIPI_CSI1_D1_P ピンに接続、 21 ピンと共通	-
7	GND	Power	電源(GND)	-
8	CSI1_CK_N	In	MIPI クロック(-)、i.MX 8M Plus の MIPI_CSI1_CLK_N ピンに接続、 23 ピンと共通	-
9	CSI1_CK_P	In	MIPI クロック(+)、i.MX 8M Plus の MIPI_CSI1_CLK_P ピンに接続、 24 ピンと共通	-
10	GND	Power	電源(GND)	-
11	CSI1_GPIO0_3V3	In/Out	拡張入出力、レベル変換 IC を経由して i.MX 8M Plus の NAND_DATA00 ピンに接続、 26 ピンと共通	VEXT_3V3
12	CSI1_GPIO1_3V3	In/Out	拡張入出力、レベル変換 IC を経由して i.MX 8M Plus の NAND_DATA01 ピンに接続、 27 ピンと共通	VEXT_3V3
13	I2C2_SCL_3V3	Out	I2C クロック、レベル変換 IC を経由して i.MX 8M Plus の I2C2_SCL ピンに接続、基板上で 4.7k プルアップ(VDD_3V3)、 28 ピンと共通	VEXT_3V3
14	I2C2_SDA_3V3	In/Out	I2C データ、レベル変換 IC を経由して i.MX 8M Plus の I2C2_SDA ピンに接続、基板上で 4.7k プルアップ(VDD_3V3)、 29 ピンと共通	VEXT_3V3
15	VEXT_3V3	Power	電源(VEXT_3V3)	-
16	GND	Power	電源(GND)	-
17	CSI1_DN_0	In	MIPI データ 0(-)、i.MX 8M Plus の MIPI_CSI1_D0_N ピンに接続、 2 ピンと共通	-
18	CSI1_DP_0	In	MIPI データ 0(+)、i.MX 8M Plus の MIPI_CSI1_D0_P ピンに接続、 3 ピンと共通	-
19	GND	Power	電源(GND)	-
20	CSI1_DN_1	In	MIPI データ 1(-)、i.MX 8M Plus の MIPI_CSI1_D1_N ピンに接続、 5 ピンと共通	-

ピン番号	ピン名	I/O	説明	電圧グループ
21	CSI1_DP_1	In	MIPI データ 1(+)、i.MX 8M Plus の MIPI_CSI1_D1_P ピンに接続、 6 ピンと共通	-
22	GND	Power	電源(GND)	-
23	CSI1_CK_N	In	MIPI クロック(-)、i.MX 8M Plus の MIPI_CSI1_CLK_N ピンに接続、 8 ピンと共通	-
24	CSI1_CK_P	In	MIPI クロック(+)、i.MX 8M Plus の MIPI_CSI1_CLK_P ピンに接続、 9 ピンと共通	-
25	GND	Power	電源(GND)	-
26	CSI1_GPIO0_3V3	In/Out	拡張入出力、レベル変換 IC を経由して i.MX 8M Plus の NAND_DATA00 ピンに接続、基板上で 4.7k プルアップ(VEXT_3V3)、 11 ピンと共通	VEXT_3V3
27	CSI1_GPIO1_3V3	In/Out	拡張入出力、レベル変換 IC を経由して i.MX 8M Plus の NAND_DATA01 ピンに接続、基板上で 4.7k プルアップ(VEXT_3V3)、 12 ピンと共通	VEXT_3V3
28	I2C2_SCL_3V3	Out	I2C クロック、レベル変換 IC を経由して i.MX 8M Plus の I2C2_SCL ピンに接続、基板上で 4.7k プルアップ(VDD_3V3)、 13 ピンと共通	VDD_3V3
29	I2C2_SDA_3V3	In/Out	I2C データ、レベル変換 IC を経由して i.MX 8M Plus の I2C2_SDA ピンに接続、基板上で 4.7k プルアップ(VDD_3V3)、 14 ピンと共通	VDD_3V3
30	VEXT_3V3	Power	電源(GND)	-

14.2.11. CON11、CON12 (拡張インターフェース)

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

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

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

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

表 14.23 CON11、CON12 搭載コネクタと対向コネクタ例

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

[a] xx にはピン数が入ります。

表 14.24 CON11 信号配列

ピン番号	ピン名	I/O	説明	電圧グループ
1	VIN	Power	電源入出力(VIN)、CON14、CON15 と共通	-
2	VIN	Power	電源入出力(VIN)、CON14、CON15 と共通	-
3	GND	Power	電源(GND)	-
4	GND	Power	電源(GND)	-
5	I2C4_SCL	In/Out	拡張入出力、i.MX 8M Plus の I2C4_SCL ピン、CON9 7 ピンに接続 基板上で 4.7k プルアップ	VDD_1V8
6	ECSPI1_MISO	In/Out	拡張入出力、i.MX 8M Plus の ECSPI1_MISO ピンに接続	VDD_1V8
7	I2C4_SDA	In/Out	拡張入出力、i.MX 8M Plus の I2C4_SDA ピン、CON9 8 ピンに接続 基板上で 4.7k プルアップ	VDD_1V8
8	ECSPI1_MOSI	In/Out	拡張入出力、i.MX 8M Plus の ECSPI1_MOSI ピンに接続	VDD_1V8
9	ECSPI2_MISO	In/Out	拡張入出力、i.MX 8M Plus の ECSPI2_MISO ピンに接続	VDD_1V8
10	ECSPI1_SCLK	In/Out	拡張入出力、i.MX 8M Plus の ECSPI1_SCLK ピンに接続	VDD_1V8
11	ECSPI2_MOSI	In/Out	拡張入出力、i.MX 8M Plus の ECSPI2_MOSI ピンに接続	VDD_1V8
12	ECSPI1_SS0	In/Out	拡張入出力、i.MX 8M Plus の ECSPI1_SS0 ピンに接続	VDD_1V8
13	ECSPI2_SCLK	In/Out	拡張入出力、i.MX 8M Plus の ECSPI2_SCLK ピンに接続	VDD_1V8
14	SAI3_TXFS	In/Out	拡張入出力、i.MX 8M Plus の SAI3_TXFS ピンに接続	VDD_1V8
15	ECSPI2_SS0	In/Out	拡張入出力、i.MX 8M Plus の ECSPI2_SS0 ピンに接続	VDD_1V8
16	SAI3_TXC	In/Out	拡張入出力、i.MX 8M Plus の SAI3_TXC ピンに接続	VDD_1V8
17	SAI5_RXC	In/Out	拡張入出力、i.MX 8M Plus の SAI5_RXC ピンに接続	VDD_1V8
18	SAI3_TXD	In/Out	拡張入出力、i.MX 8M Plus の SAI3_TXD ピンに接続	VDD_1V8
19	SAI5_RXD0	In/Out	拡張入出力、i.MX 8M Plus の SAI5_RXD0 ピンに接続	VDD_1V8
20	SAI3_RXD	In/Out	拡張入出力、i.MX 8M Plus の SAI3_RXD ピンに接続	VDD_1V8
21	SAI5_RXD1	In/Out	拡張入出力、i.MX 8M Plus の SAI5_RXD1 ピンに接続	VDD_1V8
22	SAI3_MCLK	In/Out	拡張入出力、i.MX 8M Plus の SAI3_MCLK ピンに接続	VDD_1V8
23	SAI5_RXD2	In/Out	拡張入出力、i.MX 8M Plus の SAI5_RXD2 ピンに接続	VDD_1V8
24	GPIO1_IO15	In/Out	拡張入出力、i.MX 8M Plus の GPIO1_IO15 ピンに接続	VDD_1V8
25	SAI5_RXD3	In/Out	拡張入出力、i.MX 8M Plus の SAI5_RXD3 ピンに接続	VDD_1V8
26	USBDM_DN2	In/Out	USB 2.0 データ(-)、USB HUB 経由で i.MX 8M Plus の USB2 に接続	-
27	SAI5_MCLK	In/Out	拡張入出力、i.MX 8M Plus の SAI5_MCLK ピンに接続	VDD_1V8
28	USBDP_DN2	In/Out	USB 2.0 データ(+)、USB HUB 経由で i.MX 8M Plus の USB2 に接続	-
29	SAI5_RXFS	In/Out	拡張入出力、i.MX 8M Plus の SAI5_RXFS ピンに接続	VDD_1V8
30	VDD_1V8	Power	電源出力(VDD_1V8)	-
31	VDD_5V	Power	電源出力(VDD_5V)	-

