

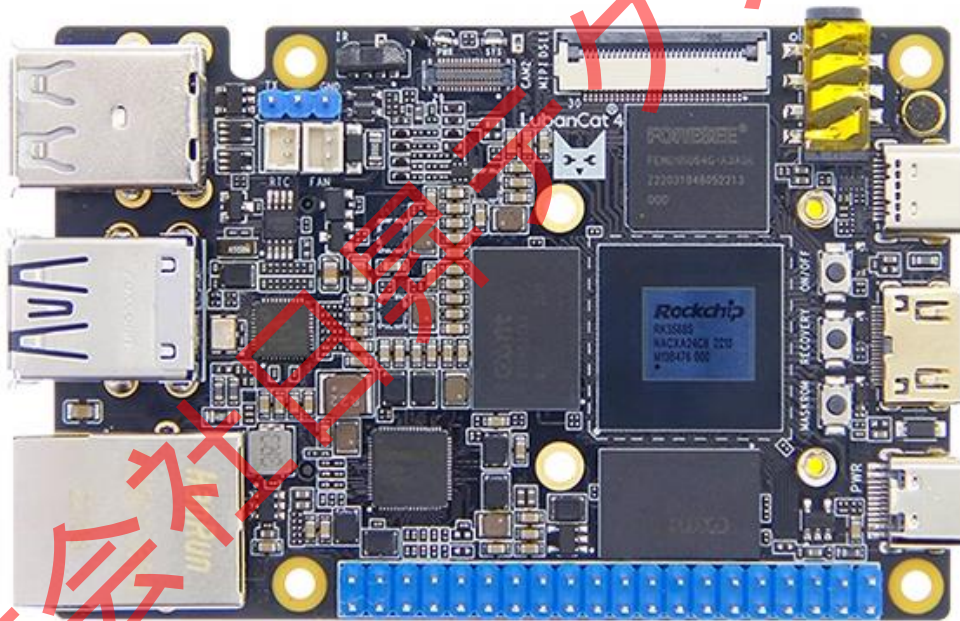
# 組み込み Linux イメージの構築と展開

株式会社日昇テクノロジー

<https://www.csun.co.jp>

[info@csun.co.jp](mailto:info@csun.co.jp)

作成日 2024/07/04



copyright@2024

- 修正履歴

NO	バージョン	修正内容	修正日
1	Ver1.0	新規作成	2024/07/04

※ この文書の情報は、文書を改善するため、事前の通知なく変更されることがあります。最新版は弊社ホームページからご参照ください。【<https://www.csun.co.jp>】

※ (株)日昇テクノロジーの書面による許可のない複製は、いかなる形態においても厳重に禁じられています。

## 目 次

文書説明.....	15
0.0.1 開発環境.....	15
第 1 章 Linux システム構成の簡単な紹介.....	16
1.1 Uboot.....	16
1.2 Linux カーネル.....	17
1.3 デバイスツリー.....	18
1.4 ルートファイルシステム.....	19
第 2 章 LubanCat-SDK.....	20
2.1 ボードのサポート状況.....	21
2.1.1 LubanCat-RK356x-Linux-SDK.....	21
2.1.2 LubanCat-RK3588-Linux-SDK.....	21
2.2 LubanCat-SDK の紹介.....	21
2.3 extboot と rkboot パーティションの比較.....	22
2.3.1 extboot パーティションシステムイメージの特徴.....	22
2.3.2 rkboot パーティションシステムイメージの特徴.....	23
2.3.3 比較.....	23
2.4 SDK 開発環境の構築.....	24
2.5 repo のインストール.....	24
2.6 SDK ソースコードの取得.....	25

2.6.1 Python 3 バージョンの切り替え .....	26
2.6.2 Git の設定 .....	27
2.6.3 SDK のオンラインダウンロードと同期 .....	27
2.6.4 SDK のオフラインインストールとダウンロード .....	29
2.6.5 SDK 更新 .....	32
2.7 LubanCat-SDK 自動ビルド .....	34
2.7.1 ルートファイルシステムイメージを使用して Debian システムイメージを構築 .....	38
2.7.2 ROOT ファイルシステムイメージを使用して Ubuntu システムイメージをビルド .....	40
2.8 LubanCat-SDK の個別モジュールビルド .....	43
2.8.1 SDK 設定ファイルの選択 .....	43
2.8.2 U-Boot のビルド .....	43
2.8.3 Kernel のビルド .....	43
2.8.4 rootfs のビルド .....	44
2.8.5 ファームウェアのパッケージング .....	46
2.9 SDK 設定ファイルの説明 .....	47
第3章 U-boot の紹介 .....	51
3.1 U-boot について .....	51
3.2 U-boot の起動 .....	52
3.3 U-boot のショートカットキー .....	56
3.4 U-boot のコマンド .....	56

3.4.1 U-boot の一般的なコマンド .....	62
3.4.2 mmc コマンド .....	64
3.4.3 ファイルシステム操作コマンド .....	67
3.5 U-boot でのカーネル起動プロセス .....	69
3.6 U-boot 環境パラメータの紹介 .....	76
第 4 章 U-boot の変更とコンパイル .....	77
4.1 U-boot の取得 .....	78
4.1.1 ソースコードのダウンロード .....	78
4.1.2 指定ブランチのダウンロード .....	78
4.2 U-boot のコンパイル .....	79
4.3 U-boot の変更 .....	83
4.3.1 U-boot の設定ファイル .....	83
4.3.2 デバイスツリーファイル .....	84
4.4 参考資料について .....	85
第 5 章 Linux の紹介 .....	86
5.1 Unix の簡単な紹介 .....	86
5.2 Linux の簡単な紹介 .....	86
5.3 Linux の歴史 .....	86
5.4 モノリシックカーネルとマイクロカーネルの違い .....	87
5.5 Linux カーネル .....	88

5.6 Linux カーネルの構成 .....	88
5.7 Linux 公式サイト .....	90
第 6 章 Linux カーネルのコンパイル .....	91
6.1 自分で Kernel をコンパイルする理由 .....	91
6.2 Kernel の取得 .....	91
6.2.1 ソースコードのダウンロード .....	91
6.3 kernel のプロジェクト構造分析 .....	92
6.4 カーネルの設定オプション .....	95
6.4.1 カーネルの設定オプションの配置 .....	95
6.5 カーネルのコンパイル .....	99
6.5.1 extboot パーティションのコンパイル .....	100
6.6 カーネル deb パッケージの構築 .....	103
6.6.1 カーネル deb パッケージのインストール .....	105
第 7 章 スタートアップロゴの変更 .....	107
7.1 SDK ソースコードからスタートアップロゴを変更 .....	107
7.1.1 画像の準備 .....	107
7.1.2 元のロゴファイルを置き換え .....	108
7.2 ボードシステムからスタートアップロゴを変更 .....	109
7.3 注意事項 .....	109
第 8 章 デバイスツリーの紹介 .....	109

8.1 デバイスツリーの紹介 .....	109
8.2 デバイスツリーの一般的なファイル .....	111
8.3 デバイスツリーのノードの書き方 .....	111
第9章 デバイスツリーのコンパイル .....	112
9.1 デバイスツリーファイルのコンパイル .....	112
9.2 同じボードでデバイスツリーファイルを変更する方法 .....	113
9.2.1 extboot パーティションでデバイスツリーファイルを変更する .....	113
第10章 ROOT ファイルシステムの紹介 .....	114
10.1 ROOT ファイルシステムの概要 .....	114
10.2 ROOT ファイルシステムディレクトリの概要 .....	114
10.3 ROOT ファイルシステムの一般的な特徴 .....	117
第11章 DebianROOT ファイルシステムの構築 .....	118
11.1 Debian のサポート状況 .....	118
11.1.1 LubanCat-RK3588 .....	118
11.2 Debian とは .....	118
11.3 Debian ルートファイルシステム構築リポジトリ .....	120
11.4 Debian ルートファイルシステムのビルドプロセス .....	121
11.5 ビルド環境の構築 .....	122
11.6 Debian ルートファイルシステムイメージの構築 .....	122
11.6.1 debian-base 基本ルートファイルシステムの構築 .....	123

11.6.2	完全な debian ルートファイルシステムの構築.....	124
11.6.3	debian-lite ルートファイルシステムイメージのパッケージング .....	126
11.7	Debian ルートファイルシステムのカスタマイズ.....	126
11.7.1	プリインストールソフトウェアの追加.....	126
11.7.2	外部デバイスの firmware の追加.....	127
11.7.3	サービス項目と設定ファイルの追加.....	127
11.7.4	ルートファイルシステムイメージの再パッケージング.....	129
11.8	LubanCat-SDK を使用したワンクリック構築.....	130
11.8.1	SDK 設定ファイルの説明.....	130
11.8.2	build.sh の自動ビルドスクリプト.....	132
11.8.3	ビルド前の準備.....	134
11.8.4	rootfs の単独構築とパッケージング.....	135
11.8.5	ワンクリックで完全なイメージを構築.....	136
第 12 章	Ubuntu ルートファイルシステムの構築.....	136
12.1	Ubuntu のサポート状況.....	137
12.1.1	LubanCat-RK358x.....	137
12.1.2	LubanCat-RK3588.....	137
12.2	Ubuntu Base とは.....	137
12.3	Ubuntu 構築リポジトリの取得.....	138
12.4	Ubuntu ルートファイルシステムの構築プロセス.....	140



12.5 構築環境の構築 .....	141
12.6 base イメージの構築 .....	142
12.6.1 基本ルートファイルシステムの構築 .....	142
12.7 Ubuntu ルートファイルシステムイメージのフルビルド .....	144
12.7.1 ルートファイルシステムのビルド .....	144
12.8 Ubuntu ルートファイルシステムのカスタマイズ .....	147
12.8.1 事前にインストールするソフトウェアパッケージの追加 .....	147
12.8.2 外部デバイスファームウェアの追加 .....	148
12.8.3 サービス項目及び設定ファイルの追加 .....	148
12.8.4 ルートファイルシステムイメージの再パッケージング .....	149
12.9 LubanCat-SDK を使用したワンクリック構築 .....	150
12.9.1 SDK 設定ファイルの説明 .....	150
12.9.2 build.sh の自動構築スクリプト .....	151
12.9.3 コンパイル前の準備 .....	153
12.9.4 rootfs の単独構築とパッケージング .....	153
12.9.5 ワンクリックで完全なイメージを構築 .....	154
第 13 章 Docker を使用したルートファイルシステムの構築 .....	154
13.1 Docker とは .....	155
13.2 Docker のインストール .....	156
13.2.1 旧バージョンのアンインストール .....	156

13.2.2	リポジトリからのインストール	156
13.3	ローカル Docker イメージの作成	157
13.4	Docker コンテナの作成	159
13.5	ルートファイルシステムの構築	160
13.6	Docker の関連操作コマンド	161
13.6.1	現在のコンテナから退出	161
13.6.2	現在実行中のコンテナを表示	161
13.6.3	停止したコンテナを再起動して入る	161
13.6.4	コンテナの削除	161
13.6.5	イメージの削除	161
第 14 章	Systemd の探求	162
14.1	システム管理	162
14.2	Systemd の核心概念	163
14.2.1	unit (ユニット)	163
14.2.2	ユニットの管理	167
14.2.3	ユニットの依存関係	168
14.2.4	ユニットの設定ファイル	169
14.2.5	ユニットのシステム管理	171
14.2.6	ユニットのログ管理	171
14.3	Systemd の実例分析	172

14.3.1 起動順序と依存関係 .....	172
14.3.2 起動コマンド .....	173
14.3.3 起動タイプと振る舞い .....	174
14.3.4 インストール方式 .....	175
14.4 Systemd で自分の Systemd サービスを作成する .....	175
14.4.1 スクリプトの作成 .....	176
14.4.2 設定ファイルの作成 .....	176
14.4.3 hello.service の自動起動機能を有効化 .....	177
14.5 Systemd の基本ツール .....	179
14.5.1 systemctl .....	179
14.5.2 systemd-analyze .....	182
14.5.3 hostnamectl .....	187
14.5.4 localectl .....	189
14.5.5 timedatectl .....	190
14.5.6 loginctl .....	191
第 15 章 Linux で deb パッケージを作成する方法 .....	191
15.1 deb パッケージとは？ .....	191
15.2 deb パッケージの構造 .....	194
15.3 自分の deb パッケージをゼロから作成 .....	194
15.4 既存の deb パッケージの内容を変更する方法 .....	201

15.5 deb パッケージをインストールして自動起動サービスを作成する .....	204
第 16 章 自動起動サービスを追加する .....	209
16.1 Systemd 方式 .....	209
16.1.1 スクリプトを書く .....	209
16.1.2 設定ファイルを作成する .....	210
16.1.3 hello.service の自動起動機能を有効にする .....	211
16.2 デスクトップシステム方式 .....	212
16.2.1 自動起動設定スクリプトの作成 .....	212
16.2.2 自動起動スクリプトの作成 .....	213
第 17 章 システムサービスをルートファイルシステムのビルドに追加する .....	214
17.1 スクリプトを書く .....	215
17.2 設定ファイルを作成する .....	215
17.3 hello.service の自動起動機能を有効にする .....	216
第 18 章 バックアップと量産について .....	217
18.1 イメージのダウンロード .....	218
18.1.1 SD カード .....	218
18.1.2 eMMC .....	219
18.2 完全イメージの分割と統合 .....	220
18.3 バックアップと復元 .....	220
18.3.1 SD カードまたは eMMC の内容を完全バックアップ及び書込 .....	220

18.3.2	ルートファイルシステムパーティションのバックアップ .....	221
18.3.3	PC で rootfs.img イメージを修正 .....	221
18.4	システムサービスの追加 .....	221
18.4.1	ボードへのシステムサービスの追加または変更 .....	222
18.4.2	ビルドスクリプトへのシステムサービスの追加または変更 .....	222
第 19 章	SD カード起動イメージの書き込み .....	222
第 20 章	eMMC 起動鏡像書き込み .....	224
20.1	SD カードを使用した eMMC へのイメージ書き込み .....	225
20.2	Rockchip 開発ツールによる USB 経由での eMMC へのイメージ書き込み .....	228
20.2.1	RKDevTool (Windows) .....	229
20.2.2	Linux_Upgrade_Tool (Linux) .....	241
20.3	USB 量産ツールによる eMMC へのイメージ書き込み .....	246
第 21 章	完全イメージの解体と再構築 .....	251
21.1	RKDevTool による解体と再構築 (Windows) .....	252
21.1.1	解体 .....	252
21.1.2	再構築 .....	253
21.2	Linux_Pack_Firmware による解体とパッケージング (Linux) .....	259
21.2.1	解体 .....	259
21.2.2	パッケージング .....	262
第 22 章	システムイメージのバックアップと再書き込み .....	265
22.1	Win32DiskImager で SD カードシステムイメージの全体バックアップと再書き込み .....	266

22.2 dd コマンドで SD カードシステムイメージの圧縮バックアップと再書き込み .....	267
22.3 dd コマンドで eMMC のシステムイメージの圧縮バックアップ .....	273
22.3.1 拡張とランダム MAC アドレスのネットワークポート .....	277
22.4 RKDevTool を使用して eMMC に RAW 形式のイメージを書き込む .....	283
22.5 dd コマンドを使用して eMMC に RAW 形式のイメージを書き込む .....	286
第 23 章 ルートファイルシステムのバックアップと再書き込み .....	289
23.1 バックアップ前の準備 .....	290
23.2 ルートファイルシステムのバックアップ .....	290
23.3 パーティションの書き込み .....	294
23.4 完整イメージのパッケージングと書き込み .....	294
23.5 修正結果の検証 .....	296
第 24 章 rootfs.img イメージ内のファイルを修正 .....	297
24.1 RKDevTool での解凍、修正、パッケージング (Windows) .....	297
24.2 Linux_Pack_Firmware で解凍、編集、パッケージング(Linux) .....	303

# 文書説明

この文書は、Linux システムの使用経験があるユーザーを対象にしており、LubanCat のボード上での開発、デプロイ、および量産のためのアプリケーションのバックアップ方法について説明しています。

この文書では、LubanCat OS を例に、LubanCat-RK のボードに適したイメージを構築する方法と、イメージとアプリケーションのデプロイに関する詳細な説明を含め、Linux ベースのエンジニアリングアプリケーションの迅速な開発に非常に便利で明確な参考を提供しています。

以下のリンクをクリックすると、このプロジェクトの文書をオンラインで読むことができます。「組み込み Linux イメージの構築とデプロイメント - LubanCat-RK ボードベース」

## 0.0.1 開発環境

本チュートリアルで使用される開発環境は以下の通りです：

- ・ PC システム Windows：デフォルトでは Win10 64 ビットを使用し、Win7 などのシステムにも対応しています。RockChip の書き込みツールを使用する際には、Windows システムが必要です。
- ・ PC システム Linux：Ubuntu18.04 を使用し、同じバージョンの使用を強く推奨します。
- ・ Lubancat：主に Lubancat イメージを用いて説明します。

# 第 1 章 Linux システム構成の簡単な紹介

完全な Linux システムは、通常、Uboot、kernel、デバイスツリー、およびルートファイルシステムを含んでいます。

## 1.1 Uboot

U-Boot は、組み込みシステム用のブートローダーで、PPC、ARM、AVR32、MIPS、x86、68k、Nios、MicroBlaze など、多くの異なるコンピューターシステムアーキテクチャをサポートできます。これは GNU 一般公衆ライセンスの下でリリースされたフリーソフトウェアの一つです。Uboot の正式名称は Universal Boot Loader で、GPL 条項に従うオープンソースプロジェクトです。U-Boot の主な役割は、オペレーティングシステムのカーネルを起動することで、boot + loader の二段階で構成されています。boot 段階では、システムを起動し、ハードウェアデバイスを初期化し、メモリスペースのマッピングを構築し、システムのハードウェアとソフトウェアを適切な状態にします。loader 段階では、オペレーティングシステムのカーネルファイルをメモリにロードし、その後、カーネルのアドレスにジャンプして実行します。

また、一部の BootLoader には、オペレーティングシステムのイメージを検証したり、複数のオペレーティングシステムイメージから適切なものを選択してブートする機能、またはネットワーク機能を追加してシステムが自動的に適切なイメージをネット上で探し、ブートするなどの高度な特徴を持つことがあります。

Bootloader	Monitor	説明	X86	ARM	PowerPC
LILO	×	Linux ディスクブートローダー	○	×	×
GRUB	×	GNU の LILO の代替プログラム	○	×	×
Loadlin	×	DOS から Linux をブート	○	×	×
ROLO	×	ROM から BIOS を必要とせずに Linux をブート	○	×	×
Etherboot	×	イーサネットカード経由で Linux システムを起動	○	×	×

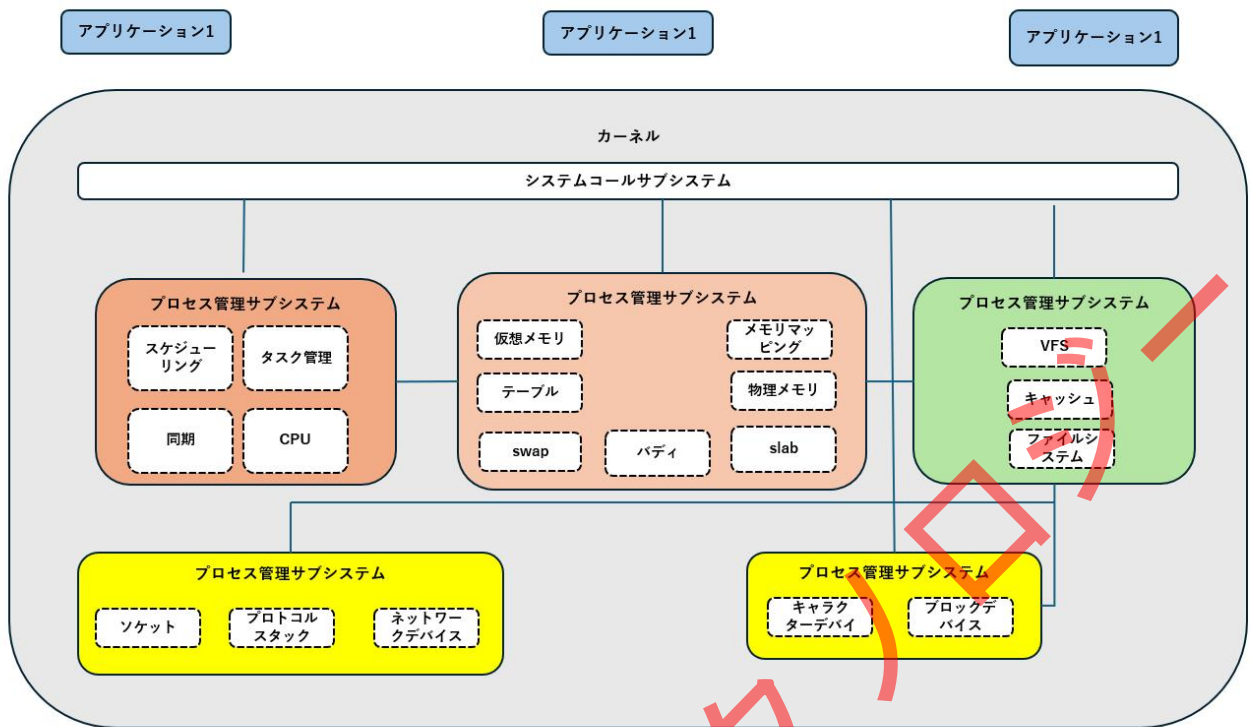


		するファームウェア			
LinuxBIOS	×	BIOS を完全に置き換える Linux ブートローダー	○	×	×
BLOB	×	LART などのハードウェアプラットフォームのブートローダー	×	○	×
U-boot	○	ブートローダーを介して	○	○	○
RedBoot	○	eCos のブートローダーを提供します	○	○	○

## 1.2 Linux カーネル

Linux は、オープンソースのコンピュータオペレーティングシステムカーネルです。これは C 言語で書かれ、POSIX 標準に準拠した Unix ライクなオペレーティングシステムです。Linux カーネルは、ハードウェアとやり取りし、ユーザープログラムに限定されたサービスセットを提供する低レベルのサポートソフトウェアです。

コンピューターシステムは、相互依存し、分割不可能なハードウェアとソフトウェアの共生体です。コンピュータのハードウェアは、周辺機器、プロセッサ、メモリ、ハードディスク、その他の電子デバイスで構成されており、コンピュータのエンジンを形成します。しかし、ソフトウェアがそれを操作して制御しなければ、それ自体では機能しません。この制御作業を行うソフトウェアをオペレーティングシステムといい、Linux の用語では「カーネル」と呼ばれます。Linux カーネルの主要なモジュール（またはコンポーネント）には、プロセス管理サブシステム、メモリ管理サブシステム、ファイルサブシステム、ネットワークサブシステム、デバイスサブシステムなどがあります。



### 1.3 デバイスツリー

デバイスツリーは、ハードウェアのデータ構造を記述するもので、これらのハードウェアデバイスの情報を記述し、このファイルがデバイスツリー (Device Tree) です。デバイスツリーには、デバイスツリーソース (Device Tree Source、DTS) ファイル、デバイスツリーコンパイラ (Device Tree Compiler、DTC)、およびバイナリ形式のデバイスツリー (Device Tree Blob、DTB) が含まれています。DTS に含まれるヘッダーファイルの形式は DTSI です。

リスト 1: デバイスツリーの説明

```

1 node1 {
2 a-string-property = "A string";
3 a-string-list-property = "first string", "second string";
4 a-byte-data-property = [0x01 0x23 0x34 0x56];
5

```

```

6 child-node1 {
7 first-child-property;
8 second-child-property = <1>;
9 a-string-property = "Hello, world";
10 };
11 };
  
```

Uboot と Linux は DTS ファイルを直接認識できませんが、DTB はカーネルと BootLoader によって認識され解析されます。通常、NAND Flash や SD カードの起動イメージを作成する際には、DTB ファイル用に一部のストレージ領域を予約します。BootLoader がカーネルを起動する際には、まず DTB をメモリに読み込み、その後カーネルに提供します。

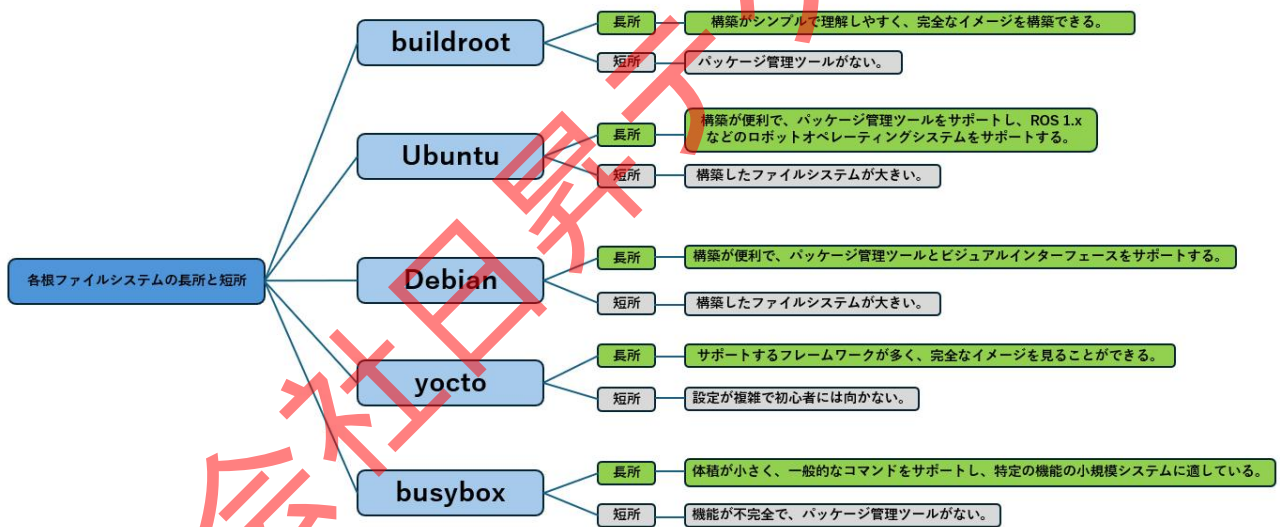


## 1.4 ルートファイルシステム

ルートファイルシステム (rootfs) は、Linux が初期化時に最初にロードするファイルシステムで、ルートディレクトリと実際のファイルシステムを含みます。これには、システムブートと他のファイルシステムをマウント (mount) するために必要なファイルが含まれています。ルートファイルシステムは、Linux の起動時に必要なディレクトリや重要なファイルを含んでおり、例えば、Linux の起動時に必要な初期化ファイルが init ディレクトリ下にあります。また、ルートファイルシステムには多くのアプリケーション bin ディレクトリなども含まれており、これら Linux システムの起動に必要なファイルを含むものは何でもルートファイルシステムとなることができます。

Linux カーネルの起動初期段階では、まずカーネルはメモリベースのファイルシステム、例えば `initramfs` や `initrd` などを初期化し、その後、ルートファイルシステム (`rootfs`) を読み取り専用でロードし、`/sbin/init` 初期化ファイルを読み取り実行します。`/etc/inittab` 設定ファイルに基づいてシステムの初期化作業を完了します（ヒント：`/sbin/init` はシステムの初期化プログラムのためのバイナリ実行可能ファイルで、`/etc/inittab` はその設定ファイルです）。初期化の過程で、ルートファイルシステムは読み書き可能な形で再マウントされ、システム起動後にはデータの保存に使用できるようになります。Linux の起動時にルートファイルシステムが必要です。

一般的なルートファイルシステム作成ツールには `buildroot`、`Ubuntu`、`Debian`、`yocto`、`busybox` があり、これらのツールの利点と欠点が以下に示されています。



後では、`Ubuntu` と `Debian` のルートファイルシステムの作成プロセスについて詳しく説明します。これら2つのルートファイルシステムは、主にサポートされているバージョンです。

## 第2章 LubanCat-SDK

注意: LubanCat 専用イメージは 2023 年 4 月 11 日にサポート終了しました。専用イメージに関連する一部の内容は削除されていませんが、参考のために残されています。また、本書類では関連する位置にサポート終了のマークを追加しています。

注意: LubanCat Buildroot に関連する内容は 2023 年 8 月 3 日にサポート終了しました。Buildroot 構築に関連する一部の内容は削除されていませんが、参考のために残されています。また、本書類では関連する位置にサポート終了のマークを追加しています。

## 2.1 ボードのサポート状況

### 2.1.1 LubanCat-RK356x-Linux-SDK

LubanCat-RK356x の LubanCat ボードシステムイメージの構築をサポート

### 2.1.2 LubanCat-RK3588-Linux-SDK

LubanCat-RK3588 の LubanCat ボードシステムイメージの構築をサポート

- ・ RK3588s メインチップを使用

本書類では、LubanCat-RKxxxx-Linux-SDK を LubanCat-SDK と総称します。

## 2.2 LubanCat-SDK の紹介

LubanCat-SDK は、Rockchip プロセッサをメインチップとする LubanCat-RK ボードに適した、Rock Chip 汎用 Linux SDK エンジニアリングのカスタマイズバージョンです。

SDK エンジニアリング全体には、buildroot、debian、app、kernel、u-boot、device、external などのディレクトリが含まれています。各ディレクトリまたはそのサブディレクトリは、それぞれのディレクトリでのコミットが必要な git プロジェクトに対応しています。

- ・ app : (サポート終了) 上層アプリケーション app を保存し、主に qcamera/qfm/qplayer/settings などのアプリケーションが含まれます。buildroot 構築時にこれらのプログラムが使用されます。
- ・ buildroot : (サポート終了) buildroot (2018.02-rc3)を基に開発されたルートファイルシステム。
- ・ debian : Debian ルートファイルシステム構築スクリプト。
- ・ device/rockchip : 各チップのボードレベルの設定と Parameter ファイル、およびいくつかの構築およびパッケージングファームウェアのスクリプトと予備ファイルを保存。

- external：（サポート終了）第三者関連のリポジトリを保存しており、音声、ビデオ、ネットワーク、リカバリーなどが含まれています。buildroot の構築時に使用されます。
- kernel：カーネルソースコードを保存します。
- prebuilts：クロスコンパイルツールチェーンを保存します。
- rkbin：Rockchip 関連のバイナリとツールを保存します。
- rockdev：コンパイル出力ファームウェアを保存します。
- tools：Linux と Windows のオペレーティングシステム環境で一般的に使用されるツールを保存します。
- u-boot：v2017.09 バージョンに基づいて開発された uboot コードを保存します。
- ubuntu：Ubuntu ルートファイルシステム構築スクリプト。

## 2.3 extboot と rkboot パーティションの比較

注意：rkboot パーティションに基づく LubanCat 専用イメージのサポートが終了したため、関連する SDK 設定ファイルは device/rockchip/rk358x/.others ディレクトリに移動されました。以下の内容は参考としてのみ提供されます。

### 2.3.1 extboot パーティションシステムイメージの特徴

extboot パーティションシステムは、Rock Chip Linux\_SDK フレームワークに基づいて構築した LubanCat-RK ボード用の汎用イメージ実装方式です。同一モデルのプロセッサを使用するすべてのボードに一つのイメージを書き込むことができ、デフォルトの rkboot パーティション方式ではデバイスツリーが固定されており、一つのイメージが一つのボードにのみ対応する問題を解決し、多数のモデルによる後期のメンテナンスの複雑さを大幅に低減しました。

extboot パーティションは ext4 ファイルシステム形式を使用し、コンパイルプロセス中にすべての LubanCat-RK ボードのデバイスツリーをコンパイルしてパーティション内にパッケージし、SDRADC を利用してボードのハードウェア ID を読み取り、デバイスツリーの自動切り替えを実現します。また、デバイ

スツリープラグインのサポート、カーネルdebパッケージの自動更新、カーネルとドライバーモジュールのオンライン更新などの機能をサポートしています。

システムイメージのパーティションも調整され、一部の冗長な機能を削除し、uboot、boot、rootfsパーティションのみを含むことで、システムストレージの効率的な利用を実現しています。

このバージョンは、主にサポートするバージョンです。

### 2.3.2 rkboot パーティションシステムイメージの特徴

rkbootパーティションは、Rock Chip Linux\_SDKに付属するbootパーティションパッケージングスクリプトを使用して構築されたパーティションで、カーネルの設定ファイルとボードのデバイスツリーのみを変更し、システムをよりオリジナルに近づけました。このタイプのbootパーティションを使用する利点は、起動が速く、冗長機能が強いことです。さらに設定を行うことでOTA、マルチシステム、イメージ検証などの機能を実現することが可能です（二次開発による自己適応が必要）。しかし、デバイスツリープラグインのような機能は欠けています。

rkbootパーティションのシステムイメージでは、一つのモデルのボードに一つのモデルのシステムイメージが対応しており、ハードウェアが変更されるとデバイスツリーを変更し、イメージを再構築する必要があります。

### 2.3.3 比較

機能	extboot	rkboot
起動速度	やや遅い	速い
同一プロセッサボード間のイメージ共有	サポート	サポートなし
システム内でのデバイスツリー切り替え	サポート	サポートなし
デバイスツリープラグイン	サポート	サポートなし

ストレージ空間利用率	高い	低い
カーネルのオンラインアップグレード	サポート	サポートなし

2つのタイプのパーティションシステムイメージについては、uboot は汎用的に処理され、rootfs も汎用的に処理されるため、uboot.img と rootfs.img は共通で使用できます。

## 2.4 SDK 開発環境の構築

LubanCat-SDK は Ubuntu LTS システムで開発、テストされています。開発中には主に Ubuntu 20.04 バージョンを使用しています。不必要なトラブルを避けるため、Ubuntu20.04 以上のバージョンの使用を推奨します。

注意: LubanCat-RK3588 SDK は Ubuntu20.04 以下のバージョン環境ではイメージの構築をサポートしていません

SDK ソフトウェアパッケージのインストール

```

1 # SDK 構築に必要なソフトウェアパッケージをインストール
2 # 以下の内容をコピーしてターミナルでインストール
3 sudo apt install git ssh make gcc libssl-dev liblz4-tool u-boot-tools curl ¥
4 expect g++ patchelf chrpath gawk texinfo chrpath diffstat binfmt-support ¥
5 qemu-user-static live-build bison flex fakeroot cmake gcc-multilib g++-multilib ¥
6 unzip device-tree-compiler python3-pip libncurses5-dev python3-pyelftools dpkg-dev
  
```

## 2.5 repo のインストール

```

1 mkdir ~/bin
2 curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
  
```



3 # 上記のアドレスがアクセス不可の場合、以下を使用：

```
4 # curl -sSL 'https://gerrit-googleusercontent.proxy.ustclug.org/git-repo/+master/repo?format=TEXT'
```

```
|base64 -d > ~/bin/repo
```

```
5 chmod a+x ~/bin/repo
```

```
6 echo PATH=~/.bin:$PATH >> ~/.bashrc
```

```
7 source ~/.bashrc
```

上記のコマンド実行後、repo が正常にインストールされ、動作するかを確認します。

```
1 repo --version
2
3 # 以下の情報が返されます
4 # 返される情報は Ubuntu のバージョンによって若干異なります
5 <repo not installed>
6 repo launcher version 2.32
7 (from /home/he/bin/repo)
8 git 2.25.1
9 Python 3.8.10 (default, Nov 14 2022, 12:59:47)
10 [GCC 9.4.0]
11 OS Linux 5.15.0-60-generic (#66~20.04.1-Ubuntu SMP Wed Jan 25 09:41:30 UTC 2023)
12 CPU x86_64 (x86_64)
13 Bug reports: https://bugs.chromium.org/p/gerrit/issues/entry?template=Repo+tool+issue
```

## 2.6 SDK ソースコードの取得

LubanCat-SDK のコードは複数の git リポジトリに分割されてバージョン管理されています。これらの git リポジトリに対して、repo ツールを使用して一元的にダウンロード、コミット、ブランチの切り替えなど

の操作を行うことができます。

以下のコマンドを実行すると、現在のユーザーのホームディレクトリに「LubanCat\_SDK」という名前のディレクトリが作成され、SDK ソースコードが配置されます。

```
1 mkdir ~/LubanCat_SDK
```

## 2.6.1 Python 3 バージョンの切り替え

```
1 #現在の Python バージョンを確認します。
```

```
2 python -V
```

返されたバージョンが Python3 であれば、Python のバージョンを切り替える必要はありません。Python2 バージョンまたは python が見つからない場合は、以下の方法で Python のバージョンを切り替えることができます。

```
1 # 現在のシステムにインストールされている Python バージョンを確認
```

```
2 ls /usr/bin/python*
```

```
3
```

```
4 # python を python3 にリンク
```

```
5 sudo ln -sf /usr/bin/python3 /usr/bin/python
```

```
6
```

```
7 # デフォルトの Python バージョンを再確認
```

```
8 python -V
```

この操作により、システムのデフォルトの Python バージョンが python3 に切り替わります。

```

dev@dev_152:~/LINUX$ python -V
Python 2.7.17
dev@dev_152:~/LINUX$ ls /usr/bin/python*
/usr/bin/python          /usr/bin/python3.6      /usr/bin/python3m
/usr/bin/python2         /usr/bin/python3.6m    /usr/bin/python-config
/usr/bin/python2.7       /usr/bin/python3-jsondiff /usr/bin/pythontex
/usr/bin/python2.7-config /usr/bin/python3-jsonpatch /usr/bin/pythontex3
/usr/bin/python2-config  /usr/bin/python3-jsonpointer
/usr/bin/python3         /usr/bin/python3-jsonschema
dev@dev_152:~/LINUX$ sudo ln -sf /usr/bin/python3 /usr/bin/python
dev@dev_152:~/LINUX$ python -V
Python 3.6.9
dev@dev_152:~/LINUX$
  
```

## 2.6.2 Git の設定

後でコードを正常にプルするために、自分の git 情報を設定します。コードを提出する必要がない場合は、ユーザー名とメールアドレスを任意に設定できます。

```

1 git config --global user.name "your name"
2 git config --global user.email "your mail"
  
```

## 2.6.3 SDK のオンラインダウンロードと同期

LubanCat-SDK は異なるボードに適応することができます。この方法を使用すると、GitHub から多くのリポジトリをプルする必要があるため、データの量が非常に大きくなります。GitHub に素早くアクセスできないユーザーにはこの方法は推奨されません。

```

1 cd ~/LubanCat_SDK
2
3 # LubanCat-RK358x の Linux_SDK を取得します
4 repo init --depth=1 -u https://github.com/LubanCat/manifests.git -b linux -m rk358x_linux_release.xml
5
6
7 # LubanCat-RK3588 の Linux_SDK を取得します
8 repo init --depth=1 -u https://github.com/LubanCat/manifests.git -b linux -m rk3588_linux_release.xml
  
```

9

10 # 上記のコマンドを実行時に失敗し、「fatal: Cannot get <https://gerrit.googlesource.com/git-repo/clone.bundle>」というメッセージが表示された場合

11 # 上記のコマンドに--repo-url <https://mirrors.tuna.tsinghua.edu.cn/git/git-repo> オプションを追加してください

12

13 .repo/repo/repo sync -c -j4

--depth=1 は、最新のコミットのみを浅くクローンするために使用され、ネットワークからのプル量を大幅に減らすことができます。すべての Git コミット情報を含む完全な内容をプルしたい場合は、このオプションを削除してください。

```
dev@dev_152:~/LubanCat_SDK$ repo init -u https://github.com/LubanCat/manifests.git -b linux -m rk356x_linux_release.xml
warning: Python 3 support is currently experimental. YMMV.
Please use Python 2.6 - 2.7 instead.

... A new version of repo (2.29) is available.
... New version is available at: /home/dev/LubanCat_SDK/.repo/repo/repo
... The launcher is run from: /usr/bin/repo
!!! The launcher is not writable. Please talk to your sysadmin or distro
!!! to get an update installed.

Your identity is: ██████████
If you want to change this, please re-run 'repo init' with --config-name

repo has been initialized in /home/dev/LubanCat_SDK
```

同期に失敗した場合は、sync コマンドを再実行して同期できます。

```

dev@dev_152:~/LubanCat_SDK$ .repo/repo/repo sync -c -j4
Fetching: 58% (17/29) u-boot
remote: Enumerating objects: 1708, done.
remote: Counting objects: 0% (1/300)
remote: Counting objects: 1% (3/300)
remote: Counting objects: 2% (6/300)
remote: Counting objects: 3% (9/300)
remote: Counting objects: 4% (12/300)
remote: Counting objects: 5% (15/300)
remote: Counting objects: 6% (18/300)
remote: Counting objects: 7% (21/300)
remote: Counting objects: 8% (24/300)
remote: Counting objects: 9% (27/300)
remote: Counting objects: 10% (30/300)
remote: Counting objects: 11% (33/300)
remote: Counting objects: 12% (36/300)
remote: Counting objects: 13% (39/300)
remote: Counting objects: 14% (42/300)
  
```

```

dev@dev_152:~/LubanCat_SDK$ .repo/repo/repo sync -c -j4
Fetching: 86% (25/29) camera_engine_rkaiq
Fetching: 100% (29/29), done in 28m49.416s
NOT Garbage collecting: 0% (0/29), done in 0.010s
Checking out files: 100% (17367/17367), done.
Checking out files: 100% (696/696), done.
Checking out files: 100% (1196/1196), done.
Checking out files: 100% (1487/1487), done.
Checking out files: 100% (141/141), done.
Checking out files: 100% (960/960), done.
Checking out files: 100% (75371/75371), done.
Checking out files: 100% (17900/17900), done.
Checking out files: 100% (7165/7165), done.
Checking out files: 100% (237/237), done.
Checking out files: 100% (13583/13583), done.
Checking out: 100% (29/29), done in 44.538s
repo sync has finished successfully.
  
```

## 2.6.4 SDK のオフラインインストールとダウンロード

ネットワークが不安定な時にダウンロードに失敗する可能性があります。このため、必要なリポジトリを全てパッケージ化し、ネットディスクからダウンロードする方法を採用して、GitHub への接続による問題を減らします。

### 2.6.4.1 ダウンロードアドレス

注意: ソースコード圧縮パッケージの容量が大きいため、大きな変更があった安定版のみを更新し、そのリリース日がイメージのリリース日と一致しない場合があります。ローカルで圧縮パッケージを解凍した後、GitHub を少量更新するだけで、最新バージョンに同期できます。

## 2.6.4.2 ソースコードの解凍

以下の手順は LubanCat\_RK358x\_Linux\_SDK を例にしていますが、実際のファイル名はダウンロードした SDK によって異なります。

```
1 # 7z 圧縮ツールのインストール
2 sudo apt install p7zip-full
3
4 # ユーザーホームディレクトリに LubanCat_SDK ディレクトリを作成
5 mkdir ~/LubanCat_SDK
6
7 # ダウンロードした SDK ソースコードを LubanCat_SDK ディレクトリに移動、xxx は日付
8 mv LubanCat_RK358x_Linux_SDK_xxx.7z ~/LubanCat_SDK
9
10 # LubanCat_SDK ディレクトリに移動
11 cd ~/LubanCat_SDK
12
13 # SDK 圧縮パッケージを解凍
14 7z x LubanCat_RK358x_Linux_SDK_xxx.7z
15
16 # .repo ディレクトリ内の git リポジトリをチェックアウト
17 .repo/repo/repo sync -l
18
19 # すべてのソースコードリポジトリを最新バージョンに同期
```

```
20 .repo/repo/repo sync -c
```

repo sync -c の実行中にネットワーク接続タイムアウトが発生した場合は、GitHub へのアクセスがスムーズか確認してください。GitHub へのアクセスが問題ない場合は、repo sync -c コマンドを複数回実行して同期を試みてください。GitHub にアクセスできない場合は、ソースコードリポジトリを最新バージョンに同期するステップをスキップしてください。

```
● dev@dev_152:~$ sudo apt install p7zip-full
Reading package lists... Done
Building dependency tree
Reading state information... Done
p7zip-full is already the newest version (16.02+dfsg-6).
0 upgraded, 0 newly installed, 0 to remove and 5 not upgraded.
● dev@dev_152:~$ mkdir ~/LubanCat_SDK
● dev@dev_152:~$ mv LubanCat_Linux_SDK_20221130.7z ~/LubanCat_SDK
● dev@dev_152:~$ cd ~/LubanCat_SDK
● dev@dev_152:~/LubanCat_SDK$ ls
LubanCat_Linux_SDK_20221130.7z
● dev@dev_152:~/LubanCat_SDK$ 7z x LubanCat_Linux_SDK_20221130.7z

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,88 CPUs: Intel(R) Xeon(R) CPU E5-2696 v4 @ 2.20GHz (406F1),ASM,AES-NI)

Scanning the drive for archives:
1 file, 4837528980 bytes (4614 MiB)

Extracting archive: LubanCat_Linux_SDK_20221130.7z
--
Path = LubanCat_Linux_SDK_20221130.7z
Type = 7z
Physical Size = 4837528980
Headers Size = 23593
Method = LZMA2:24
Solid = +
Blocks = 3

Everything is Ok

Folders: 1025
Files: 1446
Size: 5105741187
Compressed: 4837528980
● dev@dev_152:~/LubanCat_SDK$ ls -a
. .. LubanCat_Linux_SDK_20221130.7z .repo
○ dev@dev_152:~/LubanCat_SDK$
```

解凍後、指定されたコミットにチェックアウトします。

```
● dev@dev_152:~/LubanCat_SDK$ .repo/repo/repo sync -l
Checking out files: 100% (17367/17367), done.
Checking out files: 100% (696/696), done.
Checking out files: 100% (1198/1198), done.
Checking out files: 100% (1487/1487), done.
Checking out files: 100% (73/73), done.
Checking out files: 100% (960/960), done.
Checking out files: 100% (75373/75373), done.
Checking out files: 100% (17900/17900), done.
Checking out files: 100% (7165/7165), done.
Checking out files: 100% (237/237), done.
Checking out: 100% (29/29), done in 31.111s
repo sync has finished successfully.
● dev@dev_152:~/LubanCat_SDK$ .repo/repo/repo sync -c
remote: Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
Fetching: 100% (29/29), done in 0.824s
NOT Garbage collecting: 0% (0/29), done in 0.001s
repo sync has finished successfully.
○ dev@dev_152:~/LubanCat_SDK$
```

オフラインソースパッケージを使用する顧客は、初回のリポジトリチェックアウト後にソースを最新バージョンに更新することをお勧めします。

## 2.6.5 SDK 更新

LubanCat-SDK は継続的に更新され、変更内容はリアルタイムで GitHub に同期されます。ローカルの LubanCat-SDK で最新の更新内容を同期するには、repo または git を利用できます。

### 2.6.5.1 全 SDK を repo で更新

repo を使用して SDK を最新バージョンに更新できます。

最初に .repo/manifests を更新し、repo の設定ファイルが保存されていることを確認します。これにはリポジトリのバージョン情報が記録されています。

```
1 # .repo/manifests ディレクトリに入る
2 cd .repo/manifests
3
4 # Linux ブランチに切り替える
5 git checkout linux
6
```



```
7 # 最新の manifests をプルする
8 git pull
9
10 # SDK のルートディレクトリに戻る
11 cd ~/LubanCat_SDK
12
13 # リモートリポジトリを同期する
14 .repo/repo/repo sync -c
```

```
● dev@dev_152:~/LubanCat_SDK$ cd .repo/manifests
● dev@dev_152:~/LubanCat_SDK/.repo/manifests$ git pull
Updating eb35cbe..e3f0107
Fast-forward
 README.md | 11 +++++-----
 rk356x linux/rk356x linux dev.xml | 10 +++++-----
 2 files changed, 10 insertions(+), 11 deletions(-)
● dev@dev_152:~/LubanCat_SDK/.repo/manifests$ cd ~/LubanCat_SDK
● dev@dev_152:~/LubanCat_SDK$ .repo/repo/repo sync -c --no-tags
Fetching: 100% (29/29), done in 0.662s
NOT Garbage collecting: 0% (0/29), done in 0.001s
kernel/: manifest switched refs/heads/stable-4.19-rk356x...27fc0130643a8be83f86c80c5d4fced232c7a694
kernel/: discarding 4 commits removed from upstream
project kernel/
First, rewinding head to replay your work on top of it...
Applying: kernel-android-config:refresh kernel configuration
Applying: add JL2101 rx/tx delayline
Applying: kernel:lubancat2 fix ir infrared and headphone unplugging problem
Applying: DTSはLubancat mipiとデュアルスクリーン表示デバイスツリーを追加

project kernel/

Checking out: 100% (29/29), done in 1.520s
repo sync has finished successfully.
○ dev@dev_152:~/LubanCat_SDK$
```

## 2.6.5.2 Git で個別のソースコードリポジトリを更新

時には、全 SDK を更新するのではなく、特定のリポジトリのみを更新したい場合があります。また、SDK の一部のリポジトリに変更を加えた場合、repo で同期すると失敗することがあります。このような場合には、個別のリポジトリを更新する必要があります。

例として、Kernel リポジトリを更新する方法は以下の通りです。

```
1 # kernel ディレクトリに入る
2 cd kernel
3
4 # 現在のコミットが属するブランチにチェックアウトする
5 # .repo/manifests/rk358x_linux_release.xml を確認
6 # name="kernel"項目の dest-branch が切り替えるべきブランチです
7 git checkout stable-4.19-rk358x
8
9 # git リポジトリをプルする
10 git pull
```

## 2.7 LubanCat-SDK 自動ビルド

LubanCat ボードに対応する設定ファイルリスト：

LubanCat-RK3588 共通イメージ: extboot パーティション、debian、

BoardConfig-LubanCat-RK3588-debian-(バージョン).mk

LubanCat-SDK のビルドスクリプトを使用して自動ビルドを実行できます。具体的な操作方法は以下の通りです：

SDK のルートディレクトリで以下のコマンドを実行し、SDK の設定ファイルを選択して、ビルドするボードとファイルシステムタイプを選択します。

```
1 # SDK 設定ファイルを選択
2 ./build.sh lunch
3
```

4 # 構築したいボードとファイルシステム設定ファイルの番号を入力し、確認してください。ここでは設定ファイル BoardConfig-LubanCat-RK3588-debian-xfce.mk を選択します。

5 どれを希望しますか？ [0]: 4

SDK 設定ファイルを直接設定する方法もあります

1 # LubanCat-RK3588 の一般的な debian イメージを構築するために選択

2 ./build.sh BoardConfig-LubanCat-RK3588-debian-xfce.mk

```
● jiawen@dev120:~/RK356X_LINUX_SDK$ ./build.sh lunch
processing option: lunch

You're building on Linux
Lunch menu...pick a combo:

0. default BoardConfig.mk
1. BoardConfig-LubanCat-Backup.mk
2. BoardConfig-LubanCat-RK3566-debian-lite.mk
3. BoardConfig-LubanCat-RK3566-debian-xfce-full.mk
4. BoardConfig-LubanCat-RK3566-debian-xfce.mk
5. BoardConfig-LubanCat-RK3566-ubuntu-lite.mk
6. BoardConfig-LubanCat-RK3566-ubuntu-xfce-full.mk
7. BoardConfig-LubanCat-RK3566-ubuntu-xfce.mk
8. BoardConfig-LubanCat-RK3568-debian-lite.mk
9. BoardConfig-LubanCat-RK3568-debian-xfce-full.mk
10. BoardConfig-LubanCat-RK3568-debian-xfce.mk
11. BoardConfig-LubanCat-RK3568-ubuntu-lite.mk
12. BoardConfig-LubanCat-RK3568-ubuntu-xfce-full.mk
13. BoardConfig-LubanCat-RK3568-ubuntu-xfce.mk
14. BoardConfig.mk
Which would you like? [0]: 4
switching to board: /home/jiawen/RK356X_LINUX_SDK/device/rockchip/rk356x/BoardConfig-LubanCat-RK3566-debian-xfce.mk
● jiawen@dev120:~/RK356X_LINUX_SDK$ ./build.sh BoardConfig-LubanCat-RK3566-debian-xfce.mk
processing option: BoardConfig-LubanCat-RK3566-debian-xfce.mk
switching to board: /home/jiawen/RK356X_LINUX_SDK/device/rockchip/rk356x/BoardConfig-LubanCat-RK3566-debian-xfce.mk
○ jiawen@dev120:~/RK356X_LINUX_SDK$
```

SDK の設定ファイルを選択した後、debian のルートファイルシステムのビルドに必要なソフトウェアパッケージをインストールします。

1 # ローカルソフトウェアパッケージをインストール

2 sudo dpkg -i debian/ubuntu-build-service/packages/\*

3 sudo apt-get install -f

```
dev@dev_152:~/LubanCat_SDK$ sudo dpkg -i debian/ubuntu-build-service/packages/*
(Reading database ... 160958 files and directories currently installed.)
Preparing to unpack .../debootstrap_1.0.87_all.deb ...
Unpacking debootstrap (1.0.87) over (1.0.87) ...
Preparing to unpack .../linaro-image-tools_2012.12-0ubuntu1~linaro1_all.deb ...
Unpacking linaro-image-tools (2012.12-0ubuntu1~linaro1) over (2012.12-0ubuntu1~linaro1) ...
Preparing to unpack .../live-build_3.0.5-1linaro1_all.deb ...
Unpacking live-build (3.0.5-1linaro1) over (3.0.5-1linaro1) ...
Preparing to unpack .../python-linaro-image-tools_2012.12-0ubuntu1~linaro1_all.deb ...
Unpacking python-linaro-image-tools (2012.12-0ubuntu1~linaro1) over (2012.12-0ubuntu1~linaro1) ...
Setting up debootstrap (1.0.87) ...
dpkg: dependency problems prevent configuration of linaro-image-tools:
 linaro-image-tools depends on python-debian (>= 0.1.16ubuntu1~); however:
  Package python-debian is not installed.
 linaro-image-tools depends on python-parted; however:
  Package python-parted is not installed.
 linaro-image-tools depends on python-yaml; however:
  Package python-yaml is not installed.

dpkg: error processing package linaro-image-tools (--install):
 dependency problems - leaving unconfigured
Setting up live-build (3.0.5-1linaro1) ...
dpkg: dependency problems prevent configuration of python-linaro-image-tools:
 python-linaro-image-tools depends on python-support (>= 0.90.0); however:
  Package python-support is not installed.

dpkg: error processing package python-linaro-image-tools (--install):
 dependency problems - leaving unconfigured
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Errors were encountered while processing:
 linaro-image-tools
 python-linaro-image-tools
dev@dev_152:~/LubanCat_SDK$ sudo apt-get install -f
Reading package lists... Done
Building dependency tree
Reading state information... Done
Correcting dependencies... Done
The following additional packages will be installed:
 python-chardet python-debian python-linaro-image-tools python-parted python-yaml
The following NEW packages will be installed:
```

注意: debian と ubuntu のルートファイルシステムの異なるバージョンのビルドに必要な依存パッケージのバージョンが異なります。ルートファイルシステムのバージョンを切り替えた後は、選択したルートファイルシステムのバージョンに応じて異なる依存パッケージをインストールする必要があります。

ubuntu のルートファイルシステムの SDK 設定ファイルを選択した場合、ubuntu のルートファイルシステムのビルドに必要な依存ソフトウェアパッケージをインストールします。

- 1 # ローカルソフトウェアパッケージをインストール
- 2 sudo dpkg -i ubuntu/ubuntu-build-service/packages/\*
- 3 sudo apt-get install -f

インストール中にエラーが発生することがありますが、これは通常現象です。パッケージのインストー

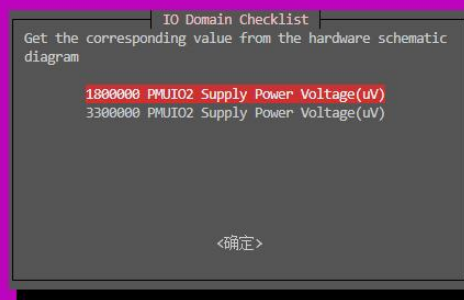
ルが完了したら、ワンクリックでビルドを開始できます。

1 # U-Boot、kernel、Rootfs を一括コンパイルして update.img イメージにパッケージ化

2 ./build.sh

```
dev@dev_152:~/LubanCat_SDK$ ./build.sh
processing option: allsave
=====
TARGET_ARCH=arm64
TARGET_PLATFORM=rk356x
TARGET_UBOOT_CONFIG=rk3566
TARGET_SPL_CONFIG=
TARGET_KERNEL_CONFIG=lubancat2_defconfig
TARGET_KERNEL_DTS=rk356x-lubancat-rk_series
TARGET_TOOLCHAIN_CONFIG=
TARGET_BUILDROOT_CONFIG=rockchip_rk3566
TARGET_RECOVERY_CONFIG=
TARGET_PCBA_CONFIG=
TARGET_RAMBOOT_CONFIG=
=====
=====Start building uboot=====
TARGET_UBOOT_CONFIG=rk3566
=====
grep: .config: No such file or directory
## make rk3568_defconfig rk3566.config -j176
HOSTCC scripts/basic/fixdep
HOSTCC scripts/kconfig/conf.o
```

警告: コンパイルプロセス中に以下のようなプロンプトが表示された場合、対応するデバイスツリーファイル内の&pmu\_io\_domains ノードの電圧値を選択し、pmuio2-supply から順に注意してください。ここで vccio\_acodec は 3v3、vccio\_sd は 3v3 です。



```
IO Domain Checklist
Get the corresponding value from the hardware schematic
diagram

1800000 PMUI02 Supply Power Voltage(uV)
3300000 PMUI02 Supply Power Voltage(uV)

<確定>
```

ビルドされたイメージは rockdev/ディレクトリに保存されます。./build.sh save コマンドを使用して、IMAGE/ディレクトリにバックアップも可能です。

## 2.7.1 ルートファイルシステムイメージを使用して Debian システムイメージを構築

対応する debian 設定ファイルを選択し、設定ファイルをアクティブにします。

対応するバージョンの debian ルートファイルシステムイメージ圧縮パッケージを debian ディレクトリに解凍します。注意：linaro-(バージョン)-rootfs.img は debian ディレクトリ内にある必要があります。

ワンクリックで完全なシステムイメージを生成します。

```
1 # SDK 設定ファイルを選択
2 # LubanCat-rk3588 ボード xfce デスクトップ版
3 ./build.sh BoardConfig-LubanCat-RK3588-debian-xfce.mk
4 # LubanCat-rk3588 ボード lite 版
5 ./build.sh BoardConfig-LubanCat-RK3588-debian-lite.mk
6 # LubanCat-rk3588 ボード xfce デスクトップ版
7 ./build.sh BoardConfig-LubanCat-RK3588-debian-xfce.mk
8 # LubanCat-rk3588 ボード lite 版
9 ./build.sh BoardConfig-LubanCat-RK3588-debian-lite.mk
10
11 # ダウンロードした ROOT ファイルシステムイメージ圧縮パッケージを debian ディレクトリにコピーして解凍します
12 # ダウンロードしたイメージに応じて以下のコマンドを実行します
13 # xfce デスクトップバージョン
14 7z x rootfs-debian10-xfce-(日付).7z
15 # lite バージョン
```

```

16 7z x rootfs-debian10-lite-(日付).7z
17
18 # バージョンに応じて、解凍後のファイルがdebian ディレクトリにあることを確認します
19 # ファイル名は linaro-xfce-rootfs.img または linaro-lite-rootfs.img です
20
21 # 完全なシステムイメージを一括でビルドします
22 ./build.sh
  
```

```

● jiawen@dev120:~/RK356X_LINUX_SDK$ ./build.sh BoardConfig-LubanCat-RK3566-debian-xfce.mk
processing option: BoardConfig-LubanCat-RK3566-debian-xfce.mk
switching to board: /home/jiawen/RK356X_LINUX_SDK/device/rockchip/rk356x/BoardConfig-LubanCat-RK3566-debian-xfce.mk
● jiawen@dev120:~/RK356X_LINUX_SDK$ mv rootfs-debian10-xfce-20230419.7z debian/
● jiawen@dev120:~/RK356X_LINUX_SDK$ cd debian/
● jiawen@dev120:~/RK356X_LINUX_SDK/debian$ 7z x rootfs-debian10-xfce-20230419.7z

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,52 CPUs Intel(R) Xeon(R) CPU E
5-2696 v4 @ 2.20GHz (406F1),ASM,AES-NI)

Scanning the drive for archives:
1 file, 761363230 bytes (727 MiB)

Extracting archive: rootfs-debian10-xfce-20230419.7z
--
Path = rootfs-debian10-xfce-20230419.7z
Type = 7z
Physical Size = 761363230
Headers Size = 162
Method = LZMA2:24
Solid = -
Blocks = 1

Everything is Ok

Size:      2968518656
Compressed: 761363230
● jiawen@dev120:~/RK356X_LINUX_SDK/debian$ ls
binary          mk-image.sh      overlay-firmware  readme.md
debian          mk-rootfs.sh     packages           rootfs-debian10-xfce-20230419.7z
mk-base-debian.sh overlay          packages-patches  ubuntu-build-service
mk-buster-rootfs.sh overlay-debug    post-build.sh
● jiawen@dev120:~/RK356X_LINUX_SDK/debian$ ls debian/
linaro-xfce-rootfs.img
● jiawen@dev120:~/RK356X_LINUX_SDK/debian$ mv debian/linaro-xfce-rootfs.img ./
● jiawen@dev120:~/RK356X_LINUX_SDK/debian$ ls
binary          mk-buster-rootfs.sh overlay-debug    post-build.sh
debian          mk-image.sh        overlay-firmware  readme.md
linaro-xfce-rootfs.img mk-rootfs.sh      packages         rootfs-debian10-xfce-20230419.7z
mk-base-debian.sh overlay            packages-patches  ubuntu-build-service
● jiawen@dev120:~/RK356X_LINUX_SDK/debian$ cd ..
● jiawen@dev120:~/RK356X_LINUX_SDK$ ./build.sh
processing option: allsave
=====
TARGET_ARCH=arm64
TARGET_PLATFORM=rk356x
  
```

DebianROOT ファイルシステムイメージのダウンロードアドレス：

DebianROOT ファイルシステムイメージ圧縮パッケージの説明：

- rootfs-debian10-xfce-xxx.7z
- rootfs-debian10-xfce-full-xxx.7z
- rootfs-debian10-lite-xxx.7z
- debian10 : Debian システムとリリースバージョン
- lite : デスクトップなし、ターミナル版
- xfce : xfce スイートを使用したデスクトップ版
- xfce-full : xfce スイート+追加推奨パッケージを含むデスクトップ版
- xxx : リリース日
- 7z : 圧縮形式

## 2.7.2 ROOT ファイルシステムイメージを使用して Ubuntu システムイメージをビルド

以前の操作で選択した debian 設定ファイルを選択し、選択した設定ファイルを確認します。

ダウンロードしたバージョンに応じた ubuntuROOT ファイルシステムイメージ圧縮パッケージを ubuntu ディレクトリに解凍します。注意：ubuntu-(バージョン)-rootfs.img は ubuntu ディレクトリ内にある必要があります。

完全なシステムイメージを一括でビルドするためのコマンドを使用します。

```
1 # SDK 設定ファイルを選択
2 # LubanCat-rk3588 デスクトップ版 xfce
3 ./build.sh BoardConfig-LubanCat-RK3588-ubuntu-xfce.mk
4 # LubanCat-rk3588 ライト版
```



```
5 ./build.sh BoardConfig-LubanCat-RK3588-ubuntu-lite.mk
6 # LubanCat-rk3588 デスクトップ版 xfce
7 ./build.sh BoardConfig-LubanCat-RK3588-ubuntu-xfce.mk
8 # LubanCat-rk3588 ライト版
9 ./build.sh BoardConfig-LubanCat-RK3588-ubuntu-lite.mk
10
11 # ダウンロードした ROOT ファイルシステムイメージ圧縮パッケージを ubuntu ディレクトリにコピー
    して解凍
12 # 実際にダウンロードしたイメージに基づいて、以下のコマンドを実行します
13 # xfce デスクトップバージョン
14 7z x rootfs-ubuntu20.04-xfce-(日付).7z
15 # ライトバージョン
16 7z x rootfs-ubuntu20.04-lite-(日付).7z
17
18 # バージョンに応じて、解凍後のファイルが debian ディレクトリにあることを確認します。
19 # ファイル名は ubuntu-xfce-rootfs.img または ubuntu-lite-rootfs.img です。
20
21 # 完全なシステムイメージを一括でビルドします。
22 ./build.sh
```

```
● jiawen@dev120:~/RK356X_LINUX_SDK$ ./build.sh BoardConfig-LubanCat-RK3566-ubuntu-xfce.mk
processing option: BoardConfig-LubanCat-RK3566-ubuntu-xfce.mk
switching to board: /home/jiawen/RK356X_LINUX_SDK/device/rockchip/rk356x/BoardConfig-LubanCat-RK3566-ubuntu-xfce.mk
● jiawen@dev120:~/RK356X_LINUX_SDK$ mkdir ubuntu
● jiawen@dev120:~/RK356X_LINUX_SDK$ mv rootfs-ubuntu20.04-xfce-20230419.7z ubuntu/
● jiawen@dev120:~/RK356X_LINUX_SDK$ cd ubuntu
● jiawen@dev120:~/RK356X_LINUX_SDK/ubuntu$ ls
rootfs-ubuntu20.04-xfce-20230419.7z
● jiawen@dev120:~/RK356X_LINUX_SDK/ubuntu$ 7z x rootfs-ubuntu20.04-xfce-20230419.7z

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,52 CPUs Intel(R) Xeon(R) CPU E
5-2696 v4 @ 2.20GHz (406F1),ASM,AES-NI)

Scanning the drive for archives:
1 file, 799288338 bytes (763 MiB)

Extracting archive: rootfs-ubuntu20.04-xfce-20230419.7z
--
Path = rootfs-ubuntu20.04-xfce-20230419.7z
Type = 7z
Physical Size = 799288338
Headers Size = 162
Method = LZMA2:24
Solid = -
Blocks = 1

Everything is Ok

Size:          3690242048
Compressed:    799288338
● jiawen@dev120:~/RK356X_LINUX_SDK/ubuntu$ ls
rootfs-ubuntu20.04-xfce-20230419.7z  ubuntu
● jiawen@dev120:~/RK356X_LINUX_SDK/ubuntu$ ls ubuntu/
ubuntu-xfce-rootfs.img
● jiawen@dev120:~/RK356X_LINUX_SDK/ubuntu$ mv ubuntu/ubuntu-xfce-rootfs.img ./
● jiawen@dev120:~/RK356X_LINUX_SDK/ubuntu$ ls
rootfs-ubuntu20.04-xfce-20230419.7z  ubuntu-xfce-rootfs.img
● jiawen@dev120:~/RK356X_LINUX_SDK/ubuntu$ cd ..
● jiawen@dev120:~/RK356X_LINUX_SDK$ ./build.sh
processing option: allsave
=====
TARGET_ARCH=arm64
TARGET_PLATFORM=rk356x
TARGET_UBOOT_CONFIG=rk3566
TARGET_SPI_CONFIG=
```

UbuntuROOT ファイルシステムイメージのダウンロードアドレス：

UbuntuROOT ファイルシステムイメージの説明：

- rootfs-ubuntu20.04-xfce-xxx.7z

- rootfs-ubuntu20.04-xfce-full-xxx.7z

- rootfs-ubuntu20.04-lite-xxx.7z

- ubuntu : Ubuntu システム

- 20.04: リリースバージョン番号、は 20.04 バージョンを提供し、18.04 と 22.04 のバージョンはソース

コードから自分でビルド可能

- lite : デスクトップなし、ターミナル版
- xfce : xfce スイートを使用したデスクトップ版
- xfce-full : xfce スイート+追加推奨パッケージを含むデスクトップ版
- xxx : リリース日
- 7z : 圧縮形式

## 2.8 LubanCat-SDK の個別モジュールビルド

ファームウェア開発中に全てを一度にビルドすると時間がかかりすぎます。変更がある度に全イメージを再ビルドしてパッケージングするのは大きな時間の浪費です。この時、SDK の個別モジュールビルド機能を使用することができます。

### 2.8.1 SDK 設定ファイルの選択

まず、SDK の設定ファイルを選択する必要があります。ここでは、LubanCat2 ボードの一般的な debian イメージを例にします。

```
1 # SDK 設定ファイルを選択
2 ./build.sh BoardConfig-LubanCat-RK3588-debian-xfce.mk
```

### 2.8.2 U-Boot のビルド

```
1 ./build.sh ubo
```

ビルドされた U-boot イメージは u-boot/uboot.img になります

### 2.8.3 Kernel のビルド

#### 2.8.3.1 extboot パーティション

extboot パーティション用のカーネルイメージは、BoardConfig-LubanCat-RK358x-xxx.mk の設定ファイル

を使用します。カーネル deb パッケージを最初に生成し、その後カーネルをビルドして extboot パーティションに deb パッケージをパッケージングします。

以下のコマンドで自動的に kernel のビルドとパッケージングが完了します。

```
1 ./build.sh kerneldeb
2
3 ./build.sh extboot
```

ビルドされた kernel イメージは kernel/extboot.img になります。

## 2.8.4 rootfs のビルド

LubanCat は主に Ubuntu、Debian、OpenWrt のいくつかの rootfs をサポートしています。異なる rootfs のビルドプロセスは異なりますので、ここで個別に説明します。

注意: OpenWrt システムのビルドは LubanCat-SDK を使用しないため、ここでは説明しません。具体的なビルドプロセスは LubanCatWRT ビルド説明書を参照してください。

### 2.8.4.1 Debian

ビルドする rootfs と SDK の設定ファイルが一致していることを確認する必要があります。現在の設定ファイルがビルドする rootfs と一致していない場合は、設定ファイルを切り替える必要があります。

```
1 # SDK 設定ファイルを選択
2 ./build.sh BoardConfig-LubanCat-RK3588-debian-xfce.mk
3
4 # Debian をビルド
5 ./build.sh debian
```

ビルドされた rootfs イメージは debian/linaro-xfce-rootfs.img になり、rockdev/rootfs.ext4 にシンボリックリンクされます。

注意: debian/linaro-xfce-rootfs.img が存在しない場合にのみ、Debian rootfs が再ビルドされます。

linaro-xfce-rootfs.img を再ビルドする場合は、手動で削除する必要があります。

ヒント: 異なるバージョンの rootfs をビルドする際に環境依存の問題に直面しやすいため、Debian rootfs を再ビルドする必要がないユーザーは、提供されたカスタマイズされた rootfs イメージを直接ダウンロードするか、Docker を使用してビルドすることをお勧めします。詳細は Debian rootfs ビルド専用セクションを参照してください。

## 2.8.4.2 Ubuntu

現在、Ubuntu18.04/20.04/22.04 の rootfs ビルドスクリプトを提供しており、デフォルトでは 20.04 のバージョンブランチを使用しています。他のバージョンをビルドする場合は、ビルド前にソースコードリポジトリをプルして対応するブランチに切り替える必要があります。具体的な操作は Ubuntu rootfs ビルド専用セクションを参照してください。

注意: Ubuntu20.04 バージョンを推奨します。

```
1 # SDK 設定ファイルを選択
2 ./build.sh BoardConfig-LubanCat-RK3588-ubuntu-xfce.mk
3
4 # Ubuntu をビルド
5 ./build.sh ubuntu
```

ビルドされた rootfs イメージは ubuntu/ubuntu-xfce-rootfs.img になり、rockdev/rootfs.ext4 にシンボリックリンクされます。

注意: ubuntu/ubuntu-xfce-rootfs.img が存在しない場合にのみ、Ubuntu rootfs が再ビルドされます。

ubuntu-xfce-rootfs.img を再ビルドする場合は、手動で削除する必要があります。

ヒント: 異なるバージョンの rootfs をビルドする際に環境依存の問題に直面しやすいため、Ubuntu rootfs

を再ビルドする必要がないユーザーは、提供されたカスタマイズされた rootfs イメージを直接ダウンロードするか、Docker を使用してビルドすることをお勧めします。詳細は Ubuntu rootfs ビルド専用セクションを参照してください。

## 2.8.5 ファームウェアのパッケージング

u-Boot、kernel、Rootfs がすべて構築された後、./mkfirmware.sh を実行してファームウェアをパッケージングする必要があります。これは、パーティションテーブルファイルの存在をチェックし、各パーティションがパーティションテーブルの設定と一致しているかを確認し、設定ファイルに従ってすべてのファイルを rockdev/ディレクトリにコピーまたはリンクすることを主に行います。

イメージのリリースを容易にするために、各個別のパーティションを1つのファイルにパッケージングすることもできます。パッケージングされたファイルは書き込み用に使用できます。

```

1 # ファームウェアのパッケージング
2 ./mkfirmware.sh
3
4 # update.img の生成
5 ./build.sh updateimg
    
```

Debian システムを例に、rockdev/ディレクトリ下のファイルは以下のようになります。

rockdev/	リンクファイル
boot.img	kernel/boot.img
MiniLoaderAll.bin	u-boot/rk358x_spl_loader_v1.15.112.bin
parameter.txt	device/rockchip/rk358x/parameter-ubuntu-fit.txt
rootfs.ext4	debian/debian-xfce-rootfs.img
rootfs.img	debian/debian-xfce-rootfs.img

uboot.img	u-boot/uboot.img
-----------	------------------

## 2.9 SDK 設定ファイルの説明

./build.sh スクリプトが SDK の各部分の構築作業を行う「筋肉」であるなら、SDK の設定ファイルはそれらの筋肉の動きを制御する「脳」です。

SDK の設定ファイルには多くの項目がありますが、実際に変更する必要があるのはそれほど多くありません。

ここでは、LubanCat-RK の設定ファイルを例に、他のボードの設定ファイルも同様です。

```
1 # ターゲットアーキテクチャ
2 export RK_ARCH=arm64
3 # Uboot defconfig
4 export RK_UBOOT_DEFCONFIG=rk3588
5 # Uboot イメージフォーマットタイプ: fit(フラット化イメージツリー)
6 export RK_UBOOT_FORMAT_TYPE=fit
7 # カーネル defconfig
8 export RK_KERNEL_DEFCONFIG=lubancat2_defconfig
9 # カーネル defconfig フラグメント
10 export RK_KERNEL_DEFCONFIG_FRAGMENT=
11 # カーネル dts
12 export RK_KERNEL_DTS=rk358x-lubancat-rk_series
13 # ブートイメージタイプ
14 export RK_BOOT_IMG=boot.img
```

```
15 # カーネルイメージパス
16 export RK_KERNEL_IMG=kernel/arch/arm64/boot/Image
17 # カーネルイメージフォーマットタイプ: fit(フラット化イメージツリー)
18 export RK_KERNEL_FIT_ITS=boot.its
19 # GPT テーブル用パラメータ
20 export RK_PARAMETER=parameter-ubuntu-fit.txt
21 # パーティションテーブルに対応するパッケージファイル
22 export RK_PACKAGE_FILE=rk358x-package-file-ubuntu
23 # Buildroot 設定
24 export RK_CFG_BUILDROOT=
25 # リカバリ設定
26 export RK_CFG_RECOVERY=
27 # リカバリイメージフォーマットタイプ: fit(フラット化イメージツリー)
28 export RK_RECOVERY_FIT_ITS=boot4recovery.its
29 # ラムブート設定
30 export RK_CFG_RAMBOOT=
31 # Pcba 設定
32 export RK_CFG_PCBA=
33 # ビルドジョブ
34 export RK_JOBS=24
35 # ターゲットチップ
36 export RK_TARGET_PRODUCT=rk358x
```



```
37 # rootfs タイプを設定、ext2 ext4 squashfs を含む
38 export RK_ROOTFS_TYPE=ext4
39 # yocto マシン
40 export RK_YOCTO_MACHINE=rockchip-rk3588-evb
41 # rootfs イメージパス
42 export RK_ROOTFS_IMG=rockdev/rootfs.${RK_ROOTFS_TYPE}
43 # ラムブートイメージタイプを設定
44 export RK_RAMBOOT_TYPE=
45 # OEM パーティションタイプを設定、ext2 squashfs を含む
46 export RK_OEM_FS_TYPE=ext2
47 # ユーザーデータパーティションタイプを設定、ext2, fat を含む
48 export RK_USERDATA_FS_TYPE=ext2
49 #OEM 設定
50 export RK_OEM_DIR=
51 # Buildroot での OEMビルド
52 #export RK_OEM_BUILDIN_BUILDROOT=YES
53 #ユーザーデータ設定
54 export RK_USERDATA_DIR=
55 #misc イメージ
56 export RK_MISC=
57 #distro モジュールを有効にするか選択
58 export RK_DISTRO_MODULE=
```

```
59 # このボード用のプレビルドスクリプトを定義
60 export RK_BOARD_PRE_BUILD_SCRIPT=app-build.sh
61
62 # SOC
63 export RK_SOC=rk358x
64 # build.sh save の際の名前
65 export RK_PKG_NAME=lubancat-${RK_UBOOT_DEFCONFIG}
66
67 # デフォルトの rootfs を debian に定義
68 export RK_ROOTFS_SYSTEM=debian
69 # debian バージョンを設定(debian10: buster)
70 export RK_DEBIAN_VERSION=10
71 # デフォルトの rootfs がデスクトップ版 xfce か、ライト版か、コンソール版か、デスクトップ版+推奨
ソフトウェアパッケージかを定義
72 export RK_ROOTFS_TARGET=xfce
73 # デフォルトの rootfs に DEBUG ツールを追加するかどうかを定義、debug: 追加、none:
```

