

マニュアル

産業用 PC NP/AP/NPL/APL/AS/EC/ECL 4C シリーズ用
Debian12.1 Linux ディストリビューション
『Algonomix10』 について

目 次

はじめに

- 1) お願いと注意 1
- 2) 保証について 1

第 1 章 概要

- 1-1 Algonomix10 とは 1-1
- 1-2 Linux の仕組み 1-2

第 2 章 システム構成

- 2-1 Algonomix10 パッケージについて 2-1
 - 2-1-1 パッケージについて 2-3
- 2-2 Algonomix10 のディレクトリ構造 2-8
- 2-3 Algonomix10 設定ツール「ASD Config」について 2-12
 - 2-3-1 ASD Application Config について 2-14
 - 2-3-2 ASD Date Config について 2-15
 - 2-3-3 ASD Misc Config について 2-18
 - 2-3-4 ASD Battery Config について 2-20
 - 2-3-5 ASD WatchdogTimer Config について 2-24
 - 2-3-6 ASD RAS Config について 2-26
 - 2-3-7 ASD Shutdown Menu について 2-28
- 2-4 有線 LAN の設定について 2-29
- 2-5 無線 LAN の設定について 2-32
- 2-6 sysfs ファイルシステム 2-35
 - 2-6-1 基板情報 2-35
 - 2-6-2 温度センサ 2-36
 - 2-6-3 LCD バックライト 2-36
- 2-7 データの保護について 2-37
 - 2-7-1 データ保護の必要性と方法 2-37

第3章 開発環境

3-1 クロス開発環境	3-1
-------------------	-----

第4章 産業用 PC NP/AP/NPL/APL/AS/EC/ECL 4C シリーズについて

4-1 汎用入出力	4-11
4-1-1 汎用入出力について	4-11
4-1-2 汎用入出力デバイスドライバについて	4-11
4-2 シリアルポート	4-15
4-2-1 シリアルポートについて	4-15
4-3 ネットワークポート	4-20
4-3-1 ネットワークポートについて	4-20
4-3-2 ネットワークソケット用システムコールについて	4-20
4-4 RAS 機能	4-25
4-4-1 汎用入力 IN0 リセットについて	4-25
4-4-2 汎用入力 IN0 リセットデバイスドライバについて	4-25
4-4-3 汎用入力 IN0 リセットサンプル	4-26
4-4-4 汎用入力 IN1 割込みについて	4-28
4-4-5 汎用入力 IN1 割込みデバイスドライバについて	4-28
4-4-6 汎用入力 IN1 割込みサンプル	4-29
4-4-7 ハードウェア・ウォッチドッグタイマ機能について	4-32
4-4-8 ハードウェア・ウォッチドッグタイマデバイスについて	4-34
4-4-9 ハードウェア・ウォッチドッグタイマサンプル	4-36
4-4-10 Wake On RTC 機能について	4-43
4-4-11 S. M. A. R. T. 監視機能について	4-44
4-4-12 S. M. A. R. T. 機能サンプルプログラム	4-45
4-4-13 バックアップ RAM 機能について	4-49
4-4-14 バックアップ RAM 機能デバイスドライバについて	4-49
4-4-15 バックアップ RAM 制御サンプル	4-50
4-4-16 タイマ割込み機能について	4-52
4-4-17 タイマ割込み機能デバイスについて	4-52
4-4-18 タイマ割込み機能サンプルプログラム	4-54
4-5 UPS 機能	4-59

4-5-1 UPS 機能について	4-59
4-5-2 UPS 機能サンプルプログラム	4-59
4-6 ストレージデバイスについて	4-62
4-6-1 外部ストレージデバイスの使用方法	4-62
4-6-2 ストレージデバイス名の割り振りについて	4-63
4-6-3 外部ストレージデバイスの起動時マウントについて	4-64
4-7 LTE について	4-65

第5章 システムリカバリ

5-1 リカバリ DVD について	5-1
5-1-1 リカバリ準備	5-3
5-1-2 出荷イメージリカバリ USB 起動	5-6
5-1-3 リカバリ USB 起動	5-9
5-1-4 リカバリ作業	5-10
5-2 システムの復旧（バックアップデータ）	5-11
5-3 システムのバックアップ	5-16

付録

A-1 参考文献	5-1
----------------	-----

はじめに

この度は、アルゴシステム製品をお買い上げいただきありがとうございます。

弊社製品を安全かつ正しく使用していただく為に、お使いになる前に本書をお読みいただき、十分に理解していただくようお願い申し上げます。

1) お願いと注意

本書では、産業用 PC NP/AP/NPL/APL/AS/EC/ECL 4C シリーズ（以降 **4C シリーズ**）用 Linux ディストリビューション（以降 **Algonomix10**）に特化した部分について説明します（※）。Algonomix10 は、「Debian 12.1」という Linux ディストリビューションをベースにした、4C シリーズ用の Linux ディストリビューションです。Algonomix10 は、「Debian 12.1」に 4C シリーズ用の独自の I/O ドライバを組込んだものです。

一般的な Linux についての詳細は省略させていただきます。Linux に関する資料および文献は、現在インターネット上や書籍など多数ございます。これらの書籍等と併せて本書をお読みください。

また、本製品用アプリケーションの開発におきましては「Algonomix10 開発環境ユーザズマニュアル」をお読みください。

2) 保証について

Algonomix10 の動作は出荷パッケージのバージョンでのみ動作確認しております。Algonomix10 はお客様でソースの改変、ライブラリの追加と変更、プログラム設定の変更等を行うことができますが、これらの変更を行われた場合は動作保証することができません。

※) 本書の内容は以下の機種に対応しています。

- ・ NP4C-1xxAN ・ NP4C-1xxAS
- ・ AP4C-1xxAN
- ・ EC4C-100AT

- ・ NPL4C-1xxBN/ NPL4C-1xxBS
- ・ APL4C-1xxBN
- ・ AS4C-101CN/ AS4C-101CS

- ・ NPL4C-070BN/ NPL4C-070BS
- ・ ECL4C-010BT
- ・ NPL4C-070CN/ NPL4C-070CS
- ・ ECL4C-010CT

- ・ NPL4C-10HES
- ・ NPL4C-10HDS

第 1 章 概要

本章では、Algonomix10 の具体的な内容を説明する前に、Algonomix10 の概要について説明します。

1-1 Algonomix10 とは

「Linux」とは、Linux カーネルのみを指す言葉です。しかし、Linux カーネルのみでは、オペレーティングシステム（以下 OS）としての役割を果たすことができません。OS として使うには、Linux カーネルのほかに、以下のような各種ソフトウェアパッケージと併せて使用する必要があります。

- シェル (bash、ash、csh、tcsh、zsh、pdksh、……)
- util-linux (init、getty、login、reset、fdisk、……)
- procs (ps、pstree、top、……)
- GNU coreutils (ls、cat、mkdir、rmdir、cut、chmod、……)
- GNU grep、find、diff
- GNU libc
- 各種基本ライブラリ (ncurses、GDBM、zlib……)
- X Window System

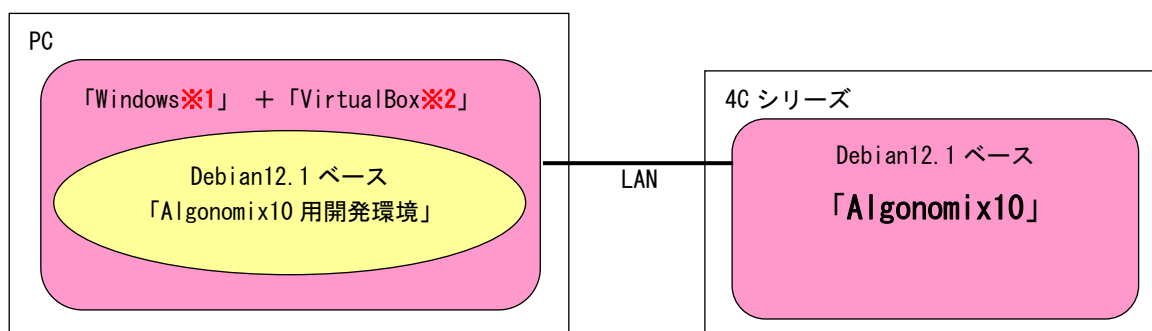
Linux カーネルといくつかの必要なソフトウェアパッケージをまとめて、OS として使えるようにしたものを Linux ディストリビューションといいます。

最初に述べましたとおり、「Linux」という言葉は、本来カーネルを指す言葉です。そのため、「カーネルとしての Linux」と「OS としての Linux」を厳密には区別する必要がありますが、本書では「Linux」とは「OS としての Linux」を指す言葉として使用します。

Algonomix10 は、「Debian 12.1」という Linux ディストリビューションをベースにした、4C シリーズ用の Linux ディストリビューションです。Algonomix10 は、「Debian 12.1」に 4C シリーズ用の独自の I/O ドライバを組込んだものです。

パソコン上に Algonomix10 用の開発環境をインストールすることで、Algonomix10 用のソフトウェアを開発することができます。

Algonomix10 用開発環境イメージを図 1-1-1 に示します。



※注 1：Windows は米国 Microsoft Corporation の米国およびその他の国における登録商標です。

※注 2：VirtualBox は、米国 Oracle Corporation, Inc. の米国およびその他の国における商標または登録商標です。

図 1-1-1. Algonomix10 の開発環境

開発環境の詳細については、別紙『Algonomix10 開発環境ユーザズマニュアル』を参照してください。

1-2 Linux の仕組み

Linux のソフトウェア構成を図 1-2-1 に示します。

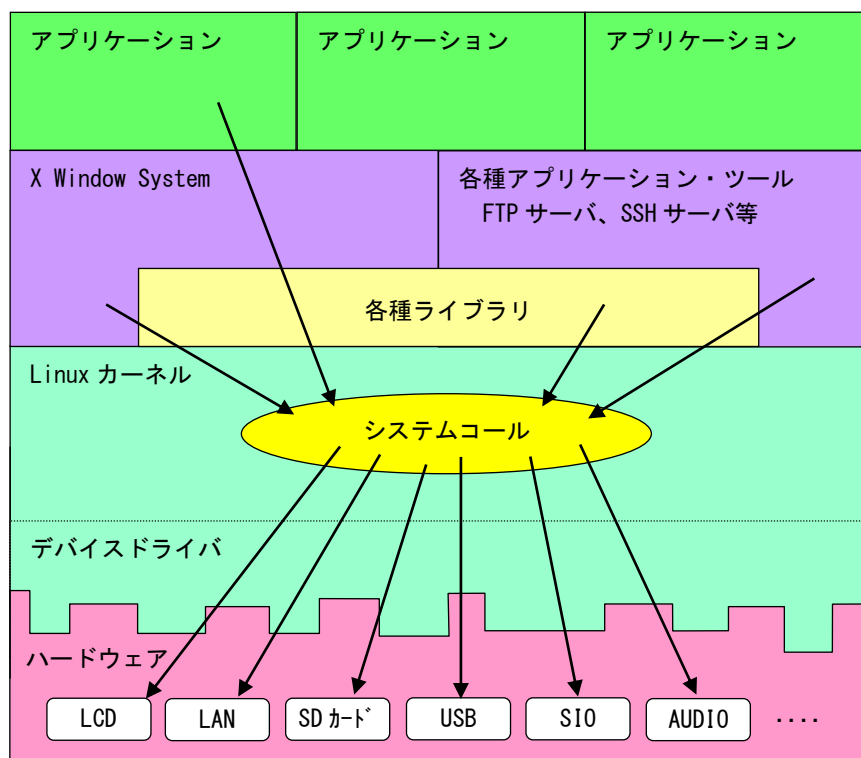


図 1-2-1. Linux ソフトウェア構成図

OS として重要な役割の一つに、ハードウェアアクセスの複雑さを隠し、統一されたプログラミングインターフェース（システムコールや API と呼ばれる）をアプリケーションに提供するというものがあります。Linux ではハードウェアを制御する為にドライバに関連付けられた「デバイスファイル」を読み書きすることで制御します。これは UNIX 系 OS の大きな特徴であり、ファイルを扱う感覚でハードウェアを制御することができます。Linux の代表的なシステムコールとして、open、close、read、write 等があります。これらのシステムコールは特別な呼び方をしてはいるわけではなく、関数の呼び出しと同じように呼び出すことができます。

もう一つ OS の重要な役割として、CPU 時間、メモリ、ネットワーク等のリソースをプログラムやプロセス、スレッドに分配するというものもあります。これは Linux カーネルが処理しており、アプリケーション作成時に特に意識する必要はありません。図 1-2-1 にあるような X Window System や、SSH サーバや FTP サーバもプロセスの一つです。CPU 時間やメモリなどのリソースには限りがある為、複数のプロセスを同時に実行すると、それぞれのパフォーマンスは落ちます。そのため、必要最低限のプロセスで実行効率のよいプログラムを作成する必要があります。

第 2 章 システム構成

2-1 Algonomix10 パッケージについて

Algonomix10 であらかじめインストールされているパッケージを表 2-1-1 に示します。ただし Debian 12.1 の基本パッケージとしてインストールされているパッケージは除きます。


表 2-1-1. プリインストールパッケージ一覧

パッケージ名	内容
linux-image-X.XX.X-XX-asdatom	Linux Kernel 本体
Xfce	デスクトップ環境
ssh	telnet よりセキュリティの高いシェルクライアントおよびサーバ
gdbserver	GNU デバッガ
aptitude	APT 用パッケージ管理ツール
apache2	Web サーバ
chromium-browser	オープンソースのウェブブラウザ
xinput-calibrator	X.org 向けの タッチスクリーンキャリブレーションプログラム

Algonomix10 では、Debian 12.1 の基本パッケージと本書に表記したパッケージが入った環境でのみ動作を確認しています。そのため、パッケージの追加に制限をかけています。

詳細に関しては、「2-1-1 パッケージについて」を参照してください。

Algonomix10 でインストール済みのパッケージ一覧を表示する為には下記手順を実行します。

- ① 画面下部中央の  をクリックするか、左上の「アプリケーション」→「ターミナルエミュレータ」をクリックしてください。図 2-1-1 のような、ターミナル画面が表示されます。

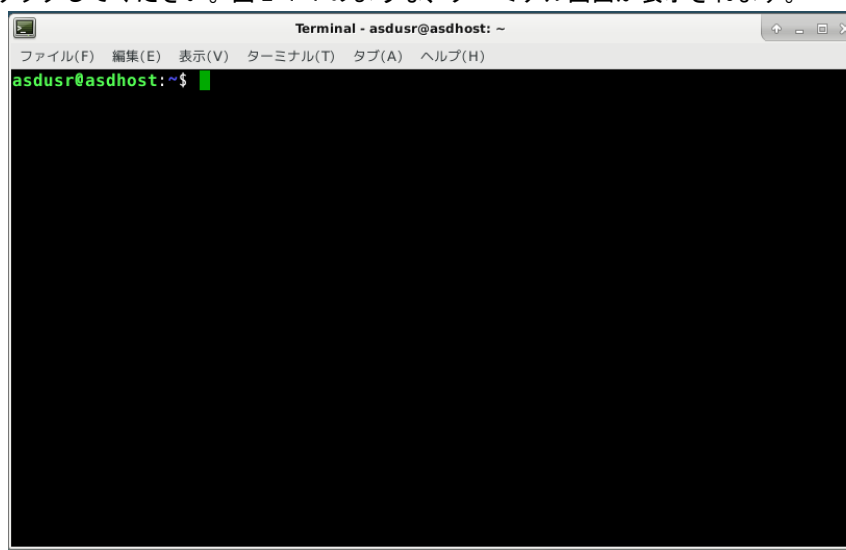


図 2-1-1. ターミナル画面

- ② 下記のコマンドを実行することで、現在インストールされているすべてのパッケージの名前が一覧で表示されます。

```
$ dpkg -l
```

または

```
$ dpkg --list
```

一覧の表示内容は次のような形式で表示されています。

```
+++-----  
ii adduser      3.113+nmu3ubuntu3 all add and remove users and groups
```

1 2 3 4 5

①: パッケージのインストール状況

1文字目: 要望 : (U) 不明 / (I) インストール / (R) 削除 / (P) 完全削除 / (H) 維持

2文字目: 状態 : (N) 無 / (I) インストール済 / (C) 設定 / (U) 展開 / (F) 設定失敗
(H) 半インストール / (W) トリガ待ち / (T) トリガ保留

3文字目: エラー : (空欄) 無 / (H) 維持 / (R) 要再インストール / X=両方 (状態, エラーの大文字=異常)

②: パッケージ名

③: パッケージのバージョンおよびリビジョン

④: パッケージが対応しているアーキテクチャ

⑤: パッケージの1行説明

2-1-1 パッケージについて

Algonomix10 では、Debian 12.1 の基本パッケージと本書に表記したパッケージが入った環境でのみ動作を確認しています。

そのため、インターネットからのパッケージの追加に制限をかけています。

以下にパッケージをさらに追加したい場合の制限の解除方法を示しますが、設定を変更してパッケージを追加された場合、動作の保証は致しかねますので、あらかじめご了承ください。

Algonomix10 では、インターネットへアクセスするリポジトリを無効にすることでパッケージのインストールに制限をかけています。

制限の解除方法としては、`/etc/apt/sources.list` を直接編集するか、「Synaptic パッケージマネージャ」を使って間接的に編集することが出来ます。ここでは、後者を紹介します。

また、Debian では、インストールした時点の基本パッケージを収納した DVD-ROM 3 枚組が用意されています。ネットワークリポジトリの最新状態にアップデートせずに、インストール時のパッケージが必要な場合は、弊社までお問い合わせください。

- ① [アプリケーション]→[設定]→[Synaptic パッケージマネージャ]をクリックします。

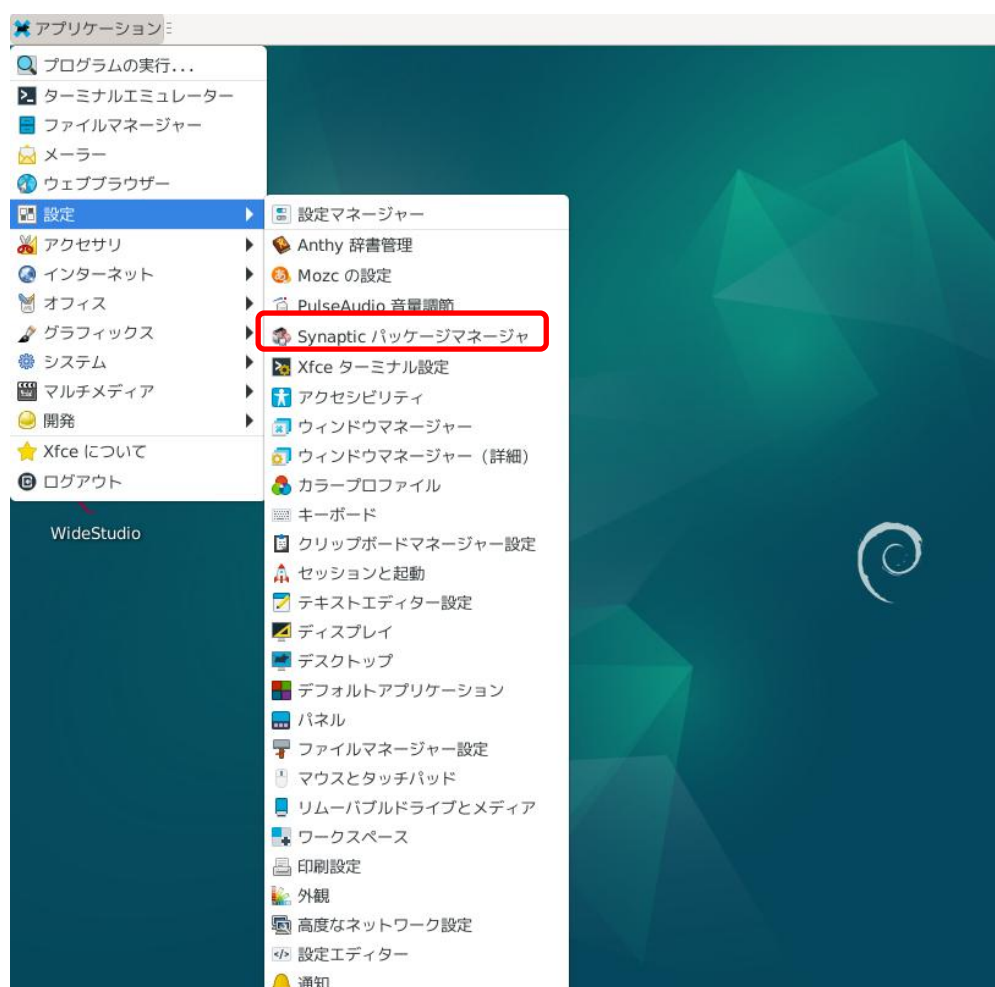


図 2-1-1-1. Synaptic パッケージマネージャの起動

- ② パッケージのインストール等を行うには管理者権限になる必要があります。図 2-1-1-2 のような、認証ウィンドウが表示されるので、管理者パスワードを入力して「認証する」をクリックしてください。

デフォルト管理者パスワード : rootroot

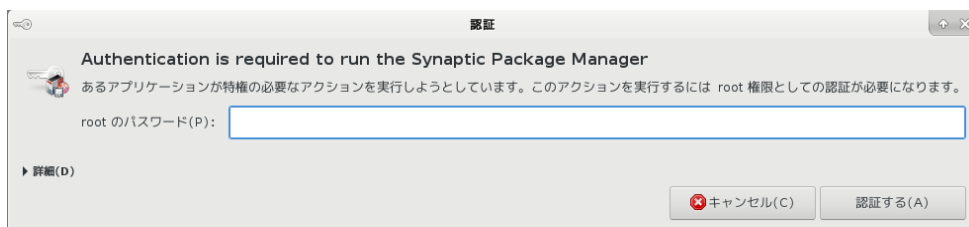


図 2-1-1-2. 管理者権限認証画面

- ③ 初回起動時は、図 2-1-1-3 のような、紹介画面が表示されます。「Close」をクリックしてください。

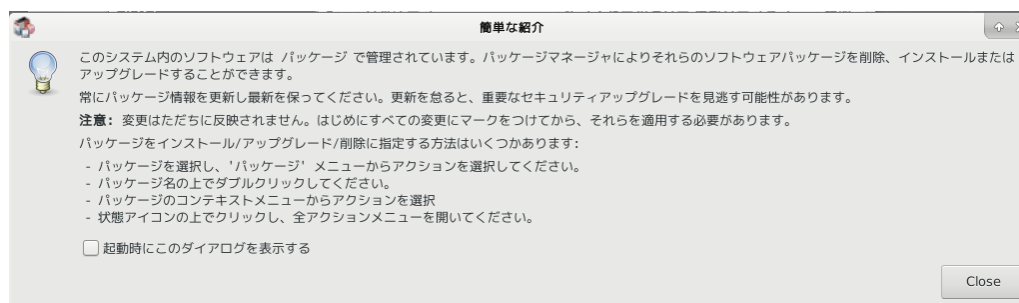


図 2-1-1-3. 紹介画面

④ Synaptic メイン画面を図 2-1-1-4 に示します。

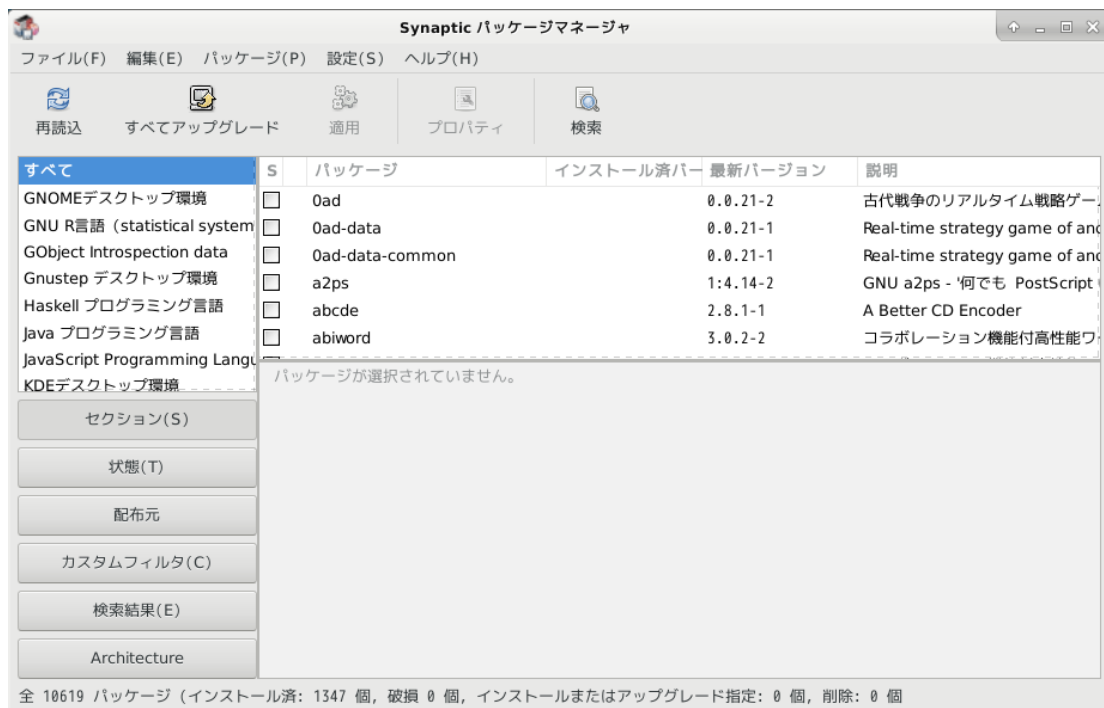


図 2-1-1-4. Synaptic パッケージマネージャメイン画面

⑤ 「設定」→「リポジトリ」をクリックしてください。

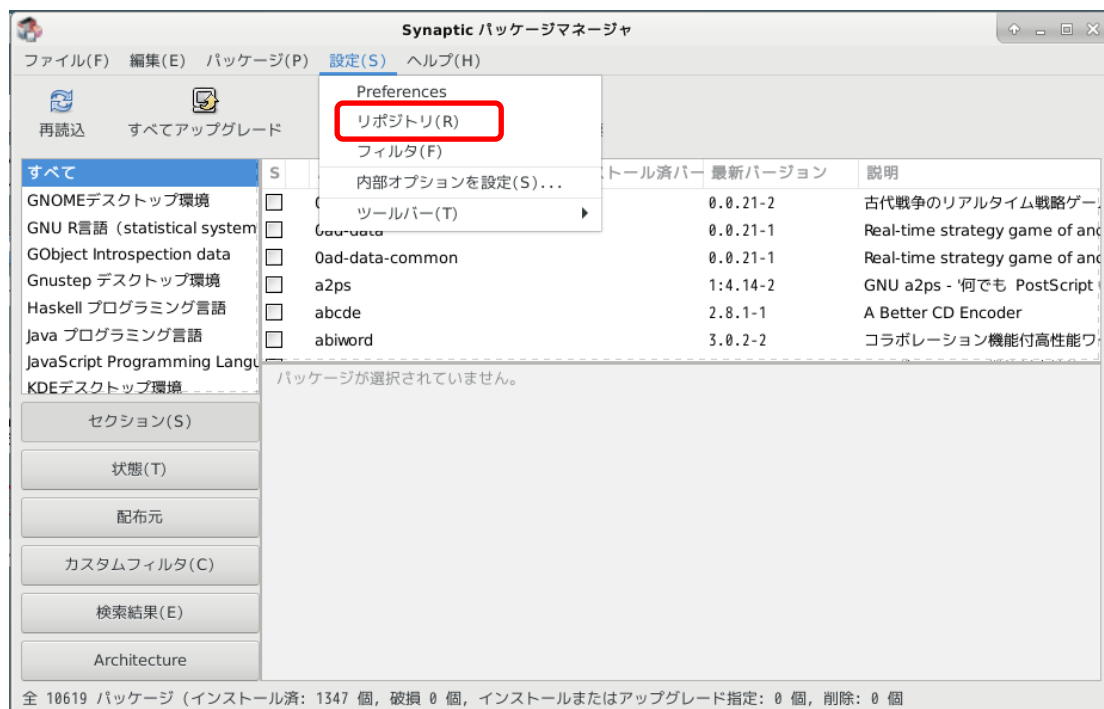


図 2-1-1-5. 設定メニュー

- ⑥ 図 2-1-1-6 のようなリポジトリ設定画面が表示されます。
「New」ボタンをクリックして、追加することも可能です。

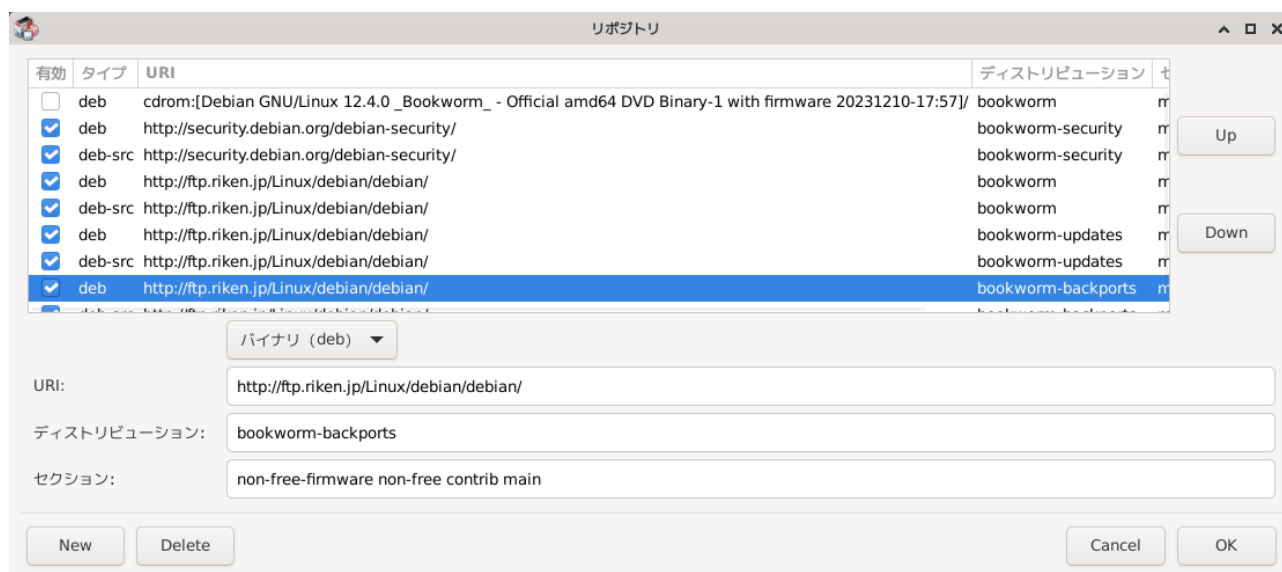


図 2-1-1-6. リポジトリ画面

- ⑦ 設定を更新して、「OK」をクリックすると、図 2-1-1-7 の画面が表示されます。
「再読み込み」をクリックして、変更を反映させてください。



図 2-1-1-7. リポジトリ変更確認画面

- ⑧ 再読み込まれると、図 2-1-1-8 のような、警告画面が表示されます。DVD-ROM リポジトリには、Release ファイルが存在しないため、下記のような警告が表示されます。下記のメッセージについては無視してください。「x」ボタンを押して、閉じてください。



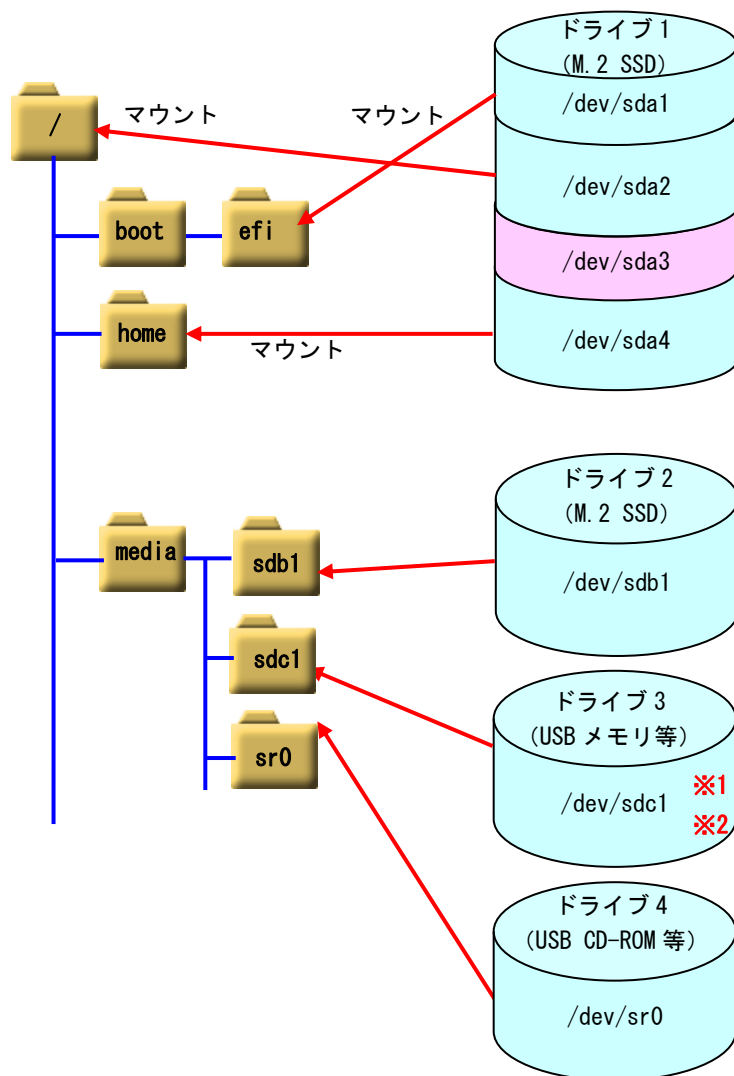
図 2-1-1-8. 警告画面

以上で、リポジトリの変更は完了します。

パッケージの追加については、「Synaptic パッケージマネージャ」を使用するか、「ターミナルエミュレータ」上で、「aptitude」または「apt-get」コマンドを使用してください。