ピン番号	ピン名	I/O	説明	電圧グループ
32	VDD_5V	Power	電源出力(VDD_5V)	-
33	GND	Power	電源(GND)	-
34	GND	Power	電源(GND)	-

表 14.25 CON12 信号配列

ピン番号	ピン名	I/O	説明	電圧グループ
1	PWR_OFF	In/Out	Armadillo Base OS で使用、ユーザーによる変更可能、 i.MX 8M Plus の GPIO1_IO01 ピンに接続	VDD_1V8
2	GPIO2_IO11	In/Out	拡張入出力、i.MX 8M Plus の GPIO2_IO11 ピンに接続	VDD_1V8
3	REBOOT	In/Out	Armadillo Base OS で使用、ユーザーによる変更可能、 i.MX 8M Plus の GPIO1_IO05 ピンに接続	VDD_1V8
4	GPIO4_IO27	In/Out	拡張入出力、i.MX 8M Plus の GPIO4_IO27 ピンに接続	VDD_1V8
5	ONOFF	In	ONOFF 信号[a]、i.MX 8M Plus の ONOFF ピンに接続、基板上で 100k プルアップ(NVCC_SNVS_1V8)	NVCC_SNVS_1V8
6	GPIO4_IO28	In/Out	拡張入出力、i.MX 8M Plus の SAI3_RXFS ピンに接続	VDD_1V8
7	FW_UPDATE_IND	In/Out	Armadillo Base OS で使用、ユーザーによる変更可能、 i.MX 8M Plus の GPIO1_IO00 ピンに接続	VDD_1V8
8	GPIO4_IO29	In/Out	拡張入出力、i.MX 8M Plus の GPIO4_IO29 ピンに接続	VDD_1V8
9	STDWN_IND	In/Out	Armadillo Base OS で使用、ユーザーによる変更可能、 i.MX 8M Plus の GPIO1_IO07 ピンに接続	VDD_1V8
10	GPIO3_IO08	In/Out	拡張入出力、i.MX 8M Plus の NAND_DATA02 ピンに接続	VDD_1V8
11	PWR_IND	In/Out	Armadillo Base OS で使用、ユーザーによる変更可能、 i.MX 8M Plus の GPIO1_IO06 ピンに接続	VDD_1V8
12	GPIO3_IO09	In/Out	拡張入出力、i.MX 8M Plus の NAND_DATA03 ピンに接続	VDD_1V8
13	GPIO2_IO08	In/Out	拡張入出力、i.MX 8M Plus の SD1_DATA6 ピンに接続	VDD_1V8
14	GPIO3_IO14	In/Out	拡張入出力、i.MX 8M Plus の NAND_DQS ピンに接続	VDD_1V8
15	GND	Power	電源(GND)	-
16	GND	Power	電源(GND)	-

[a]ONOFF 信号の詳細につきましては、「14.1.9.1. ONOFF ピンからの電源制御」をご確認ください。

拡張できる機能の詳細につきましては、「アットマークテクノ Armadillo サイト」 [https://armadillo.atmark-techno.com/]からダウンロードできる『Armadillo-IoT ゲートウェイ G4 マルチプレクス表』をご参照ください。

14.2.11.1. USB

USB 2.0 Host を 1 ポート拡張することが可能です。信号線は USB HUB 経由で USB コントローラ(USB2)に接続されています。

- 転送速度
  - High Speed(480Mbps)
  - Full Speed(12Mbps)
  - Low Speed(1.5Mbps)

14.2.11.2. GPIO

GPIO を最大 34 ポート拡張することが可能です。

- 信号レベル : VDD_1V8

14.2.11.3. SPI

SPI を最大 2 ポート拡張することが可能です。信号線は i.MX 8M Plus の ESPI(ECSPI1、ECSPI2)に接続されています。

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

14.2.11.4. UART

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

- 最大データ転送レート: 4Mbps
- 信号レベル: VDD_1V8

14.2.11.5. PDM MIC

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

- 信号レベル: VDD_1V8

14.2.11.6. I2S(SAI)

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

- 信号レベル: VDD_1V8

14.2.11.7. CAN

CAN を最大 2 ポート拡張することが可能です。信号線は i.MX 8M Plus の FLEXCAN(FLEXCAN1、FLEXCAN2)に接続されています。

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

14.2.11.8. I2C

I2C を最大 3 ポート拡張することが可能です。信号線は i.MX6ULL の I2C コントローラ(I2C4、I2C5、I2C6)に接続されています。

- 最大データ転送レート: 320kbps
- 信号レベル: VDD_1V8

14.2.11.9. PWM

PWM を最大 4 ポート拡張することが可能です。

- 最大周波数: 66MHz
- 信号レベル: VDD_1V8

14.2.12. CON13(RTC バックアップインターフェース)

CON13 はリアルタイムクロックのバックアップ用インターフェースです。長時間電源が切断されても時刻データを保持させたい場合にご使用ください。

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

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

表 14.26 CON13 信号配列

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

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

14.2.13. CON14、CON15(電源入力インターフェース)

CON14、CON15 は電源入力用のインターフェースです。

CON14 には DC ジャックが実装されており、「図 14.11. AC アダプタの極性マーク」と同じ極性マークのある AC アダプタが使用できます。対応プラグは内径 2.1mm、外形 5.5mm のものとなります。