- RK\_ARCH : プロセッサアーキテクチャ、変更不要
- RK\_UBOOT\_DEFCONFIG : Uboot の設定ファイル
- RK\_KERNEL\_DEFCONFIG : Kernel の設定ファイル
- RK\_KERNEL\_DEFCONFIG\_FRAGMENT : Kernel 設定ファイルの追加設定
- RK\_KERNEL\_DTS : カーネルのデバイスツリーファイル名
- RK\_BOOT\_IMG : boot パーティションイメージ名、変更不要
- RK\_KERNEL\_IMG : カーネルイメージの位置、変更不要

- RK\_PARAMETER : パーティションテーブル名、変更不要
- RK\_CFG\_BUILDROOT : Buildroot の設定ファイル、変更不要
- RK\_CFG\_RECOVERY : recovery パーティションの構築設定ファイル、変更不要
- RK\_JOBS : ビルドスレッド数、PC の構成に応じて調整可能
- RK\_TARGET\_PRODUCT : ターゲットチップ、変更不要
- RK\_ROOTFS\_TYPE : rootfs のフォーマット、変更不要
- RK\_ROOTFS\_IMG : rootfs イメージの位置、変更不要
- RK\_ROOTFS\_SYSTEM : rootfs のタイプ、現在 ubuntu、debian をサポート、変更不要
- RK\_DEBIAN\_VERSION : debian のリリースバージョン、変更不可
- RK\_UBUNTU\_VERSION : Ubuntu のリリースバージョン、変更不可
- RK\_ROOTFS\_TARGET: rootfs がデスクトップ表示をサポートしているかどうか、xfce: デスクトップ版、lite: コンソール版、xfce-full: デスクトップ版+推奨ソフトウェアパッケージ
- RK\_ROOTFS\_DEBUG: rootfs にデバッグツールを追加するかどうか、debug で追加、none で追加なし
- RK\_EXTBOOT: exboot カーネルパーティションを使用するかどうか

## 第3章 U-boot の紹介

### 3.1 U-boot について

U-boot は FADSROM、8xxROM、PPCBOOT から徐々に発展してきました。U-boot は現在、多くの機能を実現できるようになりました。オペレーティングシステムの面では、組み込み Linux システムのブートに限らず、NetBSD, VxWorks, QNX, RTEMS, ARTOS, LynxOS, Android などの組み込みオペレーティングシステムのブートもサポートしています。CPU アーキテクチャの面では、PowerPC、MIPS、x86、ARM、NIOS、XScale など、多くの一般的なのプロセッサをサポートしています。

一般的に、BootLoader はシステムの電源が入った時の初期化コードを提供する必要があります。システムの電源が入った後、関連する環境を初期化した後、BootLoader は完全なオペレーティングシステムをブートし、制御をオペレーティングシステムに渡す必要があります。簡単に言うと、BootLoader はシステムの電源が入った時に実行される小さなプログラムで、このプログラムを通じて CPU、SDRAM、Flash、シリアルポート、ネットワークなどのハードウェアデバイスを初期化し、初期化が完了した後にオペレーティングシステムのカーネルを呼び出します。

## 3.2 U-boot の起動

異なる LubanCat ボードは、同じバージョンの U-Boot ソースコードを使用して構築されていますが、わずかな設定が異なります。以下のドキュメントでは、RK3588 プロセッサを使用する魯班猫ボードを例に説明しますが、他のモデルのプロセッサを使用する魯班猫ボードの内容は基本的に同じであり、必要な場合のみ異なる点を指摘します。

次に、LubanCat ボードを例にして、RockChip が提供する U-boot v2017 の深いカスタマイズバージョン (next-dev と呼ばれる) の使用方法を紹介します。

ボードの Debug シリアルポートをコンピューターに接続し、電源を入れた後、U-boot が kernel を起動する前にキーボードの ctrl+c キーを押します: U-Boot コマンドラインモードに入ります。以下のように表示されます。

```
1 U-Boot 2017.09-gdde42ec6a2-211221 #jiawen (Jun 01 2022 - 14:07:06 +0800)
2
3 Model: Rockchip RK3588 Evaluation Board
4 PreSerial: 2, raw, 0xfe660000
5 DRAM: 4 GiB
6 Systemem: init
```

```
7 Relocation Offset: ed34b000
8 Relocation fdt: eb9f87e8 - eb9fecd0
9 CR: M/C/I
10 Using default environment
11
12 Hotkey: ctrl+c
13 dwmmc@fe2b0000: 1, dwmmc@fe2c0000: 2, sdhci@fe310000: 0
14 Bootdev(atags): mmc 0
15 MMC0: HS200, 200Mhz
16 PartType: EFI
17 DM: v1
18 boot mode: None
19 FIT: no signed, no conf required
20 DTB: rk-kernel.dtb
21 HASH(c): OK
22 I2c0 speed: 100000Hz
23 vsel-gpios- not found! Error: -2
24 vdd_cpu 1025000 uV
25 PMIC: RK8090 (on=0x40, off=0x00)
26 vdd_logic init 900000 uV
27 vdd_gpu init 900000 uV
28 vdd_npu init 900000 uV
```

```
29 io-domain: OK
30 Could not find baseparameter partition
31 Model: EmbedFire LubanCat2
32 Rockchip UBOOT DRM driver version: v1.0.1
33 VOP have 2 active VP
34 vp0 have layer nr:3[0 2 4 ], primary plane: 4
35 vp1 have layer nr:3[1 3 5 ], primary plane: 5
36 vp2 have layer nr:0[], primary plane: 0
37 Using display timing dts
38 dsi@fe060000: detailed mode clock 59603 kHz, flags[8000000a]
39 H: 0720 0730 0736 0756
40 V: 1280 1290 1294 1314
41 bus_format: 100e
42 VOP update mode to: 720x1280p0, type: MIPI0 for VP0
43 VOP VP0 enable Smart0[654x270->654x270@33x505] fmt[2] addr[0xedf04000]
44 final DSI-Link bandwidth: 396 Mbps x 4
45 xfer: num: 2, addr: 0x50
46 xfer: num: 2, addr: 0x50
47 Monitor has basic audio support
48 Could not find baseparameter partition
49 mode:1920x1080
50 hdmi@fe0a0000: detailed mode clock 148500 kHz, flags[5]
```

51 H: 1920 2008 2052 2200

52 V: 1080 1084 1089 1125

53 bus\_format: 2025

54 VOP update mode to: 1920x1080p0, type: HDMI0 for VP1

55 VOP VP1 enable Smart1[654x270->654x270@633x405] fmt[2] addr[0xedf04000]

56 CEA mode used vic=16

57 final pixclk = 148500000 tmdsclk = 148500000

58 PHY powered down in 0 iterations

59 PHY PLL locked 1 iterations

60 PHY powered down in 1 iterations

61 PHY PLL locked 1 iterations

62 sink has audio support

63 hdmi\_set\_clk\_regenerator: fs=48000Hz ftdms=148.500MHz N=6144 cts=148500

64 CLK: (sync kernel. arm: enter 816000 KHz, init 816000 KHz, kernel 0N/A)

65 apll 1416000 KHz

66 dppll 780000 KHz

67 gppll 1188000 KHz

68 cppll 1000000 KHz

69 nppll 1200000 KHz

70 vppll 742000 KHz

71 hppll 59000 KHz

72 pppll 200000 KHz

```
73 armclk 1416000 KHz
74 aclk_bus 150000 KHz
75 pclk_bus 100000 KHz
76 aclk_top_high 500000 KHz
77 aclk_top_low 400000 KHz
78 hclk_top 150000 KHz
79 pclk_top 100000 KHz
80 aclk_perimid 300000 KHz
81 hclk_perimid 150000 KHz
82 pclk_pmu 100000 KHz
83 Net: eth0: ethernet@fe2a0000, eth1: ethernet@fe010000
84 Hit key to stop autoboot('CTRL+C'): 0
85 => <INTERRUPT>
```

U-boot はボードの基本情報、CPU、メモリなどの情報を出力します。また、ボードの初期化プロセス中の一部のヒントメッセージ、例えば画面表示やクロック周波数なども含まれています。

### 3.3 U-boot のショートカットキー

U-Boot のコマンドラインモードに入るためのショートカットキーの使用に加えて、next-dev は他のシリアルポート組み合わせショートカットキーも提供しています。一般的に使用されるものは以下の通りです。

- ctrl+c : U-Boot のコマンドラインモードに入る ;
- ctrl+d : loader 書き込みモードに入る ;
- ctrl+b : maskrom 書き込みモードに入る ;

### 3.4 U-boot のコマンド

U-boot がどのようなコマンドをサポートしているか不明な場合、help または ? を入力して U-boot がサポー



トするコマンドリストを確認できます。以下のように示されている

- 1 ? - alias for 'help'
- 2 android\_print\_hdr- print android image header
- 3 atags - Dump all atags
- 4 base - print or set address offset
- 5 bdfinfo - print Board Info structure
- 6 bidram\_dump- Dump bidram layout
- 7 boot - boot default, i.e., run 'bootcmd'
- 8 boot\_android- Execute the Android Bootloader flow.
- 9 boot\_fit- Boot FIT Image from memory or boot/recovery partition
- 10 bootavb - Execute the Android avb a/b boot flow.
- 11 bootd - boot default, i.e., run 'bootcmd'
- 12 booti - boot arm64 Linux Image image from memory
- 13 bootm - boot application image from memory
- 14 bootp - boot image via network using BOOTP/TFTP protocol
- 15 bootrpk - Boot Linux Image from rockchip image type
- 16 bootz - boot Linux zImage image from memory
- 17 cmp - memory compare
- 18 coninfo - print console devices and information
- 19 cp - memory copy

- 20 crc32 - checksum calculation
- 21 crypto\_sum- crypto checksum engine

22 dhcp - boot image via network using DHCP/TFTP protocol

23 dm - Driver model low level access

24 download- enter rockusb/bootrom download mode

25 dtimg - manipulate dtb/dtbo Android image

26 dump\_irqs- Dump IRQs

27 dump\_resource- dump resource list

28 echo - echo args to console

29 editenv - edit environment variable

30 env - environment handling commands

31 exit - exit script

32 ext2load- load binary file from a Ext2 filesystem

33 ext2ls - list files in a directory (default /)

34 ext4load- load binary file from a Ext4 filesystem

35 ext4ls - list files in a directory (default /)

36 ext4size- determine a file's size

37 false - do nothing, unsuccessfully

38 fastboot- use USB or UDP Fastboot protocol

39 fatinfo - print information about filesystem

40 fatload - load binary file from a dos filesystem

41 fatls - list files in a directory (default /)

42 fatsize - determine a file's size

43 fatwrite- write file into a dos filesystem

44 fdt - flattened device tree utility commands

45 fstype - Look up a filesystem type

46 go - start application at address 'addr'

47 gpt - GUID Partition Table

48 help - print command description/usage

49 iomem - Show iomem data by device compatible(high priority) or node name

50 lcdputs - print string on video framebuffer

51 load - load binary file from a filesystem

52 loop - infinite loop on address range

53 ls - list files in a directory (default /)

54 md - memory display

55 mdio - MDIO utility commands

56 mii - MII utility commands

57 mm - memory modify (auto-incrementing address)

58 mmc - MMC sub system

59 mmcinfo - display MMC info

60 mtd - MTD utils

61 mtd\_blk - MTD Block device sub-system

62 mw - memory write (fill)

63 nand - NAND sub-system

64 nboot - boot from NAND device

65 nfs - boot image via network using NFS protocol

- 66 nm - memory modify (constant address)
- 67 part - disk partition related commands
- 68 ping - send ICMP ECHO\_REQUEST to network host
- 69 printenv- print environment variables
- 70 pxe - commands to get and boot from pxe files
- 71 rbrom - Perform RESET of the CPU
- 72 reboot - Perform RESET of the CPU, alias of 'reset'
- 73 reset - Perform RESET of the CPU
- 74 rkingtest- Test if storage media have rockchip image
- 75 rockchip\_show\_bmp- load and display bmp from resource partition
- 76 rockchip\_show\_logo- load and display log from resource partition
- 77 rockusb - Use the rockusb Protocol
- 78 run - run commands in an environment variable
- 79 save - save file to a filesystem
- 80 saveenv - save environment variables to persistent storage
- 81 setcurs - set cursor position within screen
- 82 setenv - set environment variables
- 83 showvar - print local hushshell variables
- 84 size - determine a file's size
- 85 source - run script from memory

86 sysboot - command to get and boot from syslinux files

87 systemem\_dump- Dump systemem layout

- 88 systemem\_search- Search a available systemem region
- 89 test - minimal test like /bin/sh
- 90 tftp - download image via network using TFTP protocol
- 91 tftpbootm- tftpbootm aosp/uImage/FIT image via network using TFTP protocol
- 92 tftpflash- flash image via network using TFTP protocol
- 93 tftpput - TFTP put command, for uploading files to a server
- 94 true - do nothing, successfully
- 95 ums - Use the UMS [USB Mass Storage]
- 96 usb - USB sub-system
- 97 usbboot - boot from USB device
- 98 version - print monitor, compiler and linker version

U-boot は多くのコマンドをサポートしており、機能は非常に強力で、Linux に似ています。特定の U-boot コマンドを実行する際には、tab を使用してコマンドを自動補完することができます。コマンド名の衝突がない場合は、コマンドの最初の数文字をコマンドの入力として使用することができます。例えば、reset コマンドを実行したい場合、res または re と入力するだけです。

注意：デフォルトでは、saveenv コマンドを使用して環境変数を保存することはサポートされていません。

これは、既存のデフォルトの env 設定を上書きしてシステムが起動できなくなる可能性があるためです。環境変数を変更する場合は、U-boot のソースコードを変更してください。

特定のコマンドの使用方法が必要な場合は、「help コマンド」または「? コマンド」の形式で特定のコマンドの使用説明を確認できます。「help printenv」を例にする

```
1 => help printenv
```

```
2 printenv - print environment variables
```

3

4 Usage:

5 printenv [-a]

6 - print [all] values of all environment variables

7 printenv name ...

8 - print value of environment variable 'name'

printenv コマンドの説明と使用方法を確認できます。

U-boot コマンドの使用については、U-boot の公式リンク

[<http://www.denx.de/wiki/DULG/Manual>](<http://www.denx.de/wiki/DULG/Manual>) の 5.9. uboot

Command Line Interface セクションを参照してください。

### 3.4.1 U-boot の一般的なコマンド

U-boot には多くのコマンドがあります。以下に、一般的に使用される U-boot のコマンドを紹介します。

コマンドの詳細な使用方法については、help [コマンド]で確認してください。

表 1: 一般的なコマンド

コマンド	説明	例え
help	現在の U-boot がサポートするすべてのコマンドをリストアップします。	
help [コマンド]	指定したコマンドのヘルプを表示します	help printenv
reset	U-boot を再起動します。	
printenv	すべての環境変数の値を表示します。	
printenv [変数名]	指定した環境変数の値を表示します。	printenv bootdelay
setenv	環境変数の値を設定/変更/削除します。	setenv bootdelay 3

saveenv	環境変数を保存します。	
ping	ネットワークが接続されているかどうかをチェックします。	ping 192.168.0.1
md	メモリアドレス上の値を表示します。	mw.b 0x80000000 10 コマンドは、アドレス 0x80000000 から始まる 10 個のデータをバイト値を表示する
mw	コマンドはメモリアドレス上の値を変更するために使用されます。	`mw.b 0x80000000 ff 10` コマンドは、アドレス 0x80000000 から始まる 10 個のデータをバイト値 `ff` に変更します。
echo	コマンドは情報を表示します。これは Linux の echo コマンドに似ています。	
run	コマンドは環境変数コマンドを実行します。	run bootcmd は bootcmd を実行します。
bootz	コマンドはメモリからカーネルをブートします。	
ls	コマンドはファイルシステム内のディレクトリ下のファイルを表示します。	
load	コマンドはファイルシステムからバイナリファイルをメモリにロードします。	

これらはユーザーがよく使用するコマンドの一部です。具体的な使用方法は `help [コマンド]` で確認できます。

### 3.4.2 mmc コマンド

mmc コマンドは、SD カードや eMMC などのストレージメディアを操作するために使用されます。以下は、mmc コマンドの基本的な機能を簡単に説明します。mmc コマンドに慣れていない場合は、`help mmc` で関連するコマンドのヘルプを表示できます。

表 2: mmc コマンドの機能

コマンド	説明
mmc list	ボード上の mmc デバイスを表示
mmc dev	現在のデフォルト mmc デバイスを表示/切替
mmc info	現在の mmc デバイス情報を表示
mmc part	現在の mmc デバイスのパーティションを表示
mmc read	現在の mmc デバイスからデータを読み取る
mmc write	現在の mmc デバイスにデータを書き込む
mmc erase	現在の mmc デバイスのデータを消去

#### 3.4.2.1 mmc デバイスの表示

`mmc list` を使用してボード関連のデバイスを表示します。例として、ボードに搭載された eMMC を見ると、以下のような情報が表示されます。

```
1 dwmmc@fe2b0000: 1
2 dwmmc@fe2c0000: 2
```



3 sdhci@fe310000: 0 (eMMC)

mmc dev index を使用して現在の mmc デバイスを切り替えます。ここで、index はデバイス番号で、mmc list の結果から eMMC のデバイス番号が 0 であることがわかります。

```
1 => mmc dev 0
2 switch to partitions #0, OK
3 mmc0(part 0) is current device
```

mmc info を使用して現在の mmc デバイスの情報を表示します。

```
1 => mmc info
2 Device: sdhci@fe310000
3 Manufacturer ID: e9
4 OEM: 10f
5 Name: K93SK
6 Timing Interface: HS200
7 Tran Speed: 200000000
8 Rd Block Len: 512
9 MMC version 5.1
10 High Capacity: Yes
11 Capacity: 7.3 GiB
12 Bus Width: 8-bit
13 Erase Group Size: 512 KiB
```

```
14 HC WP Group Size: 8 MiB
15 User Capacity: 7.3 GiB
```

16 Boot Capacity: 4 MiB ENH

17 RPMB Capacity: 4 MiB ENH

### 3.4.2.2 パーティション情報の表示

mmc part を使用して現在の mmc デバイスのパーティションをリストアップします。

```
1 => mmc part
2
3 Partition Map for MMC device 0 -- Partition Type: EFI
4
5 Part Start LBA End LBA Name
6 Attributes
7 Type GUID
8 Partition GUID
9 1 0x00004000 0x00005fff "uboot"
10 attrs: 0x0000000000000000
11 type: 030f0000-0000-440a-8000-5a0500003c68
12 guid: 54040000-0000-4235-8000-6d5c00004fb3
13 2 0x00006000 0x000045fff "boot"
14 attrs: 0x0000000000000004
15 type: b0440000-0000-4d43-8000-788400007a88
16 guid: 346f0000-0000-4136-8000-17bb00007f36
```

```
17 3 0x00046000 0x00e9ffbf "rootfs"
```

```
18 attrs: 0x0000000000000000
```

19 type: 3d410000-0000-4b71-8000-4d7e00004948

20 guid: 614e0000-0000-4b53-8000-1d28000054a9

### 3.4.3 ファイルシステム操作コマンド

U-boot は、ext2/3/4 および fat ファイルシステムデバイスにアクセスすることができます。fstype コマンドを使用して、ストレージメディアのパーティションがどのタイプのファイルシステムを使用しているかを判断することができます。例として、mmc メディアの後ろ二つのパーティションのファイルシステムタイプを判断します。

```
1 => fstype mmc 0:2
2 ext4
3 => fstype mmc 0:3
4 ext4
```

ファイルシステムが ext4 の第三パーティションは rootfs ルートファイルシステムに対応します。

ファイルシステムのタイプが分かれば、対応するコマンドを使用してパーティションの内容を操作することができます。

#### 3.4.3.1 ext4 フォーマットファイルシステム

ext4 ファイルシステムのコマンド使用方法は FAT と似ており、コマンド名だけが異なります。U-boot が提供する ext ファイルシステムコマンドは以下の通りです。

表 3: ext4 フォーマットファイルシステムコマンド

コマンド	説明
ext4ls	ストレージデバイスの ext4 パーティション内の内容を表示します。
ext4load	ext4 パーティションからファイルを指定のメモリアドレスに読み出します。

ext4write	メモリ上のデータを ext4 パーティションのファイルに書き込みます。
-----------	-------------------------------------

### 3.4.3.1.1 ext4 ファイルシステム操作

以下は、`/etc/apt/sources.list` の内容をメモリに読み出す例を使って、U-boot での ext4 ファイルシステムの操作方法を簡単に説明します。

1. `/etc/apt` ディレクトリ内のファイル内容を表示します。

```
1 => ext4ls mmc 0:3 /etc/apt/  
2 <DIR> 4096 .  
3 <DIR> 4096 ..  
4 464 sources.list  
5 <DIR> 4096 apt.conf.d  
6 <DIR> 4096 auth.conf.d  
7 <DIR> 4096 preferences.d  
8 <DIR> 4096 sources.list.d  
9 <DIR> 4096 trusted.gpg.d
```

2. `/etc/apt/sources.list` ファイルをメモリアドレス `0x80000000` に読み出します。

```
1 => ext4load mmc 0:3 0x80000000 /etc/apt/sources.list  
2 464 bytes read in 701 ms (0 Bytes/s)
```

3. メモリ `0x80000000` の部分データを表示します。

```
1 => md.b 0x80000000 0x80  
2 80000000: 64 65 62 20 68 74 74 70 3a 2f 2f 6d 69 72 72 6f deb http://mirro
```

3 80000010: 72 73 2e 75 73 74 63 2e 65 64 75 2e 63 6e 2f 64 rs.ustc.edu.cn/d

4 80000020: 65 62 69 61 6e 20 62 75 73 74 65 72 20 6d 61 69 ebian buster mai

5 80000030: 6e 20 63 6f 6e 74 72 69 62 20 6e 6f 6e 2d 66 72 n contrib non-fr

6 80000040: 65 65 0a 64 65 62 2d 73 72 63 20 68 74 74 70 3a ee.deb-src http:

7 80000050: 2f 2f 6d 69 72 72 6f 72 73 2e 75 73 74 63 2e 65 //mirrors.ustc.e

8 80000060: 64 75 2e 63 6e 2f 64 65 62 69 61 6e 20 62 75 73 du.cn/Debian bus

9 80000070: 74 65 72 20 6d 61 69 6e 20 63 6f 6e 74 72 69 62 ter main contrib

### 3.5 U-boot でのカーネル起動プロセス

bootcmd と bootargs は U-boot で最も重要な二つの環境変数です。U-boot の実行が完了した後、何も操作しなければ、bootcmd 環境変数に記載された内容が自動的に実行されます。bootargs はカーネルに渡される起動パラメータです。

printenv bootcmd を使用して bootcmd の内容を確認できます。

```
1 => printenv bootcmd
2 bootcmd=boot_android ${devtype} ${devnum};boot_fit;bootrkp;run distro_bootcmd;
```

Debian や Buildroot イメージを書き込んだ場合は、bootrkp が実行され、rk が定義した起動プロセスを通じてカーネルが起動します。詳細は RK の公式ドキュメント「U-Boot v2017(next-dev) 開発ガイド」の 2.4 起動プロセスを参照してください。

Ubuntu イメージを書き込んだ場合は、bootcmd は最終的に distro\_bootcmd を実行します。printenv distro\_bootcmd を使用して distro\_bootcmd の内容を確認できます。

```
1 => printenv distro_bootcmd
2 distro_bootcmd=for target in ${boot_targets}; do run bootcmd_${target}; done
```

3

```
4 => printenv boot_targets
```

```
5 boot_targets=mmc1 mmc0 usb0 pxe dhcp
```

すなわち distro\_bootcmd は bootcmd\_mmc1、bootcmd\_mmc0、bootcmd\_usb0、bootcmd\_pxe、bootcmd\_dhcp の五つの環境変数を実行します。

前述では、mmc1 が SD カードのストレージデバイスを表し、mmc0 が eMMC デバイスを表すことが分かります。つまり、SD カードがシステムを装備していて、そのカードが基板に挿入されている場合、システムは SD カードから優先して起動します。

後の3つの起動パラメータについて、現在のシステムはサポートしていません。

```
1 => printenv bootcmd_mmc0
2 bootcmd_mmc0=setenv devnum 0; run mmc_boot
3 => printenv bootcmd_mmc1
4 bootcmd_mmc1=setenv devnum 1; run mmc_boot
```

bootcmd\_mmc0 と bootcmd\_mmc1 はそれぞれの devnum 環境パラメータの値を設定し、最後に mmc\_boot 環境パラメータを実行します。mmc\_boot の内容は以下の通りです。

```
1 => printenv mmc_boot
2 mmc_boot=if mmc dev ${devnum}; then setenv devtype mmc; run scan_dev_for_boot_part; fi
```

mmc\_boot では devtype 環境パラメータの値を設定し、最後に scan\_dev\_for\_boot\_part 環境パラメータを実行します。これは、起動フラグがある区域分けてをデバイスでスキャンするためのものです。

```
1 => printenv scan_dev_for_boot_part
2 scan_dev_for_boot_part=part list ${devtype} ${devnum} -bootable devplist; env exists devplist || setenv
```

```
devplist 1; for distro_bootpart in ${devplist}; do if fstype ${devtype} ${devnum}:${distro_bootpart}
bootfstype; then run scan_dev_for_boot; fi; done
```

注：U-boot から直接 printenv を使用して確認すると、フォーマットが非常に乱れるため、U-boot のソースコード内で確認することをお勧めします。

- include/configs/rk3588\_common.h
- include/configs/rockchip-common.h
- include/config\_distro\_bootcmd.h

ソースファイル config\_distro\_bootcmd.h で、scan\_dev\_for\_boot\_part の定義を見ることができます。

```
1 "scan_dev_for_boot_part=" ¥
2 "part list ${devtype} ${devnum} -bootable devplist; " ¥
3 "env exists devplist || setenv devplist 1; " ¥
4 "for distro_bootpart in ${devplist}; do " ¥
5 "if fstype ${devtype} " ¥
6 "${devnum}:${distro_bootpart} " ¥
7 "bootfstype; then " ¥
8 "run scan_dev_for_boot; " ¥
9 "fi; " ¥
10 "done¥0" ¥
```

上述のスタリプトは、まず-bootable フラグがある区域分けてが存在するかどうかを判断します。存在する場合は、区域分けて番号を記録し、次に scan\_dev\_for\_boot を実行して、起動に必要なファイルが boot 区域分けてに存在するかどうかをスキャンします。

```
1 "scan_dev_for_boot=" ¥
2 "echo Scanning ${devtype} " ¥
```

```
3 "${devnum}:${distro_bootpart}...;" ¥
```

```
4 "for prefix in ${boot_prefixes}; do " ¥
```

```
5 "run scan_dev_for_scripts;" ¥
```

```
6 "run scan_dev_for_extlinux;" ¥
```

```
7 "done;" ¥
```

```
8 SCAN_DEV_FOR_EFI ¥
```

```
9 "¥0"
```

scan\_dev\_for\_boot の定義では、scan\_dev\_for\_scripts と scan\_dev\_for\_extlinux を順に実行します。起動方式の順序を変更したい場合は、この部分を変更できます。

scan\_dev\_for\_extlinux は、boot 区域分けてに extlinux.conf 設定ファイルが存在するかどうかをスキャンするためのものです。存在する場合、設定ファイルを読み取って起動します。設定ファイルには、カーネル、デバイスツリーなどが定義されており、さらに異なるバージョンのカーネルをグラフィカルに選択して起動することができます。興味があれば、さらに調査してみてください。

scan\_dev\_for\_scripts は、boot 区域分けてに boot.scr ファイルが存在するかどうかをスキャンします。存在する場合、boot.scr スクリプトに記述されている起動プロセスに従って起動します。

```
1 "scan_dev_for_scripts=" ¥
```

```
2 "for script in ${boot_scripts}; do " ¥
```

```
3 "if test -e ${devtype} " ¥
```

```
4 "${devnum}:${distro_bootpart} " ¥
```

```
5 "${prefix}${script}; then " ¥
```

```
6 "echo Found U-Boot script " ¥
```

```
7 "${prefix}${script};" ¥
```



```
8 "run boot_a_script; " ¥
9 "echo SCRIPT FAILED: continuing...;" ¥
10 "fi;" ¥
11 "done¥0" ¥
12
13 "boot_scripts=boot.scr.uimg boot.scr¥0" ¥
14
15 "boot_a_script=" ¥
16 "load ${devtype} ${devnum}:${distro_bootpart} " ¥
17 "${scriptaddr} ${prefix}${script};" ¥
18 "source ${scriptaddr}¥0" ¥
```

上記のスクリプトでは、まず `scan_dev_for_scripts` で boot パーティション内に「boot.scr.uimg」または「boot.scr」という名前の起動スクリプトが存在するかどうかを検出します。検出された場合は、「Found U-Boot script スクリプトパス」というメッセージを表示し、次に `boot_a_script` を実行します。検出されない場合は、処理を終了します。

LuBancat ボードの boot パーティションパッケージング時には、boot.scr のソースファイルがコンパイルされパッケージングされているため、ここでは `boot_a_script` が実行されます。

`boot_a_script` では、boot パーティション内の boot.scr ファイルをメモリに読み込み、source コマンドでこのスクリプトを実行します。

boot.scr のソースファイルは、SDK ディレクトリの `kernel/arch/arm64/boot/dts/rockchip/uEnv/boot.cmd` に保存されています。以下は boot.cmd の内容です。

```
1 echo [boot.cmd] run boot.cmd scripts ...;
2
```

```
3 if test -e ${devtype} ${devnum}:${distro_bootpart} /uEnv/uEnv.txt; then
4
5 echo [boot.cmd] load ${devtype} ${devnum}:${distro_bootpart} ${env_addr_r} /uEnv/uEnv.txt ...;
6 load ${devtype} ${devnum}:${distro_bootpart} ${env_addr_r} /uEnv/uEnv.txt;
7
8 echo [boot.cmd] Importing environment from ${devtype} ...
9 env import -t ${env_addr_r} 0x8000
10
11 setenv bootargs ${bootargs} root=/dev/mmcblk${devnum}p3 ${cmdline}
12 printenv bootargs
13
14 echo [boot.cmd] load ${devtype} ${devnum}:${distro_bootpart} ${ramdisk_addr_r}
/initrd-${uname_r} ...
15 load ${devtype} ${devnum}:${distro_bootpart} ${ramdisk_addr_r} /initrd-${uname_r}
16
17 echo [boot.cmd] loading ${devtype} ${devnum}:${distro_bootpart} ${kernel_addr_r}
/Image-${uname_r} ...
18 load ${devtype} ${devnum}:${distro_bootpart} ${kernel_addr_r} /Image-${uname_r}
19
```

```
20 echo [boot.cmd] loading default rk-kernel.dtb
21 load ${devtype} ${devnum}:${distro_bootpart} ${fdt_addr_r} /rk-kernel.dtb
22
```

```
23 fdt addr ${fdt_addr_r}

24 fdt set /chosen bootargs

25

26 echo [boot.cmd] dtoverlay from /uEnv/uEnv.txt

27 setenv dev_bootpart ${devnum}:${distro_bootpart}

28 dtfile ${fdt_addr_r} ${fdt_over_addr} /uEnv/uEnv.txt ${env_addr_r}

29

30 echo [boot.cmd] [${devtype} ${devnum}:${distro_bootpart}] ...

31 echo [boot.cmd] [booti] ...

32 booti ${kernel_addr_r} ${ramdisk_addr_r} ${fdt_addr_r}

33 fi

34

35 echo [boot.cmd] run boot.cmd scripts failed ...;

36

37 # Recompile with:

38 # mkimage -C none -A arm -T script -d /boot/boot.cmd /boot/boot.scr
```

1. boot.scr スクリプトが実行されたことを示すプロンプトメッセージを表示します。

2. boot パーティションに uEnv/uEnv.txt ファイルが存在するかどうかを判断し、存在する場合は、以下のスクリプトを実行します。

3. load コマンドで uEnv ファイルをメモリに読み込み、カーネル、デバイスツリープラグインなどのカスタム環境変数を保存します。

4. env import コマンドで読み込んだ環境変数をインポートします。

5. setenv コマンドで bootargs 環境変数を設定し、カーネル起動時にカーネルにパラメータを渡します。

具体的には、rootfs があるパーティションを設定し、uEnv.txt ファイル内の cmdline を bootargs に追加します。

6. printenv コマンドで bootargs 環境変数をコンソールに表示し、デバッグを容易にします。

7. boot パーティションから initrd イメージをメモリにロードします。

8. boot パーティションからカーネルイメージをメモリにロードします。

9. boot パーティションからメインデバイスツリー rk-kernel.dt をメモリにロードします。

10. fdt コマンドでロードされたメインデバイスツリーを読み取り、/chosen ノード内の bootargs 内容をクリアします。そうしないと、ステップ 5 で設定した bootargs が上書きされます。

11. dtfile コマンドで uEnv ファイル内で定義されたデバイスツリープラグインを読み取り、メインデバイスツリーに統合します。

12. booti コマンドを使用してカーネルを起動する

上記のプロセスを経ることで、uboot の作業領域から離れ、カーネル内に入ります。もし上記のプロセスが失敗した場合は、デバイスツリープラグインのロード失敗などの理由で起動できない問題を解決するために、scan\_dev\_for\_extlinux が続けて実行されます。

### 3.6 U-boot 環境パラメータの紹介

U-boot 内の環境パラメータは、U-boot のソースコードを変更することなく、カーネルの起動カウントダウン、IP アドレス、さまざまなパラメータをカーネルに渡すなどの変更を可能にします。

ボード上で printenv を使用すると、ボード上のすべての環境パラメータを確認でき、setenv を使用して環境パラメータを追加/変更/削除できます。具体的な説明は以下の通りです。

```
1 # 環境パラメータ名を abc として、値を 100 に設定
2 => setenv abc 100
3 => echo $abc
```

4 = 100

5

6 # 値を 200 に変更

7 => setenv abc 200

8 => echo \$abc

9 200

10

11 # abc 環境パラメータを削除

12 => setenv abc

13 => echo \$abc

デフォルトでは、setenv コマンドで環境パラメータを変更した後に再起動すると変更が消えてしまいます。電源を切っても保存したい場合は、saveenv を実行して環境パラメータを記憶装置に保存する必要があります。

警告: Uboot 中でこの操作を使用することは推奨されません。デフォルトの環境変数を上書きする可能性があります。

U-boot には、公式に定められた環境変数がいくつかあり、これらの環境変数は U-boot で特別な役割を果たします。以下のリンクで確認できます：<https://www.denx.de/wiki/view/DULG/UBootEnvVariables>

## 第4章 U-boot の変更とコンパイル

LubanCat-SDK を利用することで、U-boot のコンパイルを簡単に行うことができますが、時にはボードの実際の状況に応じて U-boot を変更し、いくつかのカスタマイズ機能を実現する必要があります。そのためには、U-boot について深く理解する必要があります。

以下の内容では、LubanCat-SDK が U-boot のワンクリックビルドをどのように実現しているか、ビルドパ

ラメータがどのファイルに関連しているかについて学びます。

異なる LubanCat ボードが同じバージョンの U-Boot ソースコードをビルドに使用していますが、わずかな設定の違いがあります。以下のドキュメントでは、RK3588 プロセッサを使用する LuBancat ボードを例に説明します。

## 4.1 U-boot の取得

rk358x プロセッサに対するメインライン版 U-boot のサポートは非常に限られているため、RK 版の U-Boot : next-dev を使用することにしました。RK 版の U-Boot は 2 つの部分に分かれています。一つは U-boot 自体のコードリポジトリ、つまり u-boot ディレクトリ内の内容です。もう一つはツールキットリポジトリで、RK がオープンソースで公開していないバイナリファイル、スクリプト、パッキングツールが含まれており、loader、trust、uboot ファームウェアをパッケージングするために使用され、rkbin ディレクトリ内にあります。

**注意: rkbin と U-Boot プロジェクトは同レベルのディレクトリ構造を維持する必要があります。**

LubanCat-SDK には既にこれら 2 つの部分が含まれていますが、公式からも入手することができます。

### 4.1.1 ソースコードのダウンロード

公式 GitHub :

<https://github.com/Caesar-github/u-boot-next-dev> ブランチ

<https://github.com/Caesar-github/rkbin-master> ブランチ

ソースコードリポジトリダウンロードリンク :

<https://github.com/LubanCat>

### 4.1.2 指定ブランチのダウンロード

通常、U-boot リポジトリは異なるブランチの U-boot をメンテナンスしています。リポジトリディレクトリに入り、コマンドで U-boot のブランチを確認および切り替えることができます。

```
1 git clone --branch=next-dev https://github.com/LubanCat/u-boot.git
2 git clone --branch=master https://github.com/LubanCat/rkbin.git
```

## 4.2 U-boot のコンパイル

LubanCat-SDK 内では、自動コンパイルスクリプトは主に build.sh に格納されており、これが SDK の主要な機能エントリポイントです。build.sh 内での U-boot のビルド関数は `function build\_uboot()` で、その具体的な内容は以下の通りです。

```
1 function build_uboot(){
2 check_config RK_UBOOT_DEFCONFIG || return 0
3 build_check_cross_compile
4 prebuild_uboot
5
6 echo "=====Start building uboot=====
7 echo "TARGET_UBOOT_CONFIG=$RK_UBOOT_DEFCONFIG"
8 echo "=====
9
10 if [ "$RK_RAMDISK_SECURITY_BOOTUP" = "true" ];then
11 if [ -n "$RK_CFG_RAMBOOT" ];then
12 build_ramboot
13 else
```

```
14 build_kernel
15 fi
16
17 if [ -n "$RK_CFG_RECOVERY" ]; then
```

```
18 build_recovery
19 fi
20 cp -f $TOP_DIR/rockdev/boot.img $TOP_DIR/u-boot/boot.img
21 cp -f $TOP_DIR/rockdev/recovery.img $TOP_DIR/u-boot/recovery.img || true
22 fi
23
24 cd u-boot
25 rm -f *_loader*.bin
26 if [ "$RK_LOADER_UPDATE_SPL" = "true" ]; then
27 rm -f *spl.bin
28 fi
29
30 if [ -n "$RK_UBOOT_DEFCONFIG_FRAGMENT" ]; then
31 if [ -f "configs/${RK_UBOOT_DEFCONFIG}_defconfig" ]; then
32 make ${RK_UBOOT_DEFCONFIG}_defconfig $RK_UBOOT_DEFCONFIG_FRAGMENT
33 else
34 make ${RK_UBOOT_DEFCONFIG}.config $RK_UBOOT_DEFCONFIG_FRAGMENT
35 fi
36 ./make.sh $UBOOT_COMPILE_COMMANDS
37 elif [ -d
"$TOP_DIR/prebuilts/gcc/linux-x86/arm/gcc-arm-10.3-2021.07-x86_64-arm-none-linux-gnueabihf" ];
then
```



```
38 ./make.sh $RK_UBOOT_DEFCONFIG ¥
39 $UBOOT_COMPILE_COMMANDS CROSS_COMPILE=$CROSS_COMPILE
40 elif [ -d
"$TOP_DIR/prebuilts/gcc/linux-x86/aarch64/gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu" ];
then
41 ./make.sh $RK_UBOOT_DEFCONFIG ¥
42 $UBOOT_COMPILE_COMMANDS CROSS_COMPILE=$CROSS_COMPILE
43 else
44 ./make.sh $RK_UBOOT_DEFCONFIG ¥
45 $UBOOT_COMPILE_COMMANDS
46 fi
47
48 if [ "$RK_IDBLOCK_UPDATE_SPL" = "true" ]; then
49 ./make.sh --idblock --spl
50 fi
51
52 if [ "$RK_RAMDISK_SECURITY_BOOTUP" = "true" ];then
53 ln -rsf $TOP_DIR/u-boot/boot.img $TOP_DIR/rockdev/
54 ln -rsf $TOP_DIR/u-boot/recovery.img $TOP_DIR/rockdev/ || true
55 fi
```

```
56
```

```
57 finish_build
```

58 }

- check\_config は、RK\_UBOOT\_DEFCONFIG という設定変数が存在するかをチェックします。この変数は、U-boot のコンパイルに使用される設定ファイルを定義しており

device/rockchip/rk358x/BoardConfig-\*.mk 内に定義されています。

- build\_check\_cross\_compile は、arm-none-linux-gnueabi または aarch64-none-linux-gnu の交差コンパイラチェーンが存在するかを判断しますが、使用していないのでこの部分は無視できます。

- prebuild\_uboot は、BoardConfig-\*.mk 内の定義に基づいてコンパイルパラメータを設定します。

- UBOOT\_COMPILE\_COMMANDS では、RK\_RAMDISK\_SECURITY\_BOOTUP の値が true かどうかを判断し、イメージに対してセキュアブートで署名認証を使用するかどうかを決定しますが、定義していないためスキップされます。

- cd を使用して u-boot ディレクトリに入り、以前に生成された miniloader ファイルをクリアし、uboot の前段階が SPL であるかどうかを判断します。もしそうなら、spl ファイルもクリアします。

- RK\_UBOOT\_DEFCONFIG\_FRAGMENT では、RK 版 U-boot が設定ファイルのオーバーレイ機能をサポートしているかどうかを判断します。もしそうなら、まず設定ファイルをマージしますが、これは使用していません。

- arm-none-linux-gnueabi または aarch64-none-linux-gnu を使用してコンパイルするかどうかを判断しますが、使用していないため無視できます。

- 上記の2つのコンパイラが見つからない場合、./make.sh にビルドパラメータを渡して U-boot のビルド操作を行います。

- RK\_IDBLOCK\_UPDATE\_SPL と RK\_RAMDISK\_SECURITY\_BOOTUP は定義されていないため、直接スキップされます。

- finish\_build は、コンパイル完了時の提示関数です。

## 4.3 U-boot の変更

一般に、新しいボードを適応するためには、U-boot 内で主に 3 つの部分の変更と追加が関わっています。それらは、ボードの初期化に関連する board ディレクトリ下の部分、対応するボードの設定ファイルがある configs ディレクトリ下、およびボードレベルの外部デバイスに関連するファイルがある arch ディレクトリ下です。

Rockchip U-boot が rk プロセッサに対して完全なサポートを提供しているおかげで、U-boot 段階では、使用するプロセッサ型番に基づいて Rockchip 公式の参照ボードの設定を直接選択することができます。例えば、rk358x プロセッサには evb\_rk3588 が、rk3588x には evb\_rk3588 が使用されます。

U-boot にデバイスツリープラグインの機能実装を追加し、boot\_scripts 方式での起動ガイドを使用して extboot パーティションの起動を実現しています。boot\_scripts 起動方式の起動コマンドスクリプトは、SDK ディレクトリの kernel/arch/arm64/boot/dts/rockchip/uEnv/boot.cmd にあり、boot パーティションのビルド時にコンパイルされ、extboot パーティションにパッケージされて boot.scr として名付けられ、システム起動前に U-boot によって読み込まれ、ロードされます。具体的なプロセスは前章「U-boot によるカーネル起動プロセス」を参照してください。

### 4.3.1 U-boot の設定ファイル

コンパイルに使用する設定ファイルは、BoardConfig-\*.mk の RK\_UBOOT\_DEFCONFIG 変数によって定義されています。LubanCat-RK ボードで使用される具体的なファイルは、u-boot/configs/rk3588\_defconfig と rk3588.config です。ここでは LubanCat-4 を例に、もし変更が必要な場合は menuconfig ツールを利用できます。まず U-boot のルートディレクトリに行き、次の操作を実行します:

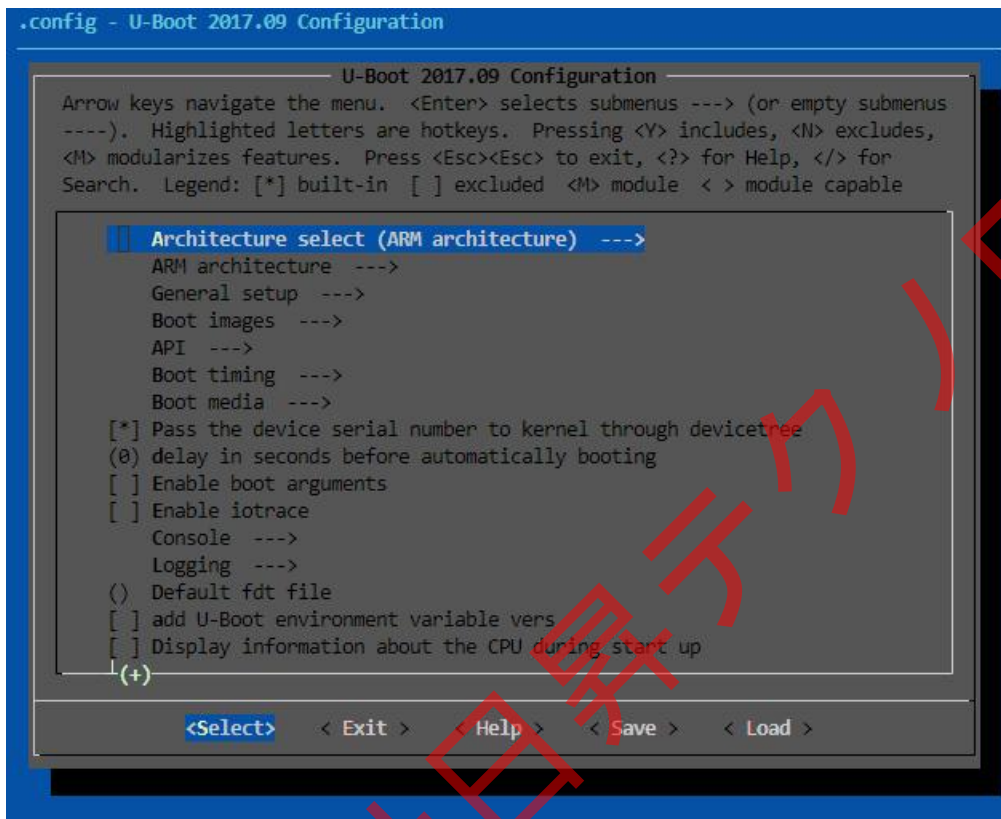
```
1 # 設定ファイルを適用
```

2 make rk3588\_defconfig

3

4 # menuconfig を使用して設定ファイルを管理・変更

5 make menuconfig



変更後、ESC キーを押して退出し、保存します。

1 # defconfig ファイルを保存

2 make savedefconfig

3

4 # 元の設定ファイルを上書き

5 cp defconfig configs/rk3588\_defconfig

### 4.3.2 デバイスツリーファイル

デバイスツリーファイルは、ボードレベルのデバイスの記述ファイルで、システムはデバイスツリーファ

イルを通じてボード上にどのような外部デバイスがあるかを知り、対応するドライバーをロードして外部デバイスを正常に動作させます。元の U-Boot は自身の DTB のみをサポートしていますが、RK プラットフォームは kernel DTB メカニズムのサポートを追加しました。つまり、kernel DTB を使用して外部デバイスを初期化します。主な目的は、電源、クロック、ディスプレイなどの外部デバイスのボードレベルの違いに対応することです。その役割は以下の通りです：

- U-Boot DTB: ストレージ、シリアルポートなどのデバイスの初期化を担当。
- Kernel DTB: ストレージ、シリアルポート以外のデバイスの初期化を担当。

U-Boot の初期化時には、まず U-Boot DTB を使用してストレージ、シリアルポートの初期化を行い、次にストレージから Kernel DTB をロードして、この DTB を使用して残りの外部デバイスの初期化を続けます。

開発者は通常、U-Boot DTB を変更する必要はありません（シリアルポートを交換する場合を除く）。RK のプロセッサで使用される defconfig は、すでに kernel DTB メカニズムを有効にしています。したがって、通常は外部デバイスの DTS を変更する場合、ユーザーは kernel DTB を変更するべきです。

U-Boot DTB について：

DTS ディレクトリ：

- u-boot/arch/arm/dts/rk3588-evb.dts
- u-boot/arch/arm/dts/rk3588-evb.dts

kernel DTB メカニズムを有効にした後、コンパイル段階では U-Boot DTS 内にある u-boot,dm-pre-reloc および u-boot,dm-spl 属性を持つノードをフィルタリングし、その上で defconfig 内の CONFIG\_OF\_SPL\_REMOVE\_PROPS で指定されたプロパティを除外します。最終的に u-boot.dtb ファイルを生成し、u-boot.bin の末尾に追加します。

## 4.4 参考資料について

RK U-boot には多くのカスタム機能が追加されていますが、その本体は公式の U-boot に基づいています。

したがって、U-boot の操作は基本的に同じです。以下の wiki を参照できます。

- 公式 U-boot ダウンロードリンク: <http://www.denx.de/wiki/uboot/WebHome>
- 公式 U-boot GitHub : <https://github.com/uboot/uboot>
- U-boot wiki: <http://www.denx.de/wiki/uboot/WebHome>
- RK U-boot のカスタム機能については、Rockchip のドキュメント「U-Boot v2017(next-dev)開発ガイド」を詳しく見ることができます。

## 第 5 章 Linux の紹介

### 5.1 Unix の簡単な紹介

Unix システムはベル研究所から始まり、ソースコードが提供されています。Unix ではほとんどすべてがファイルとして扱われます。

### 5.2 Linux の簡単な紹介

Linux はフィンランド人の Linus によって最初に開発された、Unix のようなシステムですが、Unix ではありません。Unix の API を実装しています（具体的な実装は Unix とは異なるかもしれません）。

そして Linux は GPLv2 のオープンソースライセンスを使用しています。

### 5.3 Linux の歴史

1991 年

8月25日: 21歳のフィンランドの学生 Linus Benedict Torvalds が comp.os.minix ニュースグループに、無料のオペレーティングシステムを開発中であると発表しました。

9月1日: Linux 0.01 がインターネット上で公開されました。

2007 年

6月6日: ASUS は 2007 年の台北コンピュータ展で 2 つの「Eee PC」、701 と 1001 を展示しました。最初の Eee PC は Xandros Linux をプリインストールしていました。これは Debian に基づいた、小さな画面に

最適化された軽量の Linux ディストリビューションです。

2007年8月8日: Linux 基金会在开源开发ラボ (OSDL) と自由標準グループ (FSG) によって共同で設立されました。この基金会的目的は、Linux の創設者である Linus の仕事を支援することです。基金会は、富士通、HP、IBM、Intel、NEC、Oracle、Qualcomm、Samsung を含む主要な Linux およびオープンソース企業、そして世界中の開発者からの支援を受けています。

2007年11月5日: 以前推測された Gphone のリリースとは異なり、Google は Open Handset Alliance の設立と Android の発表を行い、「最初の真にオープンな統合モバイルプラットフォーム」と称されました。

2011年

2011年5月11日: Google I/O 大会で Chrombook が発表されました。これは、いわゆるクラウドオペレーティングシステム Chrome OS を搭載したノート PC です。Chrome OS は Linux カーネルに基づいています。

2011年6月21日: Linus Torvalds が Linux 3.0 バージョンをリリースしました。

Linux の歴史に興味がある方は、オンラインで自分で Linux の歴史を検索してみてください。

## 5.4 モノリシックカーネルとマイクロカーネルの違い

モノリシックカーネル: すべてのカーネルが一つの大きなプロセスとして全体で実装され、カーネルアドレス空間で実行されます。カーネル通信は簡単です。カーネルは通常、ディスク上の単一の静的バイナリファイルとして存在します。シンプルで性能が高く、ほとんどの Unix はモノリシックカーネルです。

マイクロカーネル: マイクロカーネルの機能は複数のプロセスに分割され、各プロセスは個別のアドレス空間で実行されます。プロセス間通信 (IPC) を介してマイクロカーネル通信を処理する必要があります。強く要求されるプロセスのみがカーネルモードで実行され、他のプロセスはユーザーモードで実行されます。IPC のオーバーヘッドが大きく、ユーザーモードとカーネルモードのコンテキストスイッチが発生します。そのため、多くのマイクロカーネル実装 (Windows NT、OS X) では、すべてのマイクロカーネルプロセスをカーネルモードで実行します。

## 5.5 Linux カーネル

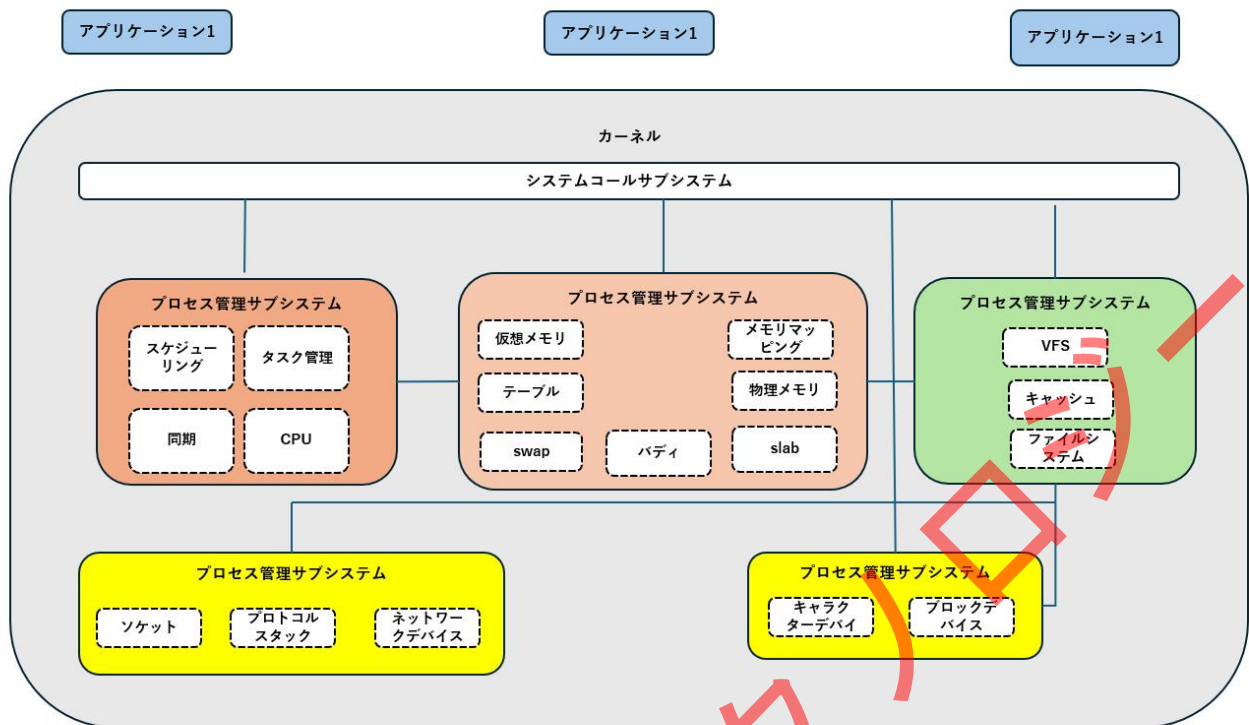
Linux はモノリシックカーネルですが、マイクロカーネルの精髓を取り入れています: モジュール化され、プリエンプティブカーネルを支持し、カーネルスレッドをサポートし、カーネルモジュールを動的にロードすることができます。

Kernel は Linux のカーネルです。Linux カーネルはマクロカーネルアーキテクチャを採用しており、プロセス管理、メモリ管理、デバイス管理、ファイル管理、ネットワーク管理などの機能のほとんどをカーネル内で実装しています。Linux の開発過程で、カーネルモジュール (Loadable Kernel Module、LKM) メカニズムが導入されました。カーネルモジュールは、動的にロード可能なカーネルモジュールのことで、カーネルの実行時に特定の機能を実現するための一連のターゲットコードを動的にロードすることができます。このプロセスでは、カーネルを再コンパイルすることなく動的な拡張が可能です。

## 5.6 Linux カーネルの構成

Linux カーネルは主に 5 つの部分から構成されています。それぞれがプロセス管理サブシステム、メモリ管理サブシステム、ファイルサブシステム、ネットワークサブシステム、デバイスサブシステムです。以下の通りです。





## 1. プロセス管理

プロセスの作成と破棄、プロセスのスケジューリングを担当します。

## 2. メモリ管理

メモリの割り当てと回収を担当し、どのメモリがどのプロセスに使用されているかを記録し、仮想メモリを管理し、メモリの物理アドレスと論理アドレスをマッピングします。主に MMU が変換を行い、ページテーブル方式を使用します。

## 3. ファイルシステム

ここでのファイルシステムは、単にハードディスクの抽象管理だけではなく、ある io ポートの抽象も指します。ファイルシステムは下層の詳細を隠蔽し、上層に統一されたインターフェースを提供します。Linux では、すべてがファイルとして扱われます。

## 4. デバイス管理

デバイス管理機能は主にドライバプログラムによって提供され、デバイスが入力または出力操作を完了するのを制御するのが主な任務です。Linux ではデバイスを特殊なファイルとみなし、システムはファイル

のインターフェース（仮想ファイルシステム VFS）を介して各種デバイスを管理および制御します。

## 5. ネットワーク機能

ネットワーク機能とは、ドライバプログラムが提供する基本的なハードウェア操作に加えて、システムが提供するメカニズムや機能、たとえば TCP プロトコルやアドレス解析などを指します。

## 5.7 Linux 公式サイト

公式サイトは <https://www.kernel.org/> です。



The Linux Kernel Archives

About Contact us FAQ Releases Signatures Site news

Protocol	Location
HTTP	<a href="https://www.kernel.org/pub/">https://www.kernel.org/pub/</a>
GIT	<a href="https://git.kernel.org/">https://git.kernel.org/</a>
RSYNC	<a href="rsync://rsync.kernel.org/pub/">rsync://rsync.kernel.org/pub/</a>

Latest Release  
**6.2.11** ↓

mainline:	6.3-rc7	2023-04-16	[tarball]	[patch]	[inc. patch]	[view diff]	[browse]		
stable:	6.2.11	2023-04-13	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	6.1.24	2023-04-13	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	5.15.107	2023-04-13	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	5.10.177	2023-04-05	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	5.4.240	2023-04-05	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	4.19.280	2023-04-05	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	4.14.312	2023-04-05	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
linux-next:	next-20230419	2023-04-19					[browse]		

Other resources

- Git Trees
- Patchwork
- Mirrors
- Documentation Wikis
- Linux.com
- Kernel Mailing Lists
- Bugzilla
- Linux Foundation

Social

- Site Atom feed
- Releases Atom Feed
- Kernel Planet

This site is operated by the Linux Kernel Organization, Inc., a 501(c)3 nonprofit corporation, with support from the following sponsors.

EQUINIX METAL fastly CONSTELLIX

ここでは最新バージョンのカーネルファイルをダウンロードできますが、ほとんどは対応するチップに適応されていません。チップの適応はチップメーカーが行うことであり、特定のチップに適応することは一般的に行いませんし、その能力もありません。組み込み開発者の主な責務は、カーネルの設定と使用に關することであり、特定のチップに適応するために多大な労力を費やすべきではありません。これは初心者にとって非常に挫折を感じさせるため、本書では主にカーネルの設定と使用について説明します。現在、LubanCat-2 と LubanCat-1 ボードは Linux Kernel に追加され、mainline 6.3-rc1 からサポートされています。関連情報は

<https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/arch/arm64/boot/dts/rockchip> または <https://github.com/torvalds/linux/tree/master/arch/arm64/boot/dts/rockchip> で確認できます。

-rw-r--r--	rk3566-anbernic-rg503.dts	4996	log	stats	plain
-rw-r--r--	rk3566-anbernic-rgxx3.dtsi	16446	log	stats	plain
-rw-r--r--	rk3566-box-demo.dts	10666	log	stats	plain
-rw-r--r--	<u>rk3566-lubancat-1.dts</u>	12107	log	stats	plain
-rw-r--r--	rk3566-pinenote-v1.1.dts	302	log	stats	plain
-rw-r--r--	rk3566-pinenote-v1.2.dts	299	log	stats	plain
-rw-r--r--	rk3566-pinenote.dtsi	15292	log	stats	plain
-rw-r--r--	rk3566-quartz64-a.dts	17700	log	stats	plain
-rw-r--r--	rk3566-quartz64-b.dts	15489	log	stats	plain
-rw-r--r--	rk3566-radxa-cm3-io.dts	5684	log	stats	plain
-rw-r--r--	rk3566-radxa-cm3.dtsi	9392	log	stats	plain
-rw-r--r--	rk3566-roc-pc.dts	14730	log	stats	plain
-rw-r--r--	rk3566-soquartz-blade.dts	3958	log	stats	plain
-rw-r--r--	rk3566-soquartz-cm4.dts	3583	log	stats	plain
-rw-r--r--	rk3566-soquartz-model-a.dts	4771	log	stats	plain
-rw-r--r--	rk3566-soquartz.dtsi	14827	log	stats	plain
-rw-r--r--	rk3566.dtsi	585	log	stats	plain
-rw-r--r--	rk3568-bpi-r2-pro.dts	17481	log	stats	plain
-rw-r--r--	rk3568-evb1-v10.dts	14288	log	stats	plain
-rw-r--r--	<u>rk3568-lubancat-2.dts</u>	14856	log	stats	plain
-rw-r--r--	rk3568-odroid-m1.dts	15270	log	stats	plain
-rw-r--r--	rk3568-pinctrl.dtsi	64535	log	stats	plain
-rw-r--r--	rk3568-radxa-cm3i.dtsi	8533	log	stats	plain
-rw-r--r--	rk3568-radxa-e25.dts	4741	log	stats	plain
-rw-r--r--	rk3568-rock-3a.dts	18080	log	stats	plain
-rw-r--r--	rk3568.dtsi	7602	log	stats	plain
-rw-r--r--	rk356x.dtsi	50005	log	stats	plain

## 第6章 Linux カーネルのコンパイル

ここでの Linux カーネルは Kernel を指し、またカーネルも Kernel を指します。

### 6.1 自分で Kernel をコンパイルする理由

提供されたカーネルはほとんどの機能をサポートしていますが、特定の機能をカスタマイズしたい顧客にとっては、必要な機能設定がない場合があります。そのため、この章ではカーネルの設定とコンパイル方法について説明します。

### 6.2 Kernel の取得

#### 6.2.1 ソースコードのダウンロード

Kernel ソースコードを取得する方法は3つあります。1つ目は Kernel の公式カーネルソースコード、2つ目は RockChip の公式 kernel ソースコード、3つ目は修正し、ボードに適合させた kernel ソースコードです。

ここでは、提供するソースコードを例に、公式のソースコードに興味がある方もダウンロードして設定を学ぶことができます。

kernel は、Rockchip 公式が提供する kernel に基づいてカスタマイズされており、Rockchip の kernel は Kernel 公式の LTS(長期サポート)バージョンに基づいてチップの適応を行っています。組み込み開発者としては、一般的にはチップメーカーが適応させた kernel を使用して開発するだけで十分であり、Kernel 公式から新しいバージョンをダウンロードして適応することはチップメーカーの仕事です。

現在、LubanCat-RK3588-Linux-SDK は kernel 5.10 バージョンを使用しています。

rk3588 は発売されてから時間が経っていないため、Rockchip が Kernel 公式に提出した適応も少なく、多くのチップ機能が実現されていないため、現在は Rockchip の古い kernel バージョンを使用するしかありません。

Kernel 公式カーネルソースコード : <https://www.kernel.org/>

Rockchip カーネルソースコード : <https://github.com/rockchip-linux/kernel/>

LubanCat カーネルソースコード : <https://github.com/LubanCat/kernel>

提供するソースコードは以下になります

```
1 # LubanCat-RK358x
2 git clone -b stable-4.19-rk358x https://github.com/LubanCat/kernel
3
4 # LubanCat-RK3588
5 git clone -b develop-5.10 https://github.com/LubanCat/kernel
```

## 6.3 kernel のプロジェクト構造分析

ソフトウェア、特にオープンソースソフトウェアを学ぶ際には、まずソフトウェアのプロジェクト構造を分析することから始めるべきです。良いソフトウェアは良好なプロジェクト構造を持っており、読者がソフトウェアのアーキテクチャやワークフローを学び、理解する上で大いに役立ちます。

カーネルソースコードディレクトリは以下の通りです。

arch	COPYING	drivers	init	kernel	make_deb.sh	EADME	sound
block	CREDITS	firmware	ipc	lib	Makefile	samples	tools
build_image	crypto	fs	Kbuild	LICENSES	mm	scripts	usr
certs	Documentation	include	Kconfig	MAINTAINERS	net	security	virt

Linux カーネルソースコードディレクトリには非常に多くのディレクトリがあり、それぞれのディレクトリにも多くのファイルが存在します。以下、これらのディレクトリの主な役割について簡単に分析します。

**arch** : ハードウェアアーキテクチャに関連するコードを主に含みます。例えば、arm、x86、MIPS、PPC など、各 CPU プラットフォームごとに対応するディレクトリがあります。arch 内のディレクトリには、各プラットフォームおよび各チップが Linux カーネルのプロセススケジューリング、メモリ管理、割り込みなどをサポートするコード、および各 SoC およびボードのボードレベルのサポートコードが格納されています。

**block** : Linux では block はブロックデバイスを意味します（複数のバイトが一つのブロックとして整理され、セクターのように一括でアクセスされます）。例えば、SD カード、Nand、ハードディスクなどがブロックデバイスです。block ディレクトリには、Linux ストレージシステム内のブロックデバイス管理に関するコードが含まれています。

**crypto** : このディレクトリには、一般的な暗号化およびハッシュアルゴリズム（例：md5、AES、SHA など）、圧縮、CRC 検証アルゴリズムが含まれています。

**Documentation** : カーネルの各部分に関する文書です。

**drivers** : デバイスドライバプログラムで、Linux カーネルがサポートするすべてのハードウェアデバイスのドライバソースコードがリストアップされています。異なるタイプのドライバは、char、block、net、mtd、i2c など、それぞれ異なるサブディレクトリに分類されています。

**fs** : fs はファイルシステムを意味し、Linux がサポートするさまざまなファイルシステム（例：EXT、FAT、

NTFS、JFFS2 など) が含まれています。

include : コンパイルの中心となるほとんどのヘッダファイルが含まれています。例えば、プラットフォームに依存しないヘッダファイルは include/linux サブディレクトリに、CPU アーキテクチャに関連するヘッダファイルは include ディレクトリの対応するサブディレクトリにあります。

init : カーネル初期化コードで、Linux カーネルが起動する際にカーネルを初期化するコードが含まれています。

ipc : ipc はプロセス間通信を意味し、Linux プロセス間の通信に関するコードがこのディレクトリに含まれています。

kernel : Linux の中心であるカーネル自体を意味し、プロセススケジューリング、タイマーなどが含まれます。プラットフォームに関連するコードの一部は arch/\*/kernel ディレクトリに配置されています。

lib : ライブラリの略で、一般的に役立つライブラリ関数が含まれています。カーネルプログラミングでは C 言語の標準ライブラリ関数は使用できないため、lib ディレクトリのライブラリ関数を使用する必要があります。プロセッサアーキテクチャに関連するライブラリ関数のコードは、arch/\*/lib/ディレクトリに配置されています。

mm : CPU アーキテクチャに依存しないすべてのメモリ管理コードを含んでおり、ページングによるメモリの割り当てと解放などがあります。特定のハードウェアアーキテクチャに関連するメモリ管理コードは、arch/\*/mm ディレクトリにあります。例えば、arch/arm/mm/fault.c などです。

net : ネットワークプロトコルスタックに関連するコードで、さまざまな一般的なネットワークプロトコルが net ディレクトリで実装されています。

scripts : このディレクトリにはスクリプトファイルが全て含まれており、これらのスクリプトファイルは Linux カーネルが動作する際に使用されるものではなく、Linux カーネルのコンパイルに使用されます。

security : カーネルのセキュリティモデルに関連するコードで、例えば最も有名な SELINUX が含まれます。

sound : ALSA や OSS のオーディオデバイスのドライバコアコードおよび一般的なデバイスドライバが含

まれます。

usr : cpio などのパッキングと圧縮を実現するためのコードが含まれます。

これらはいくつかの一般的なディレクトリのみをリストアップしたものです。

## 6.4 カーネルの設定オプション

このセクションでは、コマンドの入力が必要で、すべての操作は kernel ディレクトリで行う必要があります。

### 6.4.1 カーネルの設定オプションの配置

異なるバージョンのカーネルと異なるの LubanCat ボードで使用される設定ファイルは、カーネルディレクトリの arch/arm64/configs に保存されています。具体的な使用設定ファイルは以下の通りです：

- Kernel 5.10 : LubanCat-RK3588 ボードは `lubancat_rk3588_linux_defconfig` を使用

Linux カーネルの設定システムは以下の 3 つの部分で構成されます：

- Makefile : Linux カーネルソースコードのルートディレクトリと各レベルのディレクトリに分散しており、Linux カーネルのコンパイルルールを定義します。
- 設定ファイル : Kconfig ファイルは設定項目を定義し、コンパイル時に `arch/arm64/configs/lubancat2_defconfig` ファイルを使用して設定項目に値を割り当てます。
- 設定ツール : 設定コマンドインタプリタ (設定スクリプトで使用される設定コマンドを解釈) と設定ユーザインターフェース (Linux はテキストベース、Ncurses グラフィカルインターフェース、および Xwindows グラフィカルインターフェースのユーザ設定インターフェースを提供し、それぞれ `make config`、`make menuconfig`、`make xconfig` に対応) が含まれます。

注意: カスタム設定ファイルを使用する場合、コンパイル時にはボードに対応する

`device/rockchip/rk358x/BoardConfig.mk` ファイル内で `RK_KERNEL_DEFCONFIG` の定義を変更する必要

があります。

make menuconfig KCONFIG\_CONFIG=arch/arm64/configs/lubancat4\_defconfig ARCH=arm64 コマンドを通じて設定を確認することができます。make menuconfig はテキストベースの選択型の設定インターフェースで、文字端末での使用を推奨します。この設定ファイルは lubancat4\_defconfig で、この時 lubancat4\_defconfig の設定選択を見ることができます。キーボードの「上」「下」「左」「右」「エンター」「スペース」「?」「ESC」などのキーを使って設定を選択できます。詳細は以下を参照してください：

1. グラフィカルインターフェースの設定には libncurses-dev が追加が必要です。
2. sudo apt install libncurses-dev
3. コマンドを実行します。
4. make menuconfig KCONFIG\_CONFIG=arch/arm64/configs/lubancat2\_defconfig ARCH=arm64

例えば、ov5648 カメラドライバーの設定を選択します。この設定オプションがどこにあるか分からない場合、make menuconfig の検索機能を使用できます。英語入力モードで"/"キーを押すと検索できます。

"ov5648"と入力して該当する設定オプションの位置を見つけます。入力に誤りがある場合は、Ctrl + バックスペースキーで入力を削除できます。図からは、ov5648 設定オプションが-> Device Drivers ->

Multimedia support (MEDIA\_SUPPORT [=y]) -> I2C Encoders, decoders, sensors and other helper chips に位置していることがはっきりとわかります。"l"キーを押すと直接該当するオプションに移動でき、以下の内容を選択できます。

ov5648 ドライバーの設定を変更する際には、y キーでカーネルに組み込む、m キーでモジュールとしてコンパイル、n キーでコンパイルしない、を意味します。スペースキーを使用して ov5648 ドライバーの設定オプションを選択できます。

他のドライバーの設定も同様です。

設定が完了したら、保存して終了し、次のコマンドを使用して defconfig ファイルを保存し、元の設定ファ



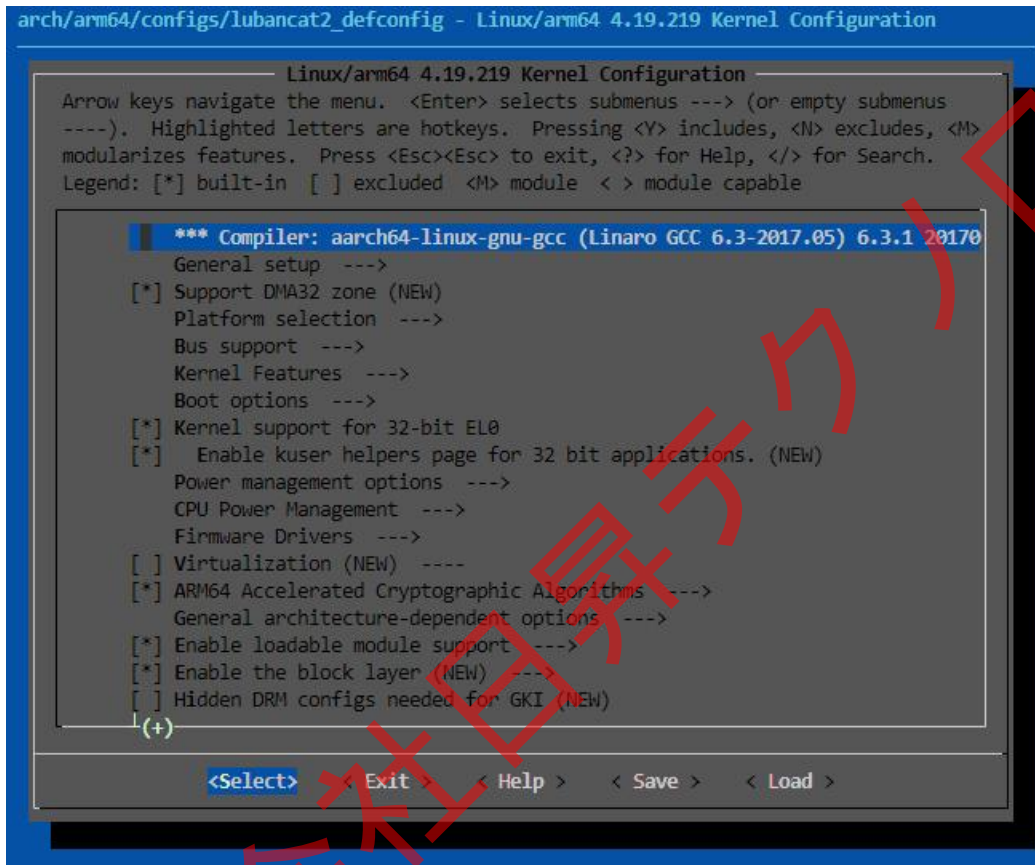
イルを上書きします。

1. defconfig ファイルを保存します。
2. make savedefconfig ARCH=arm64
3. 元の設定ファイルを上書きします。
4. cp defconfig arch/arm64/configs/lubancat2\_defconfig

```
arch/arm64/configs/lubancat2_defconfig - Linux/arm64 4.19.219 Kernel Configuration

Linux/arm64 4.19.219 Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenu
----). 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

*** Compiler: aarch64-linux-gnu-gcc (Linaro GCC 6.3-2017.05) 6.3.1 20170
General setup --->
[*] Support DMA32 zone (NEW)
Platform selection --->
Bus support --->
Kernel Features --->
Boot options --->
[*] Kernel support for 32-bit EL0
[*] Enable kuser helpers page for 32 bit applications. (NEW)
Power management options --->
CPU Power Management --->
Firmware Drivers --->
[ ] Virtualization (NEW) ----
[*] ARM64 Accelerated Cryptographic Algorithms ---->
General architecture-dependent options ---->
[*] Enable loadable module support ---->
[*] Enable the block layer (NEW) ---->
[ ] Hidden DRM configs needed for GKI (NEW)
1(+)
```



arch/arm64/configs/lubancat2\_defconfig - Linux/arm64 4.19.219 Kernel Configuration

Search Configuration Parameter  
Enter (sub)string or regexp to search for (with or without "CONFIG\_")

< Ok >    < Help >

arch/arm64/configs/lubancat2\_defconfig - Linux/arm64 4.19.219 Kernel Configuration

> Search (ov5648)

Search Results

Symbol: VIDEO\_OV5648 [=y]  
Type : tristate  
Prompt: OmniVision OV5648 sensor support  
Location:  
  -> Device Drivers  
  -> Multimedia support (MEDIA\_SUPPORT [=y])  
(1) -> I2C Encoders, decoders, sensors and other helper chips  
  Defined at drivers/media/i2c/Kconfig:1106  
  Depends on: MEDIA\_SUPPORT [=y] && VIDEO\_V4L2 [=y] && I2C [=y] && MEDIA\_CONTROLLER  
  Selects: V4L2\_FWNODE [=y]

(100%)