2-2 Algonomix10 のディレクトリ構造

Linux ではルートファイルシステムと呼ばれるツリー構造のファイルシステムを採用しています。ルートディレクトリ (/) 以下に、ストレージ上に構築されたファイルシステムをマウントすることで、ルートファイルシステムとして利用できるようにしています。Algonomix10 は Debian ベースとなっている為、Debian のディレクトリ構造に従っています。



※注1：ストレージ機器をマウントするたびに sdb1、sdc1、sdd1 というようにマウントポジションが追加されていきます。M.2 サブストレージを接続していない場合は、USB 機器が /dev/sdb になります。

※注2：USB 機器は、認識順により sdc と sdd が入れ替わる可能性があります。

図 2-2-1. Debian のツリー構造

4C シリーズでは、M.2 SSD にルートファイルシステムが構築されています。表 2-2-1 のようにパーティションが切られ、それぞれのディレクトリにマウントされています。

4C シリーズのルートファイルシステム構成をリスト 2-2-1 に、ディレクトリ内容を表 2-2-2 に示します。

表 2-2-1. 4C シリーズの初期パーティション構成 (32GByte)

ドライブ名	サイズ	使用容量	空き容量	使用%	マウントポジション
/dev/sda1	536MByte	17MByte	520MByte	4%	/boot/efi
/dev/sda2	20GByte	6.0GByte	13GByte	32%	/
/dev/sda3	977MByte	0Byte	977MByte	0%	スワップ用予約
/dev/sda4※	8.2GByte	60MByte	7.7GByte	1%	/home

※注：パーティション 4 番目は採用する M.2SSD ストレージのサイズにより増加します。

リスト 2-2-1. ルートファイルシステムのディレクトリ構成

```

/
+--bin
+--boot    --+--efi
+--dev      --+--shm
+--etc
+--home     --+--asdusr
+--lib
+--lost+found
+--md5sum.txt
+--media
+--mnt
+--opt
+--proc
+--root
+--run      --+--Lock
+--sbin
+--srv
+--sys      --+--fs    --+--cgroup
+--tmp
+--usr      --+--bin
              +--games
              +--include
              +--lib
              +--local
              +--sbin
              +--share
              +--src
+--var      --+--cache
              +--lib
              +--log
              +--tmp

```


表 2-2-2. ルートファイルシステムのディレクトリ内容

ディレクトリ名	内容	tempfs
/	ルートディレクトリ	
/bin	システム管理者・一般ユーザ共に使用するコマンド群が格納されています。	
/boot	起動時に必要とする設定ファイル群とマッピングインストーラが配置されています。	
/dev	ハードウェアをコントロールする為のデバイスファイルが格納されています。基本的なデバイスの一覧を以下に示します。 ・ターミナル：/dev/tty* 例：tty0、tty1 キーボードとプリンタにより構成され、文字を入出力できます。 ・シリアルポート：/dev/ttyS* 例：ttyS0、ttyS1、ttyS2 シリアル通信することができます。 ・SCSI ディスク：/dev/sd* 例：sdb1、sdb2、sdc USB メモリ等はこのデバイスになります。sd の後ろにつく文字がディスクを特定し、数字がパーティションを表しています。 ※注：/dev/sda*は M.2 SSD のデバイスファイルとなります。M.2 SSD のサブストレージを接続した場合、/dev/sdb*のデバイスファイルとなります。新たに追加した USB メモリ等は sdc 以降になります。サブストレージを接続していない場合は sdb 以降になります。	
/dev/shm	RAM ディスクです。	○
/etc	設定ファイルが格納されています。	
/home	システムに登録された通常ユーザの個人用ディレクトリが入っています。Algonomix10 では、「asdusr」というユーザが登録されています。ftp や ssh ではこのユーザに対してアクセスします。 ユーザ名 : asdusr パスワード : asdusr	
/lib	システム起動時や、/bin や /sbin のコマンドを実行する時に使用される共有ライブラリが格納されています。	
/media	CD-ROM やフロッピーディスクなどの外付けメディア用のディレクトリです。	
/mnt	一時的なファイルシステム用ディレクトリです。	
/opt	オプションのソフトパッケージコピー、インストールファイルが格納されているディレクトリです。	
/proc	バーチャルファイルシステム用の特別なディレクトリです。	
/root	システムの管理者権限を持ったユーザのホームディレクトリです。	
/run	実行プロセス関連データが格納されています。	○
/run/lock	保持ディレクトリです。	○
/sbin	管理用バイナリファイル用のディレクトリです。例えば、reboot、shutdown、lsmod など が格納されています。	
/srv	HTTP、FTP などのサービス用のデータが格納されています。	
/sys	デバイスの情報が格納されているディレクトリです。	
/sys/fs/cgroup	複数のグループをまとめて管理するための機能用のディレクトリです。	○
/tmp	一時ファイルを格納するディレクトリです。起動時に内容が消去されます。	○
/usr/bin	一般的なコマンドが格納されています。	
/usr/include	C 言語で使用する組込みファイルが格納されています。	
/usr/lib	一般的なライブラリファイルが格納されています。	
/usr/local	ユーザで作成されたプログラムを格納する領域です。	
/usr/sbin	システム関連のコマンドが格納されています。	
/usr/share	デフォルト設定ファイル、イメージ、ドキュメント等の共有ファイルが格納されています。	

ディレクトリ名	内容	tempfs
/usr/src	ソースコードが格納されます。	
/var	頻繁に変更されるデータが格納されています。	
/var/cache	アプリケーションのキャッシュデータが格納されています。	
/var/lib	アプリケーションの状態や APT の dpkg が管理されているパッケージ情報が格納されています。	
/var/log	システムやアプリケーションのログが格納されています。	○
/var/tmp	一時ファイルやディレクトリを必要とするプログラムで利用します。 /tmp よりもデータは長く保持されます。	○

2-3 Algonomix10 設定ツール「ASD Config」について

本項では、4C シリーズ用の各種設定ツール「ASD Config」について説明します。

ASD Config Menu を起動する方法は以下の2通りの方法があります。

1. 出荷時状態で起動したとき、自動的に起動されます。
2. コンソールを起動させ、下記のコマンドを実行します。

```
$ sudo AsdConfigMenu
```

ASD Config Menu が起動すると、図 2-3-1 のような画面が表示されます。この画面から、各種設定ツールを起動します。

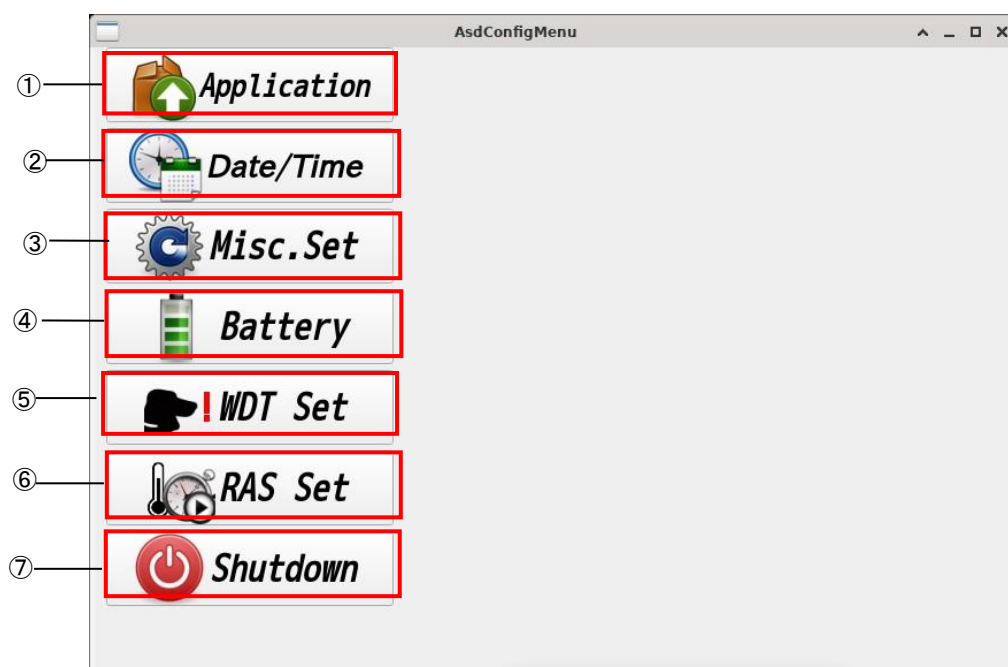


図 2-3-1. ASD Config Menu

ASD Config Menu から起動できる各ツールについて説明します。

- ① ASD Application Config
ASD Application Config は、ユーザアプリケーションのアップデートを行うためのツールです。
詳細は『2-3-1 ASD Application Config について』で説明します。
- ② ASD Date Config
ASD Date Config は、日時設定を行うためのツールです。
詳細は『2-3-2 ASD Date Config について』で説明します。
- ③ ASD Misc Setting
ASD Misc Setting は、本製品の製品情報の確認や画面表示・タッチパネルのキャリブレーション設定をするためのツールです。
詳細は『2-3-3 ASD Misc Config について』で説明します。
- ④ ASD Battery Config
ASD UPS Config は、UPS 機能の設定や RAM バックアップ機能の設定を行うためのツールです。
詳細は、『2-3-4 ASD Battery Config について』で説明します。
- ⑤ ASD WatchdogTimer Config
ASD WatchdogTimer Config は、ハードウェア・ウォッチドッグタイマの設定を行うためのツールです。

詳細は『2-3-5 ASD WatchdogTimer Config について』で説明します。

⑥ ASD Ras Config

ASD Ras Config は、CPU 温度の確認や WakeOnRTC 機能の設定を行うためのツールです。

詳細は『2-3-6 ASD RAS Config について』で説明します。

⑦ 電源オプション

シャットダウン、再起動を行います。

詳細は『2-3-7 ASD Shutdown Menu について』を参照してください。

2-3-1 ASD Application Config について

ASD Application Config は、USB メモリを用いたアップデートを行うためのツールです。

ASD Application Config を起動するには、AsdConfigMenu 画面から [Application] を選択します。

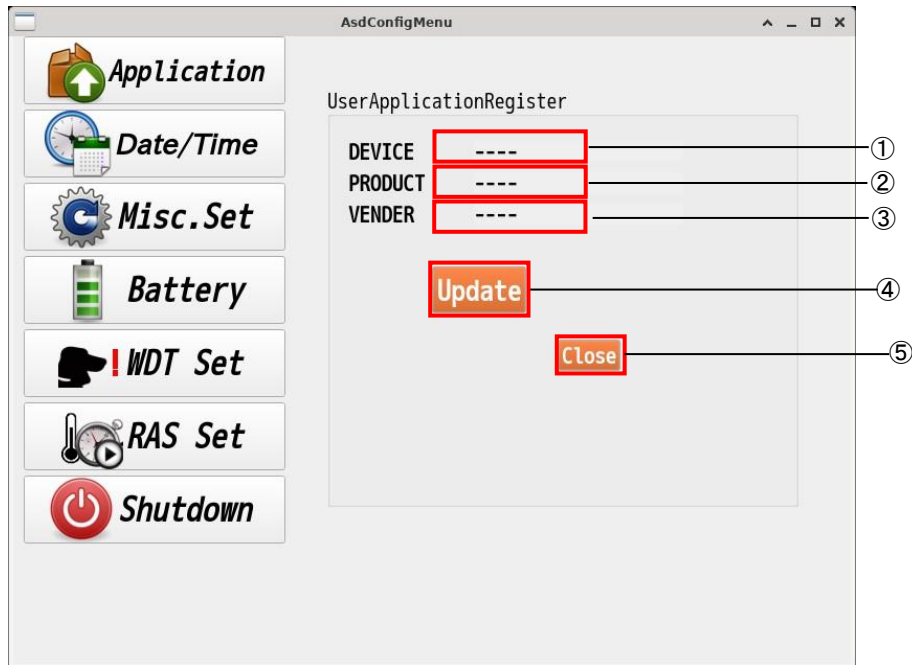


図 2-3-1-1. アップデート画面

- ① デバイス名の表示
USB メモリが認識されている場合、[usb-storage]と表示されます。
USB メモリが認識されていない場合、[-]が表示されます。
- ② プロダクト名の表示
USB メモリが認識されている場合、USB メモリの種類が表示されます。
USB メモリが認識されていない場合、[-]が表示されます。
- ③ ベンダ名の表示
USB メモリが認識されている場合、USB メモリの製造元が表示されます。
USB メモリが認識されていない場合、[-]が表示されます。
- ④ アップデート
[Update] ボタンを押下することで、USB メモリ内にある「download.sh」が実行されます。
「download.sh」はシェルスクリプトである必要があります。
- ⑤ 終了
「Close」 ボタンを押下することで、ASD Application Config を終了します。

2-3-2 ASD Date Config について

ASD Date Config は時計を設定するためのツールです。NTP サーバを設定し、自動的に時計を調整することもできます。

ASD Date Config を起動するには、AsdConfigMenu 画面から [Date/Time] を選択します。

●時計の手動設定

[Date/Time] の項目を選択することで、手動で時計を設定できます。

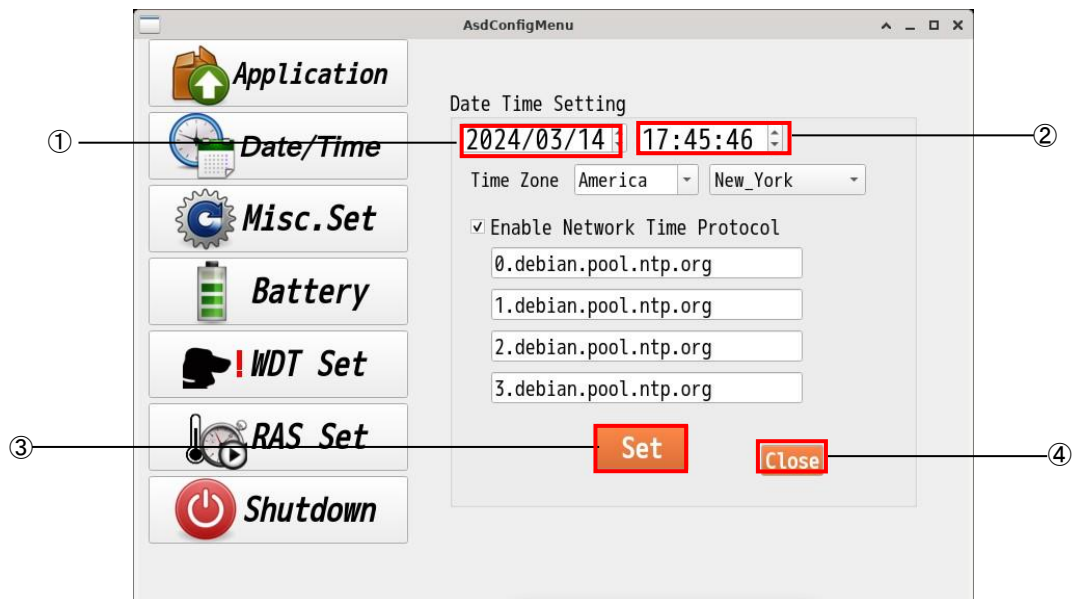


図 2-3-2-1. 時計の設定

- ① 日付を設定
年、月、日の値をクリックすることで、それぞれの値を設定できます。
▲、▼で選択した日時を Up/Down できます。
- ② 時刻を設定
時、分、秒部分を押下することで、時間を設定できます。
時、分、秒単位で指定できます。
- ③ 設定の反映
「Set」ボタンを押下することで設定を反映します。
- ④ 終了
[Close]ボタンを押下することで ASD Date Config を終了します。
[Set]ボタンを押下せずに、[Close]ボタンを押下した場合、設定は破棄されます。

●タイムゾーンの設定

Time Zone の項目からタイムゾーンを設定できます。

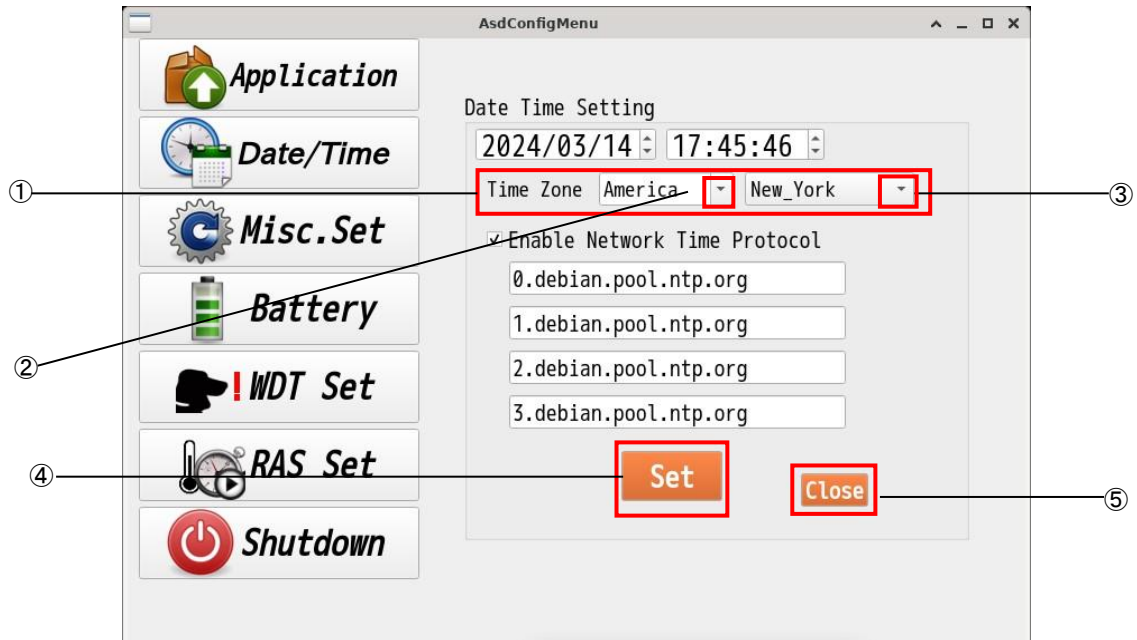


図 2-3-2-2. タイムゾーンの設定

① タイムゾーンの表示

現在設定されているタイムゾーンを表示します。

② 地域を選択

設定するタイムゾーンの地域を選択します。タイムゾーンの都市を東京に設定する場合は「Asia」を選択します。

③ 都市の選択

設定するタイムゾーンの都市を選択します。タイムゾーンの都市を東京に設定する場合は「Tokyo」を選択します。

④ 設定の反映

[Set] ボタンを押下することで設定を反映します。

⑤ 終了

[Close] ボタンを押下することで ASD Date Config を終了します。

[Set] ボタンを押下せずに、[Close] ボタンを押下した場合、変更は破棄されます。

●NTP サーバの設定

NTP は、時計を自動的に調整するためのプロトコルです。ネットワークに接続した状態で、NTP を用いると 4C シリーズの時計が NTP サーバの時計に同期されます。

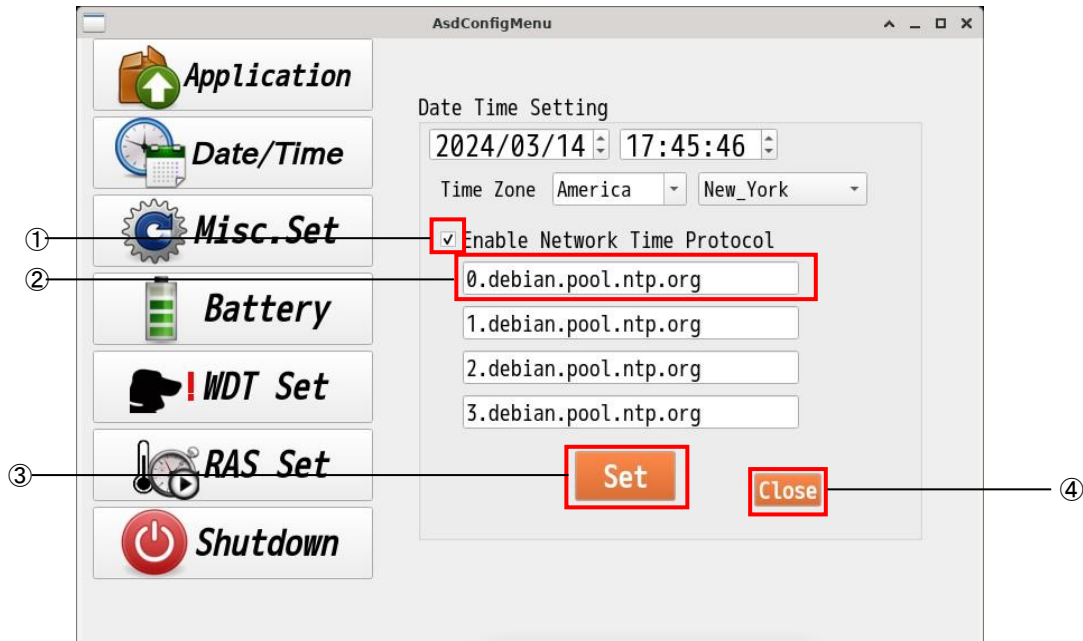


図 2-3-2-3. NTP サーバの設定

① NTP の有効、無効の切替え

チェックボックスにチェックを入れることで NTP を有効にします。
チェックを外すと NTP は無効になります。

② ntp サーバの設定

現在設定されている NTP サーバを表示します。
NTP を有効にした場合、この項目を押下することで入力画面が表示され、NTP サーバを設定できます。
NTP サーバは最大 4 個まで設定できます。

③ 設定の反映

[Set] ボタンを押下することで設定を反映します。

④ 終了

[Close] ボタンを押下することで ASD Date Config を終了します。
[Set] ボタンを押下せずに、[Close] ボタンを押下した場合、設定は破棄されます。

※注：4C シリーズの時計と NTP サーバの時計が大きくずれている場合、NTP デーモンが終了することがあります。その際は、一旦 NTP を無効にし、手動で時刻を設定後、再起動してください。

2-3-3 ASD Misc Configについて

●製品情報読み出し

AsdConfigMenu 画面の[Misc. Set]を選択することで、基板情報やFPGA バージョン、Linux カーネルビルドバージョンの情報を読み出すことができます。

表 2-3-3-1. 製品情報

項目名	内容
Soft Version	Algonomix の OS バージョン
Board Type	基板型番
Board Version	FPGA のバージョン
Kernel Version	Linux カーネルのビルドバージョン

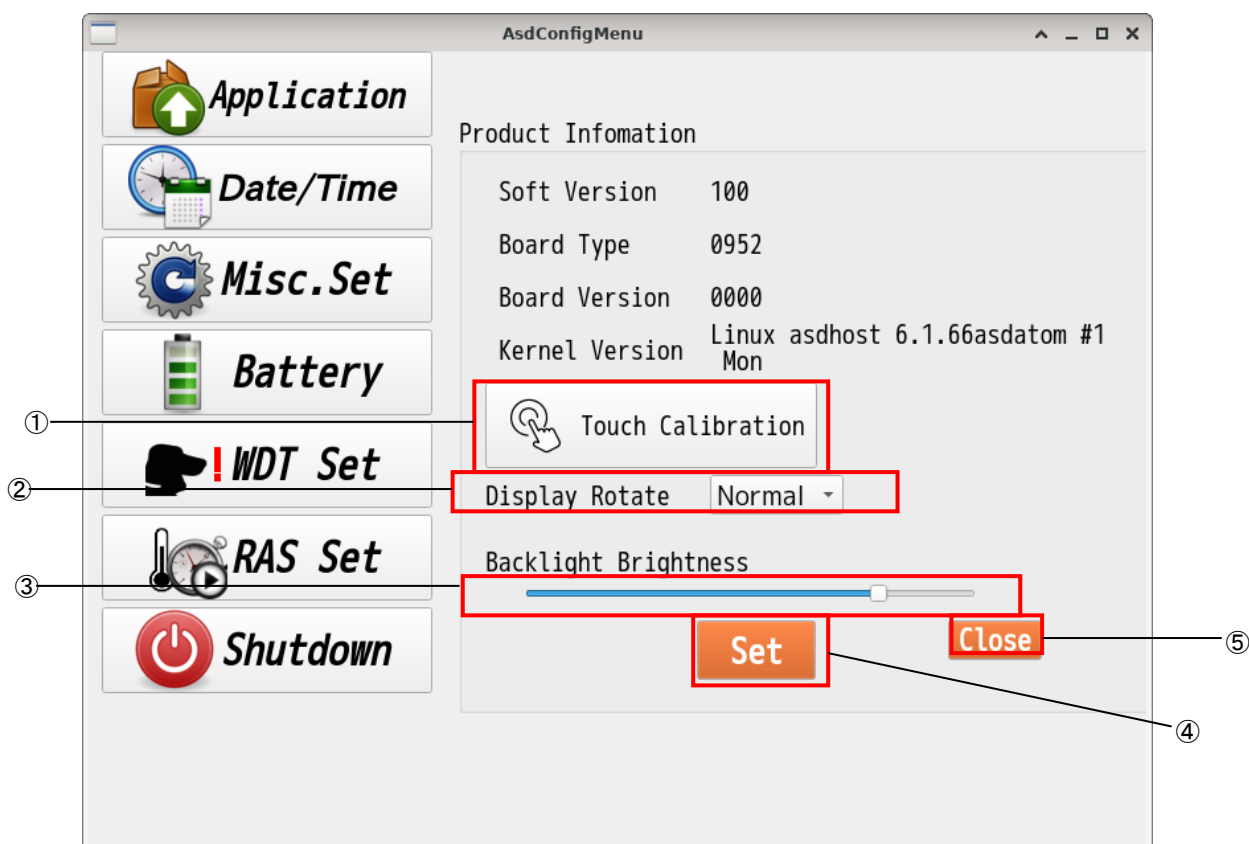


図 2-3-3-1. メニュー画面

- ① タッチパネルキャリブレーション
タッチパネルキャリブレーションを開始します。
- ② ディスプレイとタッチパネルの回転
ディスプレイとタッチパネルの回転方向を設定します。
任意の向きを選択することで、画面が回転します。
回転後、任意の位置をタッチできなくなった場合、①のキャリブレーションを行ってください。
- ③ バックライト輝度調整
スライダーを動かすことでバックライト輝度を 0～255 段階で調整します。
- ④ 設定の反映
[Set] ボタンを押下することでバックライト輝度設定を反映します。
- ⑤ 終了
[Close] ボタンを押下することで ASD Misc Config を終了します。
[Set] ボタンを押下せずに、[Close] ボタンを押下した場合、バックライト輝度設定は破棄されます。

●ディスプレイとタッチパネルの回転

②のコンボボックスをクリックすると回転方向の設定メニューが表示されます。

画面の向きと設定メニューはそれぞれ表 2-3-3-2 のように対応しています。

表 2-3-3-2. ディスプレイの向きの設定

名称	内容
Normal	デフォルトの画面の向きです。
Left	反時計回りに 90° 回転します。
Inverted	180° 回転します。
Right	時計回りに 90° 回転します。

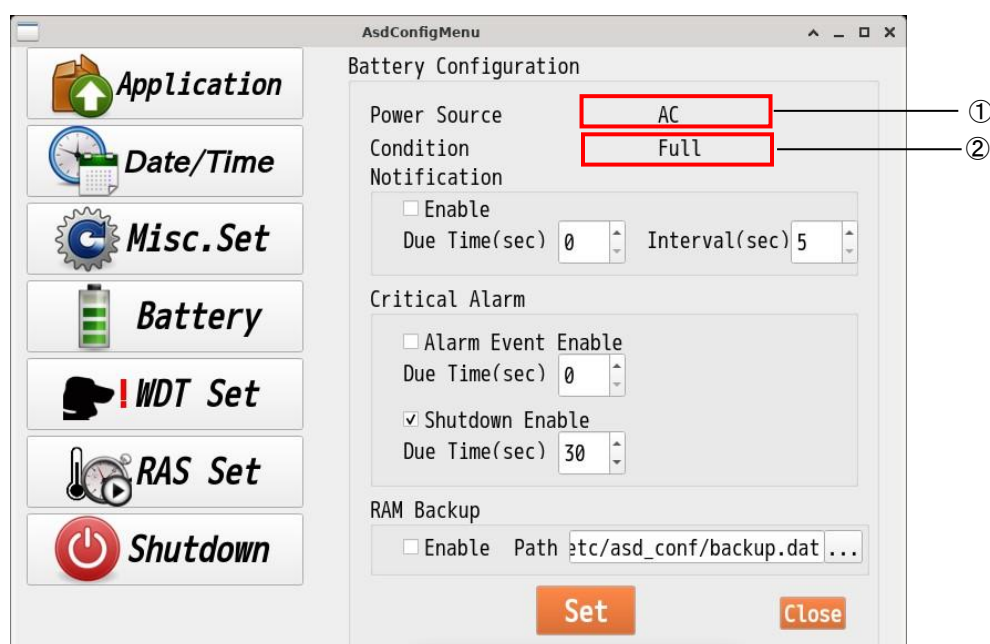
2-3-4 ASD Battery Configについて

ASD Battery Config は、UPS バッテリの状態取得、設定、シャットダウン時のバックアップ機能の設定を行うためのツールです。

ASD Battery Config を起動するには、AsdConfigMenu 画面から[Battery]を選択します。

●UPS バッテリ状態の表示

[Battery]を選択することで、UPS の UPS バッテリ状態を取得できます。



2-3-4-1. バッテリ状態の表示

① 電源種別

端末の電源種別を表示します。

表 2-3-4-1. 電源種別

名称	動作
AC	AC 電源で動作中です。
Battery	UPS バッテリ電源で動作中です。

② 状態

UPS バッテリの状態を表示します。

表 2-3-4-2. UPS バッテリの状態

名称	動作
Charging	UPS バッテリは充電中です。
Full	UPS バッテリは満充電状態です。
Discharging	バッテリーは放電中です。
Error	UPS バッテリ保護機能が動作中です。 保護機能は UPS バッテリ温度が 4~67℃の範囲外になると動作します。

●UPS バッテリ保護機能動作通知設定

Notification の項目から UPS バッテリ保護機能が動作したときの通知について設定できます。

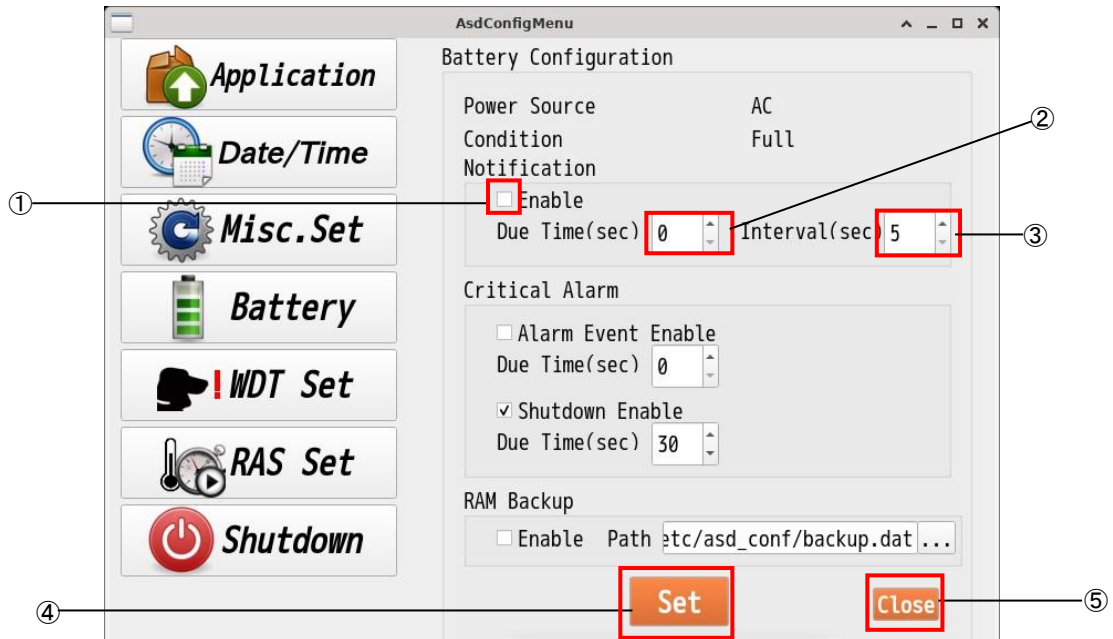


図 2-3-4-2. UPS バッテリ保護機能動作通知設定

- ① UPS バッテリ保護機能動作通知有効/無効設定
チェックボックスにチェックを入れることで、UPS バッテリ保護機能動作時の通知が有効になります。
- ② 通知開始時間
UPS バッテリ保護機能が動作してから、通知を開始するまでの時間を設定します（0～120[秒]）。
- ③ 通知間隔
②の通知後の通知間隔を設定します（5～300[秒]）。
- ④ 設定の反映
[Set] ボタンを押下することで設定を反映します。
- ⑤ 終了
[Close] ボタンを押下することで ASD Battery Config を終了します。
[Set] ボタンを押下せずに、[Close] ボタンを押下した場合、設定は破棄されます。