図 14.11 AC アダプタの極性マーク

CON15 には 2mm ピッチのライトアングルコネクタを実装しています。

表 14.27 CON15 搭載コネクタと対向コネクタ例

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

表 14.28 CON15 信号配列

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

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

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

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

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

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

14.2.14. JP1(起動デバイス設定ジャンパ)

JP1 は起動デバイス設定ジャンパです。JP1 の状態で、起動デバイスを設定することができます。

表 14.29 ジャンパの状態と起動デバイス

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

表 14.30 JP1 信号配列

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

14.2.15. SW1(ユーザースイッチ)

SW1 は、ユーザー側で自由に利用できる押しボタンスイッチです。

表 14.31 SW1 信号配列

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

14.2.16. LED3(ユーザー LED)

LED3 は、ユーザー側で自由に利用できる LED です。

表 14.32 LED3 の状態

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

14.2.17. LED4(電源 LED)

LED4 は、Armadillo-IoT ゲートウェイ G4 の電源確認用の LED です。

表 14.33 LED4 の状態

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

14.3. 形状図

14.3.1. 基板形状図

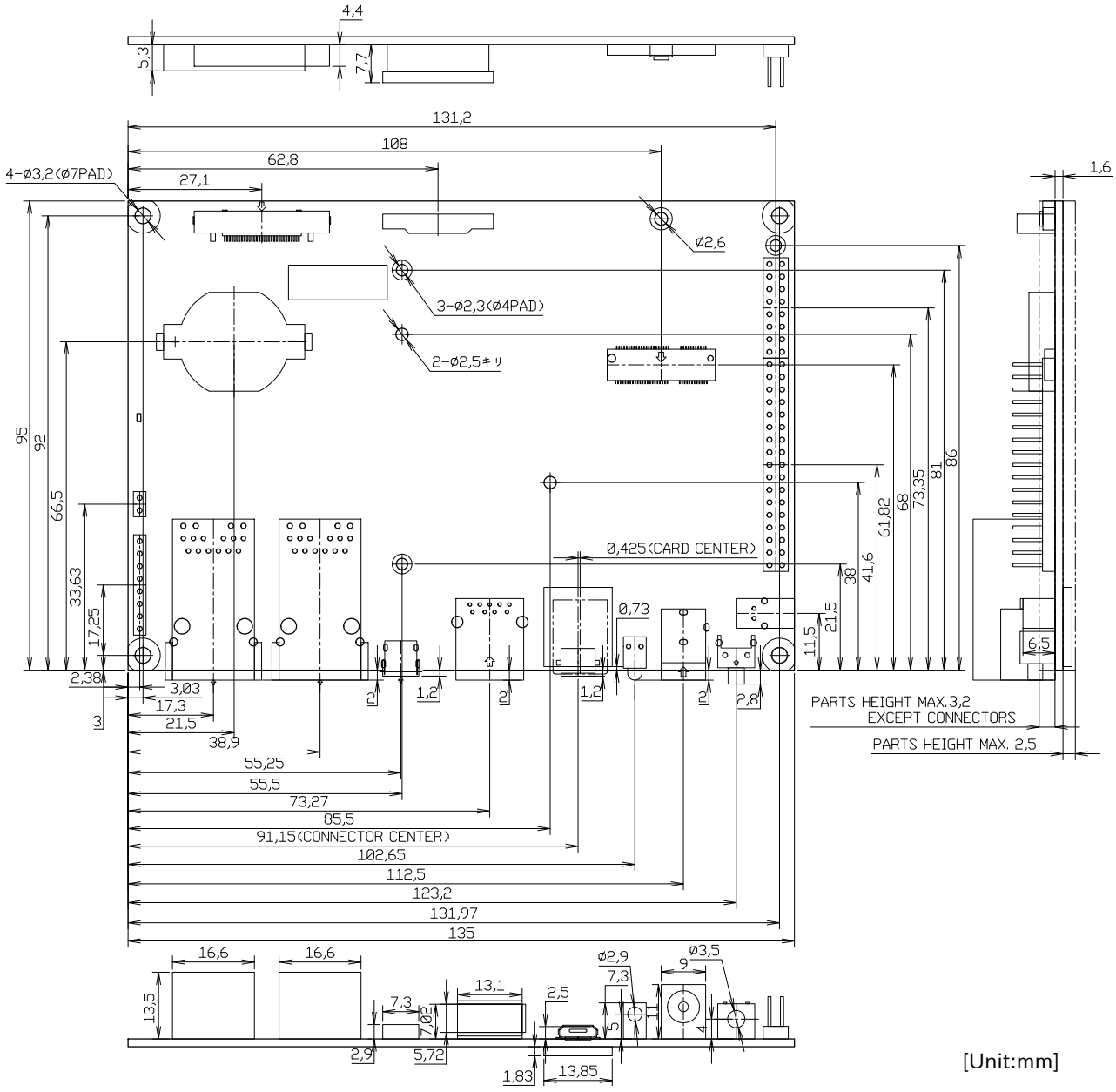


図 14.12 基板形状図

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

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

14.4. 設計情報

本章では、Armadillo-IoT ゲートウェイ G4 の機能拡張や信頼性向上のための設計情報について説明します。

14.4.1. 信頼性試験データについて

Armadillo-IoT ゲートウェイ G4 の各種信頼性試験データを、「アットマークテクノ Armadillo サイト」から「購入者向けの限定公開データ」としてダウンロード可能ですのでご確認ください。

14.4.2. 放射ノイズ

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

- シールド付のケーブルを使用する
- ケーブルは最短で接続する
- 未使用の拡張ピンは Low レベル出力とする

14.4.3. ESD/雷サージ

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

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

Armadillo-IoT ゲートウェイ G4 に接続されたケーブルが屋外に露出するような設置環境では、ケーブルに侵入した雷サージ等のストレスによりインターフェース回路が破壊される場合があります。ストレスへの耐性を向上させるには、以下の対策が効果的です。

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

14.4.4. 放熱

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

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

- 基板単体
- 周囲温度: 約 65℃

・microSD/HDMI/USB3.0/LANx2 動作

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

・基板をケースに収納(放熱シートあり)

・microSD/HDMI/USB3.0/LANx2 動作

オプションケース(金属製)は、SoC の熱をケースに伝導させて放熱する構造で設計しております。同様の構造でのケース設計をご検討の場合は、「14.5.1. オプションケース(金属製)」をご確認ください。

SoC 近辺にヒートシンク固定用の穴(φ2.5mm × 2)を準備していますので、ヒートシンクからの放熱も可能です。寸法につきましては、「14.3. 形状図」をご確認ください。

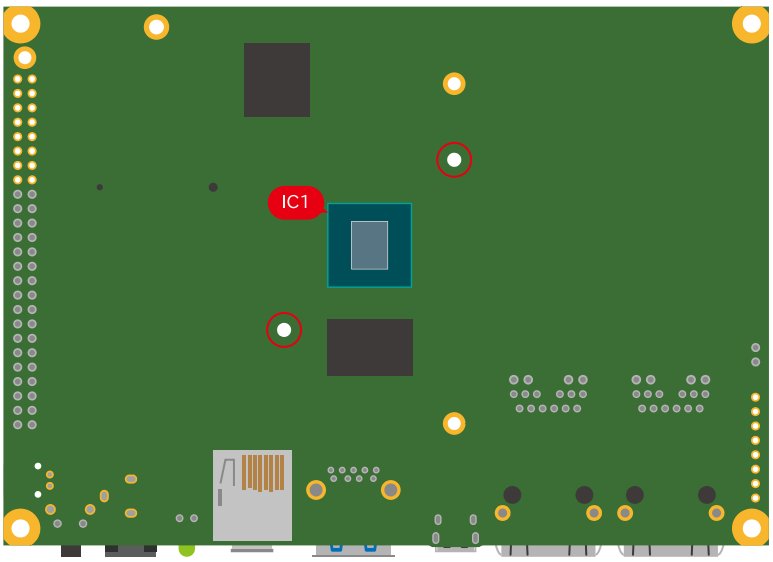


図 14.13 Armadillo-IoT ゲートウェイ G4 の IC1 とヒートシンク固定穴の位置

Armadillo-IoT ゲートウェイ G4 では、温度センサーで CPU(Arm Cortex-A53)周辺温度、SoC(ANAMIX 内部)温度を測定することが可能です。温度センサーの詳細につきましては、「12.15. 温度センサー」をご確認ください。