< Exit >

```
arch/arm64/configs/lubancat2_defconfig - Linux/arm64 4.19.219 Kernel Configuration
> Search (ov5648) > I2C Encoders, decoders, sensors and other helper chips
I2C Encoders, decoders, sensors and other helper chips
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenu
----). 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
^(-)
< > OmniVision OV2718 sensor support (NEW)
< > OmniVision OV2735 sensor support (NEW)
< > OmniVision OV2775 sensor support (NEW)
< > OmniVision OV4686 sensor support (NEW)
< > OmniVision OV4688 sensor support (NEW)
< * > OmniVision OV4689 sensor support
< * > OmniVision OV5640 sensor support
< > OmniVision OV5645 sensor support (NEW)
< > OmniVision OV5647 sensor support (NEW)
< * > OmniVision OV5648 sensor support
< > OmniVision OV5670 sensor support (NEW)
< * > OmniVision OV5695 sensor support
< > OmniVision OV6650 sensor support (NEW)
< * > OmniVision OV7251 sensor support
< > OmniVision OV772x sensor support (NEW)
< > OmniVision OV7640 sensor support (NEW)
< > OmniVision OV7670 sensor support (NEW)
< > OmniVision OV7740 sensor support (NEW)
^!(+)
<Select> < Exit > < Help > < Save > < Load >
```

## 6.5 カーネルのコンパイル

このセクションではコマンドの入力が必要で、SDKのルートディレクトリで操作を行う必要があります。

LubanCat-SDKでは、自動コンパイルスクリプトは主に build.sh に格納されており、これが SDK の主な機能エントリーポイントです。

カーネルのコンパイルには2つの方法があります。一つは rk の標準の boot パーティション、rkboot パーティションと呼ばれ、もう一つは extboot パーティションです。

rkboot パーティションはバイナリ形式でパッケージされ、各セクションのアドレスは厳格に定義されており、ファイルを読み取る際にはバイナリファイルヘッダーを使用してファイルの種類を判断します。現在、LubanCat ボードはこのサポートを停止しています。

extboot パーティションは、ext4形式でファイルを保存し、システム内で読み書き可能で、便利性を大幅に向上させました。この基盤の上に、extboot パーティションシステムにカーネルバージョンのオンライン更新、デバイスツリープラグインの変更、メインデバイスツリーの切り替え機能を追加し、最終的には同じ

プロセッサを使用する全ての LubanCat ボードに共通のイメージを実現しました。

## 6.5.1 extboot パーティションのコンパイル

extboot パーティションをパッケージする際には、カーネルの deb パッケージもパッケージされるため、extboot パーティションイメージを構築する前に、まずはカーネル deb パッケージの構築を行います。

```
1 function build_extboot(){
2 check_config RK_KERNEL_DTS RK_KERNEL_DEFCONFIG || return 0
3
4 echo "=====Start building kernel===== "
5 echo "TARGET_ARCH=$RK_ARCH"
6 echo "TARGET_KERNEL_CONFIG=$RK_KERNEL_DEFCONFIG"
7 echo "TARGET_KERNEL_DTS=$RK_KERNEL_DTS"
8 echo "TARGET_KERNEL_CONFIG_FRAGMENT=$RK_KERNEL_DEFCONFIG_FRAGMENT"
9 echo "-----"
10 pwd
11
12 build_check_cross_compile
13
14 cd kernel
15 make ARCH=$RK_ARCH $RK_KERNEL_DEFCONFIG $RK_KERNEL_DEFCONFIG_FRAGMENT
16 make ARCH=$RK_ARCH $RK_KERNEL_DTS.img -j$RK_JOBS
17 make ARCH=$RK_ARCH dtbs -j$RK_JOBS
18
```

```
19 echo -e "¥e[36m Generate extLinuxBoot image start¥e[0m"
20
21 KERNEL_VERSION=$(cat $TOP_DIR/kernel/include/config/kernel.release)
22
23 EXTBOOT_IMG=${TOP_DIR}/kernel/extboot.img
24 EXTBOOT_DIR=${TOP_DIR}/kernel/extboot
25 EXTBOOT_DTB=${EXTBOOT_DIR}/dtb/
26
27 rm -rf $EXTBOOT_DIR
28 mkdir -p $EXTBOOT_DTB/overlay
29 mkdir -p $EXTBOOT_DIR/uEnv
30 mkdir -p $EXTBOOT_DIR/kerneldeb
31
32 cp ${TOP_DIR}/${RK_KERNEL_IMG} $EXTBOOT_DIR/Image-$KERNEL_VERSION
33
34 if [ "$RK_ARCH" == "arm64" ];then
35 cp ${TOP_DIR}/kernel/arch/${RK_ARCH}/boot/dts/rockchip/*.dtb$EXTBOOT_DTB
36 cp ${TOP_DIR}/kernel/arch/${RK_ARCH}/boot/dts/rockchip/overlay/*.dtbo
$EXTBOOT_DTB/overlay
37 cp ${TOP_DIR}/kernel/arch/${RK_ARCH}/boot/dts/rockchip/uEnv/uEnv*.txt
$EXTBOOT_DIR/uEnv
38 else
```

```
39 cp ${TOP_DIR}/kernel/arch/${RK_ARCH}/boot/dts/*.dtb $EXTBOOT_DTB
40 cp ${TOP_DIR}/kernel/arch/${RK_ARCH}/boot/dts/overlay/*.dtbo$EXTBOOT_DTB/overlay
41 fi
42 cp -f $EXTBOOT_DTB/${RK_KERNEL_DTS}.dtb $EXTBOOT_DIR/rk-kernel.dtb
43
44 if [[ -e ${TOP_DIR}/kernel/ramdisk.img ]]; then
45 cp ${TOP_DIR}/kernel/ramdisk.img $EXTBOOT_DIR/initrd-$KERNEL_VERSION
46 echo -e "¥tinitrd /initrd-$KERNEL_VERSION" >> $EXTBOOT_DIR/extlinux/extlinux.conf
47 fi
48
49 cp ${TOP_DIR}/kernel/.config $EXTBOOT_DIR/config-$KERNEL_VERSION
50 cp ${TOP_DIR}/kernel/System.map $EXTBOOT_DIR/System.map-$KERNEL_VERSION
51 cp ${TOP_DIR}/kernel/logo.bmp $EXTBOOT_DIR/
52
53 cp
${TOP_DIR}/linux-headers-"$KERNEL_VERSION"_"$KERNEL_VERSION"/*.deb$EXTBOOT_DIR/kern
eldeb
54 cp
${TOP_DIR}/linux-image-"$KERNEL_VERSION"_"$KERNEL_VERSION"/*.deb$EXTBOOT_DIR/kerne
ldeb
55
56 rm -rf $EXTBOOT_IMG && truncate -s 128M $EXTBOOT_IMG

57 fakeroot mkfs.ext4 -F -L "boot" -d $EXTBOOT_DIR $EXTBOOT_IMG
```

58

```
59 finish_build
```

```
60 }
```

1. check\_config で設定ファイルの存在を確認
2. build\_check\_cross\_compile でクロスコンパイルパラメータを設定
3. 定義されたカーネル設定ファイルを適用
4. カーネルイメージをコンパイル
5. デバイスツリーをコンパイル
6. EXTBOOT\_DIR 一時フォルダを作成し、boot パーティション内に含まれるファイルを保存
7. カーネルイメージ、デバイスツリーファイル、デバイスツリープラグイン、uEnv 環境変数ファイルをコピー
8. ramdisk イメージの追加を判断
9. 起動ロゴ、カーネル設定ファイル、System.map、カーネル deb パッケージをコピー
10. EXTBOOT\_DIR フォルダ内のファイルを ext4 形式で extboot.img にパッケージ

その後、extboot.img を rockdev/boot.img へのソフトリンクとして設定します。

## 6.6 カーネル deb パッケージの構築

カーネルで make bindeb-pkg を実行すると、最大で5つの Debian パッケージが生成されます。

LubanCat-SDK では、以下のコマンドで直接構築できます。

```
1 ./build.sh kerneldeb
```

構築スクリプトは以下の通りです。

```
1 ./build.sh kerneldeb
```

```
2
```

```
3 function build_kerneldeb(){
```

```
4 check_config RK_KERNEL_DTS RK_KERNEL_DEFCONFIG || return 0
5
6 build_check_cross_compile
7
8 echo "=====Start building kernel deb=====
9 echo "TARGET_ARCH = $RK_ARCH"
10 echo "TARGET_KERNEL_CONFIG = $RK_KERNEL_DEFCONFIG"
11 echo "TARGET_KERNEL_DTS = $RK_KERNEL_DTS"
12 echo "TARGET_KERNEL_CONFIG_FRAGMENT = $RK_KERNEL_DEFCONFIG_FRAGMENT"
13 echo "=====
14 pwd
15 cd kernel
16 make ARCH=$RK_ARCH $RK_KERNEL_DEFCONFIG $RK_KERNEL_DEFCONFIG_FRAGMENT
17 make ARCH=$RK_ARCH bindeb-pkg RK_KERNEL_DTS=$RK_KERNEL_DTS -j$RK_JOBS
18 finish_build
19 }
```

1. check\_config で設定ファイルの存在を確認
2. build\_check\_cross\_compile でクロスコンパイルパラメータを設定
3. 定義されたカーネル設定ファイルを適用
4. カーネル deb パッケージをコンパイル

生成されるパッケージは以下の通りです：



- linux-image-version : 一般的にカーネルイメージとカーネルモジュールが含まれ、デバイスツリーとデバイスツリープラグインも追加されています。
- linux-headers-version : 外部モジュールの作成に必要なヘッダーファイルが含まれています。
- linux-firmware-image-version : 一部のドライバが必要とするファームウェアが含まれます (ここでは生成されません)。
- linux-image-version-dbg : linux-image-version にデバッグシンボルが追加されています。
- linux-libc-dev : GNU glibc など、ユーザープログラムライブラリに関連するヘッダーが含まれています。

主に使用されるのは linux-image と linux-headers で、linux-firmware はインターネットを通じてインストール可能です。

linux-headers をインストールすることで、ボード上に gcc コンパイラを直接インストールし、ローカルでコンパイルを実行できます。具体的な使用方法は、アプリケーションのデプロイメントセクションを参照してください。

linux-image をインストールすると、カーネルイメージ、デバイスツリーとデバイスツリープラグイン、カーネルモジュールを更新できます。

## 6.6.1 カーネル deb パッケージのインストール

生成した deb パッケージを USB ストレージデバイスやネットワークを通じて、Ubuntu や Debian イメージを実行しているボードにコピーし、以下のコマンドでパッケージをインストールします。

**注意:** アップデート中に電源を切らないでください。そうしないと、システムが損傷し、起動できなくなる可能性があります。

```
1 #カーネルバージョン 4.19 の場合  
2 sudo dpkg -i linux-headers-4.19.xxx_4.19.xxx-xxx_arm64.deb
```

```
3 sudo dpkg -i linux-image-4.19.xxx_4.19.xxx-xxx_arm64.deb
4
5 #カーネルバージョン 5.10 の場合
6 sudo dpkg -i linux-headers-5.10.xxx_5.10.xxx-xxx_arm64.deb
7 sudo dpkg -i linux-image-5.10.xxx_5.10.xxx-xxx_arm64.deb
```

注意: ローカルの deb パッケージをインストールする際は、deb パッケージがあるディレクトリで上記のコマンドを実行します。xxx は deb パッケージの実際の番号を指します。

deb パッケージのインストールが完了したら、再起動してカーネルのアップデートを完了します。

直接 deb パッケージを使用してローカルでアップデートする以外に、Wildfire のソフトウェアリポジトリを使用してオンラインでアップデートすることも可能です。

```
1 sudo apt update
2
3 # 全ての更新が必要なソフトウェアパッケージを更新
4 sudo apt upgrade
5
6 # linux-image と linux-headers のみ更新
7 # カーネルバージョンが 4.19.232
8 sudo apt install linux-image-4.19.232
9 sudo apt install linux-headers-4.19.232
10
11 # カーネルバージョンが 5.10.160
12 sudo apt install linux-image-5.10.160
13 sudo apt install linux-headers-5.10.160
```

アップグレードが完了した後、ボードを再起動して、カーネルのアップデートを完了します。

## 第7章 スタートアップロゴの変更

ボードが起動する際、もし画面に接続されている場合、LubanCat のロゴが画面上に表示されます。スタートアップロゴを変更したい場合は、以下の手順に従って操作できます。

### 7.1 SDK ソースコードからスタートアップロゴを変更

#### 7.1.1 画像の準備

変更したいロゴ画像を選択します。ここでは、例として LubanCat のロゴを使用します。

画像の背景色を透明に設定し、形式を 8 ビットまたは 24 ビットの bmp 形式に変換します。画像の解像度を画面の表示比率に合わせて変更し、画像サイズを約 500KB 程度に抑えます。



注意: 画像の解像度が画面表示解像度を超える場合、長さや幅の比率を変えずに自動的に画像を圧縮します。

### 7.1.2 元のロゴファイルを置き換え

1. 変換後、得られた bmp 形式のファイルを logo.bmp にリネームし、SDK プロジェクトの kernel ディレクトリにある元の logo.bmp ファイルと置き換えます。これでスタートアップロゴの置き換えが完了しました。
2. 全体のイメージを再コンパイルし、ボードに書き込みます。

## 7.2 ボードシステムからスタートアップロゴを変更

SDK ソースコードからスタートアップロゴを変更しない場合、Debian イメージまたは Ubuntu イメージを書き込みたボードで直接/boot/logo.bmp を置き換えることで、スタートアップロゴを更新できます。

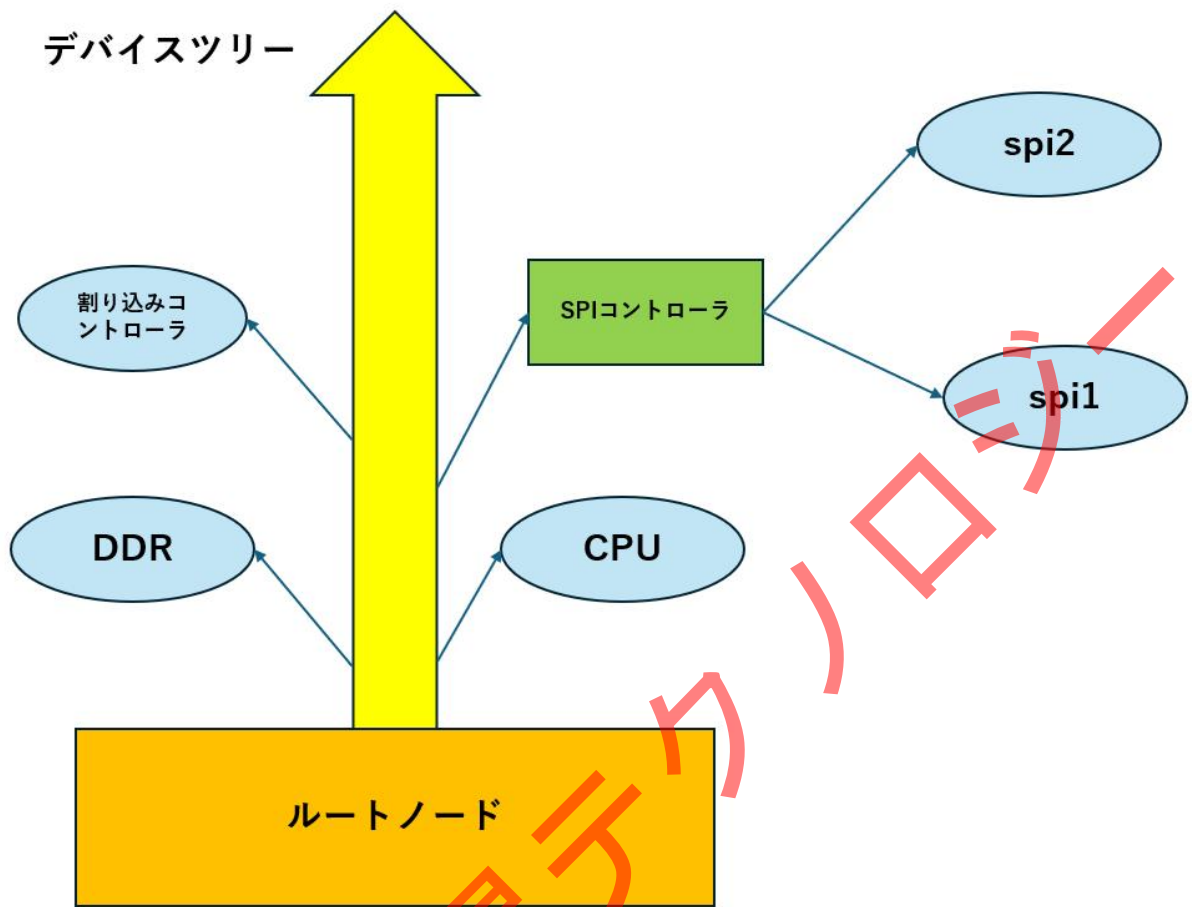
## 7.3 注意事項

- デフォルトでは、起動情報が tty1 に出力され、画面上に起動情報が表示されます。出力を無効にするには、/boot/uEnv.txt の bootargs 変数を編集し、console=tty1 を削除します。
- デバイスツリープラグインで有効にされた画面では、ロゴがデバイスツリープラグインの読み込み前に表示されるため、起動中にロゴが表示されません。mipi 画面はすべてデバイスツリープラグインを使用して有効にされているため、mipi 画面上にロゴを表示することはできません。正常に表示させるには、メインデバイスツリーを変更し、mipi 画面の設定をメインデバイスツリーに追加する必要があります。同時に、元々の HDMI との衝突に注意する必要があります。

# 第 8 章 デバイスツリーの紹介

## 8.1 デバイスツリーの紹介

デバイスツリーの役割は、ハードウェアプラットフォームのハードウェアリソースを記述することです。この「デバイスツリー」は、ブートローダー (uboot) によってカーネルに渡され、カーネルはデバイスツリーからハードウェア情報を取得できます。



デバイスツリーがハードウェアリソースを記述する際には、二つの特徴があります。

- 第一に、「木構造」でハードウェアリソースを記述します。例えば、ローカルバスを木の「幹」とし、デバイスツリーでは「ルートノード」と呼びます。ローカルバスに接続される IIC バス、SPI バス、UART バスを木の「枝」とし、デバイスツリーでは「ルートノードの子ノード」と呼びます。IIC バスの下には複数の IIC デバイスがあり、これらの「枝」はさらに分岐することができます。

- 第二に、デバイスツリーはヘッダーファイル (.h ファイル) のように、一つのデバイスツリーファイルが別のデバイスツリーファイルを参照することができます。これにより、「コード」の再利用が可能になります。例えば、複数のハードウェアプラットフォームが RK3588 をメインコントローラチップとして使用する場合、RK3588 チップのハードウェアリソースを一つのデバイスツリーファイルに記述し、通常は「.dtsi」の拡張子を使用します。他のデバイスツリーファイルは直接「#includexxx」で参照することができます。

例えば、arch/arm64/boot/dts/rockchip/rk3588.dtsi というファイルは、通常はチップメーカーが提供し、rk3588 チップ内のほぼ全てのデバイスおよび外部インターフェイスを含んでいます。使用時には、ボードのデバイスツリーソースファイル内で#include "rk3588.dtsi"を使って、rk3588 チップの全デバイスをインポートし、その後ボード上の外部デバイスに応じて修正します。

## 8.2 デバイスツリーの一般的なファイル

DTS は、.dts フォーマットのファイルを指し、ASCII テキスト形式のデバイスツリー記述であり、書く必要があるデバイスツリーソースコードです。通常、一つの.dts ファイルは一つのハードウェアプラットフォームに対応します。Linux ソースコードの「arch/arm64/boot/dts/」ディレクトリには、チップメーカーごとに分類されています。

DTSI は、チップメーカーが提供する、同一チッププラットフォームで「共有」されるデバイスツリーファイルを指します。

DTC は、デバイスツリーのソースコードをコンパイルするためのツールで、通常はこのコンパイルツールを手動でインストールする必要があります。

DTB は、デバイスツリーのソースコードがコンパイルされたファイルで、C 言語の「.C」ファイルが「.bin」ファイルにコンパイルされるのと似ています。

DTBO は、デバイスツリーのオーバーレイファイルをコンパイルして生成されるファイルで、DTB に重ねて追加することができます。

## 8.3 デバイスツリーのノードの書き方

詳しい書き方のチュートリアルについては、Linux のデバイスツリードキュメントを参照してください。

## 第9章 デバイスツリーのコンパイル

U-boot と kernel のソースコードには、デバイスツリーファイルが含まれています。U-Boot のネイティブアーキテクチャの要件に従い、1枚のカードは1つの U-Boot dts に対応し、U-Boot dts から生成された dtb は U-Boot 自身のイメージにパッケージされます。これにより、各 SoC プラットフォームで N 枚のカードには N 個の U-Boot イメージが必要になります。

一つの SoC プラットフォームで異なるカード間の主な違いは外部デバイスであり、SoC のコア部分は同じであることがわかります。RK プラットフォームは、一つの SoC プラットフォームで1つの U-Boot イメージのみが必要となるように、kernel DTB メカニズムを追加しました。これは、初期段階で kernel DTB に切り替え、その設定情報を使用して外部デバイスを初期化することを意味します。

RK プラットフォームは、kernel DTB をサポートすることで、display、pmic/regulator、pinctrl、clk などのカードの違いに対応できます。基本的には、U-Boot のデバイスツリーを使用して一般的な外部デバイスを初期化し、U-Boot 段階で内核のデバイスツリーをロードして特殊な部分を初期化するということです。したがって、デバイスツリーを変更する際は、主にカーネルのデバイスツリーを変更することになります。たとえば、LubanCat-RK の開発プロセスでは、U-Boot のデバイスツリーを変更せずに、rk3588-evb カードのデバイスツリーを直接使用しています。

### 9.1 デバイスツリーファイルのコンパイル

LubanCat-SDK のコンパイル環境を利用して、カーネルイメージをコンパイルすると同時に、対応するデバイスツリーもコンパイルします。LubanCat-SDK のルートディレクトリで以下のコマンドを実行してカーネルをコンパイルします：

```
1 # kerneldeb ファイルをコンパイル
2 ./build.sh kerneldeb
3
```



```
4 # extboot パーティションをコンパイル
5 ./build.sh extboot
```

デバイスツリーファイルとデバイスツリーソースファイルは同じディレクトリに生成されます。たとえば、LubanCat-4 ボードの場合、デバイスツリーソースファイルは arch/arm64/boot/dts/rockchip/rk3588-lubancat-4.dts にあり、コンパイル後に生成されるデバイスツリーファイルは arch/arm64/boot/dts/rockchip/rk3588-lubancat-4.dtb です。

## 9.2 同じボードでデバイスツリーファイルを変更する方法

### 9.2.1 extboot パーティションでデバイスツリーファイルを変更する

extboot パーティションを使用するイメージの場合、extboot イメージを生成する際に Makefile で指定された全てのデバイスツリーがパッケージされています。ボードが初めて起動する時には基本的なデバイスツリーを使用し、システムが初めて起動した後はボードのハードウェア ID に基づいて対応するデバイスツリーに自動的に切り替わります。

システムの boot パーティションはデフォルトで /boot ディレクトリにマウントされています。表示デバイスを切り替えたい場合は、設定ツールを使用するか、/boot/rk-kernel.dtb のシンボリックリンクが実際のデバイスツリーファイルを指しているアドレスを直接変更することができます。

具体的な切り替え方は、「デバイスツリーとデバイスツリープラグイン」を参照してください。

自分でデバイスツリーを追加したい場合は、以下の手順に従ってください：

1. kernel/arch/arm64/boot/dts/rockchip ディレクトリに新しいデバイスツリーソースファイルを作成します。例えば、rk3588-lubancat-4-xxx.dts という名前です。
2. 同じディレクトリ内の Makefile を編集し、dtb-\$(CONFIG\_CPU\_RK3588) += rk3588-lubancat-4-xxx.dtb ¥を追加します。
3. ./build.sh kerneldeb コマンドを使用してカーネル deb パッケージをビルドします。

4. ./build.sh extboot コマンドを使用して extboot パーティションをビルドします。
5. ./mkfirmware.sh コマンドを使用してファームウェアを再パッケージします。
6. rockdev ディレクトリで boot.img パーティションイメージファイルを確認します。
7. パーティション書き込みツールを使用して書き込みを行います。

## 第 10 章 ROOT ファイルシステムの紹介

### 10.1 ROOT ファイルシステムの概要

ROOT ファイルシステムはまず最初にカーネルが起動する際にマウントされるファイルシステムであり、カーネルのコードイメージファイルが ROOT ファイルシステムに保存されています。そして、システムのブートローダーは ROOT ファイルシステムのマウント後、基本的な初期化スクリプトやサービスなどをメモリにロードして実行します。これは Windows の C ドライブに相当し、システムの起動後のアプリケーションやシステム設定が保存されています。

### 10.2 ROOT ファイルシステムディレクトリの概要

/bin ディレクトリ

このディレクトリにあるコマンドは、root と一般アカウントの両方が使用可能です。これらのコマンドは他のファイルシステムをマウントする前に使用できるため、/bin ディレクトリは ROOT ファイルシステムと同じパーティション内になければなりません。/bin ディレクトリにはよく使われるコマンドとして cat、chgrp、chmod、cp、ls、sh、kill、mount、umount、mkdir などがあります。後に Busybox を使って ROOT ファイルシステムを作成する際、生成された bin ディレクトリには実行可能なファイル、つまり使用可能なコマンドがいくつか見られます。

/sbin ディレクトリ

このディレクトリにはシステムコマンドが保存されており、すなわちシステム管理者（俗にいう最高権限の root）だけが使用可能なコマンドです。システムコマンドは /usr/sbin や /usr/local/sbin ディレクトリに

も保存されていますが、/sbin ディレクトリには基本的なシステムコマンドが保存されており、これらはシステムの起動や修復などに使用されます。他のファイルシステムをマウントする前に/sbin が使用できるように、/sbin ディレクトリも ROOT ファイルシステムと同じパーティション内になければなりません。/sbin ディレクトリには shutdown、reboot、fdisk、fsck、init などのよく使われるコマンドがあります。ローカルユーザーが自分でインストールしたシステムコマンドは/usr/local/sbin ディレクトリに置かれます。

#### /dev ディレクトリ

このディレクトリにはデバイスとデバイスインターフェイスのファイルが保存されています。デバイスファイルは Linux 固有のファイルタイプであり、Linux システムでは、ファイルの形式で様々なデバイスにアクセスします。例えば、「/dev/ttySAC0」ファイルを通じてシリアルポート 0 を操作したり、

「/dev/mtdblock1」を通じて MTD デバイスの第 2 パーティションにアクセスできます。重要なファイルには /dev/null, /dev/zero, /dev/tty, /dev/lp\* などがあります。

#### /etc ディレクトリ

このディレクトリにはシステムの主要な設定ファイルが保存されています。たとえば、ユーザーのアカウントとパスワードのファイル、各種サービスの起動ファイルなどです。一般的に、このディレクトリのファイル属性は一般ユーザーが閲覧可能ですが、root のみを変更できます。PC 上の Linux システムでは、/etc ディレクトリには多くのファイルやディレクトリがあり、これらはオプションで、システムに存在するアプリケーションや、それらが設定ファイルを必要とするかどうかによって依存します。組み込みシステムでは、これらの内容を大幅に削減できます。

#### /lib ディレクトリ

このディレクトリには共有ライブラリとロード可能なドライバーが保存されています。共有ライブラリはシステムの起動時に使用され、ROOT ファイルシステム内の実行可能プログラム（例：/bin、/sbin ディレクトリ下のプログラム）の実行に用いられます。

#### /home ディレクトリ

システムのデフォルトユーザーフォルダで、オプションです。各一般ユーザーには、/home ディレクトリの下にユーザー名で命名されたサブディレクトリがあり、そこにはユーザー関連の設定ファイルが保存されます。

#### /root ディレクトリ

システム管理者 (root) のメインフォルダで、root ユーザーのディレクトリです。これに対応して、一般ユーザーのディレクトリは/home の下の何らかのサブディレクトリです。

#### /usr ディレクトリ

/usr ディレクトリの内容は別のパーティションに存在し、システム起動後に ROOT ファイルシステムの/usr ディレクトリにマウントされます。ここには共有されたり読み取り専用のプログラムやデータが保存されており、/usr ディレクトリの内容は複数のホスト間で共有可能で、これは FHS (ファイルシステム階層標準) にも合致しています。/usr の中のファイルは読み取り専用であり、他のホスト固有の変更可能なファイルは他のディレクトリ (例: /var) に保存されるべきです。組み込みシステムでは、/usr ディレクトリを簡略化できます。

#### /var ディレクトリ

/usr ディレクトリとは逆に、/var ディレクトリには変更可能なデータが保存されています。たとえば、スプールディレクトリ (mail, news)、ログファイル、一時ファイルなどです。

#### /proc ディレクトリ

これは空のディレクトリで、通常は proc ファイルシステムのマウントポイントとして使われます。proc ファイルシステムは仮想のファイルシステムで、実際のストレージデバイスは存在しません。このディレクトリとファイルはカーネルによって一時的に生成され、システムの運行状態を表したり、ファイル操作によってシステムを制御することができます。

#### /mnt ディレクトリ

これはファイルシステムを一時的にマウントするためのポイントとして使われる、通常空のディレクトリ

です。ここには、例えば/mnt/cdrom や/mnt/hda1 のような空のサブディレクトリを作成することができます。これは CD-ROM や移動式ストレージデバイスなどを一時的にマウントするために使用されます。

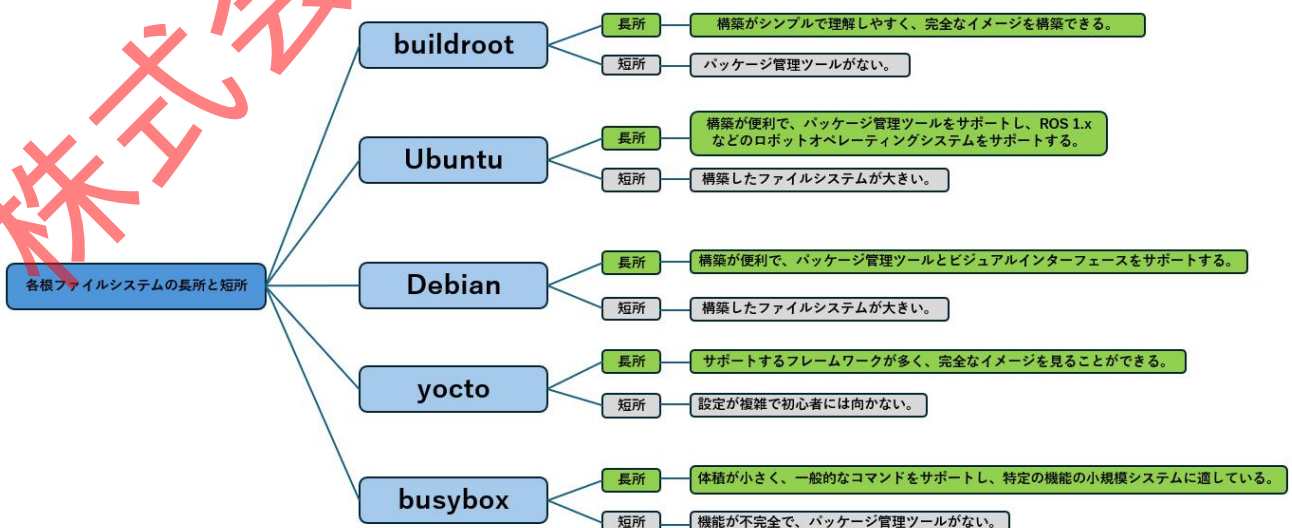
/tmp ディレクトリ

これは一時ファイルを保存するための、通常は空のディレクトリです。一時ファイルを生成する必要があるプログラムが/tmp ディレクトリを使用するため、このディレクトリは存在し、アクセス可能でなければなりません。

組み込み Linux システムの ROOT ファイルシステムに関しては、上に挙げたような複雑な構成を必要としないことが一般的です。たとえば、組み込みシステムは通常、複数ユーザーを対象としていないため、/home ディレクトリは組み込み Linux ではあまり使用されません。また、/boot ディレクトリの必要性は、使用しているブートローダーがカーネル起動前に ROOT ファイルシステムからカーネルイメージを再取得できるかどうかによって依存します。一般的に、必要なのは/bin、/dev、/etc、/lib、/proc、/var、/usr のようなディレクトリだけで、他のディレクトリはオプションです。

### 10.3 ROOT ファイルシステムの一般的な特徴

ROOT ファイルシステムは基本的にファイルや設定情報の集まりであり、その境界は明確に定義されていません。以下に、いくつかの ROOT ファイルシステムの特徴を大まかにリストアップします。



これからの章では、さまざまな ROOT ファイルシステムの構築方法について説明します。

## 第 11 章 DebianROOT ファイルシステムの構築

LubanCat-SDK を利用することで、Debian イメージの構築を簡単に一括で行うことができますが、DebianROOT ファイルシステムの構築プロセスは SDK の他の部分に依存しない、比較的独立したものです。DebianROOT ファイルシステムは live build を利用して構築されます。live build はライブシステムイメージを構築するためのスクリプトのセットです。live build はツールキットであり、構成ディレクトリを使用してライブイメージの構築のすべての面を完全に自動化し、カスタマイズします。live build 以外にも、Debian の構築リポジトリには多くのビルドスクリプトがあり、手作業を最大限に減らし、構築されたルートファイルシステムの一貫性を保つことができます。

注意: LubanCat-SDK で使用されている Debian ビルドプロジェクトは、提供する Debian イメージの機能と同じです。ユーザーには、構築したルートファイルシステムイメージを直接使用することを推奨します。具体的な使用方法とダウンロードアドレスについては、LubanCat-SDK の構築セクションをご覧ください。

### 11.1 Debian のサポート状況

#### 11.1.1 LubanCat-RK3588

Debian 11 “Bullseye”のみをサポートします。

- lite デスクトップなしバージョン
- gnome デスクトップバージョン (デフォルトデスクトップ)
- xfce デスクトップバージョン (技術サポートなし)
- lxde デスクトップバージョン (技術サポートなし)

### 11.2 Debian とは

以下のドキュメントはDebian10を例に説明しますが、Debian11の内容も類似しています。Debian11を使用する場合は、関連する内容やスクリプト中の"debain11"と"buster"を“debain11”と“bullseye”に置き換えて

ください。



Debian GNU/Linux（略称 Debian）は、世界で最も大きな非商業 Linux ディストリビューションの一つであり、完全に自由でオープンな Linux オペレーティングシステムで、さまざまなデバイスで広く使用されています。

Debian の特徴：

- ユーザー向けの Debian
- Debian は自由ソフトウェアであり、誰もが自由に使用、変更、および配布できます。
- 安定しており、セキュアです。
- 広範なハードウェアサポートを提供します。
- スムーズなアップデートを提供します。
- 多くの他のディストリビューションの基盤です。
- Debian プロジェクトはコミュニティです。
- 開発者向けの Debian

- AMD64、i386、ARM、MIPS など、多種多様なハードウェアアーキテクチャをサポートします。
- IoT や組み込みデバイスをサポートします。
- 多数のパッケージを持ち、deb 形式を使用します。
- 異なるリリースバージョンがあります。
- 公開されたバグトラッキングシステムがあります。

Debian 公式ウェブサイト：<https://www.debian.org/>

### 11.3 Debian ルートファイルシステム構築リポジトリ

LubanCat-SDK には、完全な Debian ルートファイルシステム構築プロジェクトが含まれており、SDK の debian ディレクトリに保存されています。

debian ディレクトリに入ると、以下のファイルになります

```
1 ls -hgG
2
3 -rwxrwxr-x 1 1.1K 3 月 10 11:04 mk-base-debian.sh
4 -rwxrwxr-x 1 8.1K 3 月 10 11:25 mk-buster-rootfs.sh
5 -rwxrwxr-x 1 477 3 月 10 10:49 mk-image.sh
6 -rwxrwxr-x 1 441 12 月 26 15:18 mk-rootfs.sh
7 drwxrwxr-x 7 67 11 月 30 11:19 overlay
8 drwxrwxr-x 5 49 11 月 30 11:19 overlay-debug
9 drwxrwxr-x 4 28 11 月 30 11:19 overlay-firmware
10 drwxrwxr-x 3 19 11 月 30 11:19 packages
11 drwxrwxr-x 5 47 11 月 30 11:19 packages-patches
```

```
12 -rwxrwxr-x 1 3.0K 11 月 30 11:19 post-build.sh
```



13 -rw-rw-r-- 1 2.6K 3 月 10 11:17 readme.md

14 drwxrwxr-x 7 164 11 月 30 11:19 ubuntu-build-service

- mk-base-debian.sh : ビルドディレクトリをクリーンアップし、live build を呼び出してビルドを開始します。
- mk-buster-rootfs.sh : Rockchip オーバーレイ層を追加します。
- mk-image.sh : ルートファイルシステムを img イメージファイルにパッケージングします。
- overlay : Rockchip オーバーレイ層で、主に rootfs の設定ファイルです。
- overlay-debug : Rockchip オーバーレイ層で、主にデバッグスクリプトとツールです。
- overlay-firmware : Rockchip オーバーレイ層で、主に wifi/bt/npu のファームウェアです。
- packages : ハードウェアアクセラレーションパッケージです。
- ubuntu-build-service : ビルド環境を構築するための依存ファイルと live build の設定ファイルです。

現在、ビルドスクリプトは3種類のイメージビルドをサポートしています。

- lite : デスクトップなし、ターミナル版です。
- xfce : xfce パッケージを使用したデスクトップ版です。
- xfce-full : xfce パッケージ+さらに推奨されるソフトウェアパッケージを使用したデスクトップ版です。

## 11.4 Debian ルートファイルシステムのビルドプロセス

Debian ルートファイルシステムのビルドは主に3つのステップに分かれます。

第一ステップ : live build ツールを使用して、lite-debian または xfce-debian ルートファイルシステムをビルドします。mk-base-debian.sh スクリプトで実現します。

第二ステップ : RK プロセッサベースの機能強化パッケージ、例えば GPU ドライバやハードウェアファームウェアなどを追加します。mk-buster-rootfs.sh スクリプトで実現します。

第三ステップ : ビルド完了したルートファイルシステムを img 形式にパッケージングし、簡単に書き込みや次の処理ができるようにします。mk-image.sh スクリプトで実現し、第二ステップ完了後に自動的に呼び

出されます。

## 11.5 ビルド環境の構築

debian ディレクトリ下で以下のコマンドを実行します。

```
1 sudo apt-get install binfmt-support qemu-user-static
2 sudo dpkg -i ubuntu-build-service/packages/*
3 sudo apt-get install -f
```

上記のコマンドを実行する過程で警告やエラーが出ることがありますが、これは正常な現象です。エラーは無視して直接進めます。

```
cat@lubancat:~$ sudo apt-get install binfmt-support qemu-user-static
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
binfmt-support is already the newest version (2.2.1-1+deb11u1).
The following packages were automatically installed and are no longer required:
  liba52-0.7.4 libdca0
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  qemu-user-static
0 upgraded, 1 newly installed, 0 to remove and 18 not upgraded.
Need to get 32.2 MB of archives.
After this operation, 276 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://mirrors.usfc.edu.cn/debian bullseye/main arm64 qemu-user-static arm64 1:5.2+dfsg-11+deb11u3 [32.2 MB]
Fetched 32.2 MB in 4s (8,569 kB/s)
Selecting previously unselected package qemu-user-static.
(Reading database ... 115114 files and directories currently installed.)
Preparing to unpack .../qemu-user-static_1%3a5.2+dfsg-11+deb11u3_arm64.deb ...
Unpacking qemu-user-static (1:5.2+dfsg-11+deb11u3) ...
Setting up qemu-user-static (1:5.2+dfsg-11+deb11u3) ...
Processing triggers for man-db (2.9.4-2) ...
```

```
cat@lubancat:~/LubanCat_SDK/debian$ sudo dpkg -i ubuntu-build-service/packages/*
(Reading database ... 115861 files and directories currently installed.)
Preparing to unpack .../debootstrap_1.0.123_all.deb ...
Unpacking debootstrap (1.0.123) over (1.0.123) ...
Preparing to unpack .../live-build_20210902_all.deb ...
Unpacking live-build (1:20210902) over (1:20210902) ...
```

```
cat@lubancat:~/LubanCat_SDK$ sudo apt-get install -f
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  liba52-0.7.4 libdca0
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 19 not upgraded.
```

## 11.6 Debian ルートファイルシステムイメージの構築

ubuntu-build-service ディレクトリには、lite または xfce バージョン、armhf または arm64 アーキテクチャ

に応じて、live build のいくつかのプリセットファイルが保存されています。これには、パッケージリスト、ユーザー名、パスワード、ユーザーグループ、タイムゾーンなどの設定が含まれます。

mk-base-debian.sh スクリプトを使用して、対応する Debian ルートファイルシステムを live build で構築します。

理論上は、生成されたルートファイルシステムは既にボード上で実行可能ですが、ネットワーク、ディスプレイなどのボードに対する設定がまだ追加されておらず、コアサービスのみが実行可能です。

以下は具体的な構築プロセスです：

### 11.6.1 debian-base 基本ルートファイルシステムの構築

debian ディレクトリで以下のコマンドを実行します。

```
1 ./mk-base-debian.sh
```

構築する Debian バージョンを選択します。ここでは xfce バージョンを選択し、2 を入力して Enter キーを押します。プロンプトに従ってユーザーパスワードを入力します。

```
1 -----
2 please enter TARGET version number:
3 構築するルートファイルシステムバージョンを入力してください:
4 [0] Exit Menu
5 [1] lite
6 [2] xfce
7 [3] xfce-full
```

```
8 -----
9 2
```

全体の構築時間は長いので、コマンドが終了した後、debian ディレクトリのファイル変更を確認します：

```
1 ls -hgG
2
3 -rw-rw-rw- 1 370M 3 月 10 14:04 linaro-buster-xfce-alip-20230418.tar.gz
4 -rwxrwxr-x 1 1.1K 3 月 10 11:04 mk-base-debian.sh
5 -rwxrwxr-x 1 8.1K 3 月 10 11:25 mk-buster-rootfs.sh
6 -rwxrwxr-x 1 477 3 月 10 10:49 mk-image.sh
7 -rwxrwxr-x 1 441 12 月 26 15:18 mk-rootfs.sh
8 drwxrwxr-x 7 67 11 月 30 11:19 overlay
9 drwxrwxr-x 5 49 11 月 30 11:19 overlay-debug
10 drwxrwxr-x 4 28 11 月 30 11:19 overlay-firmware
11 drwxrwxr-x 3 19 11 月 30 11:19 packages
12 drwxrwxr-x 5 47 11 月 30 11:19 packages-patches
13 -rwxrwxr-x 1 3.0K 11 月 30 11:19 post-build.sh
14 -rw-rw-r-- 1 2.6K 3 月 10 11:17 readme.md
15 drwxrwxr-x 7 164 11 月 30 11:19 ubuntu-build-service
```

新たに追加されたファイル linaro-buster-xfce-alip-20230418.tar.gz は、live build で構築された基本 debian ルートファイルシステムの圧縮パッケージです。

## 11.6.2 完全な debian ルートファイルシステムの構築

前のステップで構築されたルートファイルシステムは既にボード上で実行可能ですが、LubanCat 上での実行をさらに最適化するためには、Rockchip overlay 層を追加する必要があります。これには、主にいくつかの設定ファイルとファームウェアが含まれ、ルートファイルシステム内の既存の設定ファイルを追加ま

たは上書きするために使用されます。

```
1 ./mk-buster-rootfs.sh
```

構築する Debian バージョンを選択します。ここでは xfce バージョンを選択し、2 を入力して Enter キーを押します。プロンプトに従ってユーザーパスワードを入力します。

```
1 -----
2 please enter TARGET version number:
3 構築するルートファイルシステムバージョンを入力してください:
4 [0] Exit Menu
5 [1] lite
6 [2] xfce
7 [3] xfce-full
8 -----
9 2
```

```
1 drwxr-xr-x 23 270 3 月 10 14:57 binary
2 -rw-rw-rw- 1 370M 3 月 10 14:04 linaro-buster-xfce-alip-20230418.tar.gz
3 -rw-rw-r-- 1 1.3G 3 月 10 14:11 linaro-xfce-rootfs.img
4 -rwxrwxr-x 1 1.1K 3 月 10 11:04 mk-base-debian.sh
5 -rwxrwxr-x 1 8.1K 3 月 10 11:25 mk-buster-rootfs.sh
6 -rwxrwxr-x 1 477 3 月 10 10:49 mk-image.sh
7 -rwxrwxr-x 1 441 12 月 26 15:18 mk-rootfs.sh
8 drwxrwxr-x 7 67 11 月 30 11:19 overlay
9 drwxrwxr-x 5 49 11 月 30 11:19 overlay-debug
10 drwxrwxr-x 4 28 11 月 30 11:19 overlay-firmware
```

- 11 drwxrwxr-x 3 19 11 月 30 11:19 packages
- 12 drwxrwxr-x 5 47 11 月 30 11:19 packages-patches
- 13 -rwxrwxr-x 1 3.0K 11 月 30 11:19 post-build.sh
- 14 -rw-rw-r-- 1 2.6K 3 月 10 11:17 readme.md
- 15 drwxrwxr-x 7 164 11 月 30 11:19 ubuntu-build-service

構築が完了すると、debian ディレクトリに binary ディレクトリが追加されます。この中には解凍されたルートファイルシステムが格納されており、上書きまたは追加するファイルをコピーし、その後 chroot コマンドを使って変更を加えます。

### 11.6.3 debian-lite ルートファイルシステムイメージのパッケージング

スクリプト ./mk-buster-rootfs.sh の最後で、自動的に `IMAGE_VERSION=$TARGET ./mk-image.sh` スクリプトが呼び出され、イメージをパッケージ化します。TARGET はスクリプト実行時に選択した Debian バージョンです。

スクリプト実行後、linaro-xfce-rootfs.img という名前のルートファイルシステムイメージファイルが得られます。

## 11.7 Debian ルートファイルシステムのカスタマイズ

イメージの容量制限のため、提供するカスタマイズされた Debian イメージには一部の一般的なソフトウェアがプリインストールされていますが、ユーザーが開発する際には、さらに多くのソフトウェアをプリインストールしたり、ルートファイルシステムをさらにカスタマイズする必要があります。以下のセクションでは、ルートファイルシステムの変更について具体的に説明します。

### 11.7.1 プリインストールソフトウェアの追加

プリインストールソフトウェアを追加するには、mk-buster-rootfs.sh スクリプトに追加することをお勧めします。これにより、変更後に Rockchip overlay 層を再度追加し、img イメージをパッケージ化するプロセ

スを繰り返すだけで、開発時間を大幅に節約できます。

たとえば、ルートファイルシステムに git と vim をプリインストールしたい場合は、mk-buster-rootfs.sh に以下の内容を追加します。

```
1 export APT_INSTALL="apt-get install -fy --allow-downgrades"  
2 # export APT_INSTALL の次の行に追加  
3  
4 # 追加する内容は  
5 echo -e "¥033[47;36m ----- LubanCat ----- ¥033[0m"  
6 ${APT_INSTALL} git vim
```

### 11.7.2 外部デバイスの firmware の追加

無線 LAN カードなどの外部デバイスを使用する場合は、ルートファイルシステムにカードの firmware を追加する必要があります。この場合、対応する firmware を直接 overlay-firmware/ディレクトリに配置し、ルートファイルシステム内のパスに従って保存します。

### 11.7.3 サービス項目と設定ファイルの追加

特定のサービス項目の設定をカスタマイズしたい場合は、overlay/ディレクトリに対応する設定ファイルを追加できます。ルートファイルシステムの作成プロセス中に Rockchip overlay 層を追加する際、ルートファイルシステム内の既存の設定ファイルを追加または置換し、設定ファイルをカスタマイズする効果が得られます。

ここでは Debian コンソールログイン時のバナープロファイルを例にします。その設定ファイルはルートファイルシステムの/etc/updatemotd.d ディレクトリにあり、これに対応するのは overlay/etc/update-motd.d/ディレクトリです。

overlay/etc/update-motd.d/内に 00-header という名前の新しいファイルを作成し、次の内容をファイルに





```

23 _
24 || _ _||_ ____ /_|____||
25 || ||| || ' ¥/ _|| _ ||| /_ ||_|
26 |__||| || |)||(| ||| ||||_| (||| |
27 |____|¥_.||_._/ ¥_'||| | |¥____|¥_'|¥_|
28 ¥e[0m"
29
30 if [ -z "$DISTRIB_DESCRIPTION" ] && [ -x /usr/bin/lsb_release ]; then
31 # Fall back to using the very slow lsb_release utility
32 DISTRIB_DESCRIPTION=$(lsb_release -s -d)
33 fi
34 printf "¥n"
35 printf "Welcome to %s (%s %s %s)¥n" "$DISTRIB_DESCRIPTION" "$(uname -o)" "$(uname -r)"
"$ (uname -m)"
  
```

このスクリプトは、動的なバナーを生成するためのものです。

設定ファイルを追加した後、イメージを再構築し、ボードに書き込みて起動すると、希望するバナー効果が得られます。

注意: 追加するファイルがシェルスクリプトの場合、ファイルを作成した後、ファイルの権限を 775 に変更する必要があります。そうしないと、ルートファイルシステムで実行できない可能性があります。

## 11.7.4 ルートファイルシステムイメージの再パッケージング

ルートファイルシステムの構築スクリプトを変更した後、debian-xfce/lite-rootfs.img イメージを再パッケージする必要があります。

• live build の設定ファイルを変更していない場合、基本イメージ部分の再構築は不要です。設定ファイルを変更した場合は、基本イメージ圧縮ファイルを手動で削除し、mk-base-debian.sh スクリプトを使用して再構築する必要があります。

```
1 # 基本イメージ圧縮ファイルを削除
2 rm linaro-*-alip-*.tar.gz
3
4 # 基本イメージを構築
5 mk-base-debian.sh
```

• mk-buster-rootfs.sh、overlay、overlay-debug、overlay-firmware、packages の内容を変更した場合は、次を実行する必要があります。

```
1 # 完全なルートファイルシステムイメージを構築
2 ./mk-buster-rootfs.sh
```

## 11.8 LubanCat-SDK を使用したワンクリック構築

構築環境が整った後、ワンクリック構築コマンドを直接使用して、提供するカスタマイズされたルートファイルシステムイメージを構築することができます。また、SDK のイメージパッケージ機能を利用して、U-boot、カーネルなども含む完全なシステムイメージと一緒にパッケージすることができます。

### 11.8.1 SDK 設定ファイルの説明

LubanCat ボードの SDK 設定ファイルは device/rockchip/rk358x/ディレクトリ内にあり、BoardConfig-LubanCat-CPU モデル-システムタイプ-システムバージョン.mk という命名規則で保存されています。

Debian ルートファイルシステムの設定ファイルを見てみましょう。ここでは

BoardConfig-LubanCat-RK3588-debian-xfce.mk を例に、Debian ルートファイルシステムに関連する設定

について主に説明します。

```
1 # SOC
2 export RK_SOC=rk358x
3
4 # build.sh save パッケージ時の名前
5 export RK_PKG_NAME=lubancat-${RK_UBOOT_DEFCONFIG}
6
7 # デフォルトの rootfs を debian に設定
8 export RK_ROOTFS_SYSTEM=debian
9
10 # Debian バージョンを設定 (10: buster)
11 export RK_DEBIAN_VERSION=10
12
13 # デフォルトの rootfs がデスクトップ版かどうかを定義 xfce: デスクトップ版 lite: コンソール版
14 export RK_ROOTFS_TARGET=xfce
15
16 # デフォルトの rootfs に DEBUG ツールを追加するかどうかを定義 debug: 追加 none: 追加しない
17 export RK_ROOTFS_DEBUG=debug
```

- RK\_SOC: SOC タイプを定義します。./mk-buster-rootfs.sh で使用されます。
- RK\_PKG\_NAME: イメージのリリース時の名前を定義します。
- RK\_ROOTFS\_SYSTEM: ルートファイルシステムのタイプを定義します。
- RK\_DEBIAN\_VERSION: Debian のリリースバージョンを定義します。現在は buster のみサポート

しています。

- RK\_ROOTFS\_TARGET：ルートファイルシステムのバージョンを定義します。
- RK\_ROOTFS\_DEBUG：rockchip の overlay\_debug を追加するかどうかを定義します。

## 11.8.2 build.sh の自動ビルドスクリプト

Debian ルートファイルシステムのワンクリックビルド機能は、主に build.sh スクリプト内の以下の関数で実現されています。

```
1 function build_debian(){
2 ARCH=${RK_DEBIAN_ARCH:-${RK_ARCH}}
3 case $ARCH in
4 arm|armhf) ARCH=armhf ;;
5 *) ARCH=arm64 ;;
6 esac
7
8 echo "=====  
9 echo "RK_DEBIAN_VERSION: $RK_DEBIAN_VERSION"  
10 echo "RK_ROOTFS_TARGET: $RK_ROOTFS_TARGET"  
11 echo "RK_ROOTFS_DEBUG: $RK_ROOTFS_DEBUG"  
12 echo ""  
13
14 cd debian
15
16 if [[ "$RK_DEBIAN_VERSION" == "stretch" || "$RK_DEBIAN_VERSION" == "9" ]]; then
```

```
17 RELEASE='stretch'

18 elif [[ "$RK_DEBIAN_VERSION" == "buster" || "$RK_DEBIAN_VERSION" == "10" ]]; then

19 RELEASE='buster'

20 elif [[ "$RK_DEBIAN_VERSION" == "bullseye" || "$RK_DEBIAN_VERSION" == "11" ]]; then

21 RELEASE='bullseye'

22 else

23 echo -e "¥033[36m please input the os type,stretch or buster..... ¥033[0m"

24 fi

25

26 if [ ! -e linaro-$RK_ROOTFS_TARGET-rootfs.img ]; then

27 echo "[ No linaro-$RK_ROOTFS_TARGET-rootfs.img, Run Make Debian Scripts ]"

28 if [ ! -e linaro-$RELEASE-$RK_ROOTFS_TARGET-alip-*.tar.gz ]; then

29 echo "[ build linaro-$RELEASE-$RK_ROOTFS_TARGET-alip-*.tar.gz ]"

30 RELEASE=$RELEASE TARGET=$RK_ROOTFS_TARGET ARCH=$ARCH ./mk-base-debian.sh

31 fi

32

33 RELEASE=$RELEASE TARGET=$RK_ROOTFS_TARGET VERSION=$RK_ROOTFS_DEBUG_

SOC=$RK_SOC ARCH=$ARCH ./mk-rootfs.sh

34 else

35 echo "[ Already Exists IMG, Skip Make Debian Scripts ]"

36 echo "[ Delate linaro-$RK_ROOTFS_TARGET-rootfs.img To Rebuild Debian IMG ]"

37 fi

38
```

```
39 finish_build
40 }
```

その作業フローは以下の通りです。

- echo コマンドを使用して関連する設定情報を表示します。
- `linaro-$RK_ROOTFS_TARGET-rootfs.img` が存在するかどうかを確認します

(`$RK_ROOTFS_TARGET` は設定ファイルで定義されたデスクトップ版かどうか)。存在する場合はビルドプロセスをスキップし、存在しない場合はビルドコマンドを実行します。ルートファイルシステムのビルドには時間がかかるため、頻繁にビルドしたり、既にビルドされたルートファイルシステムイメージを使用したりしたくありません。

- 基本イメージのビルドが存在するかどうかを確認します。存在しない場合は基本イメージをビルドし、存在する場合は基本イメージのビルドプロセスをスキップします。初回ビルド以外は、基本イメージを変更することは一般的にありません。
- 基本イメージをベースに、`rockchip overlay` を追加します。
- 完全なイメージをパッケージ化します。

### 11.8.3 ビルド前の準備

ビルドを開始する前に、まず SDK の設定ファイルがコンパイルする `rootfs` と一致していることを確認する必要があります。現在の設定ファイルがコンパイルする `rootfs` と一致していない場合は、先に設定ファイルを切り替える必要があります。

```
1 # SDK 設定ファイルを選択 - 直接指定
2 ./build.sh BoardConfig-xxx-debian-バージョン.mk
3
4 # SDK 設定ファイルを選択 - 番号で選択
5 ./build.sh lunch
```

正しい設定ファイルを設定した後、次のビルド作業に進むことができます。

## 11.8.4 rootfs の単独構築とパッケージング

以下のコマンドを使用して Debian ルートファイルシステムを構築します。

```
1 # Debian を構築
2 ./build.sh debian
```

コンパイルによって生成された rootfs イメージは `linaro-$RK_ROOTFS_TARGET-rootfs.img` となり、`rockdev/rootfs.ext4` にシンボリックリンクが作成されます。

注意: `linaro-$RK_ROOTFS_TARGET-rootfs.img` が存在しない場合にのみ、Debian ルートファイルシステムが再構築されます。Debian ルートファイルシステムの設定ファイルや構築スクリプトを変更した場合は、`ubuntu/linaro-$RK_ROOTFS_TARGET-rootfs.img` を手動で削除してから再構築する必要があります。

構築が完了すると、個別のパーティションテーブル、`boot.img`、`uboot.img` などのパーティションイメージを1つの完全なイメージにパッケージングできます。

以下の操作を実行する前に、U-Bootのコンパイル、Kernelのコンパイル、および直前に完了した rootfs の単独構築プロセスがすでに完了していることを確認してください。

準備が整ったら、以下のコマンドを実行します。

```
1 # ファームウェアをパッケージング
2 ./mkfirmware.sh
3
4 # update.img を生成
5 ./build.sh updateimg
```

パッケージングが完了すると、生成されたイメージは `rockdev/update.img` となり、書き込みツールを使用して、その `update.img` をボードの eMMC や SD カードに書き込むことができます。

## 11.8.5 ワンクリックで完全なイメージを構築

正しい設定ファイルを設定した後、以下のコマンドを実行すると、U-Boot、Kernel、および rootfs のコンパイルがワンクリックで完了し、update.img イメージが生成されます。

```
1 # ワンクリックコンパイル
2 ./build.sh
```

パッケージングが完了すると、生成されたイメージは rockdev/update.img となり、書き込みツールを使用して、その update.img をボードの eMMC や SD カードに書き込むことができます。

## 第 12 章 Ubuntu ルートファイルシステムの構築

LubanCat-SDK を利用すると、Ubuntu イメージをワンクリックで簡単に構築できますが、Ubuntu ルートファイルシステムの構築プロセスは相対的に独立しており、SDK の他の部分に依存しません。

Ubuntu 公式の Ubuntu-base ルートファイルシステムを使用して、ボードに適合するルートファイルシステムを構築できます。

Ubuntu 構築リポジトリは多くの構築スクリプトで構成されており、手作業を最大限に減らし、構築されたルートファイルシステムの一貫性を保つことができます。現在、Ubuntu18.04/20.04/22.04 の 3 つのバージョンの構築をサポートしています。

現在、Ubuntu20.04 を主にサポートしており、この文書も Ubuntu20.04 を例に説明していますが、3 つのバージョンの構築方法は基本的に同じです。詳細な構築コマンドについては、対応するブランチの readme ファイルをご覧ください。

注意: Ubuntu イメージの構築環境がバージョンごとに異なるため、ファイルシステムのカスタマイズ要件がない限り、顧客に Ubuntu ルートファイルシステムを自分で構築することはお勧めしません。構築したルートファイルシステムイメージを直接使用することを推奨します。具体的な使用方法とダウンロードアドレスについては、LubanCat-SDK の構築セクションをご覧ください。



注意: もし自分でルートファイルシステムを構築する必要がある場合、異なるバージョンの Ubuntu イメージの構築環境依存問題を解決するために、Docker を使用してルートファイルシステムを構築するセクションを参照してください。

## 12.1 Ubuntu のサポート状況

### 12.1.1 LubanCat-RK358x

主に ubuntu 20.04 “Focal” をサポートし、限定的に ubuntu 18.04 “Bionic”、ubuntu 22.04 “Jammy” をサポートします。

- lite デスクトップなしバージョン
- xfce4 デスクトップバージョン
- xfce4-full デスクトップバージョン

### 12.1.2 LubanCat-RK3588

主に ubuntu 20.04 “Focal” をサポートし、限定的に ubuntu 22.04 “Jammy” をサポートします。


















- lite デスクトップなしバージョン
- gnome デスクトップバージョン (デフォルトデスクトップ)
- xfce デスクトップバージョン (技術サポートなし)

## 12.2 Ubuntu Base とは

Ubuntu は、さまざまな CPU アーキテクチャに対応する Ubuntu base ルートファイルシステムを提供しています。現在提供されているアーキテクチャには、amd64、arm64、armhf、i386、s390x、ppc64 があります。

Ubuntu Base は、特定のニーズに合わせたカスタムイメージを作成するための最小限の rootfs で、Ubuntu が動作する最小限の環境です。

Ubuntu Base のダウンロードアドレスはこちらです: <http://cdimage.ubuntu.com/ubuntu-base/releases>

	ubuntu-base-20.04.1-base-amd64.tar.gz.zsync	2020-08-06 15:19	93K
	ubuntu-base-20.04.1-base-arm64.tar.gz	2020-07-31 16:51	25M
	ubuntu-base-20.04.1-base-arm64.tar.gz.zsync	2020-08-06 15:19	88K
	ubuntu-base-20.04.1-base-armhf.tar.gz	2020-07-31 17:09	22M
	ubuntu-base-20.04.1-base-armhf.tar.gz.zsync	2020-08-06 15:19	78K
	ubuntu-base-20.04.1-base-ppc64el.tar.gz	2020-07-31 17:09	32M
	ubuntu-base-20.04.1-base-ppc64el.tar.gz.zsync	2020-08-06 15:19	111K
	ubuntu-base-20.04.1-base-s390x.tar.gz	2020-07-31 16:34	26M
	ubuntu-base-20.04.1-base-s390x.tar.gz.zsync	2020-08-06 15:19	90K
	ubuntu-base-20.04.2-base-amd64.tar.gz	2021-02-01 11:15	26M
	ubuntu-base-20.04.2-base-amd64.tar.gz.zsync	2021-02-04 17:39	93K
	ubuntu-base-20.04.2-base-arm64.tar.gz	2021-02-01 11:30	25M
	ubuntu-base-20.04.2-base-arm64.tar.gz.zsync	2021-02-04 17:39	88K
	ubuntu-base-20.04.2-base-armhf.tar.gz	2021-02-01 11:32	22M
	ubuntu-base-20.04.2-base-armhf.tar.gz.zsync	2021-02-04 17:39	78K
	ubuntu-base-20.04.2-base-ppc64el.tar.gz	2021-02-01 11:16	32M
	ubuntu-base-20.04.2-base-	2021-02-04 17:39	111K

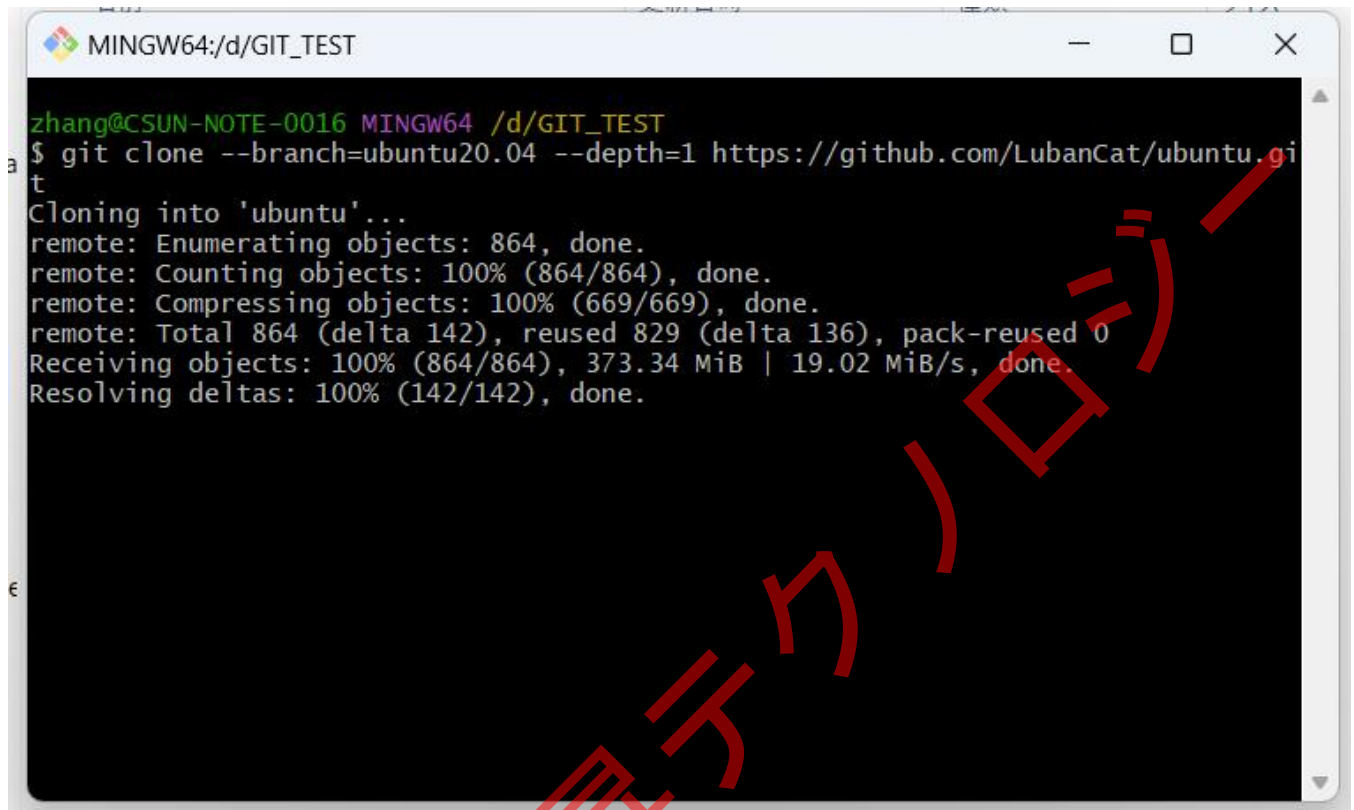
### 12.3 Ubuntu 構築リポジトリの取得

SDKでUbuntu ルートファイルシステムイメージを構築する場合、LubanCat-SDKのルートディレクトリでこの操作を実行する必要があります。

ヒント：完全な Ubuntu ルートファイルシステム構築リポジトリは容量が大きいため、構築したいイメージのバージョンに応じて特定のブランチを取得します。以下のように、Ubuntu20.04のルートファイルシステムを構築する場合は、`--branch=オプション`を使用して選択します。gitリポジトリのコミット履歴を見たくない場合は、`--depth=1`を追加して特定のブランチの最新のコミットを取得することで、リポジトリの

容量を効果的に減らすことができます。

```
1 git clone --branch=ubuntu20.04 --depth=1 https://github.com/LubanCat/ubuntu.git
```



```
MINGW64:/d/GIT_TEST
zhang@CSUN-NOTE-0016 MINGW64 /d/GIT_TEST
$ git clone --branch=ubuntu20.04 --depth=1 https://github.com/LubanCat/ubuntu.git
Cloning into 'ubuntu'...
remote: Enumerating objects: 864, done.
remote: Counting objects: 100% (864/864), done.
remote: Compressing objects: 100% (669/669), done.
remote: Total 864 (delta 142), reused 829 (delta 136), pack-reused 0
Receiving objects: 100% (864/864), 373.34 MiB | 19.02 MiB/s, done.
Resolving deltas: 100% (142/142), done.
```

画像では、社内の Git サーバーを使用していますが、GitHub にもファイルを同期しています。

取得後、ubuntu ディレクトリには以下のファイルが含まれています：

```
1 ls -hgG
2
3 -rwxrwxr-x 1 765 9 月 29 2022 ch-mount.sh
4 -rwxrwxr-x 1 7.6K 4 月 18 10:18 mk-base-ubuntu.sh
5 -rwxrwxr-x 1 841 4 月 12 14:16 mk-image.sh
6 -rwxrwxr-x 1 11K 4 月 18 10:18 mk-ubuntu-rootfs.sh
7 drwxrwxr-x 5 39 3 月 31 16:41 overlay
8 drwxrwxr-x 5 49 9 月 29 2022 overlay-debug
9 drwxrwxr-x 3 17 9 月 29 2022 overlay-firmware
10 drwxrwxr-x 3 19 3 月 31 16:41 packages
11 -rwxrwxr-x 1 1.8K 9 月 29 2022 post-build.sh
12 -rw-rw-r-- 1 1.1K 4 月 18 11:10 readme.md
13 -rw-rw-r-- 1 962 9 月 29 2022 sources.list
14 drwxrwxr-x 3 58 9 月 29 2022 ubuntu-build-service
```

- mk-base-ubuntu.sh: Ubuntu Base 上にソフトウェアパッケージをインストールし、基本的な ubuntu

ルートファイルシステムを構築します。

- mk-ubuntu-rootfs.sh: 基本的な lite バージョンのルートファイルシステム上に Rockchip overlay 層を追加します。
- mk-image.sh: ルートファイルシステムを img イメージファイルにパッケージングします。
- sources.list: ソフトウェアソースのアドレス。
- overlay: Rockchip overlay 層、主に rootfs 内の設定ファイルです。
- overlay-debug: Rockchip overlay 層、主にデバッグスクリプトとツールです。
- overlay-firmware: Rockchip overlay 層、主に wifi/bt/npu のファームウェアです。
- packages: ハードウェアアクセラレーションパッケージ。
- ubuntu-build-service: 構築環境の依存ファイル。

現在、構築スクリプトは 5 つのバージョンのイメージ構築をサポートしています：

- lite: デスクトップなし、端末版。
- xfce: xfce パッケージを使用したデスクトップ版。
- xfce-full: xfce パッケージ+より多くの推奨ソフトウェアパッケージを使用したデスクトップ版。
- gnome: gnome パッケージを使用したデスクトップ版。
- gnome-full: gnome パッケージ+より多くの推奨ソフトウェアパッケージを使用したデスクトップ版。

注意: gnome デスクトップはより多くのリソースを消費するため、RK358x プロセッサのボードで gnome イメージを使用することは推奨されません。

## 12.4 Ubuntu ルートファイルシステムの構築プロセス

Ubuntu ルートファイルシステムの構築は主に 4 つのステップに分かれています：

第一段階：Ubuntu 公式サイトから該当バージョンの最小 rootfs、つまり Ubuntu Base をダウンロードします。

第二段階：最小 rootfs の基盤上に一般的なソフトウェアパッケージやツールをインストールし、デスクトップ版かどうかに応じてデスクトップ表示スイートを追加するか決定します。このプロセスが完了すると、

基礎となるルートファイルシステムが得られます。

第三段階：第二段階の基礎の上に、RK プロセッサに基づく機能強化パッケージ、例えば GPU ドライバーやハードウェアファームウェアなどをさらに追加します。

第四段階：構築したルートファイルシステムを img 形式にパッケージングし、書き込みや次のステップの処理に便利な形式にします。

第 1 段階と第 2 段階は mk-base-ubuntu.sh スクリプトを使用して実現されます；

第 3 段階は mk-ubuntu-rootfs.sh を使用して実現されます；

第 4 段階は mk-image.sh を使用して実現され、第 3 段階のスクリプトの最後に自動的に呼び出されます。

## 12.5 構築環境の構築

ubuntu ディレクトリの下で以下のコマンドを実行します。

```
1 sudo apt-get install binfmt-support qemu-user-static
2 sudo dpkg -i ubuntu-build-service/packages/*
3 sudo apt-get install -f
```

```
cat@lubancat:~/LubanCat_SDK/debian$ sudo apt-get install binfmt-support qemu-user-static
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
binfmt-support is already the newest version (2.2.1-1+deb11u1).
qemu-user-static is already the newest version (1:5.2+dfsg-11+deb11u3).
The following packages were automatically installed and are no longer required:
  liba52-0.7.4 libdca0
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 19 not upgraded.
cat@lubancat:~/LubanCat_SDK/debian$ sudo dpkg -i ubuntu-build-service/packages/*
(Reading database ... 115861 files and directories currently installed.)
Preparing to unpack ../debootstrap_1.0.123_all.deb ...
Unpacking debootstrap (1.0.123) over (1.0.123) ...
Preparing to unpack ../live-build_20210902_all.deb ...
Unpacking live-build (1:20210902) over (1:20210902) ...
```

```
cat@lubancat:~/LubanCat_SDK/debian$ sudo apt-get install -f
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  liba52-0.7.4 libdca0
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 19 not upgraded.
```

コマンド実行中に警告やエラーが出ることがありますが、これは正常な現象であり、エラーは無視してか

まいません。

## 12.6 base イメージの構築

ubuntu-base イメージを基盤として、独自の base イメージを構築します。

ここで言う独自の base イメージとは、ubuntu-base イメージを基盤とし、特定のソフトウェアパッケージをインストールし、ユーザー名、パスワード、ユーザーグループ、タイムゾーンなどの基本設定を施したルートファイルシステムを指します。

理論上、このルートファイルシステムは既にボード上で動作可能ですが、ネットワークや表示など、ボードに特化した設定が追加されていないため、コアサービスのみが実行可能です。

以下は具体的な構築プロセスです：

### 12.6.1 基本ルートファイルシステムの構築

ubuntu ディレクトリの下で以下のコマンドを実行します。

```
1 ./mk-base-ubuntu.sh
```

構築したい Ubuntu バージョンを選択します。ここでは xfce バージョンを選択し、2 を入力して Enter キーを押し、プロンプトに従ってユーザーパスワードを入力します。

```
1 -----
2 please enter TARGET version number:
3 構築したいルートファイルシステムのバージョンを入力してください:
4 [0] Exit Menu
5 [1] gnome
6 [2] xfce
7 [3] lite
8 [4] gnome-full
9 [5] xfce-full
10 -----
11 2
```

コマンドが終了した後、ubuntu ディレクトリの下でのファイル変化を確認します：

```
1 drwxr-xr-x 19 4.0K 4 月 19 16:13 binary
2 -rwxrwxr-x 1 765 9 月 29 2022 ch-mount.sh
```

```
3 -rwxrwxr-x 1 7.6K 4 月 18 10:18 mk-base-ubuntu.sh
4 -rwxrwxr-x 1 841 4 月 12 14:16 mk-image.sh
5 -rwxrwxr-x 1 11K 4 月 18 10:18 mk-ubuntu-rootfs.sh
6 drwxrwxr-x 5 39 3 月 31 16:41 overlay
7 drwxrwxr-x 5 49 9 月 29 2022 overlay-debug
8 drwxrwxr-x 3 17 9 月 29 2022 overlay-firmware
9 drwxrwxr-x 3 19 3 月 31 16:41 packages
10 -rwxrwxr-x 1 1.8K 9 月 29 2022 post-build.sh
11 -rw-rw-r-- 1 1.1K 4 月 18 11:10 readme.md
12 drwxrwxr-x 2 6 4 月 19 16:13 rootfs
13 -rw-rw-r-- 1 962 9 月 29 2022 sources.list
14 -rw-rw-r-- 1 26M 8 月 30 2022 ubuntu-base-20.04.5-base-arm64.tar.gz
15 -rw-r--r-- 1 756M 4 月 13 16:42 ubuntu-base-xfce-arm64-20230413.tar.gz
16 drwxrwxr-x 3 58 9 月 29 2022 ubuntu-build-service
```

以下の3つの新しいファイルが追加されました。

- binary：解凍後のルートファイルシステムを格納し、ビルドプロセスはこのフォルダ内で行われます。
- ubuntu-base-20.04.5-base-arm64.tar.gz：cdimage.ubuntu.com からダウンロードした ubuntu-base のルートファイルシステムの圧縮パッケージ。
- ubuntu-base-xfce-arm64-20230413.tar.gz：上記のスクリプトを使用して構築したベース版のルートファイルシステムの圧縮パッケージ。

上記のコマンドは、./mk-base-ubuntu.sh スクリプトを使用してルートファイルシステムを構築するためのものです。その具体的なワークフローは以下の通りです：

- 指定されたバージョンの ubuntu-base のルートファイルシステムの圧縮パッケージをダウンロードして

解凍する

- 解凍後のルートファイルシステムにソフトウェアソース、DNS サービスを追加する
- ARM アーキテクチャにエミュレータを追加する
- chroot を使用してルートファイルシステムを変更する
  - システムソフトウェアパッケージをアップグレードしてインストールする
  - ユーザーとパスワードを作成し、権限を設定する
  - ホスト名、タイムゾーン、サービス項目の設定などを行う
  - ソフトウェアのインストールキャッシュをクリアする
- 設定されたルートファイルシステムを圧縮パッケージとしてパッケージングして保存管理しやすくする

以上のステップを経て、基本版のルートファイルシステムが作成されます。

## 12.7 Ubuntu ルートファイルシステムイメージのフルビルド

フルバージョンの Ubuntu ルートファイルシステムイメージは主に Rockchip overlay 層を追加しており、これには主にいくつかの設定ファイルとファームウェアが含まれており、ルートファイルシステム内の既存の設定ファイルを追加または置き換えて、望むカスタマイズ効果を達成するために使用されます。

### 12.7.1 ルートファイルシステムのビルド

ubuntu ディレクトリの下で、ビルドしたいルートファイルシステムのバージョンに基づいて

```
1 # フルルートファイルシステムのビルド
2 ./mk-ubuntu-rootfs.sh
```