●バッテリー警告設定

Critical Alarm の項目から UPS バッテリ駆動開始後の警告、シャットダウンについて設定できます。

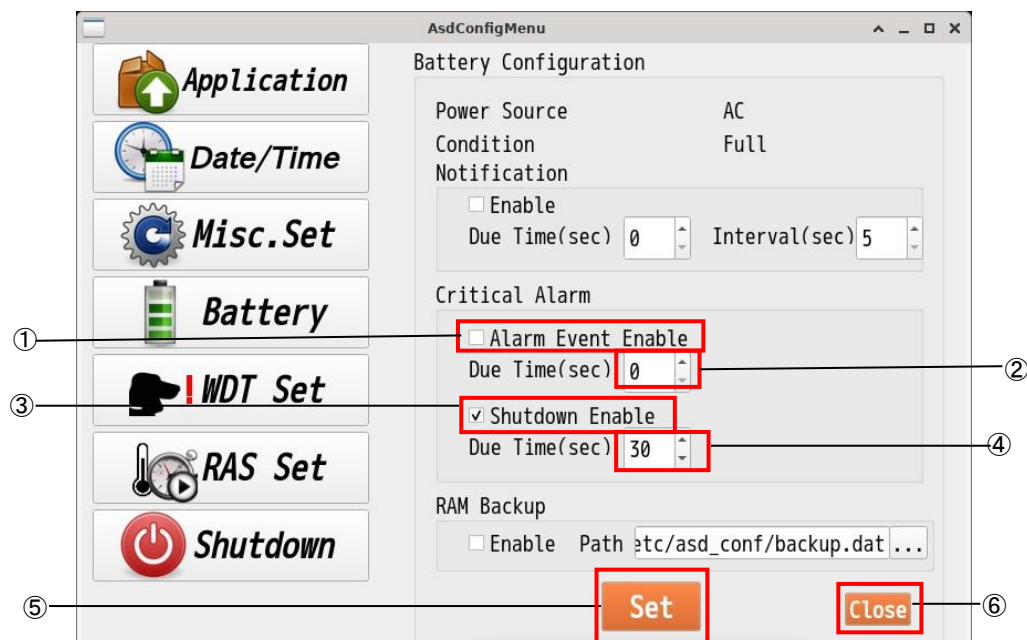


図 2-3-4-3. UPS バッテリ警告設定

- ① UPS バッテリ警告有効/無効設定
チェックボックスにチェックを入れることで、UPS バッテリ駆動開始後の警告が有効になります。
- ② 警告開始時間
UPS バッテリ駆動開始から警告を送信するまでの時間の設定します (0~60[秒])。
- ③ シャットダウン有効/無効設定
チェックボックスにチェックを入れることで、UPS バッテリ駆動開始後のシャットダウンが有効になります。
- ④ シャットダウン開始時間
UPS バッテリ駆動開始からシャットダウンするまでの時間を設定します (0~60[秒])。
- ⑤ 設定の反映
[Set] ボタンを押下することで設定を反映します。
- ⑥ 終了
[Close] ボタンを押下することで ASD Battery Config を終了します。
[Set] ボタンを押下せずに、[Close] ボタンを押下した場合、設定は破棄されます。

●RAM バックアップ設定

RAM Backup の項目から仮想 RAM デバイスのシャットダウン時のバックアップ機能について設定できます。

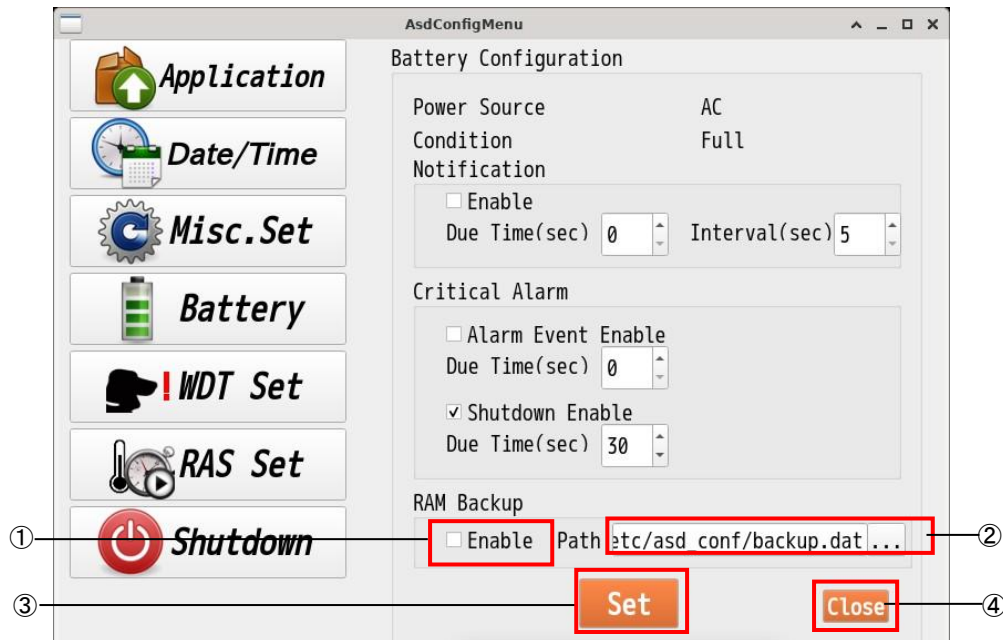


図 2-3-4-4. RAM バックアップ設定

- ① バックアップ有効/無効設定
チェックボックスにチェックを入れることで、バックアップ機能が有効になります。
- ② バックアップファイルパス
バックアップファイルの保存先を指定します。
バックアップファイルを外部ストレージに保存する場合、起動時に自動マウントする必要があります。
起動時に自動マウントする方法については、『4-6-3 外部ストレージデバイスの起動時マウントについて』を参照してください。
- ③ 設定の反映
[Set] ボタンを押下することで設定を反映します。
- ④ 終了
[Close] ボタンを押下することで ASD Battery Config を終了します。
[Set] ボタンを押下せずに、[Close] ボタンを押下した場合、設定は破棄されます。

2-3-5 ASD WatchdogTimer Configについて

ASD WatchdogTimer Config は、ハードウェア・ウォッチドッグタイマの設定の取得、変更を行うためのツールです。

ASD WatchdogTimer Config を起動するには、AsdConfigMenu 画面から[WDT Set]を選択します。

●ハードウェア・ウォッチドッグタイマの設定

Watchdog setting の項目から、ハードウェア・ウォッチドッグタイマを設定できます。

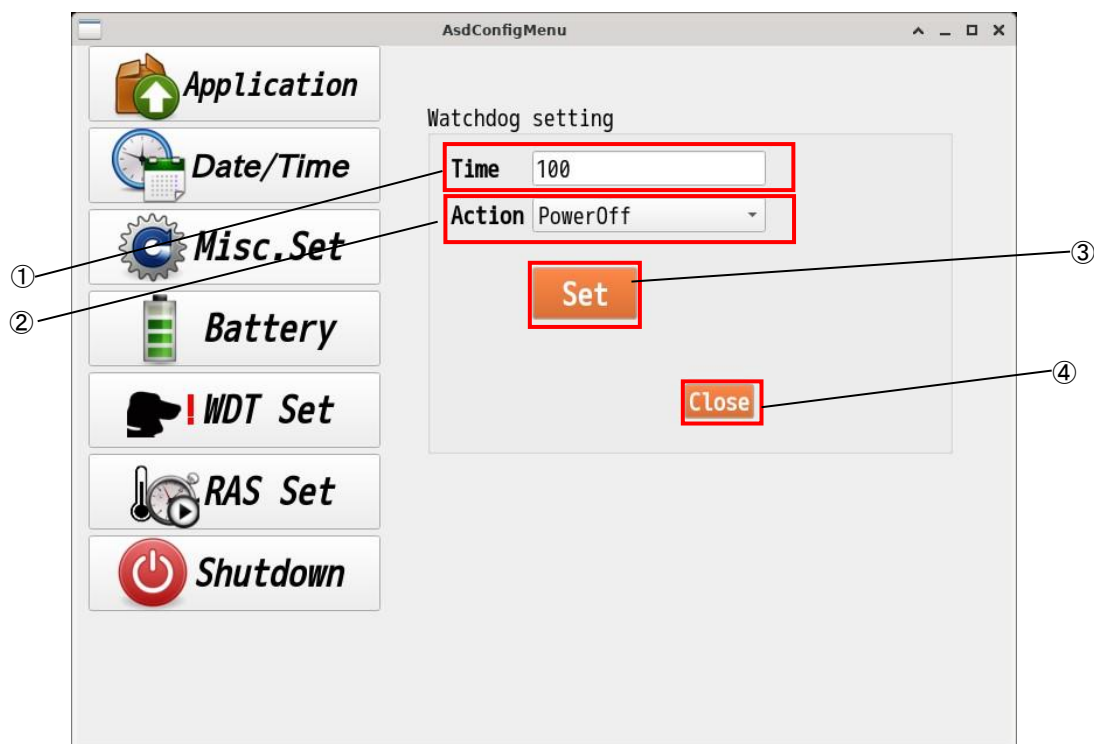


図 2-3-5-1. ハードウェア・ウォッチドッグタイマの設定

① タイマ時間の設定

ハードウェア・ウォッチドッグタイマがタイムアウトするまでのタイマ時間を設定します。
有効設定値は1～65535、タイマ時間は設定値×100【msec】になります。
デフォルト値は20です。

② 動作の設定

ハードウェア・ウォッチドッグタイマがタイムアウトした際の動作を設定します。
選択できる動作を表 2-3-5-1 に示します。デフォルトでは Reboot が選択されます。

表 2-3-5-1. ハードウェア・ウォッチドッグタイマのタイムアウト時の動作

名称	動作
Shutdown	shutdown コマンドを実行します。
Reboot	reboot コマンドを実行します。
Popup	ポップアップメッセージを表示します。
Event	ユーザアプリケーションにイベント発生を通知します。
PowerOff	強制的に電源を切ります。
Reset	強制的に再起動します。

※注：PowerOff および Reset は、ハードウェア的に電源をオフするため、システムがハングアップしても動作しますが、データが破損する可能性があります。

③ 設定の反映

[Set] ボタンを押下することで、設定を反映します。

④ 終了

[Close] ボタンを押下することで、ASD WatchdogTimer Config を終了します。

[Set] ボタンを押下せずに [Close] ボタンを押下した場合、設定は破棄されます。

2-3-6 ASD RAS Config について

ASD RAS Config は、CPU 温度の確認や WakeOnRTC の設定の取得、変更を行うためのツールです。
ASD RAS Config を起動するには、AsdConfigMenu 画面から [RAS Set] を選択します。

●CPU 温度の表示

RAS Configuration の項目から、CPU 温度の表示が確認できます。

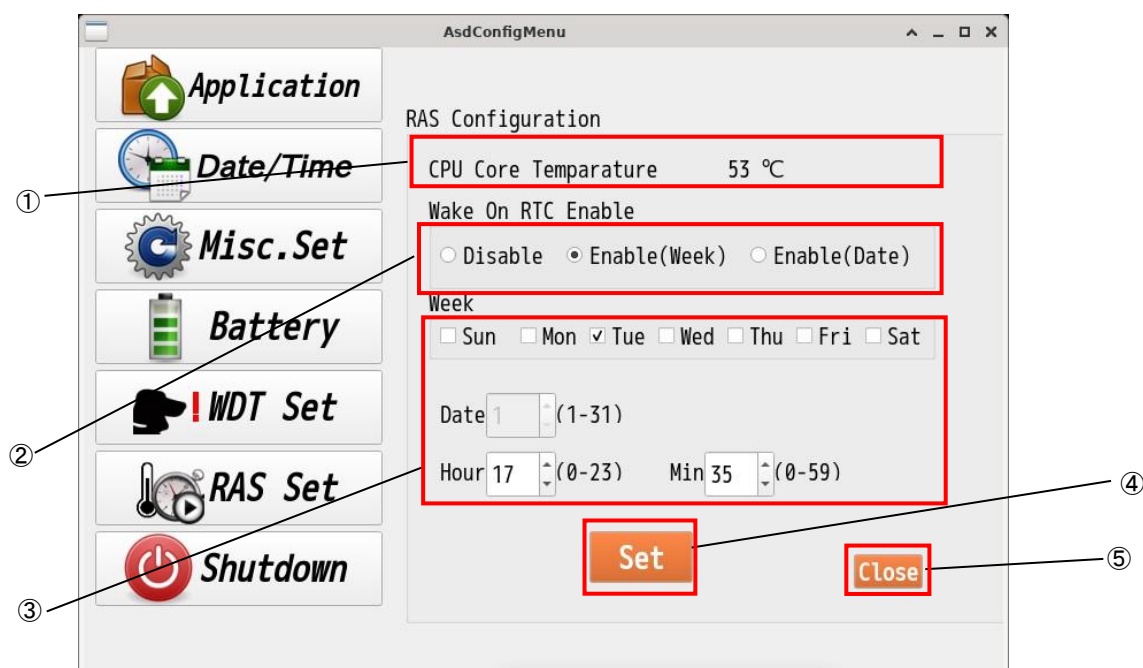


図 2-3-6-1. CPU 温度の表示

① CPU 温度

CPU の Core の温度を表示します。

●Wake On RTC の設定

Wake On RTC Enable の項目から、Wake On RTC 機能の設定ができます。

② Wake On RTC Enable

Wake On Rtc 機能の設定を行います。

表 2-3-6-1. Wake On RTC Enable

名称	動作
Disable	Wake On Rtc Timer 機能を無効にします。
Enable(Week)	曜日指定の Wake On RTC 機能を有効にします。
Enable(Day)	日付指定の Wake On RTC 機能を有効にします。

③ Wake On RTC Timer 設定

Wake On RTC 機能で電源を復帰させる時刻を設定します。

表 2-3-6-2. Wake On RTC Timer

名称	動作
Week	曜日を設定します。 (Wake On RTC Enable の設定で「Enable(Week)」設定時のみ有効)
Date	日を設定します。 (Wake On RTC Enable の設定で「Enable(Day)」設定時のみ有効)
Hour	時を設定します。
Min	分を設定します。

④ 設定の反映

[Set] ボタンを押下することで、設定を反映します。

⑤ 終了

[Close] ボタンを押下することで、ASD Ras Config を終了します。

[Set] ボタンを押下せずに [Close] ボタンを押下した場合、設定は破棄されます。

※注：端末が起動中、もしくは電源未接続の場合は Wake On RTC Timer は機能しませんので注意してください。

2-3-7 ASD Shutdown Menu について

ASD Shutdown Menu によって、4C シリーズの再起動、シャットダウンを行うことができます。

また、起動時のメニューを変更することもできます。

ASD Shutdown Menu を起動するには、AsdConfigMenu 画面から [Shutdown] を選択します。

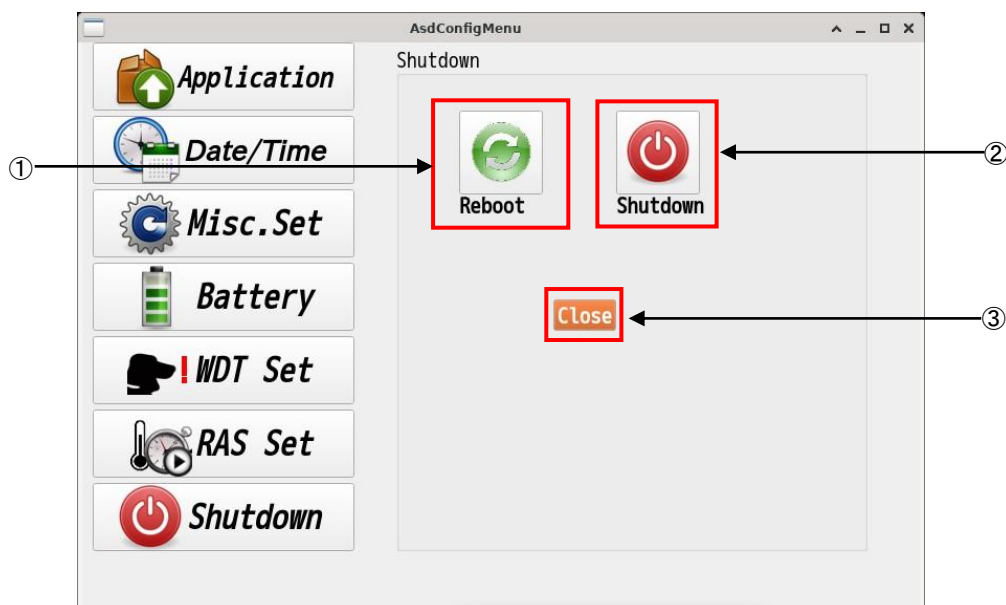


図 2-3-7-1. ASD Shutdown Menu

- ① 再起動
[Reboot] ボタンを押下することで、4C シリーズが再起動します。
- ② シャットダウン
[Shutdown] ボタンを押下することで、4C シリーズがシャットダウンします。
- ③ 終了
ASD Shutdown Menu を終了して AsdConfigMenu に戻ります。

2-4 有線 LAN の設定について

本稿では、4C シリーズにおける有線 LAN の設定方法について説明します。

- ① [アプリケーション]→[設定]→[高度なネットワーク設定]を選択します。



図 2-4-1. ネットワーク設定画面の起動

- ② 図2-4-2のようなネットワーク設定画面が起動します。認識しているLANの一覧が表示されています。設定したい接続を選択して[編集]ボタンをクリックします。

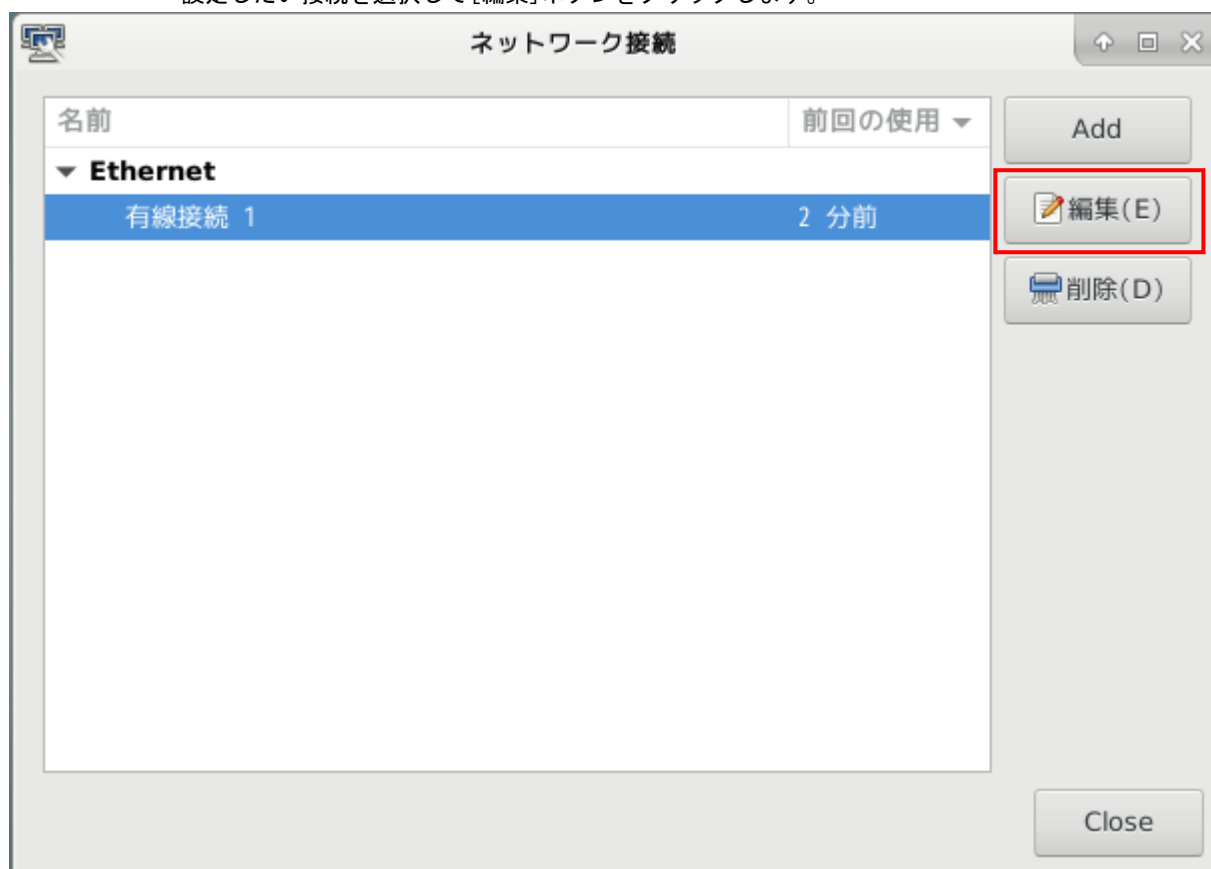


図 2-4-2. ネットワーク設定画面

- ③ 図 2-4-3 のような有線 LAN の設定画面が起動します。
接続しているネットワークの環境に合わせた設定を行ってください。



図 2-4-3. 有線 LAN 設定画面

- ④ 「OK」をクリックすることで、自動的に再接続されます。設定されている IP アドレスを確認する場合は、コンソールウィンドウを起動して下記のコマンドを実行してください。
- ・現在の設定を見る場合

```
$ ip a
```

IP アドレスの確認方法について、従来の Linux ディストリビューションでは、IP アドレスを確認するために、「ifconfig」というコマンドを使用されていました。

しかし、「ifconfig」を含む「net-tools」というパッケージは、メンテナンスされいないため、数年前に非推奨なパッケージになりました。

Debian 12.1 では「net-tools」パッケージはインストールされておらず、「ifconfig」コマンドも使えません。今日の Linux ディストリビューションでは、「iproute2」（「ip」コマンドの正式名）が正式採用されています。

「ifconfig」コマンドに対応される「ip」コマンドは以下になります。

	ifconfig	iproute2
各種インターフェースの設定と表示	ifconfig	ip a (ip addr show)
インターフェースの起動	ifconfig eth0 up	ip link set eth0 up
インターフェースの停止	ifconfig eth0 down	ip link set eth0 down

「ip」コマンドは、「ifconfig」よりも、豊富な機能を搭載しています。詳細は、インターネット等で確認してください。

2-5 無線 LAN の設定について

本項では、4C シリーズにおける無線 LAN の設定方法について説明します。

●無線 LAN の設定変更

無線 LAN の設定を変更したい場合は、下記の方法で設定してください。

- ① [メニューアイコン]→[設定]→[高度なネットワーク設定]を選択します。



図 2-5-1. ネットワーク設定画面の起動

- ② 図 2-5-2 のようなネットワーク設定画面が起動します。[+]ボタンをクリックします。

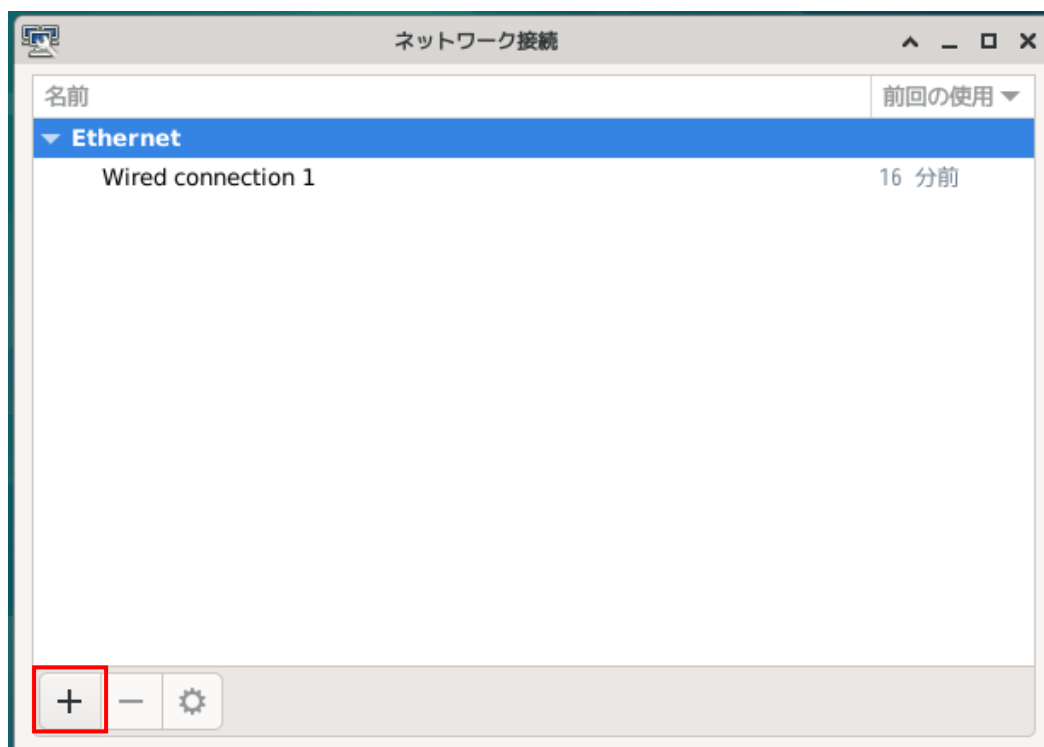


図 2-5-2. ネットワーク設定画面

- ③ 図 2-5-4 のような画面が起動します。「Wi-Fi」を選択して[作成]をクリックしてください。

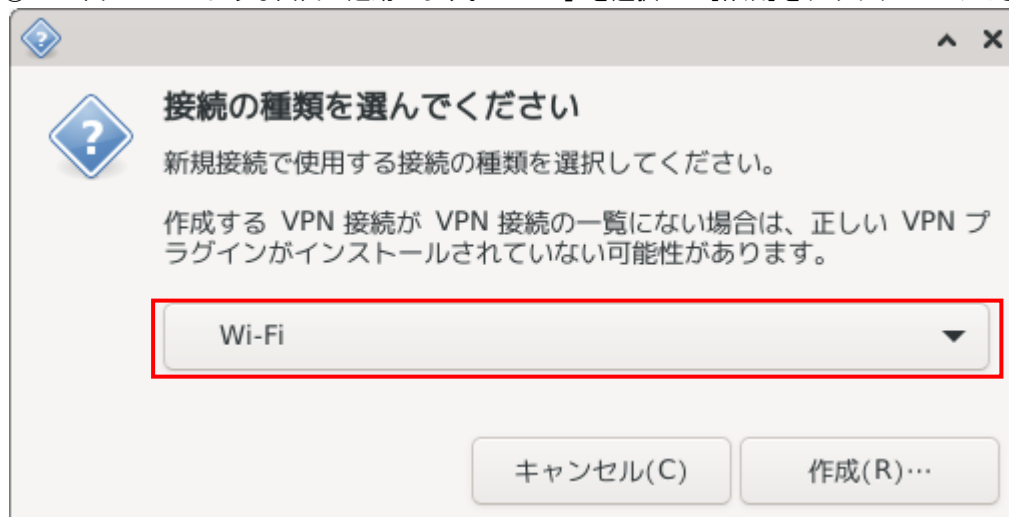
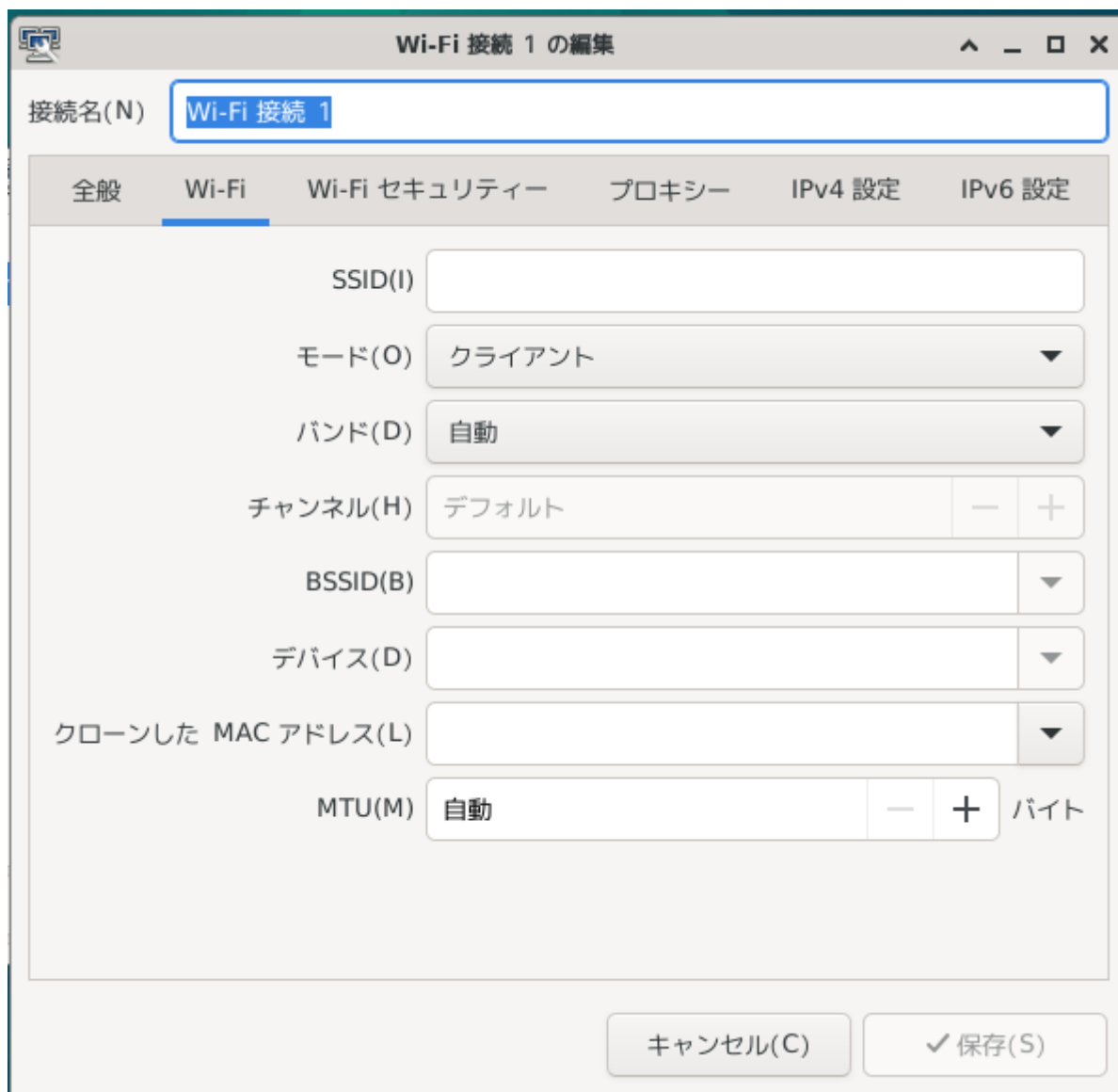


図 2-5-3. 無線 LAN 設定画面

- ④ 図 2-5-4 のような設定画面が起動します。接続しているネットワークの環境に合わせた設定を行ってください。



The image shows a software window titled "Wi-Fi 接続 1 の編集" (Edit Wi-Fi Connection 1). At the top, there is a text field for "接続名(N)" (Connection Name) containing "Wi-Fi 接続 1". Below this is a tabbed interface with six tabs: "全般" (General), "Wi-Fi", "Wi-Fi セキュリティ" (Wi-Fi Security), "プロキシ" (Proxy), "IPv4 設定" (IPv4 Settings), and "IPv6 設定" (IPv6 Settings). The "Wi-Fi" tab is currently selected. The main area of the "Wi-Fi" tab contains several settings:

- SSID(I): A text input field.
- モード(O) (Mode): A dropdown menu set to "クライアント" (Client).
- バンド(D) (Band): A dropdown menu set to "自動" (Automatic).
- チャンネル(H) (Channel): A text input field set to "デフォルト" (Default), with minus and plus buttons on the right.
- BSSID(B): A text input field with a dropdown arrow on the right.
- デバイス(D) (Device): A text input field with a dropdown arrow on the right.
- クローンした MAC アドレス(L) (Cloned MAC Address): A text input field with a dropdown arrow on the right.
- MTU(M): A text input field set to "自動" (Automatic), with minus and plus buttons on the right, followed by the unit "バイト" (Bytes).

At the bottom right of the window, there are two buttons: "キャンセル(C)" (Cancel) and "✓ 保存(S)" (Save).