14.4.5. 拡張ボードの設計

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

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

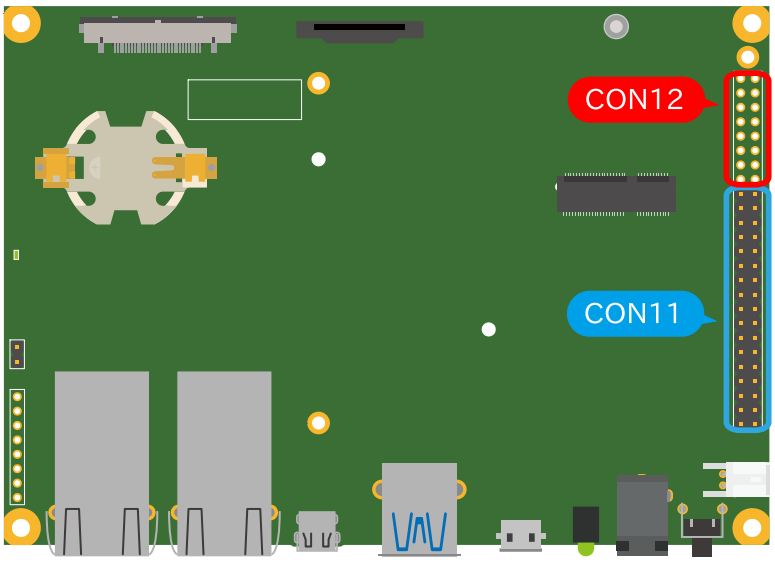


図 14.14 Armadillo-IoT ゲートウェイ G4 の拡張インターフェース

14.4.5.1. ピンアサイン

Armadillo-IoT ゲートウェイ G4 では、「表 3.2. 仕様」の拡張インターフェースの欄にあるとおりの機能が拡張できます。ただし、ここに記載の拡張数は、優先的に機能を割り当てた場合の最大数ですので、必要な機能がすべて実現できるかは、『Armadillo-IoT ゲートウェイ G4 マルチプレクス表』で検討する必要があります。

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

各機能の詳細な仕様が必要な場合は、NXP Semiconductors のホームページからダウンロード可能な、『i.MX 8M Plus Applications Processor Reference Manual』、『i.MX 8M Plus Applications Processor Datasheet for Industrial Products』をご確認ください。Armadillo-IoT ゲートウェイ G4 固有の情報を除いて、回路設計に必要な情報はこれらのマニュアルに、すべて記載されています。検索しやすいように、マルチプレクス表や「14.2.11. CON11、CON12 (拡張インターフェース)」に i.MX 8M Plus のピン名やコントローラー名を記載しておりますので、是非ご活用ください。

Armadillo-IoT ゲートウェイ G4 マルチプレクス表は「アットマークテクノ Armadillo サイト」 [https://armadillo.atmark-techno.com/]からダウンロードしてください。

14.4.5.2. 基板形状

Armadillo-IoT ゲートウェイ G4 の拡張ボードを設計する際の推奨形状は「図 14.15. Armadillo-IoT ゲートウェイ G4 の拡張ボード例」のとおりです。拡張ボード側にピンソケットを実装して Armadillo-IoT ゲートウェイ G4 と接続します。

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

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

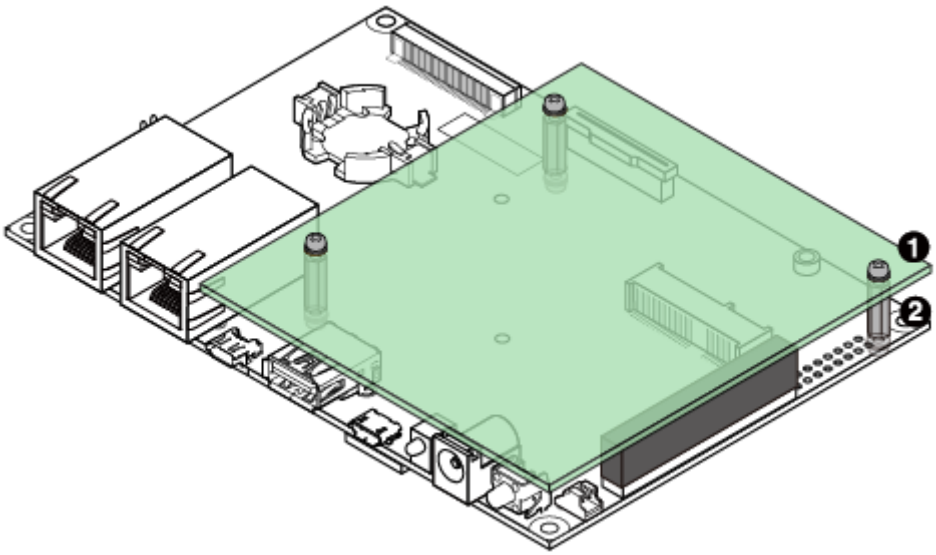


図 14.15 Armadillo-IoT ゲートウェイ G4 の拡張ボード例

❶ なべ小ねじ、ワッシャ、スプリングワッシャ付(M2、L=6mm) × 6

❷ 金属スペーサ(M2、L=11mm) × 3

基板の詳細寸法につきましては、「14.3.1. 基板形状図」をご確認ください。

14.4.6. 回路設計

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

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

14.4.6.1. スイッチ、LED、リレー

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

拡張インターフェースには、i.MX 8M Plus の信号線が直接接続されています。静電気等による内部回路の故障を防ぐため、スイッチと i.MX 8M Plus の間に、電流制限抵抗等を接続することをおすすめします。

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

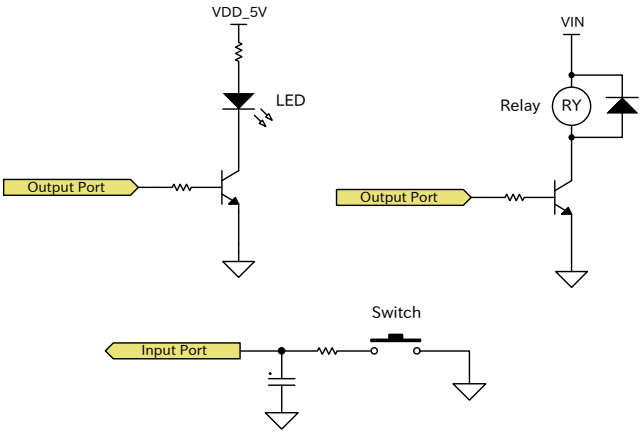


図 14.16 スイッチ、LED、リレー接続例

14.4.6.2. 電源

拡張インターフェース(CON11)から拡張ボード用に、12V 電圧(VIN)、5V 電源(VDD_5V)、1.8V 電源(VDD_1V8)を出力しています。その他の電源が必要な場合は、別途外部から入力するか、DC/DC コンバータ、LDO 等で生成してください。電源シーケンス、出力電流につきましては、「14.1.5. 電源回路の構成」をご確認ください。