最初にビルドしたい CPU モデルを選択します。ここでは rk3588/rk3588 を選択し、1 を入力してエンターキーを押します。次にビルドしたい Ubuntu のバージョンを選択します。ここでは xfce バージョンを選択し、2 を入力して Enter キーを押します。プロンプトに従ってユーザーパスワードを入力します。



```
1 jiawen@dev120:~/RK358X_LINUX_SDK/ubuntu$ ./mk-ubuntu-rootfs.sh
2 -----
3 please enter soc number:
4 CPU のビルド番号を入力してください:
5 [0] Exit Menu
6 [1] rk3588/rk3588
7 [2] rk3588/rk3588s
8 -----
9 1
10 SOC を rk358
10 SOC を rk358x に設定.....
11 -----
12 ターゲットバージョン番号を入力してください:
13 構築するルートファイルシステムのバージョンを入力してください:
14 [0] メニューを終了
15 [1] gnome
16 [2] xfce
17 [3] lite
18 [4] gnome-full
19 [5] xfce-full
20 -----
21 2
```

上記のコマンドでのスクリプトは最後に./mk-image.shスクリプトを呼び出し、binaryフォルダ内のファイ

ルを img イメージに自動的にパッケージングします。

パッケージングプロセスが終了した後、ubuntu ディレクトリのファイルを確認してみましょう。

```
1 drwxr-xr-x 19 4.0K 4 月 19 16:13 binary
2 -rwxrwxr-x 1 765 9 月 29 2022 ch-mount.sh
3 -rwxrwxr-x 1 7.6K 4 月 18 10:18 mk-base-ubuntu.sh
4 -rwxrwxr-x 1 841 4 月 12 14:16 mk-image.sh
5 -rwxrwxr-x 1 11K 4 月 18 10:18 mk-ubuntu-rootfs.sh
6 drwxrwxr-x 5 39 3 月 31 16:41 overlay
7 drwxrwxr-x 5 49 9 月 29 2022 overlay-debug
8 drwxrwxr-x 3 17 9 月 29 2022 overlay-firmware
9 drwxrwxr-x 3 19 3 月 31 16:41 packages
10 -rwxrwxr-x 1 1.8K 9 月 29 2022 post-build.sh
11 -rw-rw-r-- 1 1.1K 4 月 18 11:10 readme.md
12 drwxrwxr-x 2 6 4 月 19 16:13 rootfs
13 -rw-rw-r-- 1 962 9 月 29 2022 sources.list
14 -rw-rw-r-- 1 26M 8 月 30 2022 ubuntu-base-20.04.5-base-arm64.tar.gz
15 -rw-r--r-- 1 756M 4 月 13 16:42 ubuntu-base-xfce-arm64-20230413.tar.gz
16 drwxrwxr-x 3 58 9 月 29 2022 ubuntu-build-service
17 -rw-rw-r-- 1 3.5G 4 月 19 15:55 ubuntu-xfce-rootfs.img
```

新しく 3 つのファイルが追加されました。

- ubuntu-xfce-rootfs.img : パッケージングされた完全なルートファイルシステムイメージ
- rootfs : mk-image.sh スクリプトによってパッケージングされた一時ディレクトリ

mk-ubuntu-rootfs.sh スクリプトの構築プロセスは以下の通りです :

- binary ディレクトリを空にし、対応する base ルートファイルシステム圧縮パッケージを解凍する
- パラメータに基づいて、`packages/overlay/overlay-firmware/overlay-debug` ディレクトリ内のファイルを解凍後のルートファイルシステムにコピーする
- ARM アーキテクチャにエミュレータを追加する
- `chroot` を使用してルートファイルシステムを変更する
  - システムソフトウェアパッケージをアップグレードしてインストールする
  - ローカルの `packages` ディレクトリ内のハードウェアアクセラレーションパッケージをインストールする
  - 上記の SOC パラメータに基づいて対応する GPU ドライバをインストールする
  - ソフトウェアインストールキャッシュをクリアする
- `mk-image.sh` スクリプトを呼び出して `img` イメージをパッケージングする

## 12.8 Ubuntu ルートファイルシステムのカスタマイズ

イメージのサイズ制限のため、提供するカスタマイズ Ubuntu イメージには一部の一般的なソフトウェアが事前にインストールされていますが、ユーザーが開発する際にはさらに多くのソフトウェアを事前にインストールしたり、ルートファイルシステムをさらにカスタマイズする必要があるかもしれません。以下の部分では、ルートファイルシステムの修正について具体的に説明します。

### 12.8.1 事前にインストールするソフトウェアパッケージの追加

事前にインストールするソフトウェアの追加については、異なるバージョンに応じて `mk-ubuntu-rootfs.sh` スクリプトに配置することをお勧めします。これにより、変更後に Rockchip overlay 層を再び追加し、`img` イメージをパッケージングするプロセスを繰り返すだけで、多くの開発時間を節約できます。

たとえば、ルートファイルシステムに `git` と `vim` を事前にインストールしたい場合は、`mk-lite/desktop-rootfs.sh` に以下になります

```
1 export APT_INSTALL="apt-get install -fy --allow-downgrades"  
2 # export APT_INSTALL の下の行に追加する  
3  
4 # 追加する内容は  
5 echo -e "¥033[47;36m ----- LubanCat ----- ¥033[0m"  
6 ${APT_INSTALL} git vim
```

## 12.8.2 外部デバイスファームウェアの追加

無線LANカードのような外部デバイスを使用する場合、そのデバイスのファームウェアをルートファイルシステムに追加する必要があります。そのためには、対応するファームウェアを overlay-firmware/ディレクトリに直接配置し、ルートファイルシステム内のパスに従って保存します。

## 12.8.3 サービス項目及び設定ファイルの追加

特定のサービス項目の設定をカスタマイズしたい場合は、overlay/ディレクトリに該当する設定ファイルを追加できます。ルートファイルシステムを作成する過程で、Rockchip overlay 層を追加する際に、ルートファイルシステム内の既存の設定ファイルを追加または置換し、設定ファイルのカスタマイズを実現します。

ここでは、ネットワーク設定ツール netplan の設定を例にとります。netplan の設定ファイルはルートファイルシステムの /etc/netplan/ディレクトリ内にあり、これに対応するのは overlay/etc/netplan/ディレクトリです。

overlay/etc/netplan/に 01-network-manager-all.yaml というファイルを新しく作成し、ファイル内に以下の内容を追加します：

```
1 network:  
2 renderer: NetworkManager  
3 ethernets:  
4 eth0:  
5 dhcp4: true  
6 eth1:  
7 dhcp4: true
```

このファイルの内容は、netplanを使用してネットワークマネージャーをNetworkManagerに設定し、eth0とeth1両方でdhcpを有効にすることです。

設定ファイルを追加した後、イメージを再構築してボードに焼き込み、起動すると、求めていたネットワーク設定を達成できます。

注意：追加するのがシェルスクリプトの場合は、ファイルを作成した後にファイルの権限を775に変更する必要があります。そうでないと、ルートファイルシステム内で実行できない可能性があります。

## 12.8.4 ルートファイルシステムイメージの再パッケージング

ルートファイルシステムの構築スクリプトを修正した後、ubuntu-xfce/lite-rootfs.imgイメージを再パッケージする必要があります。

- mk-base-ubuntu.sh スクリプトの内容を変更していない場合は、base イメージ部分の再構築は不要です。変更した場合は、基本ルートファイルシステムを再構築する必要があります。

```
1 ./mk-base-ubuntu.sh
```

- mk-ubuntu-rootfs.sh、overlay、overlay-debug、overlay-firmware、packages の内容を変更した場合は、以下のコマンドを実行する必要があります。

```
1 ./mk-ubuntu-rootfs.sh
```

## 12.9 LubanCat-SDK を使用したワンクリック構築

Ubuntu 構築リポジトリの取得と構築環境のセットアップの2つのステップを完了した後、ワンクリック構築コマンドを直接使用することで、提供されたカスタムルートファイルシステムイメージを構築することができます。また、SDK のイメージパッケージング機能を利用して、U-boot やカーネルなどの部分も含めて、完全なシステムイメージを一つにパッケージングすることができます。

### 12.9.1 SDK 設定ファイルの説明

LubanCat ボードの SDK 設定ファイルは device/rockchip/rk358x/ディレクトリにあり、BoardConfig-LubanCat-CPU モデル-システムタイプ-システムバージョン.mk という命名規則で保存されています。

Ubuntu ルートファイルシステムの設定ファイルについて、BoardConfig-LubanCat-RK3588-ubuntu-xfce.mk を例にとって、Ubuntu ルートファイルに関連する設定を主に説明します。

```
1 # SOC
2 export RK_SOC=rk358x
3
4 # build.sh save の際の名前
5 export RK_PKG_NAME=lubancat-${RK_UBOOT_DEFCONFIG}
6
7 # デフォルトの rootfs を ubuntu と定義
8 export RK_ROOTFS_SYSTEM=ubuntu
9
10 # デフォルトの Ubuntu バージョン
```

```
11 export RK_UBUNTU_VERSION=20.04
12
13 # デフォルトの rootfs がデスクトップ版の xfce か、ライト版か、コンソール版か、デスクトップ版+推
奨ソフトウェアパッケージかを定義
14 export RK_ROOTFS_TARGET=xfce
15
16 # デフォルトの rootfs に DEBUG ツールを追加するかどうかを定義： debug で追加、none で追加しな
い
17 export RK_ROOTFS_DEBUG=debug
```

- RK\_SOC: SOC タイプを定義
- RK\_PKG\_NAME: イメージパッケージング時の名前を定義
- RK\_ROOTFS\_SYSTEM: ルートファイルシステムのタイプを定義
- RK\_UBUNTU\_VERSION: Ubuntu のリリースバージョンを定義、通常は変更不要でパッケージング時に使用
- RK\_ROOTFS\_TARGET: ルートファイルシステムのバージョンを定義
- RK\_ROOTFS\_DEBUG: rockchip overlay\_debug を追加するかどうかを定義

## 12.9.2 build.sh の自動構築スクリプト

Ubuntu ルートファイルシステムのワンクリック構築機能は、主に build.sh スクリプトの以下の関数によって実現されます。

```
1 function build_ubuntu(){
2 ARCH=${RK_UBUNTU_ARCH:-${RK_ARCH}}
3 case $ARCH in
```

```
4 arm|armhf) ARCH=armhf ;;
5 *) ARCH=arm64 ;;
6 esac
7
8 echo "=====Start building ubuntu for $ARCH=====
9 echo "=="RK_ROOTFS_DEBUG:$RK_ROOTFS_DEBUG RK_ROOTFS_TARGET:$RK_ROOTFS_TARGET=="
10 cd ubuntu
11
12
13 if [ ! -e ubuntu-$RK_ROOTFS_TARGET-rootfs.img ]; then
14 echo "[ No ubuntu-$RK_ROOTFS_TARGET-rootfs.img, Run Make Ubuntu Scripts ]"
15 if [ ! -e ubuntu-base-$RK_ROOTFS_TARGET-$ARCH-*.tar.gz ]; then
16 ARCH=arm64 TARGET=$RK_ROOTFS_TARGET ./mk-base-ubuntu.sh
17 fi
18
19 VERSION=$RK_ROOTFS_DEBUG TARGET=$RK_ROOTFS_TARGET ARCH=$ARCH
SOC=$RK_SOC ./mk-ubuntu-rootfs.sh
20 else
21 echo "[ Already Exists IMG, Skip Make Ubuntu Scripts ]"
22 echo "[ Delate Ubuntu-$RK_ROOTFS_TARGET-rootfs.img To Rebuild Ubuntu IMG ]"
23 fi
24
25 finish_build
26 }
```

作業の流れは以下の通りです。

- echo コマンドを使用して関連する設定情報を出力します。
- ubuntu-\$RK\_ROOTFS\_TARGET-rootfs.img が存在するかどうかを判断します

(\$RK\_ROOTFS\_TARGET は設定ファイルでデスクトップ版かどうかを定義)。存在する場合は構築プロセスをスキップし、存在しない場合は構築コマンドを実行します。ルートファイルシステムの構築には時間がかかりますので、頻繁に構築したり、既に構築されたルートファイルシステムイメージを使用したい場合があります。

- base イメージの構築が必要かどうかを判断します。存在しない場合は base イメージを構築し、存在する場合は base イメージの構築プロセスをスキップします。通常、初回構築以外では base イメージを変更することはありません。
- base イメージを基に、rockchip overlay を追加します。



- イメージをパッケージングします。

### 12.9.3 コンパイル前の準備

コンパイルを開始する前に、まず SDK の設定ファイルがコンパイルしようとしている rootfs と一致していることを確認する必要があります。現在の設定ファイルがコンパイルしようとしている rootfs と一致していない場合は、設定ファイルを切り替える必要があります。

```
1 # SDK 設定ファイルを直接指定して選択
2 ./build.sh BoardConfig-xxx-debian-バージョン.mk
3
4 # SDK 設定ファイルを番号で選択
5 ./build.sh lunch
```

正しい設定ファイルを設定した後、次の構築作業に進むことができます。

### 12.9.4 rootfs の単独構築とパッケージング

以下のコマンドを使用して Ubuntu の rootfs を構築します。

```
1 # Ubuntu を構築
2 ./build.sh ubuntu
```

コンパイルされた rootfs イメージは `ubuntu/ubuntu-$RK_ROOTFS_TARGET-rootfs.img` となり、`rockdev/rootfs.ext4` にシンボリックリンクされます。

注意: `ubuntu/ubuntu-$RK_ROOTFS_TARGET-rootfs.img` が存在しない場合のみ、Ubuntu の rootfs を再構築します。Ubuntu の rootfs の設定ファイルや構築スクリプトを変更した場合は、

`ubuntu/ubuntu-$RK_ROOTFS_TARGET-rootfs.img` を手動で削除してから再構築してください。

構築が完了したら、独立したパーティションテーブル、`boot.img`、`uboot.img` などのパーティションイメージを一つの完全なイメージにパッケージングできます。

次の操作を実行する前に、U-Boot のコンパイル、カーネルのコンパイル、そして直前に完成した rootfs の単独構築が完了していることを確認してください。

準備ができたら、以下のコマンドを実行します。

```
1 # ファームウェアをパッケージング
2 ./mkfirmware.sh
3
4 # update.img を生成
5 ./build.sh updateimg
```

パッケージングが完了すると、生成されたイメージは rockdev/update.img となり、これを書き込みツールを使用してボードの eMMC または SD カードに焼くことができます。

### 12.9.5 ワンクリックで完全なイメージを構築

正しい設定ファイルを設定した後、以下のコマンドを実行することで、U-Boot、Kernel、rootfs のコンパイルを一括で行い、update.img イメージを生成できます。

```
1 # ワンクリックコンパイル
2 ./build.sh
```

パッケージングが完了すると、生成されたイメージは rockdev/update.img となり、これを書き込みツールを使用してボードの eMMC または SD カードに焼くことができます。

## 第13章 Docker を使用したルートファイルシステムの構築

SDK は Ubuntu20.04 を標準開発環境として使用していますが、Ubuntu の異なるバージョンでのイメージ構築環境が異なるため、異なるバージョンのルートファイルシステムを構築する際には、新しい構築環境

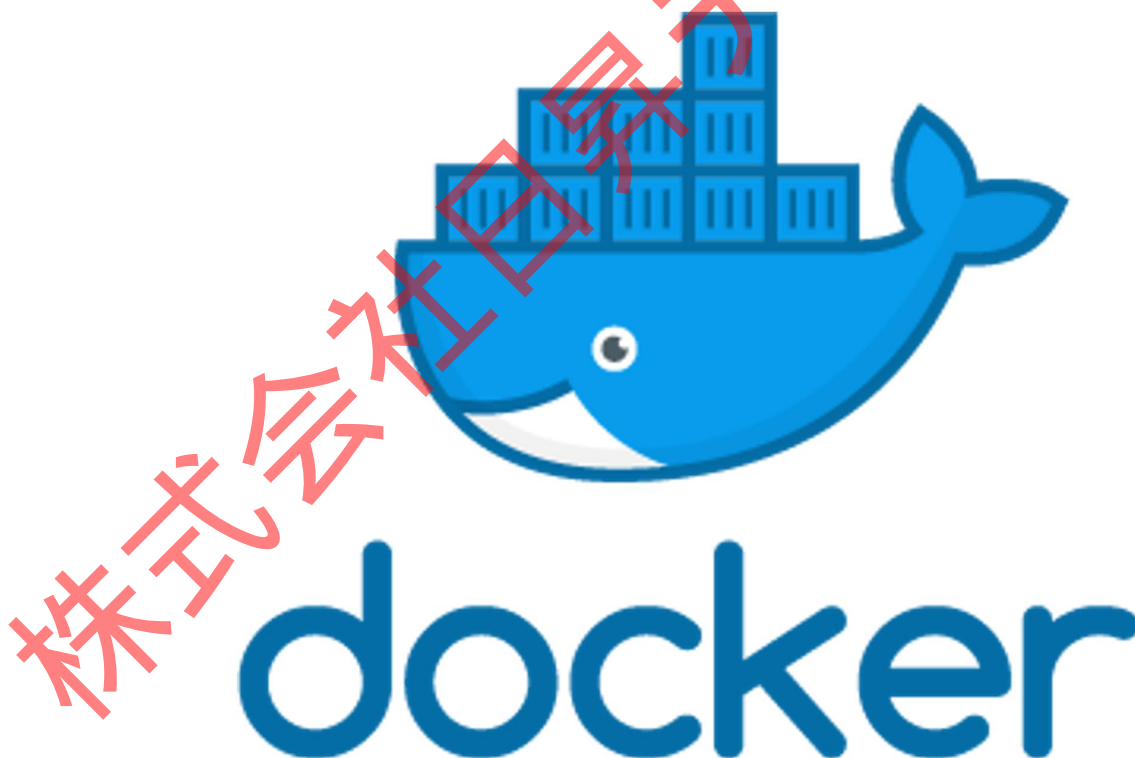
を頻繁にインストールする必要があります。これは多くの時間を浪費し、構築失敗のリスクも高まります。

この問題を解決するために、Docker を使用してルートファイルシステムの構築環境を隔離します。

## 13.1 Docker とは

Docker は、開発者がアプリケーションとその依存関係をポータブルなコンテナにパッケージ化し、任意の人気のある Linux または Windows のオペレーティングシステム上のマシンに配布できるようにするオープンソースのアプリケーションコンテナエンジンです。コンテナは完全にサンドボックス化されており、互いにインターフェースを持ちません。

簡単に言えば、Docker はホスト環境と隔離された小さな仮想化システムとして機能します。下記は Docker のロゴで、多くのコンテナを運ぶ小さな鯨を描いており、Docker の機能を生動的に表現しています。



Docker の公式ウェブサイトは <https://www.docker.com/> です。

## 13.2 Docker のインストール

異なるオペレーティングシステムで Docker をインストールする方法は異なりますが、主に Ubuntu オペレーティングシステムを使用しています。他のシステムでのインストール方法は、公式ドキュメント

<https://docs.docker.com/engine/install/>を参照してください。

Ubuntu 仮想マシン上で Docker Engine をインストールするだけで十分です。具体的なインストールプロセスは以下の通りです。

### 13.2.1 旧バージョンのアンインストール

新しいバージョンをインストールする前に、旧バージョンをアンインストールします。

```
1 sudo apt-get remove docker docker-engine docker.io containerd runc
```

### 13.2.2 リポジトリからのインストール

新しいホストに Docker Engine を初めてインストールする前に、Docker リポジトリを設定する必要があります。その後、リポジトリから Docker をインストールおよび更新できます。

#### 13.2.2.1 リポジトリの設定

- apt パッケージインデックスを更新し、apt が HTTPS を介して Docker リポジトリを使用できるように関連するソフトウェアパッケージをインストールします。

```
1 sudo apt-get update
2 sudo apt-get install ca-certificates curl gnupg lsb-release
```

- Docker の公式 GPG キーを追加します：

```
1 sudo mkdir -p /etc/apt/keyrings
2 curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
```

- リポジトリを設定します：

```
1 echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]  
  
https://download.docker.com/linux/ubuntu ¥  
  
2 $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

### 13.2.2.2 Docker Engine のインストール

```
1 sudo apt-get update  
  
2 sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

### 13.2.2.3 インストールの確認

以下のコマンドを使用してバージョン番号を出力し、それが表示されればインストール成功です。

```
1 sudo docker -v  
  
2  
  
3 # バージョン番号の出力  
  
4 Docker version 20.10.22, build 3a2c30b
```

## 13.3 ローカル Docker イメージの作成

Docker のイメージサーバーは海外にあるため、アクセス速度が遅いです。そこで、ローカルの Dockerfile を使用して Docker イメージを作成します。

ローカルの Dockerfile は、Ubuntu のルートファイルシステム構築用スクリプトリポジトリの ubuntu22.04 ブランチに保存されています。該当するブランチをクローンします。

```
1 git clone --branch=ubuntu22.04 --depth=1 https://github.com/LubanCat/ubuntu.git
```

その後、クローンした ubuntu ディレクトリに移動し、docker build コマンドを使用して Docker イメージを構築します。build\_roofs:2204 は設定したイメージ名です。

```
1 cd ubuntu/Docker  
2  
3 sudo docker build -t build_rootfs:2204 .
```

```
jiawen@dev120:~/RK356X_LINUX_SDK/ubuntu/Docker$ sudo docker build -t build_rootfs:2204 .  
Sending build context to Docker daemon 2.56kB  
Step 1/6 : FROM ubuntu:22.04  
----> 27941809078c  
Step 2/6 : MAINTAINER hejiawen  
----> Running in 2e0d07e90994  
Removing intermediate container 2e0d07e90994  
----> 69a4fe34ae51  
Step 3/6 : RUN sed -i -r 's#http://(archive|security).ubuntu.com#http://mirror.s.tuna.tsinghua.edu.cn#g' /etc/apt/sources.list && ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime && echo 'Asia/Shanghai' >/etc/timezone  
----> Running in 2627f4ec1467  
Removing intermediate container 2627f4ec1467  
----> 1cd4785a7aa8  
Step 4/6 : RUN apt update && apt install --no-install-recommends -y locales apt-utils binfmt-support qemu-user-static make sudo cpio bzip2 curl wget language-selector-common && apt-get autoremove && apt-get clean && rm -rf /var/lib/apt/lists/*  
----> Running in a81aa91cea2e
```

```
Setting up libpython3-stdlib:amd64 (3.10.6-1~22.04) ...  
Setting up python3.10 (3.10.6-1~22.04.2) ...  
Setting up python3 (3.10.6-1~22.04) ...  
running python rtupdate hooks for python3.10...  
running python post-rtupdate hooks for python3.10...  
Setting up python3-dbus (1.2.18-3build1) ...  
Setting up python3-apt (2.3.0ubuntu2.1) ...  
Setting up language-selector-common (0.219) ...  
Processing triggers for libc-bin (2.35-0ubuntu3) ...  
Processing triggers for dbus (1.12.20-2ubuntu4.1) ...  
Reading package lists...  
Building dependency tree...  
Reading state information...  
0 upgraded, 0 newly installed, 0 to remove and 24 not upgraded.  
Removing intermediate container a81aa91cea2e  
----> 0f6b83b7a84c  
Step 5/6 : RUN localedef -c -f UTF-8 -i zh_CN zh_CN.utf8  
----> Running in 876c9f4a2b56  
Removing intermediate container 876c9f4a2b56  
----> 50f9fe138cf8  
Step 6/6 : ENV LANG zh_CN.utf8  
----> Running in 847abf762ffc  
Removing intermediate container 847abf762ffc  
----> 4fa5fe7845df  
Successfully built 4fa5fe7845df  
Successfully tagged build_rootfs:2204
```

Docker イメージの構築が完了した後、以下のコマンドで構築された Docker イメージを確認します。

```
jiawen@dev120:~/RK356X_LINUX_SDK/ubuntu/Docker$ sudo docker images
REPOSITORY          TAG         IMAGE ID      CREATED       SIZE
build_rootfs        2204       4fa5fe7845df 2 minutes ago 278MB
debos-radxa         1          74091305febd 7 months ago 2.19GB
ubuntu              22.04     27941809078c 7 months ago 77.8MB
ubuntu              18.04     ad080923604a 7 months ago 63.1MB
debian              testing    46d3de8d9f7e 7 months ago 119MB
debian              buster    354ff99d6bff 7 months ago 114MB
ghcr.io/kendryte/k510_env latest     e959ea9429f9 9 months ago 809MB
hello-world         latest     feb5d9fea6a5 15 months ago 13.3kB
jiawen@dev120:~/RK356X_LINUX_SDK/ubuntu/Docker$
```

イメージ名が build\_rootfs で、その TAG が 2204 であることがわかります。これは設定したものと一致します。イメージ ID は 4fa5fe7845df で、サイズは 278M です。

## 13.4 Docker コンテナの作成

docker run コマンドを実行して Docker イメージを起動し、SDK を Docker イメージが動作するコンテナにマウントします。作業ディレクトリに入ります。

```
1 sudo docker run --name build_rootfs -it --privileged -v /home/jiawen/RK358X_LINUX_SDK:/works
build_rootfs:2204 /bin/bash
2
3 cd works/
```

- docker run : コンテナを実行します。
- --name build\_rootfs : コンテナの名前を設定します。
- -it : インタラクティブモードでコンテナを実行します。
- --privileged : 特権モードでコンテナを実行します。
- -v /home/jiawen/RK358X\_LINUX\_SDK:/works : コンテナにマウントするディレクトリを設定します。コロンの前はホストのパス、後ろはコンテナ内のパスです。ここでは、SDK 全体をコンテナの /works ディレクトリにマウントします。
- build\_rootfs:2204 : 使用するコンテナのイメージで、ここではイメージ名を指定しています。イメージ ID を使用することもできます。
- /bin/bash : インタラクティブモードで使用するシェルインタプリタ。

```
jiawen@dev120:~/RK356X_LINUX_SDK/ubuntu/Docker$ sudo docker run --name build_rootfs -it --privileged -v /home/jiawen/RK356X_LINUX_SDK:/works build_rootfs:2204 /bin/bash
root@27ddb0ffab2:/# ls
bin  dev  home  lib32  libx32  mnt  proc  run  srv  tmp  var
boot  etc  lib  lib64  media  opt  root  sbin  sys  usr  works
root@27ddb0ffab2:/# cd works/
root@27ddb0ffab2:/works# ls
app
br.log
buildroot
build.sh
debian
device
```

図からわかるように、コンテナを実行するコマンドを実行した後、jiawen@dev120 から root@27ddb0ffab2 に変わりました。これは、現在コンテナ内で root ユーザーとして動作しており、コンテナ ID が 27ddb0ffab2 であることを意味します。

この時点で現在のターミナルを閉じないでください。閉じると現在のコンテナとの接続が失われます。

### 13.5 ルートファイルシステムの構築

ルートファイルシステムの構築方法は仮想マシン内でのそれと同じです。現在のルートファイルシステム構築スクリプトリポジトリの readme ファイルを確認することができます。ここでは簡単なデモを行います。

```
1 # ubuntu ルートファイルシステム構築スクリプトリポジトリに入る
2 cd /works/ubuntu
3
4 # 関連する構築依存関係をインストール
5 apt-get install binfmt-support qemu-user-static
6 dpkg -i ubuntu-build-service/packages/*
7 apt-get install -f
8
9 # イメージの構築
10 ./mk-base-ubuntu.sh
11 ./mk-ubuntu-rootfs.sh
```



上記のコマンドを実行した後、Ubuntu ルートファイルシステムの構築が完了します。

## 13.6 Docker の関連操作コマンド

### 13.6.1 現在のコンテナから退出

現在のターミナルで `exit` を実行することで、現在のコンテナを停止して退出できます。

### 13.6.2 現在実行中のコンテナを表示

```
1 sudo docker ps
```

`-a` オプションを使用すると、終了したコンテナと実行中のコンテナを含む、すべてのコンテナを表示できます。

### 13.6.3 停止したコンテナを再起動して入る

```
1 sudo docker start build_rootfs
```

停止したコンテナを再起動することができ、コンテナ名またはコンテナ ID を指定できます。

```
1 sudo docker attach 27ddb0ffab2
```

実行中のコンテナに入るには、コンテナ名またはコンテナ ID を指定できます。

### 13.6.4 コンテナの削除

```
1 sudo docker rm build_rootfs
```

コンテナを削除することができ、コンテナ名またはコンテナ ID を指定できます。

`-f` オプションで、停止していないコンテナを強制的に削除できます。

### 13.6.5 イメージの削除

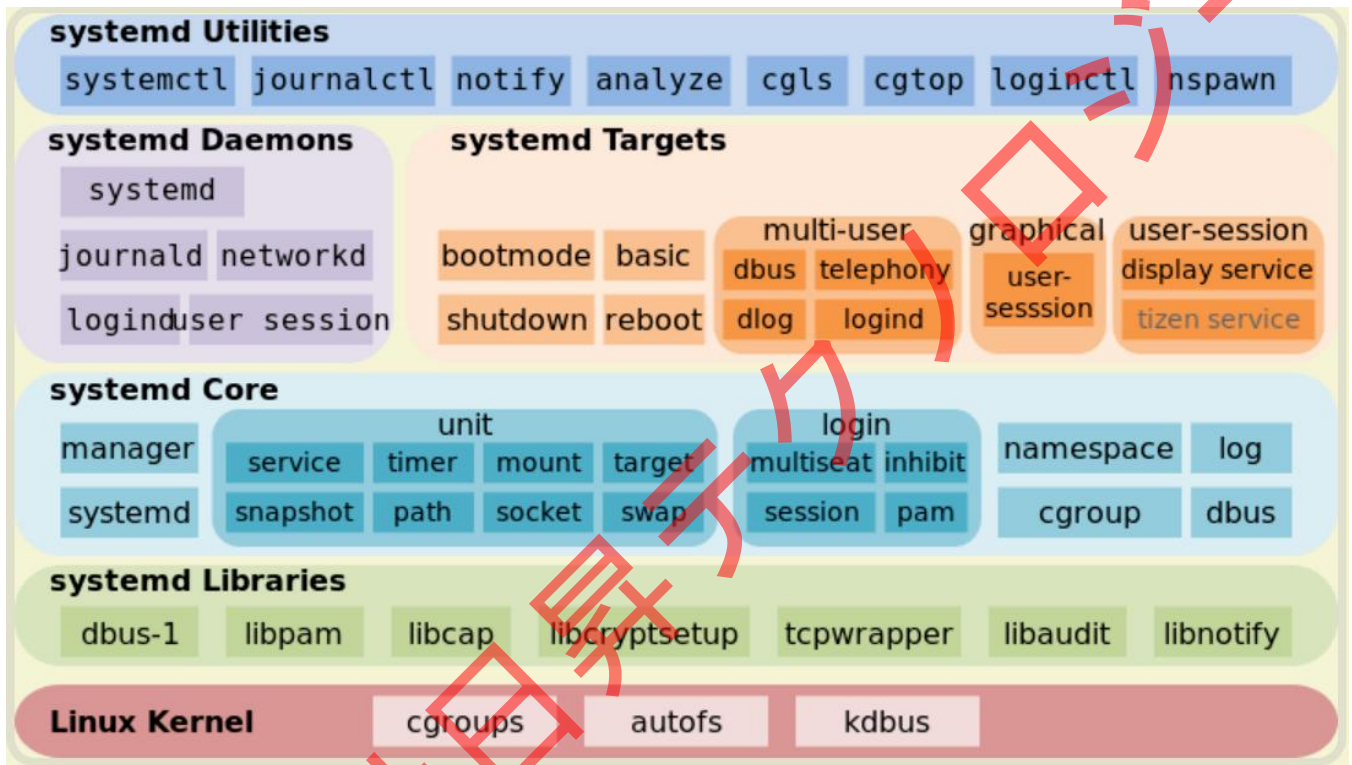
```
1 sudo docker rmi build_rootfs:2204
```

イメージを削除することができ、イメージ名:TAG またはイメージ ID を指定できます。

# 第 14 章 Systemd の探求

## 14.1 システム管理

systemd は単一のコマンドではなく、システムおよびサービスマネージャを提供するコマンドの集まりで、PID 1 として動作し他のプログラムの起動を担当します。以下は systemd のアーキテクチャ図です。



systemd の機能には、タスクの並列処理のサポート、ソケット式および D-Bus バス式でのサービスのアクティベーション、デーモンのオンデマンド起動、Linux の cgroups を利用したプロセスの監視、スナップショットとシステムの復元のサポート、マウントポイントおよび自動マウントポイントの維持、サービス間の依存関係に基づいた精密な制御が含まれます。systemd は SysV および LSB の初期スクリプトをサポートし、sysvinit に取って代わることができます。その他の機能には、ログのプロセス、基本的なシステム設定の管理、ログインユーザーのリストとシステムアカウントの維持、ランタイムディレクトリと設定の管理、コンテナおよび仮想マシンの実行、ネットワークの設定と時間同期、ログの転送と名前解決などの簡単な管理が含まれます。

## 14.2 Systemd の核心概念

### 14.2.1 unit (ユニット)

ユニットファイルは ini スタイルのプレーンテキストファイルで、Systemd がすべてのシステムリソースを管理できます。異なるリソースは Unit として総称されます。これは 12 種類のオブジェクトの情報をカプセル化しています：サービス(service)、ソケット(socket)、デバイス(device)、マウントポイント(mount)、自動マウントポイント(automount)、起動ターゲット(target)、スワップパーティションやファイル(swap)、監視されるパス(path)、タイマー(timer)、リソースコントロールグループ(slice)、外部から作成されたプロセスのグループ(scope)、スナップショット(snapshot)。

- service : バックグラウンドサービスプロセスで、デーモンの起動、停止、再起動、リロードなどの操作をカプセル化します。
- socket : このタイプのユニットは、システムまたはインターネットのソケットをカプセル化します。現在、systemd は AF\_INET、AF\_INET6、AF\_UNIX のストリーム、データグラム、シーケンシャルパケットソケットをサポートしています。各ソケットユニットには、対応するサービスユニットがあり、新しい「接続」がソケットに到達すると、該当するサービスが起動されます（例：nscd.socket が新しい接続を受けると nscd.service が起動される）。
- device : このタイプのユニットは、Linux のデバイスツリーに存在するデバイスをカプセル化します。udev ルールによってマークされた各デバイスは、systemd 内でデバイスユニットとして現れ、デバイス間の依存関係を定義します。
- mount : このタイプのユニットは、ファイルシステムの階層構造におけるマウントポイントをカプセル化します。systemd はこのマウントポイントを監視し管理し、例えば、起動時に自動でマウントしたり、特定の条件下で自動的にアンマウントしたりします。systemd は/etc/fstab のエントリをすべてマウントポイントに変換し、起動時に処理します。

- automount：このタイプのユニットは、システムの階層構造における自動マウントポイントをカプセル化します。各自動マウントユニットは、マウントユニットに対応しており、その自動マウントポイントがアクセスされた時に、systemd は定義されたマウント操作を実行します。
- swap：マウントユニットと同様に、スワップユニットはスワップ領域を管理するために使用されます。ユーザーはスワップユニットを使用してシステム内のスワップ領域を定義し、起動時にそれらをアクティブにすることができます。
- target：このタイプのユニットは、他のユニットを論理的にグループ化するために使用されます。それ自体は実際には何も行わず、他のユニットを参照するだけですが、これによりユニットを一元的に管理することができ、よく知られている実行レベルの概念を実現することができます。例えば、システムをグラフィカルモードにするために多くのサービスと設定コマンドを実行する必要があり、これらはすべて個々のユニットによって表されます。これらすべてのユニットを一つのターゲットに組み合わせることで、ターゲットが表すシステムの実行状態に入るためにこれらすべてのユニットを実行する必要があります（例：multi-user.target は従来の sysv を使用するシステムでの実行レベル 5 に相当します）。
- timer：タイマーユニットは、ユーザー定義の操作を定期的にとりガーするために使用されます。このタイプのユニットは、従来の atd や crond などのタイマーサービスを置き換えます。
- snapshot：ターゲットユニットに似ていますが、スナップショットはシステムの現在の実行状態を保存したユニットのグループです。
- slice：CGroup のツリーを表します。
- path：指定されたディレクトリまたはファイルの変更を監視し、他の Unit の実行をトリガーします。
- scope：システムサービスのグループ情報を記述するために使用されます。

各ユニットにはそれに対応する設定ファイルがあります。たとえば、avahi-daemon サービスには

avahi-daemon.service ファイルが対応しています。この種の設定ファイルの構文は非常にシンプルで、ユー



```
# 起動に失敗したすべてのユニットをリストアップ

$ systemctl list-units --failed

# 種類が service で、実行中のすべてのユニットをリストアップ

$ systemctl list-units --type=service
```

```
root@lubancat:~# systemctl list-units --type=service
UNIT                                LOAD    ACTIVE SUB    JOB    DESCRIPTION
acpid.service                       loaded active running ACPI event daemon
addb.service                         loaded active running addb for Debian
alsa-restore.service               loaded active exited Save/Restore Sound Ca
alsa-state.service                 loaded active running Manage Sound Card Sta
anacron.service                     loaded inactive dead    start Run anacron jobs
binfmt-support.service             loaded active exited Enable support for ad
bluetooth.service                  loaded active running Bluetooth service
console-setup.service              loaded active exited Set console font and
cpufrequtils.service               loaded active exited LSB: set CPUFreq kern
dbus.service                        loaded active running D-Bus System Message
getty@tty1.service                  loaded active running Getty on tty1
ifupdown-pre.service               loaded active exited Helper to synchronize
ifupdown-wait-online.service        loaded active    exited Wait for network to
keyboard-setup.service              loaded active    exited Set the console keybo
lightdm.service                     loaded active running Light Display Manager
loadcpufreq.service                loaded active    exited LSB: Load kernel modu
networking.service                  loaded active    exited Raise network interfa
NetworkManager.service              loaded active running Network Manager
ntp.service                          loaded active running Network Time Service
openvpn.service                     loaded active    exited OpenVPN service
polkit.service                       loaded active running Authorization Manager
rc-local.service                    loaded active    exited /etc/rc.local Compati
lines 1-23
```

```
# 特定のユニットが現在実行中かどうかを表示

$ systemctl is-active systemd-timesyncd.service
```

```
root@lubancat:~# systemctl is-active systemd-timesyncd.service
inactive
root@lubancat:~#
```

```
# 特定のユニットサービスが有効化されているかどうかを表示

$ systemctl is-enabled systemd-timesyncd.service
```

```
root@lubancat:~# systemctl is-enabled systemd-timesyncd.service
enabled
root@lubancat:~#
```

## 14.2.2 ユニットの管理

```
# サービスを直ちに開始
$ sudo systemctl start sshd.service
# サービスを直ちに停止
$ sudo systemctl stop sshd.service
# サービスを再起動
```

```
$ sudo systemctl restart sshd.service
# サービスの全ての子プロセスを終了
$ sudo systemctl kill sshd.service
# サービスの設定ファイルを再読み込み
$ sudo systemctl reload sshd.service
# 修正されたすべての設定ファイルを再読み込み
$ sudo systemctl daemon-reload
# 特定のユニットのすべての下層パラメータを表示
$ systemctl show httpd.service
```

```
root@lubancat:~# systemctl show httpd.service
Restart=no
NotifyAccess=none
RestartUsec=100ms
TimeoutStartUsec=1min 30s
TimeoutStopUsec=1min 30s
RuntimeMaxUsec=infinity
WatchdogUsec=0
WatchdogTimestampMonotonic=0
RootDirectoryStartOnly=no
RemainAfterExit=no
GuessMainPID=yes
MainPID=0
ControlPID=0
FileDescriptorStoreMax=0
NFileDescriptorStore=0
StatusErrno=0
Result=success
UID=[not set]
GID=[not set]
NRestarts=0
ExecMainStartTimestampMonotonic=0
ExecMainExitTimestampMonotonic=0
ExecMainPID=0
lines 1-23
```

# 特定のユニットの指定した属性の値を表示

```
$ systemctl show -p CPUShares avahi-daemon.service
```

### 14.2.3 ユニットの依存関係

systemd は多くの起動作業の依存を解消し、それらが並行して起動できるようにしました。しかし、一部のタスクには、本質的な依存関係が存在し、「ソケットアクティベーション」、「D-Bus アクティベーション」、「autofs」の3つの方法では依存を解消できません。例えば、マウントはファイルシステム内にマウントポイントが作成されるのを待たなければならず、物理デバイスが準備できているのを待ってからマウントする必要があります。このような依存問題を解決するために、systemd のユニット間で相互に依存関係を定義できます。例えば、unit Q が unit W に依存している場合、「require Q」として unit W の定義内に記述でき、これにより systemd は Q を先に起動してから W を起動することを保証します。systemd はトランザクションの整合性を保証することができます。systemd のトランザクション概念は、データベースのそれとは異なり、複数の依存ユニット間で循環参照がないことを主に保証します。循環依存が存在する場合、systemd は任意のサービスを起動することができません。この場合、systemd は問題を解決しようとします。ユニット間の依存関係には、強い依存を示す「requires」と弱い依存を示す「wants」の2種類があり、systemd は「wants」キーワードで指定された依存関係を取り除くことで循環を解消しようとします。修復



不可能な場合、systemd はエラーを報告します。systemd はこのような設定ミスを自動的に検出し修正することができ、管理者の負担を大幅に軽減します。

```
# あるユニットのすべての依存関係をリストアップ
```

```
$ systemctl list-dependencies sshd.service
```

```
root@lubancat:~# systemctl list-dependencies sshd.service
sshd.service
├─ .mount
├─ system.slice
├─ sysinit.target
│   ├── dev-hugepages.mount
│   ├── dev-mqueue.mount
│   ├── keyboard-setup.service
│   ├── kmod-static-nodes.service
│   ├── proc-sys-fs-binfmt_misc.automount
│   ├── resolvconf.service
│   ├── sys-fs-fuse-connections.mount
│   ├── sys-kernel-config.mount
│   ├── sys-kernel-debug.mount
│   ├── systemd-ask-password-console.path
│   ├── systemd-binfmt.service
│   ├── systemd-hwdb-update.service
│   ├── systemd-journal-flush.service
│   ├── systemd-journald.service
│   ├── systemd-machine-id-commit.service
│   ├── systemd-modules-load.service
│   ├── systemd-random-seed.service
│   ├── systemd-sysctl.service
│   └─ systemd-sysusers.service
└─ lines 1-23
```

上記のコマンドの出力結果には、Target タイプの依存が含まれていますが、Target タイプはデフォルトでは展開して表示されません。Target を展開して表示するには、--all オプションを追加します。

```
# あるユニットのすべての依存関係をリストアップし、Target 依存を展開して表示
```

```
$ systemctl list-dependencies --all sshd.service
```

## 14.2.4 ユニットの設定ファイル

systemd の設定ファイルは、ファイルシステムの /etc/systemd/system または /usr/lib/systemd/system ディレクトリ下にデフォルトで保存されます。以下のコマンド「ls -al」でそのディレクトリの内容を確認します。

```
root@lubancat:~# ls -al /etc/systemd/system
total 48
drwxr-xr-x 12 root root 4096 Feb 14 2019 .
drwxr-xr-x  5 root root 4096 Sep 27 16:48 ..
lrwxrwxrwx  1 root root   9 Sep 28 09:48 NetworkManager-wait-online.service -> /dev/n
ull
drwxr-xr-x  2 root root 4096 Sep 27 16:46 bluetooth.target.wants
drwxr-xr-x  2 root root 4096 Feb 14 2019 boot-complete.target.requires
lrwxrwxrwx  1 root root   9 Sep 27 16:45 bootlogs.service -> /dev/null
lrwxrwxrwx  1 root root   9 Sep 27 16:45 bootmisc.service -> /dev/null
lrwxrwxrwx  1 root root   9 Sep 27 16:45 brightness.service -> /dev/null
lrwxrwxrwx  1 root root   9 Sep 27 16:45 checkfs.service -> /dev/null
lrwxrwxrwx  1 root root   9 Sep 27 16:45 checkroot-bootclean.service -> /dev/null
lrwxrwxrwx  1 root root   9 Sep 27 16:45 checkroot.service -> /dev/null
lrwxrwxrwx  1 root root  33 Feb 14 2019 ctrl-alt-del.target -> /lib/systemd/system/r
eboot.target
lrwxrwxrwx  1 root root  42 Sep 27 16:46 dbus-fi.wl.wpa_supplicant1.service -> /lib/s
ystemd/system/wpa_supplicant.service
lrwxrwxrwx  1 root root  37 Sep 27 16:46 dbus-org.bluez.service -> /lib/systemd/syste
m/bluetooth.service
lrwxrwxrwx  1 root root  44 Feb 14 2019 dbus-org.freedesktop.network1.service -> /li
b/systemd/system/systemd-networkd.service
lrwxrwxrwx  1 root root  53 Sep 27 16:47 dbus-org.freedesktop.nm-dispatcher.service -
> /lib/systemd/system/NetworkManager-dispatcher.service
lrwxrwxrwx  1 root root  44 Feb 14 2019 dbus-org.freedesktop.resolve1.service -> /li
b/systemd/system/systemd-resolved.service
lrwxrwxrwx  1 root root  45 Sep 27 16:39 dbus-org.freedesktop.timesync1.service -> /l
ib/systemd/system/systemd-timesyncd.service
lrwxrwxrwx  1 root root  35 Sep 27 16:47 display-manager.service -> /lib/systemd/syst
em/lightdm.service
```

このディレクトリには多くのリンクシンボル「->」があり、これはファイルが「->」で指示されたファイルにリンクされていることを意味します。これは、Windowsのショートカットに似ており、多くのシンボリックリンクが/lib/systemd/systemディレクトリを指していますが、/dev/nullファイルを指しているものもあり、これは空のファイルを意味します。systemdは初期化プロセス中に/etc/systemd/systemディレクトリ内の設定ファイルのみを実行します。systemdプログラムをインストールした後、/lib/systemd/systemディレクトリに自動的にそのプログラムに対応する設定ファイルが生成されます。"systemctl enable xxx.service"の方法でサービスのシンボリックリンクを作成することができます。これにより、システム起動時に自動的に起動するように設定されますが、"systemctl disable"コマンドを使用すると、シンボリックリンクが削除されるため、システム起動時には起動しません。

```
# 特定のユニットがシステム起動時に自動起動するかどうかをチェック
```

```
$ systemctl is-enabled sshd.service
```

```
root@lubancat:~# systemctl is-enabled sshd.service
enabled
root@lubancat:~# █
```

## 14.2.5 ユニットのシステム管理

# システムを再起動（非同期操作）

```
$ systemctl reboot
```

```
root@lubancat:~# systemctl reboot
root@lubancat:~# [ OK ] Closed Load/Save RF Kill Switch Status /dev/rfkill Watch.
Stopping Session c1 of user cat.
Stopping Session c2 of user root.
[ 7589.470863] ttyFIQ ttyFIQ0: tty_port_close_start: tty->count = 1 port count = 2
Stopping Setup zram based device zram0...
[ OK ] Stopped target Sound Card.
Stopping Save/Restore Sound Card State...
[ OK ] Stopped target Remote Encrypted Volumes.
[ OK ] Stopped Daily Cleanup of Temporary Directories.
[ OK ] Stopped target Host and Network Name Lookups.
Stopping Bluetooth service...
Stopping Light Display Manager...
Stopping Disk Manager...
Stopping Daemon for power management...
Stopping OpenBSD Secure Shell server...
Stopping LSB: set CPUFreq kernel parameters...
Stopping triggerhappy global hotkey daemon...
Stopping LSB: layer 2 tunnelling protocol daemon...
Stopping Network Time Service...
[ OK ] Stopped target Login Prompts.
Stopping Serial Getty on ttyFIQ0...
Stopping Getty on tty1...
Stopping System Logging Service...
[ OK ] Stopped OpenVPN service.
Stopping ACPI event daemon...
Stopping Authorization Manager...
```

# システムをシャットダウンし、電源を切る（非同期操作）

```
$ systemctl poweroff
```

# CPU のみを停止し、他のハードウェアは起動状態のままにする（非同期操作）

```
$ systemctl halt
```

# システムをサスペンド（非同期操作）、suspend.target を実行

```
$ systemctl suspend
```

# システムを休止状態にする（非同期操作）、hibernate.target を実行

```
$ systemctl hibernate
```

## 14.2.6 ユニットのログ管理

Systemd はすべてのユニットの起動ログを一元管理します。このおかげで、journalctl コマンド一つですべてのログ（カーネルログとアプリケーションログ）を見ることができます。

ログの設定ファイルは/etc/systemd/journald.conf にあり、保存ディレクトリは/var/log/journal/です。デフォルトではログの最大制限はファイルシステムの容量の 10%ですが、/etc/systemd/journald.conf の SystemMaxUse フィールドでログの最大制限を指定することができます。

注意: /var/log/journal/ディレクトリは systemd ソフトウェアパッケージの一部です。削除された場合、systemd はそれを自動的に再作成しませんが、ソフトウェアパッケージを次にアップグレードするときにディレクトリが再作成されます。このディレクトリがない場合、systemd はログを/run/systemd/journal に書き込みます。これは、システムが再起動するとログが失われることを意味します。

```
# すべてのログを表示
```

```
$ sudo journalctl
```

```
root@lubancat:~# sudo journalctl
-- Logs begin at Tue 2022-10-11 13:43:18 CST, end at Tue 2022-10-11 13:46:08 CST
Oct 11 13:43:18 lubancat kernel: Booting Linux on physical CPU 0x0000000000 [0x4
Oct 11 13:43:18 lubancat kernel: Linux version 4.19.232 (jiawen@dev120.embedfire
Oct 11 13:43:18 lubancat kernel: Machine model: EmbedFire LubanCat2 HDMI
Oct 11 13:43:18 lubancat kernel: earlycon: uart8250 at MMI032 0x00000000fe660000
Oct 11 13:43:18 lubancat kernel: bootconsole [uart8250] enabled
Oct 11 13:43:18 lubancat kernel: cma: Reserved 16 MiB at 0x000000007ec00000
Oct 11 13:43:18 lubancat kernel: On node 0 totalpages: 519680
Oct 11 13:43:18 lubancat kernel: DMA32 zone: 8184 pages used for memmap
Oct 11 13:43:18 lubancat kernel: DMA32 zone: 0 pages reserved
Oct 11 13:43:18 lubancat kernel: DMA32 zone: 519680 pages, LIFO batch:63
Oct 11 13:43:18 lubancat kernel: psci: probing for conduit method from DT.
Oct 11 13:43:18 lubancat kernel: psci: PSCIv1.1 detected in firmware.
Oct 11 13:43:18 lubancat kernel: psci: Using standard PSCI v0.2 function IDs
Oct 11 13:43:18 lubancat kernel: psci: Trusted OS migration not required
Oct 11 13:43:18 lubancat kernel: psci: SMC Calling Convention v1.2
Oct 11 13:43:18 lubancat kernel: percpu: Embedded 24 pages/cpu s57896 r8192 d322
Oct 11 13:43:18 lubancat kernel: pcpu-alloc: s57896 r8192 d32216 u98304 alloc=24
Oct 11 13:43:18 lubancat kernel: pcpu-alloc: [0] 0 [0] 1 [0] 2 [0] 3
Oct 11 13:43:18 lubancat kernel: Detected VIPT I-cache on CPU0
Oct 11 13:43:18 lubancat kernel: CPU features: detected: Virtualization Host Ext
Oct 11 13:43:18 lubancat kernel: CPU features: detected: Speculative Store Bypas
Oct 11 13:43:18 lubancat kernel: Built 1 zonelists, mobility grouping on. Total
lines 1-23
```

```
# ログファイルが占有する最大スペースを指定
```

```
$ sudo journalctl --vacuum-size=8M
```

## 14.3 Systemd の実例分析

### 14.3.1 起動順序と依存関係

前述の通り、サービスの設定ファイルについて説明しました。システムが起動すると、systemd は各サービスの設定ファイルを読み込み、それに基づいて各システムサービスを実行します。設定ファイルはサー

ビスの起動方法を詳細に記述しています。例として SSH サービスを取り上げ、その設定ファイルを詳しく分析します。/etc/systemd/system ディレクトリ内にある sshd.service ファイルは、SSH サービスの起動方法を記述しています。vim.tiny を使用してこの設定ファイルを開くと、以下のように表示されます：

```
[[Unit]]
Description=OpenBSD Secure Shell server
Documentation=man:sshd(8) man:sshd_config(5)
After=network.target auditd.service
ConditionPathExists=!/etc/ssh/sshd_not_to_be_run

[Service]
EnvironmentFile=-/etc/default/ssh
ExecStartPre=/usr/sbin/sshd -t
ExecStart=/usr/sbin/sshd -D $SSHD_OPTS
ExecReload=/usr/sbin/sshd -t
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartPreventExitStatus=255
Type=notify
RuntimeDirectory=sshd
RuntimeDirectoryMode=0755

[Install]
WantedBy=multi-user.target
Alias=sshd.service
~/
"/etc/systemd/system/sshd.service" 22L, 538C 1,1 All
```

設定ファイルには Unit、Service、Install の 3 つのセクションがあり、それぞれが多くのキーバリューペアを含んでいます。

Unit セクションでは、Description が現在のサービスを記述し、Documentation フィールドがドキュメントの場所を示しています。次に重要なのが After フィールドで、これはサービスの起動順序を指定しますが、依存関係には関与しません。対応する Before フィールドもあります。この設定ファイルでは、After はこのサービスが network.target と auditd.service の 2 つのサービスの後に起動することを示しています。Wants と Requires フィールドは依存関係にのみ関与し、起動順序とは無関係です。デフォルトでは同時に起動します。サービス間の依存関係を設定したい場合は、Wants（弱依存）と Requires（強依存）フィールドを使用します。

### 14.3.2 起動コマンド

EnvironmentFile フィールドは現在のサービスの環境パラメータファイルを指定します。等号の後ろの

「-」は、もし/etc/default/ssh ファイルが存在しなくてもエラーが発生しないことを意味します。最も重要

なフィールドは ExecStart で、これはプロセスを起動するために実行するコマンドを定義します。図では「/usr/sbin/sshd -D \$SSHD\_OPTS」というコマンドが実行されます。この中の変数 \$SSHD\_OPTS は、EnvironmentFile フィールドで指定された環境パラメータファイルから来ています。ExecStartPre フィールドはサービスを起動する前に実行するコマンドを表します。

### 14.3.3 起動タイプと振る舞い

1. Type フィールドはサービスの起動タイプを指定し、以下のようなタイプがあります。

- simple (デフォルト) : ExecStart フィールドで起動されたプロセスがメインプロセスになります。
- forking : ExecStart フィールドが fork() 方式で起動し、親プロセスが終了し子プロセスがメインプロセスになります。
- oneshot : simple に似ていますが、一度だけ実行され、Systemd は他のサービスを起動する前にそれが完了するのを待ちます。
- dbus : simple に似ていますが、D-Bus シグナルを待ってから起動します。
- notify : simple に似ていますが、起動が終了した後に通知信号を出し、その後 Systemd が他のサービスを起動します。
- idle : simple に似ていますが、他のタスクがすべて実行された後にサービスを起動します。この使用例の一つは、このサービスの出力が他のサービスの出力と混ざらないようにすることです。

2. KillMode フィールドは、systemd が SSH サービスをどのように停止するかを定義します。この例では process と設定されており、メインプロセスのみを停止し、sshd の子プロセスは停止しません。

3. Restart フィールドは、sshd が終了した後の systemd による再起動の方法を定義します。Restart が on-failure に設定されている場合、予期せぬ失敗が発生した場合にのみ sshd を再起動します。sshd が正常に停止した場合 (例えば systemctl stop コマンドを実行した場合) は再起動しません。

設定可能な値は以下の通りです。

- no (デフォルト値) : 終了後に再起動しません
- on-success : 正常終了時 (終了ステータスコード 0) のみ再起動します
- on-failure : 異常終了時 (終了ステータスコード非0) 、信号による終了やタイムアウトを含む、再起動します
- on-abnormal : 信号による終了やタイムアウトの場合のみ再起動します
- on-abort : キャッチされない信号によって終了した場合のみ再起動します
- on-watchdog : タイムアウトによって終了した場合のみ再起動します
- always : 終了原因に関わらず常に再起動します

### 14.3.4 インストール方式

WantedBy フィールドは、現在のサービスが属するターゲットを表します。ターゲットはサービスグループを意味し、sshd は multi-user.target サービスグループに属しています。"systemctl enable sshd.service" コマンドを実行すると、sshd.service のシンボリックリンクが/etc/systemd/system ディレクトリ内の multi-user.target.wants サブディレクトリに配置されます。

設定ファイルを変更した場合は、設定ファイルを再読み込みしてからサービスを再起動する必要があります。

```
# 設定ファイルを再読み込み
$ sudo systemctl daemon-reload

# 関連サービスを再起動
$ sudo systemctl restart ssh
```

## 14.4 Systemd で自分の Systemd サービスを作成する

自作アプリをシステム起動時に自動起動させたいというニーズがよくあります。Systemd サービスを作成することでこれを実現できます。ここでは、簡単な hello.service サービスを例に、自分の Systemd サービス

の作り方を説明します。

### 14.4.1 スクリプトの作成

/opt ディレクトリに cd で移動し、vim を使用して hello.sh スクリプトを作成します。

```
1 #!/bin/bash
2
3 while true
4 do
5 echo Hello Lubancat >> /tmp/hello.log
6 sleep 3
7 done
```

このスクリプトは、3秒ごとに「Hello Lubancat」という文字列を/tmp/hello.log ファイルに出力する機能を実現します。書き終わったら、hello.sh に実行権限を与えてください。

```
sudo chmod 0755 hello.sh
```

### 14.4.2 設定ファイルの作成

/etc/systemd/system/ディレクトリ下に hello.service 設定ファイルを作成します。内容は以下の通りです。

```
1 [Unit]
2 Description=hello daemon
3
4 [Service]
5 ExecStart=/opt/hello.sh
6 Restart=always
7 Type=simple
```



```
8
9 [Install]
10 WantedBy=multi-user.target
```

ExecStart フィールドでは、hello.service サービスの自動起動スクリプトを/opt/hello.sh と定義しています。hello.service が開始時自動起動機能を有効にした場合、システム起動後に/opt/hello.sh が実行されます。

Restart=always はプロセスまたはサービスが予期せぬ故障で停止した場合に自動的に再起動されるモードを意味します。Type=simple はデフォルトの設定で、省略可能です。WantedBy はサービスの起動を誰が要求するかを指定し、WantedBy=multi-user.target はシステムがマルチユーザーモードになった時にこのサービスを起動順序に追加することを意味します。グラフィカルユーザーインターフェイスの後で起動する必要がある場合は、WantedBy=graphical.target と設定して、システムがグラフィカルユーザーインターフェイスマードに入る時に起動するようにします。

WantedBy パラメータは、サービスがデスクトップ環境を必要とするかどうかにとって非常に重要です。サービスがデスクトップ環境を必要とし、WantedBy=multi-user.target と設定されている場合、デスクトップが起動する前に自分の起動スクリプトを呼び出し、サービスの起動に失敗する可能性があります。

```
1 # デスクトップの前にサービスを起動
2 WantedBy=multi-user.target
3
4
5 # デスクトップの後にサービスを起動
6 WantedBy=graphical.target
```

### 14.4.3 hello.service の自動起動機能を有効化

コマンド `sudo systemctl list-unit-files --type=service | grep hello` を入力して、hello.service がサービスリス

トに追加されているか確認します。

```
sudo systemctl list-unit-files --type=service | grep hello
```

```
root@lubancat:/etc/systemd/system# sudo systemctl list-unit-files --type=service | gre
p hello
hello.service                                disabled
root@lubancat:/etc/systemd/system#
```

hello.service が disable 状態にあることがわかります。上記のコマンドを入力した後に何も表示されない場合は、作成したサービスに問題があり、詳細な確認が必要です。以下のコマンドを入力して、hello.service をシステム起動時に自動起動するように設定します。

```
sudo systemctl enable hello
```

```
sudo systemctl start hello
```

システムを再起動する reboot コマンドを使用し、システム起動後に sudo systemctl status hello コマンドを入力すると、hello.service が実行中であることが確認できます。

```
sudo systemctl status hello
```

```
cat /tmp/hello.log
```

```
root@lubancat:~# sudo systemctl status hello
● hello.service - hello daemon
   Loaded: loaded (/etc/systemd/system/hello.service; enabled; vendor preset: en
   Active: active (running) since Tue 2022-10-11 13:59:32 CST; 58s ago
   Main PID: 344 (hello.sh)
   Memory: 1.3M
   CGroup: /system.slice/hello.service
           └─ 344 /bin/bash /opt/hello.sh
              1046 sleep 3
root@lubancat:~# cat /tmp/hello.log
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
```

systemd に関する知識点はまだまだ多くありますが、ここでは基本的な使用方法とサービスの起動プロセスを簡単に紹介しました。興味のある方は、オンラインで関連文書を参照してください。

デスクトップ環境を使用しているユーザーは、デスクトップシステムに組み込まれた自動起動機能を利用することをお勧め

## 14.5 Systemd の基本ツール

### 14.5.1 systemctl

systemd を監視し制御する主要なコマンドは systemctl です。このコマンドは、システムの状態を確認し、システムやサービスを管理するために使用されます。

```
# アクティブなユニットを出力

systemctl

# プリントメッセージは以下のよう

UNIT                                LOAD  ACTIVE SUB    JOB           DESCRIPTION
sys-devices-platform-3c080000.pcie-pci0002:20-0002:20:00.0-0002:21:00.0-nvme-nv
sys-devices-platform-backlight-backlight-backlight.device loaded active  plug
sys-devices-platform-fe010000.ethernet-net-eth1.device loaded active  plugged
sys-devices-platform-fe2a0000.ethernet-net-eth0.device loaded active  plugged
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcbk0-mmcbk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcbk0-mmcbk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcbk0-mmcbk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcbk0-mmcbk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcbk0-mmcbk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcbk0-mmcbk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcbk0-mmcbk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcbk0-mmcbk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcbk0-mmcbk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcbk0-mmcbk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcbk0-mmcbk
```

```
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcbk0-mmcbk0
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcbk0.device
sys-devices-platform-fiq_debugger.0-tty-ttyFIQ0.device loaded active plugged
sys-devices-platform-rk809x2dsound-sound-card0.device loaded active plugged
sys-devices-virtual-block-ram0.device loaded active plugged /sys/dev
sys-devices-virtual-block-zram0.device loaded active plugged /sys/de
sys-devices-virtual-misc-rfkill.device loaded active plugged /sys/de
sys-devices-virtual-tty-ttyGS0.device loaded active plugged /sys/dev
sys-module-configfs.device loaded active plugged /sys/module/configf
```

# システム状態表示

```
systemctl status -l
```

# プリントメッセージは以下のように

● lubancat

State: starting

Jobs: 11 queued

Failed: 0 units

Since: Tue 2022-10-11 13:59:30 CST; 5min ago

CGroup: /

└─user.slice

| └─user-0.slice

```
| | |—session-c2.scope
| | | |— 557 /bin/login -p --
| | | |—1021 -bash
| | | |—1160 systemctl status -l
| | | |—1161 pager
| | |—user@0.service
| | |—init.scope
| | |—1006 /lib/systemd/systemd --user
| | |—1009 (sd-pam)
| |—user-1000.slice
| |—user@1000.service
| | |—pulseaudio.service
| | | |—686 /usr/bin/pulseaudio --daemonize=no --log-target=journal
| | |—init.scope
| | |—674 /lib/systemd/systemd --user
```

lines 1-23

```
# システムを再起動
sudo systemctl reboot

# システムをサスペンド
sudo systemctl suspend
```

```
# システムをシャットダウンし、電源を切る

sudo systemctl poweroff

# CPU を停止させる

sudo systemctl halt

# システムを休止状態にする

sudo systemctl hibernate

# システムをハイブリッドスリープ状態にする

sudo systemctl hybrid-sleep

# システムをレスキューモード（シングルユーザーモード）で起動

sudo systemctl rescue

# 起動に失敗したユニットを出力

systemctl --failed
```

## 14.5.2 systemd-analyze

systemd-analyze コマンドは、起動にかかった時間を確認するために使用されます。

```
# 起動時間を確認

systemd-analyze
```

```
# プリントメッセージは以下のように
```

```
Startup finished in 1.809s (kernel) + 49.887s (userspace) = 51.696s
```

```
graphical.target reached after 49.834s in userspace
```

```
# 各サービスの起動にかかった時間を表示
```

```
systemd-analyze blame
```

```
# プリントメッセージは以下のように
```

```
44.661s snapd.seeded.service
```

```
3.820s man-db.service
```

```
3.586s blueman-mechanism.service
```

```
3.500s fstrim.service
```

```
2.270s networkd-dispatcher.service
```

```
2.214s dev-mmcbk0p3.device
```

```
2.051s adbd.service
```

```
1.884s snapd.service
```

```
1.504s async.service
```

```
1.355s logrotate.service
```

```
1.025s accounts-daemon.service
```

```
960ms NetworkManager.service
```

```
823ms polkit.service
```

```
687ms systemd-resolved.service
```

627ms avahi-daemon.service  
613ms ModemManager.service  
550ms systemd-journald.service  
530ms systemd-logind.service  
505ms switcheroo-control.service  
481ms apport.service  
451ms wpa\_supplicant.service  
441ms e2scrub\_reap.service  
427ms colord.service  
425ms systemd-udev-trigger.service  
408ms networking.service  
391ms kerneloops.service  
355ms fwupd-refresh.service  
345ms ssh.service  
303ms packagekit.service  
284ms alsa-restore.service  
265ms user@0.service  
249ms user@112.service  
225ms systemd-udevd.service  
205ms ntp.service  
184ms rsyslog.service  
168ms lightdm.service



156ms pppd-dns.service  
148ms rkwifi.service  
136ms systemd-user-sessions.service  
127ms e2scrub\_all.service  
124ms systemd-fsck@dev-disk-by $\Upsilon$ x2dpartlabel-boot.service  
106ms systemd-journal-flush.service  
102ms dev-mqueue.mount  
101ms bluetooth.service  
97ms sys-kernel-debug.mount  
93ms vsftpd.service  
90ms motd-news.service  
90ms sys-kernel-tracing.mount  
89ms systemd-sysctl.service  
88ms systemd-sysusers.service  
86ms systemd-backlight@backlight:backlight.service  
82ms systemd-random-seed.service  
73ms modprobe@pstore\_blk.service  
72ms modprobe@pstore\_zone.service  
71ms modprobe@efi\_pstore.service  
67ms systemd-pstore.service  
62ms modprobe@chromeos\_pstore.service  
60ms systemd-modules-load.service

```
58ms systemd-remount-fs.service
49ms systemd-tmpfiles-setup.service
47ms plymouth-quit-wait.service
44ms systemd-update-utmp-runlevel.service
41ms user-runtime-dir@112.service
38ms systemd-tmpfiles-setup-dev.service
35ms user-runtime-dir@0.service
35ms systemd-update-utmp.service
27ms sys-fs-fuse-connections.mount
24ms sys-kernel-config.mount
21ms boot.mount
21ms plymouth-read-write.service
18ms ifupdown-pre.service
18ms rtkit-daemon.service
13ms tmp.mount
3ms snapd.socket
```

```
# スタートアッププロセスの流れを表示
```

```
systemd-analyze critical-chain
```

```
# プリントメッセージは以下のように
```

```
The time when unit became active or started is printed after the "@" character.
```

The time the unit took to start is printed after the "+" character.

```
graphical.target @49.834s
└─multi-user.target @49.834s
└─snapd.seeded.service @5.169s +44.661s
└─snapd.service @3.271s +1.884s
└─basic.target @3.094s
└─sockets.target @3.094s
└─snapd.socket @3.089s +3ms
└─sysinit.target @3.069s
└─systemd-update-utmp.service @3.033s +35ms
└─systemd-tmpfiles-setup.service @2.973s +49ms
└─local-fs.target @2.960s
└─boot.mount @2.938s +21ms
└─systemd-fsck@dev-disk-by $\%x2$ dpartlabel-boot.service @2.809s +124ms
└─dev-disk-by $\%x2$ dpartlabel-boot.device @2.778s

# 指定されたサービスの起動フローを表示
$ systemd-analyze critical-chain sshd.service
```

### 14.5.3 hostnamectl

hostnamectl コマンドは、現在のホスト情報を確認するために使用できます。また、hostname コマンドを直接入力して確認することもできますが、ホスト名を変更する際には hostnamectl の方が便利です。

```
# 現在のホスト情報を表示
```

```
hostnamectl
```

```
# プリントメッセージは以下のように
```

```
Static hostname: lubancat
```

```
Icon name: computer
```

```
Machine ID: 994b8b2798844992a0691925b398fab5
```

```
Boot ID: 21de30a7c6ee4a75b619fe31bd53a415
```

```
Operating System: Debian GNU/Linux 10 (buster)
```

```
Kernel: Linux 4.19.232
```

```
Architecture: arm64
```

```
# ホスト名を変更
```

```
root@npi:~# sudo hostnamectl set-hostname cat
```

```
# もう一度ホスト情報を確認する
```

```
root@npi:~# hostnamectl
```

```
Static hostname: cat
```

```
Icon name: computer
```

```
Machine ID: 994b8b2798844992a0691925b398fab5
```

```
Boot ID: 21de30a7c6ee4a75b619fe31bd53a415
```

```
Operating System: Debian GNU/Linux 10 (buster)
```

```
Kernel: Linux 4.19.232
```

```
Architecture: arm64
```

Static hostname が変更されたことが確認できます。

## 14.5.4 localectl

localectl コマンドは、システムのローカライゼーション設定とキーボードレイアウト設定を確認および変更するために使用されます。これらの設定は、ユーザーインターフェイスの言語、文字タイプとエンコーディング、日付と時間の表示形式、通貨記号など多くの詳細を制御します。

```
# ローカライゼーション設定を表示

localectl

# プリントメッセージは以下のように

System Locale: LANG=C.UTF-8

    VC Keymap: n/a

    X11 Layout: us

    X11 Model: pc105

# システムの言語とキーボード設定を設定

root@zhan:~# sudo localectl set-locale LANG=en_GB.utf8

root@zhan:~# sudo localectl set-keymap en_GB

# も一回確認

root@lubancat:~# localectl

System Locale: LANG=en_GB.utf8

    VC Keymap: en_GB

    X11 Layout: us

    X11 Model: pc105
```

## 14.5.5 timedatectl

timedatectl コマンドを使用して、システムクロックと設定を確認および変更できます。このコマンドを使用して、現在の日付、時間、タイムゾーンを設定したり、NTP サーバーとの自動システムクロック同期を有効にしたりできます。

```
# 現在のタイムゾーン設定を表示

timedatectl

# 利用可能なすべてのタイムゾーンを表示

timedatectl list-timezones

# 東京のタイムゾーンを選択

timedatectl set-timezone "Asia/Tokyo"

# ネットワーク時間同期をオフにする

timedatectl set-ntp no

# ネットワーク時間同期をオンにする

timedatectl set-ntp yes

# 時間と日付を設定

timedatectl set-time 9:40:20

timedatectl set-time 2021-4-16
```

時間と日付を設定する際は、時間同期機能をオフにする必要があります。

## 14.5.6 loginctl

loginctl コマンドは、systemd の状態をチェックし制御するために使用できます。ログインしているユーザーセッションの情報を表示することができます。

```
# すべてのセッションと属性を表示

loginctl -a

# セッションの設定情報を表示

loginctl show-session

# 指定ユーザーの情報をリスト表示

loginctl show-user root
```

## 第 15 章 Linux で deb パッケージを作成する方法

プログラムやスクリプトを管理しやすくするために、これらを deb パッケージにまとめることができます。この章では、deb パッケージの作成方法について紹介します。deb パッケージの作成方法にはいくつかありますが、dpkg-deb 方式、checkinstall 方式、dh\_make 方式、既存の deb パッケージを修正する方法などがあります。ここでは、自分の deb パッケージをゼロから作成する方法と、既存の deb パッケージを修正する方法について説明します。

### 15.1 deb パッケージとは？

deb パッケージは Linux システム下で使用されるインストールパッケージの一種で、Web からダウンロードした Linux ソフトウェアのインストールパッケージも deb 形式で提供されることがあります。tar パッケージに基づいているため、ファイルの権限情報（読み取り、書き込み、実行）、所有者、ユーザーグループなどを記録しています。

システムにどの deb パッケージがインストールされているかを確認するには、`dpkg -l` コマンドを使用し

ます。

```
1 cat@lubancat:~$ dpkg -l
2 Desired=Unknown/Install/Remove/Purge/Hold
3 | Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
4 |/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
5 ||/ Name Version Architecture Description
6
7 ii acpi-support-base 0.142-8 all scripts for handling base ACPI events such as the power button
8 ii acpid 1:2.0.31-1 arm64 Advanced Configuration and Power Interface event daemon
9 ii adduser 3.118 all add and remove users and groups
10 ii adwaita-icon-theme 3.30.1-1 all default icon theme of GNOME
11 ii alsa-utils 1.1.8-2 arm64 Utilities for configuring and using ALSA
12 ii anacron 2.3-28 arm64 cron-like program that doesn't go by time
13 ii apt 1.8.2.3 arm64 commandline package manager
14 ii apt-transport-https 1.8.2.3 all transitional package for https support
15 ii apt-utils 1.8.2.3 arm64 package management related utility programs
16 ii aspell 0.60.7~20110707-6+deb10u1 arm64 GNU Aspell spell-checker
```



```
17 ii aspell-en 2018.04.16-0-1 all English dictionary for GNU Aspell
18 ii base-files 10.3+deb10u13 arm64 Debian base system miscellaneous files
19 ii base-passwd 3.5.46 arm64 Debian base system master password and group files
20 ii bash 5.0-4 arm64 GNU Bourne Again SHell
21 ii binfmt-support 2.2.0-2 arm64 Support for extra binary formats
22 ii blueman 2.0.8-1+deb10u1 arm64 Graphical bluetooth manager
23 ii bluez 5.50-1.2~deb10u2 arm64 Bluetooth tools and daemons
24 ii bluez-obexd 5.50-1.2~deb10u2 arm64 bluez obex daemon
25 ii bsdmainutils 11.1.2+b1 arm64 collection of more utilities from FreeBSD
26 ii bsduutils 1:2.33.1-0.1 arm64 basic utilities from 4.4BSD-Lite
27 ii bubblewrap 0.3.1-4 arm64 setuid wrapper for unprivileged chroot and namespace manipulation
28 ii busybox 1:1.30.1-4 arm64 Tiny utilities for small and embedded systems
29 ii bzip2 1.0.6-9.2~deb10u2 arm64 high-quality block-sorting file compressor - utilities
30 ii ca-certificates 20200601~deb10u2 all Common CA certificates
31 ii camera-engine-rkaiq 2.0x60.1 arm64 3A libraries match Rockchip rkisp v21(rk358x).
32 ii can-utils 2018.02.0-1 arm64 SocketCAN userspace utilities and tools
33 ii cheese 3.31.90-1 arm64 tool to take pictures and videos from your webcam
34 ii cheese-common 3.31.90-1 all Common files for the Cheese tool to take pictures and videos
35 lines 1-33
```

これは LubanCat ボード Debian イメージにインストールされた deb パッケージのリストです。Name 列はインストールされた deb パッケージの名前、Version 列はパッケージのバージョン、Architecture 列はパッケージがサポートするプロセッサアーキテクチャを示します。

## 15.2 deb パッケージの構造

deb パッケージは一般に以下の 2 つの部分から構成されます：

- インストールされる内容。これは Linux のルートディレクトリに似ており、Linux システムにソフトウェアをインストールするためのファイルディレクトリを示します。
- 制御情報 (DEBIAN ディレクトリ内)。通常、DEBIAN ディレクトリには以下のファイルが含まれます。
  - changelog: deb パッケージの作者、バージョン、最終更新日などの情報を記録したファイル；
  - control: パッケージ名、バージョン番号、アーキテクチャ、メンテナー、説明などの情報を記録したファイル；
  - copyright: いくつかの著作権情報を記録したファイル；
  - postinst: ソフトウェアが正規のディレクトリにコピーされた後に実行する必要があるスクリプト。
  - postrm: ソフトウェアがアンインストールされた後に実行するスクリプト。

control、postinst、postrm は必須のファイルです。

## 15.3 自分の deb パッケージをゼロから作成

ツールと依存関係のインストール：

```
sudo apt-get install build-essential debhelper make autoconf automake dpkg-dev fakeroot pbuilder gnupg
```

最初に、以下のディレクトリとファイルを作成します。

```
1 hello_deb/  
2 |—— DEBIAN  
3 | |—— control  
4 | |—— postinst  
5 | |—— postrm
```

```
6 └── opt
7 └── hello_deb
8 └── hello_deb.sh
```

hello\_deb ディレクトリ下に DEBIAN ディレクトリと opt/hello\_deb ディレクトリを作成します。DEBIAN ディレクトリには制御情報ファイルが含まれ、opt/hello\_deb ディレクトリに hello\_deb.sh ファイルを作成することで、hello\_deb.sh ファイルを Linux システムの opt/hello\_deb ディレクトリにインストールする必要があることを示します。