図 2-5-4. 無線 LAN 設定画面

2-6 sysfs ファイルシステム

Linux2.6 カーネルから導入された sysfs ファイルシステムは、proc、devfs、devpty のファイルシステムの統合だと言えます。sysfs ファイルシステムは、システムに接続されているデバイスとバスを、ユーザースペースからアクセスできるファイルシステム内に階層式に列記します。

sysfs ファイルシステムは /sys/ でマウントされ、いくつか異なる方法でシステムに接続されたデバイスを構成する複数のディレクトリを含んでいます。

4C シリーズの固有デバイスとして以下のデバイス制御が可能です。

2-6-1 基板情報

基板情報を管理します。

表 2-6-1-1. sysfs の基板情報を制御する項目

sysfs ファイル名	データ	内容
/sys/devices/platform/mainboard/boardversion	0XXX	Read することで FPGA バージョンが読み出せます。
/sys/devices/platform/mainboard/buildversion	XXXXXXXX	Read することでカーネルのビルドバージョンが読み出せます。
/sys/devices/platform/mainboard/machcode	0000XXXX	Read することでマシンコードが読み出せます。
/sys/devices/platform/asd_sram/size	XXXX	Read することで仮想 RAM のサイズが読み出せます。単位は[kbyte]です。
/sys/devices/platform/asd_ups/condition	X	Read することで UPS 状態が読み出せます。 0 : バッテリー充電中 1 : バッテリー満充電 2 : バッテリー放電中 3 : バッテリー異常
/sys/devices/platform/asd_ups/powersource	X	Read することで UPS 電力源が読み出せます。 0 : AC 1 : バッテリー
/sys/devices/platform/mainboard/battery	X	Read することでバックアップバッテリー電池のステータスが読み出せます。 0 : 正常 1 : バッテリー低下

2-6-2 温度センサ

基板上の温度センサの情報を取得します。

表 2-6-2-1. sysfs の基板上温度センサの情報を取得する項目

sysfs ファイル名	データ	内容
/sys/bus/i2c/devices/0-004c/hwmon/hwmon3/temp1_input	XX000	Read することで基板上の温度センサ (DRAM 付近) の温度を読み取ります。
/sys/bus/i2c/devices/0-004d/hwmon/hwmon4/temp1_input	XX000	Read することで基板上の温度センサ (PMIC 付近) の温度を読み取ります。

2-6-3 LCD バックライト

産業用パネル PC の LCD バックライトを制御します。

表 2-6-3-1. sysfs の LCD バックライトを制御する項目

sysfs ファイル名	データ	内容
/sys/class/backlight/asd-bl/bl_power	0: 点灯 1: 消灯	0 を書き込むことでバックライトが点灯します。1 を書き込むことでバックライトが消灯します。
/sys/class/backlight/asd-bl/brightness	0~255	255 : 最大輝度、0 : 最小輝度の間でバックライトの輝度を調整できます。

2-7 データの保護について

2-7-1 データ保護の必要性と方法

Linux ではハードディスクや CF カード、SD カード、USB メモリ等のストレージ上にファイルを Write する際、一端書き込みデータをキャッシュ領域に保存して、Write 処理を完了し、OS のアイドル時間に実際のストレージへ書き込むことで CPU リソースの有効活用を行っています。

このため、キャッシュ領域にデータがあり、実際のストレージ上に書き込まれる前に電源を落としたりした場合、そのファイルまたは書き込み途中のセクタが破損する可能性があります。これを防ぐ為に、sync というコマンドがあります。このコマンドを実行することで、キャッシュ内にたまっているデータを実際のストレージ上にすべて書き出すことができます。ストレージにファイルを書込んだ際は電源を落とす前に sync コマンドを実行してください。

また、SD カードや USB メモリ等の抜き差しが可能なデバイスの取扱いには注意が必要となります。使用中にデバイスが抜かれた場合などは、デバイス内のファイルが破損してしまう場合があります。

また、デバイスにファイルを書込んだ場合は、sync コマンドなどを使用して書込んだ内容が確実にデバイスに書き込まれるようにしてください。また、デバイスを抜く際には、必ずデバイスをアンマウントしてから抜くようにしてください。

● sync コマンドの使用

Linux でファイル操作を行った場合、ファイルデータがファイルキャッシュとしてシステムメモリに保存され、実際の SD カードなどのデバイスには反映されていないことがあります。デバイスのファイル操作後、変更されたデータがデバイスに確実に反映されるようにするには、sync コマンドを使用してキャッシュとデバイスのデータの同期をとるようにしてください。

デバイスにファイルコピー後、sync コマンドで同期

```
# cp /home/asdusr/FILE.dat /media/asdusr/デバイス名
# sync
```

● USB メモリの書き込み不可/可能の切り替え

USB メモリを書込み不可状態でマウントし、書き込み可否の切り替えを行います。書き込み不可状態では、ファイルの書き込みを行えませんがファイルの読み出しは行うことができます。

USB メモリを書込み不可でマウントします。

```
# mount -o ro /dev/sdb1 /mnt
```

書き込み不可でマウントされている USB メモリを書込み可能にします。

```
# mount -o remount,rw /mnt
```

書き込み可能でマウントされている USB メモリを書込み不可にします。

```
# mount -o remount,ro /mnt
```

第 3 章 開発環境

Algonomix10 の開発環境について、簡単な説明が記述されています。詳細については、「Algonomix10 開発環境 ユーザーズマニュアル」を参照してください。

3-1 クロス開発環境

プログラムを開発する場合に必要なのが、ソースコードを記述するエディタ、ソースコードをコンパイルするコンパイラ、コンパイルされたプログラムを実行する為の実行環境です。

例えば、Microsoft 社の Windows 上で動作するアプリケーションを開発する場合、エディタでソースを書き、Visual Studio 等のコンパイラでコンパイルを行い、作成された exe ファイルを実行します。これで作成したアプリケーションが Windows 上で実行されます。

Linux の場合でも同じです。Linux マシン上で動作するエディタでソースを書き、gcc でコンパイル後に生成された実行ファイルを実行します。

両者とも、コンパイルと実行を同じパソコン環境上で行うことができます。このような開発方式をセルフ開発といいます。

4C シリーズでは、クロス開発方式を採用しています。クロス開発とは、コンパイル環境と実行する環境が異なる方式です。ソースコードの記述やコンパイルはパソコン上で行い、LAN 等で実行ファイルをターゲットに送って実行することになります。(図 3-1-1 参照)。4C シリーズはターゲットマシンとして開発された商品である為、セルフコンパイル環境は用意していません。

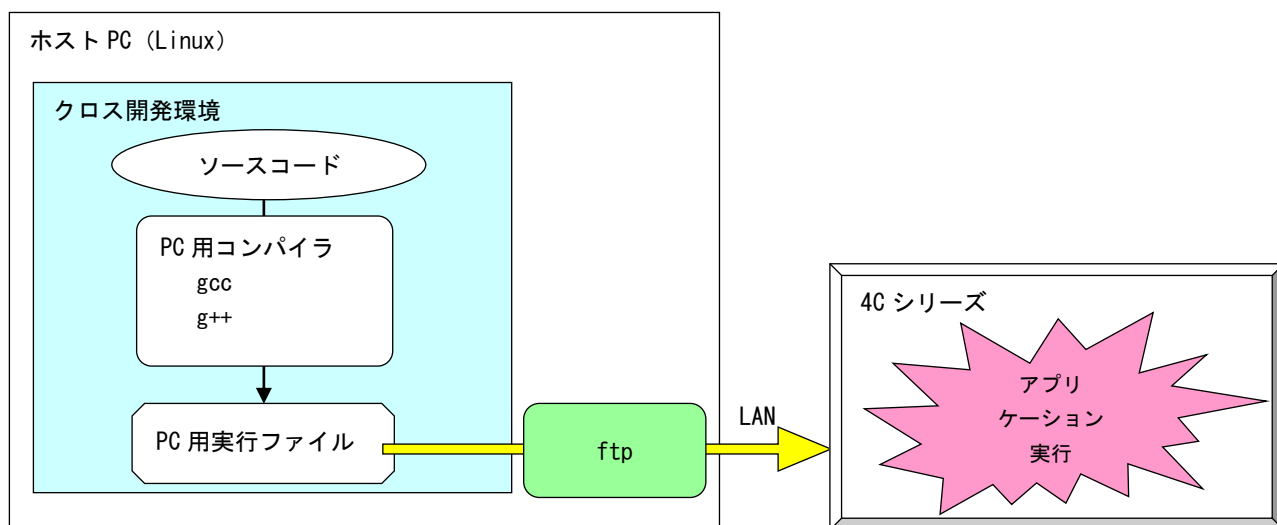


図 3-1-1. クロス開発方式イメージ図

このような開発環境を構築するには、表 3-1-1 に示すような開発環境ツールが必要となります。

表 3-1-1. クロス開発に必要なツール

ツール名	説明
GCC	GNU C コンパイラ
binutils	リンカ、アセンブラ等のソフトウェア開発ツール
GDB	デバッガ
glibc	GNU C ライブラリ

Algonomix10 開発環境は、Debian 12.1 ディストリビューションイメージに、Intel CPU 用の開発環境を組み込んだものになります。

VirtualBox という仮想マシン上で Algonomix10 開発環境を起動すれば、それぞれの CPU 搭載端末用のアプリケーションを開発することが可能です。

第 4 章 産業用 PC NP/AP/NPL/APL/AS/EC/ECL 4C シリーズについて

本章では、4C シリーズに実装されているデバイスの使用方法およびアプリケーション作成について説明しています。

Algonomix10 開発環境には、コンソール用サンプルプログラムのソースコードを用意しています。表 4-1 にコンソール用サンプルのディレクトリ名と内容を示します。

コンソール用サンプルプログラムは、コンソール上で「make」コマンドを実行することでコンパイルします。コンパイル方法は『第 3 章 開発環境』を参照してください。

※注：コンソール用サンプルプログラムは「/usr/local/tools-x64/samples/sampleConsole」に格納されています。

※注：機種により搭載されているデバイスは異なります。そのため、一部のサンプルでは 4C シリーズにデバイスが実装されていない為動作しないことがあります。

表 4-1. コンソール用サンプルプログラムのソースコード一覧

ディレクトリ名	内容	機能有無
sample_DIO	汎用入出力制御方法	『4-1 汎用入出力』を参照
sample_SerialPortChange	シリアルポート RS422/R485/RS232C 切り替え	『4-2 シリアルポート』を参照
sample_Serial	シリアルポート制御方法	『4-2 シリアルポート』を参照
sample_TcpIp	ネットワークポート制御方法	『4-3 ネットワークポート』を参照
sample_BeepOnOff	ブザーON/OFF 制御方法	—
sample_Reset	汎用入力 IN0 リセット制御方法	『4-4 RAS 機能』を参照
sample_Interrupt	汎用入力 IN1 割込み制御	『4-4 RAS 機能』を参照
sample_Timer sample_Timer2	タイマ割り込み制御	『4-4 RAS 機能』を参照
sample_SRAM	バックアップ SRAM 制御方法 (read/write)	『4-4 RAS 機能』を参照
sample_HwWdtKeepAlive	ハードウェア・ウォッチドッグタイマ操作	『4-4 RAS 機能』を参照
sample_HwWdtEventWait	ハードウェア・ウォッチドッグタイマイベント待ち	『4-4 RAS 機能』を参照
sample_WOR	WakeOnRTC 機能制御方法	『4-4 RAS 機能』を参照
sample_SMART	S. M. A. R. T. 監視イベント通知待ち	『4-4 RAS 機能』を参照
sample_UPS	UPS 通知待ち	『4-5 UPS 機能』を参照
sample_InputKey	入力されたキーコードの表示	—

固有デバイスの説明の前に、Linux の一般的なデバイスドライバアクセスについて説明します。デバイスにアクセスするには「/dev」以下に格納されているデバイスファイルに対しシステムコール(open、close、read、write、ioctl 等)を使用します。

デバイスドライバ操作は、デバイスファイルを「open」関数にてオープンし、「read」関数や「write」関数を使用してデータを読み書きします。デバイスによっては、「ioctl」関数で各種設定を行う場合もあります。また、デバイスによっては、独自のシステムコールが用意される場合もあります。

デバイス毎にどのような設定があるかは、インターネットや書籍で確認してください。ここでは、4C シリーズに搭載されたデバイスの仕様について説明します。

**※注：搭載されているデバイスの種類や実装位置は、機種毎に異なります。
それぞれの機種の詳細はハードウェアマニュアルをご参照ください。**

各機種の 1 例として、NPL4C-101A、APL4C-121A、NP4C-101A、AP4C-121A、EC4C-100AT、NPL4C-070B および NPL4C-10HE の外形図を以下に示します。他の機種については、ハードウェアのマニュアルを参照してください。各部名称を表 4-2 に、各部の実装数を表 4-3 に示します。

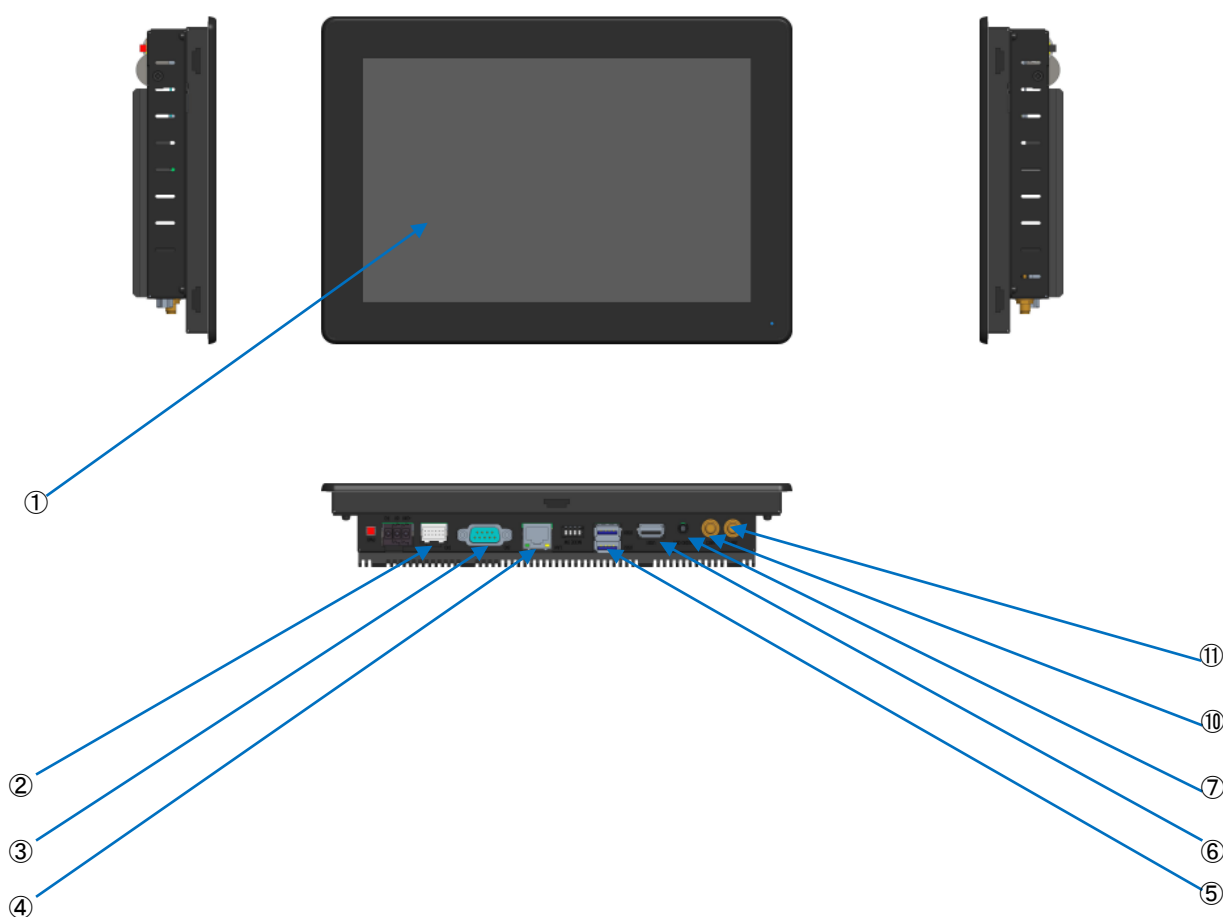


図 4-1. 外形図 (NPL4C-101B)

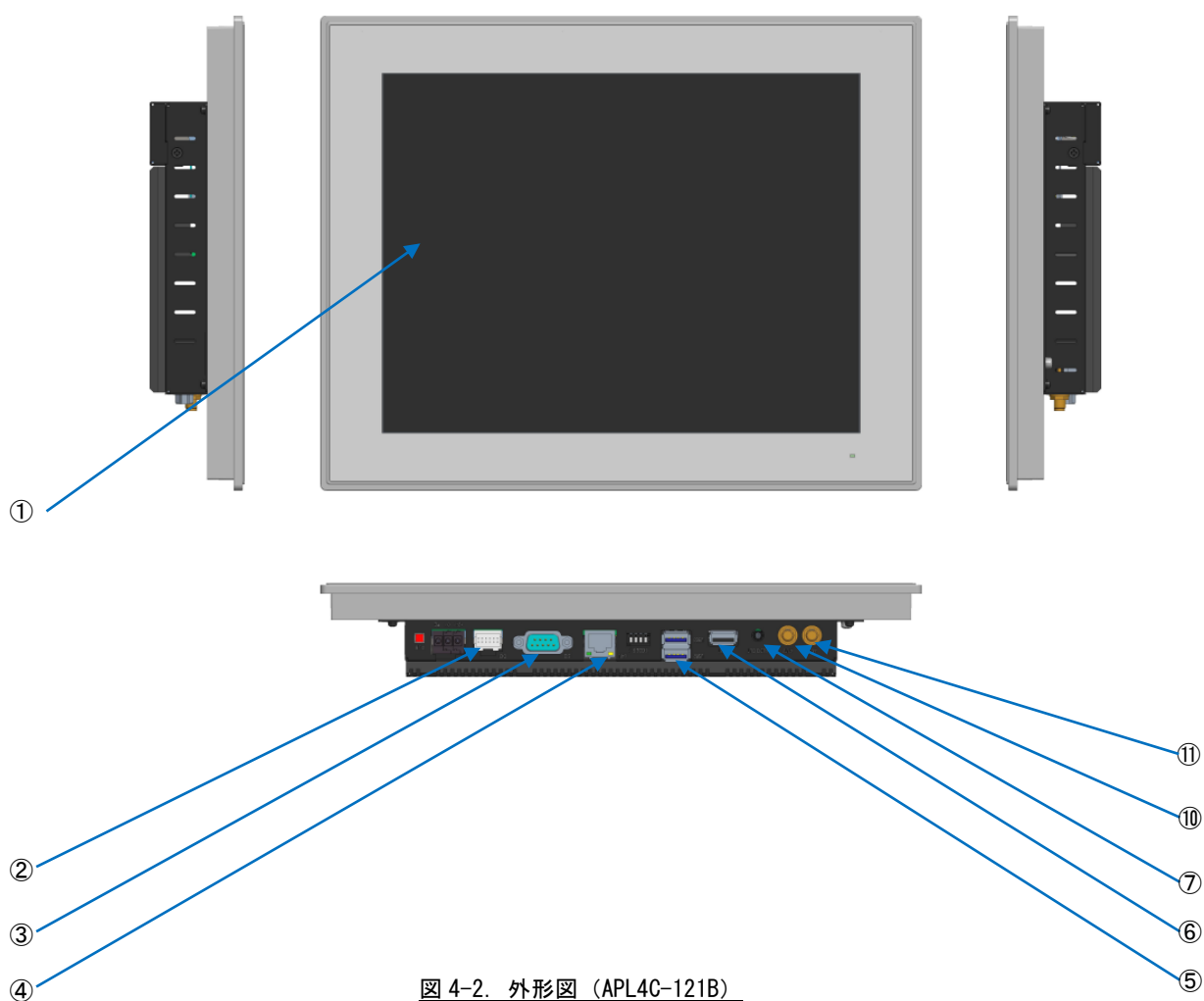


図 4-2. 外形図 (APL4C-121B)

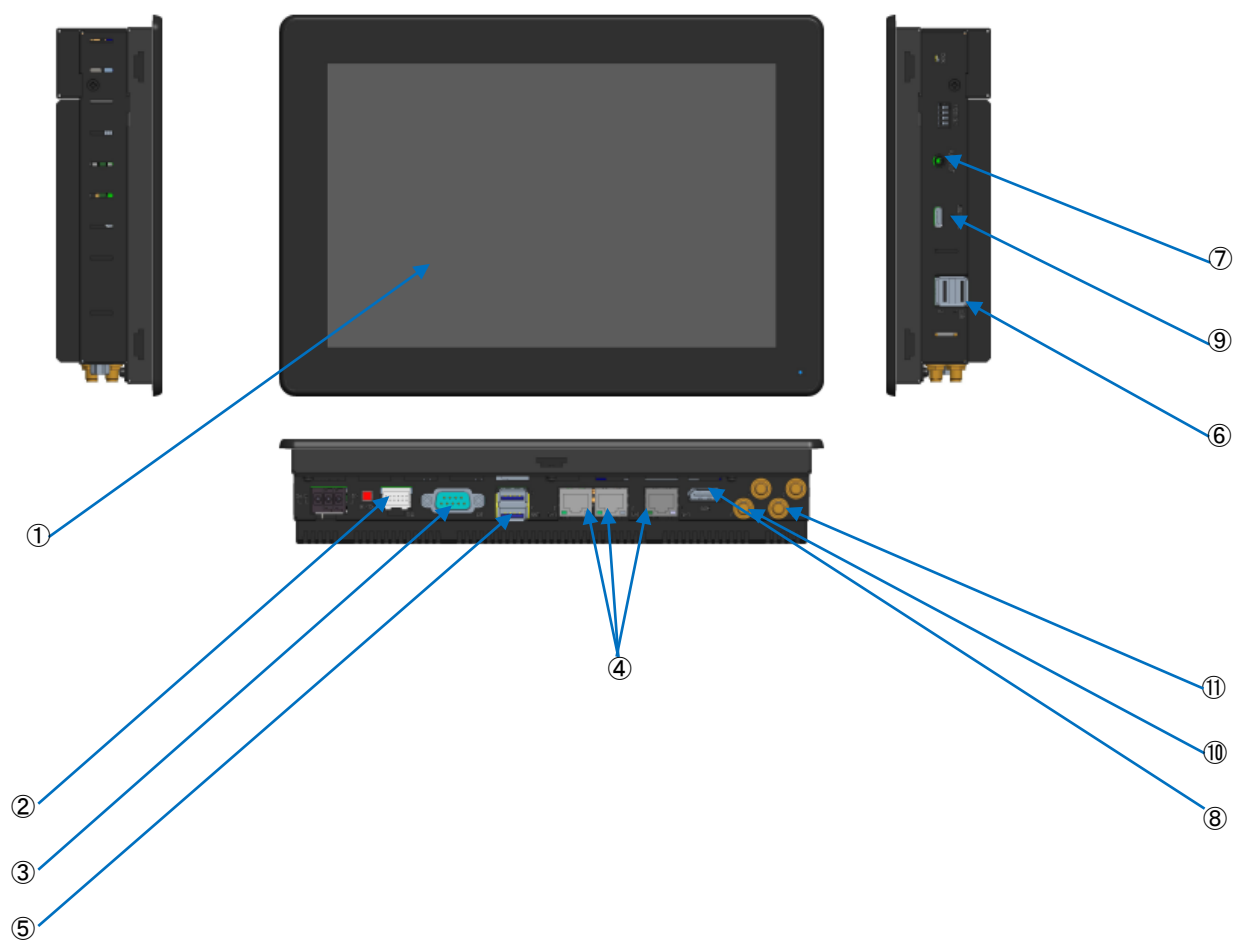


図 4-3. 外形図 (NP4C-101A)

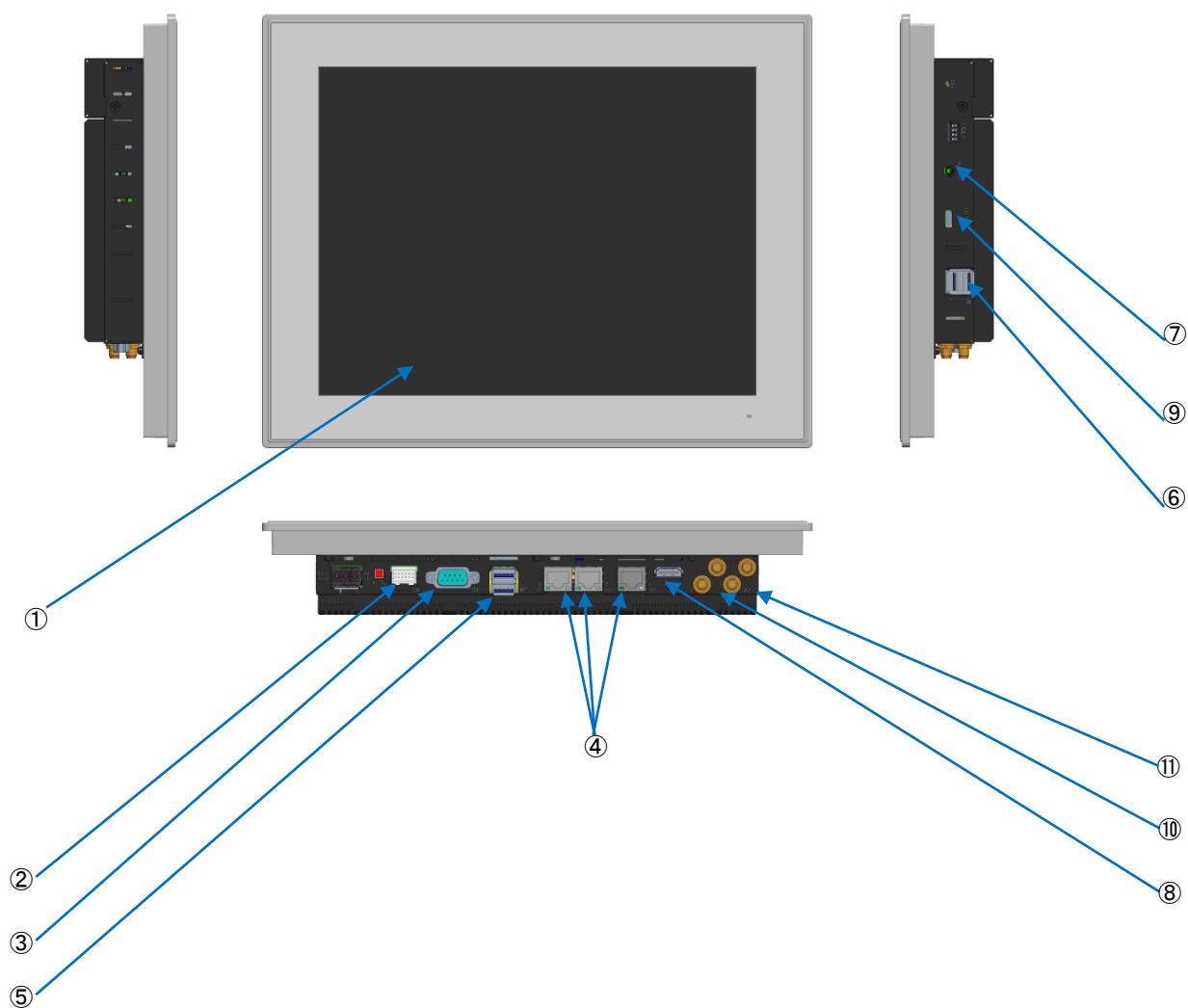


図 4-4. 外形図 (AP4C-121A)

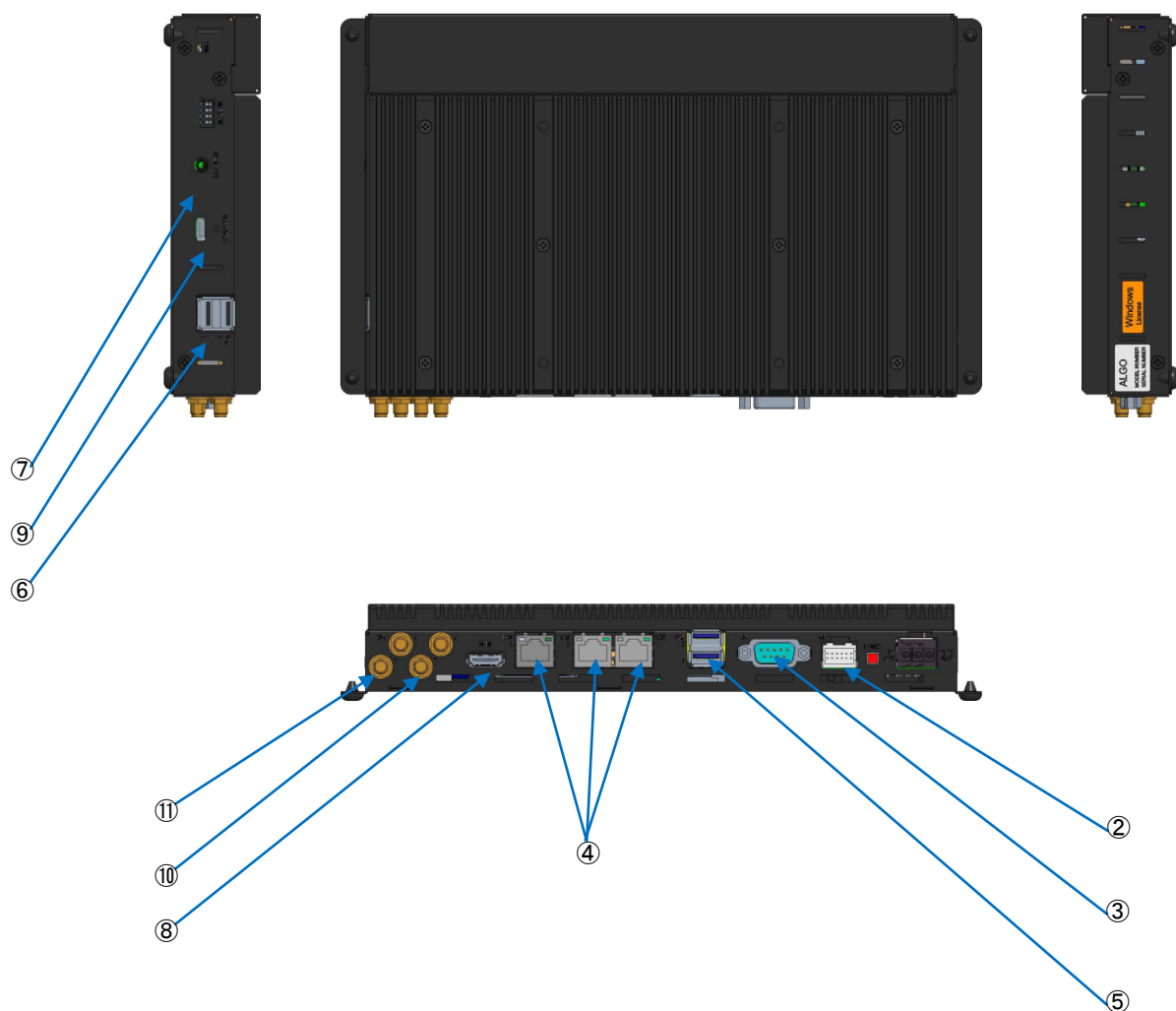


図 4-5. 外形図 (EC4C-100AT)

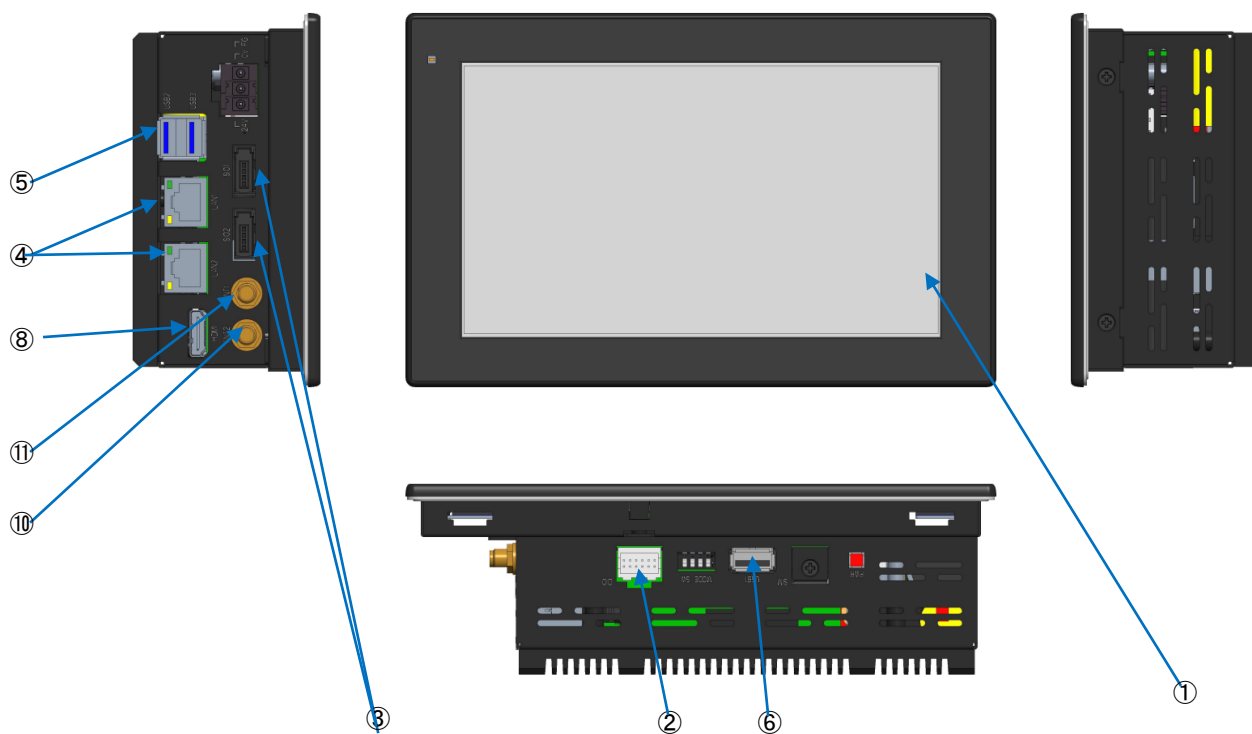


図 4-6. 外形図 (NPL4C-070B)