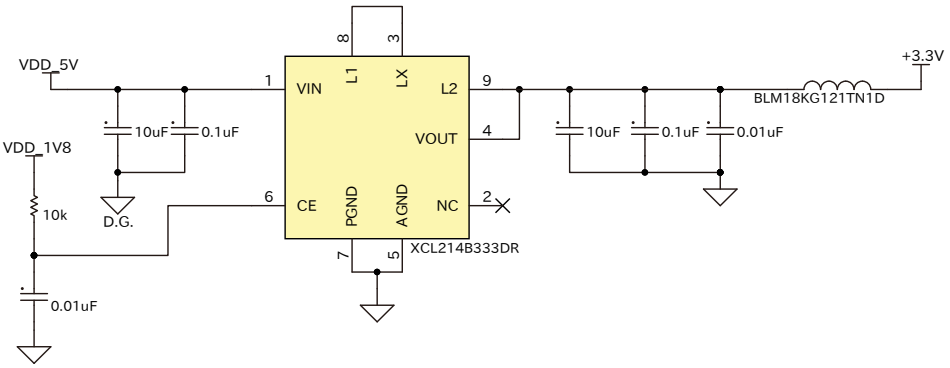


図 14.17 DC/DC コンバータ回路(VDD_5V 入力、3.3V 1.5A 出力)例

「図 14.1. 電源回路の構成」のインターフェース名(LVDS I/F 等)の左横にはコネクタもしくはノイズフィルタの定格電流値を最大値として記載しています。また、パワースイッチの下には、パワースイッチの制限電流値を最大値として記載しています。スイッチングレギュレータの供給能力を超えてしまうため、インターフェースすべての最大値まで電流供給することはできません。それぞれのインターフェースへの推奨の電流供給値は以下のとおりです。

表 14.34 各インターフェースへの電流供給例

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

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

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

14.4.6.3. レベル変換

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

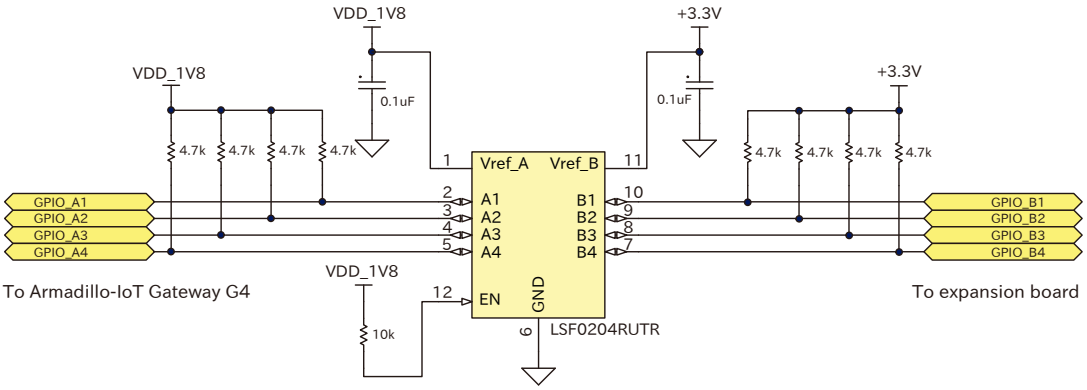


図 14.18 1.8V ←→ 3.3V 双方向レベル変換回路の例

上記レベル変換 IC は 1.8V ←→ 5V でも使用可能です。

14.5. オプション品

本章では、Armadillo-IoT ゲートウェイ G4 のオプション品について説明します。

表 14.35 Armadillo-IoT ゲートウェイ G4 関連のオプション品

名称	型番	備考
オプションケース(金属製)	-	開発セットに付属
AC アダプタ（12V/3.0A φ2.1mm）標準品	OP-AC12V6-00	開発セットに付属

14.5.1. オプションケース(金属製)

14.5.1.1. 概要

Armadillo-IoT ゲートウェイ G4 用のアルミ製ケースです。基板を収めた状態で、DC ジャック、LAN ×2、USB、HDMI、USB コンソール、スイッチ、LED にアクセスすることが可能となっています。

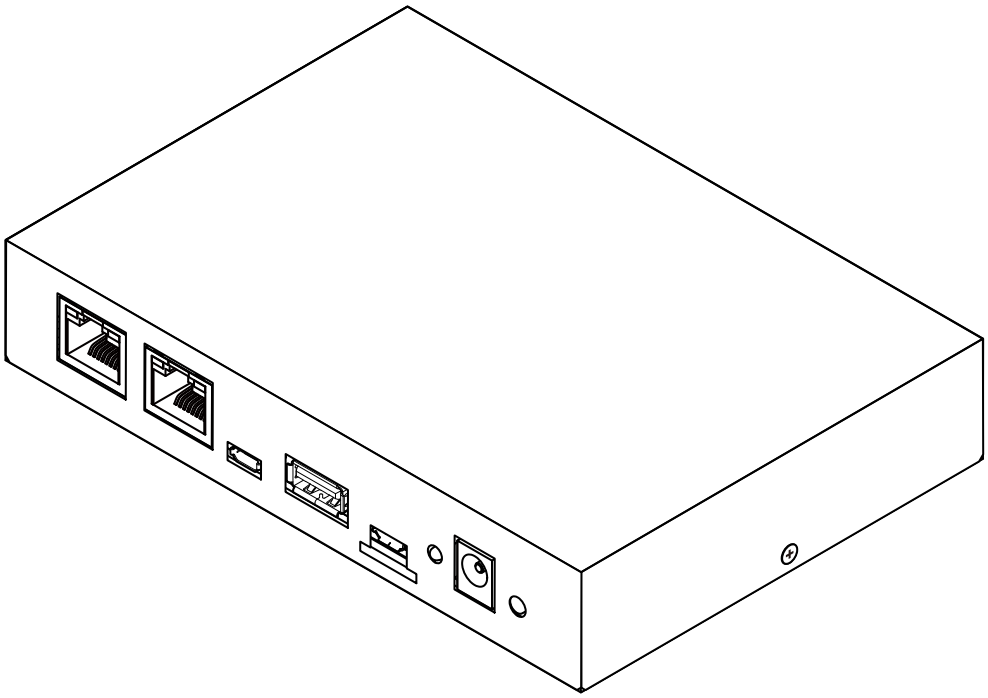


図 14.19 オプションケース(金属製)