次に、postinst、postrm、hello\_deb.sh ファイルに実行権限を与えます。postinst と postrm の権限は 0555 以上 0775 以下でなければなりません。

control ファイルに含まれる情報は以下の通りです：

リスト 1: hello\_deb/DEBIAN/control

```
1 Package: hello-deb
2 Version: 1.0.0
3 Section: free
4 Priority: optional
5 Essential: no
6 Architecture: arm64
7 Maintainer: embedfire <embedfire@embedfire.com>
8 Provides: hell_deb
9 Description: deb test
```

注：control ファイルの最後には空行を追加する必要があります。そうしないと「終端の改行が欠けている」というエラーが発生します。

deb パッケージを将来的にアップグレードしたい場合は、パッケージのバージョン番号 Version を変更でき

ます。Architecture については、この deb パッケージがサポートするプロセッサアーキテクチャを意味します。最終的にこの deb パッケージを arm64 プロセッサのボードにインストールするため、Architecture には arm64 を記載します。必要に応じて適宜変更してください。プロセッサアーキテクチャを知らない場合は、dpkg -l コマンドでインストールされている deb パッケージがサポートするアーキテクチャを確認するか、lscpu コマンドでプロセッサ情報を確認できます。aarch64 は arm64 アーキテクチャを意味します。すべてのアーキテクチャをサポートする場合は all を記載できます。Architecture が現在のプロセッサアーキテクチャと一致しない場合、deb パッケージは正常にインストールできません。また、control の属性情報は英数字で始まる必要があります。そうでないとパッケージ作成時にエラーが発生する可能性があります。

postinst ファイルに含まれる情報は以下の通りです：

リスト 2: hello\_deb/DEBIAN/postinst

```
1 #!/bin/bash
2
3 if [ "$1" = "upgrade" ] || [ "$1" = "install" ];then
4 echo "hello_deb installing"
5 fi
```

この deb パッケージがインストールされた後、システムはデフォルトで postinst スクリプトを実行します。通常、このスクリプトを使用してソフトウェアの実行環境を構築します（例：ディレクトリの作成、権限の変更など）。このファイルには実行権限が必要です。ここでは非常にシンプルに記述しており、第1引数を判断する例を示しています。

postrm ファイルに含まれる情報は以下の通りです：

リスト 3: hello\_deb/DEBIAN/postrm

```
1 #!/bin/bash
2
3 if [ "$1" = "upgrade" ]; then
4 echo "upgrade"
5 elif [ "$1" = "remove" ] || [ "$1" = "purge" ]; then
6 echo "remove"
7 fi
```

この deb パッケージがアンインストールされた後、システムはデフォルトで postrm スクリプトを実行します。通常、このスクリプトを使用して環境をクリーンアップします。このファイルには実行権限が必要です。ここでは非常にシンプルに記述しており、第 1 引数を判断する例を示しています。

最後に、実際のプログラムの本体について見てみましょう。

リスト 4: hello\_deb/opt/hello\_deb/hello\_deb.sh

```
1 #! /bin/bash
2
3 echo Hello deb!
4 echo This is a test script!!
```

このスクリプトは単純に 2 つのメッセージを出力するだけです。ユーザーは実行したいプログラムを自由に設定できます。

全て準備が整ったら、自分だけの deb パッケージの作成を開始できます。hello\_deb ディレクトリ

(DEBIAN および home フォルダがあるディレクトリ) に移動し、以下のコマンドを入力してパッケージを構築します。

```
sudo dpkg-deb -b ../hello_deb ../hello_deb_1.0.0_arm64.deb
```

ここで、dpkg-deb は deb パッケージを構築するコマンドで、-b オプションは deb パッケージを構築するこ

とを意味し、../hello\_deb は deb パッケージの原材料があるパス、../hello\_deb\_1.0.0\_arm64.deb は現在のディレクトリの親ディレクトリに deb パッケージを構築することを指します。通常、deb パッケージの名前は「ソフトウェア名+ソフトウェアのバージョン番号+そのソフトウェアがサポートするプロセッサアーキテクチャ」という原則に従って命名されます。例えば、ソフトウェア名が hello\_deb で、バージョン番号が 1.0.0、サポートするプロセッサアーキテクチャが arm64 です。

パッケージが成功裏に構築されると、以下のような情報が出力され、親ディレクトリで deb インストールパッケージを確認できます：

```
1 jiawen@dev120:~/deb/hello_deb$ sudo dpkg-deb -b ../hello_deb ../hello_deb_1.0.0_arm64.deb
2 dpkg-deb: ../hello_deb_1.0.0_arm64.deb で hello-deb パッケージを構築中です。」。
```

自分の deb パッケージを作成した後、以下のコマンドで作成した deb パッケージファイルの内容を確認して、正常に作成されたかを検証できます：

```
1 # コマンド
2 dpkg -c hello_deb_1.0.0_arm64.deb
3
4 # プリントメッセージ
5 drwxrwxr-x jiawen/jiawen 0 2022-10-12 09:27 ./
6 drwxrwxr-x jiawen/jiawen 0 2022-10-12 09:28 ./opt/
7 drwxrwxr-x jiawen/jiawen 0 2022-10-12 09:28 ./opt/hello_deb/
8 -rwxrwxrwx jiawen/jiawen 59 2022-10-12 09:41 ./opt/hello_deb/hello_deb.sh
```

または、以下のコマンドで deb パッケージの情報を確認できます：

```
1 # コマンド
2 dpkg --info hello_deb_1.0.0_arm64.deb
3
4 # プリントメッセージ
5 new Debian package, version 2.0.
6 size 976 bytes: control archive=496 bytes.
7 190 バイト,  9 行 control
8 100 バイト,  4 行* postinst #!/bin/bash
9 138 バイト,  7 行* postrm #!/bin/bash
10 Package: hello-deb
11 Version: 1.0.0
12 Section: free
13 Priority: optional
14 Essential: no
15 Architecture: arm64
16 Maintainer: embedfire <embedfire@embedfire.com>
17 Provides: hell_deb
18 Description: deb test
```

この deb パッケージを LubanCat ボードのファイルシステムにコピーし、「sudo dpkg -i hello\_deb\_1.0.0\_arm64.deb」コマンドを入力してインストールします。ここで、-i オプションはソフトウェアのインストールを意味します。ソフトウェアのインストールが完了した後、「dpkg -s hello-dev」コマンドでソフトウェアがインストールされているかを確認できます。

```
1 cat@lubancat:~$ sudo dpkg -i hello_deb_1.0.0_arm64.deb
2 Selecting previously unselected package hello-deb.
3 (Reading database ... 74783 files and directories currently installed.)
4 Preparing to unpack hello_deb_1.0.0_arm64.deb ...
5 Unpacking hello-deb (1.0.0) ...
6 Setting up hello-deb (1.0.0) ...
7 cat@lubancat:~$ dpkg -s hello-deb
8 Package: hello-deb
9 Status: install ok installed
10 Priority: optional
11 Section: free
12 Maintainer: embedfire <embedfire@embedfire.com>
13 Architecture: arm64
14 Version: 1.0.0
15 Provides: hell_deb
16 Description: deb test
```

また、「dpkg -l | grep hello-deb」コマンドでインストール済みソフトウェアリストにソフトウェアが存在するかを確認できます。

```
1 cat@lubancat:~$ dpkg -l | grep hello-deb
2 ii hello-deb 1.0.0 arm64 deb test
```

インストールが完了した後、開発ボードの/opt/hello\_deb ディレクトリに hello\_deb.sh ファイルが存在するかを確認してください。



```
1 cat@lubancat:~$ ls /opt/  
2 hello_deb  
3 cat@lubancat:~$ ls /opt/hello_deb/  
4 hello_deb.sh
```

実行して効果を確認してください。

```
1 cat@lubancat:~$ /opt/hello_deb/hello_deb.sh  
2 Hello deb!  
3 This is a test script!!!
```

これで、deb パッケージの作成の基本的な流れは説明完了です

## 15.4 既存の deb パッケージの内容を変更する方法

ここでは、既存の deb パッケージを変更する方法について説明します。例として、先ほど作成した deb パッケージを使用し、LubanCat ボード上で直接変更します。

LubanCat ボード上でツールと依存関係をインストールします：

```
sudo apt-get install build-essential debhelper make autoconf automake dpkg-dev fakeroot pbuilder gnupg
```

新しい update\_deb ディレクトリを作成し、dpkg -X コマンドを使用して deb パッケージを update\_deb ディレクトリに解凍します。

```
1 cat@lubancat:~$ ls  
2 Desktop Downloads Pictures Templates hello_deb_1.0.0_arm64.deb  
3 Documents Music Public Videos  
4 cat@lubancat:~$ mkdir update_deb  
5 cat@lubancat:~$ sudo dpkg -X hello_deb_1.0.0_arm64.deb update_deb/  
6 ./
```

```
7 ./opt/  
8 ./opt/hello_deb/  
9 ./opt/hello_deb/hello_deb.sh  
10 cat@lubancat:~$
```

update\_deb ディレクトリに移動すると、DEBIAN 関連のディレクトリは見当たりません。update\_deb ディレクトリで dpkg -e を使用して、制御ファイル関連の情報を解凍します。

```
1 cat@lubancat:~/update_deb$ ls  
2 opt  
3 cat@lubancat:~/update_deb$ sudo dpkg -e ../hello_deb_1.0.0_arm64.deb  
4 cat@lubancat:~/update_deb$ ls -al  
5 total 16  
6 drwxrwxr-x 4 1001 1001 4096 Oct 12 13:37 .  
7 drwxr-xr-x 15 cat cat 4096 Oct 12 13:35 ..  
8 drwxr-xr-x 2 root root 4096 Oct 12 09:28 DEBIAN  
9 drwxrwxr-x 3 1001 1001 4096 Oct 12 09:28 opt  
10  
11 cat@lubancat:~/update_deb$ tree  
12 .  
13 |-- DEBIAN  
14 | |-- control  
15 | |-- postinst  
16 | `-- postrm
```

```
17 `-- opt
18 `-- hello_deb
19 `-- hello_deb.sh
20
21 3 directories, 4 files
```

この時点で、プログラム本体を変更できます。opt/hello\_deb/hello\_deb.sh にメッセージを追加します、例えば：

リスト 5: update\_deb/opt/hello\_deb/hello\_deb.sh

```
1 #! /bin/bash
2
3 echo Hello deb!
4 echo This is a test script!!!
5 echo update deb!
```

DEBIAN/control のバージョン情報を変更し、バージョンを 1.0.1 にします。

リスト 6: update\_deb/DEBIAN/control

```
1 Package: hello-deb
2 Version: 1.0.1
3 Section: free
4 Priority: optional
5 Essential: no
6 Architecture: arm64
7 Maintainer: embedfire embedfire@embedfire.com
```

```
8 Provides: hell_deb
9 Description: deb test
```

これで、アプリケーションを再パッケージしてインストールする準備が整いました。

```
1 cat@lubancat:~/update_deb$ sudo dpkg-deb -b ../update_deb ../hello_deb_1.0.1_arm64.deb
2 dpkg-deb: building package 'hello-deb' in '../hello_deb_1.0.1_arm64.deb'.
3
4 cat@lubancat:~$ sudo dpkg -i hello_deb_1.0.1_arm64.deb
5 (Reading database ... 103608 files and directories currently installed.)
6 Preparing to unpack hello_deb_1.0.1_arm64.deb ...
7 Unpacking hello-deb (1.0.1) over (1.0.0) ...
8 upgrade
9 Setting up hello-deb (1.0.1) ...
10
11 cat@lubancat:~$ /opt/hello_deb/hello_deb.sh
12 Hello deb!
13 This is a test script!!!
14 update deb!
```

## 15.5 deb パッケージをインストールして自動起動サービスを作成する

deb パッケージをインストールしてLubanCat ボード上でプログラムを自動起動させることは、実際のプロジェクトで非常に一般的です。ここでは、deb プログラムを使用して自動起動する簡単な例を紹介します。

新しい deb パッケージディレクトリを作成し、ファイル構造は以下の通りです：

```
1 hello_deb
2 |—— DEBIAN
3 | |—— control
4 | |—— postinst
5 | |—— postrm
6 |—— etc
7 | |—— systemd
8 | |—— system
9 | |—— hello.service
10 |—— opt
11 |—— hello_deb
12 |—— hello_deb.sh
```

/etc/systemd/system/hello.service は自動起動サービスで、/opt/hello\_deb/hello\_deb.sh はプログラム本体です。DEBIAN ディレクトリには deb パッケージの説明情報やインストール・アンインストール時に実行するスクリプトなどが含まれます。基本的な概念についてはここでは詳しく説明しませんが、各ファイルの内容を直接確認します。

DEBIAN/control の内容は以下の通りです：

リスト 7: DEBIAN/control

```
1 Package: hello-deb
2 Version: 1.0.2
3 Section: free
4 Priority: optional
```

```
5 Essential: no
6 Architecture: arm64
7 Maintainer: embedfire <embedfire@embedfire.com>
8 Provides: hell_deb
9 Description: deb test
```

DEBIAN/postinst の内容は以下の通りです。スクリプトはシンプルで、hello 関連のサービスを有効にして起動するだけです。

リスト 8: DEBIAN/postinst

```
1 #!/bin/bash
2
3 if [ "$1" = "upgrade" ] || [ "$1" = "install" ];then
4 echo "hello_deb installing"
5 systemctl enable hello
6 systemctl start hello
7 fi
```

DEBIAN/postrm の内容は以下の通りです。プログラムをアンインストールするときに、hello 関連のサービスを停止してください。

リスト 9: DEBIAN/postrm

```
1 if [ "$1" = "upgrade" ]; then
2 echo "upgrade"
3 elif [ "$1" = "remove" ] || [ "$1" = "purge" ]; then
4 echo "remove"
```

```
5 systemctl disable hello
6 fi
```

/etc/systemd/system/hello.service のサービス内容は以下の通りで、第 6 行で実行するプログラムが指定されています。

リスト 10: /etc/systemd/system/hello.service

```
1 [Unit]
2 Description = hello daemon
3
4 [Service]
5 ExecStart = /opt/hello_deb/hello_deb.sh
6 Restart = always
7 Type = simple
8
9 [Install]
10 WantedBy = multi-user.target
```

/opt/hello\_deb/hello\_deb.sh はプログラム本体で、3 秒ごとに /tmp/hello.log に "Hello Embedfire" と書き込みます。

リスト 11: /opt/hello\_deb/hello\_deb.sh

```
1 #!/bin/bash
2
3 while true
4 do
```

```
5 echo Hello Embedfire >> /tmp/hello.log  
6 sleep 3  
7 done
```

これらのディレクトリをパッケージ化して deb ファイルを作成し、ボード上で dpkg コマンドを使用してインストールします。再起動後、「systemctl status hello」でサービスの状態を確認できます。

```
1 root@lubancat:~# systemctl status hello  
2 ● hello.service - hello daemon  
3 Loaded: loaded (/etc/systemd/system/hello.service; enabled; vendor preset: >  
4 Active: active (running) since Wed 2022-10-12 13:54:55 CST; 12s ago  
5 Main PID: 372 (hello_deb.sh)  
6 Memory: 2.5M  
7 CGroup: /system.slice/hello.service  
8 └─372 /bin/bash /opt/hello_deb/hello_deb.sh  
9 └─817 sleep 3  
10 lines 1-8/8 (END)  
11  
12 root@lubancat:~# cat /tmp/hello.log  
13 Hello Embedfire  
14 Hello Embedfire  
15 Hello Embedfire  
16 Hello Embedfire  
17 Hello Embedfire
```

また、/tmp/hello.log ファイルを確認してプログラムが正しく実行してるかどうかを確認できます。



## 第 16 章 自動起動サービスを追加する

この章では、Systemd サービスおよびデスクトップシステムが持つ自動起動の方法を用いて、プログラムの自動起動を実現する二つの方法をまとめます。

この章は、イメージをボードに書き込んで正常にシステムが起動している状況に適用されます。

### 16.1 Systemd 方式

hello.service サービスの作成プロセスは既に Systemd の探索で説明されていますので、ここでは自動起動を実現する方法のみを説明します。Systemd に関連する内容の詳細は、Systemd の探索の章をご覧ください。

#### 16.1.1 スクリプトを書く

/opt ディレクトリに cd で入り、vim を使用して hello.sh スクリプトを作成します。

```
1 #!/bin/bash
2
3 while true
4 do
5 echo Hello Lubancat >> /tmp/hello.log
6 sleep 3
7 done
```

このスクリプトは、3 秒ごとに「Hello Lubancat」の文字列を /tmp/hello.log ファイルに出力する機能を実現します。作成後は、hello.sh に実行権限を与えることを忘れないでください。

```
sudo chmod 0755 hello.sh
```

## 16.1.2 設定ファイルを作成する

/etc/systemd/system/ ディレクトリに hello.service の設定ファイルを作成します。内容は以下の通りです。

```
1 [Unit]
2 Description = hello daemon
3
4 [Service]
5 ExecStart = /opt/hello.sh
6 Restart = always
7 Type = simple
8
9 [Install]
10 WantedBy = multi-user.target
```

ExecStart フィールドでは、hello.service サービスの自動起動スクリプトが /opt/hello.sh であることを定義しています。hello.service の自動起動機能を有効にした後、システムを起動すると /opt/hello.sh が実行されます。Restart = always は、プロセスまたはサービスが予期せずに失敗した場合に自動的に再起動するモードを表します。Type = simple はデフォルトのもので、省略可能です。WantedBy はサービスの起動元を指定し、WantedBy = multi-user.target はシステムが多ユーザーモードになった時にこのサービスを起動順序に追加することを意味します。グラフィカルユーザーインターフェースの後で起動する必要がある場合は、WantedBy = graphical.target と設定する必要があり、これはシステムがグラフィカルユーザーインターフェースモードに入った時にサービスを起動することを意味します。

つまり、WantedBy パラメーターはサービスがデスクトップ環境を必要とするかどうかにとって非常に重

要であり、サービスがデスクトップ環境を必要とするのに WantedBy = multi-user.target と設定してデスクトップが起動する前に自分の起動スクリプトを呼び出すと、サービスの起動に失敗することになります！

```
1 # デスクトップの前にサービスを起動する
2 WantedBy = multi-user.target
3
4
5 # デスクトップに入った後にサービスを起動する
6 WantedBy = graphical.target
```

### 16.1.3 hello.service の自動起動機能を有効にする

「sudo systemctl list-unit-files -type=service | grep hello」というコマンドを入力して、hello.service がサービスリストに追加されたかどうかを確認します。

```
sudo systemctl list-unit-files --type=service | grep hello
```

```
root@lubancat:/etc/systemd/system# sudo systemctl list-unit-files --type=service | gre
p hello
hello.service                                disabled
root@lubancat:/etc/systemd/system#
```

hello.service が disable 状態であることが確認できます。上記のコマンドを入力した後に何も表示されない場合は、作成したサービスに問題があり、慎重に調査する必要があります。以下のコマンドを入力して、hello.service を起動時に自動的に起動するようにします。

```
sudo systemctl enable hello
```

```
sudo systemctl start hello
```

次に、reboot コマンドを使用してシステムを再起動し、システムが起動した後に「sudo systemctl status hello」コマンドを入力すると、hello.service が実行状態であることが確認できます。

```
sudo systemctl status hello
```

```
cat /tmp/hello.log
```

```
root@lubancat:~# sudo systemctl status hello
● hello.service - hello daemon
   Loaded: loaded (/etc/systemd/system/hello.service; enabled; vendor preset: en
   Active: active (running) since Tue 2022-10-11 13:59:32 CST; 58s ago
 Main PID: 344 (hello.sh)
   Memory: 1.3M
   CGroup: /system.slice/hello.service
           └─ 344 /bin/bash /opt/hello.sh
              1046 sleep 3
root@lubancat:~# cat /tmp/hello.log
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
```

## 16.2 デスクトップシステム方式

例えば xfce、xfce-full、gnome バージョンのイメージなど、デスクトップ環境を備えたイメージを使用する場合は、デスクトップシステムが提供する自動起動サービスを通じて実現できます。デスクトップユーザーにはこの方法を強く推奨します。

### 16.2.1 自動起動設定スクリプトの作成

デスクトップにログインするユーザーのホームディレクトリに起動ファイルを作成します：

```
# cat ユーザーを例に

mkdir /home/cat/.config/autostart

# 設定ファイルを作成

vim /home/cat/.config/autostart/xfce-terminal.desktop

xfce-terminal.desktop ファイルに以下の内容を追加します：

[Desktop Entry]

Type=Application
```

```
Exec=/opt/test.sh

Hidden=false

NoDisplay=false

X-GNOME-Autostart-enabled=true

Name=My App

Comment=Start My App on login
```

この設定ファイルの内容は以下の通りです：

- Type=Application：タイプをアプリケーションとして指定します。
- Exec=/opt/test.sh：実行するコマンドまたはスクリプトのパスを指定します。
- Hidden=false：この自動起動項目を隠さない。
- NoDisplay=false：デスクトップ環境のランチャーにこの自動起動項目を表示します。
- X-GNOME-Autostart-enabled=true：デスクトップ環境での自動起動機能を有効にします。
- Name=My App：自動起動項目に名前を指定します。
- Comment=Start My App on login：ログイン時に My App を起動するという自動起動項目のコメント、その役割を説明します。

この設定ファイルを通じて、ユーザーがデスクトップ環境にログインすると、システムは自動的に /opt/test.sh スクリプトを実行し、対応するアプリケーションを起動します。

## 16.2.2 自動起動スクリプトの作成

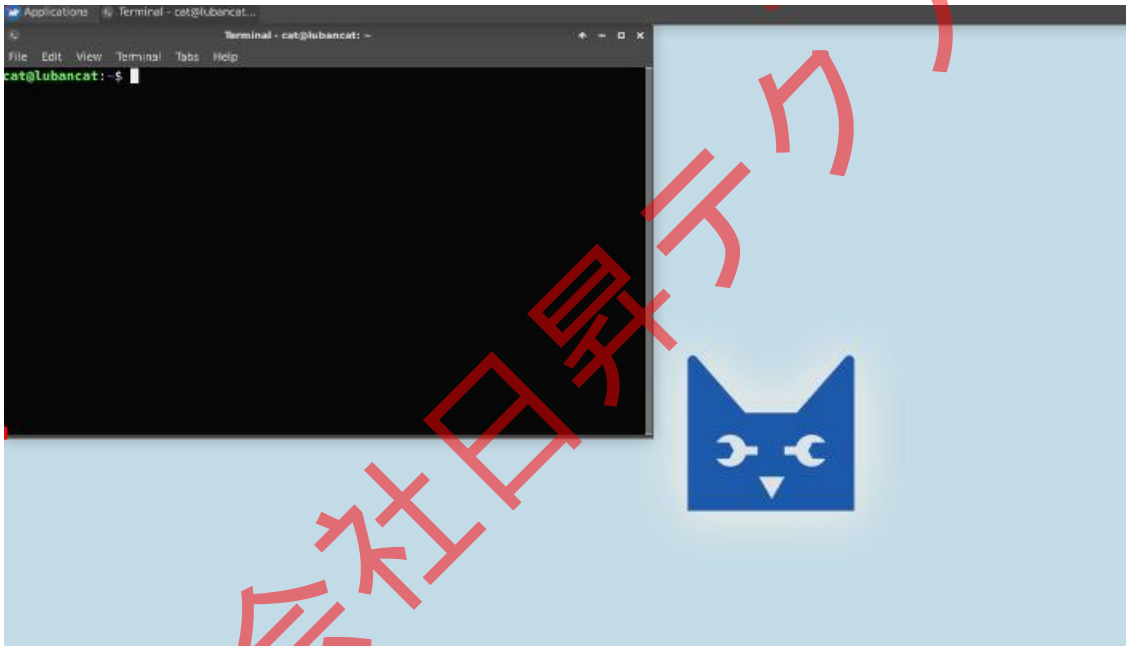
上記の自動起動設定に基づき、自動起動スクリプトを作成します。/opt ディレクトリに cd で入り、test.sh スクリプトを作成します。

```
1 #!/bin/bash
2
3 # デスクトップの端末ウィンドウを開く
4 xfce4-terminal
```

上記のスクリプトは、デスクトップで端末ウィンドウを開く機能を実現します。作成後、適切な権限を付与する必要があります。

```
sudo chmod 777 /opt/test.sh
```

上記の手順を完了した後、開発ボードを再起動すると、デスクトップに入ると自動的に端末が開きます。



## 第17章 システムサービスをルートファイルシステムのビルドに追加する

起動のボード上で直接 Systemd サービスを作成する以外に、ルートファイルシステムのビルドスクリプトにも追加することができます。

ここでは、Debian イメージのビルドスクリプトを例にとります。

## 17.1 スクリプトを書く

LubanCat-SDK を開き、debian/overlay/etc/init.d ディレクトリに hello.sh スクリプトを作成します。

```
1 #!/bin/bash
2
3 while true
4 do
5 echo Hello Lubancat >> /tmp/hello.log
6 sleep 3
7 done
```

このスクリプトは、3 秒ごとに「Hello Lubancat」という文字列を/tmp/hello.log ファイルに出力する機能を実現します。書き終わったら、hello.sh に実行権限を与えることを忘れないでください。

```
sudo chmod 0755 hello.sh
```

```
root@lubancat:/home/cat/LubanCat_SDK/debian/overlay/etc/init.d# sudo chmod 0775 hello.sh
root@lubancat:/home/cat/LubanCat_SDK/debian/overlay/etc/init.d#
```

## 17.2 設定ファイルを作成する

debian/overlay/usr/lib/systemd/system/ディレクトリに hello.service の設定ファイルを作成します。内容は以下の通りです。

```
1 [Unit]
2 Description = hello daemon
3
4 [Service]
5 ExecStart = /etc/init.d/hello.sh
6 Restart = always
```

```
7 Type = simple
8
9 [Install]
10 WantedBy = multi-user.target
```

```
root@lubancat:/home/cat/LubanCat_SDK/debian/overlay/etc/init.d# sudo chmod 0775 hello.sh
root@lubancat:/home/cat/LubanCat_SDK/debian/overlay/etc/init.d# █
```

ExecStart フィールドでは、hello.service サービスの自動起動スクリプトが/etc/init.d/hello.shであることを定義しています。hello.service の自動起動機能を有効にした後、システムが起動すると/etc/init.d/hello.sh が実行されます。Restart = always は、プロセスまたはサービスが予期せずに失敗した場合に自動的に再起動するモードを指します。Type = simple はデフォルトのもので、省略可能です。

### 17.3 hello.service の自動起動機能を有効にする

/etc/systemd/system の multi-user.target.wants ディレクトリに、/usr/lib/systemd/system/hello.service へのシンボリックリンクを作成することで、hello.service を multi-user.target に加えます。このグループ内のすべてのサービスは、システム起動時に自動的に起動します。

LubanCat-SDK の debian/overlay/etc/systemd/system/multi-user.target.wants ディレクトリに移動し、ln コマンドを使用してシンボリックリンクを作成します。

```
1 # multi-user.target.wants ディレクトリに入る
2 cd debian/overlay/etc/systemd/system/multi-user.target.wants
3
4 # シンボリックリンクを作成する
5 ln -s /usr/lib/systemd/system/hello.service hello.service
```

```
root@lubancat:/home/cat/LubanCat_SDK# cd debian/overlay/etc/systemd/system/multi-user.target.wants
root@lubancat:/home/cat/LubanCat_SDK/debian/overlay/etc/systemd/system/multi-user.target.wants# ls
async.service
root@lubancat:/home/cat/LubanCat_SDK/debian/overlay/etc/systemd/system/multi-user.target.wants# ln -s /usr/lib/systemd/system/hello.service hello.service
root@lubancat:/home/cat/LubanCat_SDK/debian/overlay/etc/systemd/system/multi-user.target.wants# ls -l
total 0
lrwxrwxrwx 1 cat cat 33 3月 28 13:45 async.service -> /lib/systemd/system/async.service
lrwxrwxrwx 1 root root 37 3月 28 17:52 hello.service -> /usr/lib/systemd/system/hello.service
root@lubancat:/home/cat/LubanCat_SDK/debian/overlay/etc/systemd/system/multi-user.target.wants# █
```



変更を完了したら、ルートファイルシステムを再ビルドし、待ちます。

システムイメージのビルドが完了したら、ボードを再度書き込み、システムを起動した後に「sudo systemctl status hello」というコマンドを入力すると、hello.service が実行状態であることが確認できます。

```
sudo systemctl status hello
```

```
cat /tmp/hello.log
```

```
root@lubancat:~# sudo systemctl status hello
● hello.service - hello daemon
   Loaded: loaded (/lib/systemd/system/hello.service; enabled; vendor preset:
   enabled)
   Active: active (running) since Wed 2022-08-31 23:27:36 CST; 1 mo
 nths 10 days ago
   Main PID: 375 (hello.sh)
   Memory: 2.5M
   CGroup: /system.slice/hello.service
           └─ 375 /bin/bash /etc/init.d/hello.sh
              └─ 1878 sleep 3
root@lubancat:~# cat /tmp/hello.log
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
```

## 第 18 章 バックアップと量産について

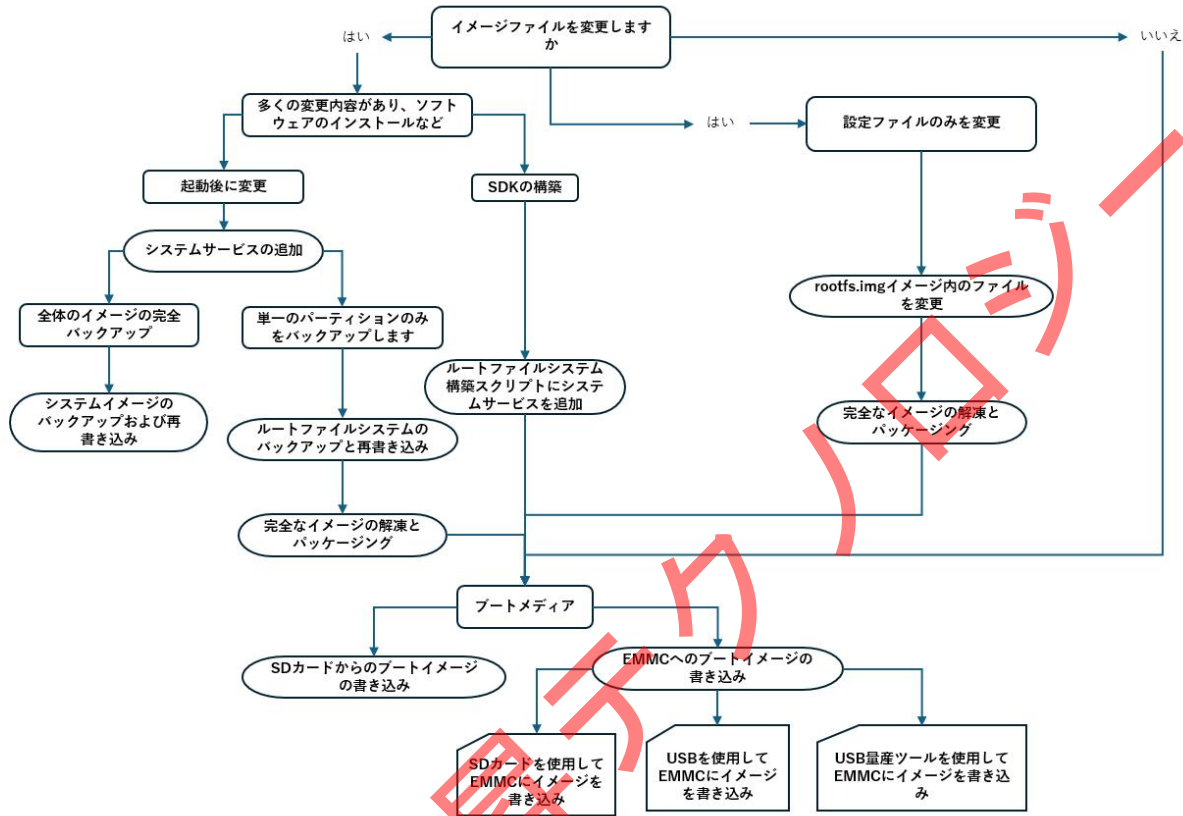
量産とバックアップに関して、この章は非常に重要です。

LubanCat-RK ボードは現在、SD カードと eMMC の 2 種類の起動方式のみをサポートしています。したがって、量産プロセスでは、製品に応じて適切なシステムイメージのダウンロード方法を選択することが主なポイントです。

システムイメージの保存メディアとしては、eMMC の使用を推奨します。eMMC はより高速な読み書き速度とより高い安定性を備えていますが、SD カードは互換性に問題があり、一部のブランドやの SD カードを使用するとシステムが起動しないか、異常に動作することがあります。また、Android システムイメージは SD カード上では動作しません。

イメージのダウンロードについて特に言及する以外にも、一部のユーザーはルートファイルシステムの変更、システムサービスのカスタマイズなどのニーズがあり、この部分の内容が複雑になりがちです。ここ

では、いくつかの一般的な技術ルートを列挙し、ユーザーが自身の状況に応じて適切な方法を見つけることができます。



## 18.1 イメージのダウンロード

LubanCat-RK ボードは現在、SD カードと eMMC の 2 種類の起動方式をサポートしています。以下では、これら 2 種類の起動メディアに対する書き込み方法について説明します。

### 18.1.1 SD カード

提供する LubanCat イメージを SD カードに書き込む方法は現在 1 つだけで、それは Rockchip の公式書き込みツールである SDDiskTool を利用する方法で、Windows 版のみが提供されています。

SD カードへの書き込みはパーティションの書き込みをサポートしておらず、SDK が生成する完全な update.img イメージのみを書き込むことができます。

実際の使用過程で、Win10 システム上で SDDiskTool を使用して SD カードにイメージを繰り返し書き込むと、既にかき込み済みの SD カードに多くのパーティションが存在するため、コンピュータがブルースク

リーンになることがあります。現在解決策はありません。このツールを使用する必要がある場合は、書き込み前に他のソフトウェアの内容を保存しておくべきです。

SD カード書き込みに関する内容は以下の章で確認してください：

SD カード起動イメージの書き込み

## 18.1.2 eMMC

提供する LubanCat イメージを eMMC に書き込む方法は 2 つあります。一つは SD アップグレードカードを使用して eMMC にイメージを書き込む方法、もう一つは USB 書き込みツールを使用してイメージを書き込む方法です。

SD アップグレードカードの作成には、SDDiskTool ツールを使用する必要があり、パーティション書き込みには対応しておらず、SDK が生成した完全な update.img イメージのみを書き込むことができます。その原理は、SD カード内に完全な起動イメージを書き込み、さらに update.img を SD カードに残し、システムを SD カードから起動した後、update.img イメージを eMMC に書き込むことです。

USB 書き込みツールを使用して eMMC にイメージを書き込む方法には 2 種類あります。一つは、Rockchip の開発ツール RKDevTool (Windows 版) または Linux\_Upgrade\_Tool (Linux 版) を使用する方法、もう一つは Rockchip の USB 量産書き込みツール FactoryTool (Windows 版のみ) を使用する方法です。

RKDevTool と Linux\_Upgrade\_Tool の機能は似ており、両方ともパーティション書き込みと完全な update.img イメージの書き込みに対応しています。これらのツールは主に開発過程でのイメージ書き込みに使用され、書き込み機能だけでなく他の高度な機能も提供しますが、一度に一枚のボードにのみ操作を行うことができます。

FactoryTool 量産ツールは主に工場での生産に使用され、update.img イメージのみをサポートし、複数のボードを同時に書き込むことができ、操作が簡単です。

これらの書き込み方法を総合的に分析すると、FactoryTool 量産ツールは大量生産に適しており、

RKDevTool と Linux\_Upgrade\_Tool は開発時や少量生産に適しています。SD カード書き込み方法は推奨されません。

eMMC 書き込みの詳細とソフトウェア使用説明は以下の章でご確認ください：

eMMC 起動イメージの書き込み

## 18.2 完全イメージの分割と統合

上記の書き込みプロセスを理解することで、生産過程では主に完全なイメージ update.img を使用しており、リリースされるのも xxx\_update.img の完全なイメージです。二次開発を行うユーザーにとって、主に変更する内容はルートファイルシステムの部分で、バックアップ時にもルートファイルシステムをバックアップします。

したがって、個々のパーティションイメージをどのようにして完全な update.img イメージに変換するか、これには完全イメージのパッケージングと分解が関わってきます。具体的な内容は以下の章を参照してください：

完全イメージの解凍とパッケージング

## 18.3 バックアップと復元

一部のユーザーは自分の機能を迅速に追加して二次開発を行いたいですが、イメージの構築やコンパイル手順にあまり慣れていない場合があります。そのような時には、LubanCat イメージを基に直接変更を行い、変更後のイメージを再配布することで、ボードイメージのバックアップと再書き込みが行われます。

変更内容や操作の難易度に応じて、以下のいくつかの方法を選択できます：

### 18.3.1 SD カードまたは eMMC の内容を完全バックアップ及び書込

SD カードや eMMC を丸ごとバックアップする方法を選択することができます。この方法でのバックアッププロセスは最も簡単ですが、バックアップで得られた RAW 形式のイメージは Rockchip の量産ツールを使用して書き込むことができません。SD カードを使用して起動する場合や、検証段階や eMMC の少量生産

段階での書き込みにこの方法を選択できます。

システムイメージのバックアップと再書き込み

### 18.3.2 ルートファイルシステムパーティションのバックアップ

変更内容が少ない場合は、ルートファイルシステムパーティションまたは他の個別のパーティションのみをバックアップし、単一のパーティションイメージを取得することもできます。この場合、Rockchip の開発ツールを使用してパーティションイメージを書き込むことも、パーティションイメージを完全なイメージにパッケージ化して、後続の書き込みプロセスをが提供する LubanCat イメージの書き込みプロセスと同じにすることもできます。

しかし、この方法の限界は、複数のパーティションをバックアップする場合、各パーティションを個別のパーティションイメージとしてバックアップする必要があり、イメージの分割と統合を含む全プロセスが非常に煩雑になることです。

ルートファイルシステムのバックアップと再書き込み

### 18.3.3 PC で rootfs.img イメージを修正

ルートファイルシステムの少量の内容のみを変更する場合は、rootfs.img を Linux PC にマウントして直接修正することができます。この方法は、ルートファイルシステムに精通している開発者のみに適していません。不適切な変更を行うと、システムが起動しないか、異常に動作する可能性があります。

開発者が chroot ツールの使用にも精通している場合、さらに、ホストのネットワークを利用してソフトウェアパッケージをインストールしたり、システムサービスを変更したりすることができます。

rootfs.img イメージ内部のファイルを修正

## 18.4 システムサービスの追加

LubanCat ボードを使用する際に、いくつかのシステムサービスを追加または変更する必要がある場合があります。追加のタイミングに応じて、以下の 2 つの方法に分けられます。

## 18.4.1 ボードへのシステムサービスの追加または変更

イメージがボードに書き込まれた後、起動しているボード上でシステムサービスを追加または変更する必要があります。自分のニーズに応じて変更を加えます。

システム自動起動サービスの追加

## 18.4.2 ビルドスクリプトへのシステムサービスの追加または変更

システムイメージがまだ生成されていない場合、ルートファイルシステムを構築する際に、対応するシステムサービスを追加し、適切なパラメータを設定する必要があります。これにより、変更が確定し、再現可能になります。

ルートファイルシステム構築スクリプトへのシステムサービスの追加

# 第 19 章 SD カード起動イメージの書き込み

SD カードの書き込みプロセスでは、Rockchip の SD カード書き込みツール SDDiskTool とボードに対応する RK 形式の完全イメージを使用する必要があります。

Rockchip の SD カード書き込みツール SDDiskTool は現在 Windows 版のみで、したがって SD カードの書き込みプロセスは Windows でのみ行うことができます。この文書では Windows 10 21H1 バージョンを使用していますが、他のバージョンの互換性はテストされていません。

注意: 書き込み後の SD カードには多くのパーティションが存在するため、SDDiskTool を再利用して書き込む際には、コンピュータがブルースクリーンになる可能性があります。書き込み前に未保存のアプリケーションデータを保存してください。

RK 形式の完全イメージとは：

- SDK を直接使用して構築された rockdev/update.img イメージ。
- リリースした Update.img で終わるシステムイメージ。

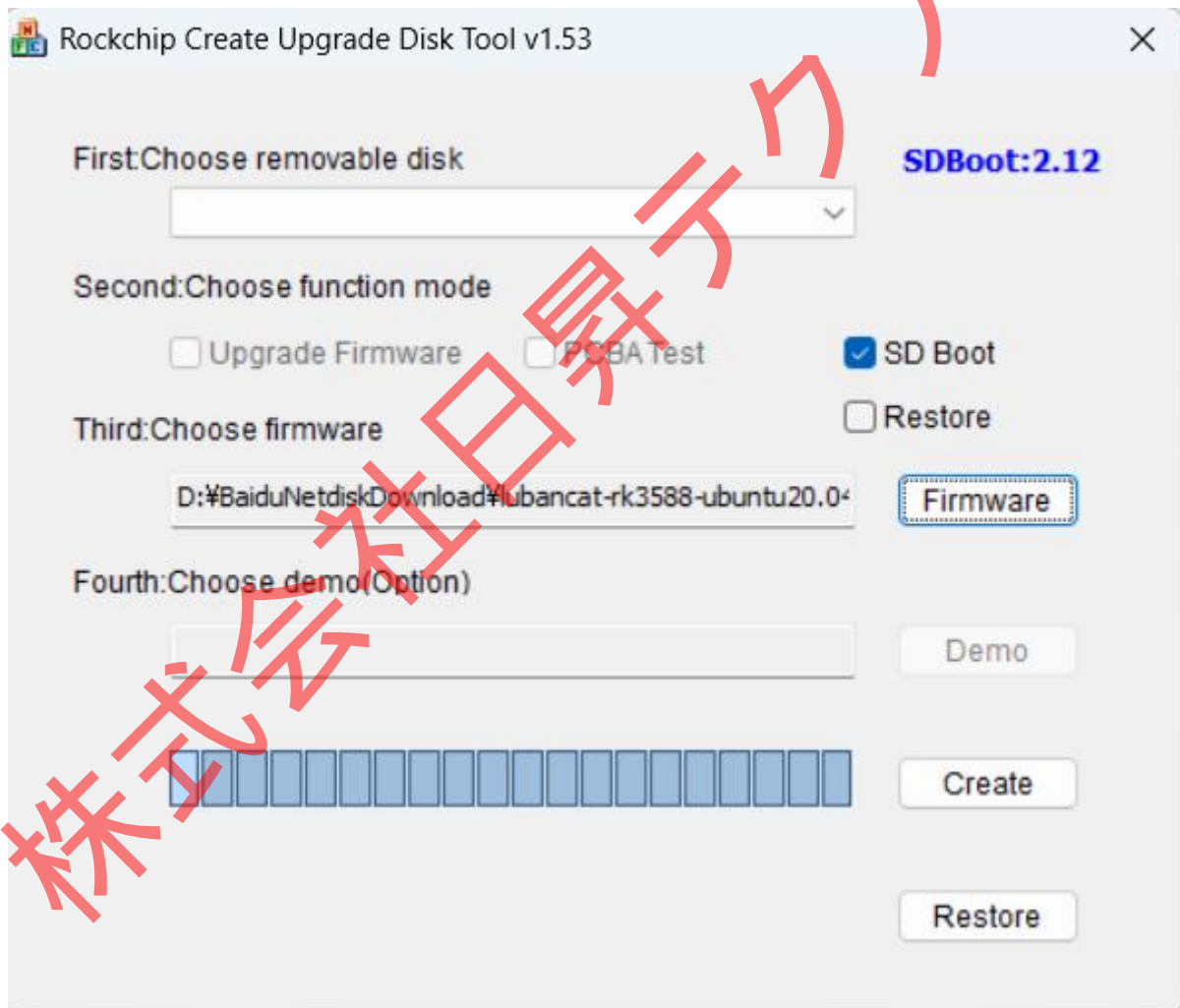
SDDiskTool 書き込みソフトウェアはネットワークドライブリソースから取得できます。

書き込み用の SDDiskTool 実行ファイル SD\_Firmware\_Tool.exe を開き、SD カードを挿入します。

警告: 複数のリムーバブルストレージデバイスを挿入している場合は、書き込む SD カードを正確に選択してください。そうしないと、他のストレージデバイスの内容が上書きされる可能性があります。

書き込む SD カードを正しく選択し、機能モードを SD 起動に設定した後、書き込む RK 形式の完全イメージを選択し、「開始」をクリックして SD カードへのイメージの書き込みを開始します。

ヒント: リリースされたイメージは zip 形式で圧縮されているので、解凍後の img 形式のイメージを使用して書き込んでください。



SD カードの書き込みが完了するまで気長に待ちます。イメージが大きい場合は、書き込み時間がそれに  
応じて長くなります。

## 第 20 章 eMMC 启动镜像書き込み

eMMC の書き込みには 2 つの方法があり、一つは SD カードを使用して eMMC にイメージを書き込みする方法と、もう一つは USB 書き込みツールを使用する方法です。

SD カードの作成には SDDiskTool を使用し、パーティションの書き込みには対応しておらず、SDK で生成された完全な update.img イメージのみを書き込みできます。その原理は、SD カードに完全な起動イメージを書き込み、さらに update.img を SD カードに残しておき、システムが SD カードから起動した後、eMMC に update.img イメージを書き込みするというものです。

USB 書き込みツールを使用して eMMC にイメージを書き込みする方法も 2 つあり、一つは Rockchip の開発ツール RKDevTool (Windows 版) または Linux\_Upgrade\_Tool (Linux 版) を使用する方法、もう一つは Rockchip の USB 量産書き込みツール FactoryTool (Windows のみ) を使用する方法です。

RKDevTool と Linux\_Upgrade\_Tool は機能が似ており、パーティション書き込みと完全な update.img イメージの書き込みをサポートしています。これら 2 つのツールは、開発プロセス中のイメージ書き込みに主に使用され、書き込み機能だけでなく、他の高度な機能も持っていますが、一度に一枚のボードのみ操作できます。

FactoryTool 量産ツールは、工場での生産に主に使用され、update.img イメージのみをサポートし、複数のボードを同時に書き込みでき、操作が簡単です。

上記の書き込み方法を総合的に分析すると、FactoryTool 量産ツールは工場での大量書き込みに適しており、RKDevTool と Linux\_Upgrade\_Tool は開発時や少量書き込みに適しています。SD カードを使用した書き込み方法は推奨されません。

ヒント: eMMC ファームウェアのダウンロードと量産書き込みのために、USB-OTG ポートの予約を強く推奨します。



## 20.1 SD カードを使用した eMMC へのイメージ書き込み

注意: 現在、SD カードを使用した eMMC へのイメージ書き込みは Buildroot イメージのみをサポートしています。

量産時には、SD カードを利用して eMMC の書き込みを行うことができます。その原理は、まず SD カードに完全なシステム起動イメージを書き込みし、次に update.img イメージを SD カードに保存し、ボードの電源が入った後、SD カードから優先してオペレーティングシステムを起動し、オペレーティングシステムの起動が完了した後、SD カードの update.img イメージを eMMC に書き込みするというものです。

まず、書き込み用の SD カードを作成する必要があります。これを SD アップグレードカードと呼び、SDDiskTool ツールとボードに対応する RK 形式の完全なイメージを使用して作成します。

注意: SD アップグレードカードの作成には、SDDiskTool\_v1.7 以上のバージョンを使用する必要があります。

Rockchip の SD カード書き込みツール SDDiskTool は現在、Windows 版のみが提供されているため、SD アップグレードカードの作成は Windows でのみ行うことができます。文書では Windows 10 21H1 版を使用していますが、他のバージョンの互換性はテストされていません。

注意: 書き込み後の SD カードには多くのパーティションが存在し、SDDiskTool を再度使用して書き込みする際には、PC がブルースクリーンになる可能性があります。書き込み前には、未保存のアプリケーションを先に保存してください。

RK 形式の完全なイメージとは以下のことを指します：

- SDK を使用して直接構築された rockdev/update.img イメージ。
- がリリースした Update.img で終わるシステムイメージ。

SDDiskTool 書き込みソフトウェアは、ネットワークドライブリソースで入手できます。

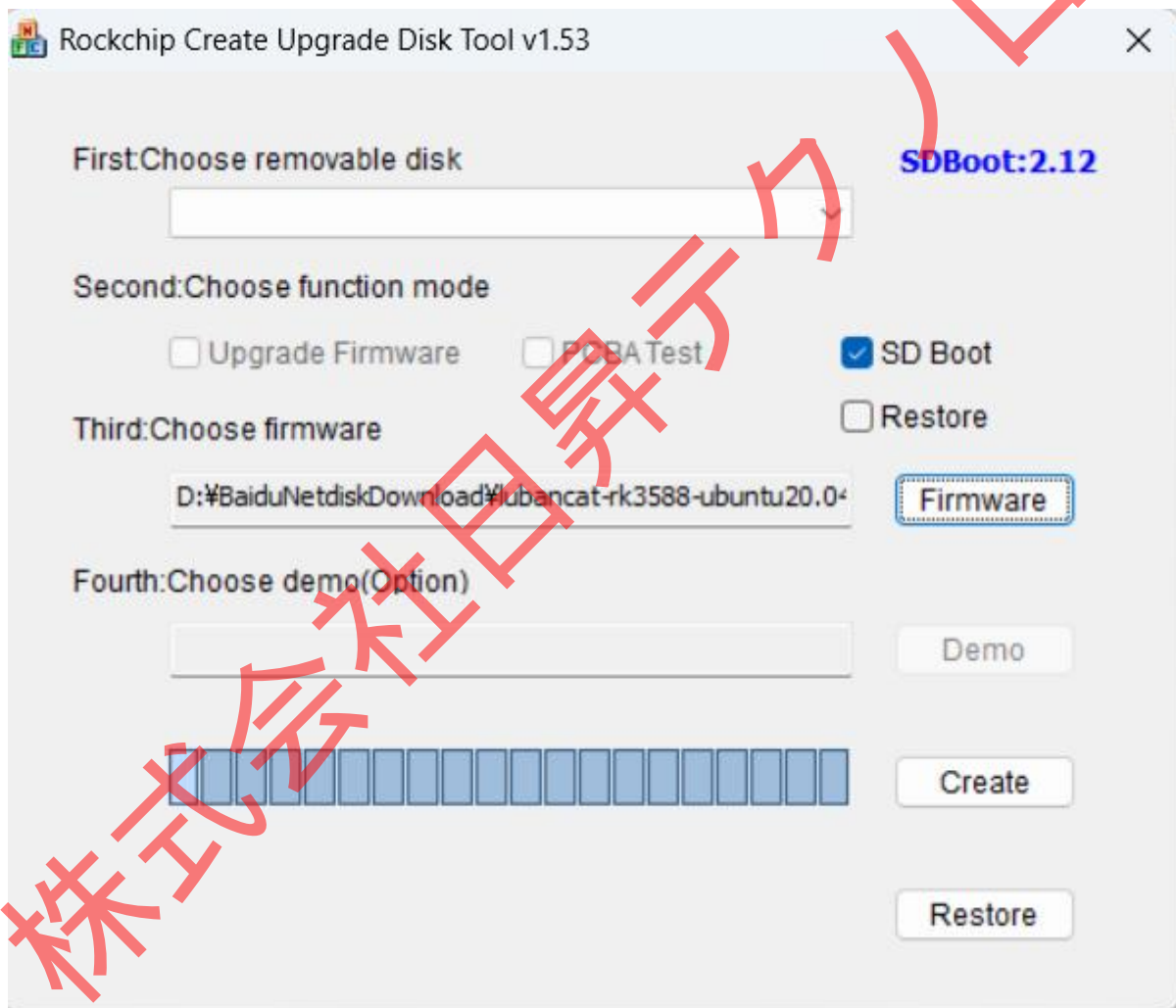
書き込み用の SDDiskTool 実行ファイル SD\_Firmware\_Tool.exe を開いて、SD カードを挿入します。

警告: 複数のリムーバブルストレージデバイスを挿入している場合は、書き込みする SD カードを正しく選

択した後に書き込みを開始してください。そうしないと、他のストレージデバイスの内容が上書きされる可能性があります。

書き込みする SD カードを正しく選択し、機能モードをファームウェアアップグレードに設定し、書き込みする RK 形式の完全イメージを選択した後、SD カードにイメージを書き込みするために開始をクリックします。

ヒント：リリースされたイメージは zip 形式で圧縮されているので、圧縮ファイルを解凍してから img 形式のイメージで書き込みしてください。



SD カードの書き込みが完了するのを待ちます。イメージが大きい場合、書き込みには時間がかかることがあります。

書き込みが完了すると、NTFS 形式の新しいパーティションが作成され、そのパーティションを開くと、

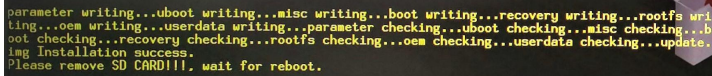
update.img ファイルが sdupdate.img として名前変更されているのが見られます。

書き込み前には、デバッグ用のシリアルポートを接続するか、画面を接続して、書き込み中にプロンプトが表示されるようにします。すべての準備が整ったら、ボードに SD カードを挿入し、再び電源を入れます。

システムが起動すると、画面には Android のロボットのインターフェイスが表示され、parameter waiting... uboot waiting.. というテキストプロンプトが表示されます。これは、対応するパーティションが書き込みされていることを意味します。rootfs パーティションが大きいと、rootfs waiting の段階で長時間停止することがあります。

シリアルターミナルに以下の内容が出力された時、eMMC の書き込みが始まります：

```
1 Starting input-event-daemon: input-event-daemon: Start parsing /etc/input-event-daemon.conf...
2 input-event-daemon: Adding device: /dev/input/event0...
3 input-event-daemon: Adding device: /dev/input/event1...
4 input-event-daemon: Adding device: /dev/input/event2...
5 input-event-daemon: Adding device: /dev/input/event3...
6 input-event-daemon: Adding device: /dev/input/event4...
7 input-event-daemon: Adding device: /dev/input/event5...
8 input-event-daemon: Adding device: /dev/input/event6...
9 input-event-daemon: Adding device: /dev/input/event7...
10 input-event-daemon: Start listening on 8 devices...
11 done
12 [root@RK358X:/]# [ 5.049542] mmcblk0: p1 p2 p3 p4 p5 p6 p7 p8
```



```
parameter writing...uboot writing...misc writing...boot writing...recovery writing...rootfs writing...oem writing...userdata writing...parameter checking...uboot checking...misc checking...boot checking...recovery checking...rootfs checking...oem checking...userdata checking...update.
img Installation success.
Please remove SD CARD!!!, wait for reboot.
```

区域分けて書き込み完成后、各区域分けての検証が行われ、検証完了後には「update.img Installation success」と表示されます。その後、画面とシリアル出力に「Please remove SD CARD!!!, wait for reboot.」と表示されます。

この時点でSDカードを抜き取り、ボードは自動的に再起動し、eMMCに書き込みされたイメージでオペレーティングシステムを起動します。

## 20.2 Rockchip 開発ツールによる USB 経由での eMMC へのイメージ書き込み

ボードに保留されたUSB-OTG接口を通じて、ファームウェアをeMMCに容易に書き込みすることができます。Rockchip開発ツールを使用することで、完全イメージの書き込みだけでなく、区域分けて書き込みも可能であり、開発者にとって開発のリズムを大幅に加速させることができます。

注意: 一部のボードのダウンロード接口は、自動でOTGモードに切り替えることができないか、電源接口と共有しているため、ボードやケース上のシルク印刷は「Download」とされていますが、本質的にはUSB-OTG接口です。理解を容易にするため、以下ではダウンロード接口と総称します。

注意: USBダウンロード方式を使用する場合、工場生産での初回書き込みを除き、マスクロムモードへの

入り方はボタンを押して行います。一部のボードでは、ボタンに「MASKROM」や「MR」とシルク印刷されており、以下では MR ボタンと統一して呼びます。

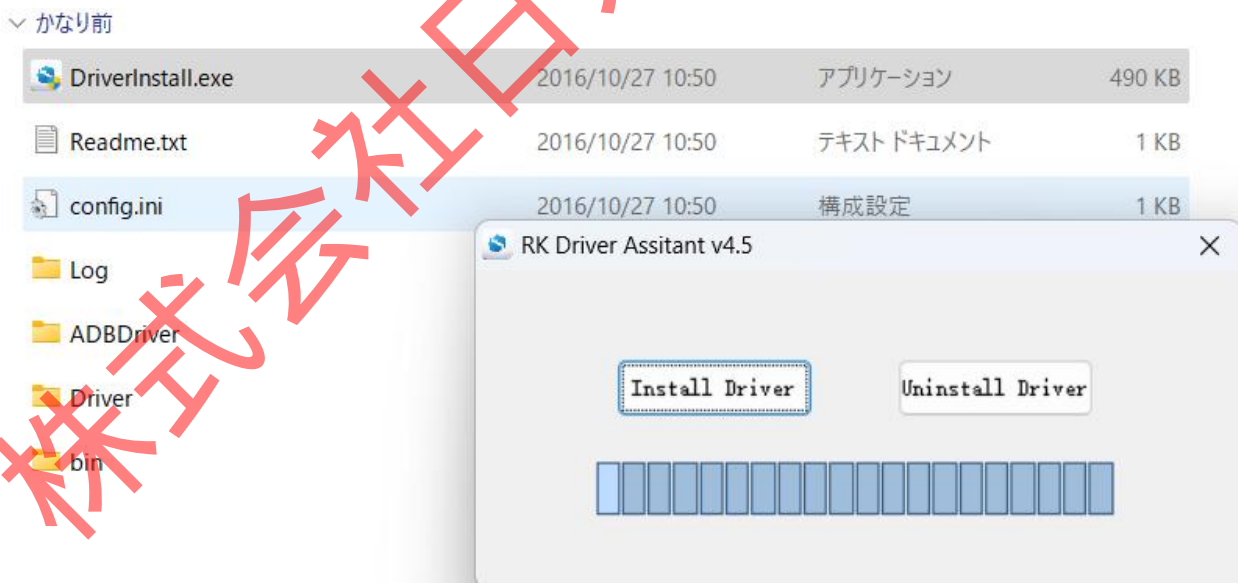
## 20.2.1 RKDevTool (Windows)

開発プロセス中は、コンパイル作業は企業のビルドサーバー上で行われており、ローカルの Windows PC から ssh、ftp などを通じてサーバーに接続します。

このようにローカルで Windows 環境を使用し、リモートで Linux ビルドサーバーを使用するシナリオは、Rockchip 開発ツール RKDevTool を使用するのに非常に適しています。v2.91 以上のバージョンの使用を推奨します。

### 20.2.1.1 準備作業

Windows プラットフォーム用 RockchipUSB デバイスドライバのインストール。ソフトウェアの圧縮ファイルをダウンロードして解凍し、DriverInstall.exe をダブルクリックしてドライバインストール画面に入ります。



インストールをクリックしてドライバのインストールを開始します。以前に古いバージョンのドライバをインストールしたか不明な場合は、先に「ドライバのアンインストール」をクリックして古いバージョン

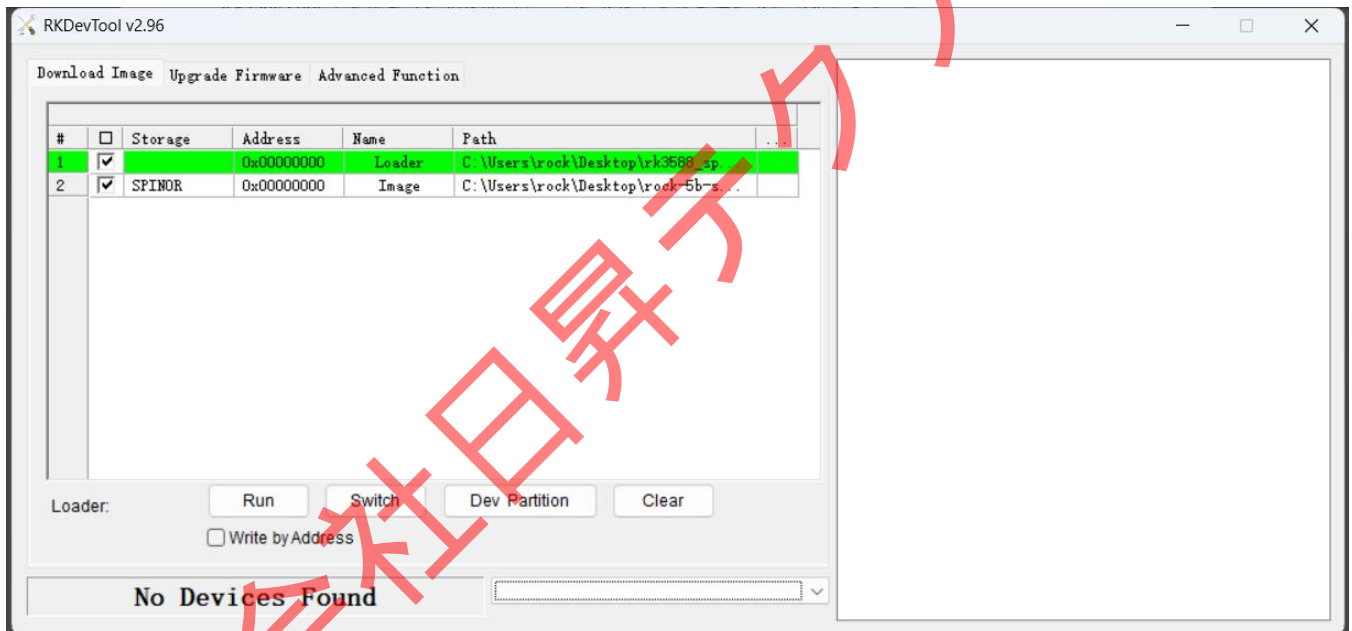
ンのドライバプログラムを削除し、その後「ドライバのインストール」をクリックします。

- 良質な USB ケーブル。USB-A から USB-C、または両端が USB-A のケーブルを、ダウンロードポートのソケット形状と実際の状況に応じて選択します。

- Windows プラットフォーム用 Rockchip 開発ツールのインストール

圧縮ファイルを解凍後、インストール不要で直接使用可能です。RKDevTool.exe をダブルクリックしてソフトウェアインターフェースに入ります。

このソフトウェアには主に 3 つの部分があり、それぞれイメージのダウンロード、ファームウェアのアップグレード、高度な機能が含まれています。



各機能の使用方法は具体的な使用時に説明され、完全な使用方法はツールの圧縮パッケージ内の「開発ツール使用文書」を参照してください。

書き込みモードへの入り方

1. 用意した USB ケーブルを一方はコンピューターに、もう一方は予備として接続します。
2. ボードに電力を供給する可能性のある全ての接続を断ち切ります。例えば、電源ケーブルや他の USB ケーブルなどです。
3. RB ボタンを押し続けながら、用意した USB ケーブルをボードのダウンロードポートに接続し、次に

電源を接続します。ダウンロードポートと電源ポートを共有するボードでは、追加の電源を接続する必要はありませんが、コンピューターの USB ポートが少なくとも 5V 0.5A の供給ができることを確認してください。

4. ソフトウェアが MASKROM デバイスを検出したことを示すプロンプトが表示されたら、さらに 5 秒以上待ってからボタンを離します。

5. 成功しなかった場合は、ステップ 2 から 4 を繰り返します。

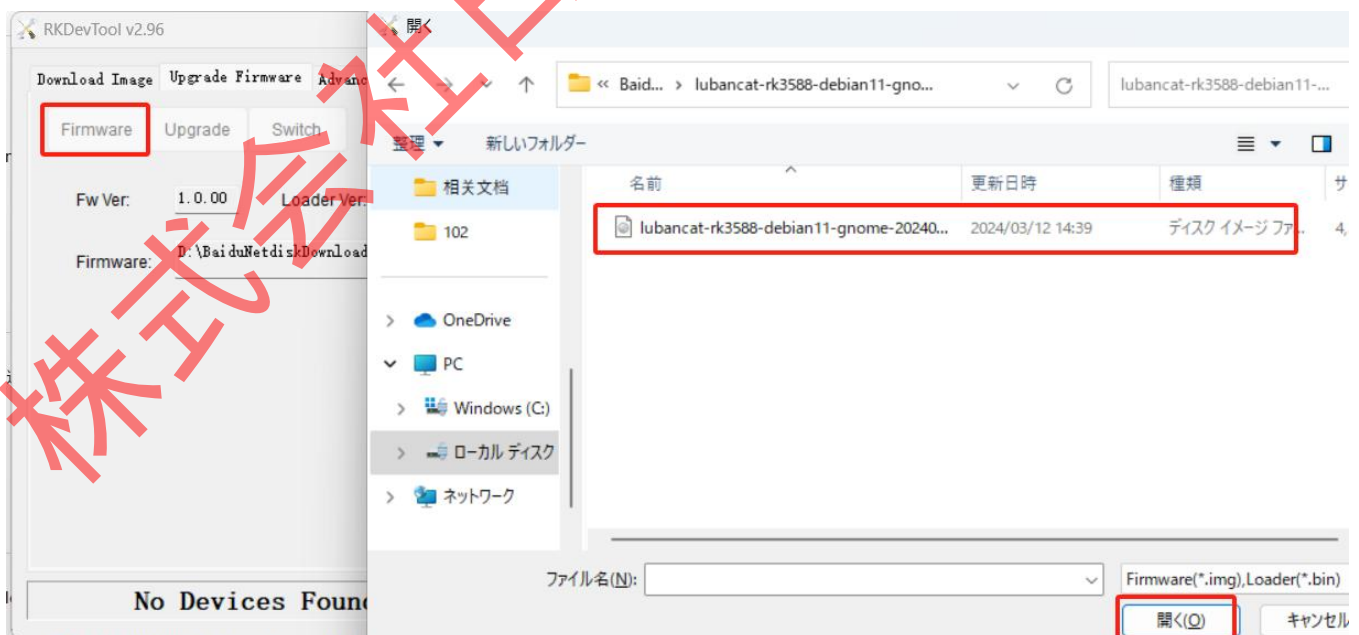
### 20.2.1.2 完全イメージの書き込み

完全イメージには、区域分けてイメージの全ファイルが含まれており、外部への配布や量産時にイメージファイルを管理しやすくなります。その書き込みプロセスは簡単で、ワンクリックで書き込みできます。

RKDevTool 書き込みツールを開いて、ボードを書き込みモードに設定します。

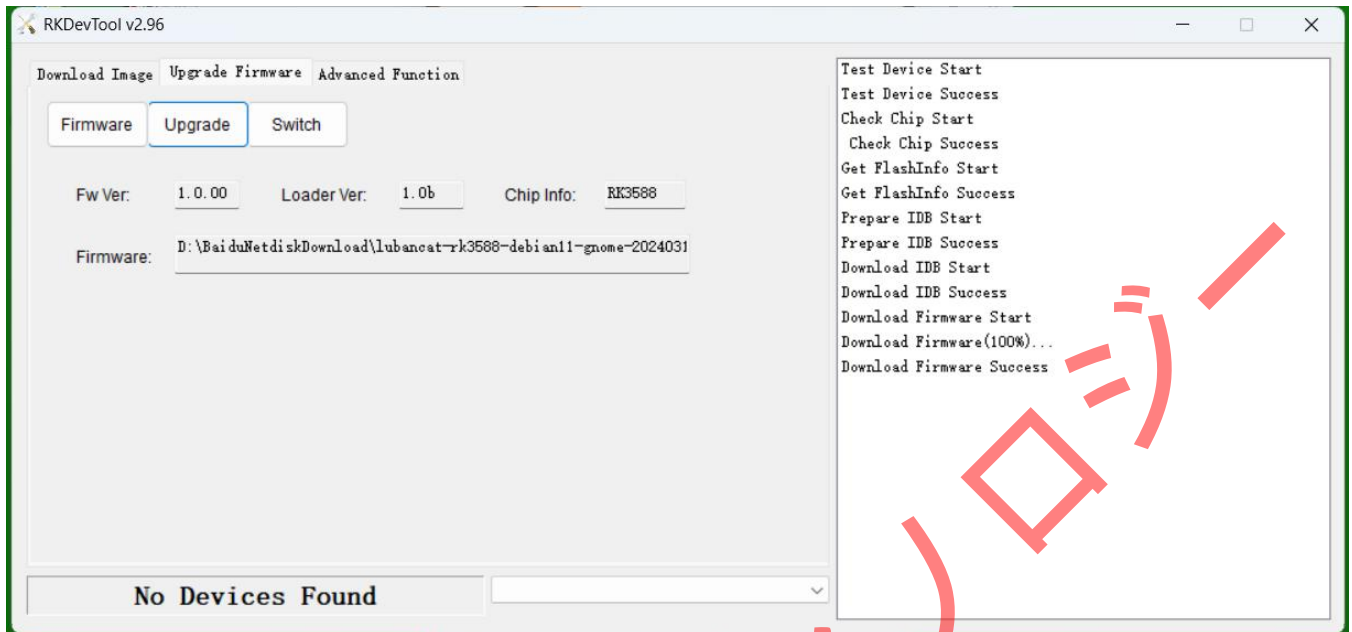
zip 圧縮パッケージ形式でリリースしたシステムイメージをダウンロードし、img 形式のイメージファイルに解凍するか、SDK でコンパイルされた update.img イメージを直接使用します。

ファームウェアの選択で書き込みするイメージを選び、ここでは update.img を例として挙げ、開きます。



ファームウェアのロードが完了するのを待ち、アップグレードをクリックしてファームウェアの書き込み

を開始します。



### 20.2.1.3 区域分けてイメージの書き込み

開発段階では、区域分けてイメージの書き込みにより、書き込みプロセス中に消費される時間を大幅に短縮できます。全イメージを完全に書き込みした後、何かの部分を変更した場合は、その部分だけを個別に書き込みし、完全なイメージを再書き込みする必要がありません。

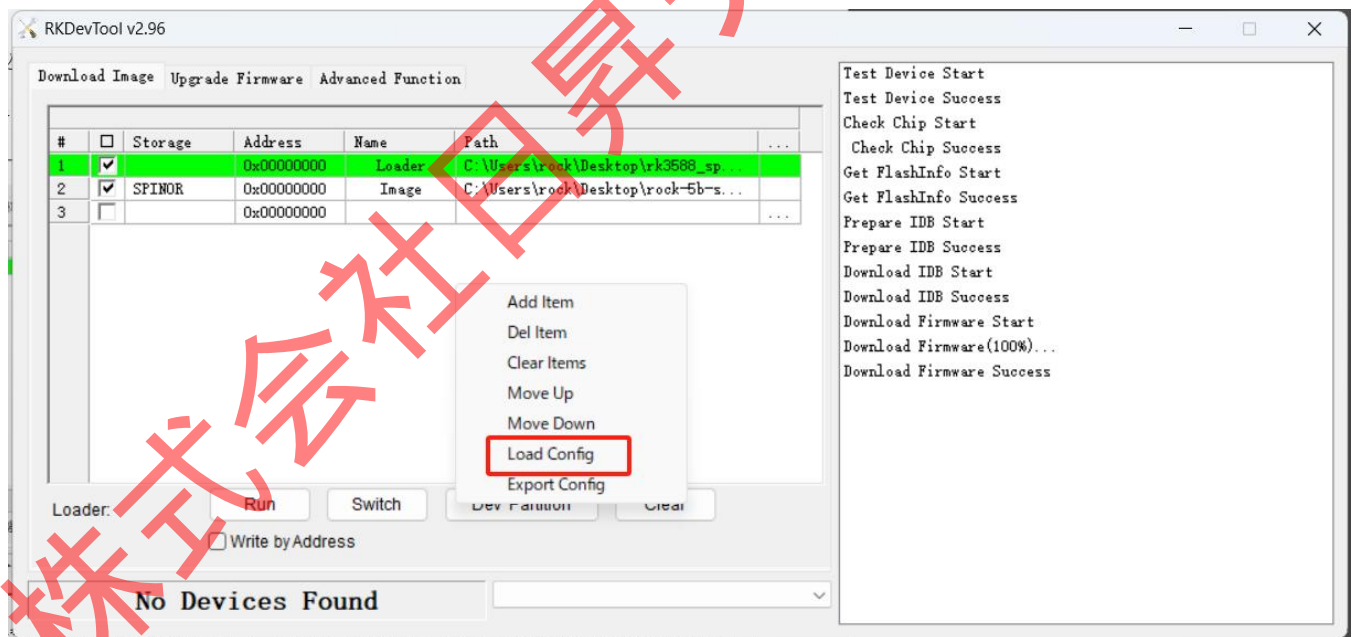
RK ファームウェアの構成ルールは GPT 区域分けて表に基づいており、これらの個別のファイルは SDK のコンパイル後に rockdev/ディレクトリに格納されます。以下の図に示されるファイルは参考用です。イメージのバージョンによって異なる場合がありますので、具体的なルールは区域分けて表を参照してください。



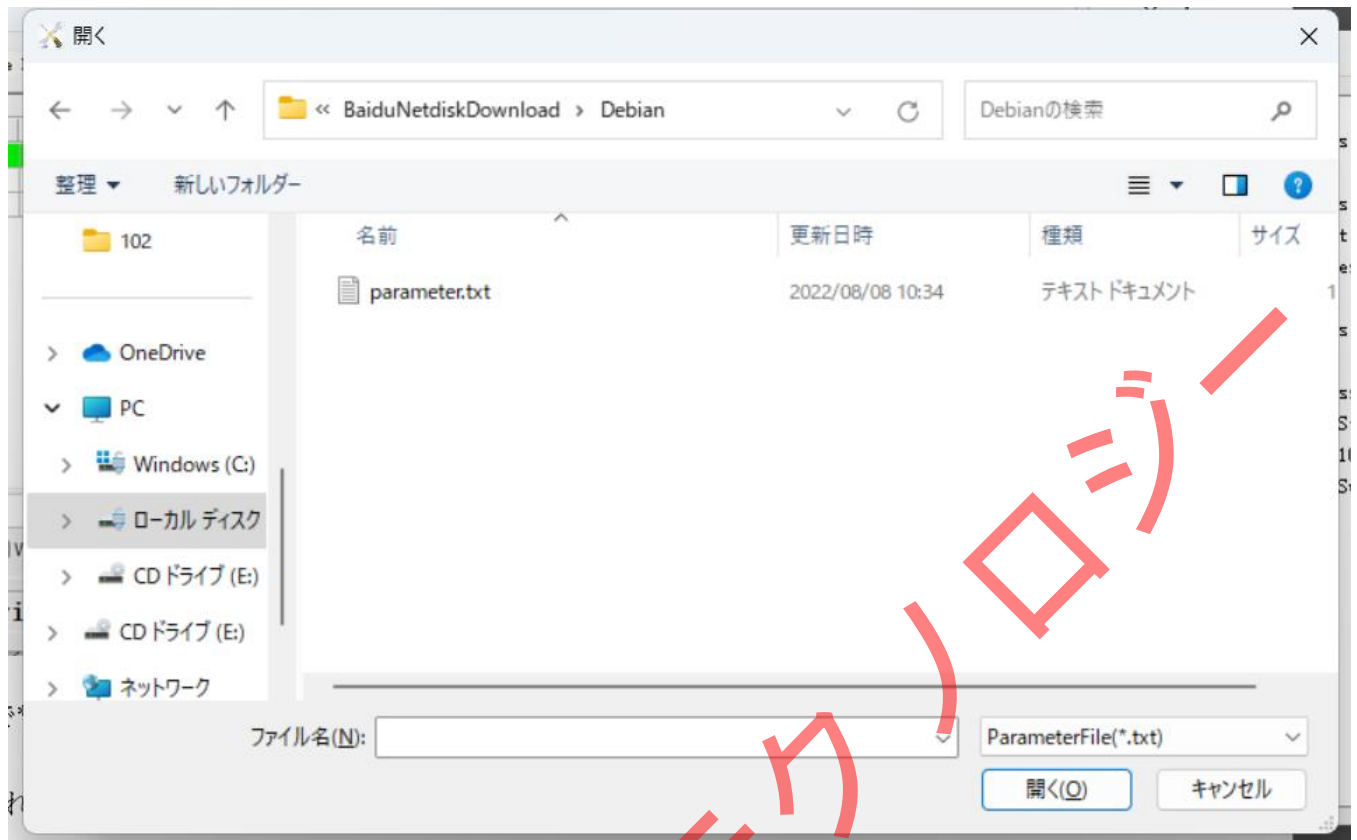
boot.img	2022/08/08 10:34	ディスクイメージファ...	6,928 KB
MiniLoaderAll.bin	2022/08/08 10:34	BIN ファイル	199 KB
misc.img	2022/08/08 10:34	ディスクイメージファ...	48 KB
oem.img	2022/08/08 10:34	ディスクイメージファ...	10,240 KB
parameter.txt	2022/08/08 10:34	テキストドキュメント	1 KB
recovery.img	2022/08/08 10:35	ディスクイメージファ...	13,526 KB
rootfs.img	2022/08/08 11:07	ディスクイメージファ...	805,232 KB
trust.img	2022/08/08 11:07	ディスクイメージファ...	4,096 KB
uboot.img	2022/08/08 11:07	ディスクイメージファ...	4,096 KB
userdata.img	2022/08/08 11:08	ディスクイメージファ...	5,120 KB

RKDevTool 書き込みツールを開いて、ボードを書き込みモードに設定します。

画面上の空白箇所を右クリックして「配置をインポート」を選択します。



ファイルタイプで\*.txtを選び、区域分けてファイルが存在するディレクトリに進んで区域分けて表ファイルを開きます。これにより、配置を成功にインポートしました。



区域分けて表の設定ファイルに基づき、最後の列の...をクリックして対応するファイルを選択します。対応関係は以下の表の通りです。

注意: システム区域分けての一部は以下の表に示される項目よりも少ない場合があります。具体的には区域分けて表 parameter.txt の内容を基準とし、異なるオペレーティングシステムの区域分けて表は混在させてはいけません。

RKDevTool v2.96

Download Image Upgrade Firmware Advanced Function

#	<input type="checkbox"/>	Storage	Address	Name	Path	...
1	<input checked="" type="checkbox"/>		0x00000000	Loader	D:\BaiduNetdiskDownload\Debian\...	
2	<input checked="" type="checkbox"/>		0x00000000	Parameter	D:\BaiduNetdiskDownload\Debian\...	
3	<input type="checkbox"/>		0x00004000	uboot	D:\BaiduNetdiskDownload\Debian\...	
4	<input type="checkbox"/>		0x00006000	misc	D:\BaiduNetdiskDownload\Debian\...	
5	<input type="checkbox"/>		0x00008000	boot	D:\BaiduNetdiskDownload\Debian\...	
6	<input type="checkbox"/>		0x00028000	recovery	D:\BaiduNetdiskDownload\Debian\...	
7	<input type="checkbox"/>		0x00048000	backup		
8	<input type="checkbox"/>		0x00058000	rootfs	D:\BaiduNetdiskDownload\Debian\...	
9	<input type="checkbox"/>		0x00058000	oem	D:\BaiduNetdiskDownload\Debian\...	
10	<input type="checkbox"/>		0x00098000	userdata	D:\BaiduNetdiskDownload\Debian\...	