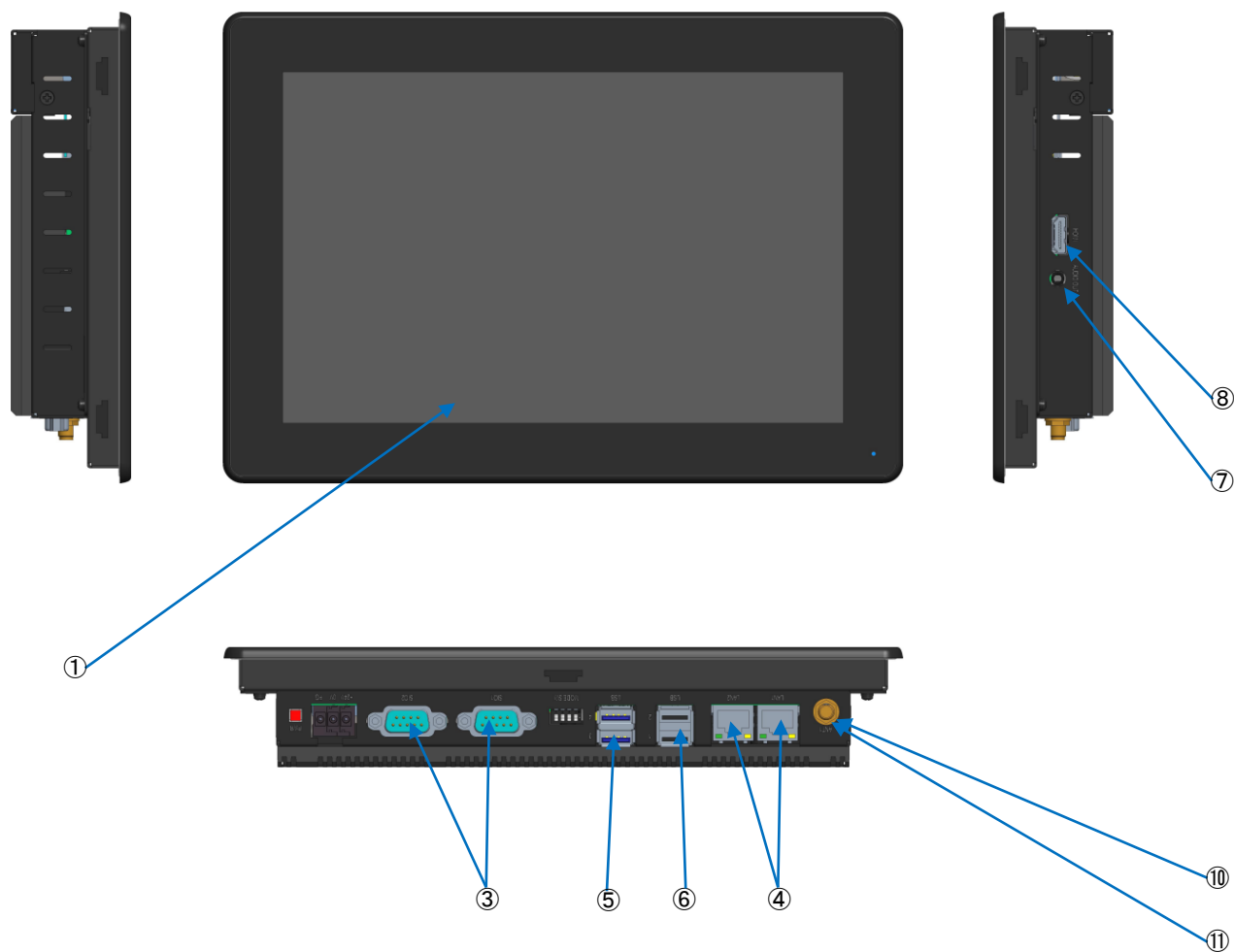


図 4-7. 外形図 (NPL4C-10HE)

表 4-2. 各部名称

No.	名称	機能	説明
①	液晶・タッチパネル	グラフィック タッチパネル	画面表示を行います。 タッチパネルはポインティングデバイスとして使用できます。
②	DIO インタフェース	汎用入出力	汎用の入出力です。 入力 6 点、出力 4 点を制御できます。
③	シリアル インタフェース	シリアルポート	シリアル通信が行えます。 シリアルポート設定を変更することで、RS232C/RS422/RS485 の通信の切替を行うことができます。 ※NPL4C/APL4C/AS4C シリーズは RS232C のみで RS422/RS485 通信に切り替えることができません。 SI01: ttyS0 SI02: ttyS1
④	ネットワーク インタフェース	有線 LAN	ネットワークポートとして使用できます。
⑤	USB3.1 インタフェース	USB3.1 ポート	USB1.1/2.0/3.1 の機器を接続することができます。
⑥	USB2.0 インタフェース	USB2.0 ポート	USB1.1/2.0 の機器を接続することができます。
⑦	サウンド	音声出力	音声出力が使用できます。
⑧	HDMI インタフェース	グラフィック サウンド	外部モニタに画面、音声を出力できます。
⑨	USB3.1 TypeC インターフェース	USB3.1 ポート	TypeC の USB 機器を接続することができます。
⑩	LTE(外部アンテナ) (オプション) LTE(内蔵アンテナ) (オプション)	LTE 通信	LTE 通信を使用することができます。 ※ NPL4C-070x/ECL4C-010x/NPL4C-10HE は無線 LAN と排他です。
⑪	無線 LAN (外部アンテナ) (オプション) 無線 LAN (内蔵アンテナ) (オプション)	無線 LAN	無線ネットワークデバイスとして使用できます。 ※ NPL4C-070x/ECL4C-010x/NPL4C-10HE は LTE 通信と排他です。

表 4-3. 各部名称

No.	名称	実装数			
		NPL4C-1xxB APL4C-1xxB AS4C-1xxC	NPL4C-070x ECL4C-010x	NPL4C-10HE	NP4C-1xxA AP4C-1xxA EC4C-100AT
①	液晶・タッチパネル	1	1	1	1
②	DIO インタフェース	1	1	－	1
③	シリアル インタフェース	1	2	2	1 「RS232C/RS422/R S485 切り替え」
④	ネットワーク インタフェース	1	2	2	3
	WakeOnLAN	1	－	－	－
⑤	USB3.1 インタフェース	2	2	2	2
⑥	USB2.0 インタフェース	1	1	2	2
⑦	サウンド	1	－	1	1
⑧	HDMI インタフェース	－	1	1	1
⑨	USB3.1 TypeC インターフェース	－	－	－	1
⑩	LTE(外部アンテナ)	(1)	(1)	(1)	(1)
	LTE(内蔵アンテナ)	(1)	(1)	(1)	(1)
⑪	無線 LAN(外部アンテナ)	(1)	(1)	(1)	(1)
	無線 LAN(内蔵アンテナ)	(1)	(1)	(1)	(1)

※ 実装数の () はオプションです。

4－1 汎用入出力

4－1－1 汎用入出力について

4C シリーズでは出力 4 点、入力 6 点を使用することができます。これらの入出力を使用することにより外部 I/O 機器を制御することが可能です。

4－1－2 汎用出力デバイスドライバについて

入力用のデバイスファイル (/dev/genin) と出力用のデバイスファイル (/dev/genout) に分かれています。キャラクタデバイス方式で入出力の状態を読み書きします。

表 4-1-2-1 に汎用入出力のリファレンスを示します。

表 4-1-2-1. 汎用出力デバイスリファレンス

GENIN, GENOUT																																					
名前																																					
genin	- 汎用入力6点																																				
genout	- 汎用出力4点																																				
説明																																					
汎用入力6点の状態読み出しと、汎用出力4点の制御を行います。																																					
OPEN																																					
汎用入力デバイス (/dev/genin, /dev/genout) をOpen関数でオープンします。 infd = open("/dev/genin", O_RDWR); outfs = open("/dev/genout", O_RDWR);																																					
READ																																					
read関数を用いて汎用入出力の状態をモニタすることができます。 char in; char out; len = read(infs, &in, 1); len = read(outfs, &out, 1);																																					
汎用入出力デバイスをリードすると1Byteのデータが即リターンされます。 リターンされた値は現在の入出力状態です。																																					
<table><tr><td>bit</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>IN</td><td></td><td></td><td>IN5</td><td>IN4</td><td>IN3</td><td>IN2</td><td>IN1</td><td>IN0</td></tr></table> <table><tr><td>bit</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>OUT</td><td></td><td></td><td></td><td></td><td>OT3</td><td>OT2</td><td>OT1</td><td>OT0</td></tr></table>		bit	7	6	5	4	3	2	1	0	IN			IN5	IN4	IN3	IN2	IN1	IN0	bit	7	6	5	4	3	2	1	0	OUT					OT3	OT2	OT1	OT0
bit	7	6	5	4	3	2	1	0																													
IN			IN5	IN4	IN3	IN2	IN1	IN0																													
bit	7	6	5	4	3	2	1	0																													
OUT					OT3	OT2	OT1	OT0																													
WRITE																																					
write関数を用いて汎用出力を制御することができます。 char out; out = 1; len = write(outfd, &out, 1);																																					
汎用出力データを1Byteライトすることで対応するビットの出力をON/OFFすることができます。																																					
<table><tr><td>bit</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>OUT</td><td></td><td></td><td></td><td></td><td>OT3</td><td>OT2</td><td>OT1</td><td>OT0</td></tr></table>		bit	7	6	5	4	3	2	1	0	OUT					OT3	OT2	OT1	OT0																		
bit	7	6	5	4	3	2	1	0																													
OUT					OT3	OT2	OT1	OT0																													

●コンソール用サンプルプログラム

「/usr/local/tools-x64/samples/sampleConsole/sample_DI0」に、汎用入出力を使ったサンプルプログラムが入っています。コンソールウィンドウを起動して、コンパイルしたコマンドを実行します。ソースコードをリスト 4-1-2-1 に示します。

まず、「open」関数で「/dev/genout」と「/dev/genin」をリードライトモードで開きます。汎用入出力には設定すべきモードは存在しない為、これでリード／ライトすることができます。

汎用入力デバイスを読み出すとき、1Byte 読み出すことができます。汎用入力の「read」関数は遅延せずに、汎用入力の ON/OFF 状態を即時に返します。サンプルプログラムのソースコードでは、前回値と比較して変化があったときのみコールバック関数を実行しています。

このサンプルプログラムを実行することで、汎用出力を順番に ON していき、現在の汎用入力の値をコンソール上に表示して終了します。

リスト 4-1-2-1. 汎用入出力 ON/OFF のソースコード (main.c)

```
/**
  汎用入出力制御サンプルプログラムのソースコード
**/
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    int          i;
    int          err_no;
    int          outfd;
    int          infd;
    unsigned char outdata=0x01;
    unsigned char indata=0x00;
    ssize_t      len;

    /* 汎用出力デバイスのオープン */
    outfd = open("/dev/genout", O_RDWR); /* 汎用出力デバイスオープン */
    if (outfd == -1) { /* エラー処理 */
        err_no = errno;
        fprintf(stderr, "out device open: %s\n", strerror(err_no));
        return(-1);
    }

    /* 汎用入力デバイスのオープン */
    infd = open("/dev/genin", O_RDONLY); /* 汎用入力デバイスオープン */
    if (infd == -1) { /* エラー処理 */
        close(infd);
        err_no = errno;
        fprintf(stderr, "out device open: %s\n", strerror(err_no));
        return (-1);
    }
}
```

```
}

/* 汎用出力
   汎用出力の 4 点を 1 点ずつ出力します。
*/
for (i=0; i<4; i++) {
    len = write(outfd, &outdata, 1);          /* 汎用出力更新 */
    if (len == -1) {
        err_no = errno;
        fprintf(stderr, "out device write: %s\n", strerror(err_no));
        /* 汎用出力デバイスのエラー */
        close(outfd);
        return (-1);
    }
    outdata <<= 1;
    usleep(500 * 1000L);
}

outdata = 0x00;
len = write(outfd, &outdata, 1);          /* 汎用出力すべて OFF */
if (len == -1) {
    err_no = errno;
    fprintf(stderr, "out device write: %s\n", strerror(err_no));
    /* 汎用出力デバイスのエラー */
    close(outfd);
    return (-1);
}

/* 汎用入力
   汎用入力の 6 点ずつ出力を行います。
*/
len = read(infd, &indata, 1);              /* 汎用入力の値を取得 */
if (len == -1) {
    err_no = errno;
    fprintf(stderr, "in device read: %s\n", strerror(err_no));
    /* 汎用入出力デバイスのエラー */
    close(outfd);
    close(infd);
    return (-1);
}
else if (len == 0) {
    fprintf(stderr, "in device read: len zero\n");
    /* 汎用入出力デバイスのエラー */
    close(outfd);
    close(infd);
    return (-1);
}
else {
    indata &= 0x3F;
    /* IN0 状態 */
    if (indata & 0x01) fprintf(stdout, "IN0: ON\n");
    else                fprintf(stdout, "IN0: OFF\n");
}
```

```
/* IN1 状態 */
if (indata & 0x02) fprintf(stdout, "IN1: ON¥n");
else                fprintf(stdout, "IN1: OFF¥n");
/* IN2 状態 */
if (indata & 0x04) fprintf(stdout, "IN2: ON¥n");
else                fprintf(stdout, "IN2: OFF¥n");
/* IN3 状態 */
if (indata & 0x08) fprintf(stdout, "IN3: ON¥n");
else                fprintf(stdout, "IN3: OFF¥n");
/* IN4 状態 */
if (indata & 0x10) fprintf(stdout, "IN4: ON¥n");
else                fprintf(stdout, "IN4: OFF¥n");
/* IN5 状態 */
if (indata & 0x20) fprintf(stdout, "IN5: ON¥n");
else                fprintf(stdout, "IN5: OFF¥n");
}

/* 汎用入出力デバイスのクローズ */
close(outfd);
close(infd);

return(0);
}
```

4-2 シリアルポート

4-2-1 シリアルポートについて

4C シリーズは、シリアルポートを最大 2 つ持っており、それぞれをユーザアプリケーションで使用できます。

ポートごとにデバイスファイルが違いますので、表 4-2-1-1 にシリアルタイプ別のデバイスファイル名について示します。

表 4-2-1-1. シリアルタイプ別のデバイスファイル名

ポート番号	デバイスファイル	用途
1	/dev/ttyS0	汎用のシリアルポート 1
2	/dev/ttyS1	汎用のシリアルポート 2

アプリケーションでシリアルポートを使用するにはそれぞれのデバイスファイルをオープンし、Read/Write することで制御します。

シリアルポートのタイプを切り替えるにはシリアルポート切り替えドライバを使用します。シリアルポートの切り替えは、下記に示す 2 種類の方法で行うことができます。

●コンソール用サンプルプログラム

「/usr/local/tools-x64/samples/sampleConsole/sample_SerialPortChange」に、シリアルポート切替えを行うサンプルプログラムが入っています。コンソールウィンドウを起動して、コンパイルしたコマンドを実行します。ソースコードをリスト 4-2-1-1 に示します。

このサンプルプログラムでは、シリアルポート 1 を RS485 に設定しています。

リスト 4-2-1-1. シリアルポート切替えのソースコード (main.c)

```
/**
 * シリアルポート RS422/RS485/RS232C 切り替え制御方法サンプルプログラムのソースコード
 */
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <sys/ioctl.h>
#include "scictl_ioctl.h"

int main(void)
{
    int          err_no;
    int          ret;
    int          desc;
    struct scictl_conf conf;

    /*
     * ポート切り替えデバイスをオープン
     */
    desc = open("/dev/scictl", O_RDWR);
    if (desc == -1) {
        err_no = errno;
        fprintf(stderr, "scictl open: %s\n", strerror(err_no));
        return (-1);
    }

    /*
     * ttyS0 を RS485 に変更
     */
    conf.ch
    *   ポート番号  1: ttyS0
    *               2: ttyS1
    conf.type
    *   ポートタイプ 0: RS232C
    *                 1: RS422
    *                 2: RS485
    *
    conf.timer
    *   RS485 用 TX ディセーブルタイム[ミリ秒]
```

```

*      送信完了から設定時間の間、再送信がなければ
*      TX がディセーブルされます。
*/
conf.ch = 1;
conf.type = 2;
conf.timer = 1000;
ret = ioctl(desc, ASP_SCICTL_IOCSCONF, &conf);
if (ret == -1) {
    err_no = errno;
    fprintf(stderr, "ioctl failed: %s\n", strerror(err_no));
    close(desc);
    return (-1);
}

close(desc);
return 0;
}

```

「/usr/local/tools-x64/samples/sampleConsole/sample_Serial」に、シリアルポートで送受信を行うサンプルプログラムが入っています。コンソールウィンドウを起動して、コンパイルしたコマンドを実行します。サンプルプログラムのソースコードをリスト 4-2-1-2 に示します。

シリアルポート 1「/dev/ttyS0」を「open」関数でオープンし、「tcsetattr」関数で通信設定を行っています。通信設定は 8bit 長、パリティ無し、ストップビット 1bit、ボーレート 38400bps と設定しています。「read」関数で 100 バイト受信されるまで待ち、コンソール上に受信した文字列を表示し、同時に受信した文字列を「write」関数で送信しています。

リスト 4-2-1-2. シリアルポート送受信を行うソースコード (main.c)

```

/**
 * シリアルポート制御サンプルソース
 */

#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <termios.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    char        rbuf[100];
    int         res;
    int         err_no;
    int         comm_fd;
    int         i;
    ssize_t     size;
    struct termios tio;

```

```
/* シリアルデバイスオープン */
comm_fd = open("/dev/ttyS0", O_RDWR | O_NOCTTY);
if (comm_fd == -1) { /* エラー処理 */
    err_no = errno;
    fprintf(stderr, "ttyS0 open: %s\n", strerror(err_no));
    return (-1);
}
/* 現在の通信設定を待避 */
res = tcgetattr(comm_fd, &tio);
if(res == -1){
    err_no = errno;
    fprintf(stderr, "ttyS0 tcgetattr_1: %s\n", strerror(err_no));
    close(comm_fd);
    return (-1);
}

/* 通信設定 (データ長 8bit ストップビット 1bit パリティ無し 制御線無視) */
tio.c_cflag &= ~(CSIZE | CSTOPB | PARENB | PARODD | HUPCL);
tio.c_cflag |= CS8 | CLOCAL | CREAD;

/* 通信設定 (フレームエラー、パリティエラーなし) */
tio.c_iflag = IGNPAR;
tio.c_oflag = 0;
tio.c_lflag = 0;

tio.c_cc[VINTR] = 0;
tio.c_cc[VQUIT] = 0;
tio.c_cc[VERASE] = 0;
tio.c_cc[VKILL] = 0;
tio.c_cc[VEOF] = 0;
tio.c_cc[VTIME] = 250; /* キャラクタ間タイムアウト時間 250 */
tio.c_cc[VMIN] = 1; /* 1 文字取得するまでブロック */
tio.c_cc[VSWTC] = 0;
tio.c_cc[VSTART] = 0;
tio.c_cc[VSTOP] = 0;
tio.c_cc[VSUSP] = 0;
tio.c_cc[VEOL] = 0;
tio.c_cc[VREPRINT] = 0;
tio.c_cc[VDISCARD] = 0;
tio.c_cc[VWERASE] = 0;
tio.c_cc[VLNEXT] = 0;
tio.c_cc[VEOL2] = 0;

/* 通信設定 (ボーレート 38400) */
cfsetospeed(&tio, B38400);
cfsetispeed(&tio, B38400);

/* 通信設定変更を反映 */
res = tcsetattr(comm_fd, TCSAFLUSH, &tio);
if (res == -1) {
    err_no = errno;
```



```
fprintf(stderr, "ttyS4 tcgetattr_2: %s\n", strerror(err_no));
close(comm_fd);
return (-1);
}

/* シリアルデータ受信処理 */
while (1){
    memset(rbuf, '¥0', 100);
    /* 100 バイト単位で受信 */
    size = read(comm_fd, &rbuf[0], 100);
    if (size == -1){
        err_no = errno;
        fprintf(stderr, "ttyS0 read: %s\n", strerror(err_no));
        break;
    }
    else if ( size > 0){
        /* 受信データを 16 進数文字に変換し標準出力 */
        fprintf(stdout, "ttyS0 read data: length[%d] data[", size);
        for (i = 0; i < size; i++) fprintf(stdout, "0x%02X ", rbuf[i]);
        fprintf(stdout, "]\n");

        /* 受信したデータ数を送信 */
        size = write(comm_fd, &rbuf[0], size);          /*size バイト送信*/
        if (size == -1){
            err_no = errno;
            fprintf(stderr, "ttyS0 write: %s\n", strerror(err_no));
            break;
        }
    }
    usleep(10*1000L);
}
return(0);
}
```

4-3 ネットワークポート

4-3-1 ネットワークポートについて

ネットワーク通信ではソケットと呼ばれる概念で通信します。ソケットには接続を待つサーバと、サーバに接続に行くクライアントがあります。サーバプログラムがまず起動され、接続を待ちます。次にクライアントプログラムを起動してサーバに接続にいきます。これでネットワーク通信が確立します。

4-3-2 ネットワークソケット用システムコールについて

表 4-3-2-1 にサーバ側のソケットシステムコール、表 4-3-2-2 にクライアント側ソケットシステムコールを示します。また、表 4-3-2-3 にサーバ、クライアント共通のソケット通信用システムコールを示します。

表 4-3-2-1. サーバ側ソケットシステムコール

関数名	説明
socket	ソケットを作成し、対応するファイルディスクリプタを返します。
bind	待ちポート番号を指定します。
listen	カーネルにサーバソケットであることを伝えます。
accept	クライアントが接続してくるまで待ちます。通信が確立したら、接続済みのファイルディスクリプタを返します。

表 4-3-2-2. クライアント側ソケットシステムコール

関数名	説明
socket	ソケットを作成し、対応するファイルディスクリプタを返します。
connect	指定された IP アドレスとポート番号のサーバに接続にいきます。

表 4-3-2-3. ソケット通信用システムコール

関数名	説明
recv	ソケットからデータを受信します。
send	ソケットからデータを送信します。

それぞれのシステムコール関数の詳細については、書籍やインターネットを参照してください。

これらのシステムコールを使用して、サーバプログラムとクライアントプログラムを作成することができ、ネットワークを利用して離れた場所にある機器と通信を行うことができます。

●コンソール用サンプルプログラム

「/usr/local/tools-x64/samples/sampleConsole/sample_TcpIp」に、ネットワーク通信を行うサンプルプログラムが入っています。コンソールウィンドウを起動して、コンパイルしたコマンドを実行します。

サーバプログラムとクライアントプログラムの起動手順は以下の通りです。

```
# ./sampleTCPIP_Server &  
# ./sampleTCPIP_Client -i 127.0.0.1 &
```

この場合、1 台の 4C シリーズ上でサーバとクライアントのプログラムが動作し、お互いに通信を行います。

ソースコードをリスト 4-3-2-1 に示します。

サーバソケットを作成し、「accept」関数でクライアントプログラムが接続してくるのを待ちます。正常に通信確立すれば、通信確立済みのソケットのファイルディスクリプタが返されます。通信確立済みのファイルディスクリプタを使用してデータの送受信を行います。送信は「send」関数を、受信は「recv」関数を使用します。

このプログラムでは、受信した文字列を画面に表示します。

リスト 4-3-2-1. サーバソケットのソースコード (main.c)

```
/**  
    TCP/IP サーバソケット サンプルプログラムのソースコード  
**/  
  
#include <arpa/inet.h>  
#include <fcntl.h>  
#include <errno.h>  
#include <stdio.h>  
#include <string.h>  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <sys/socket.h>  
#include <unistd.h>  
  
int main(void)  
{  
    char            rcv_buf[11];  
    int             i;  
    int             srv_sock;  
    int             len;  
    int             res;  
    int             err_no;  
    int             sock;  
    struct sockaddr_in cli_addr;  
    struct sockaddr_in srv_addr;  
  
    /* データ受信処理 */  
    while (1){  
  
        /* サーバソケット作成 */  
        srv_sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);  
        if (srv_sock == -1) {  
            err_no = errno;
```

```
fprintf(stderr, "socket failed: %s\n", strerror(err_no));
return (-1);
}

/* ポート番号指定 */
memset(&srv_addr, 0, sizeof(srv_addr));

srv_addr.sin_family      = AF_INET;
srv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
srv_addr.sin_port        = htons(8900);           //ポート番号指定

/* ソケットに名前をつける */
res = bind(srv_sock, (struct sockaddr *) &srv_addr, sizeof(srv_addr));
if (res == -1) {
    err_no = errno;
    fprintf(stderr, "bind failed: %s\n", strerror(err_no));
    close(srv_sock);
    return (-1);
}

/* カーネル通知 */
res = listen(srv_sock, 1);
if (res == -1) {
    err_no = errno;
    fprintf(stderr, "listen failed: %s\n", strerror(err_no));
    close(srv_sock);
    return (-1);
}

/* クライアント接続待ち */
len = sizeof(cli_addr);
srv_sock = accept(srv_sock, (struct sockaddr *)&cli_addr, (socklen_t *)&len);
if (srv_sock < 0) continue;

/* クライアントソケット接続待 */
while (1) {
    memset(rcv_buf, '¥0', 11);
    len = recv(srv_sock, &rcv_buf[0], 10, 0);
    if (len <= 0) {
        err_no = errno;
        fprintf(stderr, "recv failed: %s\n", strerror(err_no));
        close(srv_sock);
        break;
    }
    else {
        fprintf(stdout, "recv data: length[%d] [", len);
        for (i = 0; i < len; i++) fprintf(stdout, "0x%02X ", rcv_buf[i]);
        fprintf(stdout, "]\n");
    }
    usleep(10 * 1000L);
}
```

```

        usleep(10 * 1000L);
    }

    close(sock);
    return (0);
}

```

クライアント側のプログラムは、表 4-3-2-2 に書かれているシステムコールを実行して、接続待ちしているサーバに接続します。

リスト 4-3-2-2 にクライアントソケット作成を行うソースコードを示します。

「socket」関数でクライアント用のソケットのファイルディスクリプタを生成します。プログラム起動時に引数として、サーバの IP アドレスを指定します。指定した IP アドレスとサーバプログラムで指定したポート番号に「connect」関数で接続します。通信が確立したら 0 が返り、通信できる状態になります。エラーなら -1 が返されます。

リスト 4-3-2-2. クライアントソケット (main.c)

```

/**
 * TCP/IP クライアントソケット サンプルプログラムのソースコード
 */

#include <arpa/inet.h>
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    char*          srv_ip=NULL;
    char           snd_buf[] = "ABCDEFGH IJKLMNOPQRST";
    int             c;
    int             res;
    int             err_no;
    int             sock;
    struct sockaddr_in  srv_addr;

    /*
     * 起動引数取得
     *  -i : IP アドレスを指定します。
     */
    while ((c = getopt(argc, argv, "i:")) != -1) {
        switch(c) {
            case 'i':
                srv_ip = optarg;
                break;
            default:
                fprintf(stdout, "argument error\n");

```

```
        fprintf(stdout, " -i : Ip Address : ex) 192.168.0.1¥n");
        return (-1);
    }
}

if (srv_ip == NULL) {
    fprintf(stdout, "argument error Ip Address : ex) -i 192.168.0.1¥n");
    return (-1);
}

/* クライアントソケット作成 */
if ((sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
    fprintf(stderr, "socket failed¥n");
    return (-1);
}

/* サーバに接続 */
memset(&srv_addr, '¥0', sizeof(srv_addr));

srv_addr.sin_family      = AF_INET;
srv_addr.sin_addr.s_addr = inet_addr(srv_ip);      /* ターゲットサーバ IP アドレス */
srv_addr.sin_port        = htons(8900);            /* ターゲットサーバポート番号 8900port */

res = connect(sock, (struct sockaddr *)&srv_addr, sizeof(srv_addr));
if (res == -1) {
    err_no = errno;
    fprintf(stderr, "connect failed: %s¥n", strerror(err_no));
    close(sock);
    return (-1);
}

/* サーバにデータを送信 */
res = send(sock, &snd_buf, strlen(snd_buf), 0);
if (res == -1) {
    err_no = errno;
    fprintf(stderr, "send failed: %s¥n", strerror(err_no));
    close(sock);
    return (-1);
}
else if (res < strlen(snd_buf)) {
    fprintf(stderr, "send failed: send size(%d)¥n", res);
    close(sock);
    return (-1);
}

/* データを送信後待機 */
while (1) usleep(100 * 1000L);

close(sock);
return (0);
}
```

4－4 RAS 機能

RAS 機能として以下の様な機能を実装しています。

- 汎用入力 IN0 リセット
- ハードウェア・ウォッチドッグタイマ

次項より各機能についての説明を行います。

4－4－1 汎用入力 IN0 リセットについて

汎用入力 IN0 リセットは、汎用入力の IN0 が ON した場合に本体をリセットする機能です。
デバイスドライバからこの機能の有効・無効を切り替えることができます。

4－4－2 汎用入力 IN0 リセットデバイスドライバについて

汎用入力 IN0 リセットデバイスファイル(/dev/rasin)にアクセスする事により設定状態を読み書きできます。
表 4-4-2-1 に汎用入出力のリファレンスを示します。

表 4-4-2-1. 汎用入力 IN0 デバイスリファレンス

RASIN	
名前	汎用入力IN0リセット機能の制御を行います。
インクルード	#include "rasin.h"
説明	汎用入力IN0リセット機能の有効・無効設定及び設定値の取得を行います。 本設定は、本体再起動後に有効となります。
OPEN	汎用入力IN0リセットデバイス (/dev/rasin) をopen関数でオープンします。 rasinfd = open("/dev/rasin", O_RDWR);
IOCTL	IN0リセットを制御するにはioctl関数を使用します。 error = ioctl(fd, ioctl_type, &conf); ● ioctl_type RASIN_IOCINORST 有効・無効を設定します。 RASIN_IOCGINORST 現在の設定値を取得します。 ● conf 0 無効 1 有効

4-4-3 汎用入力 IN0 リセットサンプル

●コンソール用サンプルプログラム

「/usr/local/tools-x64/samples/sampleConsole/sample_Reset」に、汎用入力 IN0 リセットを使用するサンプルプログラムが入っています。このサンプルプログラムはコンソールアプリケーションとして作成されています。コンソール上で動作させることができます。リスト 4-4-3-1 にソースコードを示します。

リスト 4-4-3-1. 汎用入力 IN0 リセット (main.c)

```
/**
 * 汎用入力 IN0 リセット制御方法サンプルプログラムのソースコード
 */
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <unistd.h>

#include "rasin.h"

int main(void)
{
    int desc;
    int len;
    int onoff;
    int err_no;

    /*
     * RAS/Input デバイスのオープン
     */
    desc = open("/dev/rasin", O_RDWR);
    if (desc == -1) {
        err_no = errno;
        fprintf(stderr, "rasin open: %s\n", strerror(err_no));
        return (-1);
    }

    /*
     * 現在の設定を取得
     *
     * onoff: 1    有効
     *       : 0    無効
     */
    len = ioctl(desc, RASIN_IOCGINORST, &onoff);
    if (len == -1) {
        err_no = errno;
        fprintf(stderr, "ioctl get INORST failed: %s\n", strerror(err_no));
        close(desc);
        return (-1);
    }
}
```



```
}
else {
    fprintf(stdout, "ioctl get INORST success: %d¥n", onoff);
}

/*
 * INO リセットを有効にする
 *
 * onoff: 1    有効
 *       : 0    無効
 */
onoff = 1;
len = ioctl(desc, RASIN_IOCINORST, &onoff);
if (len == -1) {
    err_no = errno;
    fprintf(stderr, "ioctl set INORST failed: %s¥n", strerror(err_no));
    close(desc);
    return (-1);
}
else {
    fprintf(stdout, "ioctl set INORST success: %d¥n", onoff);
}
close(desc);
return 0;
}
```

4-4-4 汎用入力 IN1 割込みについて

汎用入力 IN1 割込みは、汎用入力の IN1 が ON した場合に割込みを発生させる機能です。デバイスドライバからこの機能の有効・無効を切り替えることができます。ユーザーアプリケーションでは SIGIO シグナルとして通知を受けることができます。

4-4-5 汎用入力 IN1 割込みデバイスドライバについて

汎用入力 IN1 割込みデバイスファイル (/dev/rasin) にアクセスする事により設定状態を読み書きできます。表 4-4-5-1 に汎用入出力のリファレンスを示します。

表 4-4-5-1. 汎用入力 IN1 割込みデバイスリファレンス

RASIN	
名前	汎用入力IN1割込み機能の制御を行います。
インクルード	#include "rasin.h"
説明	汎用入力IN1割込み機能の有効・無効設定及び設定値の取得を行います。 本設定は、本体再起動後に有効となります。
OPEN	汎用入力IN1割込みデバイス (/dev/rasin) をopen関数でオープンします。 rasinfd = open("/dev/rasin", O_RDWR);
IOCTL	IN1割込みを制御するにはioctl関数を使用します。 error = ioctl(fd, ioctl_type, &conf); ● ioctl_type RASIN_IOC SIN1INT 有効・無効を設定します。 RASIN_IOC GIN1INT 現在の設定値を取得します。 ● conf 0 無効 1 有効

4-4-6 汎用入力 IN1 割込みサンプル

●コンソール用サンプルプログラム

「/usr/local/tools-x64/samples/sampleConsole/sample_Interrupt」に、汎用入力 IN1 割込みを使用するサンプルプログラムが入っています。このサンプルプログラムはコンソールアプリケーションとして作成されています。コンソール上で動作させることができます。リスト 4-4-6-1 にソースコードを示します。