コネクタ開口部等に存在する継ぎ目状の加工痕は正常な状態ですのでご了承ください。

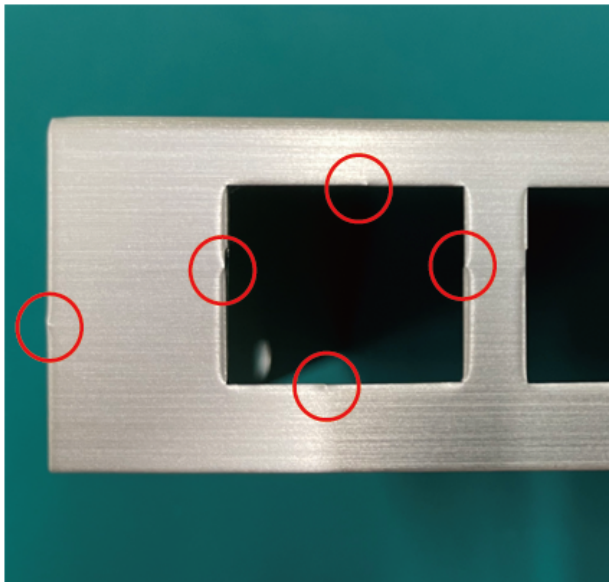


図 14.20 オプションケース(金属製)の加工痕例

14.5.1.2. 組み立て

オプションケース(金属製)は、SoC の熱をケースに伝導させて放熱する構造で設計しています。基板裏の IC1 に放熱シートを貼り付け、ケース(下)と放熱シートを接触させた状態で基板をねじ止めします。ねじの締め付けトルクは 31.5cN·m です。

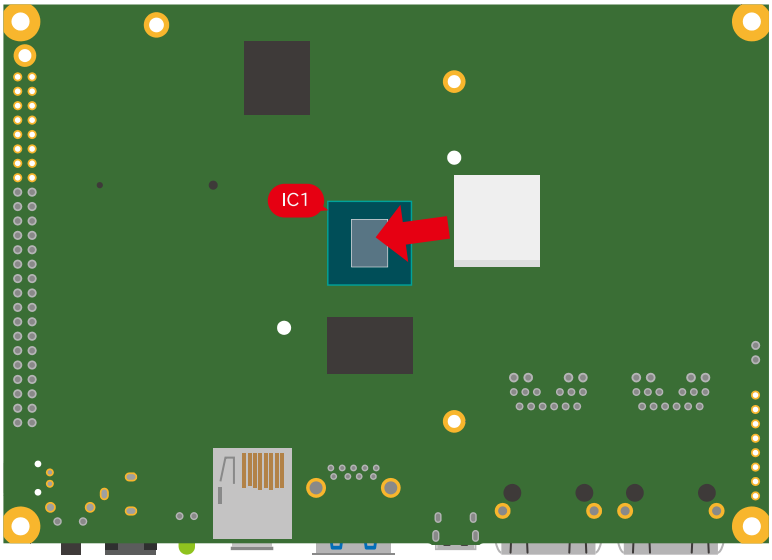


図 14.21 オプションケース(金属製) 放熱シート貼付

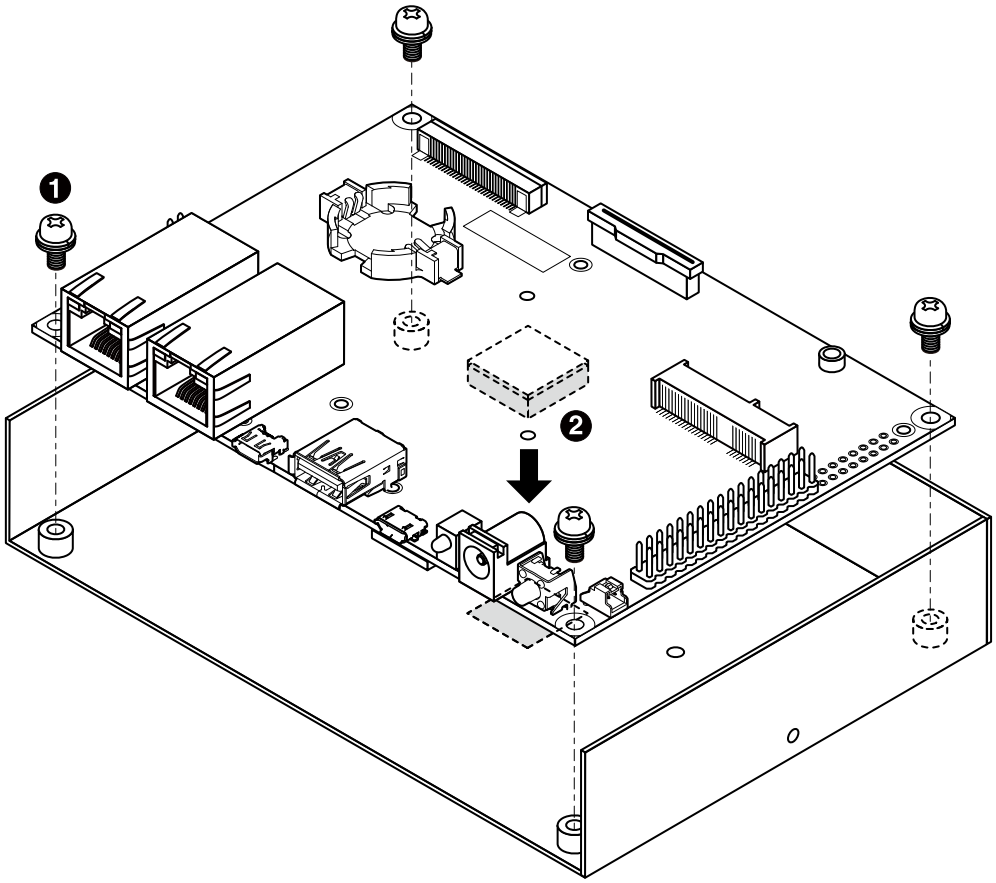


図 14.22 オプションケース(金属製) ケース(下)ねじ止め

❶ なべ小ねじ、ワッシャ、スプリングワッシャ付き(M3、L=6mm) × 4

❷ 放熱シート(15×15×4mm)

ケース(上)はコネクタ、スイッチ、LED が曲がらないように注意しながら、フロントに空いた穴に嵌めつつ閉め、2 箇所ねじ止めします。ねじの締め付けトルクは 17.5cN·m です。