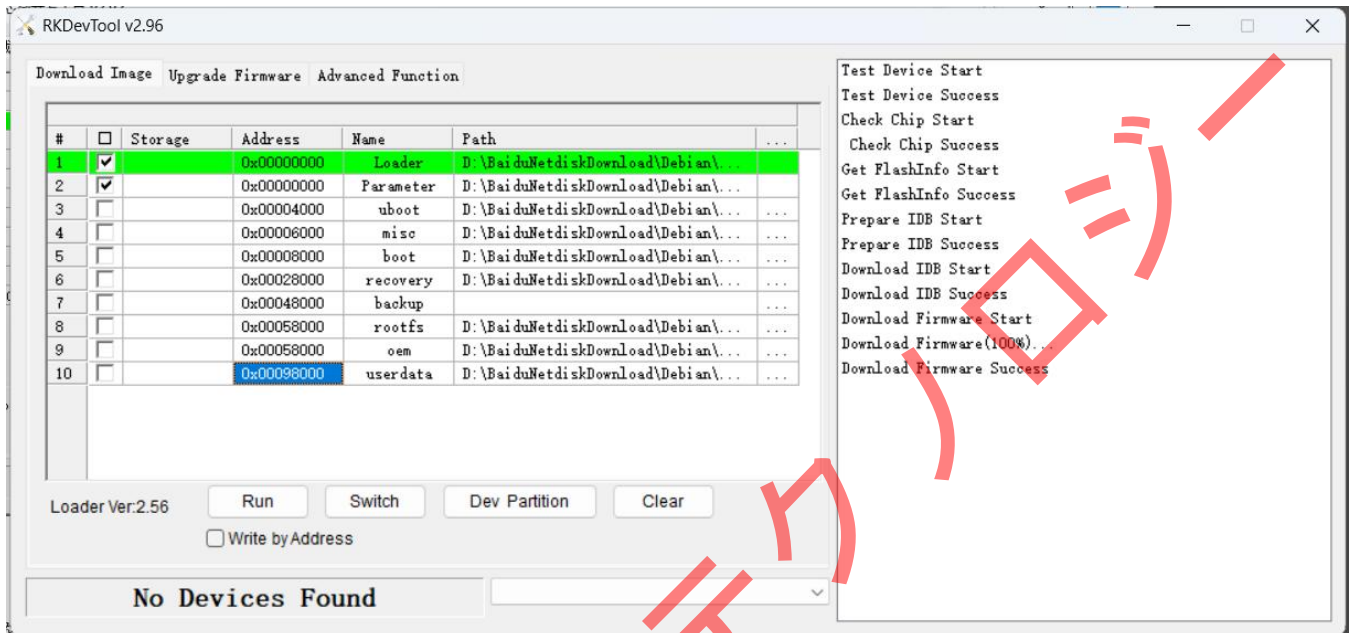
Loader Ver:2.56    Run    Switch    Dev Partition    Clear

Write by Address

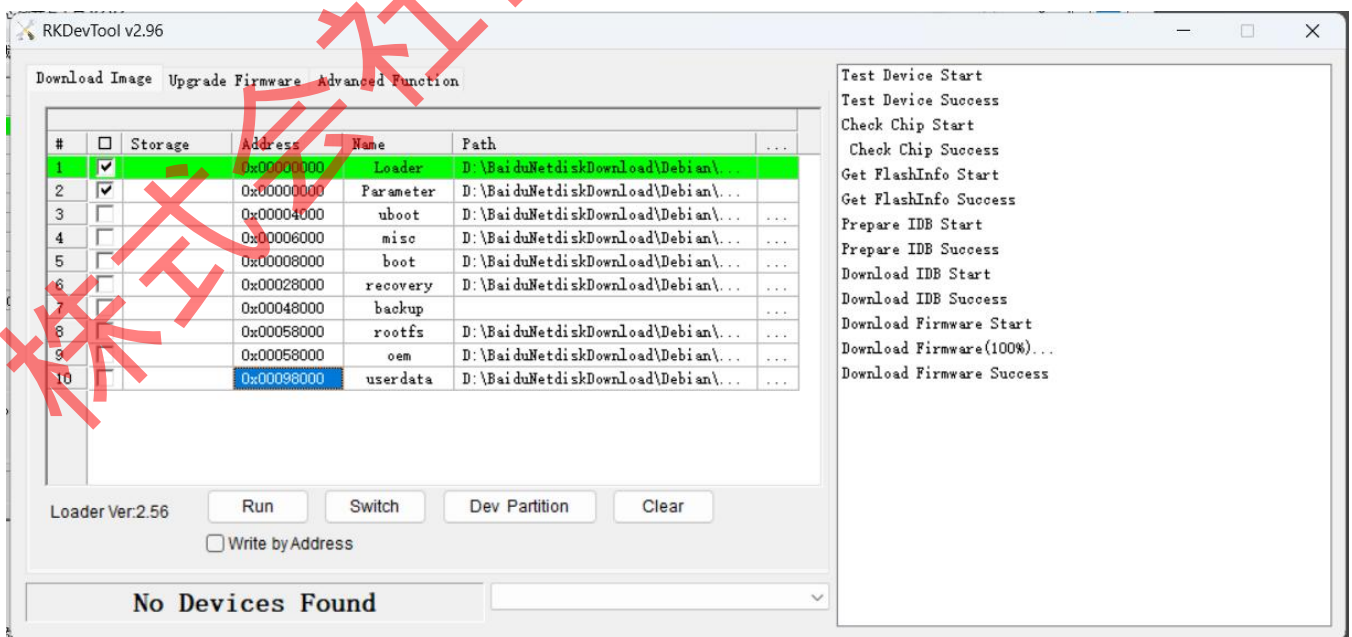
No Devices Found

名前	ファイル名	備考
bootloader	MiniLoaderAll.bin	
parameter	parameter.txt	
uboot	uboot.img	
misc	misc.img	一部のシステムにはこのパーティションが存在しません。
boot	boot.img	
recovery	recovery.img	一部のシステムにはこのパーティションが存在しません。
backup	追加せず、保留する。	
rootfs	rootfs.img	個別に提供されるファイルシステムの名前は変わることがありますが、すべて rootfs-xxx.img です。

oem	oem.img	一部のシステムにはこのパーティションが存在しません。
userdata	userdata.img	一部のシステムにはこのパーティションが存在しません。



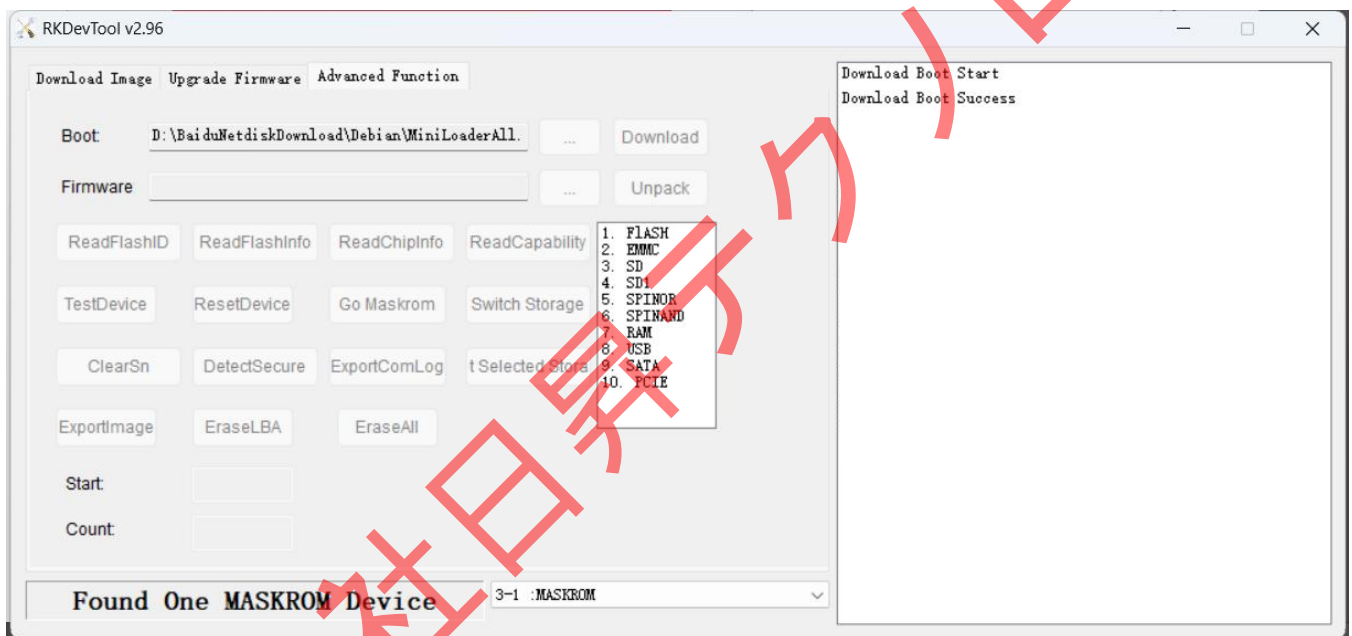
個々の区域分けてまたは部分区域分けてを書き込みする際は、書き込みしたい区域分けてと loader 区域分けてを選択して、「実行」をクリックして書き込みを開始します。以下の図は rootfs 区域分けてのみを個別に書き込みする例です。



## 20.2.1.4 高度な機能の紹介

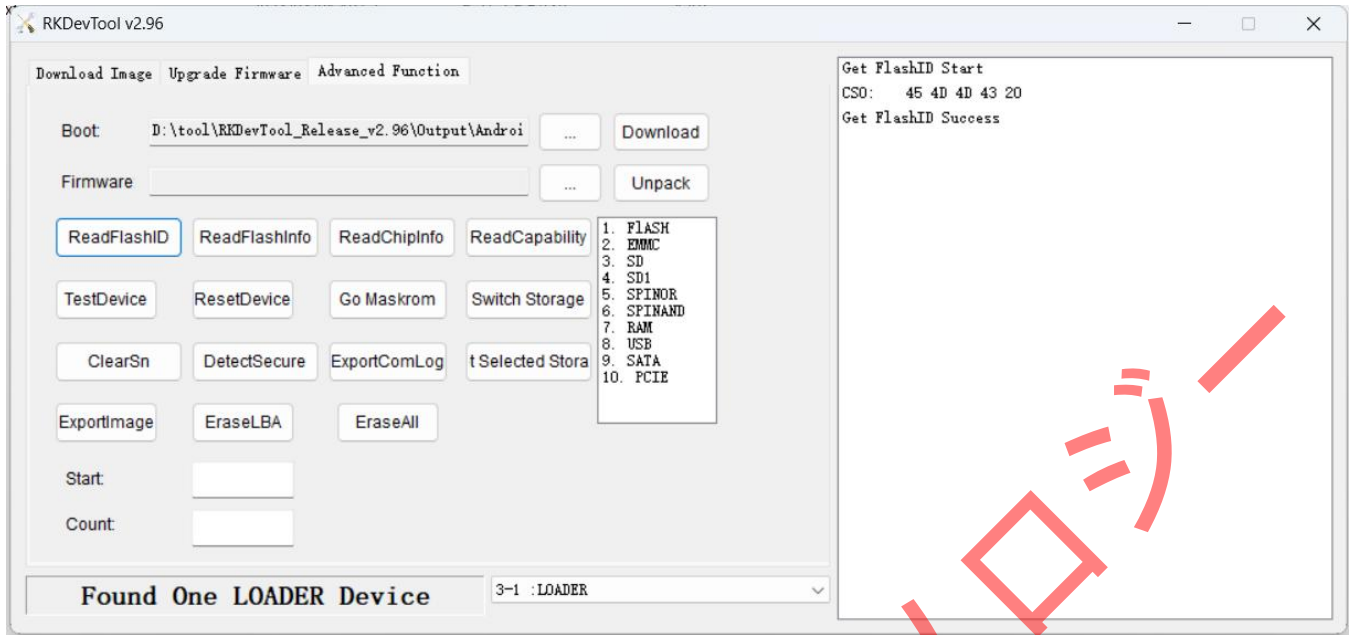
高度な機能インターフェースには主に2つの機能があります。一つはBootのロードを通じてボードとのインタラクションを実現するもの、もう一つは完全なファームウェアの解体です。解体部分の内容は、完全なイメージの解体とパッキングの章で別途説明します。

開発ツールが提供する高度な機能を使用するには、まずBootイメージをダウンロードすることが第一歩です。しかし、Bootイメージのダウンロードという表現は正確ではありません。正確には、bootloaderをメモリにロードしてボードとのインタラクションを実現することです。

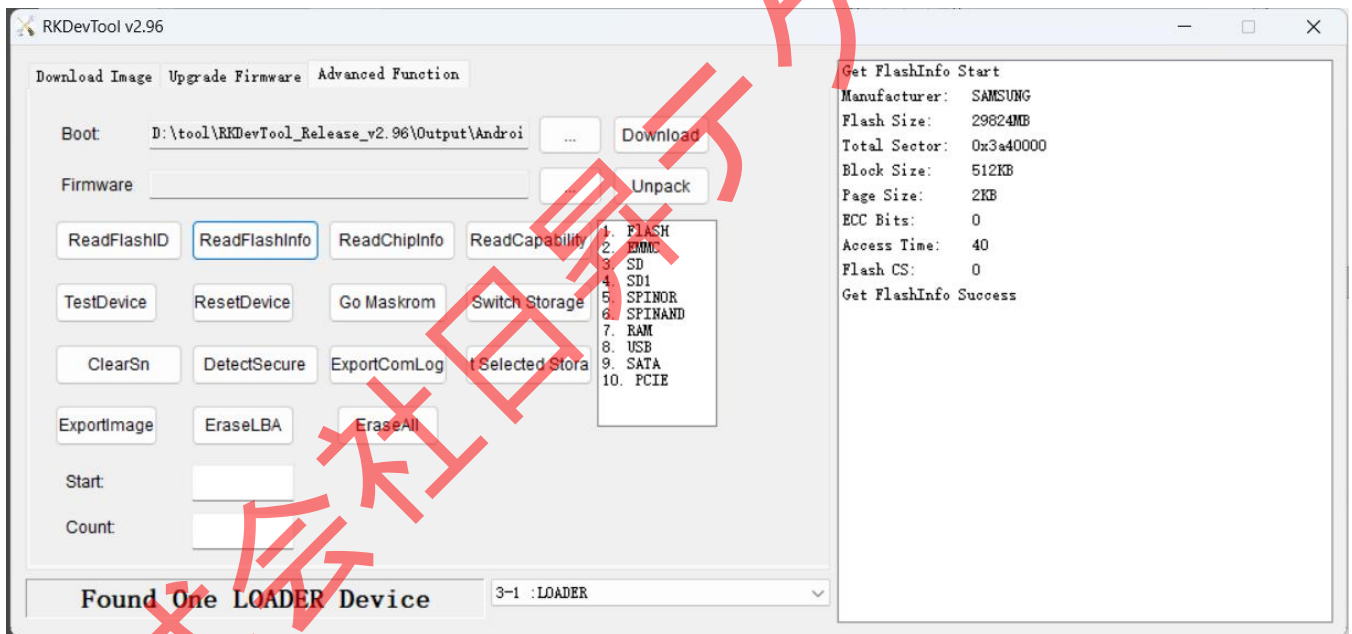


チップに対応する MiniLoaderAll.bin ファームウェアを選択し、ボードにダウンロードします。

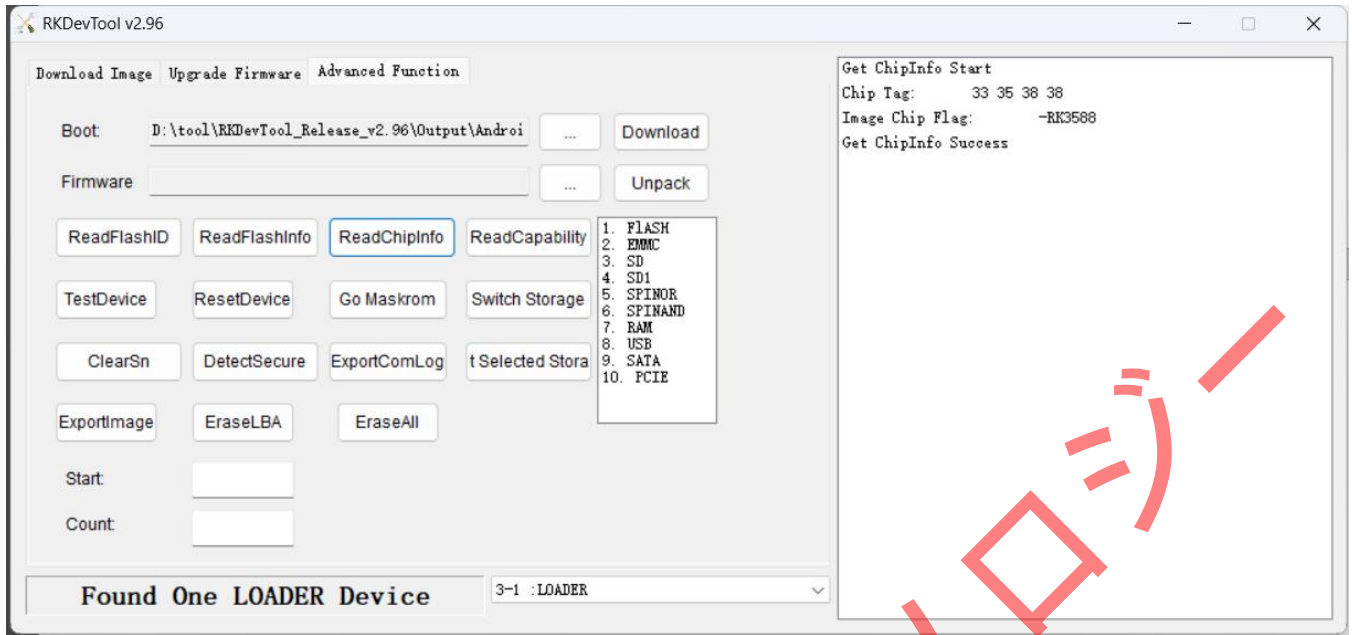
- FlashID の読取り



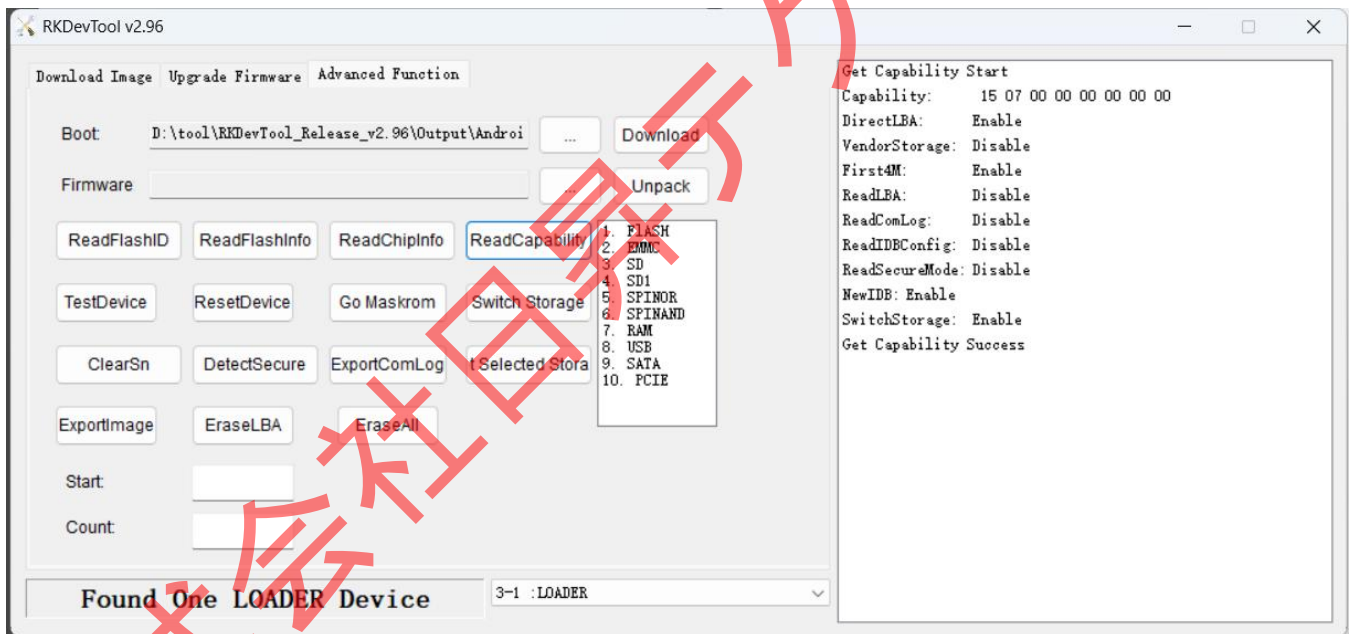
- Flash情報の読取り、読み取られたメーカー情報は誤っていますが、容量などの情報は正確です



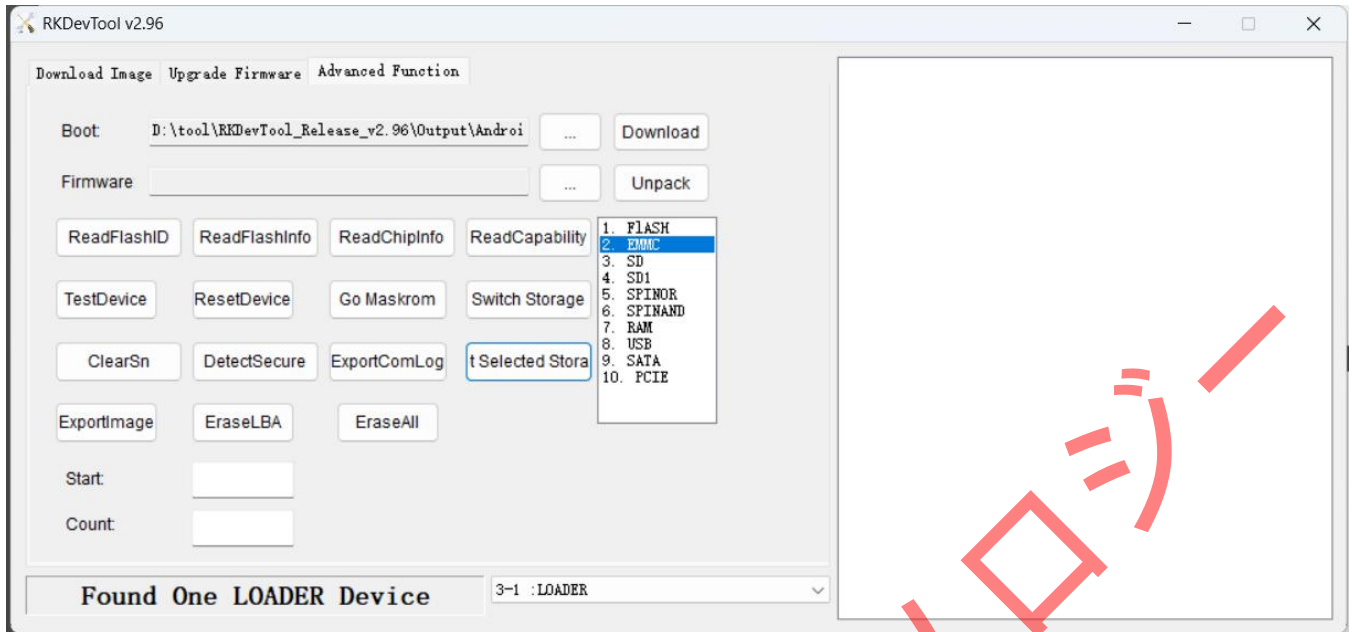
- チップ情報の読取り



- Capability の読取り



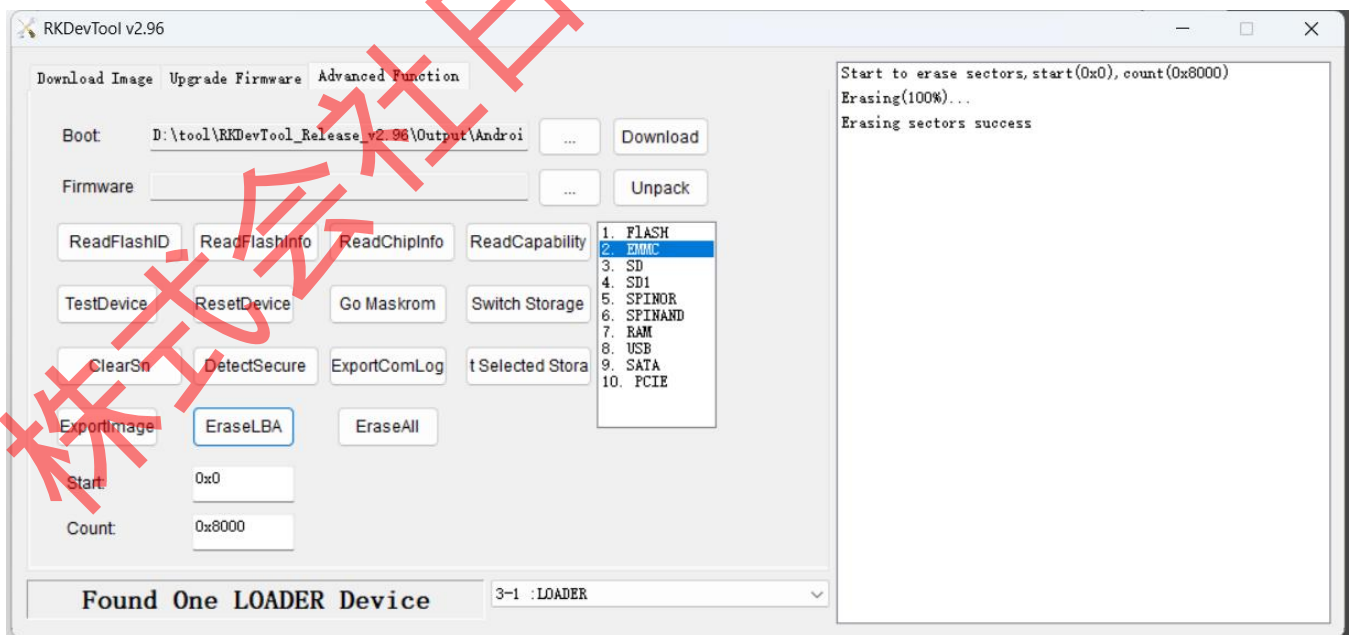
- 現在のストレージの取得



セクターの消去、「開始セクター」と「セクター数」に基づいてセクターを消去します。eMMCのみをサポートします。

まず現在のストレージデバイスを取得し、EMMCを選択してから「ストレージの切替」をクリックし、これで操作対象がeMMCになります。

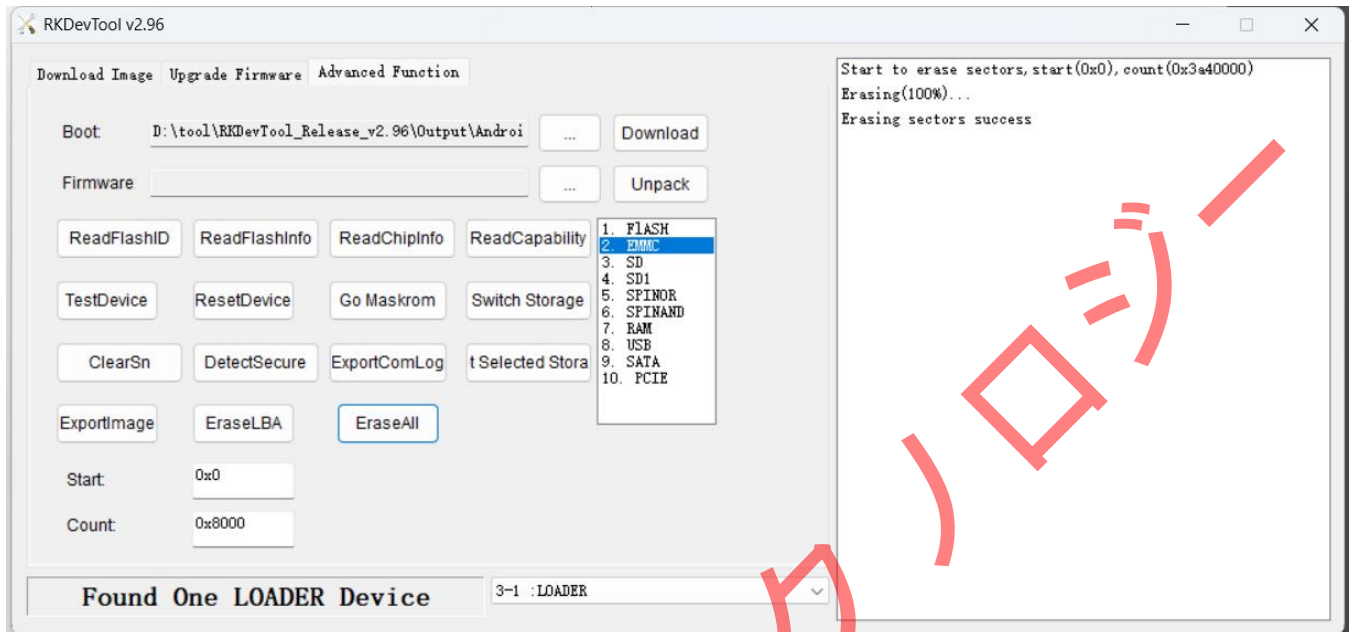
開始セクターとセクター数を設定し、セクターを消去します。



全てを消去することにより、対応するストレージデバイスの全セクタを消去します。これはeMMC/nor/nandに対応しています。



まず現在のストレージデバイスを取得し、EMMCまたは他のストレージデバイスを選択してから「ストレージを切り替える」をクリックします。これにより、選択したストレージが操作対象になります。



## 20.2.2 Linux\_Upgrade\_Tool (Linux)

ローカルのLinuxオペレーティングシステムのPCやLinuxの仮想マシン上で開発しているユーザーにとって、Linux\_Upgrade\_Toolを使うことで、オペレーティングシステムの環境を切り替える必要がなく、非常に便利です。

### 20.2.2.1 Linux 開発ツールのインストール

手に入れたLinux\_Upgrade\_Toolの圧縮ファイルを以下のコマンドで解凍し、システムのバイナリパッケージが置かれているパスに移動させれば、このツールを直接使用することができます。

```

1 # 圧縮ファイルを解凍する、ここではv2.1のバージョンを使用
2 unzip Linux_Upgrade_Tool_v2.1.zip
3
4 # 実行可能な権限を付与
5 chmod 775 Linux_Upgrade_Tool_v2.1/upgrade_tool
  
```

```
6
7 # システムのバイナリパッケージディレクトリにコピー
8 sudo cp Linux_Upgrade_Tool_v2.1/upgrade_tool /usr/local/bin/
9
10 # ツールが正常に使用できるか確認、バージョン番号が返ってくれば正常
11 upgrade_tool -V
12
13 # ダウンロードツールの使用情報を表示
14 upgrade_tool -h
```

### 20.2.2.2 PC に接続されたボードの確認

ダウンロードボタンを押しながら、USB ケーブルをボードのダウンロードポートに接続し、ボードがダウンロードモードに入るまで待ちます。

```
1 # 以下のコマンドを入力して、ボードが接続されているか確認
2 upgrade_tool LD
3
4 # 表示される情報
5 Program Log will save in the /home/hh/upgrade_tool/log/
6 List of rockusb connected(1)
7 DevNo=1 Vid=0x2207,Pid=0x350a,LocationID=32 Mode=Maskrom SerialNo=rockchip
```

表示された情報から、デバイス番号 1、ID は 32 で、現在 Maskrom モードにある 1 台のデバイスが接続されていることがわかります。

ボードが正しく接続された後、ファームウェアの書き込みが可能になります。

ここでは、SDK で生成されたイメージを例に、SDK の rockdev ディレクトリで以下の操作を行います。

### 20.2.2.3 完全イメージの書き込み

rockdev ディレクトリで、ボードが Maskrom モードにあることを確認した後、直接ボードに完全イメージを書き込みします。

```
1 # 完全イメージを書き込み
2 upgrade_tool uf update.img
3
4 # 表示される情報
5 Program Log will save in the /home/hh/upgrade_tool/log/
6 Loading firmware...
7 Support Type:RK3588 FW Ver:1.0.00 FW Time:2022-09-19 11:45:09
8 Loader ver:1.01 Loader Time:2022-09-19 11:41:11
9 Upgrade firmware ok.
```

書き込みが完了すると、ボードは自動的に再起動します

### 20.2.2.4 区域分けてイメージの書き込み

rockdev ディレクトリで、ボードに区域分けてイメージを書き込みます。書き込み前に、ボードが Maskrom モードにあることを確認します。

```
1 # loader を書き込み、デバイスを再起動しない、どの区域分けてをダウンロードする場合でも実行
2 upgrade_tool ul MiniLoaderAll.bin -noreset
3
4 # 区域分けて表をダウンロード
5 upgrade_tool di -p parameter.txt
```

6

7 # 個別の区域分けてをダウンロード、1つのコマンドで1つの区域分けてをダウンロード

8 `upgrade_tool di -u uboot.img`9 `upgrade_tool di -b boot.img`10 `upgrade_tool di -r recovery.img`11 `upgrade_tool di -m misc.img`12 `upgrade_tool di -oem oem.img`13 `upgrade_tool di -userdata userdata.img`14 `upgrade_tool di -rootfs rootfs.img`

15

16 # 複数の区域分けてを連続でダウンロード

17 `upgrade_tool di -u uboot.img -b boot.img`

18

19 # デバイスを再起動

20 `upgrade_tool rd`

## 20.2.2.5 その他のソフトウェア操作

### 20.2.2.5.1 loader をメモリにロード

デバイスが Maskrom モードにある場合、最初に Boot をロードして通信を行う必要があります。

```
upgrade_tool db MiniLoaderAll.bin
```

#### 20.2.2.5.2 デバイス情報の読み取り

```
1 # ストレージ情報を読み取り
2 upgrade_tool rfi
3
4 # チップ id を読み取り
5 upgrade_tool rci
6
7 # 区域分けて表を読み取り
8 upgrade_tool pl
```

#### 20.2.2.5.3 アドレスによるファイルの読み書き

```
1 # LBA 0x12000 にファイルを書き込み
2 upgrade_tool wl 0x12000 oem.img
3
4 # アドレスからデータを読み取り、ファイルに保存、例: 0x12000 から 0x2000 セクターのデータを
out.img に読み取り
5 upgrade_tool rl 0x12000 0x2000 out.img
```

#### 20.2.2.5.4 デバイスの消去

```
1 # ストレージの全データを消去、maskrom モードで実行、Boot を事前にダウンロードする必要はない
2 upgrade_tool ef rkxxloader.bin
3
4 # アドレスによるセクター消去、emmc ストレージのみサポート、例: 0 セクターから 0x2000 セクターを
```

消去

5 upgrade\_tool el 0 0x2000

## 20.3 USB 量産ツールによる eMMC へのイメージ書き込み

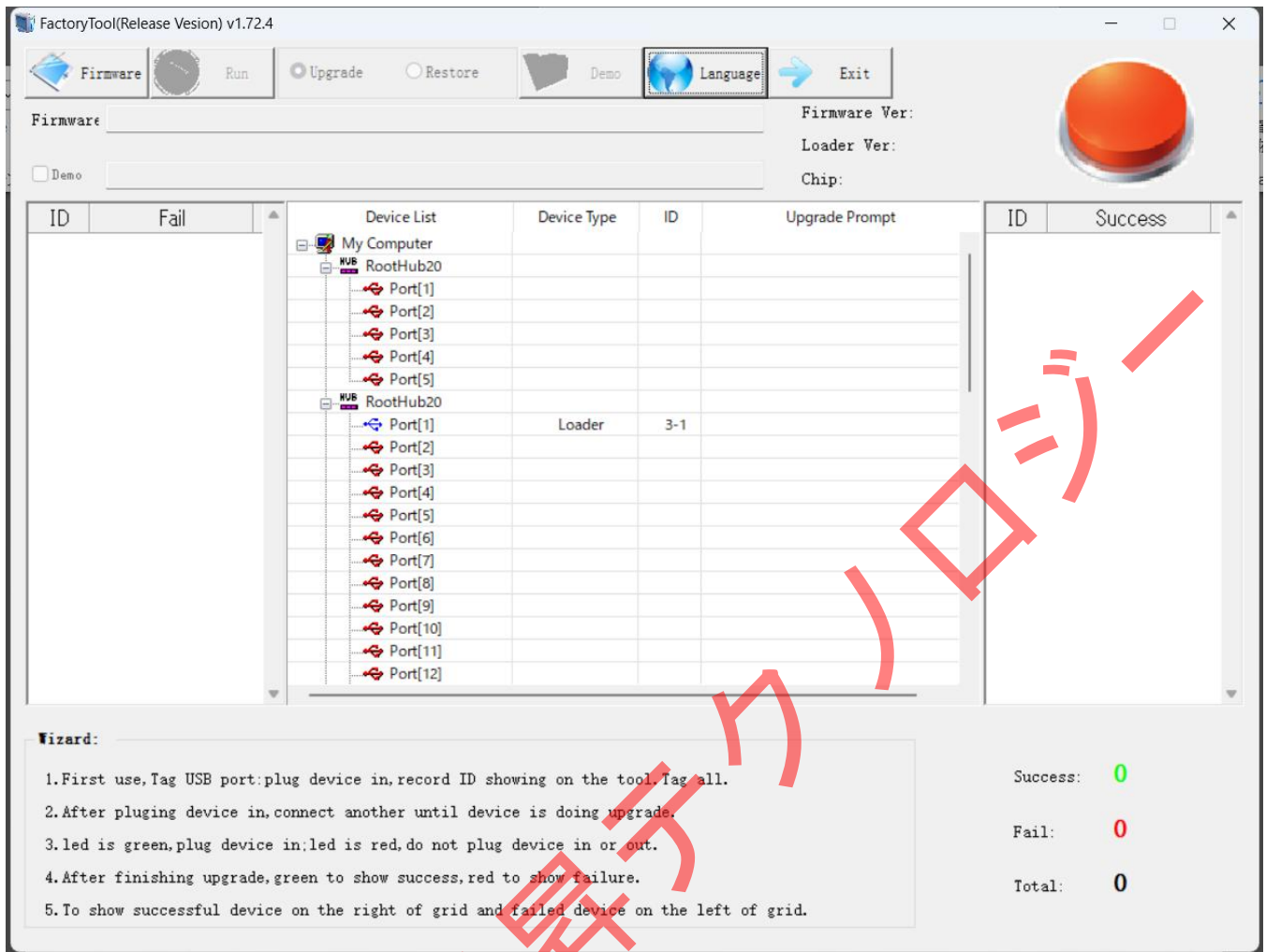
USB 量産ツール FactoryTool は Windows プラットフォームのみをサポートし、RKDevTool の準備作業と同様に、最初に DriverInstall ドライバをインストールし、対応する USB ケーブルを準備します。

FactoryTool を使用して複数のボードに同時にダウンロードするため、複数のダウンロードケーブルを準備します。USB ポートが少ないコンピュータでは、USB-HUB を使用して USB ポートの数を増やすことができます。

以下に FactoryTool の使用方法を説明します。

技術サポートに連絡して FactoryTool ソフトウェアパッケージの圧縮ファイルを取得し、ローカルに解凍します。インストールは必要ありません。

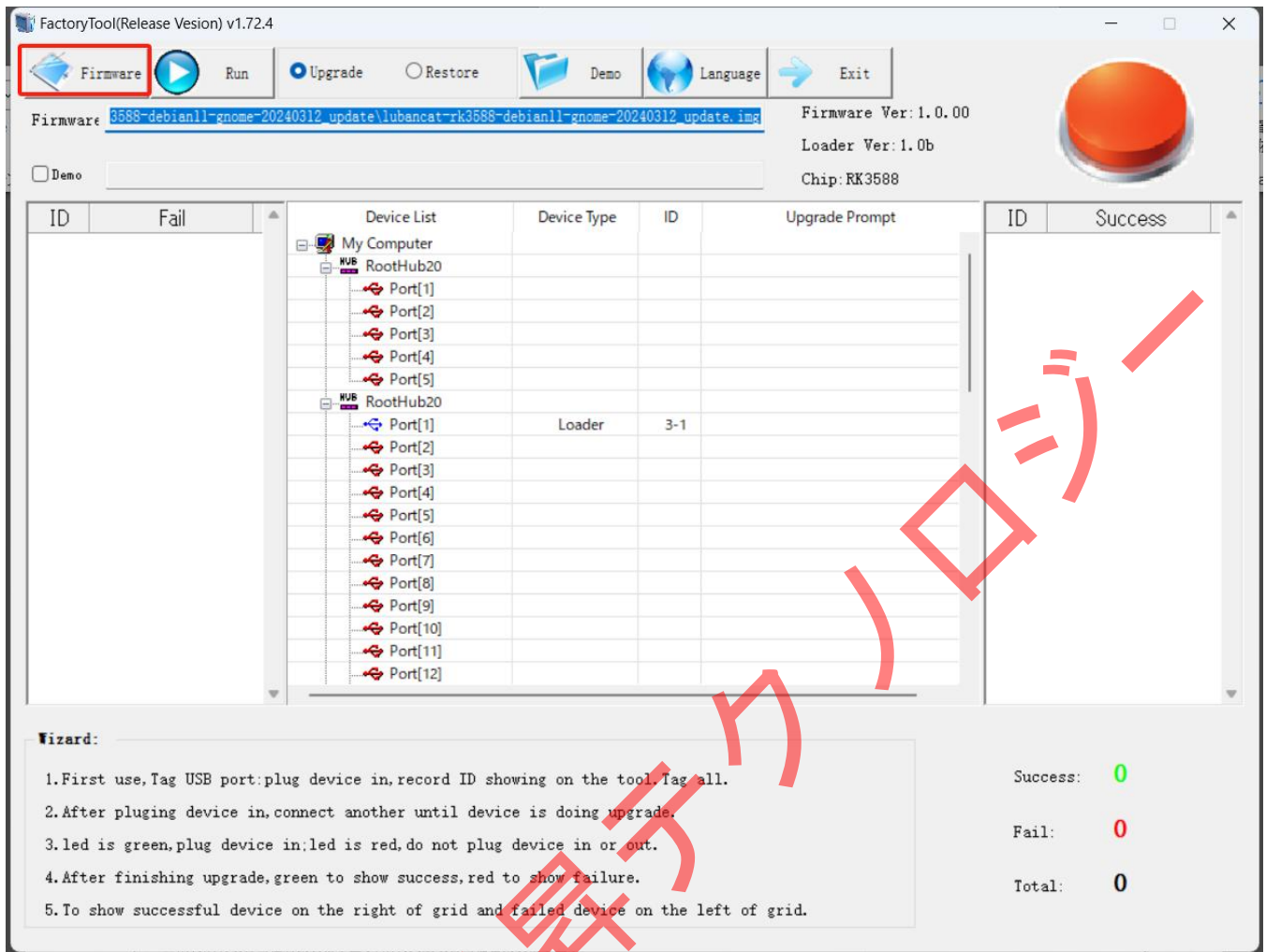
FactoryTool.exe をダブルクリックしてソフトウェアのインターフェースに入ります。



ソフトウェアのインターフェースは整然としており、3つの主要なエリアに分かれています。上部は設定エリアで、書き込むファームウェアを選択し、書き込み機能を開始するためのものです。中央は機能エリアで、USBポートの情報や書き込みの進行状況を表示し、左右にはそれぞれ書き込み成功と失敗のデバイスが表示されます。下部はヒントエリアで、ソフトウェアの使用方法を簡潔に説明し、書き込み回数や成功・失敗の回数を記録する小さなセクションがあります。

インターフェースの説明が終わったので、実際の操作を開始します。

「ファームウェア」ボタンをクリックし、書き込むイメージを選択します。ここではRK形式でパッケージされた完全なイメージである update.img を例とします。



FactoryTool(Relase Vesion) v1.72.4

Firmware: 3588-debian11-gnome-20240312\_update\lubancat-rk3588-debian11-gnome-20240312\_update.img

Firmware Ver: 1.0.00  
 Loader Ver: 1.0b  
 Chip: RK3588

ID	Fail	Device List	Device Type	ID	Upgrade Prompt	ID	Success
		My Computer					
		RootHub20					
		Port[1]					
		Port[2]					
		Port[3]					
		Port[4]					
		Port[5]					
		RootHub20					
		Port[1]	Loader	3-1			
		Port[2]					
		Port[3]					
		Port[4]					
		Port[5]					
		Port[6]					
		Port[7]					
		Port[8]					
		Port[9]					
		Port[10]					
		Port[11]					
		Port[12]					

Wizard:

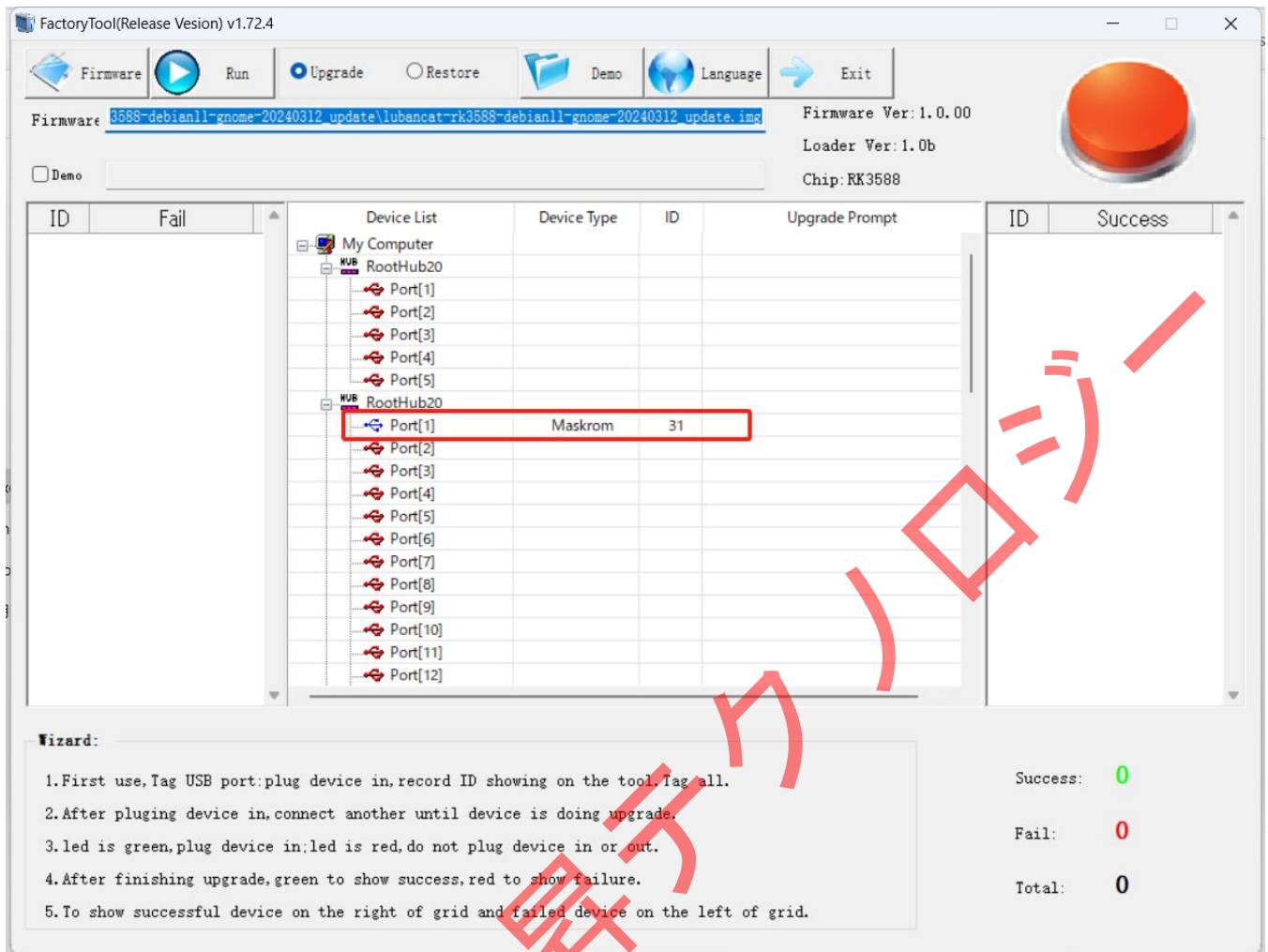
- 1.First use,Tag USB port:plug device in,record ID showing on the tool.Tag all.
- 2.After plugging device in,connect another until device is doing upgrade.
- 3.led is green,plug device in:led is red,do not plug device in or out.
- 4.After finishing upgrade,green to show success,red to show failure.
- 5.To show successful device on the right of grid and failed device on the left of grid.

Success: 0  
 Fail: 0  
 Total: 0

デバイスリストのポートと実際のダウンロードケーブルの対応関係をマークし、どのボードのダウンロードが完了し、どのボードが失敗したかを識別します。

具体的には、ボードを最初のUSBポートに接続し、ボードをMASKROMモードにします。ソフトウェアリストには対応するIDが表示されるので、そのIDを対応するインターフェースまたはダウンロードケーブルにマークし、次にポートを変更して上記のプロセスを繰り返します。





上の図を見るとボードを USB-HUB1 の Port3 に接続したところ、ID は 27 でした。

「スタート」ボタンをクリックして書き込み状態に入ります。

書き込みツールが書き込み状態に入ると、MASKROM モードに入ったボードが自動的に検出され、ダウンロードが開始されます。

FactoryTool(Release Vesion) v1.72.4

Firmware: 3588-debian11-gnome-20240312\_update\lubancat-rk3588-debian11-gnome-20240312\_update.img

Firmware Ver: 1.0.00  
Loader Ver: 1.0b  
Chip: RK3588

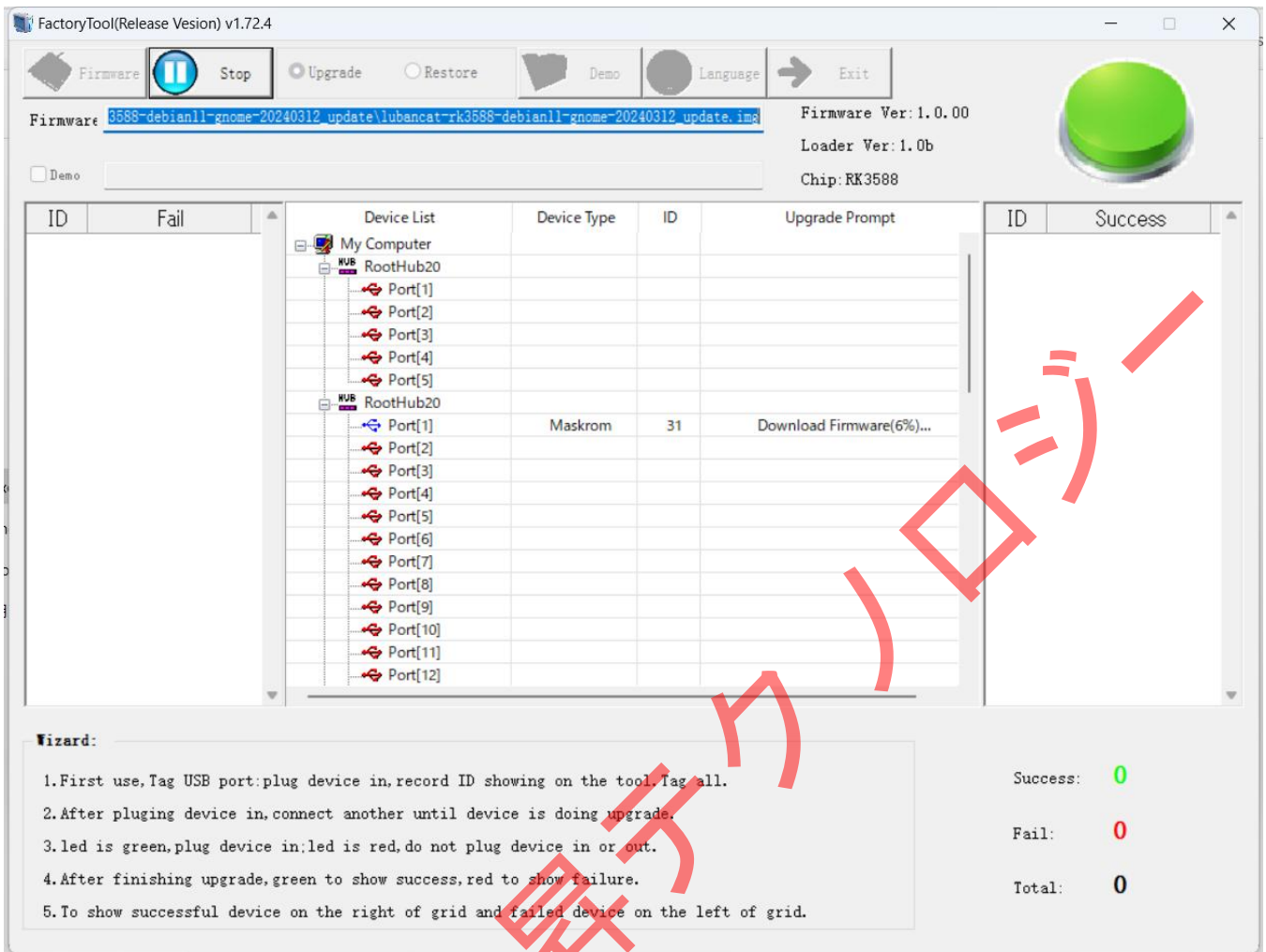
ID	Fail	Device List	Device Type	ID	Upgrade Prompt	ID	Success
		My Computer					
		RootHub20					
		Port[1]					
		Port[2]					
		Port[3]					
		Port[4]					
		Port[5]					
		RootHub20					
		Port[1]	Maskrom	31	Download Firmware(6%)...		
		Port[2]					
		Port[3]					
		Port[4]					
		Port[5]					
		Port[6]					
		Port[7]					
		Port[8]					
		Port[9]					
		Port[10]					
		Port[11]					
		Port[12]					

Wizard:

1. First use, Tag USB port: plug device in, record ID showing on the tool. Tag all.
2. After plugging device in, connect another until device is doing upgrade.
3. led is green, plug device in: led is red, do not plug device in or out.
4. After finishing upgrade, green to show success, red to show failure.
5. To show successful device on the right of grid and failed device on the left of grid.

Success: 0  
Fail: 0  
Total: 0

複数のボードを接続すると、同時に複数のボードへのダウンロードが行われます。



ID	Fail	Device List	Device Type	ID	Upgrade Prompt	ID	Success
		My Computer					
		RootHub20					
		Port[1]					
		Port[2]					
		Port[3]					
		Port[4]					
		Port[5]					
		RootHub20					
		Port[1]	Maskrom	31	Download Firmware(6%)...		
		Port[2]					
		Port[3]					
		Port[4]					
		Port[5]					
		Port[6]					
		Port[7]					
		Port[8]					
		Port[9]					
		Port[10]					
		Port[11]					
		Port[12]					

**Wizard:**

1. First use, Tag USB port: plug device in, record ID showing on the tool. Tag all.
2. After plugging device in, connect another until device is doing upgrade.
3. led is green, plug device in: led is red, do not plug device in or out.
4. After finishing upgrade, green to show success, red to show failure.
5. To show successful device on the right of grid and failed device on the left of grid.

Success: 0  
Fail: 0  
Total: 0

注意：SD カードを挿入せず、eMMC に起動イメージがない場合、ダウンロードキーを押さずに

MASKROM モードに入ることができます。

## 第 21 章 完全イメージの解体と再構築

完全なシステムイメージをコンパイルしてリリースし、イメージ管理を行うため、システムに必要な各区域分けてを一定のルールに従って一つにパッケージングしています。

バックアップや書き込みプロセスで個別の区域分けて操作が関わってくることや、一部のユーザーが SDK を使って自身で区域分けてを構築したくない場合があるため、ここでは SDK を使用せずにシステムイメージを個別に解体・再構築する方法について説明します。

注意：汎用イメージは Linux でのみ、Linux\_Pack\_Firmware ツールを使用してパッケージング・解体が可能です。特定のイメージは Windows と Linux の両方で正常にパッケージング・解体できます。

## 21.1 RKDevTool による解体と再構築 (Windows)

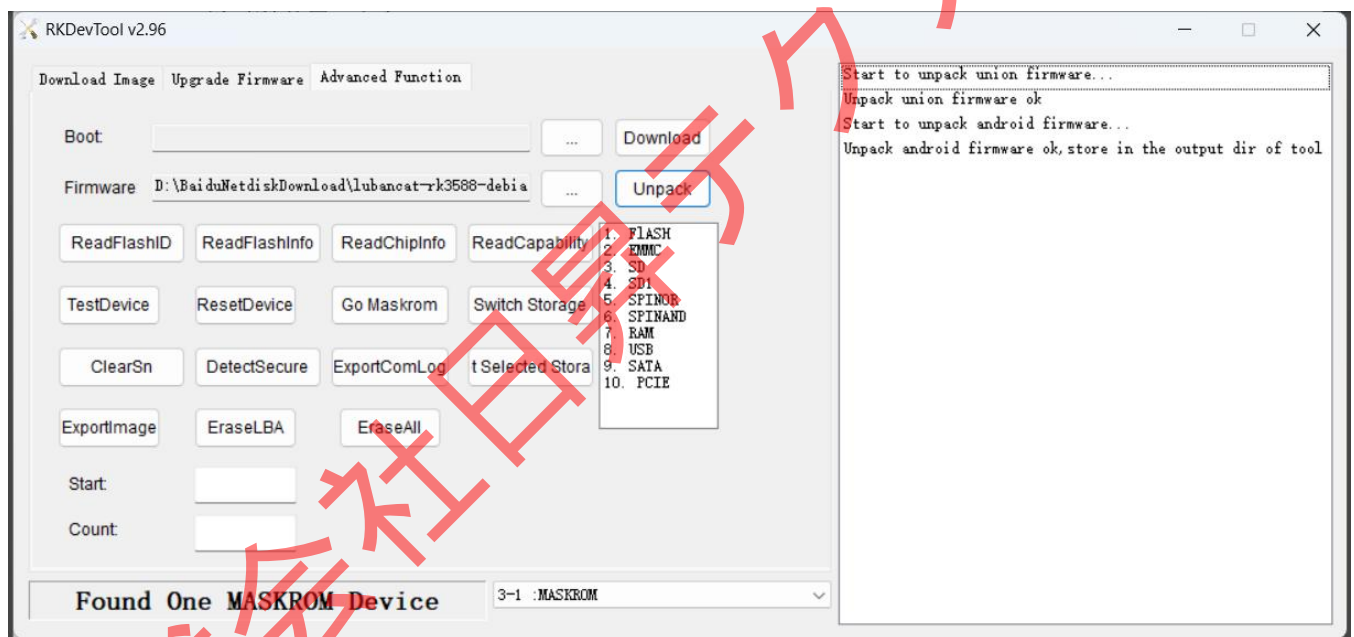
### 21.1.1 解体

完全な update.img イメージを区域分けてイメージの形式に解体するために RKDevTool を使用します。

RKDevTool のインストールは eMMC 書き込みの章を参照してください。

まず、ウェブから取得した Update.zip イメージを .img 形式に解凍するか、SDK で生成された update.img イメージを直接使用します。

RKDevTool を開き、高度な機能のタブに進んで、解体機能を見つけ、解体したいイメージを選択し、解体ボタンをクリックして解体を待ちます。



解体後のファイルは RKDevTool の Output ディレクトリに保存されます。

Output ディレクトリには 2つのファイルと 1つのフォルダがあります。

- boot.bin : パッケージング時の loader、つまり MiniLoaderAll.bin ファイルです。
- firmware.img : パッケージング時に afptool が生成したファームウェアです。
- Android フォルダは firmware.img の展開内容です。

Android フォルダに入ると、1つのファイルと 1つのフォルダがあります。

- package-file : 区域分けてと区域分けてイメージ名の対応関係です。
- Image : firmware.img が展開された内容、つまり解体後の区域分けてイメージです。

Image ディレクトリには、オペレーティングシステムに応じて以下の3種類のファイルが含まれています。

- 区域分けて表 : parameter.txt
- loader ファイル : MiniLoaderAll.bin
- 区域分けてイメージ : boot.img、misc.img など、img で終わる区域分けてイメージファイルです。

名前	更新日時	種類	サイズ
bootimg	2024/04/01 11:31	ディスク イメージファ...	131,072 KB
MiniLoaderAll.bin	2024/04/01 11:31	BIN ファイル	473 KB
package-file	2024/04/01 11:31	ファイル	1 KB
parameter.txt	2024/04/01 11:31	テキストドキュメント	1 KB
rootfs.img	2024/04/01 11:33	ディスク イメージファ...	4,219,904 ...
uboot.img	2024/04/01 11:31	ディスク イメージファ...	4,096 KB

## 21.1.2 再構築

通常、ルートファイルシステムを変更またはバックアップした後、変更された区域分けてイメージを完全なファームウェアに再構築する必要があります。

Windows 環境下の再構築ツールは SDK/tools/windows/RKDevTool/rockdev/ディレクトリにあります。

上記の再構築ツールは SDK 向けに設計されていますが、SDK に依存せずに使用できるように一部修正を加えました。修正後の再構築ツールは RKDevTool\_Release\_v2.92 ソフトウェアパッケージの rockdev ディレクトリに含まれており、RKDevTool の解体機能と共に使用できます。

名前	更新日時	種類	サイズ
bin	2023/12/11 11:45	ファイル フォルダ	
Language	2023/12/11 11:45	ファイル フォルダ	
Log	2024/04/01 10:43	ファイル フォルダ	
Output	2024/04/01 11:30	ファイル フォルダ	
config.cfg	2024/04/01 11:18	Configuration ソー...	2 KB
config.ini	2024/03/29 17:41	構成設定	2 KB
manual.pdf	2021/08/27 10:28	Adobe Acroba 文書	450 KB
revision.txt	2022/05/27 9:09	テキストドキュメント	3 KB
RKDevTool.exe	2022/05/27 9:06	アプリケーション	1,212 KB
rock-5b-emmc.cfg	2022/10/20 16:20	Configuration ソー...	2 KB
rock-5b-spinor.cfg	2022/10/20 19:01	Configuration ソー...	2 KB
rockdev	2024/04/01 12:04	ファイル フォルダ	

解体プロセスを通じて、Output フォルダの内容を取得しました。

まず、Output¥Android ディレクトリの内容を rockdev ディレクトリにコピーします。

PC > ローカルディスク (D:) > tool > RKDevTool\_Release\_v2.96 > rockdev

名前	更新日時	種類	サイズ
afptool	2022/09/29 11:09	ファイル	2,735 KB
AFPTool.exe	2009/09/14 11:18	アプリケーション	44 KB
mkcramfs.exe	2005/04/20 4:35	アプリケーション	20 KB
orion-mkupdate.sh	2022/09/29 11:09	Shell Script	1 KB
orion-package-file	2022/09/29 11:09	ファイル	1 KB
readme.txt	2022/09/29 11:09	テキストドキュメント	1 KB
revision.txt	2022/09/29 11:09	テキストドキュメント	1 KB
rk356x-mkupdate.sh	2022/09/29 11:09	Shell Script	1 KB
rk3328-mkupdate.sh	2022/09/29 11:09	Shell Script	1 KB
rk3399-mkupdate.sh	2022/09/29 11:09	Shell Script	1 KB
rk3588-mkupdate.sh	2022/09/29 11:09	Shell Script	1 KB
rklmageMaker	2022/09/29 11:09	ファイル	2,789 KB
sdcard-update-package-file	2022/09/29 11:09	ファイル	1 KB
su	2010/08/10 6:53	ファイル	26 KB
Superuser.apk	2010/11/15 19:50	APK ファイル	201 KB
unpack.sh	2022/09/29 11:09	Shell Script	1 KB

赤枠内の内容は、パッケージングツールとパッケージングを補助する実行ファイルです。Image ディレクトリには、区域分けてイメージが格納されており、package-file は区域分けてと区域分けてイメージファイルの対応関係です。

特定の区域分けてイメージファイルに変更を加えた場合、パッケージング時にイメージファイル名が正確であること、および package-file 内の命名とパスと一致していることに注意してください。

ボードのメインチップに対応する bat ファイルをダブルクリックして、パッケージングを開始します。

LubanCat-1、2、Zero はすべて rk358x メインチップを使用していますので、ここでは rk358x-mkupdate.bat をダブルクリックします。

```
Android Firmware Package Tool v1.65
F:\.LINUX\RockChip\RK3568\img\RKDevTool_Release_v2.92\rockdev>Afptool -pack ./ Image\update.img
Android Firmware Package Tool v1.65
----- PACKAGE -----
Add file: .\package-file
Add file: .\package-file done, offset=0x800, size=0x28b, userspace=0x1
Add file: .\Image\MiniLoaderAll.bin
Add file: .\Image\MiniLoaderAll.bin done, offset=0x1000, size=0x719c0, userspace=0xe4
Add file: .\Image\parameter.txt
Add file: .\Image\parameter.txt done, offset=0x73000, size=0x1f4, userspace=0x1
Add file: .\Image\uboot.img
Add file: .\Image\uboot.img done, offset=0x73800, size=0x400000, userspace=0x801
Add file: .\Image\misc.img
Add file: .\Image\misc.img done, offset=0x474000, size=0xc000, userspace=0x19
Add file: .\Image\boot.img
Add file: .\Image\boot.img done, offset=0x480800, size=0x1b9ea00, userspace=0x373e
Add file: .\Image\recovery.img
Add file: .\Image\recovery.img done, offset=0x201f800, size=0x2232e00, userspace=0x4466
Add file: .\Image\rootfs.img
Add file: .\Image\rootfs.img done, offset=0x4252800, size=0x51600000, userspace=0xa2c01
Add file: .\Image\oem.img
Add file: .\Image\oem.img done, offset=0x55853000, size=0x10a6000, userspace=0x214d
Add file: .\Image\userdata.img
Add file: .\Image\userdata.img done, offset=0x568f9800, size=0x444000, userspace=0x889
Add CRC...
Make firmware OK!
----- OK -----

F:\.LINUX\RockChip\RK3568\img\RKDevTool_Release_v2.92\rockdev>RKImageMaker.exe -RK3568 Image\MiniLoaderAll.bin
Image\update.img update.img -os_type:androidos
*****RKImageMaker ver 1.66 *****
Generating new image, please wait...
Writing head info...
Writing boot file...
Writing firmware...
Generating MD5 data...
MD5 data generated successfully!
New image generated successfully!

F:\.LINUX\RockChip\RK3568\img\RKDevTool_Release_v2.92\rockdev>rem update.img is new format, Image\update.img is
old format, so delete older format

F:\.LINUX\RockChip\RK3568\img\RKDevTool_Release_v2.92\rockdev>del Image\update.img

F:\.LINUX\RockChip\RK3568\img\RKDevTool_Release_v2.92\rockdev>pause
请按任意键继续. . .
```

パッケージングが完了すると、現在のフォルダ内に update.img ファイルが出現します。これがパッケージング後の完全なイメージファイルです。

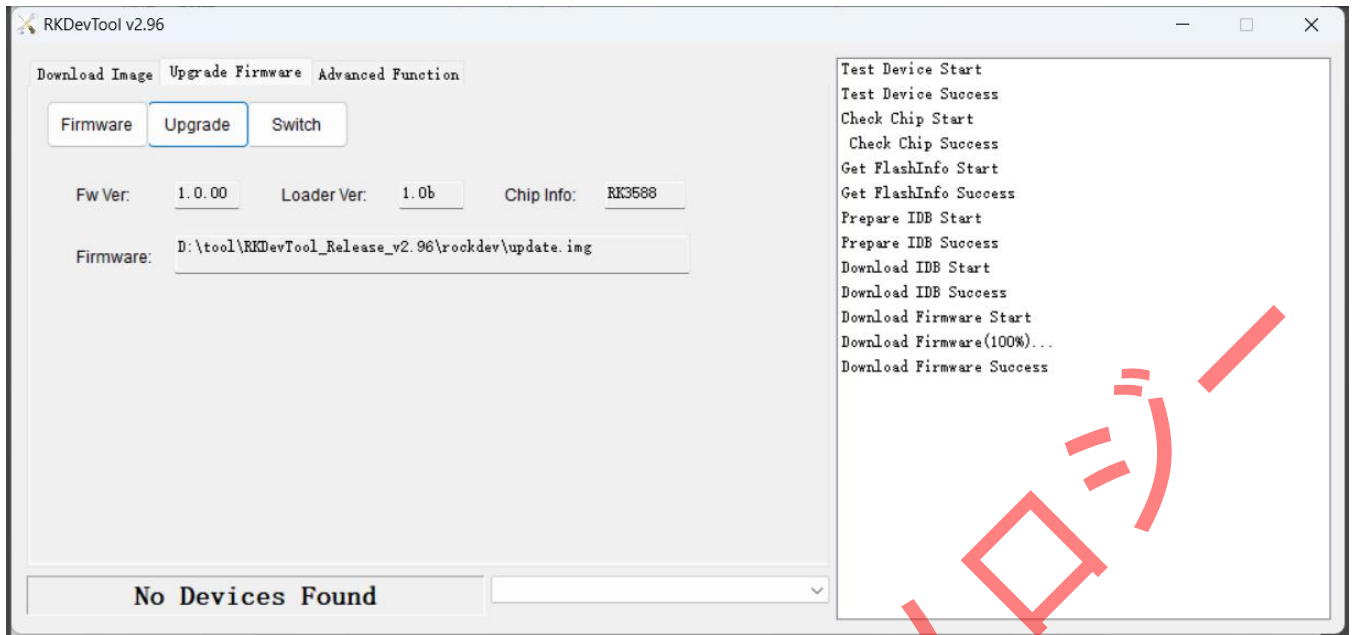


🔄 🖥️ > PC > ローカルディスク (D:) > tool > RKDevTool\_Release\_v2.96 > rockdev

✂️ 📄 🗑️ 📄 📄 🗑️ ⬆️ 並べ替え ▾ ≡ 表示 ▾ ⋮

名前	更新日時	種類	サイズ
afptool	2022/09/29 11:09	ファイル	2,735 KB
AFPTool.exe	2009/09/14 11:18	アプリケーション	44 KB
mkcramfs.exe	2005/04/20 4:35	アプリケーション	20 KB
orion-mkupdate.sh	2022/09/29 11:09	Shell Script	1 KB
orion-package-file	2022/09/29 11:09	ファイル	1 KB
readme.txt	2022/09/29 11:09	テキストドキュメント	1 KB
revision.txt	2022/09/29 11:09	テキストドキュメント	1 KB
rk356x-mkupdate.sh	2022/09/29 11:09	Shell Script	1 KB
rk3328-mkupdate.sh	2022/09/29 11:09	Shell Script	1 KB
rk3399-mkupdate.sh	2022/09/29 11:09	Shell Script	1 KB
rk3588-mkupdate.sh	2022/09/29 11:09	Shell Script	1 KB
rkImageMaker	2022/09/29 11:09	ファイル	2,789 KB
sdcard-update-package-file	2022/09/29 11:09	ファイル	1 KB
su	2010/08/10 6:53	ファイル	26 KB
Superuser.apk	2010/11/15 19:50	APK ファイル	201 KB
unpack.sh	2022/09/29 11:09	Shell Script	1 KB
<b>update.img</b>	2024/03/12 14:39	ディスク イメージ ファ...	4,356,025 ...