リスト 4-4-6-1. 汎用入力 IN1 割込み (main.c)

```
/**
 * 汎用入力 IN1 割込み制御サンプルプログラムのソースコード
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>

#include "rasin.h"

static int  SigOnOff=0;

void sighandler(int signo)
{
    /*
     * SIGIO シグナルなら終了
     */
    if(signo == SIGIO) {
        SigOnOff = 1;
    }
}

int main(void)
{
    int      err_no;
    int      desc;
    int      len;
    int      onoff;
    struct sigaction  action;

    /*
     * SIGIO シグナルのハンドラを登録
     */
    memset(&action, 0, sizeof(action));
    action.sa_handler = sighandler;
    action.sa_flags = 0;
    sigaction(SIGIO, &action, NULL);
```

```
/*
 * RAS/Input デバイスのオープン
 */
desc = open("/dev/rasin", O_RDWR);
if (desc == -1) {
    err_no = errno;
    fprintf(stderr, "rasin open: %s\n", strerror(err_no));
    return (-1);
}

/*
 * プロセスが SIGIO を受け取れるようにする
 */
fcntl(desc, F_SETOWN, getpid());
fcntl(desc, F_SETFL, fcntl(desc, F_GETFL) | FASYNC);

/*
 * 現在の設定値取得
 *
 * onoff: 1    有効
 *       : 0    無効
 */
len = ioctl(desc, RASIN_IOCTLIN1INT, &onoff);
if (len == -1) {
    err_no = errno;
    fprintf(stderr, "ioctl get IN1INT failed: %s\n", strerror(err_no));
    close(desc);
    return (-1);
}
else {
    fprintf(stdout, "ioctl get IN1INT success: %d\n", onoff);
}

/*
 * IN1 割込みを有効にする
 *
 * 有効の場合、無効に変更し終了
 * onoff: 1    有効
 *       : 0    無効
 */
if (onoff != 1) onoff = 1;
else onoff = 0;
len = ioctl(desc, RASIN_IOCTLIN1INT, &onoff);
if (len == -1) {
    err_no = errno;
    fprintf(stderr, "ioctl set IN1INT failed: %s\n", strerror(err_no));
    close(desc);
    return (-1);
}
else {
```

```
fprintf(stdout, "ioctl set IN1INT success: %d\n", onoff);
if (onoff == 0) {
    fprintf(stdout, "IN1 Interrupt off\n");
    return (1);
}
}
/*
 * シグナル (SIGIO) 受信時に標準出力に出力を行います。
 */
while (1) {
    if (SigOnOff == 1) {
        printf("IN1INT receive\n");
        break;
    }
    usleep(10*1000L);
}
close(desc);
return 0;
}
```

4-4-7 ハードウェア・ウォッチドッグタイマ機能について

4C シリーズには、ハードウェアによるウォッチドッグタイマ機能が搭載されています。デバイスドライバ (/dev/rashwwdt) にアクセスすることにより、この機能を使用することができます。シャットダウン、再起動、ポップアップ通知、ログへの出力は、デーモン (HWWDTD) によって行われています。図 4-4-7-1 に、デバイス、ドライバ、アプリケーション、設定ツール、デーモンの関係を示します。

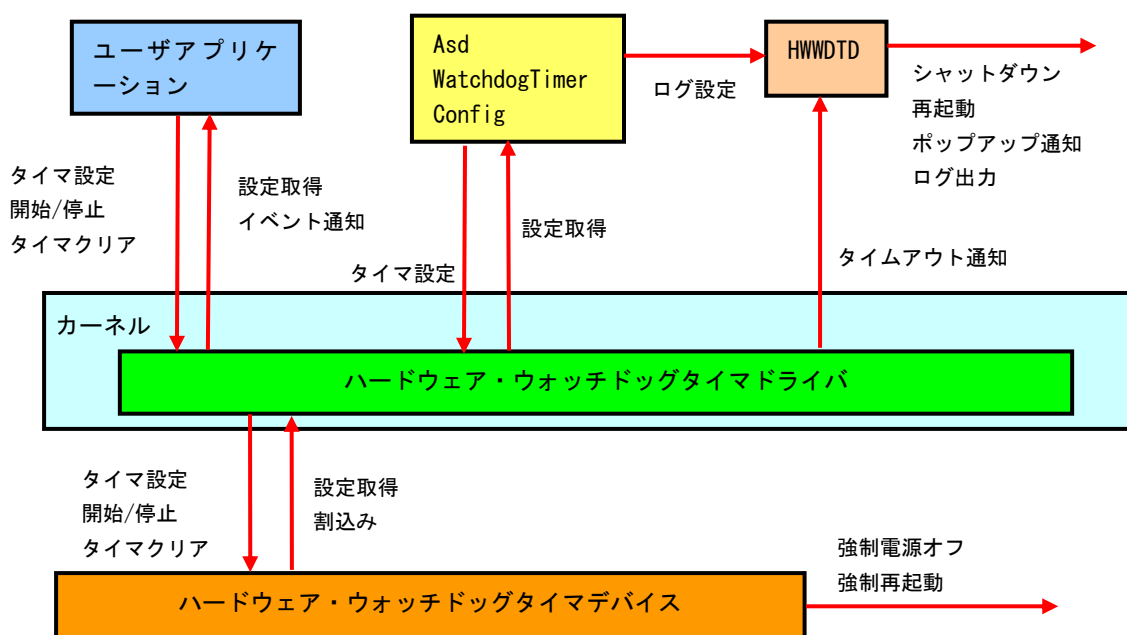


図 4-4-7-1. ハードウェア・ウォッチドッグタイマの構成

ハードウェア・ウォッチドッグタイマでは、ハード的に電源を OFF する機能（**強制電源オフ**）とリセットする機能（**強制再起動**）があります。OS が完全にフリーズした状態でも、電源 OFF やリセットを行うことが出来ます。

ハードウェア・ウォッチドッグタイマで**強制電源オフ**を設定した場合、「POWER ボタン」を長押ししたときと同じ処理になります。4C シリーズでは POWER ボタンを 4 秒間長押しすると、強制的に電源 OFF されます。また、Algonomix10 では、POWER ボタンを押されるとシャットダウン処理を実行します。

出荷時設定のままでは、シャットダウン処理中に電源が落ちることになり、M.2 ディスク破損の原因になる可能性があります。

下記のコマンドを実行して、「/etc/systemd/logind.conf」を編集してください。

```
$ sudo su
パスワード:asdsr
# vi /etc/systemd/logind.conf
```

編集前

```
# This file is part of systemd.
#
# systemd is free software; you can redistribute it and/or modify it
# under the terms of the GNU Lesser General Public License as published by
# the Free Software Foundation; either version 2.1 of the License, or
# (at your option) any later version.
#
```

```
# See logind.conf(5) for details

[Login]
#NAutoVTs=6
#ReserveVT=6
#KillUserProcesses=no
#KillOnlyUsers=
#KillExcludeUsers=root
Controllers=blkio cpu cpuacct cpuset devices freezer hugetlb memory perf_event net_cls net_prio
ResetControllers=
#InhibitDelayMaxSec=5
#HandlePowerKey=poweroff
#HandleSuspendKey=suspend
#HandleHibernateKey=hibernate
#HandleLidSwitch=suspend
#PowerKeyIgnoreInhibited=no
#SuspendKeyIgnoreInhibited=no
#HibernateKeyIgnoreInhibited=no
#LidSwitchIgnoreInhibited=yes
#IdleAction=ignore
#IdleActionSec=30min
```

編集後

```
# This file is part of systemd.
#
# systemd is free software; you can redistribute it and/or modify it
# under the terms of the GNU Lesser General Public License as published by
# the Free Software Foundation; either version 2.1 of the License, or
# (at your option) any later version.
#
# See logind.conf(5) for details

[Login]
#NAutoVTs=6
#ReserveVT=6
#KillUserProcesses=no
#KillOnlyUsers=
#KillExcludeUsers=root
Controllers=blkio cpu cpuacct cpuset devices freezer hugetlb memory perf_event net_cls net_prio
ResetControllers=
#InhibitDelayMaxSec=5
#HandlePowerKey=ignore
#HandleSuspendKey=suspend
#HandleHibernateKey=hibernate
#HandleLidSwitch=suspend
#PowerKeyIgnoreInhibited=no
#SuspendKeyIgnoreInhibited=no
#HibernateKeyIgnoreInhibited=no
#LidSwitchIgnoreInhibited=yes
#IdleAction=ignore
#IdleActionSec=30min
```

HandlePowerKey の先頭の「#」を外してコメントアウトを解除し、設定を「ignore」に変更することで、logind に ACPI イベントを無視させます。

この編集後は、POWER ボタンを押してもシャットダウン処理は実行しません。元に戻す場合は、logind.conf ファイルを編集前の状態に戻してください。

4-4-8 ハードウェア・ウォッチドッグタイマデバイスについて

ハードウェア・ウォッチドッグタイマデバイスファイル (/dev/rashwwdt) にアクセスすることにより、ハードウェア・ウォッチドッグタイマ機能进行操作します。表 4-4-8-1 に、ハードウェア・ウォッチドッグタイマの詳細を示します。

表 4-4-8-1. ハードウェア・ウォッチドッグタイマデバイスリファレンス

RASHWWDT	
名前	rashwwdt
インクルード	#include "asd_rashwwdt.h"
説明	ハードウェア・ウォッチドッグタイマの開始/停止、タイムアウト時の動作の設定、タイマ時間の設定、イベント通知待ちを行うことが出来ます。
OPEN	デバイスファイル(/dev/rashwwdt)をopen関数でオープンします。 int fd = open("/dev/rashwwdt", O_RDWR);
READ	使用しません
WRITE	1Byte以上のデータを書込むことで、タイマカウントをクリアします。 char data = `a`; size_t len = write(fd, &data, 1);

IOCTL

ハードウェア・ウォッチドッグタイマデバイス进行操作するには **ioctl** 関数を使用します。
`error = ioctl(fd, ctlcode, param);`

● **ctlcode****RASHWWDT_IOCTLSTART**

ハードウェア・ウォッチドッグタイマを開始します。
 また、param で、close 時にタイマカウントを停止するか開始、継続するかを指定します。
 param が 1 の時は、タイマが停止していても close 時に開始されます。
 0: タイマを停止、1: タイマを継続
 [param] unsigned long 型のポインタ
 [error] 0: 正常、0 以外: 異常

RASHWWDT_IOCTLSTOP

ハードウェア・ウォッチドッグタイマを停止します。
 [param] なし (NULL)
 [error] 0: 正常、0 以外: 異常

RASHWWDT_IOCSCONFIG

ハードウェア・ウォッチドッグタイマのタイムアウト時の動作とタイマ時間を設定します。
 設定を変更した場合、ハードウェア・ウォッチドッグタイマをオープンしているすべてのアプリケーションに影響を与えます。変更する場合は、他のアプリケーションがハードウェア・ウォッチドッグタイマをオープンしていないかどうかを確認してください。
 [param] RASHWWDT_CONFIG 型変数のポインタ
 [error] 0: 正常、0 以外: 異常

RASHWWDT_IOCSCONFIG

ハードウェア・ウォッチドッグタイマの設定を取得します。
 [param] RASHWWDT_CONFIG 型変数のポインタ
 [error] 0: 正常、0 以外: 異常

RASHWWDT_IOCQKEEPALIVE

タイマカウントをクリアします
 [param] なし (NULL)
 [error] 0: 正常、0 以外: 異常

RASHWWDT_IOCQWAKEUP

イベント通知を待ちます。
 このコントロールコードで ioctl を呼び出すとタイムアウト発生、または強制解除されるまで関数からリターンしません。
 [param] なし (NULL)
 [error] 1: イベント発生、2: 強制解除、その他: 異常

RASHWWDT_IOCQWAKEUP

イベント通知待ちを強制解除します。
 [param] なし (NULL)
 [error] 0: 正常、0 以外: 異常

● **param****RASHWWDT_CONFIG 構造体**

ハードウェア・ウォッチドッグタイマの設定を格納します。

```
struct RASHWWDT_CONFIG
{
    unsigned long action;    // タイムアウト時の動作  0: システム停止  1: システム再起動
                                2: ポップアップ  3: イベント通知
                                4: 強制電源OFF  5: 強制再起動
    unsigned long time;      // タイマ時間  (time × 100 【msec】)
};
```

4-4-9 ハードウェア・ウォッチドッグタイマサンプル

●コンソール用サンプルプログラム

「/usr/local/tools-x64/samples/sampleConsole/sample_HwWdtKeepalive」に、ハードウェア・ウォッチドッグタイマドライバを使用するサンプルプログラムが入っています。リスト 4-4-9-1 にサンプルプログラムのソースコードを示します。

リスト 4-4-9-1. ハードウェア・ウォッチドッグタイマデバイス操作ソースコード (main.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <sys/types.h>
#include "asd_rashwwdt.h"

#define HWWDTDEV_FILENAME "/dev/rashwwdt"

static struct RASHWWDT_CONFIG Config;
static unsigned long Sta_Type;
static volatile int fStart = 0;
static volatile int fStop = 0;
static volatile int fFinish = 0;
static pthread_t Thread_Id;

//-----
/*
 * キープレススレッド
 */
static void *TimerProc(void *arg)
{
    int wdtfd = 0;
    int status = 0;

    /*
     * ハードウェア・ウォッチドッグタイマデバイスのオープン
     */
    wdtfd = open(HWWDTDEV_FILENAME, O_RDWR);
    if(wdtfd <= 0) {
        fprintf(stderr, "TimerProc: Device open failed\n");
        exit(1);
    }

    /*
     * ハードウェア・ウォッチドッグタイマデバイスの設定
     */
    if(ioctl(wdtfd, RASHWWDT_IOCSCONFIG, &Config) != 0) {
        close(wdtfd);
        fprintf(stderr, "TimerProc: Device set error\n");
        exit(1);
    }
}
```

```

}
/*
 * 前回終了状態の取得
 */
status = ioctl(wdtfd, RASHWWDT_IOCSTATE, NULL);
if(status == RASHWWDT_NORMAL_SHUTDOWN) {
    fprintf(stdout, "Last Shutdown is normal shutdown\n");
}
if(status == RASHWWDT_FORCE_SHUTDOWN) {
    fprintf(stdout, "Last Shutdown is force shutdown\n");
}

/*
 * ハードウェア・ウォッチドッグタイマのスタート
 */
ioctl(wdtfd, RASHWWDT_IOCTSTART, &Sta_Type);
printf("TimerProc: Start\n");

fFinish = 0;
while(1) {
    if(!fStart) {
        break;
    }
    /* KeepAlive */
    if(!fStop) {
        ioctl(wdtfd, RASHWWDT_IOCQKEEPALIVE, NULL);
    }
    usleep(100 * 1000);
}

/*
 * ハードウェア・ウォッチドッグタイマ停止
 * ハードウェア・ウォッチドッグタイマ破棄
 */
ioctl(wdtfd, RASHWWDT_IOCTSTOP, NULL);
close(wdtfd);
printf("TimerProc: Stop\n");

fFinish = 1;

return 0;
}

//-----
static int CreateThread(void)
{
    pthread_attr_t thread_attr;

    pthread_attr_init(&thread_attr);
    pthread_attr_setstacksize(&thread_attr, 0x10000);

    fStart = 1;

```

```

    if(pthread_create(&Thread_Id, &thread_attr, TimerProc, NULL) != 0) {
        printf("pthread_create: error¥n");
        return -1;
    }

    return 0;
}

//-----
int main(int argc, char** argv)
{
    int c;
    int action;
    int wdt;
    int sta_type;

    wdt = -1;
    action = -1;
    sta_type = -1;

    while((c = getopt(argc, argv, "t:a:s:")) != -1) {
        switch(c) {
            case 't':
                wdt = atoi(optarg);
                break;
            case 'a':
                action = atoi(optarg);
                break;
            case 's':
                sta_type = atoi(optarg);
                break;
        }
    }

    if((wdt <= 0) || (action < 0) || (action > RASHWWDT_RESET) || (sta_type < 0) || (sta_type
> RASHWWDT_ENDCONTINUE)) {
        printf(" t:test time[ *100msec]¥n");
        printf(" a:action 0:Shutdown 1:Reboot 2:Popup 3:Event 4:PowerOff 5:Reset ¥n");
        printf(" s:sta_type 0:endstop 1:continue¥n");
        return -1;
    }

    Config.action = action;
    Config.time = wdt;
    Sta_Type = sta_type;

    CreateThread();

    while(1) {
        c = fgetc(stdin);
        if(c == 'q' || c == 'Q') {
            break;
        }
    }
}

```

```

    }
    if(c == 's' || c == 'S'){
        fStop = 1;
    }
    usleep(100 * 1000);
}
/*
 * スレッドを停止
 */
fStart = 0;
/*
 * スレッド停止待ち
 */
while(fFinish != 1){
    usleep(100 * 1000);
}
/*
 * スレッド破棄
 */
pthread_cancel(Thread_Id);

return 0;
}

```

ハードウェア・ウォッチドッグタイマのタイムアウト時の動作をイベント通知にした場合、ユーザアプリケーションでタイムアウトを取得することができます。

「/usr/local/tools-x64/samples/sampleConsole/sample_HwWdtEventWait」に、タイムアウトのイベント通知を取得するサンプルプログラムが入っています。リスト 4-4-9-2 に、サンプルプログラムのソースコードを示します。

リスト 4-4-9-2. ハードウェア・ウォッチドッグタイマイベント取得ソースコード (main.c)

```

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <stdarg.h>
#include <errno.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <bits/local_lim.h>
#include "asd_rashwwdt.h"

#define HWWDTDEV_FILENAME    "/dev/rashwwdt"
#define MAX_HWWDTEVENT      5

//-----

```

```
typedef struct {
    int No;
    volatile int fStart;
    volatile int fFinish;
    pthread_t ThreadID;
    int wdtfd;
} HWWDEVENT_INFO;

//-----
/*
 * タイマイベント処理スレッド
 */
static void *TimerEventProc(void *arg)
{
    HWWDEVENT_INFO *info = (HWWDEVENT_INFO *)arg;
    int ret;

    printf("TimerEventProc: Timer%02d: Start¥n", info->No);

    info->fFinish = 0;
    /*
     * イベント通知待ち
     */
    while(1) {
        if(!info->fStart) {
            break;
        }
        ret = ioctl(info->wdtfd, RASHWWDT_IOCQWAIТЕVENT, NULL);
        if(ret == RASHWWDT_TIMUP) {
            printf("TimerEventProc: TimeOut%02d¥n", info->No);
            break;
        } else {
            printf("TimerEventProc: WakeUp%02d¥n", info->No);
            break;
        }
    }
    info->fFinish = 1;

    printf("TimerEventProc: Timer%02d: Finish¥n", info->No);
    return 0;
}

//-----
static int CreateTimerEventInfo(int No, HWWDEVENT_INFO *info)
{
    pthread_attr_t thread_attr;

    pthread_attr_init(&thread_attr);
    pthread_attr_setstacksize(&thread_attr, 0x10000);

    info->No = No;
    info->ThreadID = 0;
}
```

```
info->fStart = 1;
info->fFinish = 0;
info->wdtfd = -1;

/*
 * ハードウェア・ウォッチドックタイマデバイスのオープン
 */
info->wdtfd = open(HWWDTDEV_FILENAME, O_RDWR);
if (info->wdtfd < 0) {
    printf("%s: open failed: err=%d\n", HWWDTDEV_FILENAME, errno);
    return -1;
}

/*
 * タイマイイベント処理スレッド作成
 */
if (pthread_create(&info->ThreadID, &thread_attr, TimerEventProc, (void *)info) != 0) {
    close(info->wdtfd);
    printf("pthread_create: error\n");
    return -1;
}

return 0;
}

//-----
static void DeleteTimer (HWWDTEVENT_INFO *info)
{
    /*
     * スレッド停止
     */
    info->fStart = 0;
    /*
     * タイマ停止
     * タイマイイベント待ち解除
     * タイマ破棄
     */
    ioctl(info->wdtfd, RASHWDT_IOCTLSTOP, NULL);
    ioctl(info->wdtfd, RASHWDT_IOCTLWAKEUP, NULL);
    close(info->wdtfd);
    /*
     * スレッド停止待ち
     */
    while (!info->fFinish) {
        usleep(10 * 1000);
    }

    pthread_cancel (info->ThreadID);
}

//-----
```

```
int main(int argc, char** argv)
{
    int i;
    HWWDEVENT_INFO info[MAX_HWWDEVENT];
    int action;
    int wdt;
    int sta_type;
    char c;

    wdt = 20;
    action = RASHWWDT_REBOOT;
    sta_type = RASHWWDT_ENDSTOP;

    for(i = 0; i < MAX_HWWDEVENT; i++) {
        if(CreateTimerEventInfo(i, &info[i]) != 0) {
            printf("CreatTimerEvent: NG: %d\n", i);
            return -1;
        }
    }
    while(1) {
        c = fgetc(stdin);
        if(c == 'q' || c == 'Q') {
            break;
        }
    }

    for(i = 0; i < MAX_HWWDEVENT; i++) {
        DeleteTimer(&info[i]);
    }

    return 0;
}
```


4-4-10 Wake On RTC 機能について

4C シリーズには RTC デバイスとして、EPOS0N 製 RX-8803LC という RTC チップが搭載されています。このチップには、通常のハードウェアクロック保持機能の他に、設定した時間で端末を起動させる機能を有しています。

WakeOnRTC の時刻設定は「2-3-6 ASD RAS Config について」の項目で説明したとおり、ASDConfig ツールで設定可能です。

また、本項で説明する、C 言語サンプルコードを使っても設定することが出来ます。

WakeOnRTC の設定を行うには、サンプルコード中の「ioctl.cpp」と「rtc_access.h」を追加して、表 4-4-10-1 に記述された関数を使って設定します。

表 4-4-10-1. Wake On RTC 設定関数一覧

表 4-1-10-1: Wake-On-RTC 設定関数一覧

関数名	引数	内容														
SetHour	時 (0~23)	時を設定します。														
GetHour	—	設定されている時を取得します。														
SetMin	分 (0~59)	分を設定します。														
GetMin	—	設定されている分を取得します。														
SetWeekSel	0: 曜日 1: 日	曜日設定か、日設定を行うのか切り替えます。														
GetWeekSel	—	現在の曜日/日の設定を取得します。														
SetDate	日 (1~31)	日を設定します。曜日指定と排他となります。														
GetDate	—	設定されている日を取得します。														
SetWeek	曜日 (bit 指定)	曜日指定します。複数設定可能です。 <table><tr><td>Bit6</td><td>Bit5</td><td>Bit4</td><td>Bit3</td><td>Bit2</td><td>Bit1</td><td>Bit0</td></tr><tr><td>土</td><td>金</td><td>木</td><td>水</td><td>火</td><td>月</td><td>日</td></tr></table>	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	土	金	木	水	火	月	日
Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0										
土	金	木	水	火	月	日										
GetWeek	—	設定されている曜日を取得します。														
EnableInt	0: 無効 1: 有効	WakeOnRTC 設定の有効/無効を設定します。														
ClearInt	—	WOR 発生フラグをクリアします。														

リスト 4-4-10-1 にソースコードを示します。現在時刻を読み出して、5 分後に WakeOnRTC が動作するように設定しています。WakeOnRTC が実行されるまでにシャットダウンさせておいてください。

リスト 4-4-10-1. WakeOnRTC 設定 (5 分後に起動設定)

```
#include <errno.h>
#include <pthread.h>
#include <semaphore.h>
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>

#include "rtc_access.h"

int main(void)
{
    time_t timer;
    struct tm *t_st;
```

```

/* 現在時刻の取得 */
time(&timer);

timer += 300; /*5 分後(300 秒後) に起動する設定*/

/* 現在時刻を構造体に変換 */
t_st = localtime(&timer);

EnableInt(1);          /* RTC 有効 */
SetHour(t_st->tm_hour); /* 時設定 */
SetMin(t_st->tm_min);   /* 分設定*/

/*日設定と曜日設定は排他です。必ずどちらかを設定してください。サンプルでは日設定で 5 分後に
起動する設定です。*/
#if 1
/*日設定*/
/*毎月※日に起動する設定にする場合、※の値を設定します。*/
SetWeekSel(1);          /* 日設定を選択 */
SetDate(t_st->tm_mday); /*日設定*/
#else
/*曜日設定*/
/*毎週※曜日に設定する場合は、起動したい曜日の bit を ON します。*/
/*bit   bit6 bit5 bit4 bit3 bit2 bit1 bit0 */
/*曜日   土   金   木   水   火   月   日 */
SetWeekSel(0); /* 曜日設定を選択 */
SetWeek(0x2A); /*月 水 金で起動 */
#endif

ClearInt(); /* RTC ステータスクリア */

printf("5 分後に WakeOnRTC が働きます。シャットダウンしておいてください。¥n");
}

```

4-4-11 S.M.A.R.T. 監視機能について

4C シリーズには、MainStorage/SubStorage である M.2 SSD の寿命監視機能が搭載されています。書き込み回数が警報値を超えた場合、または、BadBlock が検出されたときイベントが発報されます。

ユーザーアプリケーションは、それぞれのディスクのクリティカル警報とワーニング警報を受け取ることができます。警告の受け取りには、POSIX セマフォを用います。4C シリーズで利用できるセマフォを表 4-4-11-1 に示します。

表 4-4-11-1. S.M.A.R.T. 用 POSIX セマフォ名

セマフォ名称	用途
/smart1_critical	MainStorage クリティカル警報
/smart1_warning	MainStorage ワーニング警報
/smart2_critical	SubStorage クリティカル警報
/smart2_warning	SubStorage ワーニング警報

4-4-12 S.M.A.R.T. 機能サンプルプログラム

●コンソール用サンプルプログラム

「/usr/local/tools-x64/samples/sampleConsole/sample_SMART」に、S.M.A.R.T. 機能を使ったサンプルプログラムが入っています。コンソールウィンドウを起動して、コンパイルしたコマンドを実行します。ソースコードをリスト 4-4-12-1 に示します。

サンプルでは、まず「sem_open」関数でセマフォをオープンし、「sem_wait」関数で通知を待ちます。セマフォをクローズするには「sem_close」関数を呼びます。

リスト 4-4-12-1. S.M.A.R.T. 通知待ちサンプルソースコード (main.c)

```
#include <errno.h>
#include <pthread.h>
#include <semaphore.h>
#include <signal.h>
#include <stdio.h>
#include <unistd.h>

#include "asd_rassmart.h"

struct smart_t {
    int occurred;
    int thread_start;
    const char* message;
    sem_t* semaphore;
    pthread_t thread;
};

static void smart_close(struct smart_t* data);
static void* event_monitor(void* arg);

int main(void)
{
    int key;
    struct smart_t MainStrage_Critical_data = {
        .occured = 0,
        .thread_start = 0,
        .message = "MainStrage S.M.A.R.T Critical.¥n",
        .semaphore = SEM_FAILED,
    };
    struct smart_t MainStrage_Warning_data = {
        .occured = 0,
        .thread_start = 0,
        .message = "MainStrage S.M.A.R.T Warning.¥n",
        .semaphore = SEM_FAILED,
    };
    struct smart_t SubStrage_Critical_data = {
        .occured = 0,
        .thread_start = 0,
        .message = "SubStrage S.M.A.R.T Critical.¥n",
        .semaphore = SEM_FAILED,
    };
};
```

```
struct smart_t SubStrage_Warning_data = {
    .occured = 0,
    .thread_start = 0,
    .message = "SubStrage S. M. A. R. T Warning. ¥n",
    .semaphore = SEM_FAILED,
};

// open semaphore
MainStrage_Critical_data.semaphore = sem_open(DISK1SMARTCRITICAL_SEMAPHORE_PATH, 0);
if (MainStrage_Critical_data.semaphore == SEM_FAILED) {
    perror("MainStrage_Critical_data semaphore open failed");
    smart_close(&MainStrage_Critical_data);
    return -1;
}
MainStrage_Warning_data.semaphore = sem_open(DISK1SMARTWARNING_SEMAPHORE_PATH, 0);
if (MainStrage_Warning_data.semaphore == SEM_FAILED) {
    perror("MainStrage_Warning_data semaphore open failed");
    smart_close(&MainStrage_Critical_data);
    smart_close(&MainStrage_Warning_data);
    return -1;
}
SubStrage_Critical_data.semaphore = sem_open(DISK2SMARTCRITICAL_SEMAPHORE_PATH, 0);
if (SubStrage_Critical_data.semaphore == SEM_FAILED) {
    perror("SubStrage_Critical_data semaphore open failed");
    smart_close(&MainStrage_Critical_data);
    smart_close(&MainStrage_Warning_data);
    smart_close(&SubStrage_Critical_data);
    return -1;
}
SubStrage_Warning_data.semaphore = sem_open(DISK2SMARTWARNING_SEMAPHORE_PATH, 0);
if (SubStrage_Warning_data.semaphore == SEM_FAILED) {
    perror("SubStrage_Warning_data semaphore open failed");
    smart_close(&MainStrage_Critical_data);
    smart_close(&MainStrage_Warning_data);
    smart_close(&SubStrage_Critical_data);
    smart_close(&SubStrage_Warning_data);
    return -1;
}

// create threads
if (pthread_create(&MainStrage_Critical_data.thread, NULL, &event_monitor,
&MainStrage_Critical_data) != 0) {
    perror("MainStrage_Critical_data thread create failed");
    smart_close(&MainStrage_Critical_data);
    smart_close(&MainStrage_Warning_data);
    smart_close(&SubStrage_Critical_data);
    smart_close(&SubStrage_Warning_data);
    return -1;
}
if (pthread_create(&MainStrage_Warning_data.thread, NULL, &event_monitor,
```

```
&MainStrage_Warning_data) != 0) {
    perror("MainStrage_Warning_data thread create failed");
    smart_close(&MainStrage_Critical_data);
    smart_close(&MainStrage_Warning_data);
    smart_close(&SubStrage_Critical_data);
    smart_close(&SubStrage_Warning_data);
    return -1;
}
if (pthread_create(&SubStrage_Critical_data.thread, NULL, &event_monitor,
&SubStrage_Critical_data) != 0) {
    perror("SubStrage_Critical_data thread create failed");
    smart_close(&MainStrage_Critical_data);
    smart_close(&MainStrage_Warning_data);
    smart_close(&SubStrage_Critical_data);
    smart_close(&SubStrage_Warning_data);
    return -1;
}
if (pthread_create(&SubStrage_Warning_data.thread, NULL, &event_monitor,
&SubStrage_Warning_data) != 0) {
    perror("SubStrage_Warning_data thread create failed");
    smart_close(&MainStrage_Critical_data);
    smart_close(&MainStrage_Warning_data);
    smart_close(&SubStrage_Critical_data);
    smart_close(&SubStrage_Warning_data);
    return -1;
}

// wait key input
key = getchar();

smart_close(&MainStrage_Critical_data);
smart_close(&MainStrage_Warning_data);
smart_close(&SubStrage_Critical_data);
smart_close(&SubStrage_Warning_data);

return 0;
}

void smart_close(struct smart_t* p_smart)
{
    if (p_smart == NULL) {
        return;
    }
    if (p_smart->semaphore != SEM_FAILED) {
        sem_close(p_smart->semaphore);
        p_smart->semaphore = SEM_FAILED;
    }
    if (p_smart->thread_start) {
        p_smart->thread_start = 0;
        pthread_kill(p_smart->thread, SIGUSR1);
    }
}
```