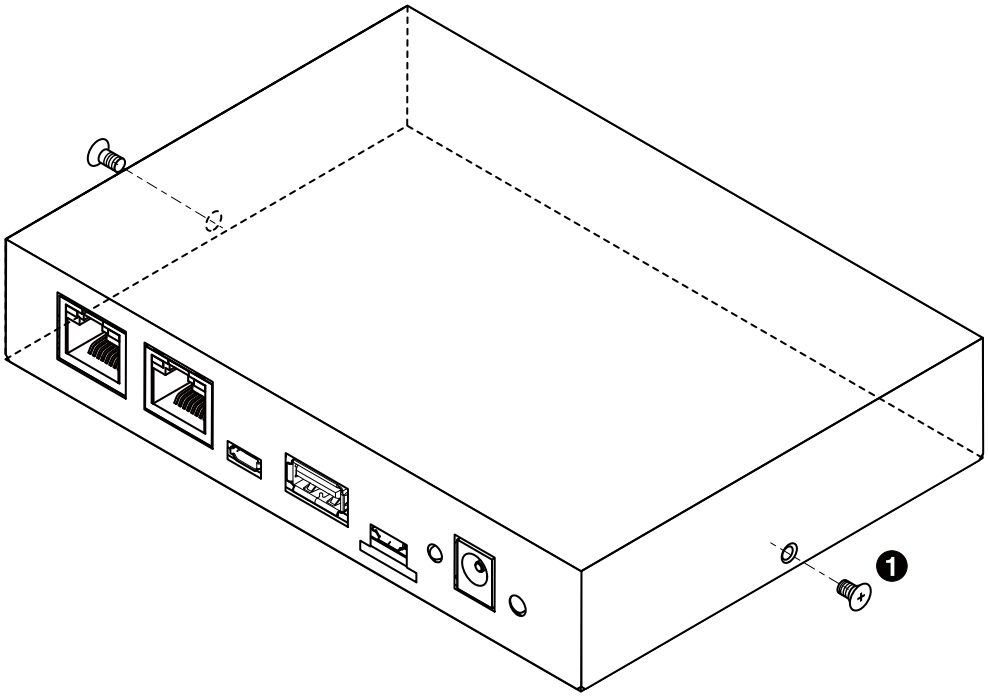


図 14.23 オプションケース(金属製) ケース(上)ねじ止め

❶ 皿ねじ(M2.6、L=4mm) × 2

14.5.1.3. 形状図

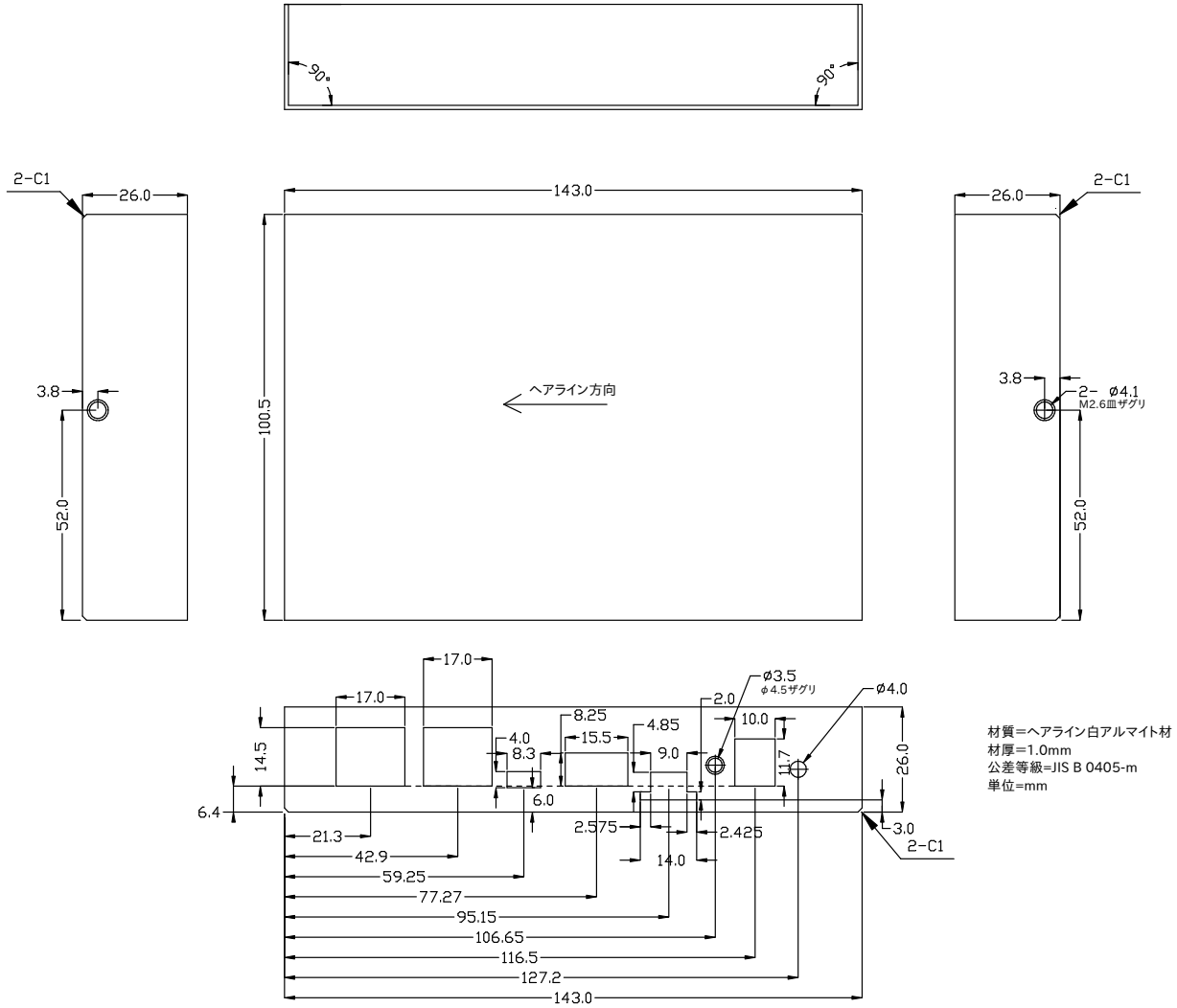


図 14.24 ケース(上)形状図

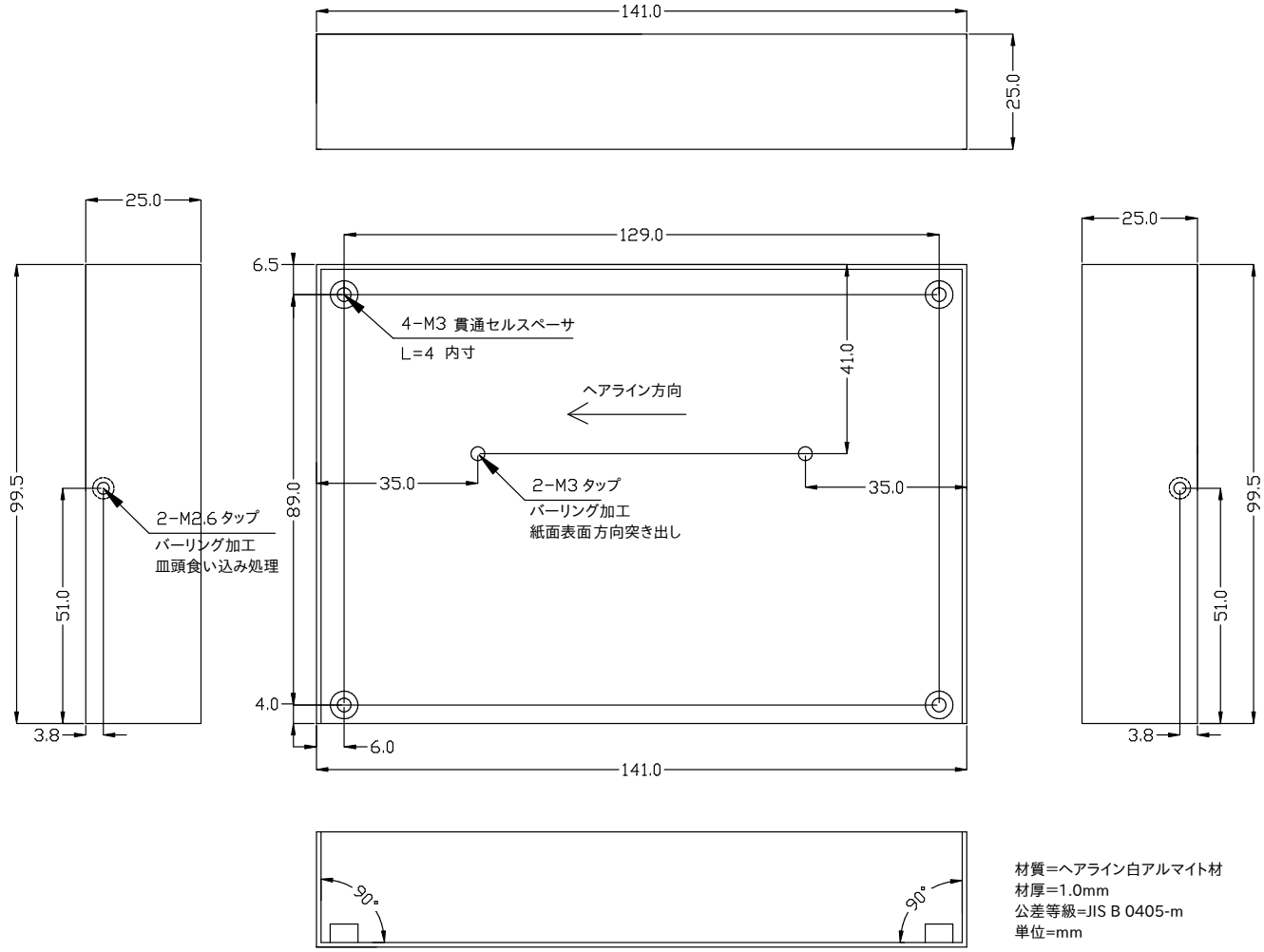


図 14.25 ケース(下)形状図

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

Armadillo-IoT ゲートウェイ G4 製品マニュアル

改訂履歴

バージョン	年月日	改訂内容
1.0.0	2021/12/10	・初版発行
1.0.1	2021/12/10	・ソフトウェアのダウンロードリンクを修正
1.1.0	2021/12/22	・「図 4.1. GNOME 端末の起動」 の画像を修正 ・「図 4.2. GNOME 端末のウィンドウ」 の画像を修正 ・「図 9.145. at-dtweb の起動開始」 の画像を修正 ・「5.3. 終了方法」 に電源再投入に関する注意を追記 ・「9.8.2. overlayfs と persist_file について」 を追加 ・「8.1.3. Podman のデータを eMMC に保存する」に詳細な説明を追加 ・「8.2.2. アプリケーションの送信」 の説明文を修正 ・「8.2.3. インストール確認：初期化」 の説明文を修正 ・「8.2.4. アプリケーションのアップデート」 の説明文を修正 ・「8.3. VPU や NPU を使用する」 のビルドツールのインストール手順を修正 ・「9.1.2.7. コンテナの自動作成やアップデート」 の説明文を修正 ・「9.2.2. pod の作成」 を追加 ・「9.2.3. network の作成」 を追加 ・「9.7.5.1. swupdate_preserve_files について」 を追加 ・「14.1.9.1. ONOFF ピンからの電源制御」 に電源再投入に関する注意を追記 ・「14.2.13. CON14、CON15(電源入力インターフェース)」 に電源再投入に関する注意を追記 ・誤記および分かりにくい表記の修正
1.2.0	2022/01/27	・「9.1.9.1. Wayland を扱う」 に weston.ini の設定例を追加 ・「9.1.13. NPU を扱う」 に PID に関する注意事項を追加 ・「9.2.1. コンテナの自動起動」 の readonly パラメータの説明を修正 ・「9.4.2. Linux カーネルをビルドする」 にカーネルコンフィギュレーションの変更方法を追加 ・「9.6.1. インストールディスクの作成」にカスタマイズしたインストールディスクイメージの作成方法を追加 ・「9.7.2. SWU イメージの作成」 の swupdate-mkimage の使用例を mkswu の使用例に修正 ・「9.9. Device Tree をカスタマイズする」 に 「9.9.3.2. 信号名の確認」 を追加 ・「12.6. HDMI」 に HDMI の映像出力の信号を停止する方法を追加 ・「12.15. 温度センサー」 のサーマルシャットダウンの温度を修正 ・「14.2.12. CON13(RTC バックアップインターフェース)」に電池を装着する際の注意事項を追加 ・「14.4. 設計情報」 に 「14.4.1. 信頼性試験データについて」 と「14.4.4. 放熱」 を追加 ・「14.5.1. オプションケース(金属製)」 にオプションケースへのネジの締付けトルクを追加 ・誤記および分かりにくい表記の修正