このパッケージングされたイメージを書き込みツールを使用してボードに書き込むと、正常に起動することができます。



株式会社日昇テクノロジー

```
U-Boot 2017.09-gf403aeed3-220621 #jiawen (Sep 16 2022 - 15:40:12 +0800)

ADC Channel:2 reading_value: 796, CH_ID: 0x05
ADC Channel:3 reading_value: 186, CH_ID: 0x00
Board Version: Board_ID 0x0500
Board: LubanCat2N
PreSerial: 2, raw, 0xfe660000
DRAM: 4 GiB
System: init
Relocation Offset: ed33a000
Relocation fdt: eb9f9080 - eb9fecd8
CR: M/C/I
Using default environment

dwmmc@fe2b0000: 1, dwmmc@fe2c0000: 2, sdhci@fe310000: 0
Bootdev(atags): mmc 0
MMC0: HS200, 200Mhz
PartType: EFI
DM: v1
boot mode: recovery (misc)
FIT: no signed, no conf required
** Unrecognized filesystem type **
DTB: rk-kernel.dtb
HASH(c): OK
I2c0 speed: 100000Hz
vsel-gpios- not found! Error: -2
vdd_cpu 1025000 uV
PMIC: RK8090 (on=0x40, off=0x00)
vdd_logic init 900000 uV
vdd_gpu init 900000 uV
vdd_npu init 900000 uV
io-domain: OK
Could not find baseparameter partition
Board Version: Board_ID 0x0500
Board: LubanCat2N
Rockchip UB00T DRM driver version: v1.0.1
VOP have 1 active VP
vp0 have layer nr:6[0 2 4 1 3 5 ], primary plane: 4
vp1 have layer nr:0[], primary plane: 0
vp2 have layer nr:0[], primary plane: 0
xfer: num: 2, addr: 0x50
xfer: num: 2, addr: 0x50
Monitor has basic audio support
can't find to match
Could not find baseparameter partition
mode:1920x1080
hdmi@fe0a0000: detailed mode clock 148500 kHz, flags[5]
  H: 1920 2008 2052 2200
  V: 1080 1084 1089 1125
bus_format: 2025
VOP update mode to: 1920x1080p0, type: HDMI0 for VP0
rockchip_vop2_init: Failed to get hdmi0_phy_pll ret=-22
rockchip_vop2_init: Failed to get hdmi1_phy_pll ret=-22
VOP VP0 enable Smart0[772x525->772x525@574x277] fmt[1] addr[0xee029000]
CEA mode used vic=17
final pixclk = 148000000 tmdsclk = 148000000
PHY powered down in 0 iterations
```

## 21.2 Linux\_Pack\_Firmware による解体とパッケージング (Linux)

### 21.2.1 解体

完全な update.img イメージを区域分けてイメージの形式に解体するために Linux\_Pack\_Firmware を使用することもできます。Linux\_Pack\_Firmware は、提供するバージョンを使用して、Linux PC や仮想マシンに解凍してください。取得した Update.zip イメージを .img 形式に解凍し、update.img としてリネームするか、SDK で生成された update.img イメージを直接使用します。

update.img を Linux\_Pack\_Firmware/rockdev ディレクトリにコピーします。

```
hh@ubuntu:~$ ls
Desktop          Pictures
Documents       Public
Downloads       RK3568-LUBANCAT2-N_UBUNTU20.04_DESKTOP_20220921_Update.img
examples.desktop Templates
Linux_Pack_Firmware.zip Videos
Music

hh@ubuntu:~$ unzip Linux_Pack_Firmware.zip
Archive: Linux_Pack_Firmware.zip
  creating: Linux_Pack_Firmware/
  creating: Linux_Pack_Firmware/rockdev/
  inflating: Linux_Pack_Firmware/rockdev/afptool
  inflating: Linux_Pack_Firmware/rockdev/orion-mkupdate.sh
  inflating: Linux_Pack_Firmware/rockdev/orion-package-file
  inflating: Linux_Pack_Firmware/rockdev/readme.txt
  inflating: Linux_Pack_Firmware/rockdev/revision.txt
  inflating: Linux_Pack_Firmware/rockdev/rk3328-mkupdate.sh
  inflating: Linux_Pack_Firmware/rockdev/rk3399-mkupdate.sh
  inflating: Linux_Pack_Firmware/rockdev/rk356x-mkupdate.sh
  inflating: Linux_Pack_Firmware/rockdev/rk3588-mkupdate.sh
  inflating: Linux_Pack_Firmware/rockdev/rkImageMaker
  inflating: Linux_Pack_Firmware/rockdev/sdcard-update-package-file
  inflating: Linux_Pack_Firmware/rockdev/unpack.sh
hh@ubuntu:~$ mv RK3568-LUBANCAT2-N_UBUNTU20.04_DESKTOP_20220921_Update.img Linux_Pack_Firmware/rockdev/update.img
hh@ubuntu:~$
```

Linux\_Pack\_Firmware 内のファイルを見てみましょう。

```
hh@ubuntu:~$ cd Linux_Pack_Firmware/
hh@ubuntu:~/Linux_Pack_Firmware$ tree
.
├── rockdev
│   ├── afptool
│   ├── orion-mkupdate.sh
│   ├── orion-package-file
│   ├── readme.txt
│   ├── revision.txt
│   ├── rk3328-mkupdate.sh
│   ├── rk3399-mkupdate.sh
│   ├── rk356x-mkupdate.sh
│   ├── rk3588-mkupdate.sh
│   ├── rkImageMaker
│   ├── sdcard-update-package-file
│   ├── unpack.sh
│   └── update.img
└── 1 directory, 13 files
hh@ubuntu:~/Linux_Pack_Firmware$
```

- afptool、rkImageMaker：バイナリ実行ファイルです。

- rk3xxx-mkupdate.sh : イメージパッケージングスクリプトです。
- unpack.sh : イメージ解体スクリプトです。
- その他 : サンプル、説明ファイルなどです。

unpack.sh スクリプトを実行して解体します。

#### 1 unpack.sh

```
hh@ubuntu:~/Linux_Pack_Firmware$ cd rockdev/  
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$ ./unpack.sh  
start to unpack update.img...  
*****rkImageMaker ver 2.0*****  
Unpacking image, please wait...  
Exporting boot.bin  
Exporting firmware.img  
Unpacking image success.  
Android Firmware Package Tool v2.0  
Check file... OK  
----- UNPACK -----  
package-file    offset=0x800    size=0xA0  
Image/MiniLoaderAll.bin offset=0x1000    size=0x719C0  
Image/parameter.txt    offset=0x73000    size=0x15D  
Image/uboot.img offset=0x73800    size=0x400000  
Image/boot.img offset=0x473800    size=0x8000000  
Image/rootfs.img    offset=0x8473800    size=0xB9159000  
Unpack firmware OK!  
----- OK -----  
Unpacking update.img OK.  
Press any key to quit:  
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$ [A
```

解体後のファイルは output ディレクトリに保存され、ファイルの内容は RKDevTool の解体後の内容と同じです。

```
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$ cd output/  
hh@ubuntu:~/Linux_Pack_Firmware/rockdev/output$ tree  
.  
├── Image  
│   ├── boot.img  
│   ├── MiniLoaderAll.bin  
│   ├── parameter.txt  
│   ├── rootfs.img  
│   └── uboot.img  
└── package-file  
  
1 directory, 6 files  
hh@ubuntu:~/Linux_Pack_Firmware/rockdev/output$
```

output ディレクトリには 1 つのファイルと 1 つのフォルダがあります。

- package-file : 区域分けてと区域分けてイメージ名の対応関係です。
- Image : firmware.img の展開後の内容、つまり解体後の区域分けてイメージです。

Image ディレクトリに入ると、異なるオペレーティングシステムイメージに応じて以下の 3 種類のファイルがあります。

- 区域分けて表 : parameter.txt
- loader ファイル : MiniLoaderAll.bin
- 区域分けてイメージ : boot.img、uboot.img など、img で終わる区域分けてイメージファイルです。

## 21.2.2 パッケージング

上述の解体プロセスを通じて、output フォルダの内容を取得しました。

まず、rockdev/output ディレクトリ下の内容を rockdev ディレクトリにコピーし、以前解体用に使用した update.img イメージを削除します。

```

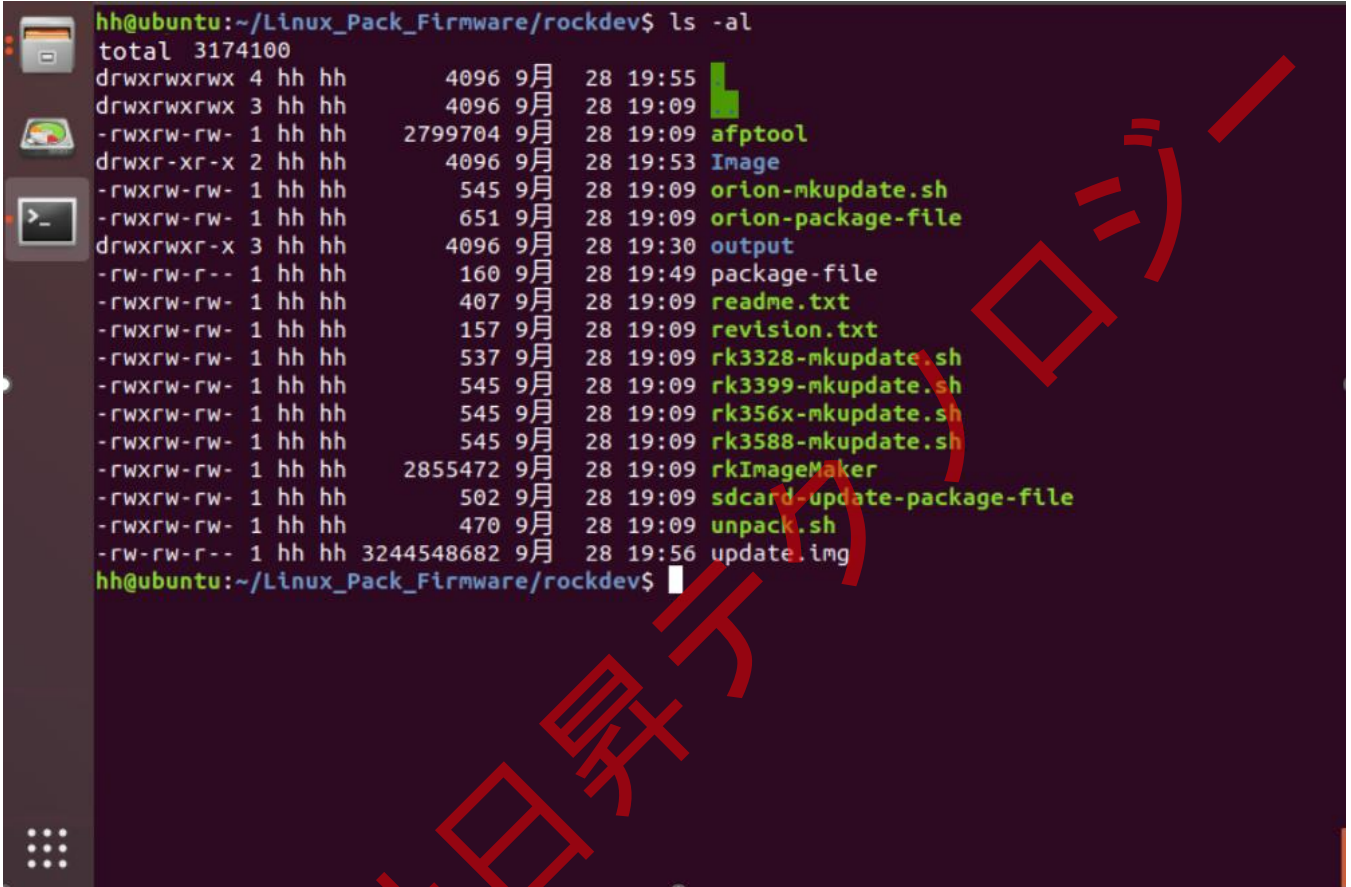
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$ ls
afptool          readme.txt      rk356x-mkupdate.sh      unpack.sh
orion-mkupdate.sh revision.txt    rk3588-mkupdate.sh      update.img
orion-package-file rk3328-mkupdate.sh rkImageMaker
output          rk3399-mkupdate.sh sdcard-update-package-file
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$ ls output/
Image package-file
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$ cp -r output/* ./
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$ ls
afptool          output          rk3328-mkupdate.sh  rkImageMaker
Image            package-file    rk3399-mkupdate.sh  sdcard-update-package-file
orion-mkupdate.sh readme.txt      rk356x-mkupdate.sh  unpack.sh
orion-package-file revision.txt    rk3588-mkupdate.sh  update.img
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$ rm update.img
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$ ls
afptool          output          rk3328-mkupdate.sh  rkImageMaker
Image            package-file    rk3399-mkupdate.sh  sdcard-update-package-file
orion-mkupdate.sh readme.txt      rk356x-mkupdate.sh  unpack.sh
orion-package-file revision.txt    rk3588-mkupdate.sh
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$
  
```

特定の区域分けてイメージファイルに変更を加えた場合、パッケージング時にイメージファイル名が正確であること、そして package-file ファイル内の命名とパスと一致していることに注意してください。

```

hh@ubuntu:~/Linux_Pack_Firmware/rockdev$ ./rk356x-mkupdate.sh
start to make update.img...
Android Firmware Package Tool v2.0
----- PACKAGE -----
Add file: ./package-file
package-file,Add file: ./package-file done,offset=0x800,size=0xa0,userspace=0x1
Add file: ./Image/MiniLoaderAll.bin
bootloader,Add file: ./Image/MiniLoaderAll.bin done,offset=0x1000,size=0x719c0,userspace
=0xe4
Add file: ./Image/parameter.txt
parameter,Add file: ./Image/parameter.txt done,offset=0x73000,size=0x169,userspace=0x1
Add file: ./Image/uboot.img
uboot,Add file: ./Image/uboot.img done,offset=0x73800,size=0x400000,userspace=0x800
Add file: ./Image/boot.img
boot,Add file: ./Image/boot.img done,offset=0x473800,size=0x8000000,userspace=0x10000
Add file: ./Image/rootfs.img
rootfs,Add file: ./Image/rootfs.img done,offset=0x8473800,size=0xb9159000,userspace=0x17
22b2
Add CRC...
Make firmware OK!
----- OK -----
*****rkImageMaker ver 2.0*****
Generating new image, please wait...
Writing head info...
Writing boot file...
Writing firmware...
Generating MD5 data...
MD5 data generated successfully!
New image generated successfully!
Making ./Image/update.img OK.
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$
  
```

パッケージングが完了すると、現在のフォルダ内に update.img ファイルが出現します。これはパッケージング後の完全なイメージファイルです。



```
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$ ls -al
total 3174100
drwxrwxrwx 4 hh hh      4096 9月  28 19:55 |
drwxrwxrwx 3 hh hh      4096 9月  28 19:09 |
-rwxr--r-- 1 hh hh    2799704 9月  28 19:09 afptool
drwxr-xr-x 2 hh hh      4096 9月  28 19:53 Image
-rwxr--r-- 1 hh hh      545 9月  28 19:09 orion-mkupdate.sh
-rwxr--r-- 1 hh hh      651 9月  28 19:09 orion-package-file
drwxrwxr-x 3 hh hh      4096 9月  28 19:30 output
-rw-r--r-- 1 hh hh      160 9月  28 19:49 package-file
-rwxr--r-- 1 hh hh      407 9月  28 19:09 readme.txt
-rwxr--r-- 1 hh hh      157 9月  28 19:09 revision.txt
-rwxr--r-- 1 hh hh      537 9月  28 19:09 rk3328-mkupdate.sh
-rwxr--r-- 1 hh hh      545 9月  28 19:09 rk3399-mkupdate.sh
-rwxr--r-- 1 hh hh      545 9月  28 19:09 rk356x-mkupdate.sh
-rwxr--r-- 1 hh hh      545 9月  28 19:09 rk3588-mkupdate.sh
-rwxr--r-- 1 hh hh    2855472 9月  28 19:09 rkImageMaker
-rwxr--r-- 1 hh hh      502 9月  28 19:09 sdcard-update-package-file
-rwxr--r-- 1 hh hh      470 9月  28 19:09 unpack.sh
-rw-r--r-- 1 hh hh    3244548682 9月  28 19:56 update.img
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$
```

このパッケージングされたイメージを書き込みツールを使用してボードに書き込むと、正常に起動することができます。



```
Starting Network Manager Script Dispatcher Service...
[ OK ] Started Network Manager Script Dispatcher Service.
Starting Time & Date Service...
[ OK ] Started Time & Date Service.
[ OK ] Started Bluetooth management mechanism.

Ubuntu 20.04.4 LTS GNU/Linux lubancat ttyFIQ0
[username:password] root:root cat:temppwd
Modify information : /etc/issue

lubancat login: [ 11.359201] rk-pcie 3c0000000.pcie: PCIe Link Fail
[ 11.359296] rk-pcie 3c0000000.pcie: failed to initialize host

lubancat login: root
Password:

Lubancat

Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 4.19.232 aarch64)

* Documentation: http://doc.embedfire.com
* Management: http://www.embedfire.com

System information as of Wed Aug 31 23:27:53 CST 2022
System load: 2.37 0.56 0.19 Up time: 0 min
Memory usage: 7 % of 3900MB IP: 192.168.103.119
CPU temp: 57°C GPU temp: 57°C
Usage of /: 9% of 29G

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@lubancat:~#
```

## 第22章 システムイメージのバックアップと再書き込み

### き込み

開発から生産過程において、システムイメージのバックアップと再書き込みは必須のプロセスです。ここでは、異なるバックアップ方法とバックアップしたイメージを書き込む方法を紹介します。

以下のバックアップ方法で得られるバックアップイメージは RAW 形式のイメージであり、USB 量産ツールで書き込むことはできませんが、Win32DiskImager、Etcher などの RAW 形式をサポートするソフトウェアや、dd コマンドを使用して書き込むことができます。

注意：RAW 形式とは、加工されていない形式を指す一般的な意味ですが、の関連文書では、イメージが完全な区域分けて情報と区域分けてファイルを含んでおり、イメージ全体が分割不可能な一体とされてい

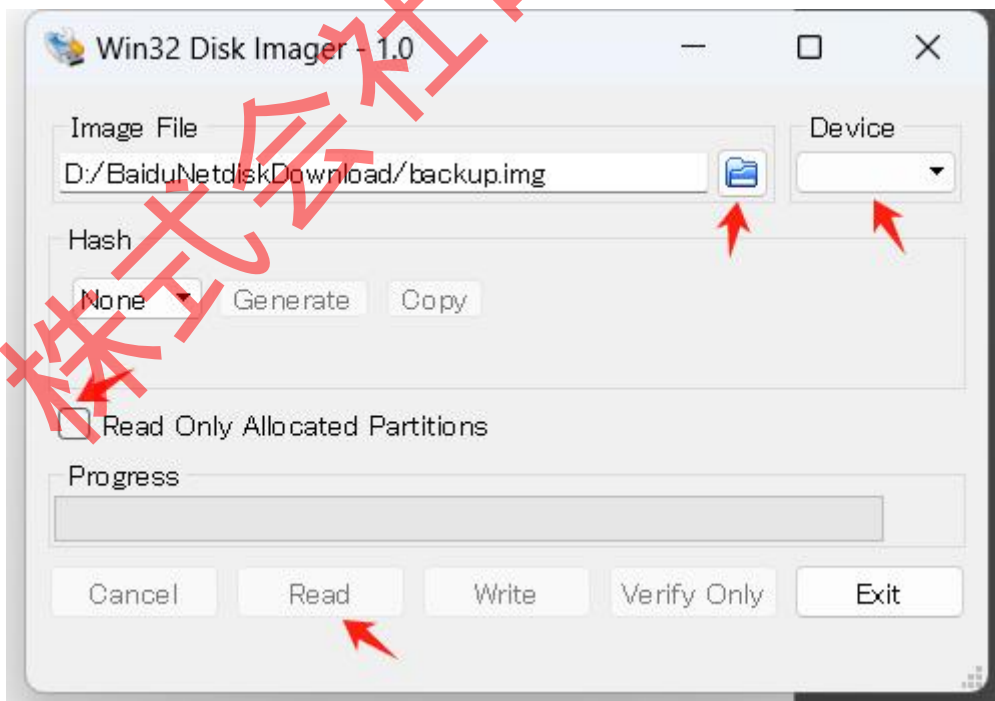
ることを指します。これは、区域分けてマークを完全なイメージにパッケージングし、書き込み時に区域分けて表に従って対応する区域分けてを適切なアドレスに書き込む RK 形式と区別されます。

## 22.1 Win32DiskImager で SD カードシステムイメージの全体バックアップと再書き込み

Win32DiskImager を使用した SD カードの全体バックアップは、最も簡単なバックアップ方法の一つですが、SD カード全体をバックアップするため、バックアップファイルと SD カードの容量が一致し、復元時にはイメージよりも大きな容量の SD カードを使用する必要があります。

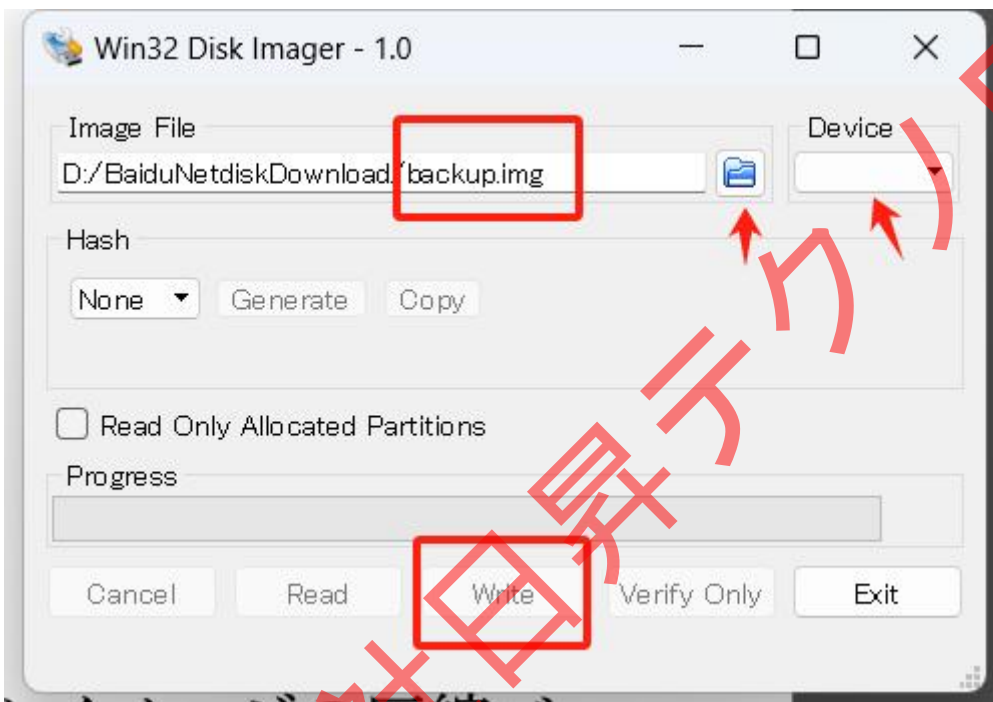
Win32 ディスクイメージャーツールのダウンロードリンク: <https://win32diskimager.org/>

Windows で空の img ファイルを新規作成し、例えば backup.img とします。バックアップしたい SD カードを Windows に挿入し、Win32 ディスクイメージャーツールを開きます。次に、ファイルフォルダアイコンをクリックし、作成した backup.img ファイルを選択し、SD カードのドライブレターを確認し、「割り当てられたパーティションのみを読み取る」のチェックを外し、最後に「読み取り」ボタンをクリックします。backup.img を上書きするかどうかのダイアログが表示されたら、「はい」をクリックし、イメージのバックアップを待ちます。



この時点で、イメージは RAW 形式の backup.img としてバックアップされています。

Win32 ディスクイメージャーツール以外にも、Etcher などの RAW 形式イメージを書き込むソフトウェアを使用して、backup.img イメージを SD カードに書き込むことができます。Win32 ディスクイメージャーツールを開き、backup.img イメージを選択し、新しい空の SD カードのドライブレターを選択し、「書き込み」ボタンをクリックすれば、注意点として、書き込まれる SD カードのサイズはイメージのサイズ以上である必要があります。



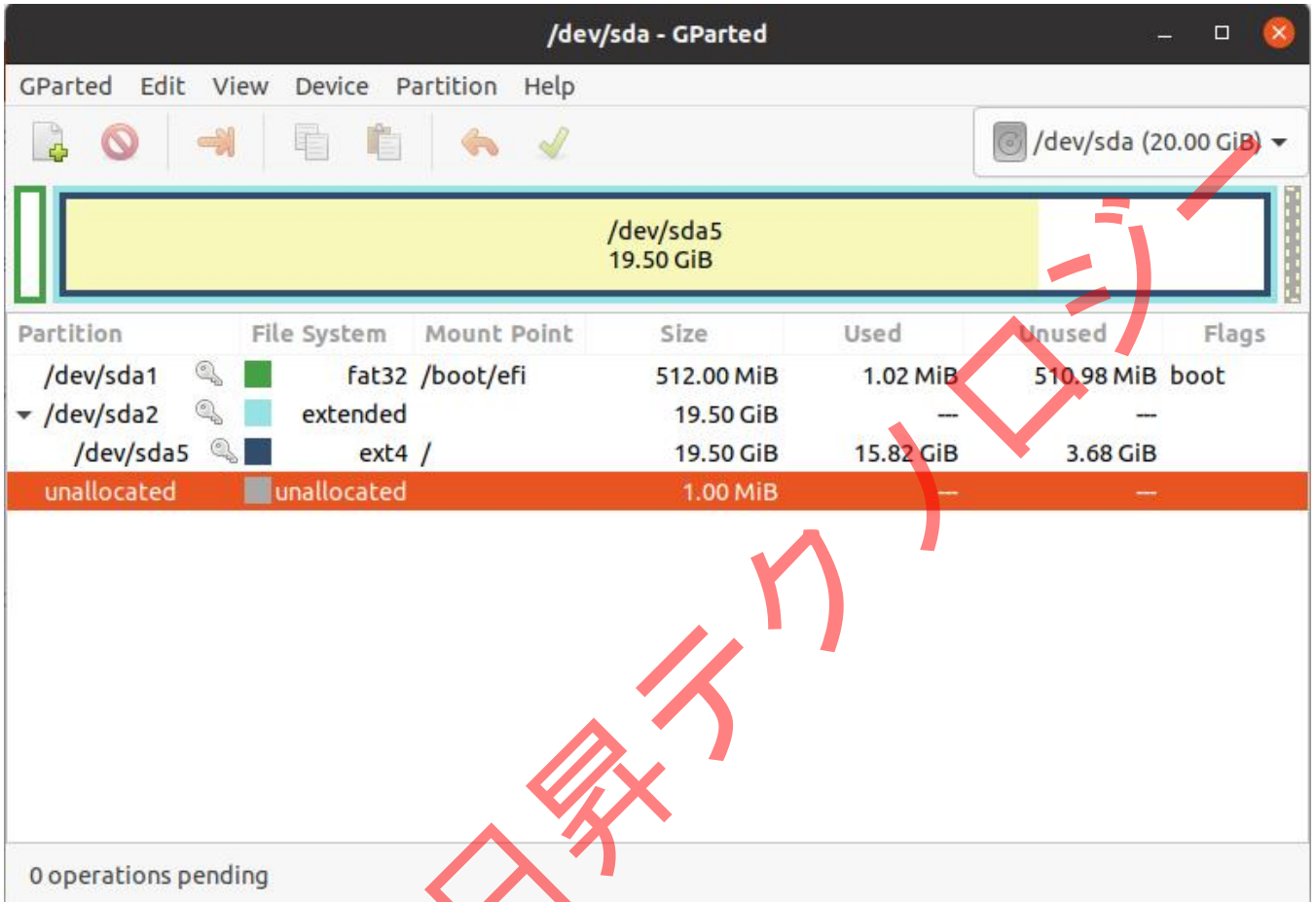
## 22.2 dd コマンドで SD カードシステムイメージの圧縮バックアップと再書き込み

dd コマンドを使用して SD カードを圧縮バックアップする場合は、Linux システムを実行しているホストマシンまたは仮想マシンが必要ですが、バックアップされたイメージのサイズは小さく、再配布や書き込みに便利です。

環境が構築された SD カードをボードから取り出し、カードリーダーに挿入してから、カードリーダーを PC (Ubuntu で) に接続します。Gparted ディスク管理ツールを開き、ディスクを SD カードに切り替えると、以下の情報が表示されます (例として Debian システムが書き込まれた SD カード)。

Gparted がインストールされていない場合は、以下のコマンドでインストールできます。

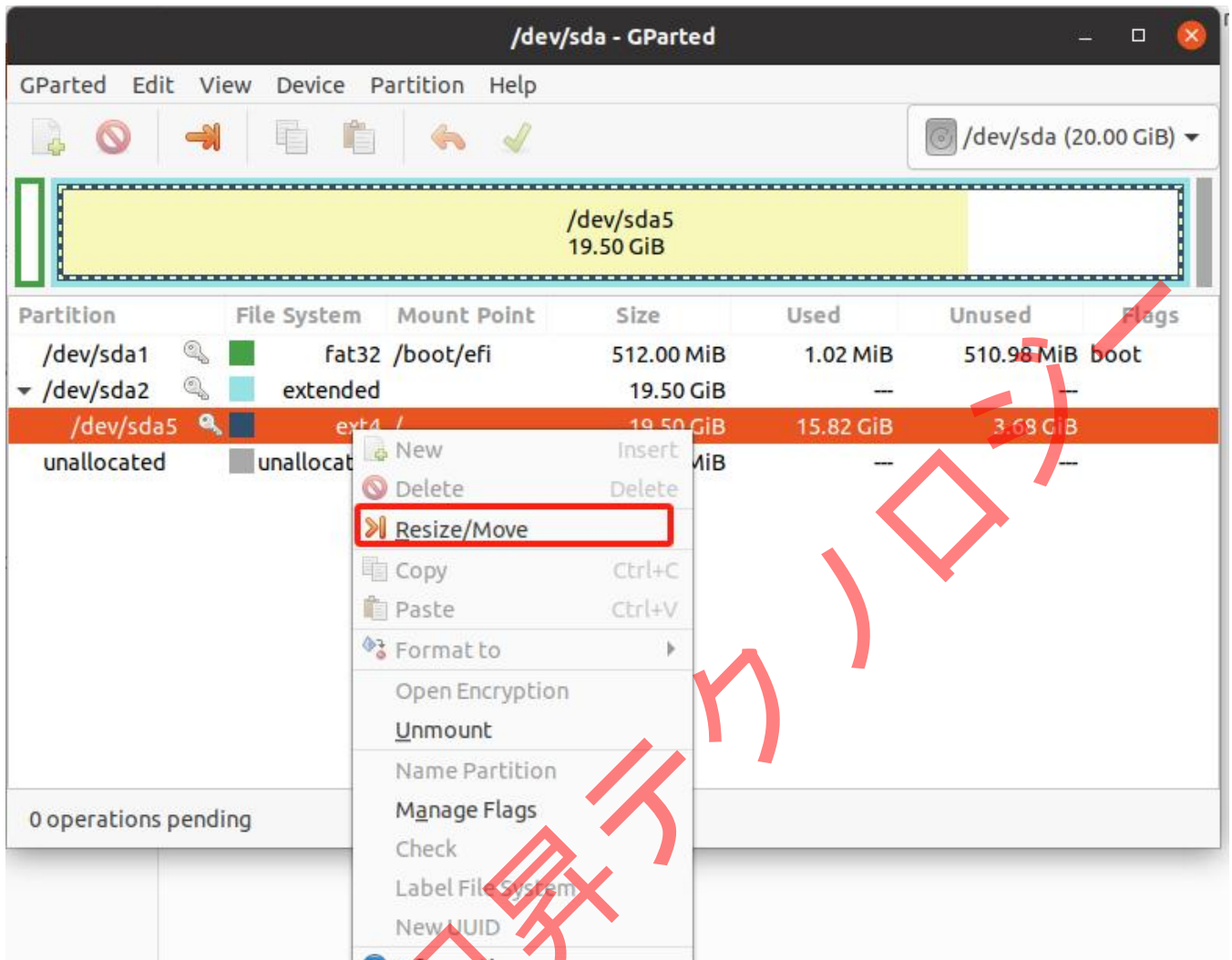
```
1 sudo apt install gparted
```



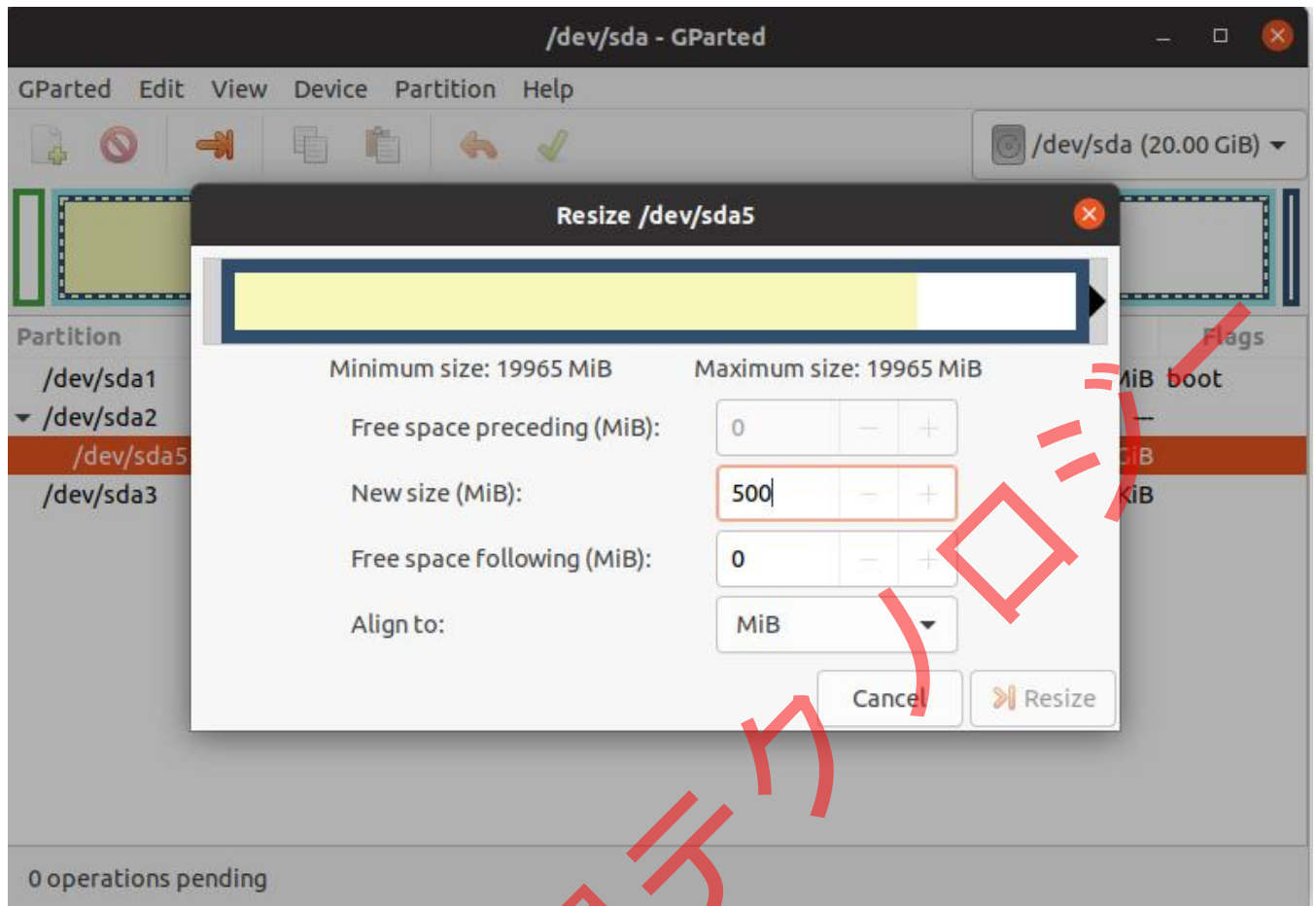
SD カードには合計8つの割り当てられたパーティションがあり、バックアップする内容は最後のパーティションとそれ以前の内容です。

最後のパーティションのサイズは 23.43GiB ですが、使用されているスペースは 450KiB のみです。最後のパーティションのサイズを圧縮することで、バックアップイメージのサイズを減らすことができます。

サイズを調整するパーティションを選択し、右クリックして「サイズ変更/移動」を選択します。



このパーティションの最小サイズは 481M ですが、500M に圧縮し、サイズ変更/移動をクリックします。



変更を適用するには、緑のチェックマークをクリックして変更を開始します。

書き込み時に最後のパーティションが自動的に拡張されるようにするには、以下のコマンドを実行します。

rootfs パーティションをマウントします。上記の図に基づくと、これは/dev/sdb6です。その後、中のファイルの一つを削除します。

```
1 # rootfs パーティションをマウント
2 sudo mount /dev/sdb6 /mnt
3
4 # このファイルを削除すると、バックアップイメージを使用して起動する際にパーティションが自動的に拡張されます
5 sudo rm /mnt/var/lib/misc/firstrun
```

6

7 # 削除が完了したら、マウントされたパーティションをアンマウント

8 sudo umount /mnt

バックアップするサイズを計算します (8+4+4+64+64+32+6.00x1024+128+500) MiB≈6948MiB。表示されるサイズは四捨五入により 2 桁の小数で保持されるため、少し余裕を持って 7000MiB のサイズをバックアップします。

Linux では、1MiB=1024KiB=1048576Byte、1MB=1000KB=1000000Byte です。

新しいディレクトリを作成して、イメージの SD カードからイメージをコピーするための場所を用意します。次に、dd コマンドを使用して SD カードからイメージを新しく作成したディレクトリにコピーします。

1 # 新しいディレクトリを作成

2 mkdir backup

3

4 # SD カードからイメージを新しいディレクトリにコピー

5 sudo dd if=/dev/sdb of=./backup/backup.img count=7000 bs=1024k conv=sync

6

7 # コピーには時間がかかりますので、お待ちください。以下のようなメッセージが表示されます。

8 7000+0 レコードイン

9 7000+0 レコードアウト

10 7340032000 バイト (7.3 GB, 6.8 GiB) が 492.74 秒でコピーされました、14.9 MB/s

ヒント: バックアップしたイメージを書き込んでも正常に動作しない場合は、bs=1024k を bs=1M に変

更し、conv パラメータを削除してください。つまり

```
sudo dd if=/dev/sdb of=./backup/backup.img count=7000 bs=1M
```

dd コマンドが完了すると、RAW 形式の backup.img イメージが作成されます。

dd コマンドのパラメータの意味：

- if=ファイル名：入力ファイル名。デフォルトは標準入力です。ここではソースファイルを指定します。< if=/dev/sdb >
- of=ファイル名：出力ファイル名。デフォルトは標準出力です。ここでは出力先ファイルを指定します。< of=./backup/backup.img >。ここでの.img はイメージの形式で、後で Etcher などでイメージを書き込むのに便利です。
- bs=バイト：読み込み/書き込みを行うブロックサイズをバイトで設定します。ここでは 1024k、つまり 1M のサイズを指定しています。
- count=ブロック数：指定したブロック数のみをコピーします。ブロックのサイズは ibs で指定されたバイト数です。ここでは 7000 を指定しており、7000M を意味します。
- conv=sync：各入力ブロックを ibs バイトにパディングし、不足分を NUL 文字で埋めます。

RAW 形式の backup.img イメージは、Win32DiskImager や Etcher などのソフトウェアを使って SD カードに書き込むことができますし、空の SD カードに dd コマンドを使って書き込むこともできます。

空の SD カードを PC に挿入し、挿入した SD カードのデバイス番号を確認します。ここではデモンストレーションとしてデバイス番号は /dev/sdc ですが、実際に SD カードに対応するデバイス番号に応じて柔軟に変更してください。

```
1 # backup.img を SD カードに書き込む
2 sudo dd if=./backup/backup.img of=/dev/sdc bs=1024k conv=sync
3
4 # 書き込み完了後、以下のようなメッセージが表示されます
5 7000+0 レコードイン
```



6 7000+0 レコードアウト

7 7340032000 バイト (7.3 GB, 6.8 GiB) が 427.027 秒でコピーされました、17.2 MB/s

count を指定しない場合は、backup.img ファイル全体がターゲットに書き込まれます。

書き込みが完了したら、書き込まれた SD カードを使用してボードを起動できます。

## 22.3 dd コマンドで eMMC のシステムイメージの圧縮バックアップ

dd コマンドを使って eMMC 内のシステムイメージをバックアップするプロセスと原理は、SD カードをバックアップする場合と似ています。どちらも Linux 環境下で、バックアップ対象のストレージデバイスから内容を読み取り、コピーします。

SD カードと eMMC のバックアップの違いは、SD カードが取り外し可能なストレージデバイスである一方で、eMMC はボードに固定されており取り外せない点です。SD カードを使用してシステムを起動する場合、eMMC のバックアップ用に Linux 環境を提供することができます。

注意：SD カードのバックアッププロセスでは rkboot パーティションの Debian イメージを例としていますが、eMMC のバックアッププロセスでは extboot パーティションの Ubuntu イメージを例にします。二つのシステムイメージはパーティションの形式と数量が若干異なりますが、バックアップ方法とプロセスには影響しません。

バックアップ用の SD カードを用意し、RK358X-LUBANCATBACKUP\_xxx\_Update.img (xxx は更新日付、最新バージョンを使用) を燃焼させます。このボード用の一般バックアップイメージは、ウェブからダウンロードでき、LubanCat-RK358x のボードに適用されます。

バックアップ後のイメージを保存するためのストレージデバイスも必要です。U ディスクが望ましいですが、高速な読み書き速度でバックアップ速度を向上させることができます。U ディスクがない場合は、ボードを起動する SD カードを使用し、バックアップイメージを保存するためのパーティションを別途作成することも可能です。

eMMC から起動した後、バックアップした内容が正確に保存されているかを検証するためのフラグファイ

ルを作成します。

```
Ubuntu 20.04.4 LTS GNU/Linux lubancat ttyFIQ0
[username:password] root:root cat:tempwd
Modify information : /etc/issue
lubancat login: root
Password:

Lubancat

Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 4.19.232 aarch64)

* Documentation: http://doc.embedfire.com
* Management: http://www.embedfire.com

System information as of Wed Sep 28 11:34:02 CST 2022

System load:  2.96 1.28 0.48   Up time:      1 min
Memory usage: 8 % of 3900MB   IP:           192.168.103.123
CPU temp:     57°C           GPU temp:     57°C
Usage of /:   9% of 29G

Last login: Wed Aug 31 23:28:26 CST 2022 on ttyFIQ0
root@lubancat:~# date
Wed Sep 28 11:34:07 CST 2022
root@lubancat:~# date > backup.txt
root@lubancat:~# cat backup.txt
Wed Sep 28 11:34:16 CST 2022
root@lubancat:~#
```

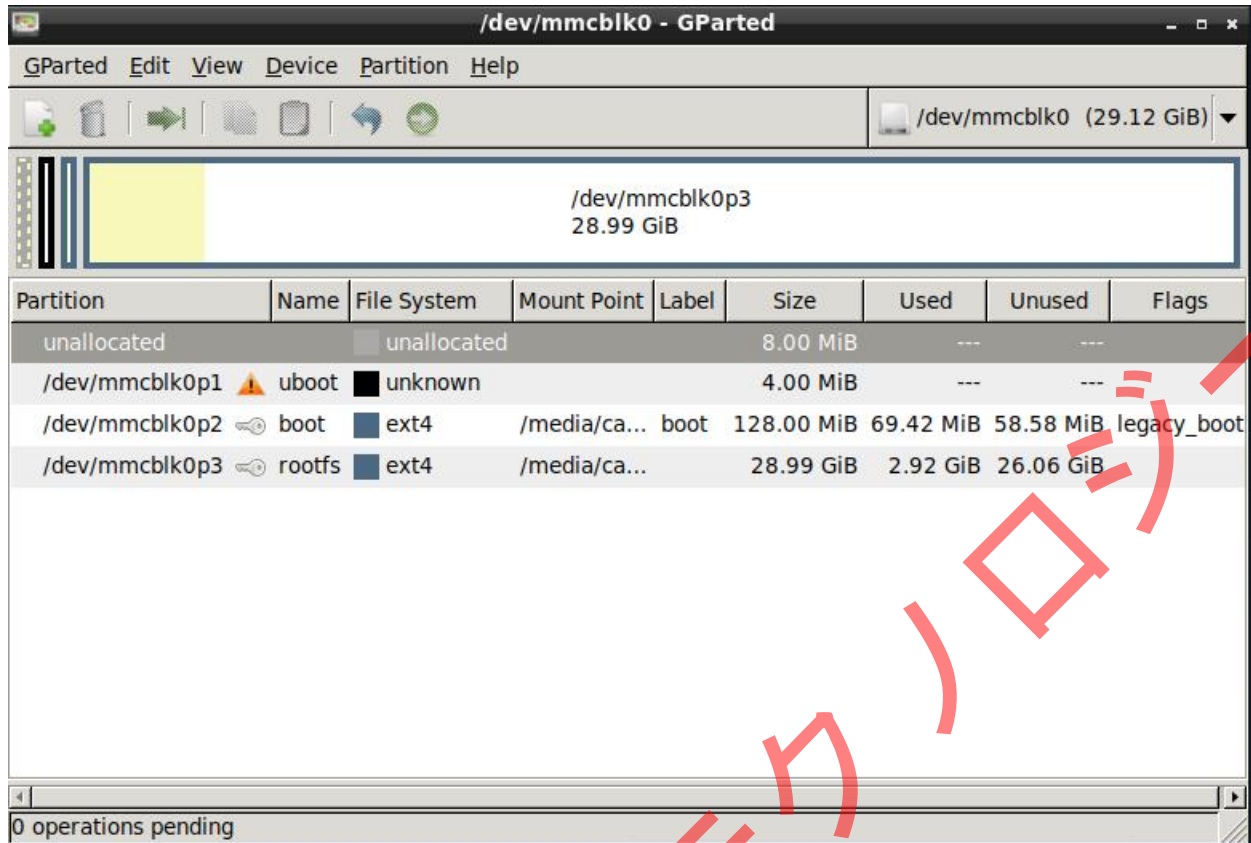
poweroff コマンドでシャットダウンし、SD カードを挿入して電源を入れると、システムは優先的に SD カードから起動します。

以降のステップは Linux PC を使用して SD カードをバックアップするプロセスと同様ですが、いくつかの注意点があります。

まず、eMMC のデバイス番号を取得する必要があります。/dev ディレクトリ下の mmc デバイスを確認し、mmcblkx[boot]x が eMMC であることを確認します。ここでは /dev/mmcblk0 です。

```
[root@RK356X:~]# ls -l /dev/mmc*
brw-rw---- 1 root disk 179, 0 Aug 4 09:00 /dev/mmcblk0
brw-rw---- 1 root disk 179, 32 Aug 4 09:00 /dev/mmcblk0boot0
brw-rw---- 1 root disk 179, 64 Aug 4 09:00 /dev/mmcblk0boot1
brw-rw---- 1 root disk 179, 1 Aug 4 09:00 /dev/mmcblk0p1
brw-rw---- 1 root disk 179, 2 Aug 4 09:00 /dev/mmcblk0p2
brw-rw---- 1 root disk 179, 3 Aug 4 09:00 /dev/mmcblk0p3
crw-rw---- 1 root root 238, 0 Aug 4 09:00 /dev/mmcblk0rpm
brw-rw---- 1 root disk 179, 96 Aug 4 09:00 /dev/mmcblk1
brw-rw---- 1 root disk 179, 97 Aug 4 09:00 /dev/mmcblk1p1
brw-rw---- 1 root disk 179, 98 Aug 4 09:00 /dev/mmcblk1p2
brw-rw---- 1 root disk 179, 99 Aug 4 09:00 /dev/mmcblk1p3
brw-rw---- 1 root disk 179, 100 Aug 4 09:00 /dev/mmcblk1p4
brw-rw---- 1 root disk 179, 101 Aug 4 09:00 /dev/mmcblk1p5
brw-rw---- 1 root disk 179, 102 Aug 4 09:00 /dev/mmcblk1p6
brw-rw---- 1 root disk 179, 103 Aug 4 09:00 /dev/mmcblk1p7
brw-rw---- 1 root disk 179, 104 Aug 4 09:00 /dev/mmcblk1p8
[root@RK356X:~]#
```

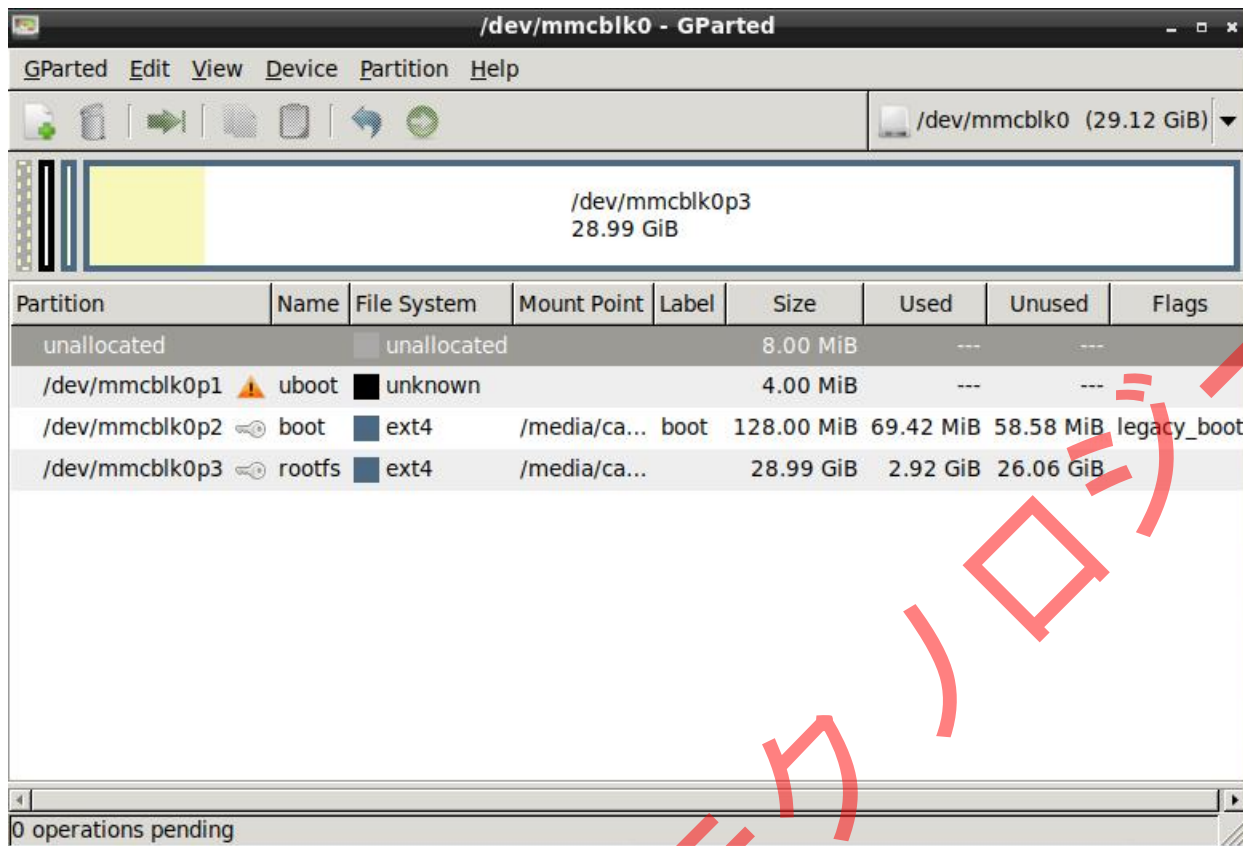
Gparted ディスク管理ツールを開き、ディスクを eMMC に切り替えます。



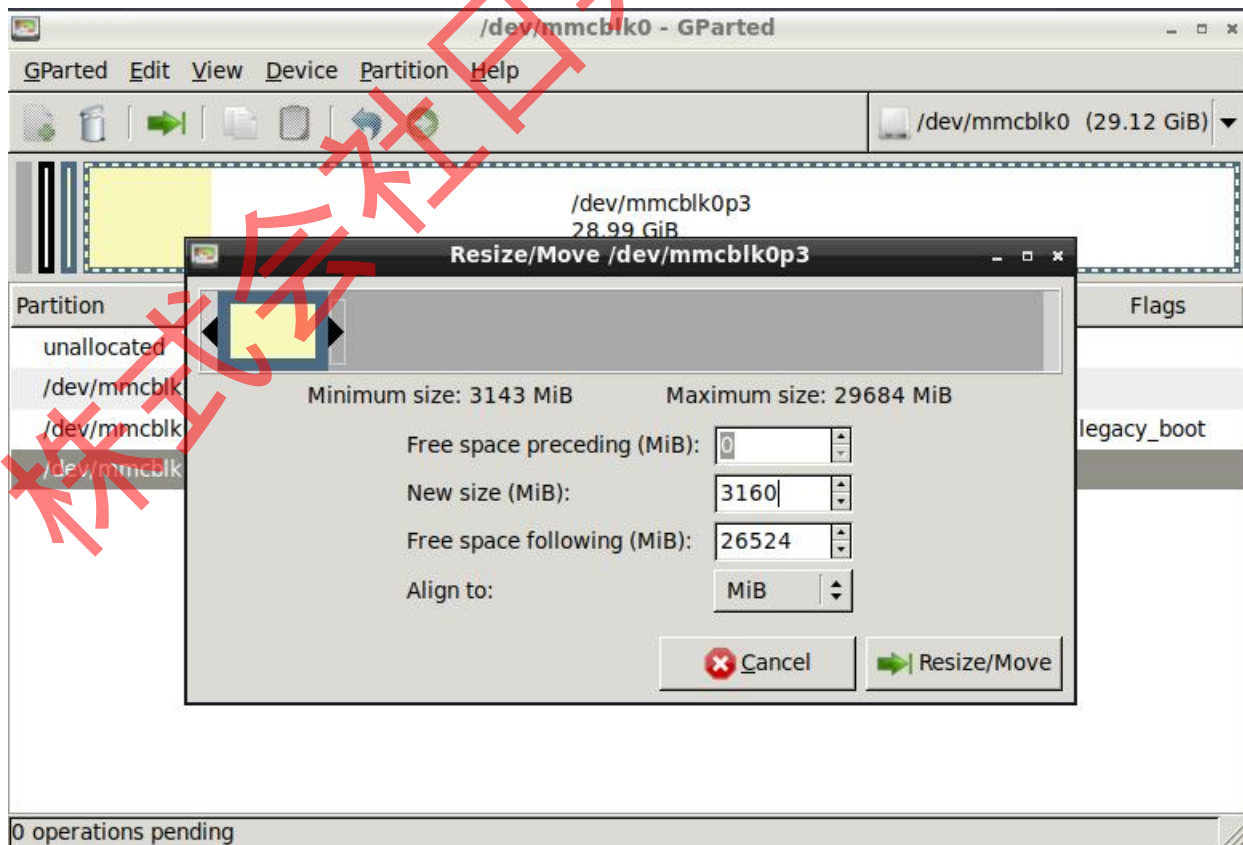
表示されるSDカードには3つの割り当てられたパーティションがあり、バックアップする内容は最後のパーティションとそれ以前の空間です。

最後のパーティション、つまり rootfs パーティションの合計空間は 28.99GiB ですが、使用済み空間は 2.92GiB のみです。rootfs パーティションのサイズを圧縮することで、バックアップイメージのサイズを減らすことができます。

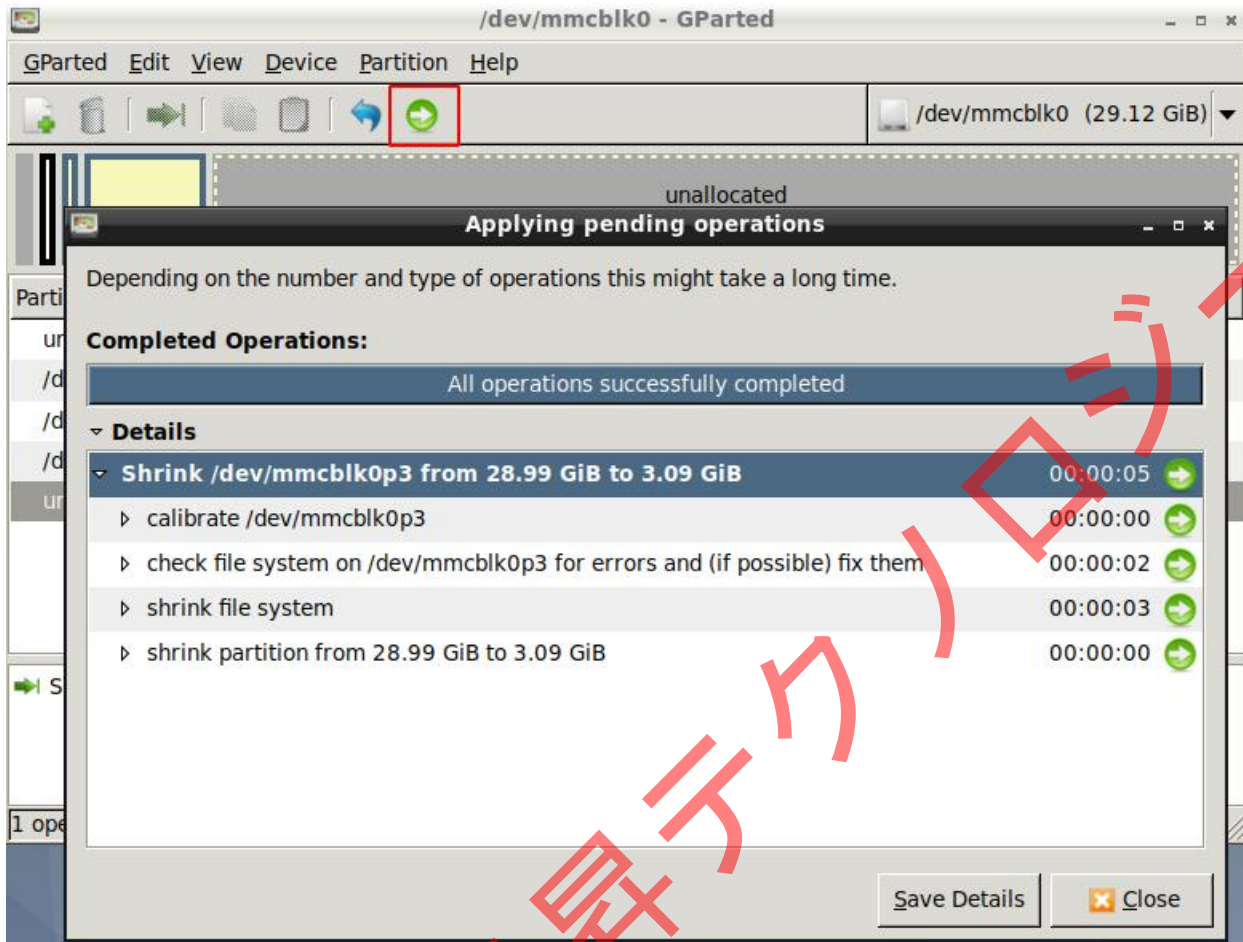
まず、eMMCにマウントされたすべてのパーティションをアンマウントします。例えば、boot パーティションや rootfs パーティションなどです。



次に、最後のパーティション、つまり rootfs パーティションのサイズを調整します。図からわかるように、最小の空間は 3143MiB です。それを 3160MiB に調整します。



調整が完了したら、緑のボタンをクリックして変更を適用します。



### 22.3.1 拡張とランダム MAC アドレスのネットワークポート

dd を使用して eMMC 内のシステムを完全にバックアップするため、ネットワークポートの MAC アドレスはランダムに生成され、uboot 環境変数に固定されます。これは上書きできませんが、MAC アドレスを変更しないと、新しく書き込まれたボードの一部または全部のネットワークポートの MAC アドレスが同じになり、ネットワーク通信ができなくなる可能性があります。しかし、私たちはスクリプトを使用してネットワークポートの MAC アドレスを再度ランダムに生成するか、自分で定義することができます。

また、バックアップされたイメージが書き込み起動時に最後のパーティションを自動的に拡張できるようにしたい場合は、以下の手順に従って操作できます。

**注意:** 以下のコマンドは管理者権限で実行する必要があります。root ユーザーで実行してい

ない場合は、コマンドの前に `sudo` を追加する必要があります。

汎用イメージと専用イメージの拡張方法は異なります。それぞれについて説明します。

汎用イメージでは以下の操作を行います：

まず、eMMC で起動し、`macchanger` ツールをインストールする必要があります。後続のランダム MAC アドレスのためにこのツールを使用します。インストール中に選択ウィンドウが表示された場合は、`NO` を選択してください。我々は自分でランダムに設定します。

1 # eMMC システムに `macchanger` ツールをインストール

2 `sudo apt update`

3 `sudo apt install macchanger`

次に、システムが書き込まれた SD カードを挿入し、SD カードからシステムを起動します。

システムに入ったら、以下の操作を実行します：

1. USB メモリをマウントするディレクトリを作成します
2. `mkdir -p /media/emmc`
- 3.
4. ルートファイルシステムパーティションをマウントします
5. `mount /dev/mmcblk0p3 /media/emmc/`
- 6.
7. システムの初期化スクリプトを開きます
8. `vim /media/emmc/etc/init.d/boot_init.sh`

システムの初期化スクリプト `boot_init.sh` の最後に以下の内容を追加します。 `boot_init.sh` スクリプトは、デフォルトでシステム起動時に自動的に実行されるように設定されています。

```
1 #-----拡張-----
2 # /boot/boot_dilatation_init ファイルが存在するか確認し、存在しない場合は拡張を実行する
3 if [ ! -e "/boot/boot_dilatation_init" ];
4 then
5
6 # /dev/mmcblk0p3 のルートファイルシステムパーティションのスペース設定を変更する
7 printf 'yes¥n-1¥nyes' | parted /dev/mmcblk0 resizepart 3 ---pretend-input-tty
8
9 # 設定に基づいてスペースを再割り当てする
10 resize2fs /dev/mmcblk0p3
11
12 # 判断ファイルを作成し、次回起動時にこのファイルが存在する場合は拡張を再実行しない
13 touch /boot/boot_dilatation_init
14 fi
15
16 #-----ランダム MAC アドレス-----
17 ifconfig eth0 down
18 ifconfig eth1 down
19
20 # /boot/boot_mac_eth0 または/boot/boot_mac_eth1 が存在しない場合はランダム MAC を実行する
21 if [ ! -e "/boot/boot_mac_eth0" ] || [ ! -e "/boot/boot_mac_eth1" ];
22 then
```

```
23 #sleep 3

24 # ランダム MAC

25 sudo macchanger -r eth0

26 sudo macchanger -r eth1

27

28 # 現在のインターフェースの MAC アドレスを取得する

29 mac_address_eth0=$(ifconfig eth0 | grep ether | awk '{ print $2 }')

30 mac_address_eth1=$(ifconfig eth1 | grep ether | awk '{ print $2 }')

31

32 # 判断ファイルを作成し、次回起動時にこのファイルが存在する場合はランダム MAC を再実行しない

33 touch /boot/boot_mac_eth0

34 touch /boot/boot_mac_eth1

35

36 # ランダム生成された MAC を判断ファイルに保存する

37 echo "$mac_address_eth0" > /boot/boot_mac_eth0

38 echo "$mac_address_eth1" > /boot/boot_mac_eth1

39 fi

40

41 # 生成された MAC を取得して変更する

42 mac_address_eth0=$(cat /boot/boot_mac_eth0)

43 mac_address_eth1=$(cat /boot/boot_mac_eth1)

44 sudo ifconfig eth0 hw ether $mac_address_eth0
```



```
45 sudo ifconfig eth1 hw ether $mac_address_eth1
```

```
46
```

```
47 sudo ifconfig eth0 up
```

```
48 sudo ifconfig eth1 up
```

変更が完了したら、マウントされているパーティションをアンマウントします

```
1. # 変更が完了したら、マウントされているパーティションをアンマウントします
```

```
2. umount /media/emmc/
```

ヒント：二次バックアップが必要な場合は、上記の判断ファイルを削除してください。そうしないと、二次バックアップ後に他のボードに書き込む際に、ファイルが存在すると拡張とランダム MAC が実行されません。

専用イメージでは以下の操作を行います：

```
1. #USB メモリをマウントするディレクトリを作成します
```

```
2. mkdir -p /media/emmc
```

```
3.
```

```
4. #USB メモリをマウントします
```

```
5. mount /dev/mmcblk0p3 /media/emmc/
```

```
6.
```

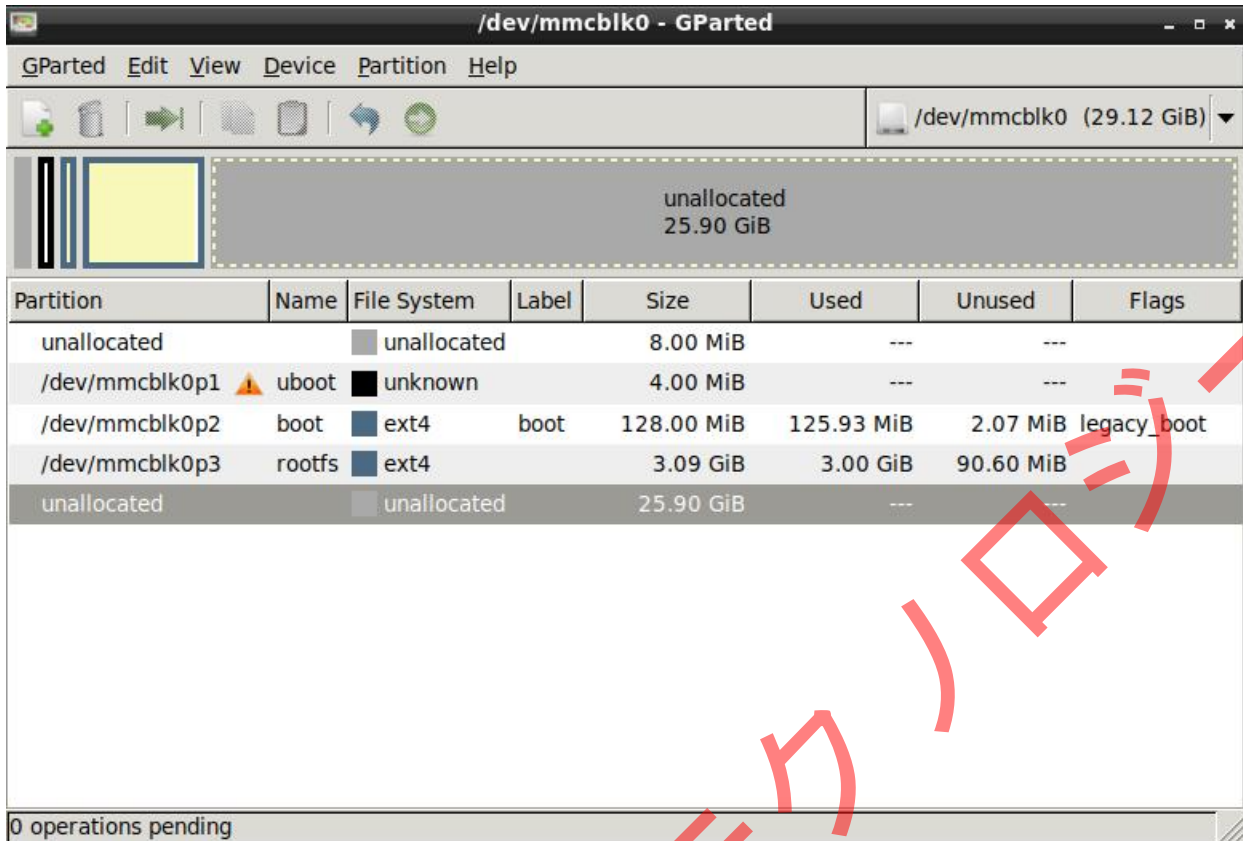
```
7. #このファイルを削除します。バックアップイメージで起動すると、パーティションが自動的に拡張されます
```

```
8. rm /media/emmc/var/lib/misc/firstru
```

```
9.
```

```
10. #削除が完了したら、マウントされているパーティションをアンマウントします
```

```
11. umount /media/emmc/
```



Partition	Name	File System	Label	Size	Used	Unused	Flags
unallocated		unallocated		8.00 MiB	---	---	
/dev/mmcblk0p1	uboot	unknown		4.00 MiB	---	---	
/dev/mmcblk0p2	boot	ext4	boot	128.00 MiB	125.93 MiB	2.07 MiB	legacy_boot
/dev/mmcblk0p3	rootfs	ext4		3.09 GiB	3.00 GiB	90.60 MiB	
unallocated		unallocated		25.90 GiB	---	---	

バックアップのサイズを手で計算してみましょう  $(8+4+128+3.09 \times 1024) \text{MiB} \approx 3305 \text{MiB}$ 。表示されるサイズは小数点以下 2 桁を四捨五入しているため、少し余裕を見て、3350MiB のサイズでバックアップを取ります。

USB メモリをマウントするためのディレクトリを作成するには `mkdir` コマンドを使用し、次に `mount` コマンドを使用して USB メモリをマウントし、最後に `dd` コマンドを使用してマウントされた USB メモリにイメージをバックアップします。

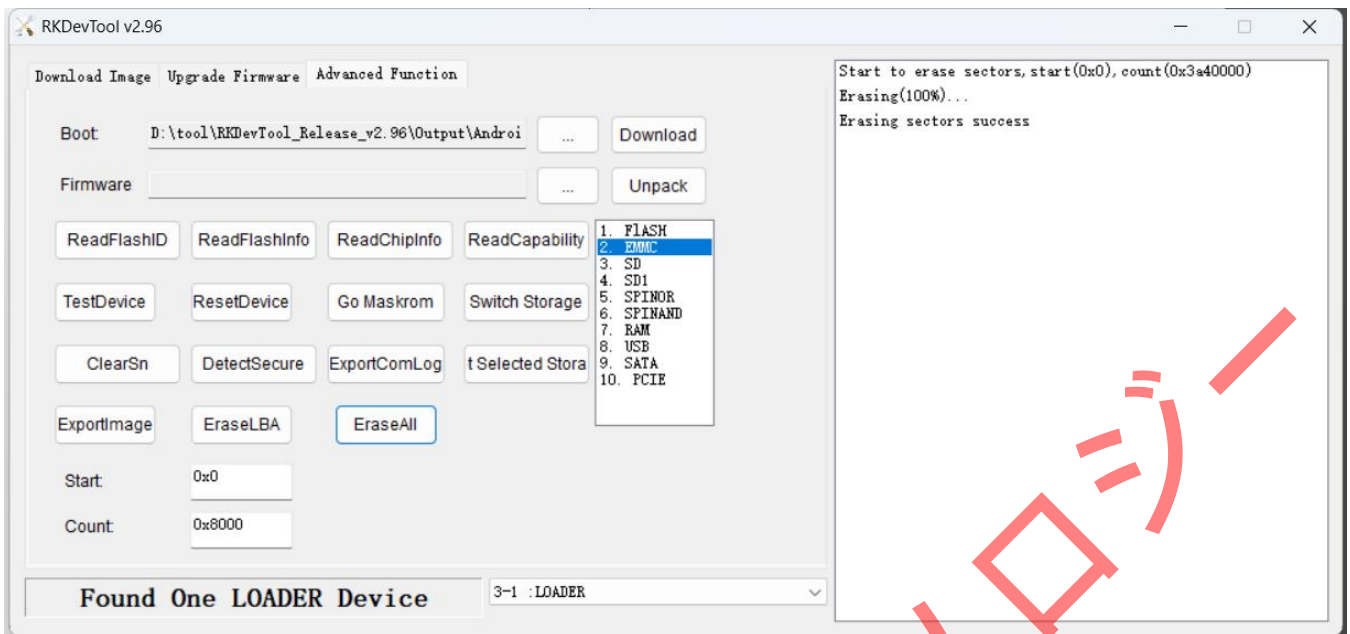
1. # USB メモリをマウントするディレクトリを作成します
2. `mkdir -p /media/usb`
- 3.
4. # USB メモリをマウントします
5. `mount /dev/sda1 /media/usb/`
- 6.

```
7. # イメージをマウントされた USB メモリにバックアップします
8. dd if=/dev/mmcblk0 of=/media/usb/backup.img count=3350 bs=1024k conv=sync
9.
10. # コピーには時間がかかりますので、以下のメッセージが表示されるまで、しばらくお待ちください
11. 3350+0 records in
12. 3350+0 records out
13. 3512729600 bytes (3.5 GB, 3.3 GiB) copied, 198.883 s, 17.7 MB/s
14.
15. # バックアップが完了したら、USB メモリをアンマウントします
16. umount /media/usb/
```

dd コマンドの実行が完了すると、RAW 形式の backup.img イメージが得られます。USB メモリにイメージが完全に書き込まれていることを確認するために、直接 USB メモリを抜かずに、umount コマンドを使用してアンマウントした後に USB メモリを抜いてください。

## 22.4 RKDevTool を使用して eMMC に RAW 形式のイメージを書き込む

システムイメージが実際に eMMC に書き込まれていることを確認するために、まず元のイメージを削除します。RKDevTool の高度な機能を使用して、eMMC を完全に消去します。

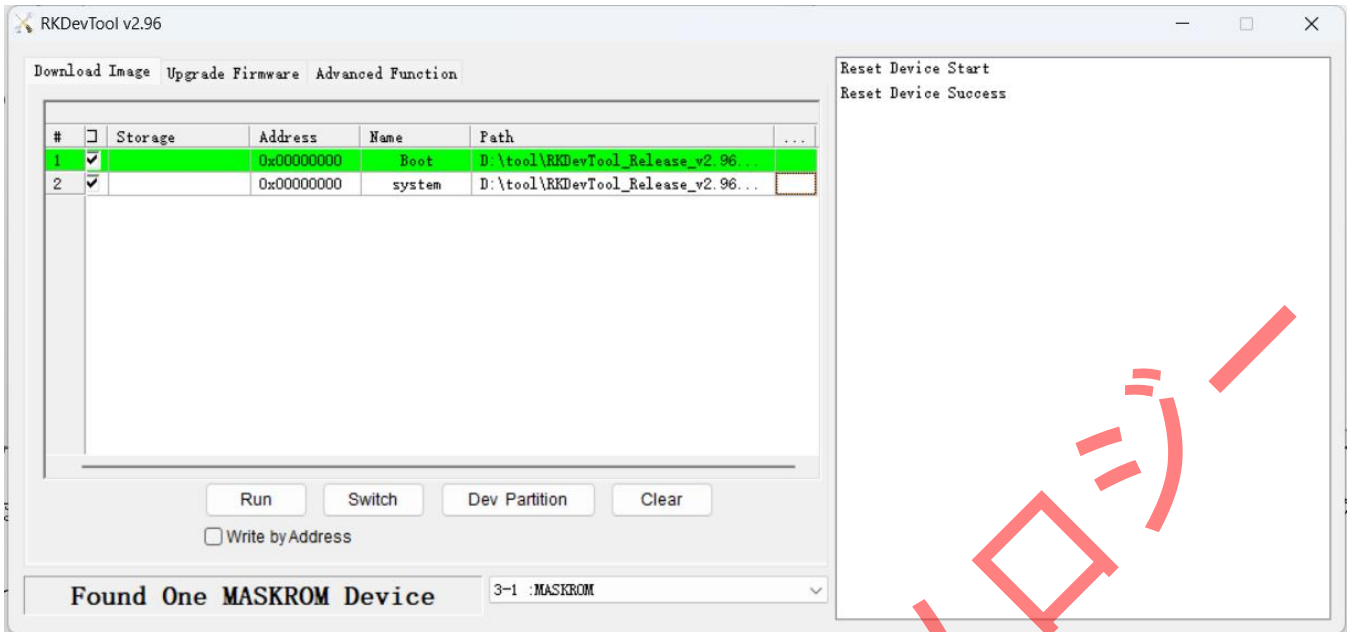


注意：RKDevTool の高度な機能における「すべて消去」機能は Loader ファイルを消去できませんが、他のパーティションは消去できます。

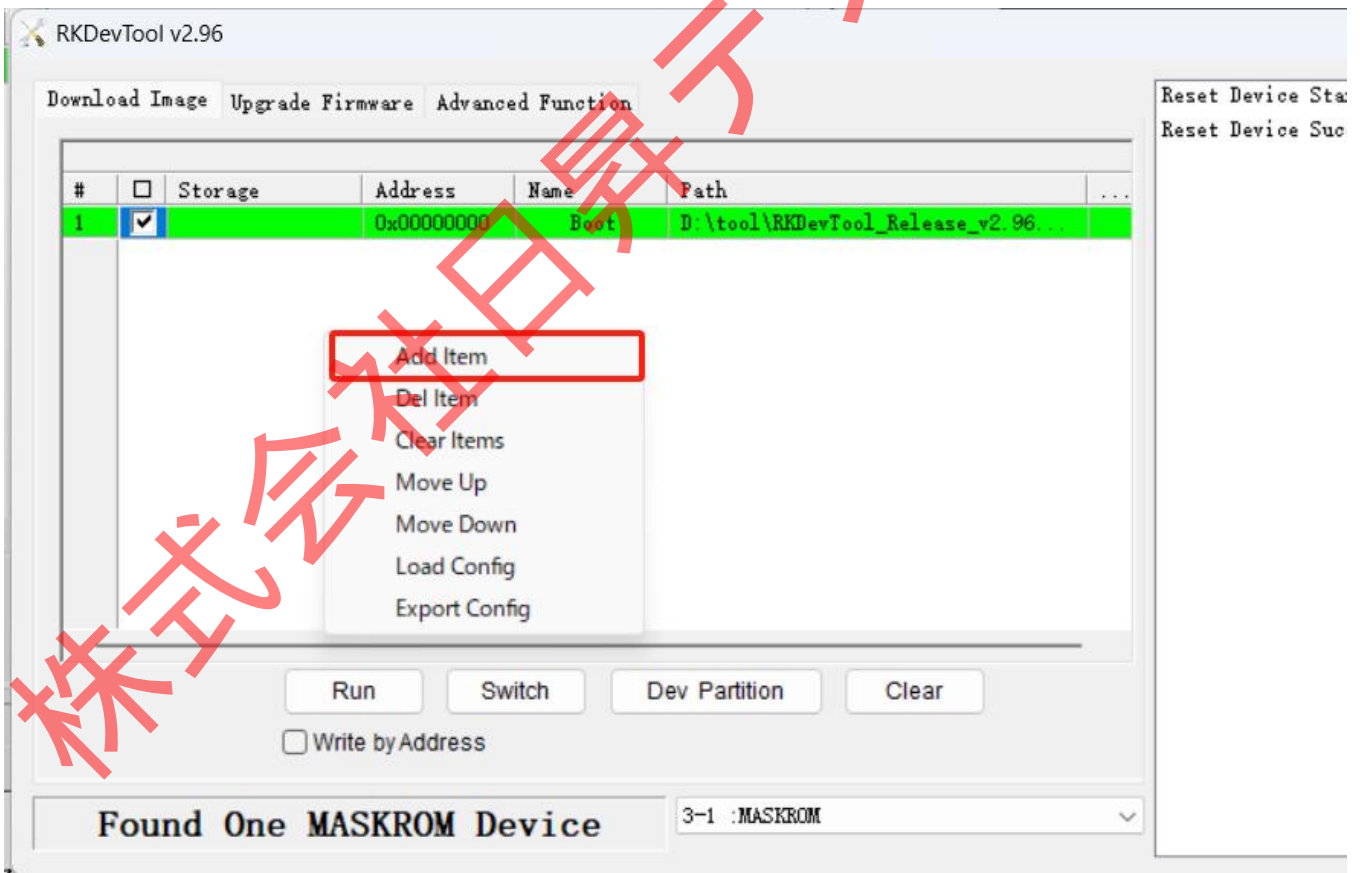
次に、RKDevTool を使用して eMMC に RAW 形式のイメージを書き込みます。