```
void* event_monitor(void* arg)
{
    struct smart_t* data = (struct smart_t*)arg;

    data->thread_start = 1;

    while (1) {
        sem_wait(data->semaphore);
        printf("%s\n", data->message);
        usleep(100 * 1000);
    }

    data->thread_start = 0;
    return NULL;
}
```

4-4-13 バックアップ RAM 機能について

4C シリーズではバックアップ機能の RAM エリアが用意されています。

端末起動時、指定されたファイルからバックアップ RAM 領域に内容が読み出され、端末シャットダウン時に指定されたファイルにバックアップ RAM 領域の内容が書き込まれます。

バックアップ機能の有効/無効、バックアップファイルの保存先は、ASD UPS Config ツールで指定できます。指定方法については、『2-3-4 ASD Battery Config について』を参照してください。

4-4-14 バックアップ RAM 機能デバイスドライバについて

デバイスファイル(/dev/asd_sram)に対して Read/Write する事によりバックアップ RAM を読み書きできます。

表 4-4-14-1 にデバイスの詳細を示します。

表 4-4-14-1. バックアップ RAM デバイスリファレンス

RASRAM	
名前	バックアップRAMの制御を行います。
説明	バックアップRAMは、バックアップ機能付きのRAM領域です。 デバイスドライバからRAMのデータを読み書きできます。
OPEN	バックアップSRAMデバイス(/dev/asd_sram)をopen関数でオープンします。 <code>rasramfd = open("/dev/asd_sram", O_RDWR);</code>
READ	read関数を用いてバックアップRAMの値を読み込みます。
WRITE	write関数を用いてバックアップRAMに値を書込みます。
IOCTL	ioctl関数を用いてバックアップRAMの情報を読み出すことができます。 <code>error = ioctl(fd, ctlcode, param);</code> ● ctlcode SRAM_IOCFSIZE バックアップRAMのサイズを取得します。 [param] unsigned long型変数のポインタ [error] 0:正常、0以外:異常

4-4-15 バックアップ RAM 制御サンプル

●コンソール用サンプルプログラム

「/usr/local/tools-x64/samples/sampleConsole/sample_SRAM」に、バックアップ付き RAM を read/write システムコールで操作したサンプルプログラムが入っています。このサンプルプログラムはコンソールアプリケーションとして作成されています。コンソール上で動作させることができます。インクリメントデータの読み書きとデクリメントデータの読み書きを行い、データが正常に書き込まれているかを確認しています。リスト 4-4-15-1 にソースコードを示します。

リスト 4-4-15-1. バックアップ RAM 制御 Read/Write 方式 (rasram_read.c)

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>

#define RASRAM_SIZE 0x3ff000

unsigned char rasram[RASRAM_SIZE];

int main(void)
{
    int desc;
    int i;
    int len;
    char d;
    char *dp;

    desc = open("/dev/asd_sram", O_RDWR);
    if(desc < 0) {
        printf("open failed: err=%d\n", errno);
        return -1;
    }

    /* inc data */
    d = 1;
    dp = &rasram[0];
    for(i = 0; i < RASRAM_SIZE; i++) {
        *dp++ = d++;
    }
    lseek(desc, 0, SEEK_SET);
    len = write(desc, &rasram[0], RASRAM_SIZE);
    if(len != RASRAM_SIZE) {
        printf("inc data write failed: err=%d\n", errno);
        close(desc);
        return -1;
    }
    memset(&rasram[0], 0x00, RASRAM_SIZE);
    lseek(desc, 0, SEEK_SET);
```



```
len = read(desc, &rasram[0], RASRAM_SIZE);
if(len != RASRAM_SIZE) {
    printf("dec data read failed: err=%d\n", errno);
    close(desc);
    return -1;
}
d = 1;
dp = &rasram[0];
for(i = 0; i < RASRAM_SIZE; i++) {
    if(*dp++ != d++) {
        printf("inc data compare failed\n");
        close(desc);
        return -1;
    }
}

/* dec data */
d = 0xff;
dp = &rasram[0];
for(i = 0; i < RASRAM_SIZE; i++) {
    *dp++ = d--;
}
lseek(desc, 0, SEEK_SET);
len = write(desc, &rasram[0], RASRAM_SIZE);
if(len != RASRAM_SIZE) {
    printf("dec write failed: err=%d\n", errno);
    close(desc);
    return -1;
}
memset(&rasram[0], 0x00, RASRAM_SIZE);
lseek(desc, 0, SEEK_SET);
len = read(desc, &rasram[0], RASRAM_SIZE);
if(len != RASRAM_SIZE) {
    printf("dec read failed: err=%d\n", errno);
    close(desc);
    return -1;
}
d = 0xff;
dp = &rasram[0];
for(i = 0; i < RASRAM_SIZE; i++) {
    if(*dp++ != d--) {
        printf("dec data compare failed\n");
        close(desc);
        return -1;
    }
}

close(desc);
printf("read/write check ok\n");
return 0;
}
```

4-4-16 タイマ割込み機能について

4C シリーズには、ハードウェアによるタイマ割込み機能が実装されています。タイマ割込み機能を使用することで、指定した時間で周期的に割込みを発生させることができます。タイマドライバによって生成されるタイマデバイスを操作することによって、ユーザアプリケーションからタイマ割込み機能を利用できるようになります。タイマデバイスでは、タイマ設定、タイマイベント通知などの機能を使用することができます。

4-4-17 タイマ割込み機能デバイスについて

タイマドライバはタイマデバイスを生成します。ユーザアプリケーションは、デバイスファイルにアクセスすることによってタイマ機能を操作します。表 4-4-17-1 にタイマデバイスの詳細を示します。

表 4-4-17-1. タイマデバイスリファレンス

タイマデバイス	
名前	rastime
ヘッダ	#include "asd_rastime.h"
説明	タイマ時間設定、タイマ開始/停止、タイマイベント待機を行うことができます。
OPEN	デバイスファイル(/dev/rastime)をopen関数でオープンします。 システム全体で、同時に16までオープンすることができます。 timerfd = open("/dev/rastime", 0_RDWR);
READ	使用しません
WRITE	使用しません

IOCTL

タイマデバイスを操作するには **ioctl** 関数を使用します。

```
error = ioctl(fd, ctlcode, param);
```

● **ctlcode****RASTIME_IOCTLSTART**

タイマを開始します。

[param] なし (NULL)

[error] 0: 正常、0以外: 異常

RASTIME_IOCTLSTOP

タイマを停止します。

[param] なし (NULL)

[error] 0: 正常、0以外: 異常

RASTIME_IOCSCONFIG

タイマを設定します。

[param] RASTIME_CONFIG型変数のポインタ

[error] 0: 正常、0以外: 異常

RASTIME_IOCGETCONFIG

タイマ設定を取得します。

[param] RASTIME_CONFIG型変数のポインタ

[error] 0: 正常、0以外: 異常

RASTIME_IOCSTIMERRES

タイマ解像度を設定します。(設定値: 1~65535[msec]、初期値: 10[msec])

システムで使用されている全てのrastimeがこのタイマ解像度で動作します。

変更する場合は、他のrastimeが動作していないかどうか注意してください。

[param] unsigned long型変数のポインタ

RASTIME_IOCGETIMERRES

タイマ解像度を取得します。

[param] unsigned long型変数のポインタ

[error] 0: 正常、0以外: 異常

RASTIME_IOCQWAKEVENT

タイマイイベントを待ちます。

このコントロールコードで **ioctl** を呼び出すとタイマイイベント発生、または強制解除されるまで関数からリターンしません。

[param] なし (NULL)

[error] 1: イベント発生、2: 強制解除、その他: 異常

RASTIME_IOCQWAKEUP

タイマイイベント待ちを強制解除します。

[param] なし (NULL)

[error] 0: 正常、0以外: 異常

● **param****RASTIME_CONFIG構造体**

タイマの設定を格納します。

```
struct RASTIME_CONFIG
{
    unsigned long type;        // タイマタイプ 0:1回で終了, 1:繰り返し
    unsigned long duetime;    // タイマ時間 1~ [msec]
};
```

4-4-18 タイマ割込み機能サンプルプログラム

●コンソール用サンプルプログラム

「/usr/local/tools-x64/samples/sampleConsole/sample_Timer2」に、タイマデバイスを制御するサンプルプログラムが入っています。リスト 4-4-18-1 にサンプルプログラムのソースコードを示します。タイマ解像度の変更、タイマ設定、タイマ開始/停止、タイマイベント待ちを行うサンプルです。このサンプルプログラムでは 10 個のタイマを同時に使用しています。

リスト 4-4-18-1. タイマデバイス操作ソースコード (main.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <stdarg.h>
#include <errno.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <bits/local_lim.h>
#include "asd_rastime.h"

#define TIMERDEV_FILENAME    "/dev/rastime"
#define MAX_TIMEREVENT      2

//-----
typedef struct {
    int No;
    volatile int fStart;
    volatile int fFinish;
    pthread_t ThreadID;
    int TimerDesc;
    struct RASTIME_CONFIG Config;
} TIMEREVENT_INFO;

//-----
static void _time_print(const char *format, ...)
{
    struct timeval tv;
    struct tm *tmptr = NULL;
    char txt[256];
    va_list arg;

    va_start(arg, format);
    vsprintf(txt, format, arg);
    va_end(arg);

    gettimeofday(&tv, NULL);
```

```

    tmptr = localtime(&tv.tv_sec);
    printf("%02d:%02d:%02d.%03ld: %s",
        tmptr->tm_hour,          // hour
        tmptr->tm_min,           // min
        tmptr->tm_sec,           // sec
        (tv.tv_usec + 500) / 1000, // msec
        txt
    );
}

//-----
/*
 * タイマイイベント処理スレッド
 */
static void *TimerEventProc(void *arg)
{
    TIMEREVENT_INFO *info = (TIMEREVENT_INFO *)arg;
    int ret;

    printf("TimerEventProc: Timer%02d: Start¥n", info->No);

    info->fFinish = 0;
    while(1) {
        ret = ioctl(info->TimerDesc, RASTIME_IOCQWAIТЕVENT, NULL);
        if(ret != RASTIME_TIMER) {
            printf("ioctl: RASTIME_IOCQWAIТЕVENT: WAKEUP¥n");
            break;
        }
        if(!info->fStart) {
            break;
        }
        _time_print("Timer%02d¥n", info->No);
    }
    info->fFinish = 1;

    printf("TimerEventProc: Timer%02d: Finish¥n", info->No);
    return 0;
}

//-----
static int CreateTimerEventInfo(int No, TIMEREVENT_INFO *info)
{
    pthread_attr_t thread_attr;

    pthread_attr_init(&thread_attr);
    pthread_attr_setstacksize(&thread_attr, 0x10000);

    info->No = No;
    info->ThreadID = 0;
    info->fStart = 0;
    info->fFinish = 0;
}

```

```

info->TimerDesc = -1;

/*
 * タイマデバイスのオープン
 */
info->TimerDesc = open(TIMERDEV_FILENAME, O_RDWR);
if (info->TimerDesc < 0) {
    printf("%s: open failed: err=%d\n", TIMERDEV_FILENAME, errno);
    return -1;
}

/*
 * タイマの設定
 */
info->Config.type = 1;
info->Config.duetime = 200 * (No + 1);
if (ioctl(info->TimerDesc, RASTIME_IOCSCONFIG, &info->Config) != 0) {
    close(info->TimerDesc);
    printf("ioctl: RASTIME_IOCSCONFIG: error\n");
    return -1;
}

/*
 * タイマイベント処理スレッド作成
 */
info->fStart = 1;
if (pthread_create(&info->ThreadID, &thread_attr, TimerEventProc, (void *)info) != 0) {
    close(info->TimerDesc);
    printf("pthread_create: error\n");
    return -1;
}

return 0;
}

//-----
static void StartTimer(TIMEREVENT_INFO *info)
{
    /*
     * タイマスタート
     */
    ioctl(info->TimerDesc, RASTIME_IOCTLSTART, NULL);
}

//-----
static void DeleteTimer(TIMEREVENT_INFO *info)
{
    /*
     * スレッド停止
     */
    info->fStart = 0;

```

```

/*
 * タイマ停止
 * タイマイベント待ち解除
 * タイマ破棄
 */
ioctl(info->TimerDesc, RASTIME_IOCTLSTOP, NULL);
ioctl(info->TimerDesc, RASTIME_IOCQWAKEUP, NULL);
close(info->TimerDesc);

/*
 * スレッド停止待ち
 */
while(!info->fFinish) {
    usleep(10 * 1000);
}
pthread_cancel(info->ThreadID);
}

//-----
static int SetTimerResolution(void)
{
    int timerfd;
    unsigned long timer_resolution;

    /*
     * タイマデバイスのオープン
     */
    timerfd = open(TIMERDEV_FILENAME, O_RDWR);
    if(timerfd < 0) {
        printf("%s: open faild: err=%d\n", TIMERDEV_FILENAME, errno);
        return -1;
    }

    /*
     * タイマ解像度取得
     */
    if(ioctl(timerfd, RASTIME_IOCTLTIMERRES, &timer_resolution) != 0) {
        close(timerfd);
        printf("ioctl: RASTIME_IOCTLTIMERRES: error\n");
        return -1;
    }
    printf("SetTimerResolution: RASTIME_IOCTLTIMERRES: timer_resolution=%ld\n",
timer_resolution);

    /*
     * タイマ解像度変更
     *
     * タイマ解像度が変わります。
     * 他のプロセスで rastime を使用している場合は注意してください。
     */
}

```

```
    if(timer_resolution == 1000) {
        close(timerfd);
        return 0;
    }
    timer_resolution = 1000;
    if(ioctl(timerfd, RASTIME_IOCSTIMERRES, &timer_resolution) != 0) {
        close(timerfd);
        printf("ioctl: RASTIME_IOCSTIMERRES: error¥n");
        return -1;
    }
    printf("SetTimerResolution: RASTIME_IOCSTIMERRES: timer_resolution=%ld¥n",
timer_resolution);

    close(timerfd);
    return 0;
}

//-----
int main(void)
{
    int i;
    int c;
    TIMEREVENT_INFO info[MAX_TIMEREVENT];

    if(SetTimerResolution() != 0) {
        return -1;
    }

    for(i = 0; i < MAX_TIMEREVENT; i++) {
        if(CreateTimerEventInfo(i, &info[i]) != 0) {
            printf("CreatTimerEvent: NG: %d¥n", i);
            return -1;
        }
    }
    for(i = 0; i < MAX_TIMEREVENT; i++) {
        StartTimer(&info[i]);
    }

    while(1) {
        c = fgetc(stdin);
        if(c == 'q' || c == 'Q') {
            break;
        }
    }

    for(i = 0; i < MAX_TIMEREVENT; i++) {
        DeleteTimer(&info[i]);
    }

    return 0;
}
```


4-5 UPS 機能

4-5-1 UPS 機能について

4C シリーズには、UPS 機能が搭載されています。UPS 機能により、電源断が発生したときに、DC 電源駆動からバッテリー駆動に切り替わり、安全にデータ退避、シャットダウン処理を行うことができます。

ユーザーアプリケーションは、バッテリー保護機能動作の通知、電源断の警告を受け取ることができます。通知、警告の受け取りには、POSIX セマフォを用います。4C シリーズで利用できるセマフォを表 4-5-1-1 に示します。

表 4-5-1-1. UPS 用 POSIX セマフォ名

セマフォ名称	用途
/ups_notification	バッテリー保護機能動作通知
/ups_alarm	電源断警告

バッテリー保護機能動作通知、電源断警告は、機能が有効に設定されていなければ発生しません。また、バッテリー保護機能動作/電源断が発生してから、実際に通知が行われるまでの時間は設定可能です。詳細は、『2-3-4 ASD Battery Config について』を参照してください。

バッテリー保護機能はバッテリー温度が 4~67℃の範囲外になると動作します。

4-5-2 UPS 機能サンプルプログラム

●コンソール用サンプルプログラム

「/usr/local/tools-x64/samples/sampleConsole/sample_UPS」に、UPS 機能を使ったサンプルプログラムが入っています。コンソールウィンドウを起動して、コンパイルしたコマンドを実行します。ソースコードをリスト 4-5-2-1 に示します。

サンプルでは、まず「sem_open」関数でセマフォをオープンし、「sem_wait」関数で通知を待ちます。セマフォをクローズするには「sem_close」関数を呼びます。

リスト 4-5-2-1. UPS 通知待ちサンプルソースコード (main.c)

```
#include <errno.h>
#include <pthread.h>
#include <semaphore.h>
#include <signal.h>
#include <stdio.h>
#include <unistd.h>

#define NOTIFICATION_SEMAPHORE_PATH "/ups_notification"
#define ALARM_SEMAPHORE_PATH      "/ups_alarm"

struct ups_t {
    int occurred;
    int thread_start;
    const char* message;
    sem_t* semaphore;
    pthread_t thread;
};

static void ups_close(struct ups_t* data);
static void* event_monitor(void* arg);

int main(void)
```

```
{
    int key;
    struct ups_t notification_data = {
        .occured = 0,
        .thread_start = 0,
        .message = "UPS error occurred.¥n",
        .semaphore = SEM_FAILED,
    };
    struct ups_t alarm_data = {
        .occured = 0,
        .thread_start = 0,
        .message = "Power down detected.¥n",
        .semaphore = SEM_FAILED,
    };

    // open semaphore
    notification_data.semaphore = sem_open(NOTIFICATION_SEMAPHORE_PATH, 0);
    if (notification_data.semaphore == SEM_FAILED) {
        perror("notification semaphore open failed");
        ups_close(&notification_data);
        return -1;
    }
    alarm_data.semaphore = sem_open(ALARM_SEMAPHORE_PATH, 0);
    if (alarm_data.semaphore == SEM_FAILED) {
        perror("alarm semaphore open failed");
        ups_close(&notification_data);
        ups_close(&alarm_data);
        return -1;
    }

    // create threads
    if (pthread_create(&notification_data.thread, NULL, &event_monitor,
&notification_data) != 0) {
        perror("notification thread create failed");
        ups_close(&notification_data);
        ups_close(&alarm_data);
        return -1;
    }
    if (pthread_create(&alarm_data.thread, NULL, &event_monitor, &alarm_data) != 0) {
        perror("alarm thread create failed");
        ups_close(&notification_data);
        ups_close(&alarm_data);
        return -1;
    }

    // wait key input
    key = getchar();

    ups_close(&notification_data);
    ups_close(&alarm_data);

    return 0;
}
```

```
void ups_close(struct ups_t* p_ups)
{
    if (p_ups == NULL) {
        return;
    }
    if (p_ups->semaphore != SEM_FAILED) {
        sem_close(p_ups->semaphore);
        p_ups->semaphore = SEM_FAILED;
    }
    if (p_ups->thread_start) {
        p_ups->thread_start = 0;
        pthread_kill(p_ups->thread, SIGUSR1);
    }
}

void* event_monitor(void* arg)
{
    struct ups_t* data = (struct ups_t*)arg;

    data->thread_start = 1;

    while (1) {
        sem_wait(data->semaphore);
        printf("%s\n", data->message);
        usleep(100 * 1000);
    }

    data->thread_start = 0;
    return NULL;
}
```

4-6 ストレージデバイスについて

ここでは、外部ストレージデバイスの使用方法について説明します。

4-6-1 外部ストレージデバイスの使用方法

外部ストレージデバイスはブロックデバイスを使用して通常のディスクと同様に操作することができます。Algonomix10 の初期設定では、外部ストレージデバイスは自動的にマウントされます。手動でマウントする必要がある場合に、本項目を参照してください。

●外部ストレージデバイスのマウント

外部ストレージデバイスのブロックデバイスをマウントします。(/dev/sdb と認識した場合)

例：外部ストレージデバイスのパーティション 1 をマウント

```
# mount /dev/sdb1 /mnt
```

●外部ストレージデバイスのファイルへのアクセス

マウント以後は、マウントしたディレクトリから外部ストレージデバイス内のファイルにアクセスができます。ファイル操作方法は、ルートファイルシステム上のファイルと同様です。

例：外部ストレージデバイス内ファイル一覧表示

```
# cd /mnt
# ls
file1    file2    file3
```

例：外部ストレージデバイスへのファイルコピー

```
# cp /home/asdusr/FILE /mnt
```

●外部ストレージデバイスのアンマウント

外部ストレージデバイスを使用し終わったらブロックデバイスをアンマウントします。外部ストレージデバイスを抜く前に必ずアンマウントを行ってください。

例：外部ストレージデバイスのアンマウント

```
# umount /mnt
```

※注：外部ストレージデバイスのデバイス名 (/dev/sdb など) については『4-6-2 ストレージデバイス名の割り振りについて』を参照してください。

4-6-2 ストレージデバイス名の割り振りについて

各ストレージデバイスは Linux 上で使用するためにデバイス名が以下のような名前でも割り振られます。

- /dev/sd x : ストレージデバイス全体のブロックデバイス
 x は a, b, c, d... と認識順に増えていきます
 /dev/sd x * : ストレージデバイス上パーティションのブロックデバイス
 $*$ はパーティション毎に 1, 2, 3... と増えていきます。
 (例: /dev/sdb1, /dev/sdb2)

各ストレージデバイスは以下のルールに従い、デバイス名にアルファベットの若い文字から割り振られます。

- (1) OS 起動時に各ストレージデバイススロットにデバイスが挿入されていた場合、接続されているデバイスが表 4-6-2-1 の優先順位に従ってデバイス名が割り振られます。(接続されていない場合はスキップします。)
- (2) OS 起動後、新たに USB メモリなどを挿入した場合、それらの USB メモリにデバイス名が割り振られます。

4C シリーズでの認識優先度を表 4-6-2-1 に示します。

表 4-6-2-1. ストレージデバイス認識優先度

優先度	スロット名
①	M.2 メインストレージ
②	M.2 サブストレージ
③	USB スロット 0~3

※注: USB スロットや SSD スロットに複数のストレージを接続したときは、それぞれにデバイス名が若いアルファベット文字から割り振られます。

例 1: M.2 メインストレージのみ挿入して OS 起動したとき

M.2 メインストレージが sda として認識されます。その後挿入された USB メモリは sdb, sdc... として認識されます。

表 4-6-2-2. デバイス名割り振り

スロット名	デバイス名
M.2 メインストレージ	/dev/sda
USB スロット 0~3	/dev/sdb, /dev/sdc...

例 2: M.2 メインストレージと M.2 サブストレージを挿入して OS 起動したとき

M.2 メインストレージが sda として、M.2 サブストレージが sdb として認識されます。その後挿入された USB メモリは sdc, sdd... として認識されます。

表 4-6-2-3. デバイス名割り振り

スロット名	デバイス名
M.2 メインストレージ	/dev/sda
M.2 サブストレージ	/dev/sdb
USB スロット 0~3	/dev/sdc, /dev/sdd...

4-6-3 外部ストレージデバイスの起動時マウントについて

ここでは、外部ストレージデバイス (USB メモリなど) を、起動時に任意のディレクトリにマウントする方法について記述します。

Linux では、ストレージのマウント情報は/etc/fstab というファイルに記述されます。/etc/fstab を編集することで、起動時にデバイスをマウントすることができます。

以下に、USB メモリを/home/asdusr/usb ディレクトリにマウントする例を示します。マウント先のディレクトリは、mkdir コマンドで事前に作成する必要があります。

リスト 4-6-3-1. SD カードの起動時マウント設定例 (/etc/fstab)

```
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point> <type> <options>          <dump> <pass>
# / was on /dev/sda2 during installation
UUID=0b45b6fb-1646-4d70-bd5b-6e221decd535 /          ext4    noatime,errors=remount-ro 0
0
# /boot was on /dev/sda1 during installation
UUID=372c52f7-6bea-474c-98f3-101637bb047e /boot      ext4    noatime,defaults          0
0
# /home was on /dev/sda3 during installation
UUID=41e73ccd-2280-4e91-8018-0644747ec214 /home      ext4    noatime,defaults          0
0
# /usr/local was on /dev/sda4 during installation
UUID=8a2f8345-1607-4ef0-8a5e-ad951606bacc /usr/local ext4    noatime,defaults          0
0

tmpfs    /tmp          tmpfs    defaults,size=192m 0    0
tmpfs    /var/log      tmpfs    defaults,size=64m  0    0
/dev/    /home/asdusr/usb vfat     defaults,nofail 0    0
```

この行を追加

4-7 LTE について

LTE 機能を使用して LTE 通信を行うことができます。

- ① コンソールを起動させ、下記のコマンドを実行します。

```
# vi /etc/wvdial.conf
```

- ② wvdial.conf ファイルが開くので以下のように編集します。

リスト 4-9-1. wvdial.conf の設定 (/etc/ wvdial.conf)

```
[Dialer Defaults]
Init1 = ATZ
Init2 = ATQ0 V1 E1 S0=0 &C1 &D2 +FCLASS=0
Init3 = AT+CGDCONT=1,"IP","netwireless.com"
Dial Attempts = 3
Modem Type = Analog Modem
Dial Command = ATD
Stupid Mode = yes
Baud = 460800
New PPPD = yes
Modem = /dev/ttyLTE2
ISDN = 0
APN = netwireless.com
Phone = *99***1#
;Username = <Set Username>
;Password = <Set Password>
Carrier Check = no
Auto DNS = 1
Check Def Route = 1
```

この行を変更

「Username」と「Password」の先頭の「;」を削除して、「<Set Username>」、「<Set Password>」の部分を SIM カードに付属していた設定に変更してください。

3. 以下のコマンドを実行します。