1.3.0	2022/02/24	・「9.1.4.1. GPIO を扱う」 に GPIO の制御に関する説明を追加 ・「9.1.4.7. USB を扱う」に USB ホットプラグに関する説明を追加 ・「9.1.9.1. Wayland を扱う」の weston.conf ファイルの設定例を修正 ・「9.1.9.1. Wayland を扱う」 に at-weston-launch コマンドに関する説明を追加 ・「9.1.13. NPU を扱う」 の PID に関する注意事項を削除

Armadillo-IoT ゲートウェイ G4 製品マニュアル

		・「9.2. コンテナの運用」 の podman_start の設定に関する説明を修正 ・「9.7.4. hawkBit サーバーから複数の Armadillo に配信する」 に hawkBit サーバー運用に関する手順を追加 ・「9.9. Device Tree をカスタマイズする」 に DTS overlays に関する説明を追加 ・「12.14. GPIO」 を追加 ・HTML 版の検索機能を修正 ・誤記および分かりにくい表記の修正
1.4.0	2022/03/30	・「表 3.2. 仕様」 の eMMC の容量に関する記載を修正 ・「表 3.2. 仕様」 の SD に関する記載の一部を削除 ・「表 3.2. 仕様」 の消費電力(参考値)の値を修正 ・「9.1.9.1. Wayland を扱う」 にスクリーンショットの保存に関する説明を追加 ・「9.1.13. NPU を扱う」 に 「9.1.13.1. ONNX Runtime を使う」 を追加 ・「9.1.13. NPU を扱う」に「9.1.13.2. TensorFlow Lite を使う」 を追加 ・「9.1.13. NPU を扱う」 に 「9.1.13.3. Arm NN を使う」 を追加 ・「9.2.1. コンテナの自動起動」 のボリュームマウントの説明に shared と slave オプションに関する説明を追加 ・「9.8. Armadillo Base OS の操作」 を追加 ・「9.9.4. DTS overlays によるカスタマイズ」 に 「9.9.4.2. カスタマイズした DTS overlay の作成」 を追加 ・「9.11. デモアプリケーションを実行する」 に 「9.11.9. pose estimation demo」 を追加 ・「9.11. デモアプリケーションを実行する」 に 「9.11.10. image segmentation demo」 を追加 ・「12.11. ユーザースイッチとイベント信号」 の説明文を修正 ・誤記および分かりにくい表記の修正

Armadillo-IoT ゲートウェイ G4 製品マニュアル
Version 1.4.0
2022/03/29