RAW 形式のイメージを書き込むには、RKDevTool\_Release\_v2.86 のバージョンを使用する必要があります。最新版の書き込みツールを使用すると失敗します。

RKDevTool\_Release\_v2.86 を開き、デフォルトの設定ファイルが RAW 形式の書き込み用である場合、以下の画像の赤い枠の内容と一致していなければ、アドレスの下の空白エリアを右クリックして設定をインポートし、RKDevTool\_Release\_v2.86 と同じフォルダの RAW.cfg を選択します。



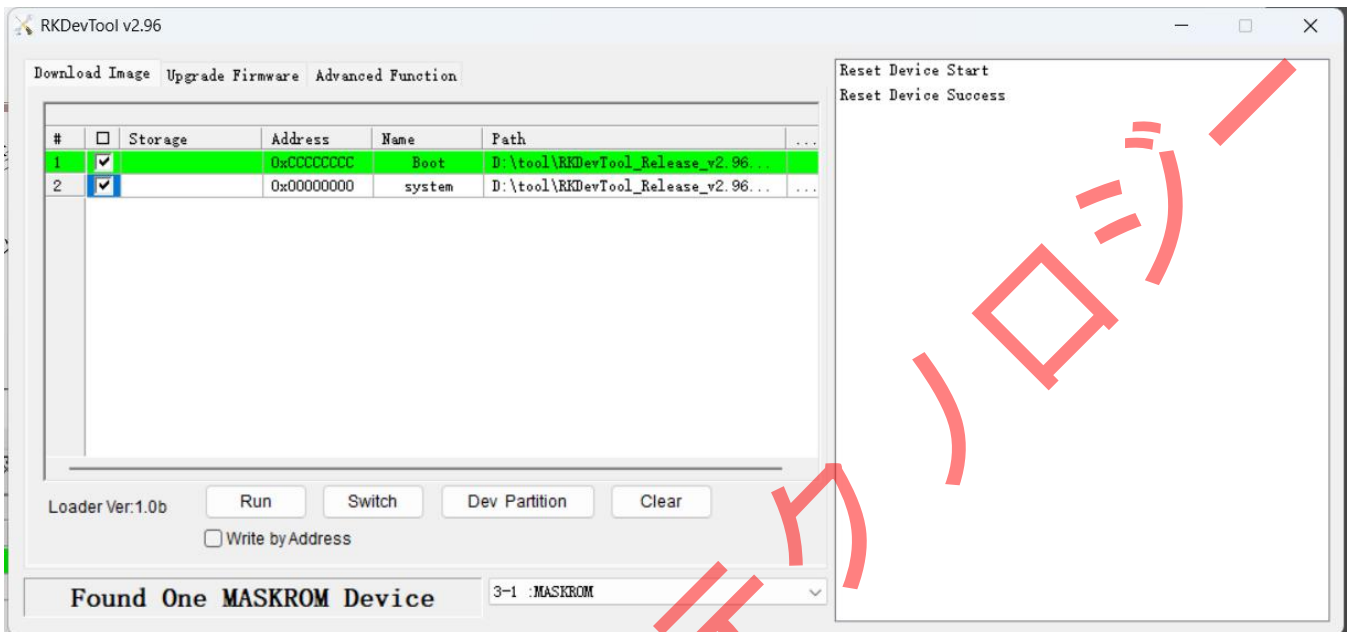
また、画像のアドレスと名前に従って手動でアドレスと名前をクリックして変更することも、右クリックで空白部分に項目を追加することもできます。



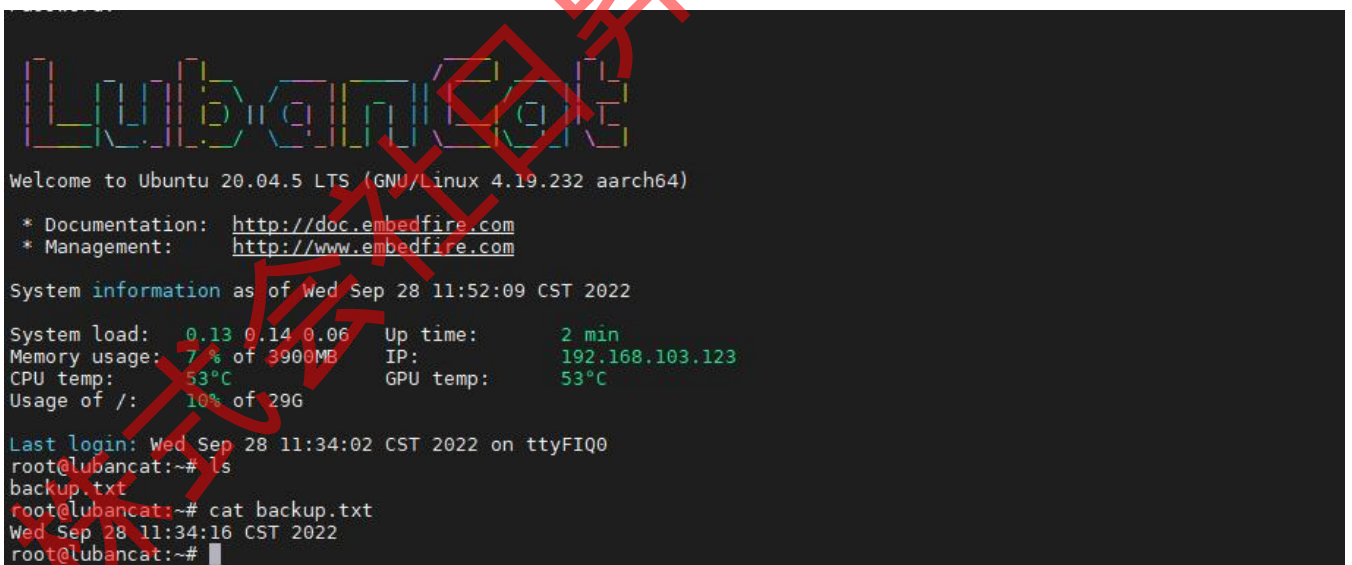
準備が整ったら、パスの後ろの...をクリックして該当するファイルを追加します。BootはLoaderファイル、つまりチップモデルに対応するMiniLoaderAll.binを指し、systemはRAW形式のイメージ、バックアップ

された RAW 形式のイメージや OpenWrt の RAW 形式のイメージなどが対象です。

ここではバックアップイメージを例に、イメージを選択した後に実行をクリックしてダウンロードを開始します。



ダウンロードが完了したらボードを起動し、eMMC に以前作成したファイルを確認します。



## 22.5 dd コマンドを使用して eMMC に RAW 形式のイメージを書き込む

RKDevTool を使用して eMMC に書き込む以外に、dd コマンドを使用して RAW 形式の backup.img イメージを eMMC に書き込むこともできます。

dd コマンドを使用して eMMC に RAW 形式のイメージを書き込むプロセスは、eMMC の内容を RAW 形式で backup.img イメージにバックアップするプロセスに似ていますが、書き込みプロセスは RAW 形式の backup.img イメージを eMMC に書き込むことです。

システムイメージが実際に eMMC に書き込まれていることを確認するために、まず元のイメージを削除します。dd コマンドを使用して eMMC の最初の 16M を空の内容で書き込み、元のパーティションテーブルを破壊すれば十分です。

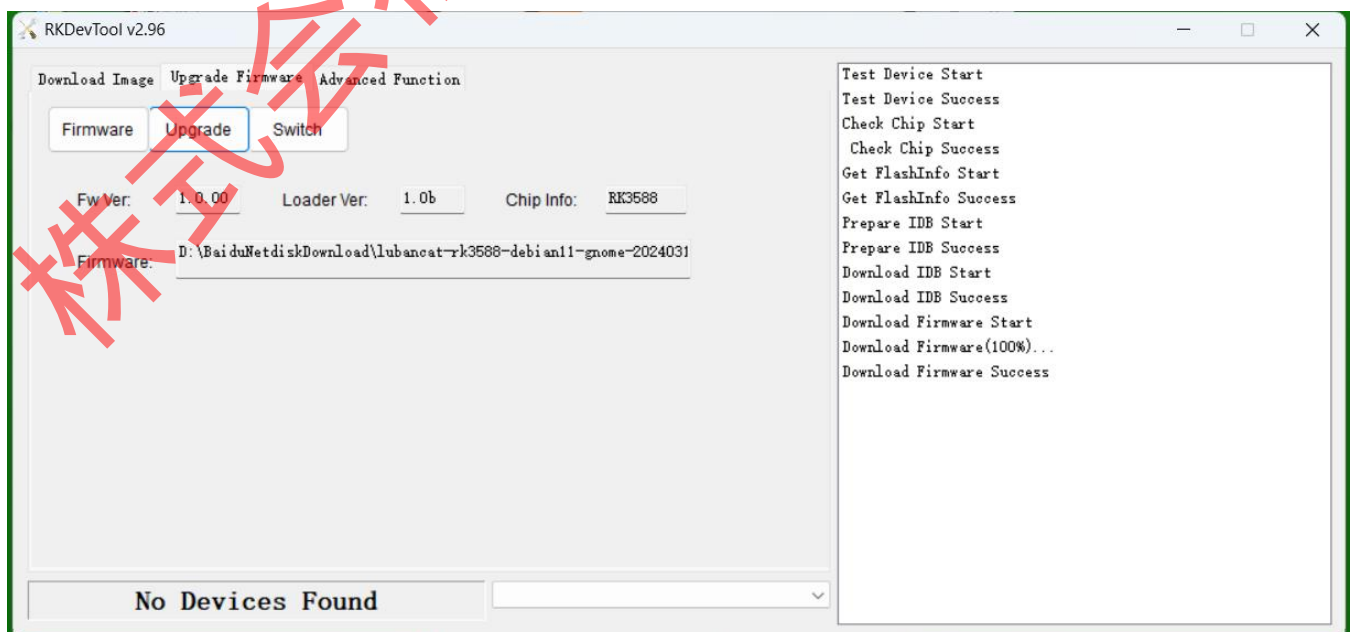
1. # eMMC の最初の 16M を空の内容で書き込みます
2. dd if=/dev/zero of=/dev/mmcblk0 bs=1M count=16

SD カードを取り外し、再び電源を入れると、Maskrom モードに直接入ります。

バックアップ用のボードの汎用バックアップイメージ RK358X-LUBANCATBACKUP\_xxx\_Update.img (xxx は更新日) を書き込むための SD カードを準備する必要があります。xxx は更新日です。

書き込む RAW 形式のイメージを保存するためのストレージデバイスも必要です。この目的には、先ほどバックアップに使用した USB メモリを使用します。

dd コマンドを使用して書き込んだイメージが、バックアップしたイメージであることを検証するために、まず Debian イメージを eMMC に書き込んで内容を上書きします。



SD カードを挿入し、SD カードからオペレーティングシステムを起動します。システムが起動すると自動的にパーティションがマウントされるため、まずは`/media/cat/`ディレクトリ下のすべてのパーティションを手動でアンマウントし、その後再マウントします。

```
1. # /media/cat/ディレクトリ下にマウントされたすべてのパーティションをアンマウントします
2. umount /media/cat/*
3.
4. # USB メモリを`/mnt`ディレクトリにマウントします
5. mount /dev/sda1 /mnt
6.
7. # eMMC の内容を`/mnt/backup.img`にバックアップします
8. dd if=/mnt/backup.img of=/dev/mmcblk0 bs=1024k conv=sync
9.
10. # コピーには時間がかかりますので、以下のメッセージが表示されるまでお待ちください
11. 3350+0 records in
12. 3350+0 records out
13. 3512729600 bytes (3.5 GB, 3.3 GiB) copied, 133.043 s, 26.4 MB/s
14.
15. # USB メモリをアンマウントします
16. umount /mnt/
17.
18. # メモリ内のキャッシュをストレージデバイスに書き込みます
19. sync
```



20.

21. # システムをシャットダウンします

22. poweroff

SD カードを取り外し、再び電源を入れて起動します。

```
Ubuntu 20.04.4 LTS GNU/Linux lubancat ttyFIQ0
[username:password] root:root cat:tempPWD
Modify information : /etc/issue
lubancat login: [ 11.360602] rk-pcie 3c0000000.pcie: PCIe Link Fail
[ 11.360644] rk-pcie 3c0000000.pcie: failed to initialize host
lubancat login: root
Password:

Lubancat

Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 4.19.232 aarch64)

 * Documentation:  http://doc.embedfire.com
 * Management:    http://www.embedfire.com

System information as of Wed Aug 31 23:27:58 CST 2022
System load:  2.44 0.64 0.22   Up time:      0 min
Memory usage: 8 % of 3900MB   IP:          192.168.103.123
CPU temp:    53°C           GPU temp:    52°C
Usage of /:  9% of 29G

Last login: Wed Sep 28 11:34:02 CST 2022 on ttyFIQ0
root@lubancat:~# ls
backup.txt
root@lubancat:~# cat backup.txt
Wed Sep 28 11:34:16 CST 2022
root@lubancat:~#
```

## 第23章 ルートファイルシステムのバックアップ と再書き込み

システムイメージを完全にバックアップする利点は、処理が便利で、手順が簡単であることです。しかし、完全バックアップされたシステムイメージの容量は大きく、量産ツールを使っての書き込みができません。

ルートファイルシステムのみを変更した場合、ルートファイルシステムパーティションのみを個別にバックアップすることができます。同様に、boot パーティションや他のパーティションを変更した場合も、それぞれのパーティションを個別にバックアップできます。しかし、開発過程では、他のパーティションを変更することは一般的ではないため、ここでは rootfs パーティションのバックアップに重点を置いて説明

します。他のパーティションのバックアップと書き込み方法も同様です。

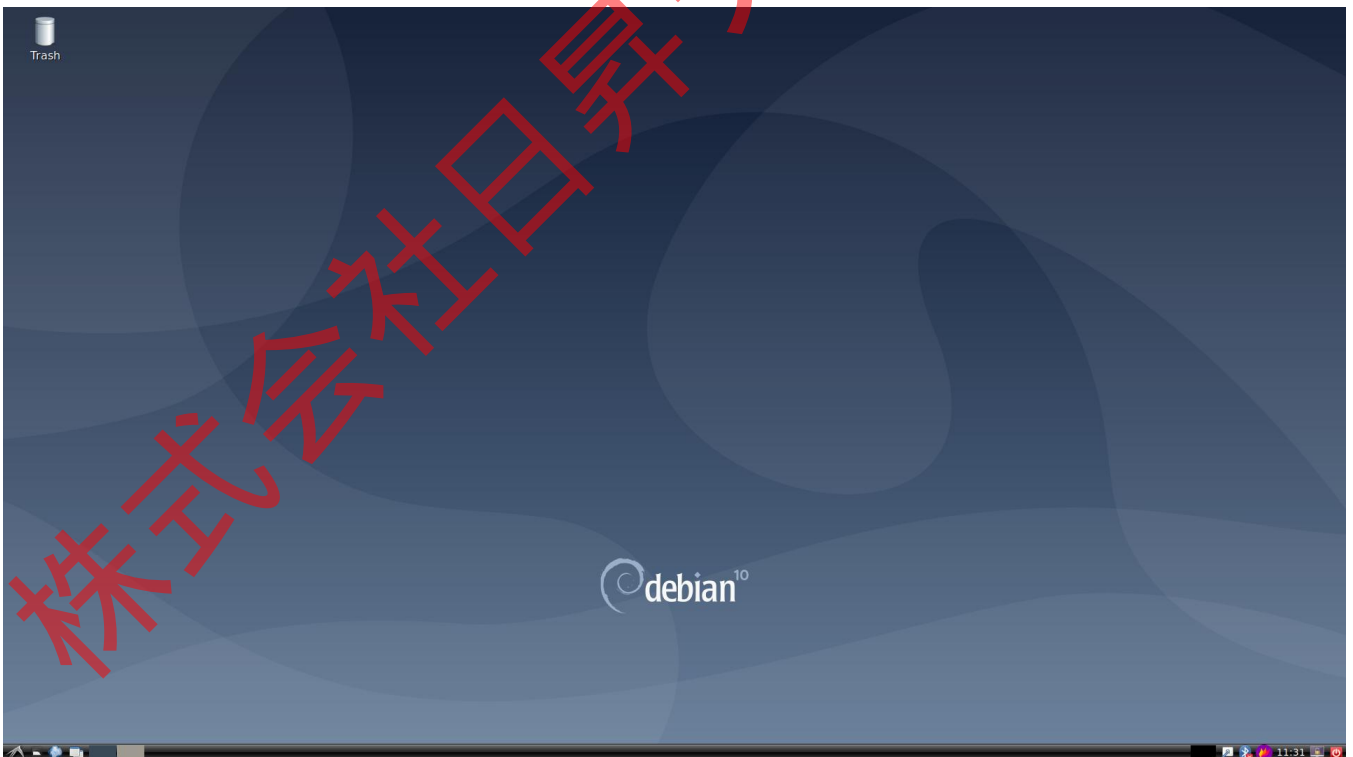
## 23.1 バックアップ前の準備

ルートファイルシステムのバックアップをスムーズに行うために、いくつかの準備作業を先に行います。

- eMMC または SD カードに書き込まれる RK 形式のイメージソースファイルを解凍し、各パーティションのイメージを取得します。
- 後期のイメージ処理用の Linux PC または仮想マシン
- eMMC のバックアップを行う場合は、一般的なバックアップイメージが書き込まれた SD カードが必要です。

## 23.2 ルートファイルシステムのバックアップ

rootfs パーティションが正しくバックアップされたことを確認するために、デスクトップの背景画像を以下の画像に変更します。



次に以下の手順を実行し、再起動時にパーティションが自動的に拡張されるようにします。

専用イメージでは以下の操作を行います：

1. # このファイルを削除すると、再起動時にパーティションが自動的に拡張されます
2. rm /var/lib/misc/firstrun
- 3.
4. # 削除が完了したら、シャットダウンします
5. poweroff
- 6.
7. 汎用イメージと専用イメージの拡張方法が異なりますので、それぞれについて説明します。

汎用イメージでは以下の操作を行います：

システム初期化スクリプト/etc/init.d/boot\_init.shを開きます。このスクリプトは、デフォルトでシステム起動時に自動的に実行されるように設定されています。

1. vim /etc/init.d/boot\_init.sh

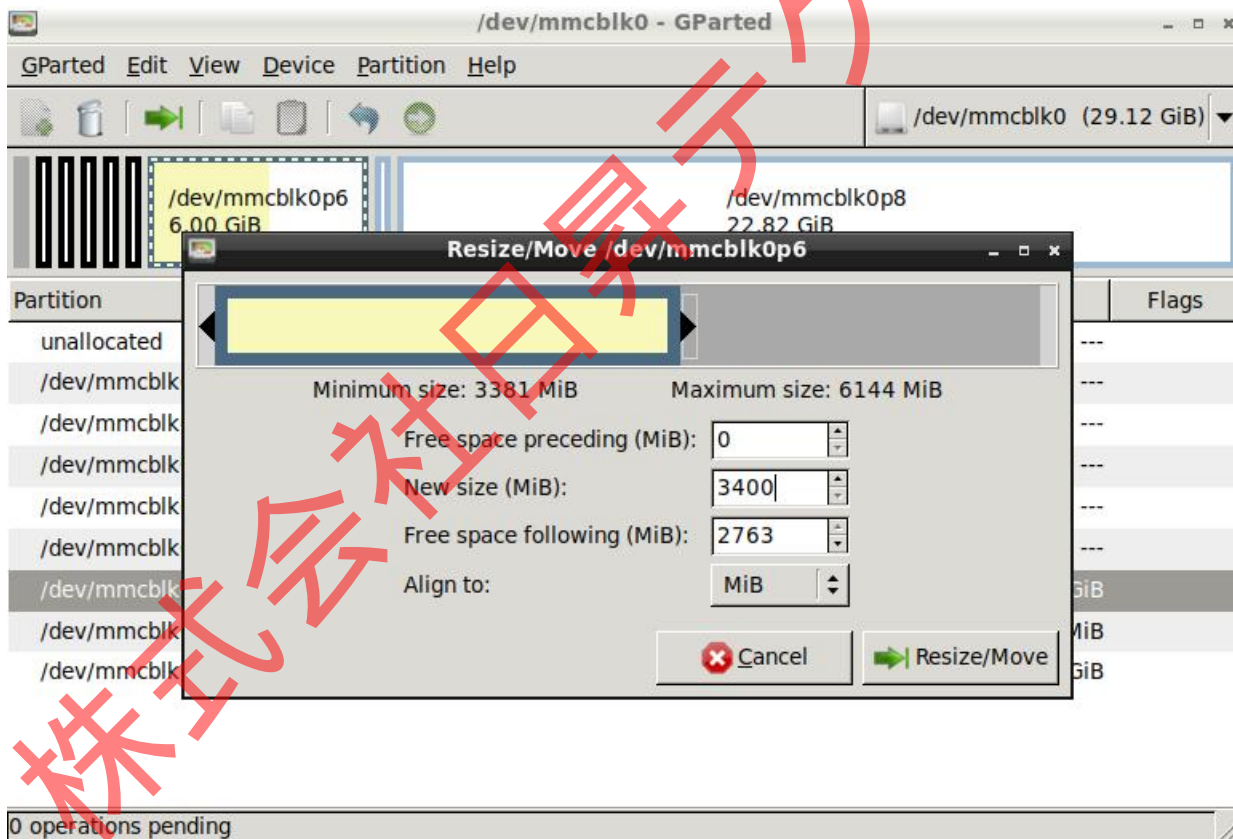
汎用イメージのルートファイルシステムパーティションが/dev/mmcbk0p3であることを、df -h コマンドで確認できます。システム初期化スクリプトの末尾に以下の内容を追加します：

1. if [ ! -e "/boot/boot\_dilatation\_init" ] ;
2. then
- 3.
4. # /dev/mmcbk0p3 のルートファイルシステムパーティションの空間設定を変更
5. printf 'yes\n-1\nyes' | parted /dev/mmcbk0 resizepart 3 ---pretend-input-tty
- 6.
7. # 設定に基づいて空間を再割り当て
8. resize2fs /dev/mmcbk0p3

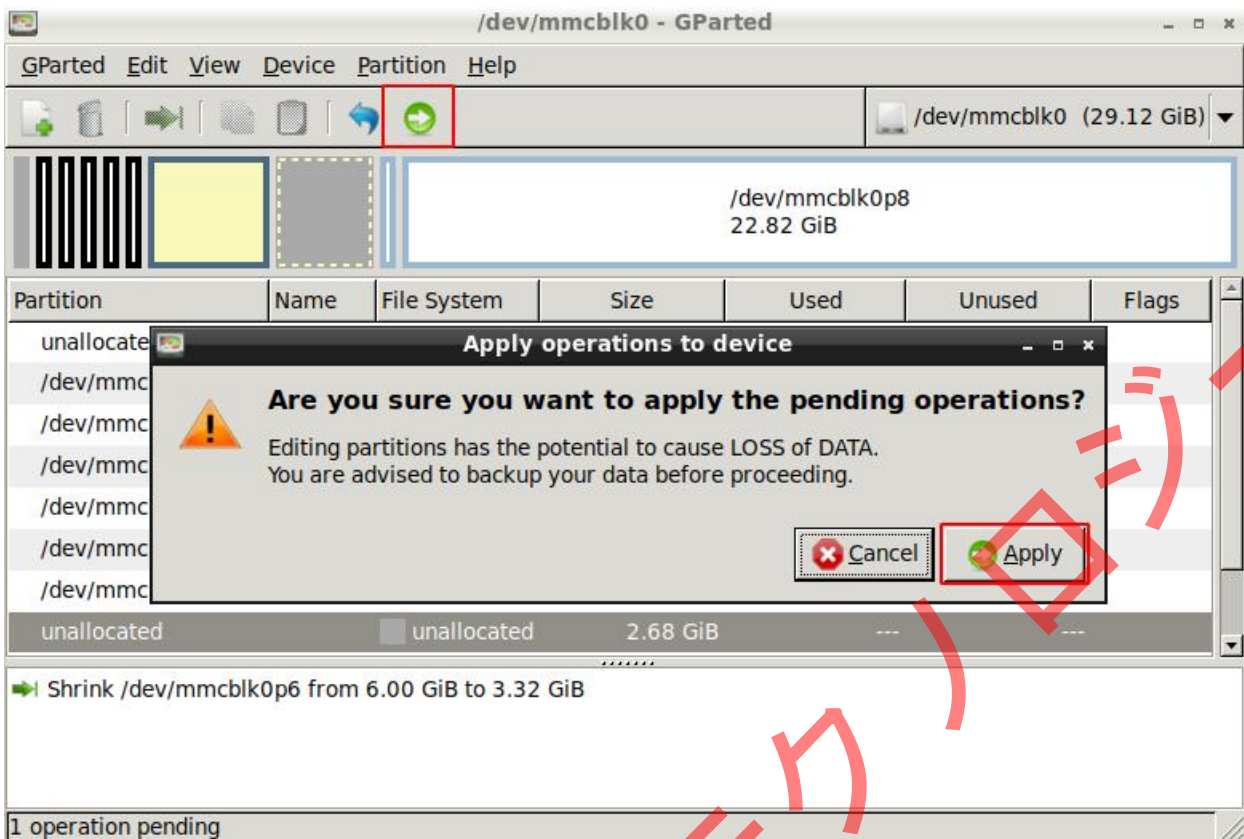
- 9.
10. # 判断ファイルを作成し、2回目の起動時にこの拡張を実行しない
11. touch /boot/boot\_dilatation\_init
12. fi

汎用バックアップイメージが焼かれたSDカードを挿入し、SDカードからシステムを起動します。バックアップされるSDカード上のrootfsパーティションの場合、このステップはLinux PCにバックアップされるSDカードを挿入することです。

gparted ディスク管理ツールを開き、バックアップするrootfsパーティションをアンマウントして圧縮します。



変更を適用し、rootfs のボリュームを縮小します。



mkdir コマンドを使用して新しいディレクトリを作成し、イメージが含まれたSDカードからコピーしたイメージを保存します。次に、dd コマンドを使用して、イメージが含まれたSDカードのイメージを新しく作成されたディレクトリにコピーします。ここでは、マウントされたUSBメモリを選択します。

パーティションを圧縮する際に、eMMC の rootfs パーティションは /dev/mmcblk0p6 パーティションであることがわかりました。

1. # USBメモリをマウント
2. mount /dev/sda1 /mnt/udisk/
- 3.
4. # イメージが含まれたSDカードから作成したディレクトリにイメージをコピー
5. dd if=/dev/mmcblk0p6 of=/mnt/udisk/backup-rootfs.img bs=1024k conv=sync
- 6.
7. # コピーには時間がかかりますので、お待ちください。メッセージは以下の通りです。

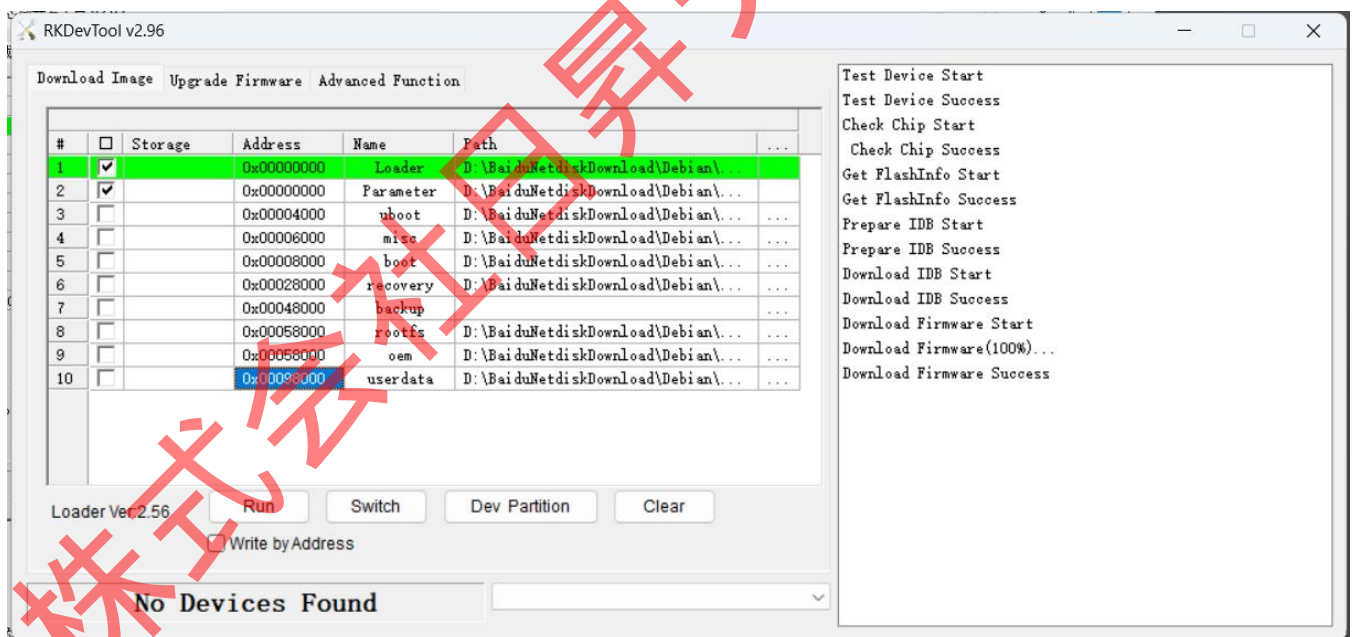
8. 3400+0 records in
9. 3400+0 records out
10. 3585158400 bytes (3.6 GB, 3.3 GiB) copied, 185.89 s, 19.2 MB/s

バックアップが完了したら、poweroff コマンドでシャットダウンし、USB メモリを取り外します。これで、rootfs.img パーティションのイメージを個別に取得しました。

開発ツールを使用してパーティションに直接書き込むことも、バックアップされた rootfs.img を他のパーティションと統合して完全なイメージにすることも選択できます。

### 23.3 パーティションの書き込み

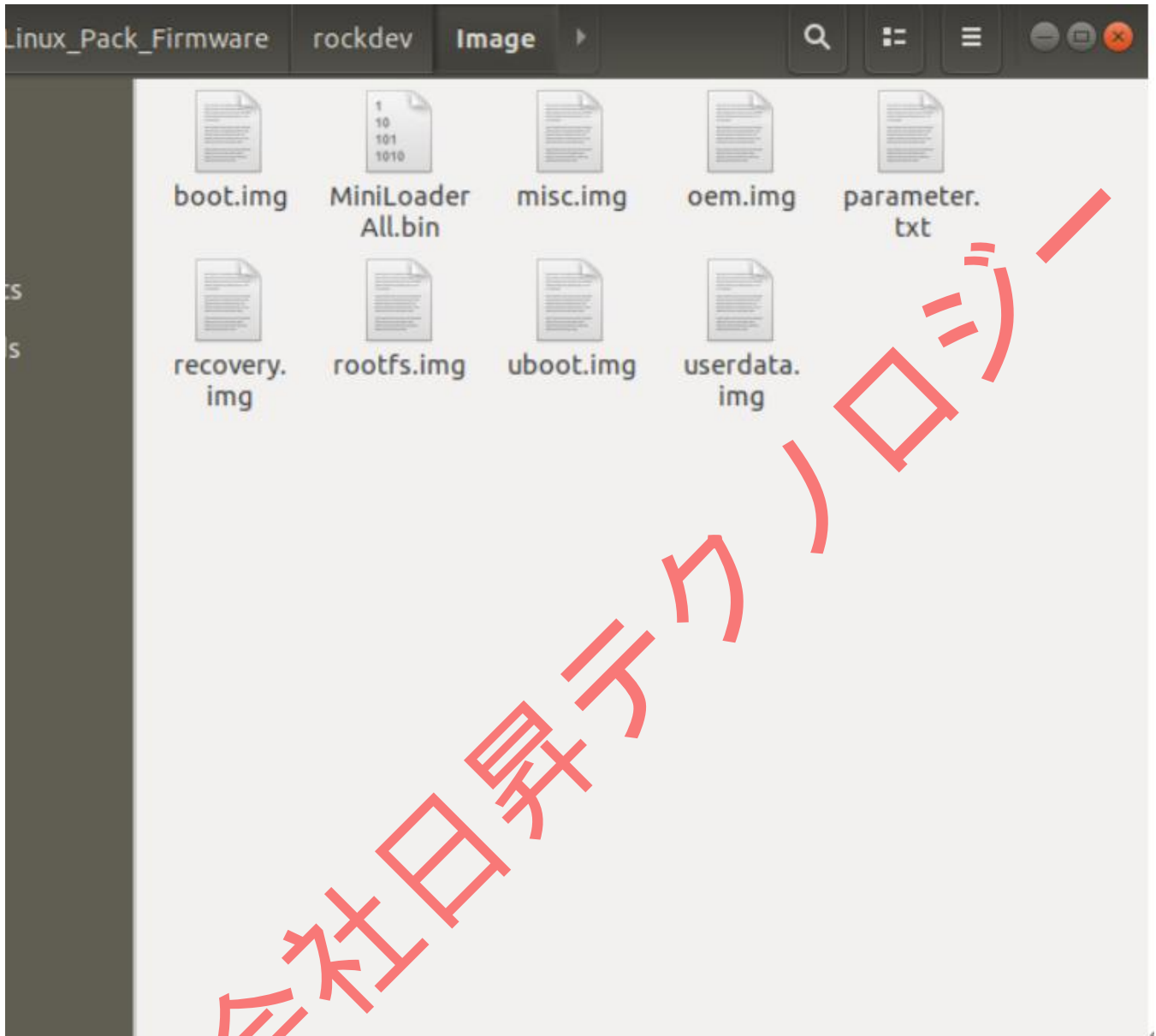
開発ツールを使用して対応するパーティションテーブルをインポートし、次に RK 形式の完全なイメージの解凍後の各パーティションイメージを順番に選択します。rootfs パーティションには、バックアップで取得したばかりの backup-rootfs.img イメージファイルを選択し、書き込みを行います。



### 23.4 完整イメージのパッケージングと書き込み

パッケージングツールを開き、以前に解凍したパーティションイメージと最近バックアップした backup-rootfs.img を Image ディレクトリにコピーします。そして、backup-rootfs.img を rootfs.img にリネームして元のファイルを置き換え、rk358x-mkupdate.sh を実行して完全なイメージをパッケージングしま

す。

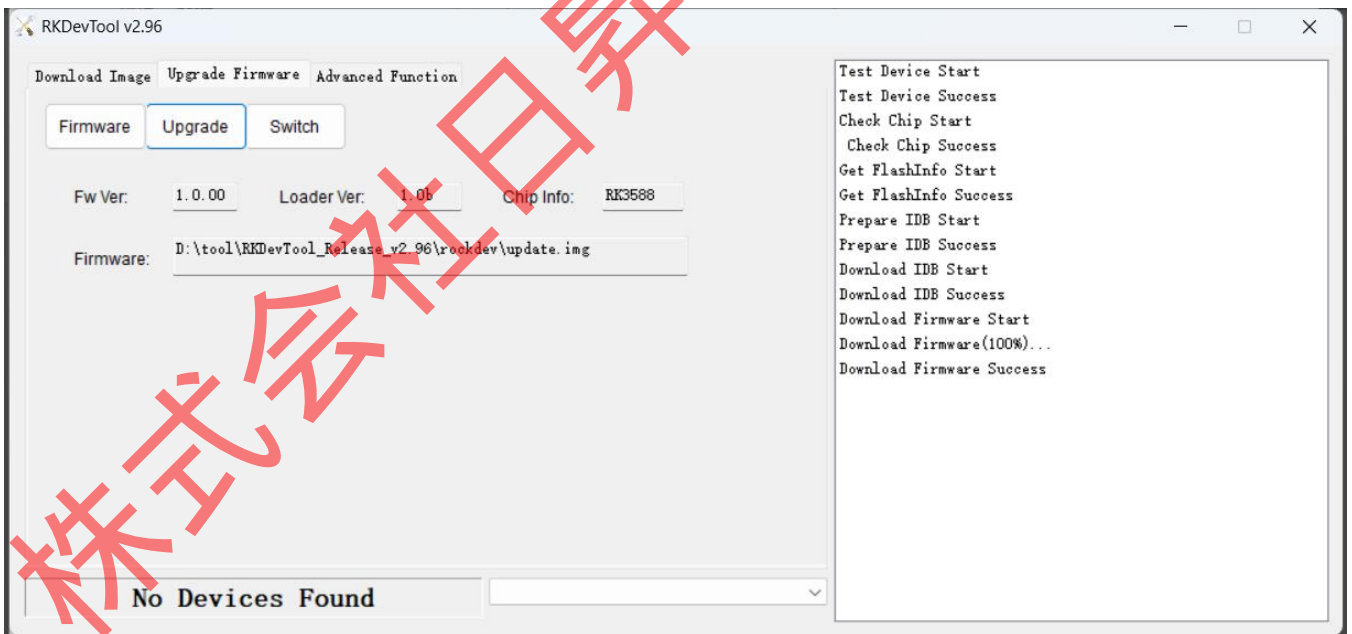


```

Add file: ./Image/boot.img
boot,Add file: ./Image/boot.img done,offset=0x47f800,size=0x1b9e600,userspace=0x373d
Add file: ./Image/recovery.img
recovery,Add file: ./Image/recovery.img done,offset=0x201e000,size=0x2232600,userspace=0x4465
Add file: ./Image/rootfs.img
rootfs,Add file: ./Image/rootfs.img done,offset=0x4250800,size=0xd4800000,userspace=0x1a9000
Add file: ./Image/oem.img
oem,Add file: ./Image/oem.img done,offset=0xd8a50800,size=0x10a6000,userspace=0x214c
Add file: ./Image/userdata.img
userdata,Add file: ./Image/userdata.img done,offset=0xd9af6800,size=0x444000,userspace=0x888
Add CRC...
Make firmware OK!
----- OK -----
*****rkImageMaker ver 2.0*****
Generating new image, please wait...
Writing head info...
Writing boot file...
Writing firmware...
Generating MD5 data...
MD5 data generated successfully!
New image generated successfully!
Making ./Image/update.img OK.
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$

```

パッケージングされた完全なイメージ update.img をボードに書き込みます。



## 23.5 修正結果の検証

イメージを再書き込みした後、デフォルトのデスクトップ背景は以前に変更した画像のままです。



## 第 24 章 rootfs.img イメージ内のファイルを修正

時には、rootfs の内容を変更する必要がある場合がありますが、ROOT 本的なファイルシステムを一から構築するのは避けたい場合があります。この場合、が提供するイメージを直接使用して、rootfs パーティション内のファイルを直接変更することができます。

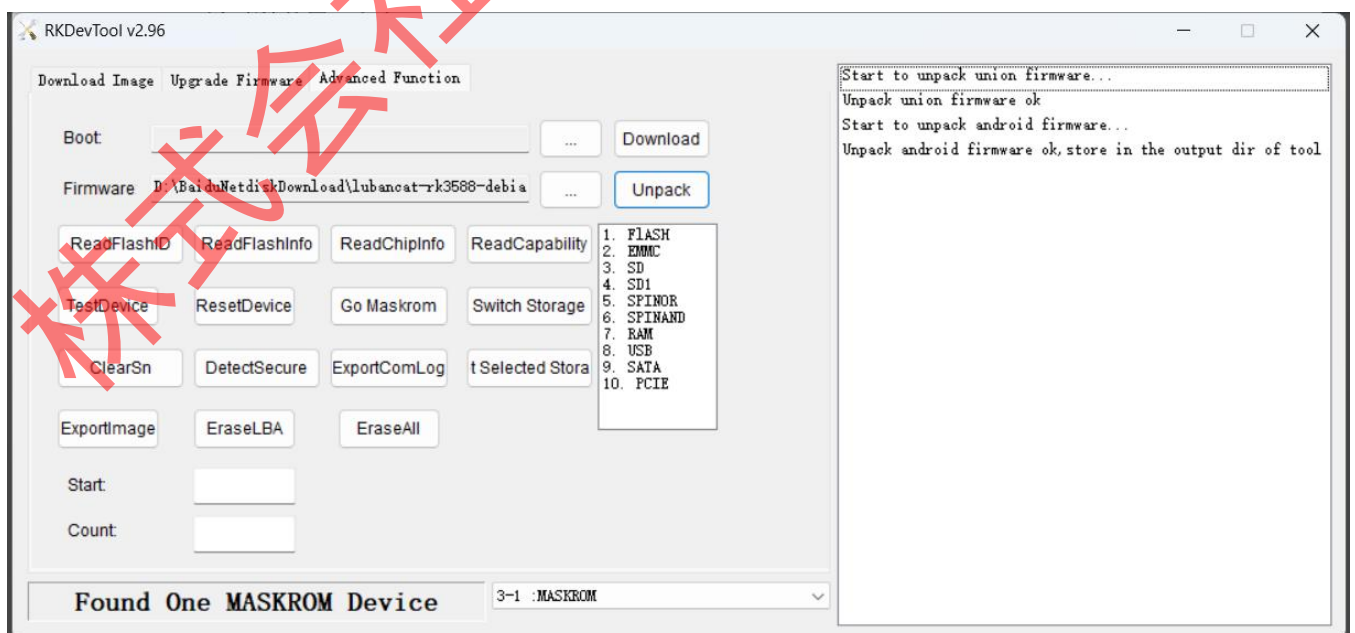
この方法は、変更量が少なく、ROOT 本的なファイルシステムのファイル構造に精通している開発者のにみ適しています。

ここでは、RK3588-LUBANCAT4-N\_DEBIAN\_BUSTER\_DESKTOP\_20220928\_Update イメージを例に説明します。具体的な手順は以下の通りです。

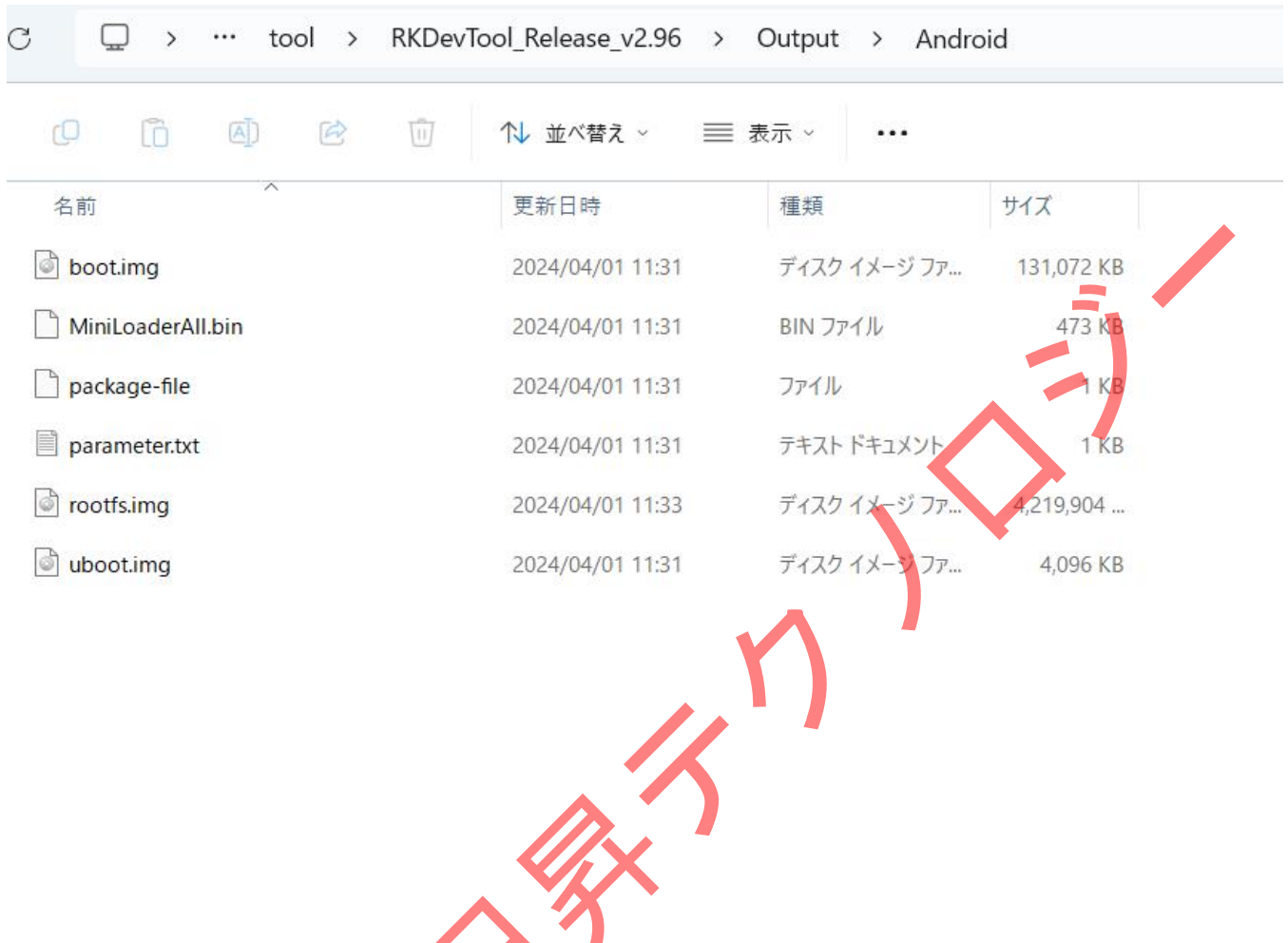
注意：汎用イメージは Linux で、Linux\_Pack\_Firmware ツールを使用してパッケージング、解凍する必要があります。そうでないと、パッケージされたイメージは正常に使用できません。専用イメージは Windows と Linux の両方で正常にパッケージング、解凍できます。

### 24.1 RKDevTool での解凍、修正、パッケージング (Windows)

1. RKDevTool を使用して update.img の完全なイメージを解凍し、単独の ROOT ファイルシステムイメージ rootfs.img を取得します。



2. 解凍された rootfs.img イメージは RKDevTool\_Release\_v2.92¥Output¥Android¥Image に保存されます。

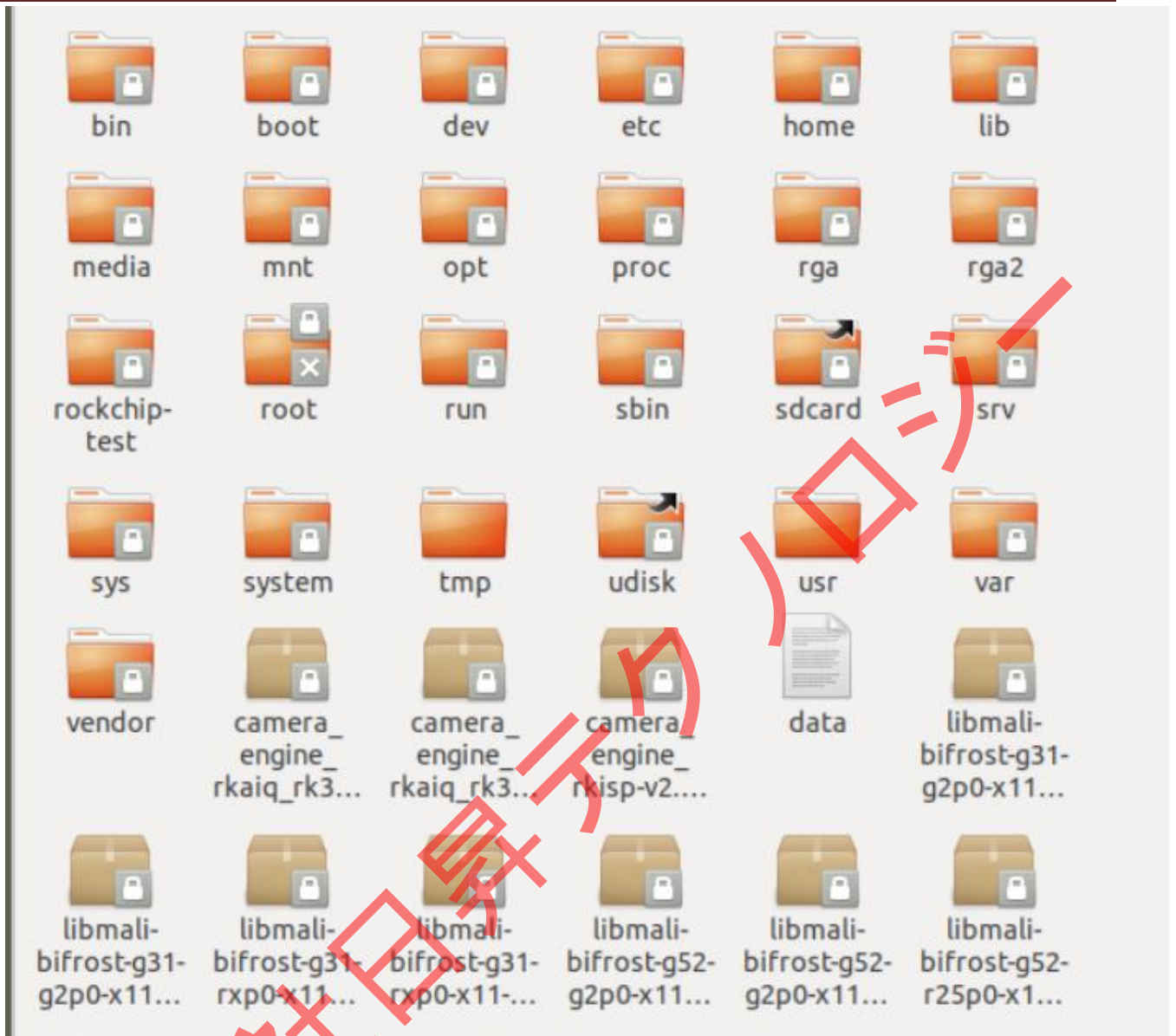


3. rootfs.img を Linux PC または仮想マシンにコピーし、イメージをマウントします。

1. # rootfs.img をマウントするためのディレクトリを作成
2. mkdir rootfs
- 3.
4. # マウント
5. sudo mount rootfs.img rootfs/
- 6.
7. # マウントが成功したか確認
8. df -h

```
csun@ubuntu:~$ sudo mkdir rootfs
csun@ubuntu:~$ sudo mount rootfs.img rootfs/
csun@ubuntu:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            1.9G   0    1.9G   0% /dev
tmpfs           388M  1.9M  386M   1% /run
/dev/sda5       20G   16G   2.7G  86% /
tmpfs           1.9G   0    1.9G   0% /dev/shm
tmpfs           5.0M   4.0K  5.0M   1% /run/lock
tmpfs           1.9G   0    1.9G   0% /sys/fs/cgroup
/dev/loop0      347M  347M   0 100% /snap/gnome-3-38-2004/119
/dev/loop1      128K  128K   0 100% /snap/bare/5
/dev/loop2       64M   64M   0 100% /snap/core20/1828
/dev/loop3       50M   50M   0 100% /snap/snapd/18357
/dev/loop4       46M   46M   0 100% /snap/snap-store/638
/dev/loop5       92M   92M   0 100% /snap/gtk-common-themes/1535
/dev/sda1        511M  4.0K  511M   1% /boot/efi
tmpfs           388M   84K  388M   1% /run/user/1000
/dev/loop6      3.9G  3.9G   0 100% /home/csun/rootfs
```

- マウントディレクトリに入ってファイルを編集します。



ここでは、etc/hostname ファイルを編集し、もとの「lubancat」を「embedfire」に変更します。

```

csun@ubuntu:~/rootfs$ cat etc/hostname
lubancat
csun@ubuntu:~/rootfs$ sudo vi etc/hostname
csun@ubuntu:~/rootfs$ cat etc/hostname
csun

csun@ubuntu:~/rootfs$ cd ..
csun@ubuntu:~$ umount rootfs
umount: /home/csun/rootfs: umount failed: Operation not permitted.
csun@ubuntu:~$ sudo umount rootfs
csun@ubuntu:~$
    
```

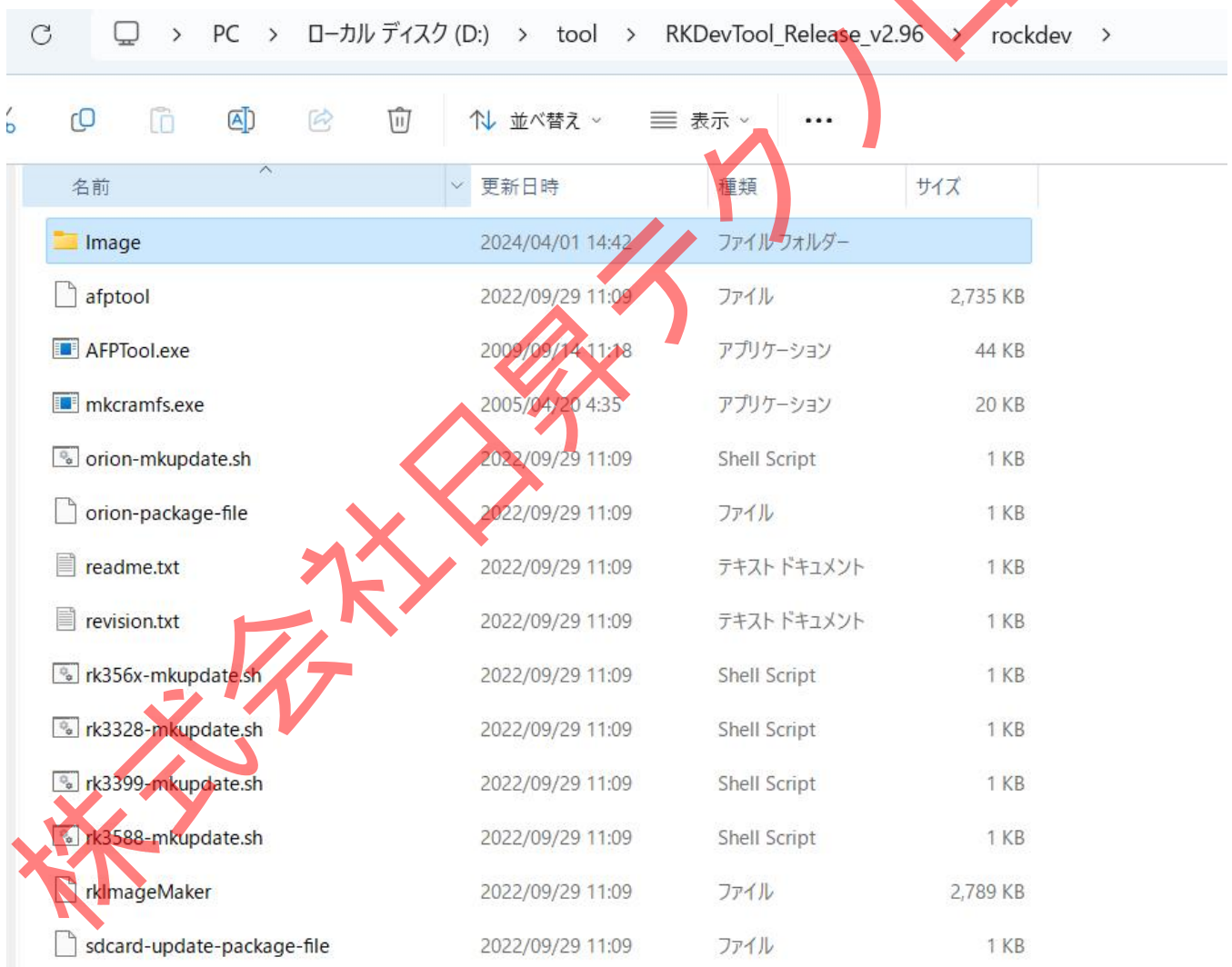
また、必要に応じて他の設定ファイルを編集したり、新しいファイルを作成したり、ファイルを削除することもできます。

編集が完了したら、マウントを解除します。

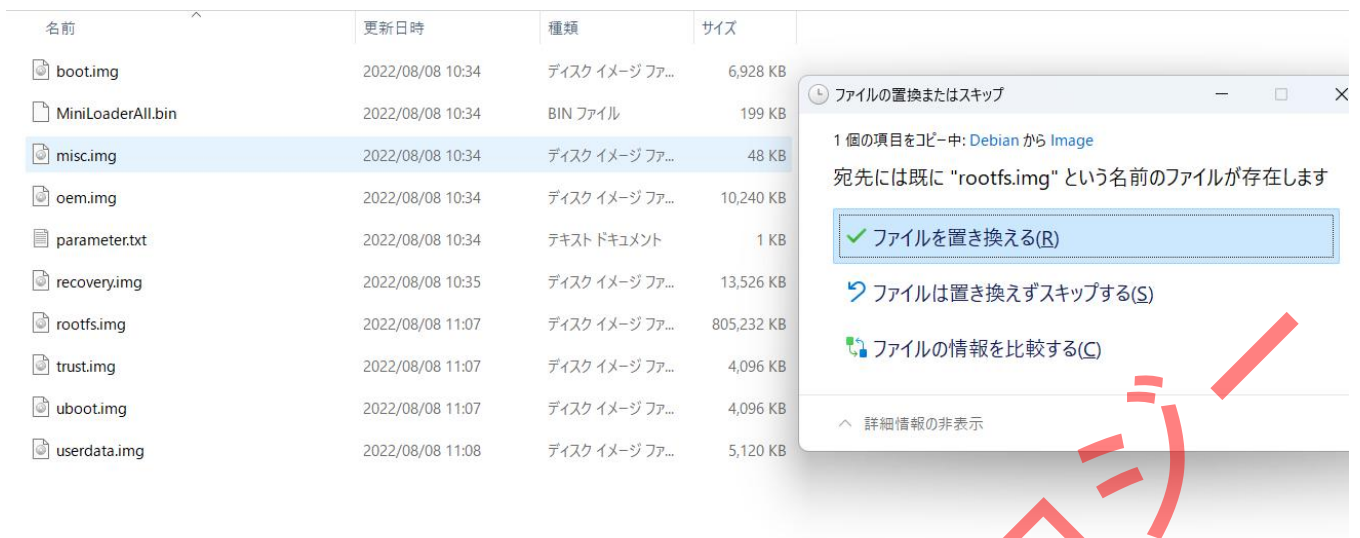
```
sudo umount rootfs/
```

1. RKDevTool\_Release\_v2.92 のパッキングツールを使用して、完全なイメージを再パッケージングします。

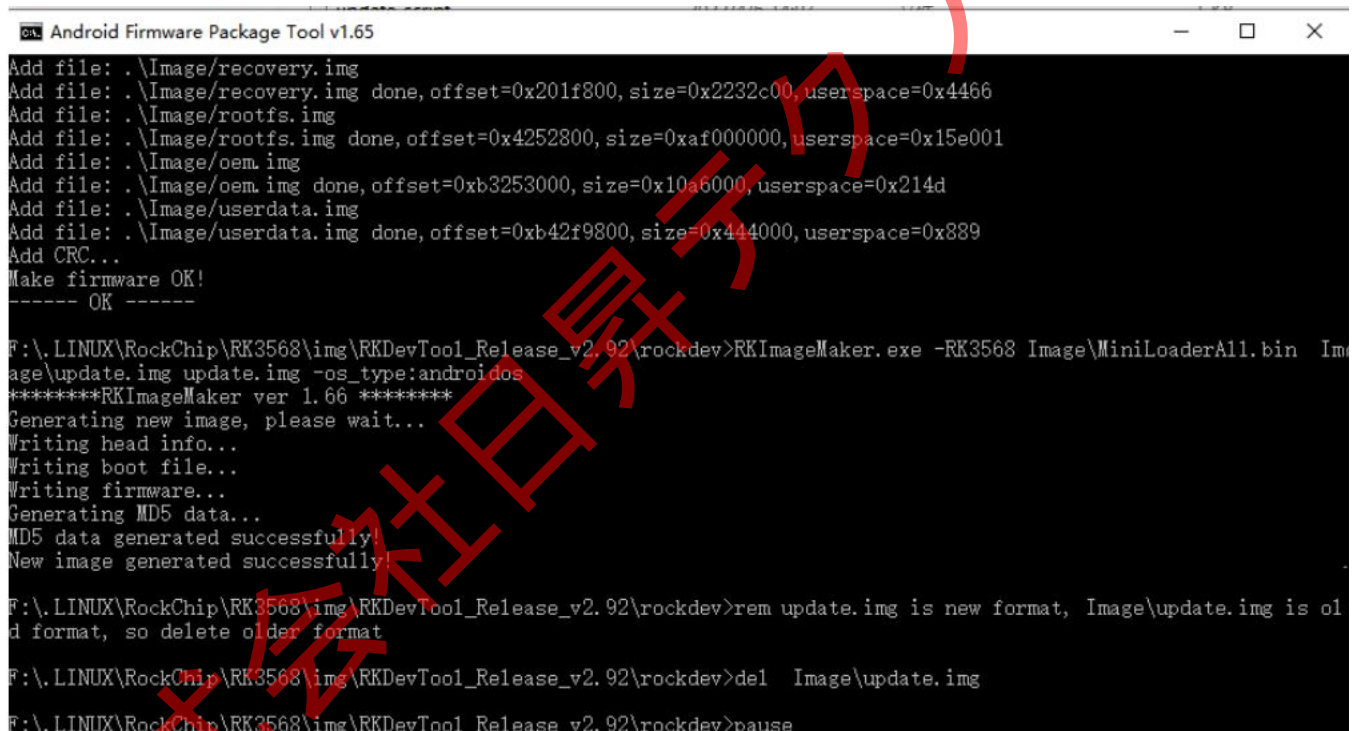
RKDevTool\_Release\_v2.92¥rockdev を開き、以前に解凍した Output¥Android 下の全てのファイルを rockdev ディレクトリにコピーします。



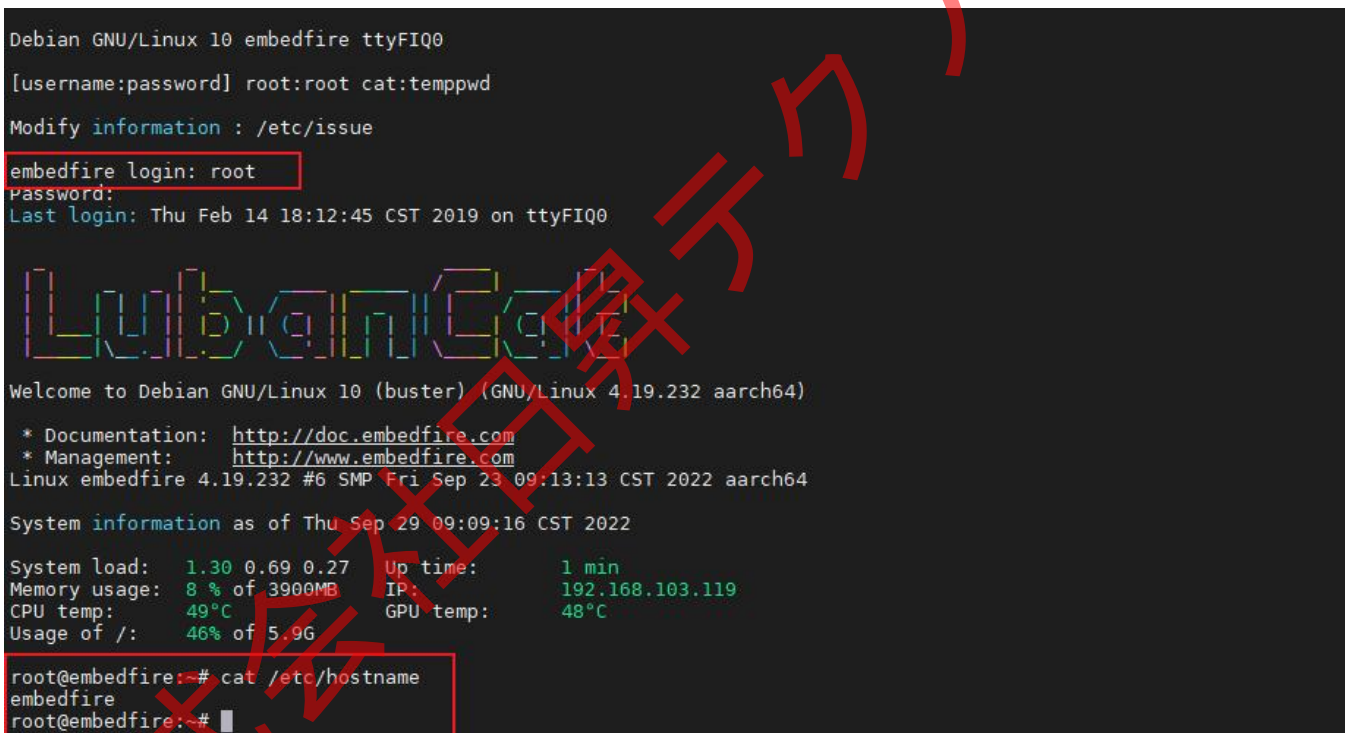
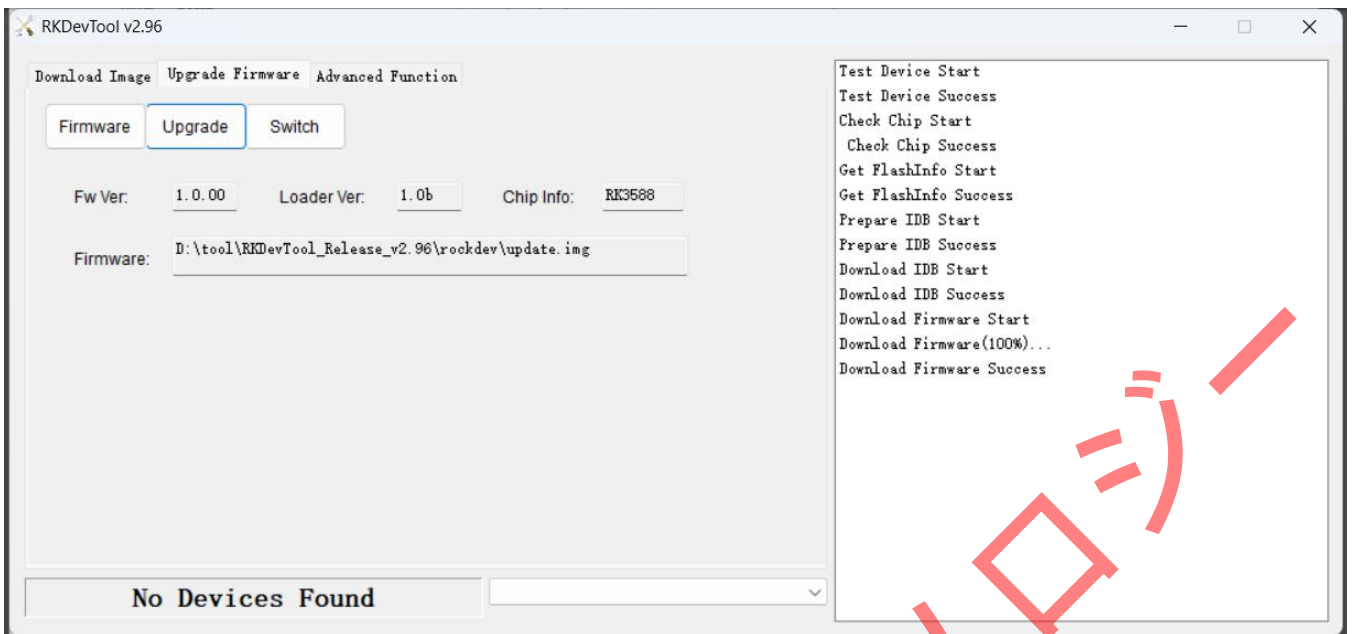
次に、編集済みの rootfs.img ファイルで現在の rockdev¥Image ディレクトリ下の rootfs.img ファイルを置き換えます。



rk358x-mkupdate.bat をダブルクリックしてイメージのパッキングを開始します。



6. パッケージングしたイメージを書き込みます。



## 24.2 Linux\_Pack\_Firmware で解凍、編集、パッケージング(Linux)

1. ウェブからダウンロードしたイメージを.img形式で解凍し、update.imgにリネームします。SDKで生成されたupdate.imgイメージを直接使用することもできます。

解凍したupdate.imgイメージをLinux\_Pack\_Firmware/rockdevディレクトリにコピーします。

```

hh@ubuntu:~$ ls
Desktop          Pictures
Documents       Public
Downloads       RK3568-LUBANCAT2-N_UBUNTU20.04_DESKTOP_20220921_Update.img
examples.desktop Templates
Linux_Pack_Firmware.zip Videos
Music
>_
hh@ubuntu:~$ unzip Linux_Pack_Firmware.zip
Archive: Linux_Pack_Firmware.zip
  creating: Linux_Pack_Firmware/
  creating: Linux_Pack_Firmware/rockdev/
  inflating: Linux_Pack_Firmware/rockdev/afptool
  inflating: Linux_Pack_Firmware/rockdev/orion-mkupdate.sh
  inflating: Linux_Pack_Firmware/rockdev/orion-package-file
  inflating: Linux_Pack_Firmware/rockdev/readme.txt
  inflating: Linux_Pack_Firmware/rockdev/revision.txt
  inflating: Linux_Pack_Firmware/rockdev/rk3328-mkupdate.sh
  inflating: Linux_Pack_Firmware/rockdev/rk3399-mkupdate.sh
  inflating: Linux_Pack_Firmware/rockdev/rk356x-mkupdate.sh
  inflating: Linux_Pack_Firmware/rockdev/rk3588-mkupdate.sh
  inflating: Linux_Pack_Firmware/rockdev/rkImageMaker
  inflating: Linux_Pack_Firmware/rockdev/sdcard-update-package-file
  inflating: Linux_Pack_Firmware/rockdev/unpack.sh
hh@ubuntu:~$ mv RK3568-LUBANCAT2-N_UBUNTU20.04_DESKTOP_20220921_Update.img Linux_Pack_Firmware/rockdev/update.img
hh@ubuntu:~$
  
```

2. unpack.sh スクリプトを実行して解凍します。

```
1 unpack.sh
```

```

hh@ubuntu:~/Linux_Pack_Firmware$ cd rockdev/
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$ ./unpack.sh
start to unpack update.img...
*****rkImageMaker ver 2.0*****
Unpacking image, please wait...
Exporting boot.bin
Exporting firmware.img
Unpacking image success.
Android Firmware Package Tool v2.0
Check file... OK
----- UNPACK -----
package-file      offset=0x800      size=0xA0
Image/MiniLoaderAll.bin offset=0x1000    size=0x719C0
Image/parameter.txt  offset=0x73000  size=0x15D
Image/uboot.img      offset=0x73800  size=0x400000
Image/boot.img       offset=0x473800 size=0x8000000
Image/rootfs.img     offset=0x8473800 size=0xB9159000
Unpack firmware OK!
----- OK -----
Unpacking update.img OK.
Press any key to quit:
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$ [A
  
```

解凍後のファイルは output ディレクトリに保存され、RKDevTool で解凍した内容と同じです。



```
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$ cd output/  
hh@ubuntu:~/Linux_Pack_Firmware/rockdev/output$ tree  
├── Image  
│   ├── boot.img  
│   ├── MiniLoaderAll.bin  
│   ├── parameter.txt  
│   ├── rootfs.img  
│   └── uboot.img  
└── package-file  
  
1 directory, 6 files  
hh@ubuntu:~/Linux_Pack_Firmware/rockdev/output$
```

output ディレクトリには、ファイルとフォルダが1つずつあります。

- package-file : パーティションとパーティションイメージ名の対応関係
- Image : firmware.img の解凍後の内容、つまり解凍後のパーティションイメージです。

Image ディレクトリに入り、異なる OS イメージに応じて、以下の3種類のファイルがあります。

- パーティション表 : parameter.txt
- loader ファイル : MiniLoaderAll.bin
- パーティションイメージ : boot.img、uboot.img など、img で終わるパーティションイメージファイル

3. 生成した rootfs.img をマウントし、必要に応じて内容を編集します。

```
1 # ルートファイルシステムを/mnt にマウント  
2 sudo mount Image/rootfs.img /mnt  
3  
4 # 編集が完了したらアンマウント  
5 sudo umount /mnt
```

#### 4. パッケージング

前の解凍プロセスで得た output フォルダの内容を基にします。

まず、rockdev/output ディレクトリ下の内容を rockdev ディレクトリにコピーし、以前に解凍用に使用した update.img イメージを削除します。

```
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$ ls
afptool          readme.txt      rk356x-mkupdate.sh  unpack.sh
orion-mkupdate.sh  revision.txt   rk3588-mkupdate.sh  update.img
orion-package-file rk3328-mkupdate.sh rkImageMaker
output          rk3399-mkupdate.sh sdcard-update-package-file
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$ ls output/
Image  package-file
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$ cp -r output/* ./
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$ ls
afptool          output          rk3328-mkupdate.sh  rkImageMaker
Image           package-file   rk3399-mkupdate.sh  sdcard-update-package-file
orion-mkupdate.sh  readme.txt    rk356x-mkupdate.sh  unpack.sh
orion-package-file  revision.txt  rk3588-mkupdate.sh  update.img
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$ rm update.img
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$ ls
afptool          output          rk3328-mkupdate.sh  rkImageMaker
Image           package-file   rk3399-mkupdate.sh  sdcard-update-package-file
orion-mkupdate.sh  readme.txt    rk356x-mkupdate.sh  unpack.sh
orion-package-file  revision.txt  rk3588-mkupdate.sh
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$
```

特定のパーティションイメージファイルを変更した場合、パッキング時にイメージファイル名が正しく、package-file ドキュメント内の命名とバスと一致していることを確認する必要があります。

LubanCat-1、2、Zero はすべて rk358x メインチップを使用しており、対応するパッキングスクリプトを実行します。

```

hh@ubuntu:~/Linux_Pack_Firmware/rockdev$ ./rk356x-mkupdate.sh
start to make update.img...
Android Firmware Package Tool v2.0
----- PACKAGE -----
Add file: ./package-file
package-file,Add file: ./package-file done,offset=0x800,size=0xa0,userspace=0x1
Add file: ./Image/MiniLoaderAll.bin
bootloader,Add file: ./Image/MiniLoaderAll.bin done,offset=0x1000,size=0x719c0,userspace
=0xe4
Add file: ./Image/parameter.txt
parameter,Add file: ./Image/parameter.txt done,offset=0x73000,size=0x169,userspace=0x1
Add file: ./Image/uboot.img
uboot,Add file: ./Image/uboot.img done,offset=0x73800,size=0x400000,userspace=0x800
Add file: ./Image/boot.img
boot,Add file: ./Image/boot.img done,offset=0x473800,size=0x8000000,userspace=0x10000
Add file: ./Image/rootfs.img
rootfs,Add file: ./Image/rootfs.img done,offset=0x8473800,size=0xb9159000,userspace=0x17
22b2
Add CRC...
Make firmware OK!
----- OK -----
*****rkImageMaker ver 2.0*****
Generating new image, please wait...
Writing head info...
Writing boot file...
Writing firmware...
Generating MD5 data...
MD5 data generated successfully!
New image generated successfully!
Making ./Image/update.img OK.
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$
  
```

パッケージングが完了した後、現在のフォルダにupdate.img ファイルが生成されます。これは、パッケージング後の完全なイメージファイルです。

```

total intu:~/Linux_Pack_Firmware/rockdev$ ls -al
total 3174100
drwxrwxrwx 4 hh hh 4096 9月 28 19:55
drwxrwxrwx 3 hh hh 4096 9月 28 19:09
-rwxr--r-- 1 hh hh 2799704 9月 28 19:09 afptool
drwxr-xr-x 2 hh hh 4096 9月 28 19:53 Image
-rwxr--r-- 1 hh hh 545 9月 28 19:09 orion-mkupdate.sh
-rwxr--r-- 1 hh hh 651 9月 28 19:09 orion-package-file
drwxrwxr-x 3 hh hh 4096 9月 28 19:30 output
-rw-rw-r-- 1 hh hh 160 9月 28 19:49 package-file
-rwxr--r-- 1 hh hh 407 9月 28 19:09 readme.txt
-rwxr--r-- 1 hh hh 157 9月 28 19:09 revision.txt
-rwxr--r-- 1 hh hh 537 9月 28 19:09 rk3328-mkupdate.sh
-rwxr--r-- 1 hh hh 545 9月 28 19:09 rk3399-mkupdate.sh
-rwxr--r-- 1 hh hh 545 9月 28 19:09 rk356x-mkupdate.sh
-rwxr--r-- 1 hh hh 545 9月 28 19:09 rk3588-mkupdate.sh
-rwxr--r-- 1 hh hh 2855472 9月 28 19:09 rkImageMaker
-rwxr--r-- 1 hh hh 502 9月 28 19:09 sdcard-update-package-file
-rwxr--r-- 1 hh hh 470 9月 28 19:09 unpack.sh
-rw-rw-r-- 1 hh hh 3244548682 9月 28 19:56 update.img
hh@ubuntu:~/Linux_Pack_Firmware/rockdev$
  
```

書き込みツールを使用して、パッケージングされたイメージをボードに書き込み、正常に起動できるようにします。