```
# wvdial
```

wvdial コマンド実行後、LTE 通信が確立します。

wvdial プロセスを終了させることで LTE 通信を切断することができます。

**注) wvdial プロセスが起動している間は LTE 通信したままとなります。
この場合、通信料が発生し続けるのでご注意ください。**

第 5 章 システムリカバリ

本章では、「産業用 PC 4C シリーズ用 Algonomix10 リカバリ DVD」を使用したシステムのリカバリとバックアップについて説明します。

5-1 リカバリ DVD について

リカバリ DVD は 3 枚構成です。

	フォルダ名	ファイル or サブフォルダ名	備考
DVD1	DOC	産業用パネル PC 4C シリーズ _Algonomix10 実行環境ユーザー ズマニュアル.pdf	本書です。
	Recovery	DDWin	DD for Windows ツール
		ReadMe.txt	USB リカバリーイメージ書き込み方 ReadMe.txt ファイル
		RecoveryUSBImage.zip	バックアップ取得用リカバリ USB イメージ
		ReleaseRecoveryUSBImage.exe	出荷イメージリカバリ USB イメージ結合用 exe ファイル
		ReleaseRecoveryUSBImage.7z.003	出荷イメージリカバリ USB イメージ結合用 003 ファイル
DVD2	Recovery	ReleaseRecoveryUSBImage.7z.001	出荷イメージリカバリ USB イメージ結合用 001 ファイル
DVD3	Recovery	ReleaseRecoveryUSBImage.7z.002	出荷イメージリカバリ USB イメージ結合用 002 ファイル

4C シリーズ本体は、システムのリカバリを行うことができます。リカバリで行える処理は以下のとおりです。

1. 出荷イメージリカバリ
 - 出荷イメージ書き込み
 2. システムのバックアップおよび復旧
 - システムのバックアップ
 - システムの復旧（バックアップデータ）
1. 出荷イメージリカバリ
- リカバリを実行するにはリカバリ DVD 以外に以下のものを用意する必要があります。
- 16GByte 以上の USB メモリ（リカバリ USB 用）
2. システムのバックアップおよび復旧
- システムのバックアップおよび復旧を実行するにはリカバリ DVD 以外に以下のものを用意する必要があります。
- 8GByte 以上の USB メモリ（リカバリ USB 用）
 - 8GByte 程度の空き容量がある USB 接続可能なストレージメディア（USB メモリなど）
（バックアップイメージ保存用）

リカバリ DVD を実行する場合、BIOS 設定が必要となります。

● リカバリ実行の流れ

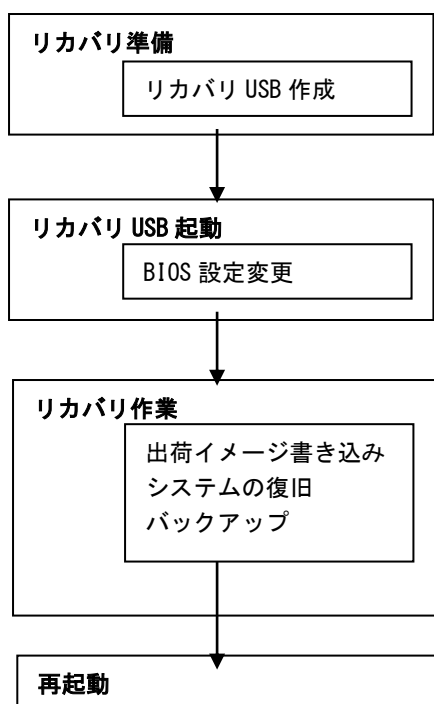


図 5-1-1. リカバリ DVD 使用の流れ

5-1-1 リカバリ準備

リカバリを実行する前に、準備する必要があるものを表 5-1-1-1 に記します。

表 5-1-1-1. リカバリ作業に必要な準備物

必要物	本章での名称	内容
USB メモリ (16GByte 以上)	出荷イメージリカバリ USB	出荷イメージリカバリ起動用の USB メモリです。 16GByte 以上のサイズのものを用意してください。 この USB メモリの中身は全て削除されます。 あらかじめバックアップを取るなどしておいてください。
USB メモリ (8GByte 以上)	リカバリ USB	リカバリ起動用の USB メモリです。 8GByte 以上のサイズのものを用意してください。 この USB メモリの中身は全て削除されます。 あらかじめバックアップを取るなどしておいてください。
USB 接続可能な ストレージメディア (8GByte 以上の 空き容量が必要)	バックアップ USB	リカバリ時の OS イメージを保管するメディアです。 8GByte 以上の空き容量がある必要があります。 また、 ファイルシステムは NTFS である必要があります。 メディアによってはバックアップ時に認識しない場合があります。 この場合は別のメディアでお試してください。(※1) 出荷イメージを書き込む際は必要ありません。
USB キーボード	キーボード	
USB マウス	マウス	
PC	WindowsPC	Windows OS 搭載の DVD ドライブが使用できる PC を用意してください。

※1 バックアップ USB が認識できない場合の主な例として、MBR がないことが挙げられます。

MBR がないメディアの場合は、メディアのディスクごとフォーマットすることで認識するようになる可能性があります。

用意したリカバリ USB は以下の手順でデータを作成してください。

● リカバリ USB 作成手順

- ① WindowsPC にリカバリ DVD を挿入します。
- ② 用意したリカバリ USB を、手順①の PC に接続します。
- ③ リカバリ DVD の以下のファイルを解凍します。

出荷イメージリカバリイメージ

[リカバリ DVD1]¥Recovery¥ReleaseRecoveryUSBImage.exe
 [リカバリ DVD1]¥Recovery¥ReleaseRecoveryUSBImage.7z.003
 [リカバリ DVD2]¥Recovery¥ReleaseRecoveryUSBImage.7z.001
 [リカバリ DVD3]¥Recovery¥ReleaseRecoveryUSBImage.7z.002
 4 つのファイルを同じフォルダに置いて、exe を実行してください。

バックアップ用リカバリイメージ

[リカバリ DVD]¥Recovery¥RecoveryUSBImage.zip

- ④ PC 上の任意の場所に解凍してください。(解凍先のストレージに 16GByte 程度の空き領域が必要です。)
解凍が完了するまでお待ちください。
解凍が完了すると「ReleaseRecoveryUSBImage.ddi」または「RecoveryUSBImage.ddi」というファイルが展開されます。
- ⑤ リカバリ DVD の以下のファイルを**管理者権限**で実行します。
[リカバリ DVD]¥Recovery¥DDwin¥DDwin.exe
※注: WindowsVista 以降の OS をご使用の場合は管理者権限で起動する必要があります。

- ⑥ DD for Windows というツールが起動します。
- ⑦ 「対象ディスク」の項目に手順②で接続した USB メモリが表示されていることを確認してください。
「ディスク選択」ボタンを押して接続した USB メモリを選択してください。

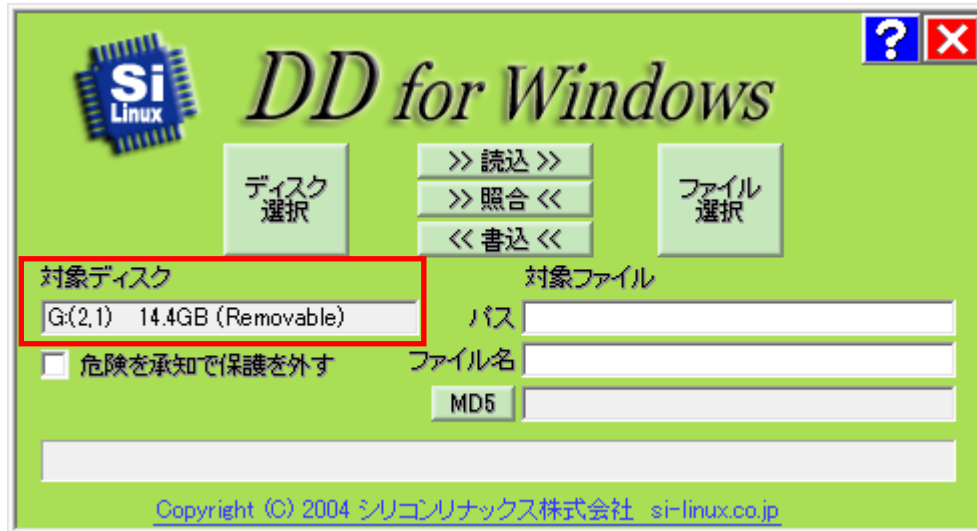


図 5-1-1-1. DD for Windows

- ⑧ 「ファイル選択」ボタンを押してください。
ファイル選択画面が開くので、手順④で解凍した出荷イメージリカバリ USB を作成する場合は「ReleaseRecoveryUSBImage.ddi」を、リカバリ USB を作成する場合は「RecoveryUSBImage.ddi」を選択してください。



図 5-1-1-2. ファイル選択

- ⑨ 「対象ファイル」の項目に RecoveryUSBImage.ddi が表示されたことを確認してください。

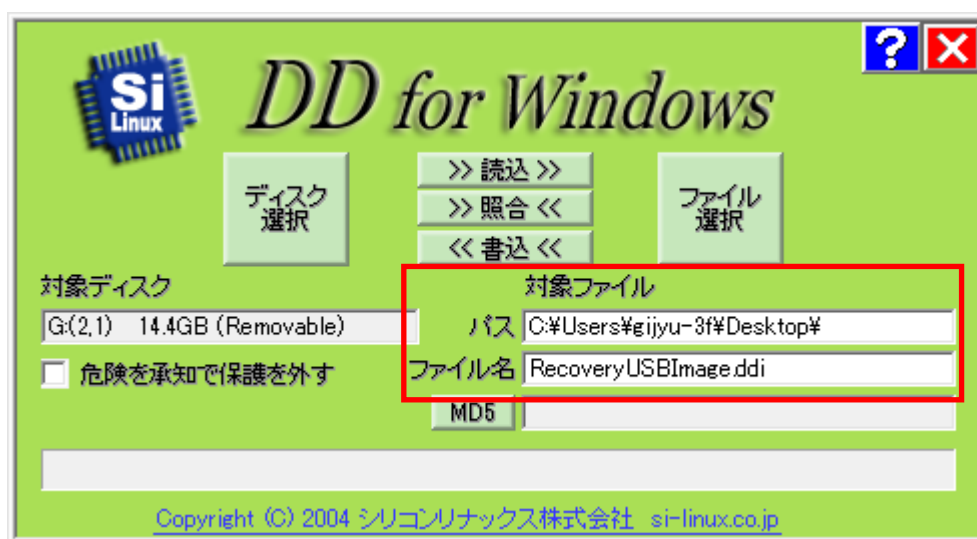


図 5-1-1-3. 対象ファイル

- ⑩ 「<<書込<<」ボタンを押してください。
USB メモリへ書き込みが始まります。
書き込みが完了するまでお待ちください。

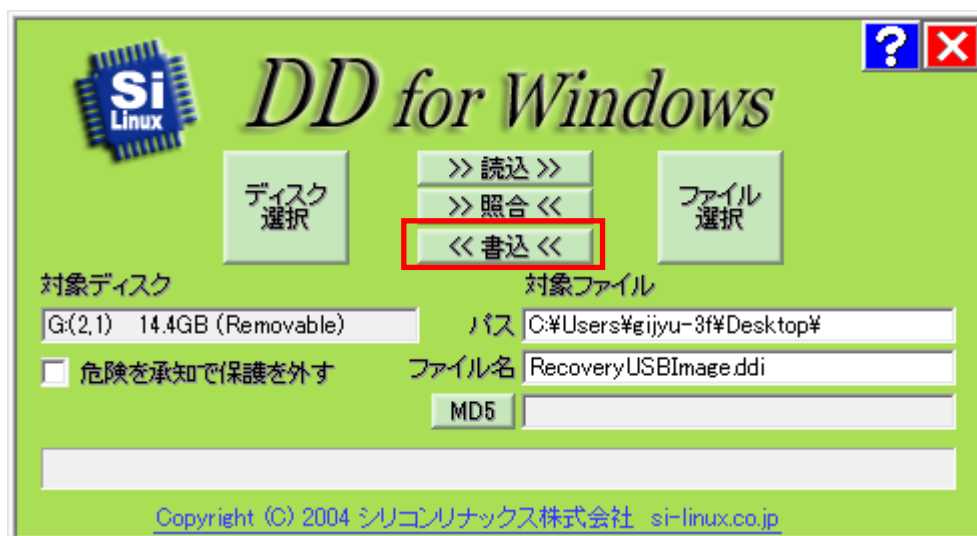


図 5-1-1-4. 書き込み開始

以上でリカバリ USB の作成は完了です。
リカバリ USB は一度作成すれば次回以降も使用することができます。

5-1-2 出荷イメージリカバリ USB 起動

出荷イメージリカバリ USB を起動させる前に、本体に接続されている LAN ケーブル、ストレージ（USB メモリ、SD カードなど）を取り外してください。サブストレージを接続している場合は、サブストレージを取り外してください。

● 出荷イメージリカバリ USB 起動手順

出荷イメージリカバリ USB から起動するために BIOS 設定が必要です。

以下の手順に従って BIOS 設定を変更してください。

- ① 出荷イメージリカバリ USB を産業用 PC 本体に接続します。
- ② キーボード、マウスを接続します。
- ③ 本体右側の DipSW の 3 を ON 側に倒します。
- ④ 電源を入れ、[F5] キーを連打してください。
しばらくすると起動ディスク選択ダイアログが開きます。（図 5-1-2-1）
- ⑤ 出荷イメージリカバリ USB を選択してエンターキーを押してください。



図 5-1-2-1. BIOS 設定画面 ASD WDT 設定

- ⑥ リカバリ USB から LinuxOK が起動され、自動的に出荷イメージ書き込みが始まります。
書き込み完了されるまでお待ちください。

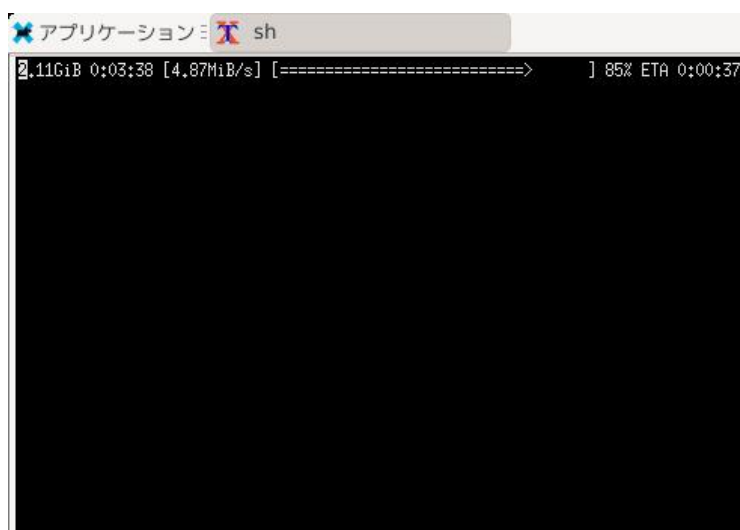


図 5-1-2-2. 書き込み中画面

- ⑦ 終了画面（図 5-1-2-3）が表示されるとバックアップファイルの書き込みは完了です。[ShutDown] ボタンを押して電源を落とし、リカバリ USB メモリ、バックアップ USB を外します。

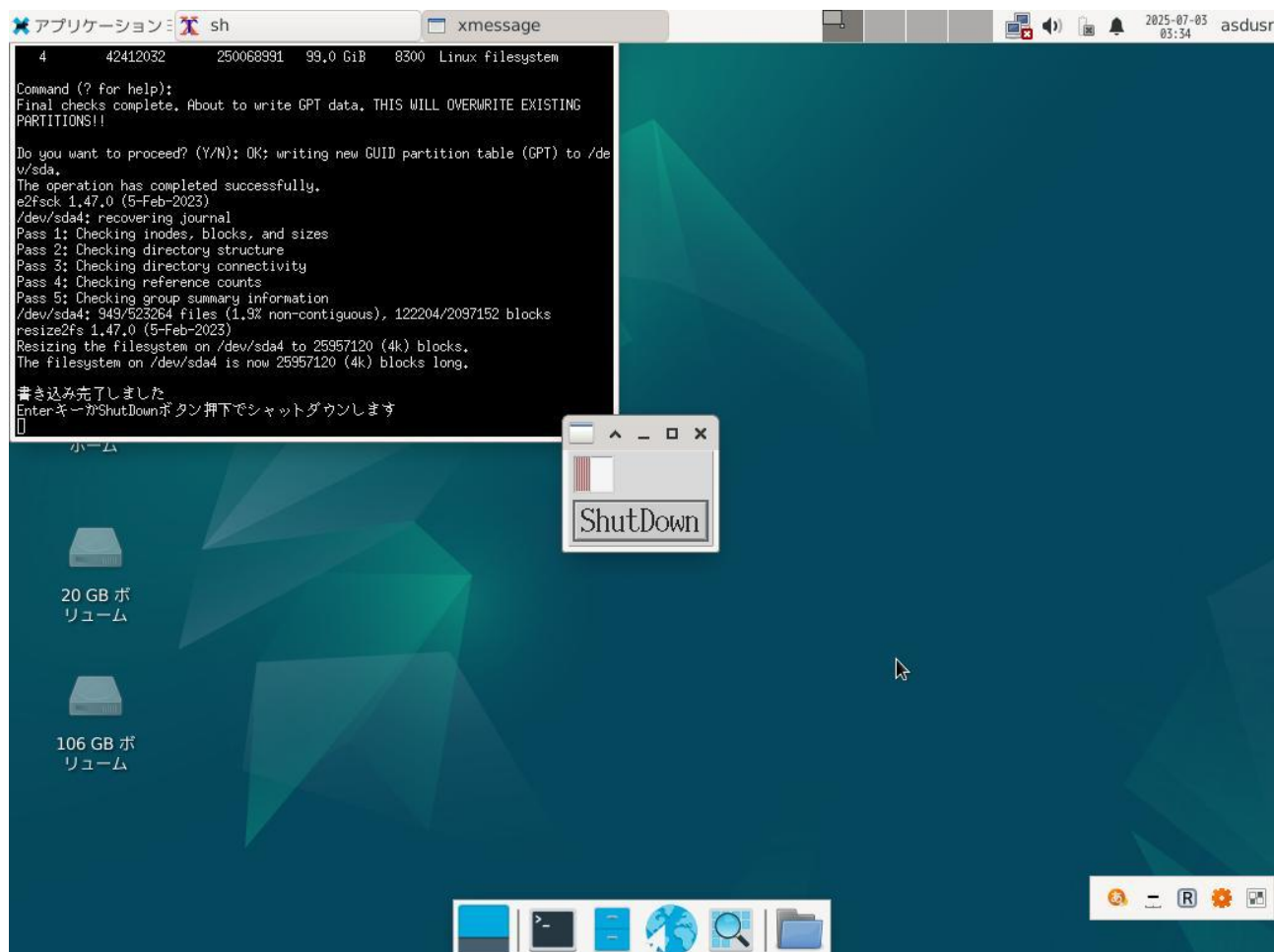


図 5-1-2-3. 終了画面

- ⑧ 電源を入れ、デスクトップが表示されて正常に起動すれば、システム復旧は完了です。
作業前に LAN、サブストレージ、USB メモリ、SD カードなどを取り外している場合は、
再度取り付けをしてください。

5-1-3 リカバリ USB 起動

リカバリ USB を起動させる前に、本体に接続されている LAN ケーブル、ストレージ（USB メモリ、SD カードなど）を取り外してください。サブストレージを接続している場合は、サブストレージを取り外してください。

● リカバリ USB 起動手順

リカバリ USB から起動するために BIOS 設定が必要です。

以下の手順に従って BIOS 設定を変更してください。

- ① リカバリ USB を産業用 PC 本体に接続します。
- ② キーボード、マウスを接続します。
- ③ 本体右側の DipSW の 3 を ON 側に倒します。
- ④ 電源を入れ、[F5] キーを連打してください。
しばらくすると起動ディスク選択ダイアログが開きます。（図 5-1-2-1）
- ⑤ リカバリ USB を選択してエンターキーを押してください。



図 5-1-3-1. BIOS 設定画面 ASD WDT 設定

- ⑥ リカバリ USB から起動が開始されます。しばらくお待ちください。
- ⑦ 正常にリカバリ USB から起動すると図 5-1-3-2 のリカバリメイン画面が表示されます。

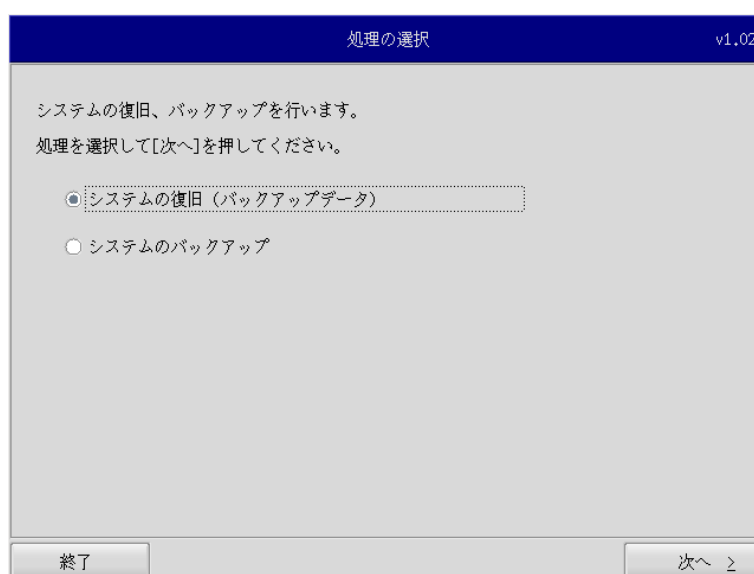


図 5-1-3-2. リカバリメイン画面

5-1-4 リカバリ作業

リカバリメイン画面から処理を選んでリカバリ作業を行います。

- システムの復旧（バックアップデータ）
- システムのバックアップ

リカバリ作業の詳細は、「5-2 システムの復旧（バックアップデータ）」、「5-3 システムのバックアップ」を参照してください。

5-2 システムの復旧（バックアップデータ）

「システムのバックアップ」で作成したバックアップファイルを使用して、メインストレージ（M.2）をバックアップファイルの状態に復旧させることができます。

- ※ バックアップファイルは、必ず対象となる本体で作成されたものを使用してください。他の機種のバックアップファイルでは動作しないので注意してください。
- ※ バックアップデータで復旧を行うとメインストレージのデータは、バックアップファイルの状態に戻ります。必要なデータがある場合は、復旧作業を行う前に保存してください。
- ※ 作業を始める前に、LAN ケーブルが接続されている場合は LAN ケーブルを取り外してください。サブストレージ、USB メモリ、SD カードなどのストレージメディアが接続されている場合は取り外してください。

●システムの復旧（バックアップデータ）の手順

- ① 「5-1-3 リカバリ USB 起動」を参考にリカバリ USB を起動させます。
- ② リカバリメイン画面（図 5-1-3-2）で[システムの復旧（バックアップデータ）]を選択し、[次へ]ボタンを押します。

- ③ メディアの選択画面(図 5-2-1)が表示されます。コピー先となるメディアを選択し、「次へ」ボタンを押します。



図 5-2-1. メディア選択画面

- ④ メディアとパーティション選択画面(図 5-2-2)が表示されます。あらかじめ用意しておいたバックアップ USB を本体に接続し、[メディア情報更新]ボタンを押してください。バックアップ USB のパーティションを選択し、[次へ]ボタンを押します。
- バックアップ USB の認識には少し時間がかかります。バックアップ USB を接続してすぐに[メディア情報更新]ボタンを押すと、バックアップ USB 情報が現れないことがあります。この場合は、1 分程度待って再度、[メディア情報更新]ボタンを押してみてください。

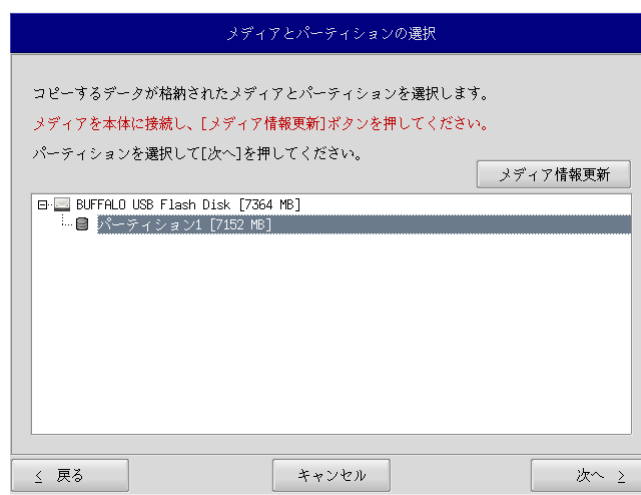


図 5-2-2. メディアとパーティション選択画面

- ⑤ フォルダ選択画面（図 5-2-3）が表示されます。[参照]ボタンを押します。

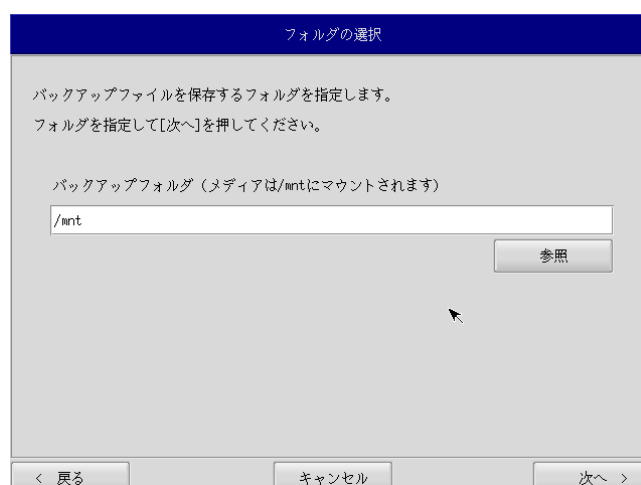


図 5-2-3. フォルダ選択画面

- ⑥ ファイル参照画面（図 5-2-4）が表示されます。接続したバックアップ USB のパーティションは、/mnt にマウントされているので、/mnt 以下から目的のファイルを探してください。[OK]を押すとファイル選択画面にもどります。

※ USB メモリの¥backup¥xxxxxx. img というバックアップファイルを指定する場合
/mnt/backup/xxxxxx. img
を指定します。

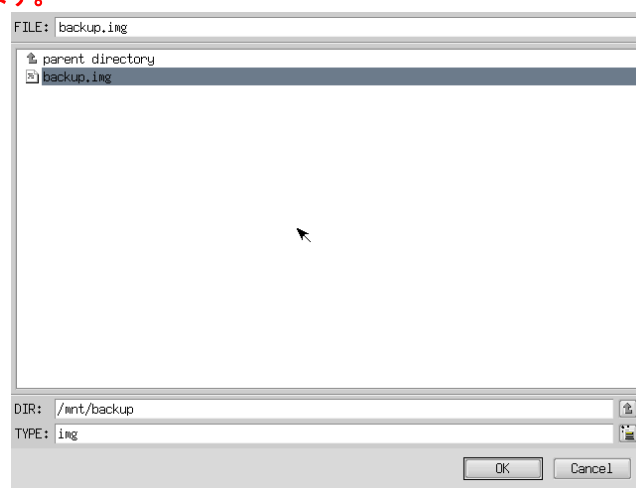


図 5-2-4. ファイル参照画面

- ⑦ ファイル参照画面（図 5-2-4）で指定したバックアップファイルが入力されていることを確認します。[次へ]ボタンを押します。

- ⑧ コンペア処理の選択画面(図 5-2-5)が表示されます。
データ書き込み時のコンペア処理の有無を選択します。

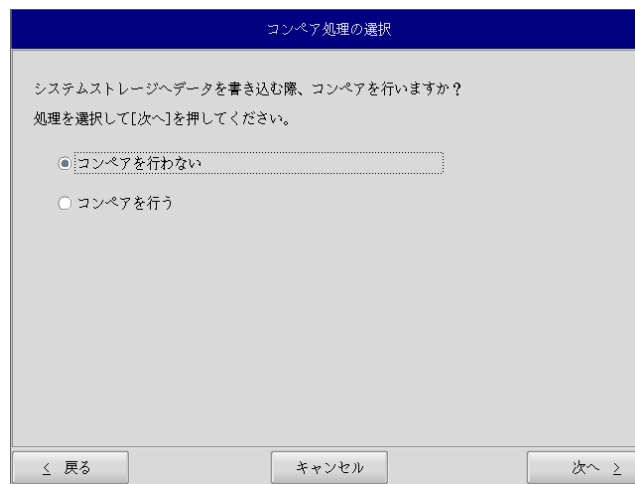


図 5-2-5. コンペア処理選択画面

- ⑨ 確認画面(図 5-2-6)が表示されます。メディア、パーティション、バックアップファイルを確認します。
[次へ] ボタンを押します。

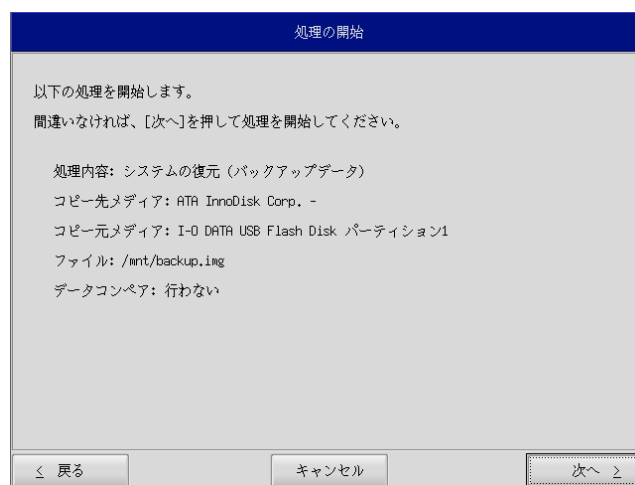


図 5-2-6. 確認画面

- ⑩ 実行中画面（図 5-2-7）が表示され、処理が開始されます。実行中はリカバリ USB メモリ、バックアップ USB を外さないでください。また、電源を落とさないようにしてください。

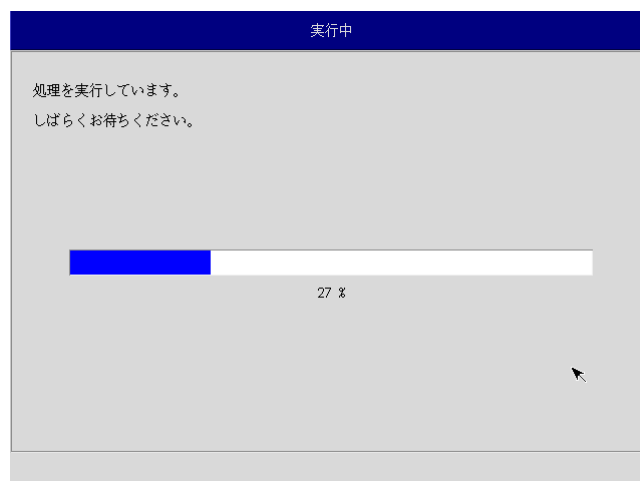


図 5-2-7. 実行中画面

- ⑪ 終了画面（図 5-2-8）が表示されるとバックアップファイルの書き込みは完了です。[終了]ボタンを押して電源を落とし、リカバリ USB メモリ、バックアップ USB を外します。

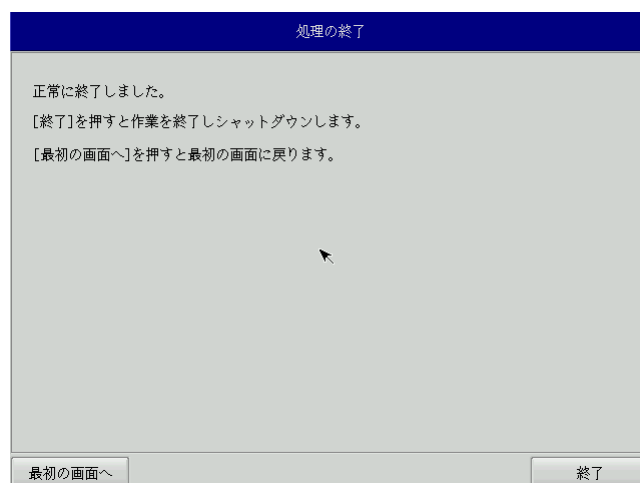


図 5-2-8. 終了画面

- ⑫ 電源を入れ、デスクトップが表示されて正常に起動すれば、システム復旧は完了です。作業前に LAN、サブストレージ、USB メモリ、SD カードなどを取り外している場合は、再度取り付けをしてください。

5-3 システムのバックアップ

メインストレージの状態をファイルに保存します。

- ※ 保存するバックアップファイルのサイズは、システムの状態によって変化しますので注意してください。
- ※ 作成されたバックアップファイルは、バックアップ作業を行った本体でのみ動作します。同じ型の本体であっても、他の機種では動作しませんので注意してください。
- ※ 作業を始める前に、LAN ケーブルが接続されている場合は LAN ケーブルを取り外してください。サブストレージ、USB メモリ、SD カードなどのストレージメディアが接続されている場合は取り外してください。

●システムのバックアップの手順

- ① 「5-1-3 リカバリ USB 起動」を参考にリカバリ USB を起動させます。
- ② リカバリメイン画面（図 5-1-3-2）で[システムのバックアップ]を選択し、[次へ]ボタンを押します。
- ③ メディアの選択画面（図 5-3-1）が表示されます。コピー元となるメディアを選択し、[次へ]ボタンを押します。

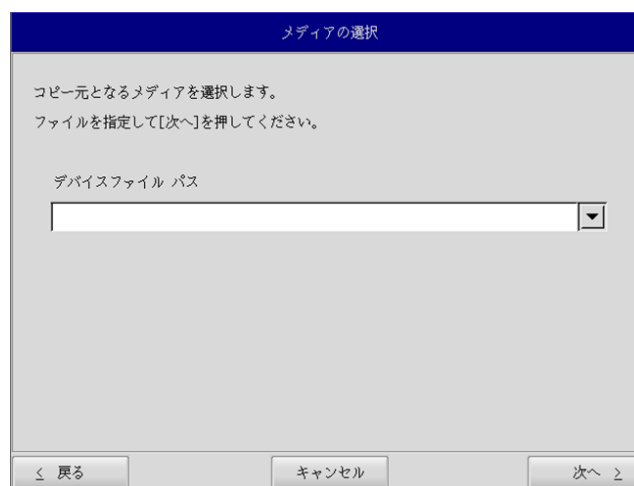


図 5-3-1. メディア選択画面

- ④ メディアとパーティション選択画面（図 5-3-2）が表示されます。本体にバックアップ USB を接続し、[メディア情報更新] ボタンを押してください。バックアップファイルを保存するバックアップ USB のパーティションを選択し、[次へ] ボタンを押します。
- バックアップ USB の認識には少し時間がかかります。バックアップ USB を接続してすぐに[メディア情報更新] ボタンを押すと、バックアップ USB 情報が現れないことがあります。この場合は、1 分程度待って再度、[メディア情報更新] ボタンを押してみてください。

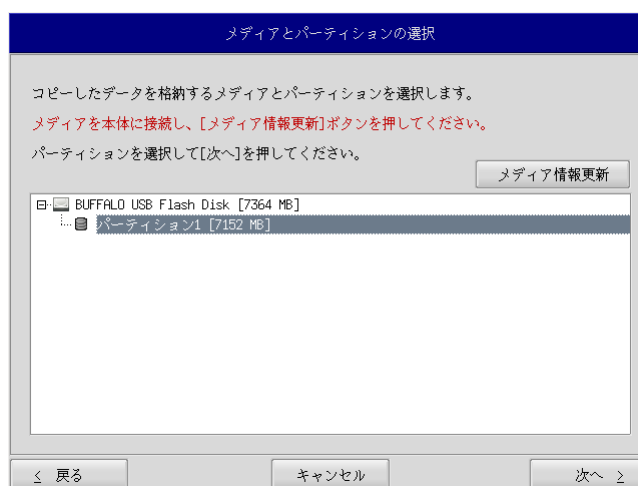


図 5-3-2. メディアとパーティション選択画面

- ⑤ フォルダ選択画面（図 5-3-3）が表示されます。[参照] ボタンを押します。

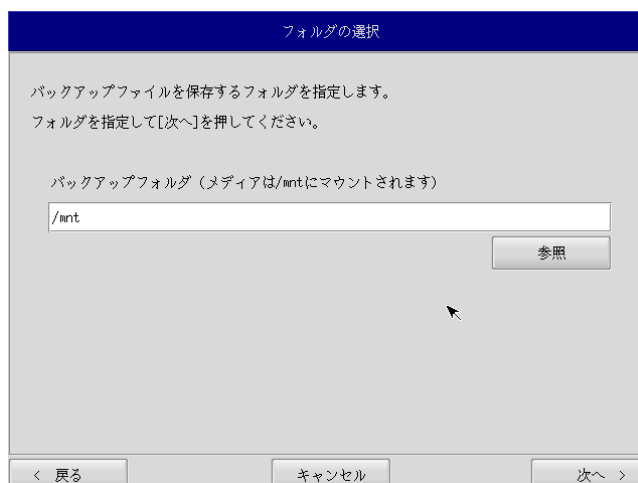


図 5-3-3. フォルダ選択画面

- ⑥ フォルダ参照画面（図 5-3-4）が表示されます。②で接続したパーティションは、/mnt にマウントされますので、/mnt 以下のフォルダを選択してください。[OK]を押すとフォルダ選択画面にもどります。

※ USB メモリに backup というフォルダがあり、このフォルダに保存する場合
/mnt/backup
を指定します。

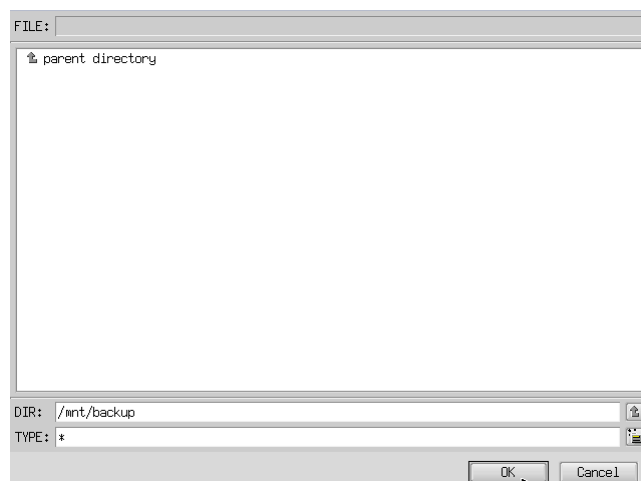


図 5-3-4. フォルダ参照画面

- ⑦ フォルダ選択画面（図 5-3-3）で指定したバックアップフォルダが入力されていることを確認します。[次へ]ボタンを押します。
- ⑧ 確認画面（図 5-3-5）が表示されます。メディア、パーティション、保存ファイルを確認します。[次へ]ボタンを押します。

※ 保存ファイル名は、現在時刻から自動生成されます。

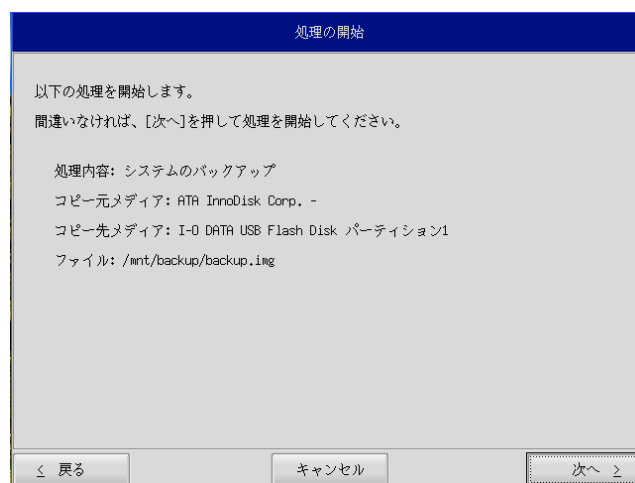


図 5-3-5. 確認画面

- ⑨ 実行中画面（図 5-3-6）が表示され、処理が開始されます。実行中はリカバリ USB、バックアップ USB を外さないでください。また、電源を落とさないようにしてください。

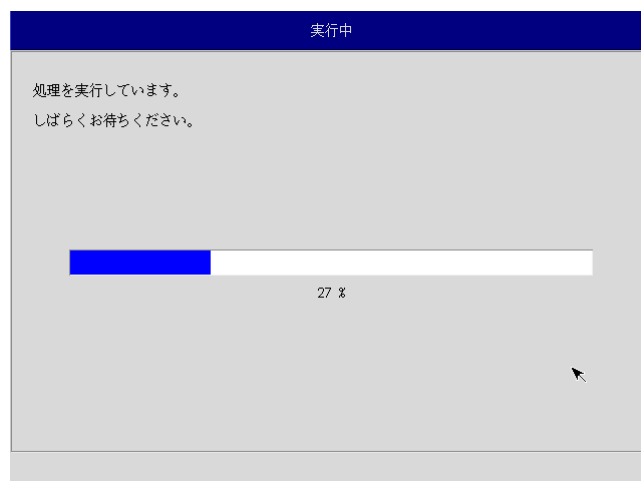


図 5-3-6. 実行中画面

- ⑩ 終了画面（図 5-3-7）が表示されるとバックアップ作業は完了です。[終了]ボタンを押して電源を落としてください。

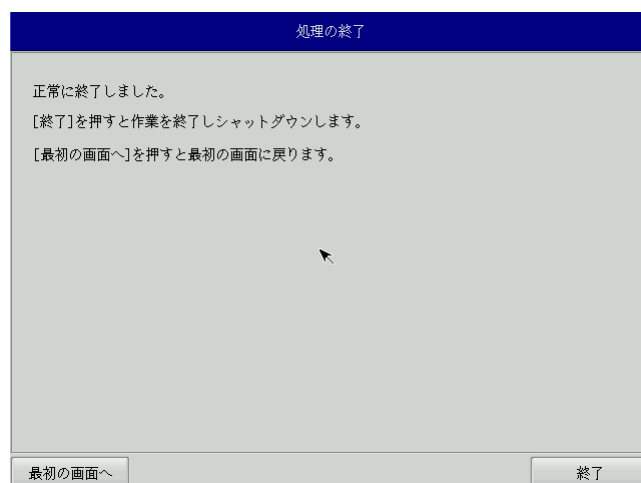


図 5-3-7. 終了画面

- ⑪ 電源を入れ、デスクトップが表示されて正常に起動すれば、システム復旧は完了です。作業前に LAN、サブストレージ、USB メモリ、SD カードなどを取り外している場合は、再度取り付けをしてください。

付録

A-1 参考文献

- 「ふつうのLinux プログラミング Linux の仕組みから学べる GCC プログラミングの王道」

著者 青木 峰郎
発行所 ソフトバンク パブリッシング
発行年 2005 年

- 「How Linux Works Linux の仕組み」

著者 Brian Ward
訳 吉川 典秀
発行所 毎日コミュニケーションズ
発行年 2006 年

- 「Linux デバイスドライバ 第3版」

著者 Jonathan Corbet
Alessandro Rubini
Greg Kroah-hartman
訳 山崎 康宏
山崎 邦子
長原 宏治
長原 陽子
発行所 オライリー・ジャパン
発行年 2005 年

このマニュアルについて

- (1) 本書の内容の一部又は全部を当社からの事前の承諾を得ることなく、無断で複写、複製、掲載することは固くお断りします。
- (2) 本書の内容に関しては、製品改良のためお断りなく、仕様などを変更することがありますのでご了承下さい。
- (3) 本書の内容に関しては万全を期しておりますが、万一ご不審な点や誤りなどお気づきのことがございましたらお手数ですが巻末記載の弊社までご連絡下さい。その際、巻末記載の書籍番号も併せてお知らせ下さい。